import random

import numpy as np

import math

# Define the Problem: Cities with coordinates (can be modified with real data)

class City:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def distance(self, city):

return math.sqrt((self.x - city.x)\*\*2 + (self.y - city.y)\*\*2)

# Ant Colony Optimization for TSP

class ACO\_TSP:

def \_\_init\_\_(self, cities, num\_ants, num\_iterations, alpha=1.0, beta=2.0, rho=0.5, q0=0.9):

self.cities = cities

self.num\_ants = num\_ants

self.num\_iterations = num\_iterations

self.alpha = alpha # Importance of pheromone

self.beta = beta # Importance of heuristic (distance)

self.rho = rho # Pheromone evaporation rate

self.q0 = q0 # Probability of choosing the best path

self.num\_cities = len(cities)

self.pheromone = np.ones((self.num\_cities, self.num\_cities)) # Initial pheromone values

self.heuristic = np.zeros((self.num\_cities, self.num\_cities)) # Heuristic info (inverse of distance)

self.best\_tour = None

self.best\_tour\_length = float('inf')

# Compute heuristic (inverse of distance)

for i in range(self.num\_cities):

for j in range(i + 1, self.num\_cities):

dist = cities[i].distance(cities[j])

self.heuristic[i][j] = 1.0 / dist if dist != 0 else 0

self.heuristic[j][i] = self.heuristic[i][j]

def select\_next\_city(self, current\_city, visited\_cities):

probabilities = np.zeros(self.num\_cities)

total\_pheromone = 0.0

# Calculate the transition probabilities

for city in range(self.num\_cities):

if city not in visited\_cities:

pheromone = self.pheromone[current\_city][city] \*\* self.alpha

heuristic = self.heuristic[current\_city][city] \*\* self.beta

probabilities[city] = pheromone \* heuristic

total\_pheromone += probabilities[city]

# Normalize probabilities

if total\_pheromone == 0:

return random.choice([city for city in range(self.num\_cities) if city not in visited\_cities])

probabilities /= total\_pheromone

# Exploration vs Exploitation

if random.random() < self.q0:

# Exploitation: choose the city with the highest probability

next\_city = np.argmax(probabilities)

else:

# Exploration: choose based on probabilities

next\_city = np.random.choice(self.num\_cities, p=probabilities)

return next\_city

def update\_pheromone(self, ants):

# Evaporate pheromone

self.pheromone \*= (1 - self.rho)

# Deposit pheromone based on the ants' solutions

for ant in ants:

pheromone\_deposit = 1.0 / ant.tour\_length

for i in range(self.num\_cities):

current\_city = ant.tour[i]

next\_city = ant.tour[(i + 1) % self.num\_cities]

self.pheromone[current\_city][next\_city] += pheromone\_deposit

self.pheromone[next\_city][current\_city] += pheromone\_deposit

def run(self):

for iteration in range(self.num\_iterations):

ants = [Ant(self.num\_cities, self) for \_ in range(self.num\_ants)]

for ant in ants:

ant.construct\_solution()

# Update pheromones

self.update\_pheromone(ants)

# Update the best solution found so far

for ant in ants:

if ant.tour\_length < self.best\_tour\_length:

self.best\_tour\_length = ant.tour\_length

self.best\_tour = ant.tour

print(f"Iteration {iteration + 1}/{self.num\_iterations}: Best Tour Length = {self.best\_tour\_length}")

return self.best\_tour, self.best\_tour\_length

# Ant class to simulate each ant's behavior

class Ant:

def \_\_init\_\_(self, num\_cities, aco\_tsp):

self.num\_cities = num\_cities

self.aco\_tsp = aco\_tsp

self.tour = []

self.tour\_length = 0.0

def construct\_solution(self):

start\_city = random.randint(0, self.num\_cities - 1)

self.tour = [start\_city]

self.tour\_length = 0.0

visited\_cities = set(self.tour)

current\_city = start\_city

while len(self.tour) < self.num\_cities:

next\_city = self.aco\_tsp.select\_next\_city(current\_city, visited\_cities)

self.tour.append(next\_city)

visited\_cities.add(next\_city)

self.tour\_length += self.aco\_tsp.cities[current\_city].distance(self.aco\_tsp.cities[next\_city])

current\_city = next\_city

# Add the return to the starting city

self.tour\_length += self.aco\_tsp.cities[self.tour[-1]].distance(self.aco\_tsp.cities[self.tour[0]])

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Define cities (x, y coordinates)

cities = [City(0, 0), City(1, 3), City(4, 3), City(6, 1), City(3, 0)]

# Initialize and run ACO

aco = ACO\_TSP(cities=cities, num\_ants=10, num\_iterations=100, alpha=1.0, beta=2.0, rho=0.5, q0=0.9)

best\_tour, best\_tour\_length = aco.run()

# Output the best tour and its length

print("\nBest tour found:", best\_tour)

print("Best tour length:", best\_tour\_length)