Implementation of Hill Climbing Algorithm to solve N-Queens Problem

```
import random


class NQueens:
    def __init__(self, n):
        self.n = n
        self.solutions = set()  # To store unique solutions

    def random_state(self):
        """Generate a random state (initial placement of queens)."""
        return [random.randint(0, self.n - 1) for _ in range(self.n)]

    def fitness(self, state):
        """Calculate the number of pairs of queens that are attacking each other."""
        attacks = 0
        for i in range(self.n):
            for j in range(i + 1, self.n):
                if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
                    attacks += 1
        return attacks

    def get_neighbors(self, state):
        """Generate all neighboring states by moving one queen to a different row in its column."""
```

```
        neighbors = []
        for col in range(self.n):
            for row in range(self.n):
                if row != state[col]:
                    new_state = state[:]
                    new_state[col] = row
                    neighbors.append(new_state)
        return neighbors


    def hill_climbing(self):
        """Perform the hill climbing algorithm to solve the N-Queens problem."""
        attempts = 0
        while attempts < 1000:  # Limit the number of attempts to avoid infinite loops
            current = self.random_state()  # Start with a random state
            current_fitness = self.fitness(current)

            while True:
                # If the current state is a solution
                if current_fitness == 0:
                    self.solutions.add(tuple(current))  # Store unique solution
                    break

                neighbors = self.get_neighbors(current)
                next_state = None
```

```
            next_fitness = float('inf')

            # Evaluate neighbors
            for neighbor in neighbors:
                neighbor_fitness = self.fitness(neighbor)
                if neighbor_fitness < next_fitness:
                    next_fitness = neighbor_fitness
                    next_state = neighbor

            # If no better neighbor found, break to start over
            if next_fitness >= current_fitness:
                break

            current = next_state
            current_fitness = next_fitness

        attempts += 1  # Increment the number of attempts

def print_board(state):
    """Print the board state in a readable format."""
    for row in range(len(state)):
        line = ['Q' if col == state[row] else '.' for col in range(len(state))]
        print(' '.join(line))
    print("\n")
```

```
def main():
    n = int(input("Enter the number of queens (N): "))
    nqueens = NQueens(n)
    nqueens.hill_climbing()

    if nqueens.solutions:
        print(f"Found {len(nqueens.solutions)} unique solutions:")
        for idx, solution in enumerate(nqueens.solutions):
            print(f"Solution {idx + 1}:")
            print_board(solution)
    else:
        print("No solution found.")

if __name__ == "__main__":
    main()
```

Output:

Enter the number of queens (N): 4

Found 2 unique solutions:

Solution 1:

. . Q .

Q . . .

. . . Q

. Q . .

Solution 2:

. Q . .

. . . Q

Q . . .

. . Q .