**Ant Colony Algorithm**

**Travelling Salesman:**

import numpy as np

import random

class AntColonyTSP:

    def \_\_init\_\_(self, dist\_matrix, n\_ants, n\_iterations, decay, alpha=1, beta=2):

        self.dist\_matrix = dist\_matrix

        self.n\_ants = n\_ants

        self.n\_iterations = n\_iterations

        self.decay = decay

        self.alpha = alpha

        self.beta = beta

        self.n\_cities = len(dist\_matrix)

        self.pheromone = np.ones((self.n\_cities, self.n\_cities)) / self.n\_cities

    def run(self):

        shortest\_path = None

        shortest\_distance = float('inf')

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

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

            self.spread\_pheromone(all\_paths)

            for path, dist in all\_paths:

                if dist < shortest\_distance:

                    shortest\_distance = dist

                    shortest\_path = path

            self.pheromone \*= self.decay

        return shortest\_path, shortest\_distance

    def spread\_pheromone(self, all\_paths):

        for path, dist in all\_paths:

            for move in path:

                self.pheromone[move] += 1.0 / dist

    def gen\_path(self, start):

        visited = set()

        visited.add(start)

        path = []

        current\_city = start

        while len(visited) < self.n\_cities:

            next\_city = self.pick\_next\_city(current\_city, visited)

            path.append((current\_city, next\_city))

            visited.add(next\_city)

            current\_city = next\_city

        return path

    def pick\_next\_city(self, current\_city, visited):

        pheromone = self.pheromone[current\_city]

        for city in visited:

            pheromone[city] = 0

        pheromone = pheromone \*\* self.alpha

        distance = np.array([self.dist\_matrix[current\_city][i] for i in range(self.n\_cities)])

        distance = np.where(distance == 0, np.inf, distance)  # Prevent division by zero for self-loops

        distance = 1 / distance \*\* self.beta

        probability = pheromone \* distance

        if np.sum(probability) == 0:

            probability = np.ones(self.n\_cities) / self.n\_cities  # uniform probability if sum is zero

        probability = probability / np.sum(probability)  # Normalize the probabilities

        return np.random.choice(range(self.n\_cities), p=probability)

    def gen\_all\_paths(self):

        all\_paths = []

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

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

            path = self.gen\_path(start)

            distance = self.calculate\_path\_distance(path)

            all\_paths.append((path, distance))

        return all\_paths

    def calculate\_path\_distance(self, path):

        distance = 0

        for move in path:

            from\_city, to\_city = move

            distance += self.dist\_matrix[from\_city][to\_city]

        from\_city, to\_city = path[-1]

        distance += self.dist\_matrix[to\_city][from\_city]  # Return to the start city

        return distance

# Example distance matrix for a small TSP problem

dist\_matrix = np.array([

    [0, 2, 2, 5, 7],

    [2, 0, 4, 8, 2],

    [2, 4, 0, 1, 3],

    [5, 8, 1, 0, 4],

    [7, 2, 3, 4, 0]

])

# Initialize and run the Ant Colony Optimization algorithm

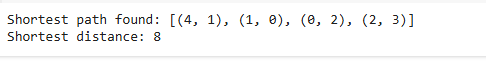
aco = AntColonyTSP(dist\_matrix=dist\_matrix, n\_ants=10, n\_iterations=100, decay=0.95)

shortest\_path, shortest\_distance = aco.run()

print("Shortest path found:", shortest\_path)

print("Shortest distance:", shortest\_distance)

**OUTPUT:**

****