Mathematics IA

The mathematics behind the Ant Colony Optimization Algorithm

Personal Code: hzw206

Session: May 2021

Page Count: 17

Introduction

I had completed my internship in the field of machine learning (ML) during my sophomore year in high school Initially, I had no idea what to do as I didn't have any experience in the same for developing such programs. Being a beginner in the field, one of the employees suggested that I could understand the mathematics correctly in an algorithmic manner by completing online university courses. It was during these courses that I felt I should further develop my understanding of the utilization of probability and associate it with real life scenarios and situations such as designing circuit boards in the field of nanotechnology.

Thinking about this, led me to the Travelling Salesman Problem (TSP) - calculating the path an individual can take when visiting many cities having the least distance - that was discussed. Being perplexed, I researched more into how the TSP problem can be solved, because I felt that understanding the mathematics behind solving the TSP Problem would allow me to get a fair understanding of how the software designers work. While I found many algorithms, I could not personally understand them due to insufficient knowledge on the same and how it can be connected to other real life scenarios. Finally, I came across an algorithm called the **Ant Colony Optimization** (ACO), a combinatorial optimization algorithm which was linked to how ants work in real life to ultimately travel in the most optimal route.[1]

[1] Selvi, V., and Dr.R. Umarani. "Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques." *International Journal of Computer Applications*, vol. 5, no. 4, 2010, pp. 1–6., doi:10.5120/908-1286.

Rationale

I felt that understanding the mathematics involved in the ACO Algorithm would lead me closer to understand how software designers work, how electric towers are placed along the city so as to minimize the total wire used, autonomous car routing and also discover other applications such as in medicine.

Theory

In the natural world, upon finding food an ant returns to its colony while laying down pheromone trails. If other ants find such a path, it would follow the trail, returning and reinforcing it if it eventually finds food.

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, however, will get marched over more frequently, and thus the pheromone density will be higher on the short path when compared to a longer path. Hence, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and give positive feedback which eventually leads to many ants following a single short path.

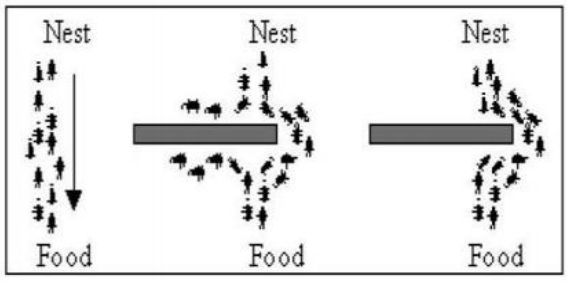

Figure 1: Behavior of real ant movements[2]

Ant Colony Optimization Algorithm: Calculations Involved

In the ACO Algorithm, to simulate the actual process of how an ant colony works, artificial ants are created that will search for an optimal solution for the optimization problem i.e. in this case the TSP Problem. As the process is iterative, in the first step of the iterations, every ant stochastically constructs a solution. In the second step, the paths found by the different ants are compared based on which the pheromone levels on the nodes are updated.[3]

Now, we know that each ant will need to create a solution to move. So we can say that each ant is moving from a node *x* to node *y*. Let us suppose that there are *k* ants in an ant colony, therefore for an ant k, the probability of p^k_{xy} for the ant to from move from one node *x* to another node *y* has to be calculated, which will ultimately be based on a certain level of attractiveness, η_{xy} for the ant which is a priori desirability, meaning that this value is not based on any past experience. It is also based on the trail level τ_{xy} that's based on the past experience (a posteriori desirability) indicating the past success to make choose that particular node.

[2] Selvi, V., and Dr.R. Umarani. "Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques." *International Journal of Computer Applications*, vol. 5, no. 4, 2010, pp. 1–6., doi:10.5120/908-1286.

[3] "Ant Colony Optimization Algorithms." *Wikipedia*, Wikimedia Foundation, 14 Mar. 2021, en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

Therefore, the probability that a k^{th} ant to move from node x to node y on the route is modelled by[4]:

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{z \in \text{allowed}_x} (\tau_{xz}^{\alpha})(\eta_{xz}^{\beta})}$$ Equation (1)

where τ_{xy} is the amount of pheromone deposited for travelling from node x to node y, α is a control parameter that can be fixed to a value greater than or equal to 0 (0 as the pheromone level of the path will decrease if the path is less travelled) and will control the influence of τ_{xy}, η_{xy} is the level of attractiveness i.e. the desirability for the k ant for the travelled path xy which is equal to $1/d_{xy}$ where d is the distance of the trail xy. This is so because if the path is longer, the paths desirability should be reduced whereas it should increase if the path is shorter. This is therefore facilitated by the inverse relationship. β is a control parameter which is a bias (influence control of level of attractiveness). τ_{xy} and η_{xz} represent the trail level and attractiveness for the other possible paths.

The desirability for the k[th] ant for the path xy is calculated using the formula[5]:

$$\eta_{xy} = \frac{1}{d_{xy}}$$ where d_{xy} is the distance between x and y Equation (2)

[4] "Ant Colony Optimization Algorithms." *Wikipedia*, Wikimedia Foundation, 14 Mar. 2021, en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

[5] Liu, ChunYing, and Jijiang Yu. "Multiple Depots Vehicle Routing Based on the Ant Colony with the Genetic Algorithm." *Journal of Industrial Engineering and Management*, www.jiem.org/index.php/jiem/article/view/747/519.

In addition to this, the trails also have to be updated accordingly when each ant has completed their path travel (in this case for the IA, solution), increasing or decreasing the level of trails corresponding to moves that were part of "good" or "bad" paths, respectively. This update can hence be facilitated using the formula[6]:

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k} \Delta\tau_{xy}^{k} \qquad \text{Equation (3)}$$

where, as stated before, τ_{xy} is the amount of pheromone deposited for travelling from node x to node y, ρ a control parameter which corresponds to the pheromone evaporation coefficient and

$\Delta\tau_{xy}$ is the amount of pheromone that is deposited by the k^{th} ant and is calculated using the formula[7]:

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \qquad \text{Equation (4)}$$

Where L_k is the cost function value of the kth ant's tour i.e. length of the ant's tour and Q being the control variable that affects the amount of pheromone deposited if the ant k uses the path xy in its travel.

Based on these, a personally designed experimentation utilizing the ACO Algorithm in the programming language Python will be done to demonstrate the working of the algorithm, to show

[6] "Ant Colony Optimization Algorithms." *Wikipedia*, Wikimedia Foundation, 14 Mar. 2021, en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

[7] "Ant Colony Optimization Algorithms." *Wikipedia*, Wikimedia Foundation, 14 Mar. 2021, en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

how the path taken by the ant gets updated to one that has the minimal length, display the probability calculated for each node in the path.

Experiment

To understand the algorithm more efficiently and to model a probability distribution function, I took 4 points on the graph labelled as A, B, C, D. There are (n-1)! number of possibilities, in this case it is (4-1)! which means there are 6 possible routes for the ants to choose to travel to the food source and return back to the starting point (in this case A). Although in this case there are only 9 possible routes, the number of possibilities increases as the number of vertices increase. Therefore, due to the size limitation of the essay, the experiment and the probability distribution function was modelled for the 9 possible routes.

To visualize the task at hand while carrying out the experiment better, a Hamilton diagram connecting all the points with each other was drawn. A Hamiltonian cycle is a closed loop on a graph where every node (vertex) is visited exactly once and at the end returns back to the starting node.[8] If a graph with more than one node (i.e. a non-singleton graph) has a Hamiltonian cycle, we call it a Hamiltonian graph. The nodes of the graph in this case are the vertices A, B, C, D that are to be taken into consideration. This allow us to gain a visual picture of all the possible routes and combinations.

