```
import sys
import os
import time
import random
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import math
import random as rn
from numpy.random import choice as np_choice


global blacklist
blacklist = []

global temporary_distance_first_to_origin

global total_distance_ACO


global iteration_measure









class wireless_sensor_networks:


    def __init__(self):
```

```
        file_data = open("inputs.txt", "a")



        print("""PRESS
1 TO CONTINUE
0 TO EXIT AND SAVE :
                """)

        file_data.writelines("""PRESS
1 TO CONTINUE
0 TO EXIT AND SAVE:\n \n""")  #saving inputs in text file




        stc=int(input())
        file_data.writelines(">>> "+ str(stc)+"\n \n")
        if (stc==1):
            self.user_input(file_data)  #updates of the time 10th august 2019
        else:
            file_data.close()

            exit(0)







    def user_input(self,file_data):
        print("ENTER THE DIMENSION OF THE FIELD AS (Length Breadth [as Int]) : ")
        file_data.writelines("ENTER THE DIMENSION OF THE FIELD AS (Length Breadth [as Int]) :\n \n")  #saving file in text file
        length,breadth=list(map(int,input().split(" ")))
        file_data.writelines(">>> " + str(length) +"  "+str(breadth)+ "\n \n")
        print("ENTER THE NUMBER OF NODES THAT IS NEEDED TO BE DEPLOYED [as INT] :")
        file_data.writelines("ENTER THE NUMBER OF NODES THAT IS NEEDED TO BE DEPLOYED [as INT] :\n \n")   #saving file in text file
```

```
        nodes=int(input())
        print(length,breadth,nodes)
        file_data.writelines(">>> " + str(nodes) + "\n \n")

        #file_data.close()



        self.cluster_creation(length,breadth,nodes,file_data)









    def cluster_creation(self,length,breadth,nodes,file_data):

        print("""ENTER TYPE OF CLUSTER CREATION:
                 1. AREA WISE CLUSTER CREATION
                 2. DEFAULT (NOT KNOWS AS OF NOW) :""")

        file_data.writelines("""ENTER TYPE OF CLUSTER CREATION:
                 1. AREA WISE CLUSTER CREATION
                 2. DEFAULT (NOT KNOWS AS OF NOW) : \n""")  #saving file in text editor

        cluster_option=int(input())
        file_data.writelines(">>> " + str(cluster_option) + "\n \n")


        if cluster_option==1:
            self.cluster_area_wise(length,breadth,nodes,file_data)
        elif cluster_option==2:
            pass

        else:
            pass
```

```
            pass


    def cluster_area_wise(self,length,breadth,nodes,file_data):

        print("ENTER THE NUMBER OF CLUSTERS YOU WANT TO CREATE :")

        file_data.writelines("ENTER THE NUMBER OF CLUSTERS YOU WANT TO CREATE : \n ")
        number_of_clusters=int(input()) #can be changed to user input later on

        file_data.writelines(">>> " + str(number_of_clusters) + "\n \n")

        x=length//(number_of_clusters**0.5)
        y=breadth//(number_of_clusters**0.5)

        cluster_list=[]
        cluster_count=1

        x=int(x)
        y=int(y)
        #print(length,breadth,x,y)
        #file_data.writelines(">>> " + str(length)+" " + str(breadth)+" "  + str(x)+" "  +str(y)+ "\n ")

        #THE CLUSTERING ASSIGNMENT CAUSES ERROR DURING UNEVEN LENGTH AND BREADTH ASSIGNMENT

        for x_axis in range(0,length,x):
            for y_axis in range(0,breadth,y):
                cluster_number="CLUSTER "+str(cluster_count)
                cluster_count+=1
                cluster_list.append([cluster_number,[],[[x_axis,x_axis+x],[y_axis,y_axis+y]]])


        #print(cluster_list)

        self.deployment(length,breadth,nodes,cluster_list,file_data)
```

```
def sergation_of_nodes_in_cluster(self,length,breadth,nodes,cluster_list,particles,file_data):




    for each_partilcle in particles:
        for node_cluster in cluster_list:
            if ((each_partilcle[1][0]>=node_cluster[2][0][0] and each_partilcle[1][0]<node_cluster[2][0][1]) and (each_partilcle[
                node_cluster[1].append(each_partilcle)
                break




    #print(cluster_list)

    for each in cluster_list:
        print(each[0],each[1],len(each[1])/nodes)
