import numpy as np

import random

def initialize\_random\_solution(grid):

"""Create a random solution based on the initial Sudoku grid."""

filled\_positions = [(i, j) for i in range(9) for j in range(9) if grid[i][j] != 0]

empty\_positions = [(i, j) for i in range(9) for j in range(9) if grid[i][j] == 0]

solution = np.copy(grid)

for (i, j) in empty\_positions:

solution[i][j] = random.randint(1, 9)

return solution

def evaluate\_energy(grid):

"""Calculate the number of conflicts in the grid."""

conflicts = 0

# Count conflicts in rows and columns

for i in range(9):

row = [num for num in grid[i] if num != 0]

conflicts += len(row) - len(set(row)) # Count duplicates

col = [grid[j][i] for j in range(9) if grid[j][i] != 0]

conflicts += len(col) - len(set(col)) # Count duplicates

# Count conflicts in 3x3 boxes

for box\_row in range(3):

for box\_col in range(3):

box = [grid[i][j] for i in range(box\_row \* 3, box\_row \* 3 + 3)

for j in range(box\_col \* 3, box\_col \* 3 + 3)

if grid[i][j] != 0]

conflicts += len(box) - len(set(box)) # Count duplicates

return conflicts

def generate\_neighbor(grid):

"""Generate a neighbor solution by swapping two random numbers."""

neighbor = np.copy(grid)

empty\_positions = [(i, j) for i in range(9) for j in range(9) if grid[i][j] == 0]

if len(empty\_positions) < 2:

return neighbor # Not enough empty spots to swap

pos1, pos2 = random.sample(empty\_positions, 2)

neighbor[pos1], neighbor[pos2] = neighbor[pos2], neighbor[pos1]

return neighbor

def calculate\_acceptance\_probability(current\_energy, neighbor\_energy, temperature):

"""Calculate the acceptance probability."""

if neighbor\_energy < current\_energy:

return 1.0

return np.exp((current\_energy - neighbor\_energy) / temperature)

def simulated\_annealing\_sudoku(initial\_grid):

"""Solve the Sudoku puzzle using simulated annealing."""

current\_solution = initialize\_random\_solution(initial\_grid)

current\_energy = evaluate\_energy(current\_solution)

temperature = 1.0

final\_temperature = 0.001

cooling\_rate = 0.99

while temperature > final\_temperature:

neighbor = generate\_neighbor(current\_solution)

neighbor\_energy = evaluate\_energy(neighbor)

if neighbor\_energy < current\_energy:

current\_solution = neighbor

current\_energy = neighbor\_energy

else:

acceptance\_probability = calculate\_acceptance\_probability(current\_energy, neighbor\_energy, temperature)

if random.random() < acceptance\_probability:

current\_solution = neighbor

current\_energy = neighbor\_energy

temperature \*= cooling\_rate # Decrease the temperature

return current\_solution if current\_energy == 0 else None # Return solution or None if unsolvable

# Example Sudoku puzzle (0 represents empty cells)

initial\_sudoku = [

[5, 3, 0, 0, 7, 0, 0, 0, 0],

[6, 0, 0, 1, 9, 5, 0, 0, 0],

[0, 9, 8, 0, 0, 0, 0, 6, 0],

[8, 0, 0, 0, 6, 0, 0, 0, 3],

[4, 0, 0, 8, 0, 3, 0, 0, 1],

[7, 0, 0, 0, 2, 0, 0, 0, 6],

[0, 6, 0, 0, 0, 0, 2, 8, 0],

[0, 0, 0, 4, 1, 9, 0, 0, 5],

[0, 0, 0, 0, 8, 0, 0, 7, 9]

]

solution = simulated\_annealing\_sudoku(initial\_sudoku)

if solution is not None:

print("Solved Sudoku:")

print(solution)

else:

print("No solution found.")

Output:

