```python
import mlrose_hiive as mlrose
import numpy as np

def queens_max(position):
    no_attack_on_j = 0
    queen_not_attacking = 0
    for i in range(len(position) - 1):
        no_attack_on_j = 0
        for j in range(i + 1, len(position)):
            if (position[j] != position[i]) and (position[j] != position[i] + (j - i)) and (position[j] != position[i] - (j - i)):
                no_attack_on_j += 1
        if (no_attack_on_j == len(position) - 1 - i):
            queen_not_attacking += 1
    if (queen_not_attacking == 7):
        queen_not_attacking += 1
    return queen_not_attacking

objective = mlrose.CustomFitness(queens_max)

problem = mlrose.DiscreteOpt(length=8, fitness_fn=objective, maximize=True, max_val=8)
T = mlrose.ExpDecay()

initial_position = np.array([4, 6, 1, 5, 2, 0, 3, 7])

best_position, best_objective,fitness_curve= mlrose.simulated_annealing(problem=problem, schedule=T, max_attempts=500,init_state=initial_position)

print('The best position found is:', best_position)
print('The number of queens that are not attacking each other is:', best_objective)
def print_chessboard(solution):
    print("\nChessboard Configuration:")
    for row in range(8):
        line = ""
        for col in range(8):
            if solution[col] == row:
                line += " Q "
            else:
                line += " . "
        print(line)

print_chessboard(best_position)
```

```
The best position found is: [5 3 0 4 7 1 6 2]
The number of queens that are not attacking each other is: 8.0

Chessboard Configuration:
 .  .  Q  .  .  .  .  .
 .  .  .  .  .  Q  .  .
 .  .  .  .  .  .  .  Q
 .  Q  .  .  .  .  .  .
 .  .  .  Q  .  .  .  .
 Q  .  .  .  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  .  Q  .  .  .
```

```
import mlrose_hiive as mlrose
import numpy as np

def queens_max(position):
    no_attack_on_j = 0
    queen_not_attacking = 0
    for i in range(len(position) - 1):
        no_attack_on_j = 0
        for j in range(i + 1, len(position)):
            if (position[j] != position[i]) and (position[j] != position[i] + (j - i)) and (position[j] != position[i] - (j - i)):
                no_attack_on_j += 1
        if (no_attack_on_j == len(position) - 1 - i):
            queen_not_attacking += 1
    if (queen_not_attacking == 7):
        queen_not_attacking += 1
    return queen_not_attacking

objective = mlrose.CustomFitness(queens_max)

problem = mlrose.DiscreteOpt(length=8, fitness_fn=objective, maximize=True, max_val=8)
T = mlrose.ExpDecay()

initial_position = np.array([4, 6, 1, 5, 2, 0, 3, 7])

best_position, best_objective,fitness_curve= mlrose.simulated_annealing(problem=problem, schedule=T, max_attempts=500,init_state=initial_

print('The best position found is:', best_position)
print('The number of queens that are not attacking each other is:', best_objective)
def print_chessboard(solution):
    print("\nChessboard Configuration:")
    for row in range(8):
        line = ""
        for col in range(8):
            if solution[col] == row:
                line += " Q "
            else:
                line += " . "
        print(line)

print_chessboard(best_position)
```

```
The best position found is: [5 3 0 4 7 1 6 2]
The number of queens that are not attacking each other is: 8.0

Chessboard Configuration:
 .  .  Q  .  .  .  .  .
 .  .  .  .  .  Q  .  .
 .  .  .  .  .  .  .  Q
 .  Q  .  .  .  .  .  .
 .  .  .  Q  .  .  .  .
 Q  .  .  .  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  .  Q  .  .  .
```