        file_data.writelines(">>> " + str(each[0]) +str(each[1])+str(len(each[1])/nodes)+ "\n ")

    print(cluster_list)
    file_data.writelines(">>> " + str(cluster_list)  + "\n ")




    self.visualization(length,breadth,cluster_list,particles,file_data)







def deployment(self,length,breadth,nodes,cluster_list,file_data):
    print("""CHOOSE A DEPLOYMENT TYPE AS OPTIONS -
             1. RANDOM DEPLOYMENT
```

```
                2. SPIRAL DEPPLYMENT
                3. SQUARE DEPLYMENT
                4. DEFAULT DEPLOYMENT""")

        file_data.writelines("""CHOOSE A DEPLOYMENT TYPE AS OPTIONS -
                1. RANDOM DEPLOYMENT
                2. SPIRAL DEPPLYMENT
                3. SQUARE DEPLYMENT
                4. DEFAULT DEPLOYMENT \n
                """)   #sving file in text file
        deployment_option=int(input())

        print(deployment_option)
        file_data.writelines(">>> " + str(deployment_option) + "\n ")

        if deployment_option==1:
            self.random_deployment(length,breadth,nodes,cluster_list,file_data)
        elif deployment_option==2:
            self.spiral_deployment(length,breadth,nodes,cluster_list,file_data)
        elif deployment_option==3:
            self.square_deployment(length,breadth,nodes,cluster_list,file_data)
        else:
            self.default_deployment(length,breadth,nodes,cluster_list,file_data)


    def random_deployment(self,length,breadth,nodes,cluster_list,file_data):
        no_of_particles = nodes
        particles = []

        print("""ENTER THE TIME INTERVAL IN UNITS""")

        file_data.writelines("""ENTER THE TIME INTERVAL IN UNITS \n""")
        time_interval = int(input())
        time_interval = time_interval / nodes

        file_data.writelines(">>> " + str(time_interval)  + "\n ")
```

```
        time = 0

        for i in range(no_of_particles):
            id = i+1  # creates a id of the particles
            x_cordinate = round(random.randrange(0,length),2)+round(random.random(),2)# creates the x cordinate of the particle in th
            y_cordinate = round(random.randrange(0,breadth),2)+round(random.random(),2)  # creates the y cordinate of the particles i
            # print(x_cordinate,y_cordinate)
            battery=0  #change it later on
            particles.append([id,[x_cordinate,y_cordinate],[time],[battery]])
            time+=time_interval

        print(particles)
        file_data.writelines(">>> " + str(particles) + "\n ")

        self.sergation_of_nodes_in_cluster(length,breadth,nodes,cluster_list,particles,file_data)



    def spiral_deployment(self,length,breadth,nodes,cluster_list,file_data):

        listx = []
        particles=[]

        def spiralPrint(m, n):


            k = 0
            l = 0

            ''' k - starting row index
                m - ending row index
                l - starting column index
                n - ending column index
                i - iterator '''

            while (k < m and l < n):
```

```
                # Print the first row from
                # the remaining rows
                for i in range(l, n):
                    particles.append([k, i])
                    #print("({},{})".format(k + 0.5, i + 0.5), end=" ")

                k += 1

                # Print the last column from
                # the remaining columns
                for i in range(k, m):
                    particles.append([i, n - 1])
                    #print("({},{})".format(i + 0.5, n - 1 + 0.5), end=" ")

                n -= 1

                # Print the last row from
                # the remaining rows
                if (k < m):

                    for i in range(n - 1, (l - 1), -1):
                        #print("({},{})".format(m - 1 + 0.5, i + 0.5), end=" ")
                        particles.append([m - 1, i])

                    m -= 1

                # Print the first column from
                # the remaining columns
                if (l < n):
                    for i in range(m - 1, k - 1, -1):
                        #print("({},{})".format(i + 0.5, l + 0.5), end=" ")
                        particles.append([i, l])

                    l += 1



        spiralPrint(length, breadth)
```

```
        listx=particles

        print("""ENTER THE TIME INTERVAL IN UNITS""")

        file_data.writelines("""ENTER THE TIME INTERVAL IN UNITS \n""") #saving flie in txt file
        time_interval=int(input())
        time_interval=time_interval/nodes
        node_interval=len(listx)//nodes

        time=0
        id=1
        particles=[]
        battery=0
        for i in range(0,len(listx),node_interval):
            particles.append([id,[listx[i][0],listx[i][1]],[battery],time])
            time+=time_interval
            id+=1

        print(particles)
        file_data.writelines(str(particles)+ " \n")


        self.sergation_of_nodes_in_cluster(length, breadth, nodes, cluster_list,particles, file_data)

    def square_deployment(self,length,breadth,nodes,cluster_list,file_data):

        print("""ENTER THE TIME INTERVAL IN UNITS""")

        file_data.writelines("""ENTER THE TIME INTERVAL IN UNITS \n """) #saving file in txt file
        time_interval = int(input())

        file_data.writelines(">>> " + str(time_interval)  + "\n ")
        time_interval=time_interval/nodes
        no_of_particles = nodes
        particles = []
        timex=0

        breadth_sqr=breadth
```

```
    no_of_clusters=len(cluster_list)

    breadth_start=int(breadth_sqr-((breadth_sqr//(no_of_clusters**0.5))//2))
    breadth_incx=int(breadth_sqr//(no_of_clusters**0.5))
    id=0
    flx=0

    while(breadth_start>0):

        y_cordinate=breadth_start
        flx+=1

        temp=[]

        battery=0 #change it later on



        for i in range(0, int(no_of_particles // (no_of_clusters ** 0.5))):
            temp.append(random.randrange(0, length))
        if (flx % 2 != 0):
            temp = sorted(temp)
        else:
            temp = sorted(temp, reverse=True)



        for x_cordinate in temp:

            id += 1
            timex+=time_interval
            particles.append([id, [x_cordinate, y_cordinate], [battery], timex])


        breadth_start = breadth_start - breadth_incx
```

```
        print(particles)
        file_data.writelines(">>> " + str(particles) + "\n ")

        self.sergation_of_nodes_in_cluster(length, breadth, nodes, cluster_list, particles,file_data)

    def default_deployment(self,length,breadth,nodes,cluster_list):
        pass









    def visualization(self,length,breadth,cluster_list,particles,file_data):


        G=nx.Graph()

        for each in particles:
            G.add_node(each[0],pos=(each[1][0],each[1][1]))


        #pos = nx.get_node_attributes(G, 'pos')

        pos = {}

        for each in particles:
            pos[each[0]] = (each[1][0], each[1][1])


        #nx.draw_networkx_labels(G, pos)
        nx.draw_networkx(G,pos=pos)


        plt.title("CLUSTERING NETWORK'S")
        plt.xlabel('X-AXIS')
        plt.ylabel('Y-AXIS')
```

```
        # Limits for the Y  and Yaxis
        incx=(len(cluster_list)**0.5)
        #plt.ylim(0, breadth)
        #plt.xlim(0, length)
        plt.xticks(np.arange(0, length+1, length//incx))
        plt.yticks(np.arange(0, breadth+1, breadth//incx))
        #plt.plot([lambda x: x[1][0] for x in particles],[lambda y: y[1][1] for y in particles[1][1]],'ro')
        plt.grid()
        plt.savefig("WSN_labels.png")
        plt.show()

        for each in cluster_list:
            for i in range(0, len(each[1])-1):
                for j in range(i + 1, len((each[1]))):
                    G.add_edge(each[1][i][0], each[1][j][0])

        #nx.draw(G,pos, with_labels=True)
        nx.draw_networkx(G, pos=pos,with_labels=True)

        plt.xticks(np.arange(0, length + 1, length // incx))
        plt.yticks(np.arange(0, breadth + 1, breadth // incx))
        plt.grid()
        plt.savefig("WSN_CLUSTER.png")
        plt.show()


        print("#################################")
        print(cluster_list)

        self.analyze(length,breadth,cluster_list,particles, file_data)



    def analyze(self,length,breadth,cluster_list,particles,file_data):

        main_flag = True

        for each in cluster list:
```

```
            #print(len(each[1]))
            lengthx = len(each[1])
            if int(lengthx) == int(0):
                main_flag = False

        print("##########")
        print(main_flag)

        global iteration_measure
        iteration_measure=0

        global total_distance_ACO
        total_distance_ACO=0
        temp_ACO = 0


        while (main_flag != False):



            iteration_measure+=1

            leader_node = []

            for cluster in cluster_list:
                min_distance = math.inf
                for cluster_value_x in cluster[1]:
                    total_distance = 0
                    for cluster_value_y in cluster[1]:
                        if cluster_value_x[0] != cluster_value_y[0]:
                            total_distance += (((cluster_value_x[1][0] - cluster_value_y[1][0]) ** 2) + (
                                        (cluster_value_x[1][1] - cluster_value_y[1][1]) ** 2) ** 0.5)
                    if total_distance < min_distance:
                        min_distance = total_distance
                        id = cluster_value_x[0]

                leader_coordinates = []
```

```
            for cluster_value_x in cluster[1]:
                if id == cluster_value_x[0]:
                    leader_node.append([cluster[0], cluster_value_x])

            for cluster_value_x in leader_node:
                leader_coordinates.append(
                    [cluster_value_x[1][0], cluster_value_x[1][1][0], cluster_value_x[1][1][0]])

        file_data.writelines("THE LEADER NODES IN RESPECTIVE CLUSTERS\n")
        for row in leader_node:
            #print(row)
            file_data.writelines(">>>" + str(row) + "\n")

        print(leader_node)

        #for row in leader_node:
            #print("#$$$$$$$$$$$$$$$$$$$$$$$$")
            #print(row[1][1][0],row[1][1][1])


        global temporary_distance_first_to_origin
        temporary_distance_first_to_origin=(((leader_node[0][1][1][0]**2)+(leader_node[0][1][1][1]**2))**0.5)

        print(temporary_distance_first_to_origin)

        xxx = -1
        yyy = -1
        zzz = 0



        for each in cluster_list:
            xxx += 1
            for row in each[1]:
                yyy += 1
                if leader_coordinates[xxx][0] == row[0]:
                    #print(leader_coordinates[xxx][0], row[0])
```

```
                    #print(cluster_list[xxx][1][yyy])
                    blacklist.append(cluster_list[xxx][1][yyy][0])
                    del cluster_list[xxx][1][yyy]
                    break
        yyy = -1

    """for each in cluster_list:
        xxx+=1
        #print(cluster_list[xxx])
        for row in each[1]:
            yyy+=1
            print(cluster_list[xxx][1][yyy])
        yyy=-1"""

    #for each in cluster_list:
        #print(each)

    #for each in cluster_list:
        #print(len(each[1]))


    """leader_node=[]

    for cluster in cluster_list:
        min_distance=math.inf
        for cluster_value_x in cluster[1]:
            total_distance=0
            for cluster_value_y in cluster[1]:
                if cluster_value_x[0]!=cluster_value_y[0]:
                    total_distance+=(((cluster_value_x[1][0]-cluster_value_y[1][0])**2)+((cluster_value_x[1][1]-cluster_value
            if total_distance<min_distance:
                min_distance=total_distance
                id =cluster_value_x[0]

        leader_coordinates=[]

        for cluster_value_x in cluster[1]:
            if id==cluster_value_x[0]:
```

```
                    leader_node.append([cluster[0],cluster_value_x])

            for cluster_value_x in leader_node:
                leader_coordinates.append([cluster_value_x[1][0],cluster_value_x[1][1][0],cluster_value_x[1][1][0]])

        file_data.writelines("THE LEADER NODES IN RESPECTIVE CLUSTERS\n")
        for row in leader_node:
            print(row)
            file_data.writelines(">>>"+str(row)+"\n")

        print(leader_node)


        for row in leader_coordinates:
            print(row)"""

        for each in cluster_list:
            # print(len(each[1]))
            lengthx = len(each[1])
            if int(lengthx) == int(0):
                main_flag = False

        print("##########")
        print(main_flag)

        if main_flag==False:
            break



        self.visualize_leader(length,breadth,cluster_list,particles,leader_node,file_data,total_distance_ACO)
        ACO=self.ACO_fine_tuning(leader_coordinates,leader_node,file_data,breadth,length,cluster_list,particles,total_distance_AC
        temp_ACO+=ACO

    print("AFTER ALL ITERATION TOTAL DISTANCE IS : {} ".format(temp_ACO))
```

```
def visualize_leader(self,length,breadth,cluster_list,particles,leader_node,file_data,total_distance_ACO):



    G = nx.Graph()

    for each in particles:
        G.add_node(each[0], pos=(each[1][0], each[1][1]))

    # pos = nx.get_node_attributes(G, 'pos')

    pos = {}

    for each in particles:
        pos[each[0]] = (each[1][0], each[1][1])

    """# nx.draw_networkx_labels(G, pos)
    nx.draw_networkx(G, pos=pos)

    #H = nx.Graph()

    for each in leader_node:
        G.add_node(each[1][0],pos=(each[1][1][0],each[1][1][1]))



    # pos = nx.get_node_attributes(G, 'pos')

    pos = {}

    for each in leader_node:
        pos[each[1][0]] = (each[1][1][0], each[1][1][1])"""

    simplified_cluster_number = []

    for j in cluster_list:
```

```
            flag = True
            temp = []
            for k in range(1, len(j)):
                if flag == True:
                    for h in range(0, len(j[k])):
                        temp.append(j[k][h][0])
                    flag = False
                break
            if len(temp)>0:
                simplified_cluster_number.append(temp)

        #print(simplified_cluster_number)

        for i in range(0,len(leader_node)):
            #for j in simplified_cluster_number:
            for k in range(0,len(simplified_cluster_number[i])):
                #print(leader_node[i][1][0],simplified_cluster_number[i][k])
                if leader_node[i][1][0]!=simplified_cluster_number[i][k]:
                    G.add_edge(leader_node[i][1][0],simplified_cluster_number[i][k])







        # nx.draw_networkx_labels(G, pos)
        nx.draw_networkx(G, pos=pos,with_labels=True)

        H=nx.Graph()

        for each in leader_node:
            #print(each[1][0], each[1][1][0], each[1][1][1])
            H.add_node(each[1][0], pos=(each[1][1][0], each[1][1][1]))

        pox = {}

        for each in leader_node:
            pox[each[1][0]] = (each[1][1][0], each[1][1][1])
```

```
        nx.draw_networkx(H, pox, node_size=700, node_color='green')

        plt.title("CLUSTERING NETWORK'S")
        plt.xlabel('X-AXIS')
        plt.ylabel('Y-AXIS')
        # Limits for the Y  and Yaxis
        incx = (len(cluster_list) ** 0.5)
        # plt.ylim(0, breadth)
        # plt.xlim(0, length)
        plt.xticks(np.arange(0, length + 1, length // incx))
        plt.yticks(np.arange(0, breadth + 1, breadth // incx))
        # plt.plot([lambda x: x[1][0] for x in particles],[lambda y: y[1][1] for y in particles[1][1]],'ro')
        plt.grid()
        plt.savefig("WSN_labels2.png")
        plt.show()




    def ACO_fine_tuning(self,leader_coordinates,leader_node,file_data,breadth,length,cluster_list,particles,total_distance_ACO):
        #print(leader_coordinates)

        data_ACO=[]

        for i in range(0,len(leader_coordinates)):
            temp=[]
            for j in range(0,len(leader_coordinates)):
                if leader_coordinates[i][0]==leader_coordinates[j][0]:
                    temp.append(np.inf)
                else:
                    distance=(((leader_coordinates[i][1]-leader_coordinates[j][1])**2)+((leader_coordinates[i][2]-leader_coordinates[
                    temp.append(distance)
            data_ACO.append(temp)

        """for each in data_ACO:
            print(each)"""
```

```
class AntColony(object):

    def __init__(self, distances, n_ants, n_best, n_iterations, decay, alpha=1, beta=1):
        """
        Args:
            distances (2D numpy.array): Square matrix of distances. Diagonal is assumed to be np.inf.
            n_ants (int): Number of ants running per iteration
            n_best (int): Number of best ants who deposit pheromone
            n_iteration (int): Number of iterations
            decay (float): Rate it which pheromone decays. The pheromone value is multiplied by decay, so 0.95 will lead to c
            alpha (int or float): exponenet on pheromone, higher alpha gives pheromone more weight. Default=1
            beta (int or float): exponent on distance, higher beta give distance more weight. Default=1
        Example:
            ant_colony = AntColony(german_distances, 100, 20, 2000, 0.95, alpha=1, beta=2)
        """
        self.distances = distances
        self.pheromone = np.ones(self.distances.shape) / len(distances)
        self.all_inds = range(len(distances))
        self.n_ants = n_ants
        self.n_best = n_best
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta

    def run(self):
        shortest_path = None
        all_time_shortest_path = ("placeholder", np.inf)
        for i in range(self.n_iterations):
            all_paths = self.gen_all_paths()
            self.spread_pheronome(all_paths, self.n_best, shortest_path=shortest_path)
            shortest_path = min(all_paths, key=lambda x: x[1])
            # print (shortest_path)
            if shortest_path[1] < all_time_shortest_path[1]:
                all_time_shortest_path = shortest_path
            self.pheromone * self.decay
        return all_time_shortest_path
```

```
        def spread_pheronome(self, all_paths, n_best, shortest_path):
            sorted_paths = sorted(all_paths, key=lambda x: x[1])
            for path, dist in sorted_paths[:n_best]:
                for move in path:
                    self.pheromone[move] += 1.0 / self.distances[move]

        def gen_path_dist(self, path):
            total_dist = 0
            for ele in path:
                total_dist += self.distances[ele]
            return total_dist

        def gen_all_paths(self):
            all_paths = []
            for i in range(self.n_ants):
                path = self.gen_path(0)
                all_paths.append((path, self.gen_path_dist(path)))
            return all_paths

        def gen_path(self, start):
            path = []
            visited = set()
            visited.add(start)
            prev = start
            for i in range(len(self.distances) - 1):
                move = self.pick_move(self.pheromone[prev], self.distances[prev], visited)
                path.append((prev, move))
                prev = move
                visited.add(move)
            path.append((prev, start))  # going back to where we started
            return path

        def pick_move(self, pheromone, dist, visited):
            pheromone = np.copy(pheromone)
            pheromone[list(visited)] = 0

            row = pheromone ** self.alpha * ((1.0 / dist) ** self.beta)
```

```
            norm_row = row / row.sum()
            move = np_choice(self.all_inds, 1, p=norm_row)[0]
            return move

    distances = np.array(data_ACO)

    ant_colony = AntColony(distances, 1, 1, 100, 0.95, alpha=1, beta=1)
    shortest_path = ant_colony.run()
    print("shorted_path: {}".format(shortest_path))
    file_data.writelines(">>>" + str("shortest_path: ") +str(shortest_path)+ "\n")


    print(iteration_measure,(shortest_path[1]+temporary_distance_first_to_origin))
    print(temporary_distance_first_to_origin)

    if iteration_measure==1:
        total_distance_ACO=0


    print(total_distance_ACO)
    total_distance_ACO+=shortest_path[1]+temporary_distance_first_to_origin