[8] Stephanie. "Hamiltonian Cycle: Simple Definition and Example." *Statistics How To*, 13 Nov. 2017, www.statisticshowto.datasciencecentral.com/hamiltonian-cycle/.

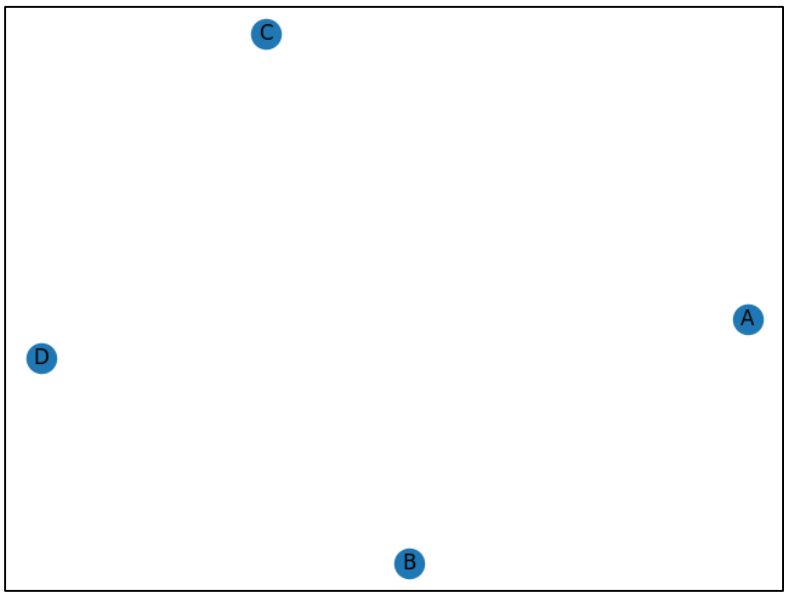

Figure 2: Location of vertices

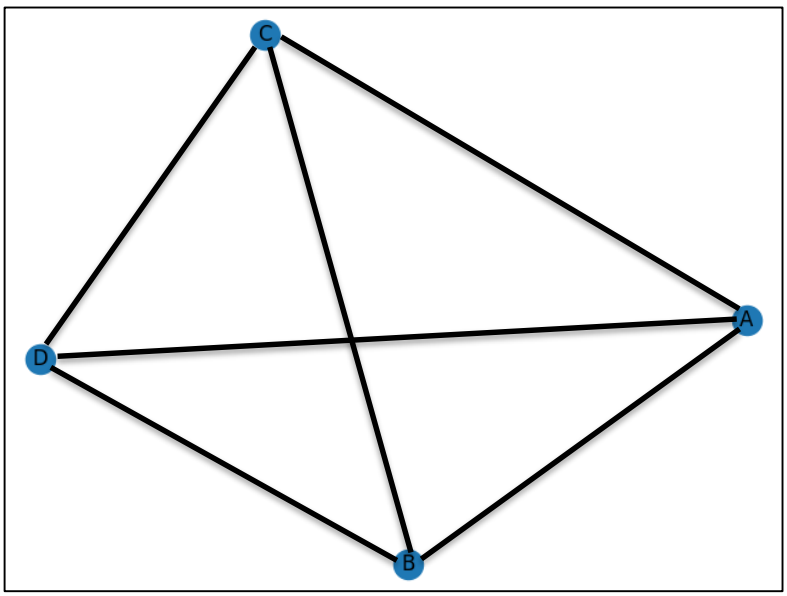

Figure 3: Hamiltonian graph of the vertices

The distance matrix for the Hamiltonian graph is given below:

	A	B	C	D
A	0	10	12	11
B	10	0	13	15
C	12	13	0	9
D	11	15	9	0

Matrix A

The following matrix shows the distance (in units) between the vertices taken in the experiment.

To carry out the experiment, the number of artificial ants is set to 6 and the number of iterations to 10. Since the ants travel from vertex A and travels back to vertex A, the probabilities of choosing the vertex B, C, D on the route for next step is calculated only and the vertex on the route which has the highest probability is chosen by the ant. Thus, using equations 1, 2, 3, the values of probability are calculated for all the ants in each iteration.

Calculation Example

Depicted below is an example of the calculation of the probability for choosing the vertices in the 1st iteration by ant 1 for its 2nd step in its travel using equations 1, 2, 3, and 4:

The vertices A, B, C, D are mapped to integer values starting from 0. Therefore, A = 0, B = 1, C = 2, D = 3 indexes.

$\alpha = 2$ -- (α is the pheromone factor) $\beta = 1$ -- (β is the desirability factor)

$e = 0.5$ -- (e is the evaporation factor)

$n = 10$ -- (no: of iterations) $Q = 1$

τ = matrix containing value of pheromone deposited by ant when moving from one vertex to another

$$= \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}$$

$\eta = \frac{1}{Matrix\ A}$ if element in Matrix A $\neq 0$ else $\eta = 0$ -- (k is equal 1 in the formula as ant 1 is travelling)

$$= \begin{bmatrix} 0 & 0.1 & 0.083 & 0.09 \\ 0.1 & 0 & 0.076 & 0.066 \\ 0.083 & 0.076 & 0 & 0.11 \\ 0.09 & 0.066 & 0.11 & 0 \end{bmatrix}$$

curr_loc = current location of the ant = 0 *temp_desirability* = η

$\{temp\_desirability[i, cur\_loc] : i \in \{0,1,2,3\}\} = 0$ which implies that

$$temp\_desirability[i, cur\_loc] = \begin{bmatrix} 0 & 0.1 & 0.083 & 0.09 \\ 0 & 0 & 0.076 & 0.066 \\ 0 & 0.076 & 0 & 0.11 \\ 0 & 0.066 & 0.11 & 0 \end{bmatrix}$$ -- (making desirability of the current city as zero)

$\tau_{xy} = \tau\,[cur\_loc, i]$

$= [0.1 \quad 0.1 \quad 0.1 \quad 0.1]$

$\eta_{xy} = temp\_desirability[cur\_loc, i]$

$= [0 \quad 0.1 \quad 0.083 \quad 0.09]$

The probability can then be modelled as:

$$Probability = \frac{\tau_{xy}^{\alpha} \times \eta_{xy}^{\beta}}{\sum_{k=0}^{3} (\tau_{xy}^{\alpha} \times \eta_{xy}^{\beta})_k}$$

$$= \frac{[0.1 \quad 0.1 \quad 0.1 \quad 0.1]^2 \times [0 \quad 0.1 \quad 0.083 \quad 0.09]^1}{\sum_{k=0}^{3} [0.1 \quad 0.1 \quad 0.1 \quad 0.1]^2 \times [0 \quad 0.1 \quad 0.083 \quad 0.09]^1{}_k}$$

$$= \frac{[0 \quad 0.001 \quad 0.00083 \quad 0.0009]}{\sum_{k=0}^{3} [0 \quad 0.001 \quad 0.00083 \quad 0.0009]_k}$$

$$= \frac{[0 \quad 0.001 \quad 0.00083 \quad 0.0009]}{[0+0.001+0.00083+0.0009]}$$

$$= [0 \quad 0.364 \quad 0.303 \quad 0.333]$$

Since the probability is the greatest in the second index, ant 1 would travel from A to B in its next step in iteration 1. Similarly, the probability is again re-calculated for choosing the next vertex as part of its tour from B as the value of the second index of the array ‘Probability’ will be updated to 0 as the ant is now at that location. Depicted below is thus the probability modelled at each step for choosing the next vertex by ant 1 in iteration 1:

	Probability of choosing vertex A	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 1	0	0.364	0.303	0.333
Step 2	0	0	0.409	0.591
Step 3	0	0	1	0

Table 1: Values of Probability for ant 1

Thus, the path taken by ant 1 is ABDCA.

