8-Queens Problem (HW1)

(Evolutionary Computation Course - Fall 2023)

Contributor

Matin Monshizadeh (9932122)

8-Queens problem:

Define the parameters:

```
population_size = 100
mutation_probability = 0.8
recombination_probability = 1.0
num_offspring = 2
max_fitness_evaluations = 10000
```

Initialize Population Function:

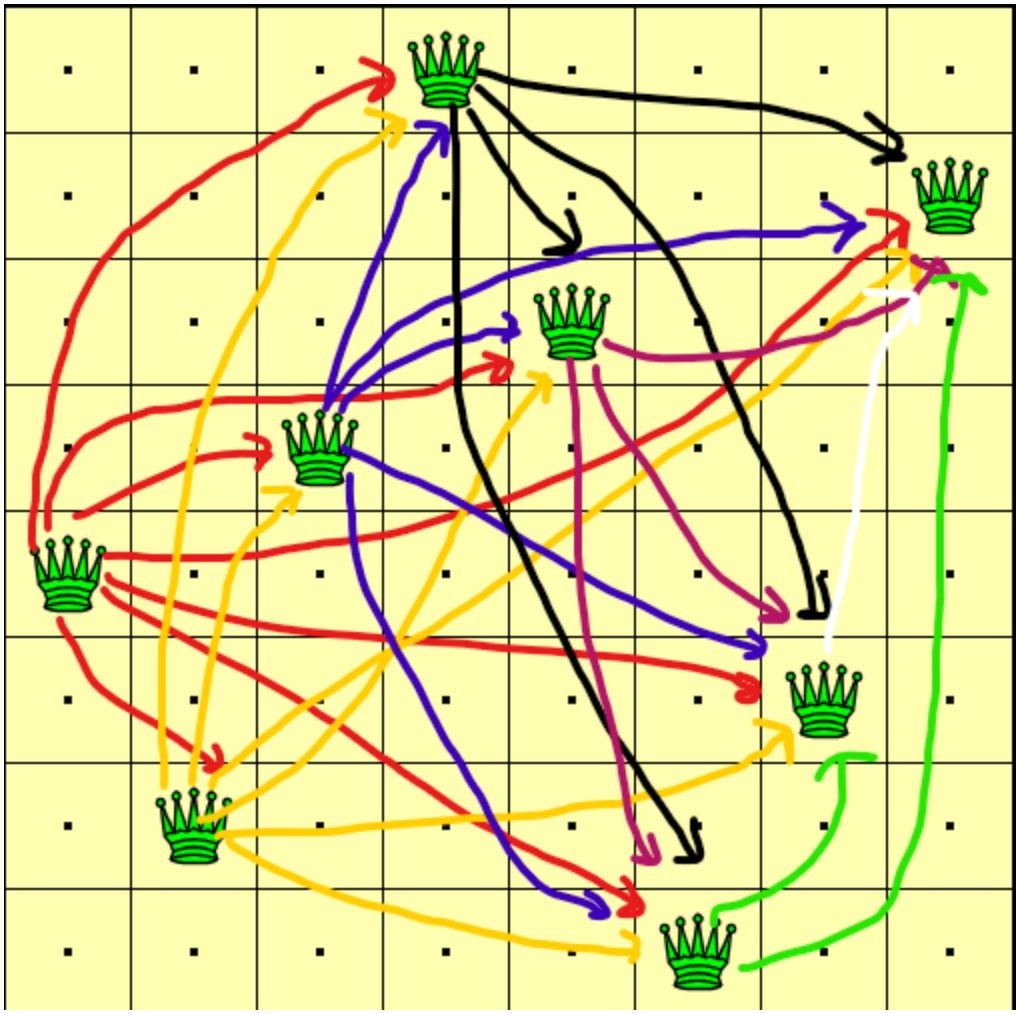
```
def initialize_population(board_size, population_size):
    population = [random.sample(range(board_size), board_size) for _ in range(population_size)]
    return population
```

This function initializes the population with random permutations of queens on the chessboard. **Each individual represents a unique arrangement**.

Fitness Function:

```
def fitness(individual):
    conflicts = 0
    board_size = len(individual)
    for i in range(board_size):
        for j in range(i + 1, board_size):
            if abs(i - j) == abs(individual[i] - individual[j]):
                conflicts += 1
    return board_size * (board_size - 1) // 2 - conflicts
```

fitness of an individual. In this case, it measures how many queen conflicts exist on the chessboard.



(Max fitness = 28)

Select Parents Function:

```
def select_parents(population, num_parents):
    parents = random.sample(population, num_parents)
    parents.sort(key=fitness, reverse=True)
    return parents[:2]
```

Selects the best 2 out of 5 individuals from the population to serve as parents for the crossover operation.

Crossover Function:

```
def crossover(parent1, parent2):
    cut_point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:cut_point] + [x for x in parent2 if x not in parent1[:cut_point]]
    child2 = parent2[:cut_point] + [x for x in parent1 if x not in parent2[:cut_point]]
    return child1, child2
```

Implements the 'Cut-and-Fill' crossover to create two child individuals from two parent individuals.

Mutate Function:

```
def mutate(individual):
    if random.random() < mutation_probability:
        i, j = random.sample(range(len(individual)), 2)
        individual[i], individual[j] = individual[j], individual[i]
```

Applies a mutation (swap) operation to an individual with a certain probability.

Plot Fitness Function:

```
def plot_fitness(iterations, max_fitness_values, avg_fitness_values):
    plt.figure()
    plt.plot(range(iterations), max_fitness_values, label='Max Fitness')
    plt.plot(range(iterations), avg_fitness_values, label='Avg Fitness')
    plt.xlabel('Iterations')
    plt.ylabel('Fitness')
    plt.title('Fitness per Iteration')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Defines a function to create a plot showing the maximum and average fitness values over iterations using matplotlib.

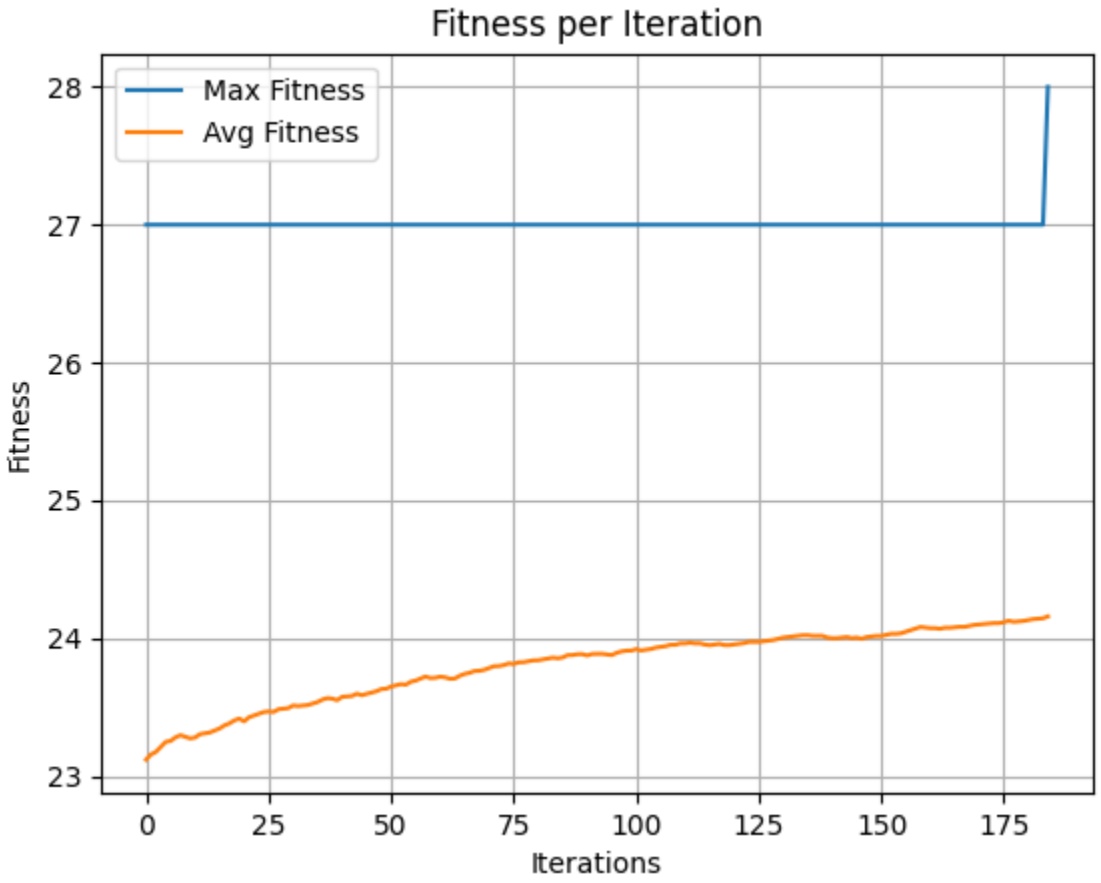
Genetic Algorithm Function:

```
def genetic_algorithm(board_size):
    population = initialize_population(board_size, population_size)
    fitness_values = [fitness(individual) for individual in population]

    max_fitness_values = []
    avg_fitness_values = []

    for _ in range(max_fitness_evaluations):
        parents = select_parents(population, 5)
        offspring = []

        for _ in range(num_offspring // 2):
            if random.random() < recombination_probability:
                child1, child2 = crossover(parents[0], parents[1])
                mutate(child1)
                mutate(child2)
                offspring.extend([child1, child2])

        offspring.sort(key=fitness, reverse=True)
        population[-2:] = offspring
        fitness_values.append(fitness(offspring[0]))

        max_fitness_values.append(max(fitness_values))
        avg_fitness_values.append(sum(fitness_values) / len(fitness_values))

        if fitness(offspring[0]) == board_size * (board_size - 1) // 2:
            print("Solution found:")
            print(offspring[0])
            break

    plot_fitness(len(max_fitness_values), max_fitness_values, avg_fitness_values)
    return max_fitness_values, avg_fitness_values
```

The main genetic algorithm function that coordinates the entire process:

- Initializes the population.
- Iterates over a specified number of fitness evaluations.
- Selects parents, performs crossover and mutation, and updates the population.
- Records the maximum and average fitness values over iterations.
- Stops when a solution is found or after reaching the maximum fitness evaluations.

Result:




```
Solution found:
[4, 7, 3, 0, 2, 5, 1, 6]
```