End Semester Project

33310: Artificial Intelligence

26001904131
Cho keunhee

I. Unpack the Data.zip archive, which you received together with this document via e-mail, to obtain the data (in the CSV format) of the semester-end project. Observe that the Data.csv file stores the adjacency matrix of a (small) transportation network represented by a graph.

i) Using the adjacency matrix, construct the undirected graph represented by the matrix, and visualize it. How many links (edges) are in the graph? How many nodes? (Write down the answers.)

In order to visualize the csv file given. This paper has used networkx and numpy library. First, converted the given csv file into a numpy list, then used networkx to an adjacency matrix. For the visualization node size was set as one and edges width has been set as 0.3. The following is the visualization of the given data.

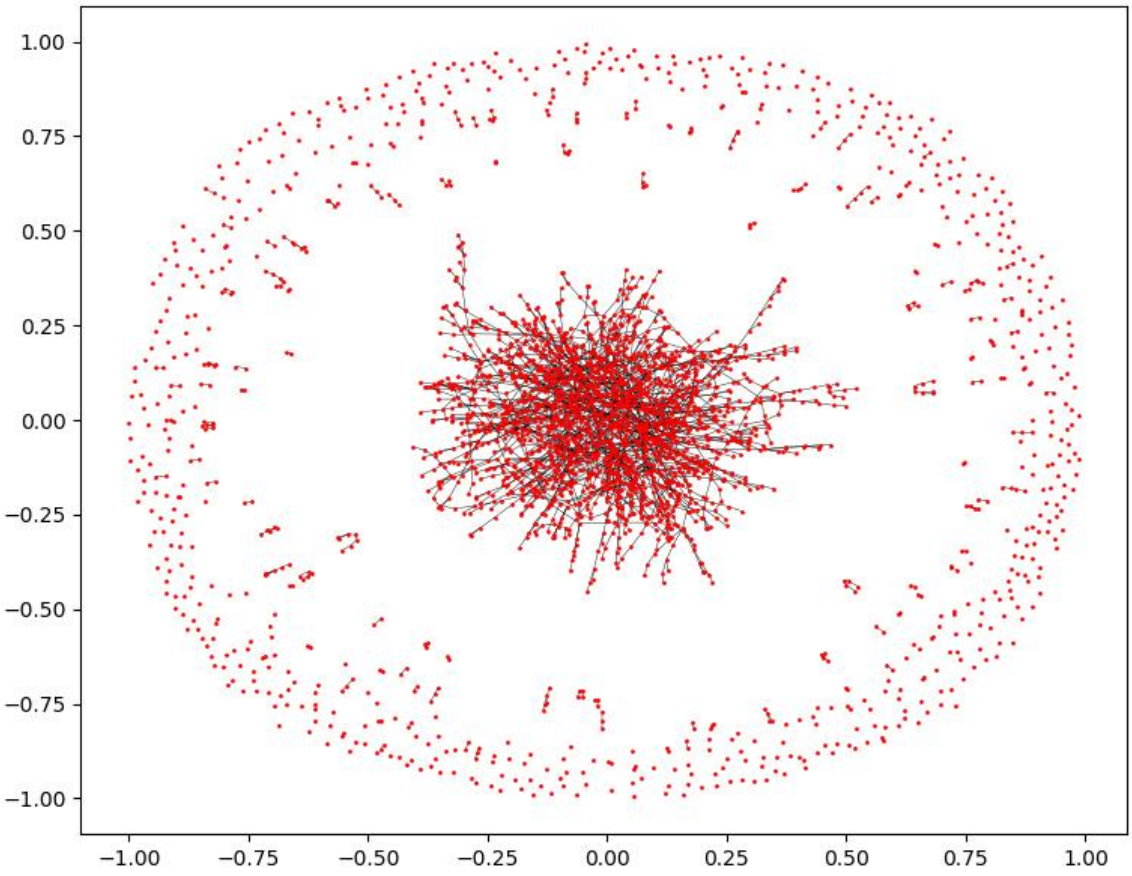

Figure 1. Visualization of Data.csv

Figure 1 visualize the nodes and the edges of the adjacency matrix given by the csv file. This paper has chosen the spring layout (node positioning using Fruchterman-Reingold force-directed algorithm) as the node positioning method to effectively show the association of each nodes. There are 555 isolated nodes at the edge of the figure with few isolated groups also. In the center of the figure, locates the main group. The following Figure2 is a closed-up version of the main nodes group visualize in Figure1.

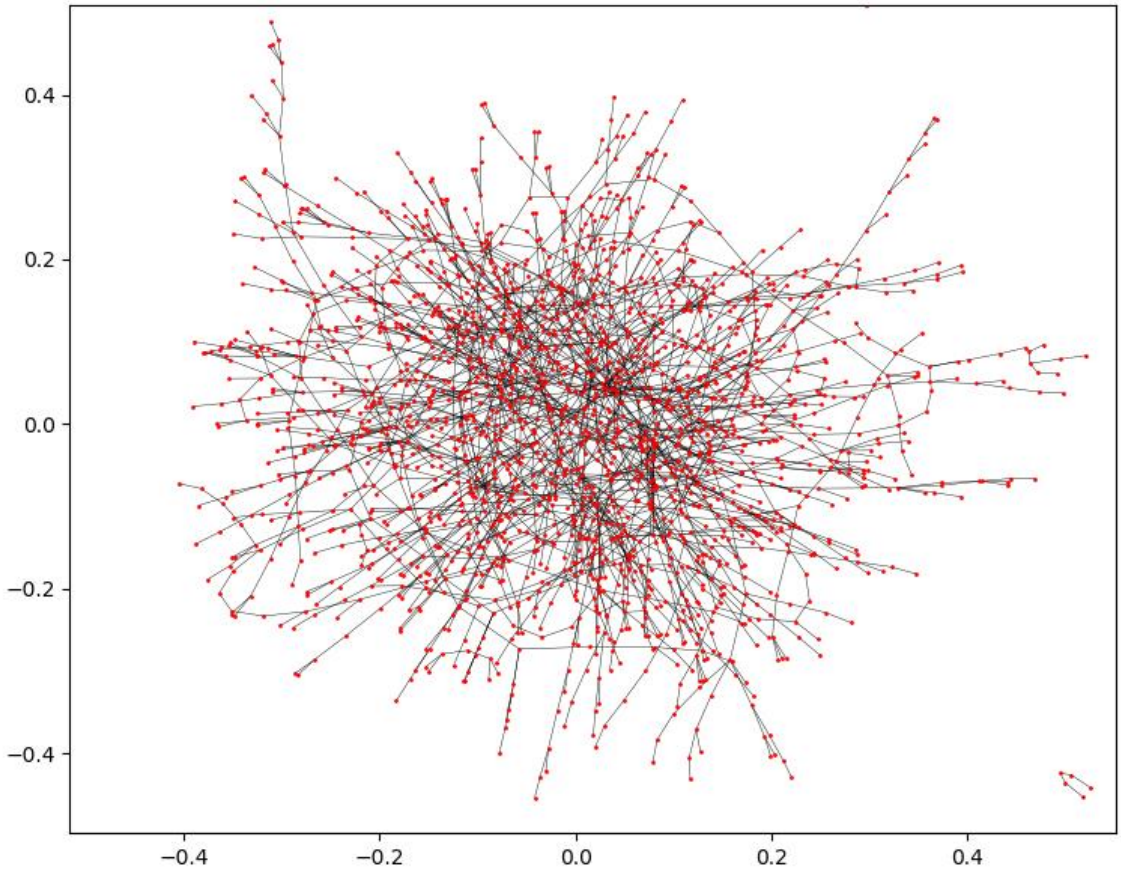

Figure 2. Zoomed version of main nodes in figure1

For the number of nodes and edges, this paper had also used function from the networkx library. The results are the following.

```
In [2]: runfile('/Users/Chokeunhee/Desktop/AI final/untitled folder/visual.py',
wdir='/Users/Chokeunhee/Desktop/AI final/untitled folder')
The number of isolated nodes are :  555
The number of nodes are :  3013
The number of edges are :  2505
```

Figure 3. number of nodes and edges

As shown from the figure, there are ***3013*** nodes and ***2505*** edges for this matrix.

ii) Use a search algorithm and find five alternative paths from node number 27 (“Start”) to node number 1445 (“Destination”) on the graph. Write the paths found as the corresponding node number sequences.

In order to find paths from node number 27 to node number 1445, This paper have used the source code from article from GeeksforGeeks[1]. From the article ‘Print all paths from a given source to a destination’. In order to implement this source code, it is necessary to initialize the given csv data into a graph format which fits the provided code. The following is the initializing process.

```
1
2   import numpy as np
3   import networkx as nx
4   from collections import defaultdict
5
6   class Graph:
7       def __init__ (self,vertices):
8           self.V = vertices
9           self.graph = defaultdict(list)
10      def addEdge(self,u,v):
11          self.graph[u].append(v)
12
13  file = open("Data.csv")
14  M = np.loadtxt(file,delimiter=',')
15  H = nx.from_numpy_matrix(M)
16  Edgelist =H.edges()
17  X = [x for x, y in Edgelist]
18  Y = [y for x, y in Edgelist]
19  g = Graph(3013)
20  for i in range (len(X)):
21      g.addEdge(X[i]+1,Y[i]+1)
22      g.addEdge(Y[i]+1,X[i]+1)
23  graph = g.graph
```

Code 1. Graph initializing code

Based on the result of Code1, the following are five answers to the given question, path from node 27 to node 1445.

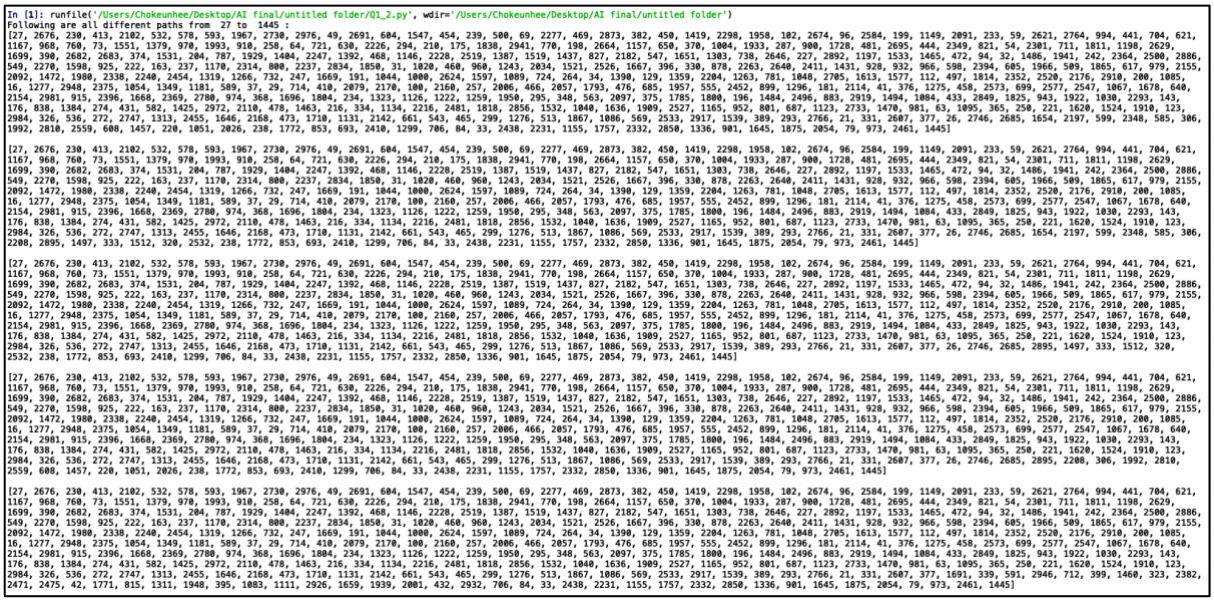

Figure 4. Output of 5 alternative path from node 27 to node 1445

iii) Assume that all the graph edges are equally weighted with the weight set at 1. Use A*, Dijkstra, or any other greedy search algorithm, and find the shortest path between the Start and Destination nodes. Write the shortest path obtained together with the shortest distance between the two nodes.

Since all the weight is set as one, it is possible to use breadth-first search as a search algorithm. This paper have used the source code provided on GeeksforGeeks[2] to solve this question. In order to implement the graph to the code, graph initializing is needed. The same process as question2 is done to convert the csv file into a fitting list of data (Graph). Using the graph initialized the following is the output of the greedy search algorithm used.

```
In [8]: runfile('/Users/Chokeunhee/Desktop/AI final/untitled folder/Q1_3.py', wdir='/Users/Chokeunhee/
Desktop/AI final/untitled folder')
Shortest path =  27 2676 230 413 2102 532 578 593 1967 2730 2976 49 2691 604 2992 2367 983 210 2375 2077
1496 365 2932 706 84 33 2438 2231 1155 1757 2332 2850 1336 901 1645 1875 2054 79 973 2461 1445
Distance =  40
```

Figure 5. The shortest path between node 27 and node 1445

From the output Figure5, the shortest path between the given two nodes are as following.
27 (Start) -> 2676 -> 230 -> 413 -> 2102 -> 532 -> 578 -> 593 -> 1967 -> 2730 -> 2976 -> 49 -> 2691 -> 604 -> 2992 -> 2367 -> 983 -> 210 -> 2375 -> 2077 -> 1496 -> 365 -> 2932 -> 706 -> 84 -> 33 -> 2438 -> 2231 -> 1155 -> 1757 -> 2332 -> 2850 -> 1336 -> 901 -> 1645 -> 1875 -> 2054 -> 79 -> 973 -> 2461 -> 1445 (Destination)

The distance between the two nodes is ***40***.

II. Given a function $\psi(x, y)$ defined as

$$\psi(x,y) = 20\sin\left(\frac{\pi}{2}(x-2\pi)\right) + 20\sin\left(\frac{\pi}{2}(y-2\pi)\right) + (x-2\pi)^2 + (y-2\pi)^2$$

where $x \in [0,10]$, $y \in [0,10]$,

i) find its global maximum, using an "optimization" algorithm (GA, simulated annealing, or the like).

Among "optimization" algorithms, this paper has chosen particle swarm optimization to find the global maximum of the given function. Based on the source code from week13 system biology class [3], this report has created particle swarm optimization algorithm to solve the given problem. To effectively see the process of particle swarm optimization, this paper has adjusted the number of particles to see the difference. The following are the output of with 100 particles(left) and 5 particles(right).

```
iteration: 0, 66.14712246134687
iteration: 1, 89.36186310300867
iteration: 2, 96.16888388871853
iteration: 3, 96.16888388871853
iteration: 4, 96.16888388871853
iteration: 5, 96.16888388871853
iteration: 6, 96.16888388871853
iteration: 7, 96.16888388871853
iteration: 8, 96.16888388871853
iteration: 9, 96.16888388871853
iteration: 10, 96.16888388871853
iteration: 11, 96.16888388871853
iteration: 12, 96.16888388871853
iteration: 13, 96.16888388871853
iteration: 14, 96.16888388871853
iteration: 15, 96.16888388871853
iteration: 16, 96.16888388871853
iteration: 17, 96.16888388871853
iteration: 18, 96.16888388871853
iteration: 19, 96.16888388871853
iteration: 20, 96.16888388871853
iteration: 21, 96.16888388871853
iteration: 22, 96.16888388871853
iteration: 23, 96.16888388871853
iteration: 24, 96.16888388871853
iteration: 25, 96.16888388871853
iteration: 26, 96.16888388871853
iteration: 27, 96.16888388871853
iteration: 28, 96.16888388871853
iteration: 29, 96.16888388871853
[x,y] coordinate :  [0, 0]
global Maximum 96.16888388871853
```

```
iteration: 0, 49.66904786997587
iteration: 1, 49.66904786997587
iteration: 2, 59.95415944956989
iteration: 3, 65.0658842679102
iteration: 4, 68.28127806034273
iteration: 5, 69.11895074227111
iteration: 6, 69.11895074227111
iteration: 7, 69.11895074227111
iteration: 8, 69.11895074227111
iteration: 9, 72.38147827894687
iteration: 10, 77.40363144912122
iteration: 11, 77.40363144912122
iteration: 12, 77.40363144912122
iteration: 13, 77.40363144912122
iteration: 14, 77.40363144912122
iteration: 15, 80.14677571491787
iteration: 16, 96.16888388871853
iteration: 17, 96.16888388871853
iteration: 18, 96.16888388871853
iteration: 19, 96.16888388871853
iteration: 20, 96.16888388871853
iteration: 21, 96.16888388871853
iteration: 22, 96.16888388871853
iteration: 23, 96.16888388871853
iteration: 24, 96.16888388871853
iteration: 25, 96.16888388871853
iteration: 26, 96.16888388871853
iteration: 27, 96.16888388871853
iteration: 28, 96.16888388871853
iteration: 29, 96.16888388871853
[x,y] coordinate :  [0, 0]
global Maximum 96.16888388871853
```

Figure 6. output of Particle swarm optimization global maximum of the given function. Particle size 100(left), 5(right)

Based on the output of Figure6, the global maximum of given function $\psi(x, y)$ is ***96.1689*** when at both coordinate of x and y is 0. Moreover, due to the difference of number of particles the learning rate has differ between the two sequences. Furthermore, with a small number of particles, there are few sequences which could not properly find the global maximum, rather local maximum as the result.

ii) Visualize results of (a) by plotting the function and the maximum found.

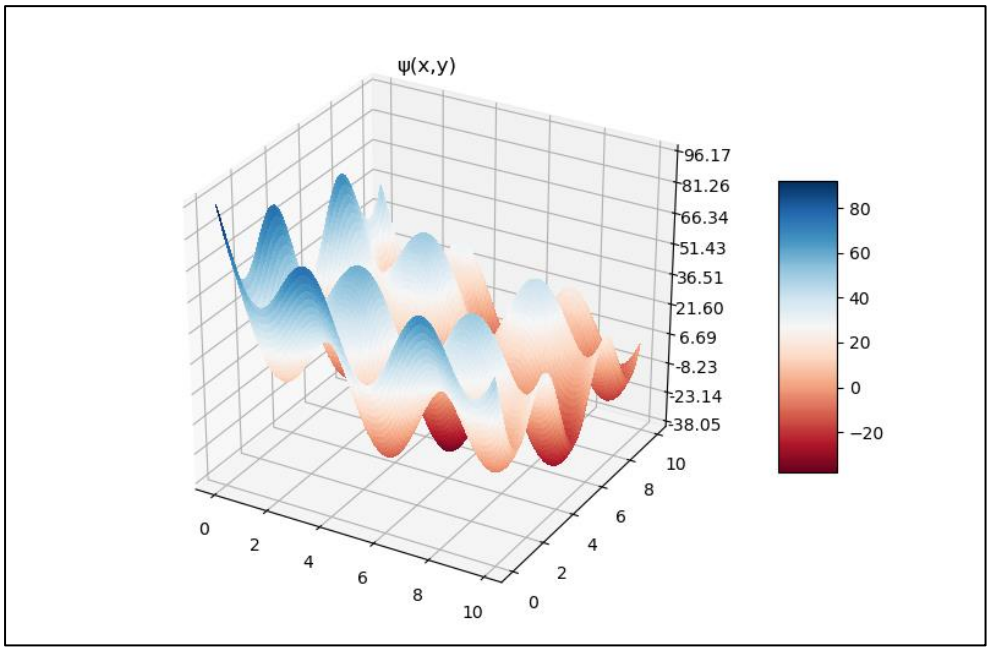

Figure 7. 3D plot of function $\psi(x, y)$

Figure7 is a 3D plot of the given function $\psi(x, y)$. As shown from the figure the global maximum is 96.17 with the coordinate of (0,0). This is the same result with the two sequences of question2.1.

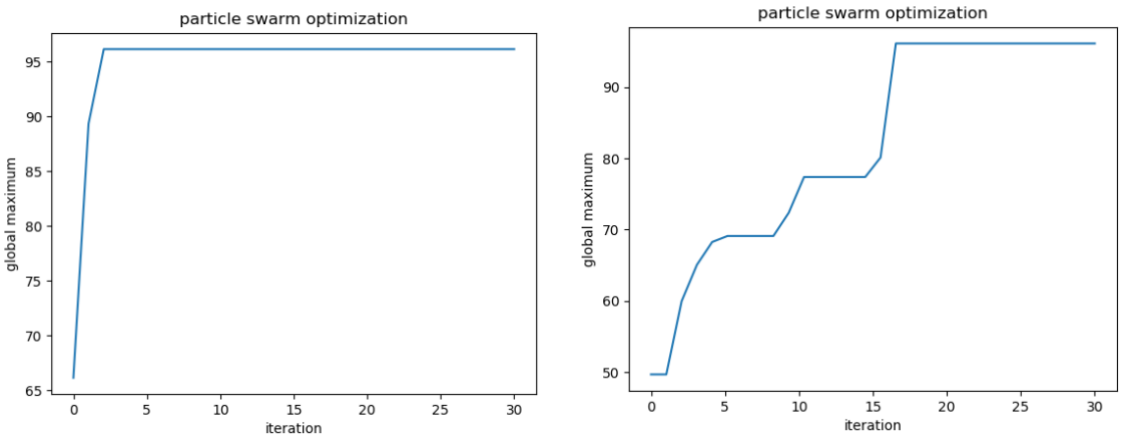

Figure 8. Graph of two sequence of Question2. 100 particles(left), 5 particle(right)

Figure is the visualization of two sequences of question2.1. The left side is the plot of the sequence with 100 particles, and the right side is the plot of the sequence with 5 particles. As shown from the graph, it is noticeable that the global maximum is 96.17. Moreover, the learning speed difference can also be observed.

Reference

[1]
https://www.geeksforgeeks.org/find-paths-given-source-destination/

[2]
https://www.geeksforgeeks.org/building-an-undirected-graph-and-finding-shortest-path-using-dictionaries-in-python/

[3]
https://github.com/Chokeunhee/Rits_SystemBio/blob/main/Week13/26001904131_Week13_Q1-F2.py

Appendix

Question 1.1

```
import numpy as np
import networkx as nx
#import matplotlib.pyplot as plt

file = open("Data.csv")
M = np.loadtxt(file,delimiter=',')
#print(M)
H = nx.from_numpy_matrix(M)
#print(H)
pos = nx.spring_layout(H)
#H.remove_nodes_from(list(nx.isolates(H)))
nx.draw_networkx_nodes(H,pos,node_size=1)
nx.draw_networkx_edges(H,pos,width=0.3)

print("The number of isolated nodes are : ", nx.number_of_isolates(H))
print("The number of nodes are : ", nx.number_of_nodes(H))
print("The number of edges are : ", nx.number_of_edges(H))

```

Question 1.2

```
import numpy as np
import networkx as nx
from collections import defaultdict


class Graph:

    def __init__(self,vertices):
        self.V = vertices
        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def printAllPathsUtil(self,u,d,visited,path):

        visited[u]=True
        path.append(u)
        if u == d:
            print(path)
        else:
            for i in self.graph[u]:
                if visited[i]==False:
                    self.printAllPathsUtil(i,d,visited,path)

        path.pop()
        visited[u]=False


    def printAllPaths(self,s,d):
        visited = [False]*(self.V)
        path = []
        self.printAllPathsUtil(s,d,visited,path)

file = open("Data.csv")
M = np.loadtxt(file,delimiter=',')
H = nx.from_numpy_matrix(M)
Edgelist =H.edges()
X = [x for x, y in Edgelist]
Y = [y for x, y in Edgelist]
g = Graph(3013)
for i in range (len(X)):
    g.addEdge(X[i]+1,Y[i]+1)
    g.addEdge(Y[i]+1,X[i]+1)
s = 27 ; d = 1445
print ("Following are all different paths from % d to % d :" %(s, d))


g.printAllPaths(s, d)

```

Question 1.3

```
import numpy as np
import networkx as nx
from collections import defaultdict

class Graph_build:
    def __init__ (self,vertices):
        self.V = vertices
        self.graph = defaultdict(list)
    def addEdge(self,u,v):
        self.graph[u].append(v)

def BFS_SP(graph, start, goal):
    explored = []
    queue = [[start]]

    if start == goal:
        print("Same Node")
        return

    while queue:
        path = queue.pop(0)
        node = path[-1]

        if node not in explored:
            neighbours = graph[node]

            for neighbour in neighbours:
                new_path = list(path)
                new_path.append(neighbour)
                queue.append(new_path)

                if neighbour == goal:
                    print("Shortest path = ", *new_path)
                    return
            explored.append(node)

    print("path doesn't exist :(")
    return

file = open("Data.csv")
M = np.loadtxt(file,delimiter=',')
H = nx.from_numpy_matrix(M)
Edgelist =H.edges()
X = [x for x, y in Edgelist]
Y = [y for x, y in Edgelist]
g = Graph_build(3013)
for i in range (len(X)):
    g.addEdge(X[i]+1,Y[i]+1)
    g.addEdge(Y[i]+1,X[i]+1)
graph = g.graph
BFS_SP(graph,27,1445)

