Week 5

Implementation of Simulated Annealing to Solve 8-Queens problem

Code:

```
import random
import math

def create_board(n):
  """Creates an initial board configuration."""
  return [random.randint(0, n - 1) for _ in range(n)]

def calculate_conflicts(board):
  """Calculates the number of conflicts (attacking pairs of queens)."""
  n = len(board)
  conflicts = 0
  for i in range(n):
    for j in range(i + 1, n):
      if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):
        conflicts += 1
  return conflicts

def generate_neighbor(board):
  """Generates a neighboring state by moving a single queen."""
  n = len(board)
  neighbor = board[:]  # Create a copy
  row_to_change = random.randint(0, n - 1)
  neighbor[row_to_change] = random.randint(0, n - 1)
  return neighbor

def simulated_annealing(n, initial_temperature, cooling_rate, iterations):
    """Solves the N-Queens problem using simulated annealing."""
    current_board = create_board(n)
    current_conflicts = calculate_conflicts(current_board)
    best_board = current_board[:]
    best_conflicts = current_conflicts

    temperature = initial_temperature
    for _ in range(iterations):
        neighbor_board = generate_neighbor(current_board)
        neighbor_conflicts = calculate_conflicts(neighbor_board)
```

```
        delta_e = neighbor_conflicts - current_conflicts

        if delta_e < 0 or random.uniform(0, 1) < math.exp(-delta_e / temperature):
            current_board = neighbor_board
            current_conflicts = neighbor_conflicts

        if current_conflicts < best_conflicts:
            best_board = current_board[:]
            best_conflicts = current_conflicts

        temperature *= cooling_rate

    return best_board, best_conflicts


# Example usage for 8 Queens
n = 8
initial_temperature = 1000
cooling_rate = 0.99
iterations = 10000

best_solution, min_conflicts = simulated_annealing(n, initial_temperature, cooling_rate,
iterations)

print("Best Solution:", best_solution)
print("Conflicts:", min_conflicts)


# Visualization (Optional - requires matplotlib)
import matplotlib.pyplot as plt

def visualize_board(board):
    n = len(board)
    board_visual = [['.' for _ in range(n)] for _ in range(n)]
    for i, col in enumerate(board):
        board_visual[col][i] = 'Q'

    for row in board_visual:
        print(''.join(row))


if min_conflicts == 0:
```

```
    print("\nSolution Visualization:")
    visualize_board(best_solution)
else:
    print("\nNo perfect solution found within the given iterations.")
```

Output:

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

Solution Visualization:
.....Q..
..Q.....
......Q.
.Q......
...Q....
.......Q
Q.......
....Q...
```

29/10/24 week-5:

Implementation of simulated Annealing to solve 8-Queens Problem

Algorithm:

de

1. Initialize:
   - Randomly place n queens on a nxn board, one per row.

2. Set temperature:
   - set an initial high temperature that decreases gradually overtime.

3. Iterate (for a set of numbers of steps or until a solution is found)
   - calculate the current conflicts on the board, representing the number of queen pairs attacking each other
   - Generate neighbour
   - move a single queen to a new column in its row to create a neighbouring configuration
   - Evaluate the neighbour
   - calculate the conflict difference between the current and neighbouring configurations.
   - If the neighbour has fewer conflicts accept it as new current state.
   - If the neighbour has more conflicts, accept it with a probability based on the tempera-ture
   - update the best solution
   - Track the best configuration (with the fewest conflicts) found during the process.

classmate

Date

Page

- cool down
  - Reduce the temperature according to the cooling rate.

4. Return:
   - when the temperature is very low or a solution is found (zero conflicts), return the best configuration.

```
current ← initial state
T ← a large positive value
while T > 0 do
    next ← a random neighbour of current
    ΔE ← current-cost − next.cost
    if ΔE > 0 then
        current ← next
    else
        current ← next with probability p = e^(ΔE/T)
    end if
    decrease T
end while
return current.
```