**ASSIGNMENT 9A (ANT COLONY ALGORITHM )**

import random as rn

import numpy as np

from numpy.random import choice as np\_choice

class AntColony(object):

def \_\_init\_\_(self, distances, n\_ants, n\_best, n\_iterations, decay, alpha=1, beta=1):

"""

Args:

distances (2D numpy.array): Square matrix of distances. Diagonal is assumed to be np.inf.

n\_ants (int): Number of ants running per iteration

n\_best (int): Number of best ants who deposit pheromone

n\_iteration (int): Number of iterations

decay (float): Rate it which pheromone decays. The pheromone value is multiplied by decay, so 0.95 will lead to decay, 0.5 to much faster decay.

alpha (int or float): exponenet on pheromone, higher alpha gives pheromone more weight. Default=1

beta (int or float): exponent on distance, higher beta give distance more weight. Default=1

Example:

ant\_colony = AntColony(german\_distances, 100, 20, 2000, 0.95, alpha=1, beta=2)

"""

self.distances = distances

self.pheromone = np.ones(self.distances.shape) / len(distances)

self.all\_inds = range(len(distances))

self.n\_ants = n\_ants

self.n\_best = n\_best

self.n\_iterations = n\_iterations

self.decay = decay

self.alpha = alpha

self.beta = beta

def run(self):

shortest\_path = None

all\_time\_shortest\_path = ("placeholder", np.inf)

for i in range(self.n\_iterations):

all\_paths = self.gen\_all\_paths()

self.spread\_pheronome(all\_paths, self.n\_best, shortest\_path=shortest\_path)

shortest\_path = min(all\_paths, key=lambda x: x[1])

print (shortest\_path)

if shortest\_path[1] < all\_time\_shortest\_path[1]:

all\_time\_shortest\_path = shortest\_path

self.pheromone \* self.decay

return all\_time\_shortest\_path

def spread\_pheronome(self, all\_paths, n\_best, shortest\_path):

sorted\_paths = sorted(all\_paths, key=lambda x: x[1])

for path, dist in sorted\_paths[:n\_best]:

for move in path:

self.pheromone[move] += 1.0 / self.distances[move]

def gen\_path\_dist(self, path):

total\_dist = 0

for ele in path:

total\_dist += self.distances[ele]

return total\_dist

def gen\_all\_paths(self):

all\_paths = []

for i in range(self.n\_ants):

path = self.gen\_path(0)

all\_paths.append((path, self.gen\_path\_dist(path)))

return all\_paths

def gen\_path(self, start):

path = []

visited = set()

visited.add(start)

prev = start

for i in range(len(self.distances) - 1):

move = self.pick\_move(self.pheromone[prev], self.distances[prev], visited)

path.append((prev, move))

prev = move

visited.add(move)

path.append((prev, start)) # going back to where we started

return path

def pick\_move(self, pheromone, dist, visited):

pheromone = np.copy(pheromone)

pheromone[list(visited)] = 0

row = pheromone \*\* self.alpha \* (( 1.0 / dist) \*\* self.beta)

norm\_row = row / row.sum()

move = np\_choice(self.all\_inds, 1, p=norm\_row)[0]

return move

import numpy as np

distances = np.array([[np.inf, 2, 2, 5, 7],

[2, np.inf, 4, 8, 2],

[2, 4, np.inf, 1, 3],

[5, 8, 1, np.inf, 2],

[7, 2, 3, 2, np.inf]])

ant\_colony = AntColony(distances, 1, 1, 100, 0.95, alpha=1, beta=1)

shortest\_path = ant\_colony.run()

print ("shorted\_path: {}".format(shortest\_path))

**OUTPUT:**



