import math

import random

import operator

import matplotlib.pyplot as plt

import time

class Graph(object):

def \_\_init\_\_(self, cost\_matrix: list, rank: int):

self.matrix = cost\_matrix

self.rank = rank

self.pheromone = [[1 / (rank \* rank) for j in range(rank)] for i in range(rank)]

class ACO(object):

def \_\_init\_\_(self, ant\_count: int, generations: int, alpha: float, beta: float, rho: float, q: int,

strategy: int):

self.Q = q

self.rho = rho

self.beta = beta

self.alpha = alpha

self.ant\_count = ant\_count

self.generations = generations

self.update\_strategy = strategy

def \_update\_pheromone(self, graph: Graph, ants: list):

for i, row in enumerate(graph.pheromone):

for j, col in enumerate(row):

graph.pheromone[i][j] \*= self.rho

for ant in ants:

graph.pheromone[i][j] += ant.pheromone\_delta[i][j]

def solve(self, graph: Graph):

best\_cost = float('inf')

best\_solution = []

for gen in range(self.generations):

ants = [\_Ant(self, graph) for i in range(self.ant\_count)]

for ant in ants:

for i in range(graph.rank - 1):

ant.\_select\_next()

ant.total\_cost += graph.matrix[ant.tabu[-1]][ant.tabu[0]]

if ant.total\_cost < best\_cost:

best\_cost = ant.total\_cost

best\_solution = [] + ant.tabu

ant.\_update\_pheromone\_delta()

self.\_update\_pheromone(graph, ants)

return best\_solution, best\_cost

class \_Ant(object):

def \_\_init\_\_(self, aco: ACO, graph: Graph):

self.colony = aco

self.graph = graph

self.total\_cost = 0.0

self.tabu = []

self.pheromone\_delta = []

self.allowed = [i for i in range(graph.rank)]

self.eta = [[0 if i == j else 1 / graph.matrix[i][j] for j in range(graph.rank)] for i in

range(graph.rank)]

start = random.randint(0, graph.rank - 1)

self.tabu.append(start)

self.current = start

self.allowed.remove(start)

def \_select\_next(self):

denominator = 0

for i in self.allowed:

denominator += self.graph.pheromone[self.current][i] \*\* self.colony.alpha \* self.eta[self.current][

i] \*\* self.colony.beta

probabilities = [0 for i in range(self.graph.rank)]

for i in range(self.graph.rank):

try:

self.allowed.index(i)

probabilities[i] = self.graph.pheromone[self.current][i] \*\* self.colony.alpha \* \

self.eta[self.current][i] \*\* self.colony.beta / denominator

except ValueError:

pass

selected = 0

rand = random.random()

for i, probability in enumerate(probabilities):

rand -= probability

if rand <= 0:

selected = i

break

self.allowed.remove(selected)

self.tabu.append(selected)

self.total\_cost += self.graph.matrix[self.current][selected]

self.current = selected

def \_update\_pheromone\_delta(self):

self.pheromone\_delta = [[0 for j in range(self.graph.rank)] for i in range(self.graph.rank)]

for \_ in range(1, len(self.tabu)):

i = self.tabu[\_ - 1]

j = self.tabu[\_]

if self.colony.update\_strategy == 1:

self.pheromone\_delta[i][j] = self.colony.Q

elif self.colony.update\_strategy == 2:

self.pheromone\_delta[i][j] = self.colony.Q / self.graph.matrix[i][j]

else:

self.pheromone\_delta[i][j] = self.colony.Q / self.total\_cost

def plot(points, path: list):

x = []

y = []

for point in points:

x.append(point[0])

y.append(point[1])

y = list(map(operator.sub, [max(y) for i in range(len(points))], y))

plt.plot(x, y, 'co')

for \_ in range(1, len(path)):

i = path[\_ - 1]

j = path[\_]

plt.arrow(x[i], y[i], x[j] - x[i], y[j] - y[i], color='r', length\_includes\_head=True)

plt.xlim(0, max(x) \* 1.1)

plt.ylim(0, max(y) \* 1.1)

plt.show()

def distance(city1: dict, city2: dict):

return math.sqrt((city1['x'] - city2['x']) \*\* 2 + (city1['y'] - city2['y']) \*\* 2)

def main():

cities = []

points = []

startTime = time.time()

with open('/content/drive/MyDrive/source/200-Machines/TRP-S200-R1.txt') as f:

for line in f.readlines():

city = line.split(' ')

cities.append(dict(index=int(city[0]), x=int(city[1]), y=int(city[2])))

points.append((int(city[1]), int(city[2])))

cost\_matrix = []

rank = len(cities)

for i in range(rank):

row = []

for j in range(rank):

row.append(distance(cities[i], cities[j]))

cost\_matrix.append(row)

aco = ACO(10, 100, 1.0, 10.0, 0.5, 10, 2)

graph = Graph(cost\_matrix, rank)

path, cost = aco.solve(graph)

print('cost: {}, path: {}'.format(cost, path))

plot(points, path)

endTime = time.time()

time\_taken = endTime - startTime

print("Time taken :")

print(time\_taken)

if \_\_name\_\_ == '\_\_main\_\_':

main()