```

Question 1 – Graph initializing

```
import numpy as np
import networkx as nx
from collections import defaultdict

class Graph:
    def __init__ (self,vertices):
        self.V = vertices
        self.graph = defaultdict(list)
    def addEdge(self,u,v):
        self.graph[u].append(v)

file = open("Data.csv")
M = np.loadtxt(file,delimiter=',')
H = nx.from_numpy_matrix(M)
Edgelist =H.edges()
X = [x for x, y in Edgelist]
Y = [y for x, y in Edgelist]
g = Graph(3013)
for i in range (len(X)):
    g.addEdge(X[i]+1,Y[i]+1)
    g.addEdge(Y[i]+1,X[i]+1)
graph = g.graph
```

Question 2 – Particle Swarm Optimization

```
import random
import matplotlib.pyplot as plt
import numpy as np

def objective_function(X):
    x=X[0]
    y=X[1]
    value =  20 * np.sin((np.pi/2) * (x - 2 * np.pi)) + 20 * np.sin((np.pi/2)
    * (y - 2 * np.pi)) + ((x - 2 * np.pi) ** 2) + ((y - 2 * np.pi) ** 2)
    return value

bounds = [(0,10),(0,10)]
num_var = 2
mm = 1
num_particles = 5
iterations =30
w = 0.8
c1 = 1
c2 = 2

class individualParticle:
    def __init__(self,bounds):
        self.individualposition = []
        self.individualvolocity = []
        self.individualBestPosition = []
        self.individualBestFitness = initial_fitness
        self.individualFitness = initial_fitness
        for i in range(num_var):
            self.individualposition.append(random.uniform(bounds[i][0],bounds[i][1]))
            self.individualvolocity.append(random.uniform(-1,1))

    def evaluate(self,objective_function):
        self.individualFitness = objective_function(self.individualposition)
        if self.individualFitness > self.individualBestFitness:
            self.individualBestPosition = self.individualposition
            self.individualBestFitness = self.individualFitness

    def velocity_update(self, groupBestPosition):
        for i in range(num_var):
            r1 = random.random()
            r2 = random.random()
        CognitiveVelocity = c1*r1*(self.individualBestPosition[i]-self.individualposition[i])
        SocialVelocity = c2*r2*(groupBestPosition[i]-self.individualposition[i])
        self.individualvolocity[i] = w*self.individualvolocity[i]+CognitiveVelocity+SocialVelocity

    def position_update(self,bounds):
        for i in range(num_var):
            self.individualposition[i]=self.individualposition[i]+self.individualvolocity[i]
            if self.individualposition[i]>bounds[i][1]:
                self.individualposition[i]=bounds[i][1]
            if self.individualposition[i]<bounds[i][0]:
                self.individualposition[i]=bounds[i][0]
```

```
initial_fitness = -float("inf")
groupBestFitness = initial_fitness
groupBestPosition = []
swarm = []
for i in range(num_particles):
    swarm.append(individualParticle(bounds))

A = []

for i in range(iterations):
    for j in range(num_particles):
        swarm[j].evaluate(objective_function)

        if swarm[j].individualFitness > groupBestFitness:
            groupBestPosition = list(swarm[j].individualposition)
            groupBestFitness = float(swarm[j].individualFitness)

    for j in range(num_particles):
        swarm[j].velocity_update(groupBestPosition)
        swarm[j].position_update(bounds)

    print('iteration: {}, {}'.format(i,groupBestFitness))
    A.append(groupBestFitness)

print("[x,y] coordinate : ",groupBestPosition)
print("global Maximum",groupBestFitness)

X = np.linspace(0,iterations,iterations)
plt.title("particle swarm optimization")
plt.xlabel("iteration")
plt.ylabel("global maximum")
plt.plot(X,A)
plt.show()
```

Question2 – 3D Graph plot

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import numpy as np
from pylab import meshgrid,title

def function(x,y):
    return  20 * np.sin((np.pi/2) * (x - 2 * np.pi)) + 20 * np.sin((np.pi/2)
                    * (y - 2 * np.pi)) + ((x - 2 * np.pi) ** 2) + ((y - 2 * np.pi) ** 2)

x = np.arange(0,10,0.1)
y = np.arange(0,10,0.1)
X,Y = meshgrid(x, y)
Z = function(X,Y)

fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,cmap=cm.RdBu,linewidth=0, antialiased=False)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
fig.colorbar(surf, shrink=0.5, aspect=5)
title("\u03C8(x,y)")
plt.show()


```