import random

import math

import matplotlib.pyplot as plt

def create\_initial\_solution(n):

return random.sample(range(n), n)

def calculate\_fitness(state):

n = len(state)

row\_conflicts = sum([state.count(i) - 1 for i in state])

diagonal\_conflicts = 0

for i in range(n):

for j in range(i + 1, n):

if abs(state[i] - state[j]) == abs(i - j):

diagonal\_conflicts += 1

return row\_conflicts + diagonal\_conflicts

def random\_neighbor(state):

neighbor = state[:]

i = random.randint(0, len(state) - 1)

neighbor[i] = random.randint(0, len(state) - 1)

return neighbor

def simulated\_annealing(n, initial\_temp=1000, cooling\_rate=0.95, max\_iterations=1000):

current\_solution = create\_initial\_solution(n)

current\_fitness = calculate\_fitness(current\_solution)

best\_solution = current\_solution

best\_fitness = current\_fitness

temperature = initial\_temp

for iteration in range(max\_iterations):

neighbor = random\_neighbor(current\_solution)

neighbor\_fitness = calculate\_fitness(neighbor)

fitness\_diff = neighbor\_fitness - current\_fitness

if fitness\_diff < 0 or random.uniform(0, 1) < math.exp(-fitness\_diff / temperature):

current\_solution = neighbor

current\_fitness = neighbor\_fitness

if current\_fitness < best\_fitness:

best\_solution = current\_solution

best\_fitness = current\_fitness

temperature \*= cooling\_rate

return best\_solution, best\_fitness

def plot\_solution(solution):

n = len(solution)

plt.figure(figsize=(n, n))

plt.xlim(-1, n)

plt.ylim(-1, n)

# Draw the chessboard

for i in range(n):

for j in range(n):

if (i + j) % 2 == 0:

plt.gca().add\_patch(plt.Rectangle((j, i), 1, 1, color='lightgrey'))

# Place the queens

for col, row in enumerate(solution):

plt.gca().add\_patch(plt.Circle((col + 0.5, row + 0.5), 0.4, color='purple'))

plt.xticks(range(n))

plt.yticks(range(n))

plt.gca().invert\_yaxis()

plt.grid(False)

plt.show()

# Parameters

n = 8 # Number of queens

best\_solution, best\_fitness = simulated\_annealing(n)

# Output results

print(f"Best state (Queen positions): {best\_solution}, Number of conflicts: {best\_fitness}")

# Plot the solution

plot\_solution(best\_solution)

output:

