```python
def hill_climbing(n, max_restarts=100):
    board = [random.randint(0, n-1) for _ in range(n)]
    restarts = 0

    while restarts < max_restarts:
        current_attacks = calculate_attacks(board)
        if current_attacks == 0:
            return board

        next_board, next_attacks = get_best_neighbor(board)

        if next_attacks == current_attacks:
            board = [random.randint(0, n-1) for _ in range(n)]
            restarts += 1
        else:
            board = next_board

    return None

def print_board(board):
    if board is None:
        print("No solution found.")
        return
    n = len(board)
    for row in range(n):
        line = ""
        for col in range(n):
            if board[col] == row:
                line += "Q "
            else:
                line += ". "
        print(line)

n = 4
solution = hill_climbing(n)
print_board(solution)
```

```
. Q . .
. . . Q
Q . . .
. . Q .
```

```
                best_energy = current_energy

        temperature *= cooling_rate
        iteration += 1

        if best_energy == 0:
            break

    return best_state, best_energy

def print_solution(state):
    N = len(state)
    board = [["." for _ in range(N)] for _ in range(N)]
    for i in range(N):
        board[state[i]][i] = "Q"

    for row in board:
        print(" ".join(row))

N = 6
initial_temperature = 1000
cooling_rate = 0.99
max_iterations = 10000

solution, conflicts = simulated_annealing(N, initial_temperature, cooling_rate, max_iterations)

if conflicts == 0:
    print("Solution found:")
    print_solution(solution)
else:
    print(f"Solution not found, conflicts: {conflicts}")
```

```
Solution found:
. . Q . . .
. . . . . Q
. Q . . . .
. . . . Q .
Q . . . . .
. . . Q . .
```

[ ]: