Ant Colony Optimization

Program:

import numpy as np

from numpy import inf

#given values for the problems

d = np.array([[0,10,12,11,14]

,[10,0,13,15,8]

,[12,13,0,9,14]

,[11,15,9,0,16]

,[14,8,14,16,0]])

iteration = 100

n\_ants = 5

n\_citys = 5

# intialization part

m = n\_ants

n = n\_citys

e = .5 #evaporation rate

alpha = 1 #pheromone factor

beta = 2 #visibility factor

#calculating the visibility of the next city visibility(i,j)=1/d(i,j)

visibility = 1/d

visibility[visibility == inf ] = 0

#intializing pheromne present at the paths to the cities

pheromne = .1\*np.ones((m,n))

#intializing the rute of the ants with size rute(n\_ants,n\_citys+1)

#note adding 1 because we want to come back to the source city

rute = np.ones((m,n+1))

for ite in range(iteration):

rute[:,0] = 1 #initial starting and ending positon of every ants '1' i.e city '1'

for i in range(m):

temp\_visibility = np.array(visibility) #creating a copy of visibility

for j in range(n-1):

#print(rute)

combine\_feature = np.zeros(5) #intializing combine\_feature array to zero

cum\_prob = np.zeros(5) #intializing cummulative probability array to zeros

cur\_loc = int(rute[i,j]-1) #current city of the ant

temp\_visibility[:,cur\_loc] = 0 #making visibility of the current city as zero

p\_feature = np.power(pheromne[cur\_loc,:],beta) #calculating pheromne feature

v\_feature = np.power(temp\_visibility[cur\_loc,:],alpha) #calculating visibility feature

p\_feature = p\_feature[:,np.newaxis] #adding axis to make a size[5,1]

v\_feature = v\_feature[:,np.newaxis] #adding axis to make a size[5,1]

combine\_feature = np.multiply(p\_feature,v\_feature) #calculating the combine feature

total = np.sum(combine\_feature) #sum of all the feature

probs = combine\_feature/total #finding probability of element probs(i) =

comine\_feature(i)/total

cum\_prob = np.cumsum(probs) #calculating cummulative sum

#print(cum\_prob)

r = np.random.random\_sample() #randon no in [0,1)

#print(r)

city = np.nonzero(cum\_prob>r)[0][0]+1 #finding the next city having probability

higher then random(r)

#print(city)

rute[i,j+1] = city #adding city to route

left = list(set([i for i in range(1,n+1)])-set(rute[i,:-2]))[0] #finding the last untraversed

city to route

rute[i,-2] = left #adding untraversed city to route

rute\_opt = np.array(rute) #intializing optimal route

dist\_cost = np.zeros((m,1)) #intializing total\_distance\_of\_tour with zero

for i in range(m):

s = 0

for j in range(n-1):

s = s + d[int(rute\_opt[i,j])-1,int(rute\_opt[i,j+1])-1] #calcualting total tour distance

dist\_cost[i]=s #storing distance of tour for 'i'th ant at location 'i'

dist\_min\_loc = np.argmin(dist\_cost) #finding location of minimum of dist\_cost

dist\_min\_cost = dist\_cost[dist\_min\_loc] #finging min of dist\_cost

best\_route = rute[dist\_min\_loc,:] #intializing current traversed as best route

pheromne = (1-e)\*pheromne #evaporation of pheromne with (1-e)

for i in range(m):

for j in range(n-1):

dt = 1/dist\_cost[i]

pheromne[int(rute\_opt[i,j])-1,int(rute\_opt[i,j+1])-1] = pheromne[int(rute\_opt[i,j])-

1,int(rute\_opt[i,j+1])-1] + dt

#updating the pheromne with delta\_distance

#delta\_distance will be more with min\_dist i.e adding more weight to that route

peromne

print('route of all the ants at the end :')

print(rute\_opt)

print()

print('best path :',best\_route)

print('cost of the best path',int(dist\_min\_cost[0]) + d[int(best\_route[-2])-1,0])

Output:

Route of all ants at the end:

[[1.4.3.5.2.1]

[1.4.3.5.2.1]

[1.4.3.5.2.1]

[1.4.3.5.2.1]

[1.4.3.5.2.1]]

Best path: [1.4.3.5.2.1]

Cost of the best path=52.