|  |  |  |  |
| --- | --- | --- | --- |
|  | Bansilal Ramnath Agarwal Charitable Trust's  Vishwakarma Institute of Information Technology  **Department of**  **Artificial Intelligence and Data Science** | | |
| Name: Siddhesh Dilip Khairnar | | | |
| Class: TY | Division: B | | Roll No: 372028 |
| Semester: V | | Academic Year: 2023-2024 | |
| Subject Name & Code: ADUA31201: Artificial Intelligence | | | |
| Title of Assignment: A salesperson visits a number of cities exactly once and return to the first city, find the shortest route (Hill Climbing) | | | |
| Date of Performance:24-10-2023 | | Date of Submission: 11-11-2023 | |

**ASSIGNMENT NO. 6**

**CODE:**

import random

import numpy as np

import networkx as nx

coordinate = np.array([[1, 2], [30, 21], [56, 23], [8, 18], [20, 50], [3, 4], [11, 6], [

                      6, 7], [15, 20], [10, 9], [12, 12], [46, 17], [60, 55], [100, 80], [16, 13]])

def generate\_matrix(coordinate):

    matrix = []

    for i in range(len(coordinate)):

        for j in range(len(coordinate)):

            p = np.linalg.norm(coordinate[i] - coordinate[j])

            matrix.append(p)

    matrix = np.reshape(matrix, (len(coordinate), len(coordinate)))

    return matrix

def solution(matrix):

    points = list(range(0, len(matrix)))

    solution = []

    for i in range(0, len(matrix)):

        random\_point = points[random.randint(0, len(points) - 1)]

        solution.append(random\_point)

        points.remove(random\_point)

    return solution

def path\_length(matrix, solution):

    cycle\_length = 0

    for i in range(0, len(solution)):

        cycle\_length += matrix[solution[i]][solution[i - 1]]

    return cycle\_length

def neighbors(matrix, solution):

    neighbors = []

    for i in range(len(solution)):

        for j in range(i + 1, len(solution)):

            neighbor = solution.copy()

            neighbor[i] = solution[j]

            neighbor[j] = solution[i]

            neighbors.append(neighbor)

    best\_neighbor = neighbors[0]

    best\_path = path\_length(matrix, best\_neighbor)

    for neighbor in neighbors:

        current\_path = path\_length(matrix, neighbor)

        if current\_path < best\_path:

            best\_path = current\_path

            best\_neighbor = neighbor

    return best\_neighbor, best\_path

def hill\_climbing(coordinate):

    matrix = generate\_matrix(coordinate)

    current\_solution = solution(matrix)

    current\_path = path\_length(matrix, current\_solution)

    neighbor = neighbors(matrix, current\_solution)[0]

    best\_neighbor, best\_neighbor\_path = neighbors(matrix, neighbor)

    while best\_neighbor\_path < current\_path:

        current\_solution = best\_neighbor

        current\_path = best\_neighbor\_path

        neighbor = neighbors(matrix, current\_solution)[0]

        best\_neighbor, best\_neighbor\_path = neighbors(matrix, neighbor)

    return current\_path, current\_solution

def graph(coordinate):

    final\_solution = hill\_climbing(coordinate)

    G = nx.DiGraph()

    temp = final\_solution[1]

    G.add\_nodes\_from(final\_solution[1])

    for i in range(1, len(final\_solution[1])):

        G.add\_edge(temp[i - 1], temp[i])

    G.add\_edge(temp[len(temp) - 1], temp[0])

    color\_map = []

    for node in G:

        if node == final\_solution[1][0]:

            color\_map.append('lime')

        else:

            color\_map.append('plum')

    nx.draw(G, with\_labels=True, node\_color=color\_map, node\_size=1000)

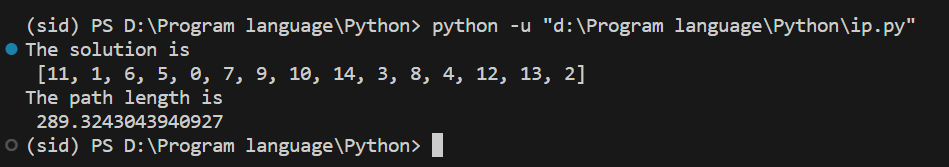
    print("The solution is \n",

          final\_solution[1], "\nThe path length is \n", final\_solution[0])

    return

graph(coordinate)

**OUTPUT:**

****