Additionally, after ant 1 moves to B, the path also has to be updated accordingly, increasing or decreasing the level desirability of the path corresponding to moves that were part of "good" or "bad" paths. Therefore, the amount of pheromone deposited that has to be changed based on whether the path AB taken was good or bad is calculated using equation 4:

$$\Delta\tau_{AB} = \frac{Q}{d_{AB}}$$

$$= \frac{1}{46} = 0.021$$

Now after travelling, given time, the pheromone will evaporate and therefore has to be updated. Therefore, the amount of pheromone now in the vertices visible to the next ant in iteration 1 i.e. ant 2 can be calculated using equation 2:

$$\tau = (1 - e) \times \tau$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$$

Updating the pheromone level deposited by the next ant if the ant moves to B first from the starting point A is therefore done using equation 3:

$$\tau\,[0,0] = \tau\,[0,0] + \Delta\tau_{AB}$$

$$= 0.05 + 0.021$$

$$= 0.071$$

Therefore, $\tau = \begin{bmatrix} 0.071 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$

Therefore, if the path taken AB is longer, the inverse relationship between $\Delta\tau_{AB}$ and d_{AB} ensures that the amount of ant deposited by the next ant if it uses the path will be relatively small when compared to if the path AB is shorter, thus influencing the final pheromone value τ that will be deposited by the next ant. This allows the next set of ants to choose the next vertex in the route based on the updated pheromone deposit value, wherein, the ants will choose the vertex having a relatively greater amount of pheromone deposit τ when compared to other vertices as the vertex with higher value of τ demonstrates that that is the vertex most travelled and is shorter.

Data Collected

Depicted below are few of the data collected for ant 1, ant 2, and ant 5 in the 1st iteration, 2nd iteration and last iteration (10th iteration):

Iteration 1

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.364	0.303	0.333
Step 3	0	0.409	0.591
Step 4	0	1	0

Table 2: Values of Probability for ant 1

Thus, the path taken by ant 1 is ABDCA.

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.364	0.303	0.333
Step 3	0	0.375	0.625
Step 4	0	1	0

Table 3: Values of Probability for ant 2

Thus, the path taken by ant 2 is ABDCA.

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.364	0.303	0.333
Step 3	0	0.535	0.465
Step 4	0	0	1

Table 4: Values of Probability for ant 5

Thus, the path taken by ant 5 is ABCDA.

Iteration 2

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.209	0.530	0.261
Step 3	0.220	0	0.780
Step 4	1	0	0

Table 5: Values of Probability for ant 5

Thus, the path taken by ant 1 is ACDBA.

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.209	0.261	0.530
Step 3	0.212	0.788	0
Step 4	1	0	0

Table 6: Values of Probability for ant 2

Thus, the path taken by ant 2 is ADCBA.

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.028	0.034	0.938
Step 3	0.016	0.984	0
Step 4	1	0	0

Table 7: Values of Probability for ant 5

Thus, the path taken by ant 2 is ADCBA.

Iteration 10

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.209	0.261	0.530
Step 3	0.220	0.780	0
Step 4	1	0	0

Table 8: Values of Probability for ant 1

Thus, the path taken by ant 1 is ADCBA.

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.209	0.261	0.530
Step 3	0.212	0.788	0
Step 4	1	0	0

Table 9: Values of Probability for ant 2

Thus, the path taken by ant 2 is ADCBA.

	Probability of choosing vertex B	Probability of choosing vertex C	Probability of choosing vertex D
Step 2	0.028	0.034	0.938
Step 3	0.016	0.984	0
Step 4	1	0	0

Table 10: Values of Probability for ant 5

Thus, the path taken by ant 2 is ADCBA.

Results

The experiment was carried out and completed. After completion of the experiment, the number of times a path taken was recorded and depicted below:

	Route Chosen	**ABCDA**	**ABDCA**	**ACDBA**	**ACBDA**	**ADCBA**	**ADBCA**
Number of times travelled by the ants		10	4	6	3	24	3

Table 11: Number of times route chosen

Comparing the table values in the 'Data Collected' section, we see that in the first few iterations (Iteration 1 and 2), the ants initially travelled along different routes including routes ACDBA and ADBCA which are comparatively longer (53 units) when compared to other routes such as ACDBA and ADCBA which have a shorter distance to be covered (46 and 43 units respectively). However, in the last iterations, we see that a majority of the ants have shifted their path to that of ADCBA, the most optimal route and is supported by the result recorded in Table 11.

Conclusion

From the calculation example demonstrated, we see how the artificial ants change their route accordingly based on the amount of pheromones on the path travelled and based on which they update the pheromones deposited based on whether it's an optimal path. Moreover, for the probability of choosing a particular node (vertex) while moving from the current node modelled for future prediction, we see that the calculation involved is dynamic as the probability keeps changing for every step for the particular ant in each iteration as the matrices are continuously updated in each step. Due to this, the normal representation of the probability function modelled cannot be represented using a discrete probability distribution graph.

Additionally, while depicting the calculations, while we have defined a distance matrix, we see that it is not utilized in the dynamic probability distribution modelled to calculate the probability of choosing a particular vertex (node) for its tour but it rather used to calculate the desirability of choosing a particular node for the next step as nodes that appear far away for the ants (low visibility) will be less 'desirable' to travel. Therefore, from this we can draw an inference that in the real world, the ants might rather utilize the time taken to reach the food source based on which the pheromone levels deposited will be updated accordingly.

Thus, by understanding the mathematics involved behind the algorithm, I was successfully able to design a mini-circuit wherein the amount of wire utilized was minimum. However, due to the algorithm's iterative nature, the time taken to converge is uncertain, and therefore might not be efficient for carrying out memory-intensive experiments.

Bibliography

1. “Ant Colony Optimization Algorithms.” *Wikipedia*, Wikimedia Foundation, 14 Mar. 2021, en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

2. Liu, ChunYing, and Jijiang Yu. “Multiple Depots Vehicle Routing Based on the Ant Colony with the Genetic Algorithm.” *Journal of Industrial Engineering and Management*, www.jiem.org/index.php/jiem/article/view/747/519.

3. Selvi, V., and Dr.R. Umarani. “Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques.” *International Journal of Computer Applications*, vol. 5, no. 4, 2010, pp. 1–6., doi:10.5120/908-128

4. Stephanie. “Hamiltonian Cycle: Simple Definition and Example.” *Statistics How To*, 13 Nov. 2017, www.statisticshowto.datasciencecentral.com/hamiltonian-cycle/.

Appendix

Source code

```
import numpy as np
from numpy import inf
import matplotlib.pyplot as plt
import networkx as nx

# initializing distance matrix


d = np.array([[0, 10, 12, 11]
                  , [10, 0, 13, 15]
                  , [12, 13, 0, 9]
                  , [11, 15, 9, 0]])

mapping = {0: "A", 1: "B", 2: "C", 3: "D"}

G = nx.from_numpy_matrix(d)
print(nx.number_of_nodes(G))
G = nx.relabel_nodes(G, mapping)
nx.draw(G, with_labels=True)
plt.show()

iteration = 10
n_ants = 5
n_citys = 4

# intialization part

m = n_ants
n = n_citys
e = .5  # evaporation rate
beta = 1  # pheromone factor
alpha = 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):
    print("For iteration " + str(ite))

    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(4)  # intializing combine_feature
array to zero
            cum_prob = np.zeros(4)  # 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, :], alpha)  # calculating
pheromne feature
            v_feature = np.power(temp_visibility[cur_loc, :], beta)  #
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]

            # noinspection PyRedeclaration
            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
            print("For ant " + str(i + 1))
            print(probs)

            # noinspection PyRedeclaration
            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
        print(rute)

    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):
            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):
            print("hello")
            print(dist_cost[i])
            # input()
            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])
```