    print("TOTAL DISTANCE AFTER ITERATION {} is {} ".format(iteration_measure,total_distance_ACO))


    #for each in shortest_path[0]:
        #print(each[0])

    G = nx.Graph()

    for each in particles:
        G.add_node(each[0], pos=(each[1][0], each[1][1]))

    # pos = nx.get_node_attributes(G, 'pos')
```

```
        pos = {}

        for each in particles:
            pos[each[0]] = (each[1][0], each[1][1])

        """# nx.draw_networkx_labels(G, pos)
        nx.draw_networkx(G, pos=pos)

        #H = nx.Graph()

        for each in leader_node:
            G.add_node(each[1][0],pos=(each[1][1][0],each[1][1][1]))



        # pos = nx.get_node_attributes(G, 'pos')

        pos = {}

        for each in leader_node:
            pos[each[1][0]] = (each[1][1][0], each[1][1][1])"""

        simplified_cluster_number = []

        for j in cluster_list:
            flag = True
            temp = []
            for k in range(1, len(j)):
                if flag == True:
                    for h in range(0, len(j[k])):
                        temp.append(j[k][h][0])
                    flag = False
                break
            if len(temp) > 0:
                simplified_cluster_number.append(temp)

        # print(simplified_cluster_number)
```

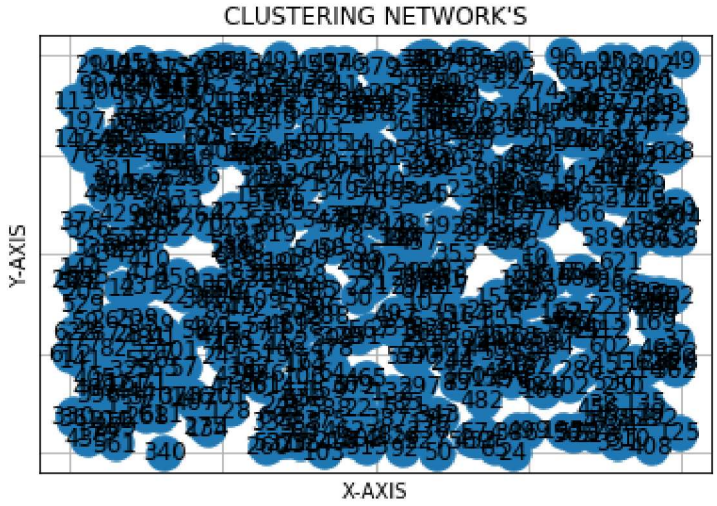
```
"""for i in range(0, len(leader_node)):
    # for j in simplified_cluster_number:
    for k in range(0, len(simplified_cluster_number[i])):
        # print(leader_node[i][1][0],simplified_cluster_number[i][k])
        if leader_node[i][1][0] != simplified_cluster_number[i][k]:
            G.add_edge(leader_node[i][1][0], simplified_cluster_number[i][k])"""

# nx.draw_networkx_labels(G, pos)
nx.draw_networkx(G, pos=pos,node_color='yellow', with_labels=True)

K = nx.Graph()

#print("######################################################################")
#print(blacklist)

pos = {}

for each in particles:
    for eachx in blacklist:
        if each[0]==eachx:
            #print(each[0],eachx)
            #print("######################################################################")
            K.add_node(each[0], pos=(each[1][0], each[1][1]))
            pos[each[0]] = (each[1][0], each[1][1])

            # pos = nx.get_node_attributes(G, 'pos')




nx.draw_networkx(K, pos=pos, node_color='grey', with_labels=True)

H = nx.Graph()

for each in leader_node:
```

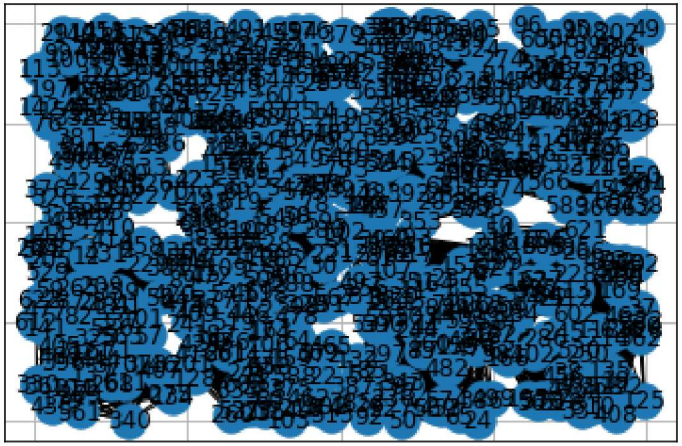
```
        # print(each[1][0], each[1][1][0], each[1][1][1])
        H.add_node(each[1][0], pos=(each[1][1][0], each[1][1][1]))

    pox = {}

    for each in leader_node:
        pox[each[1][0]] = (each[1][1][0], each[1][1][1])


    for each in shortest_path[0]:
        H.add_edge(leader_coordinates[each[0]][0],leader_coordinates[each[1]][0],color='g',weight=2)

    pos = nx.circular_layout(H)

    edges = H.edges()
    colors = [H[u][v]['color'] for u, v in edges]
    weights = [H[u][v]['weight'] for u, v in edges]




    nx.draw_networkx(H, pox, node_size=600, node_color='red',edges=edges, edge_color=colors, width=weights)

    plt.title("ACO SHORTEST PATH")
    plt.xlabel('X-AXIS')
    plt.ylabel('Y-AXIS')
    # Limits for the Y  and Yaxis
    incx = (len(cluster_list) ** 0.5)
    # plt.ylim(0, breadth)
    # plt.xlim(0, length)
    plt.xticks(np.arange(0, length + 1, length // incx))
    plt.yticks(np.arange(0, breadth + 1, breadth // incx))
    # plt.plot([lambda x: x[1][0] for x in particles],[lambda y: y[1][1] for y in particles[1][1]],'ro')
    plt.grid()
    plt.savefig("ACO.png")
    plt.show()
```

```
        return (total_distance_ACO)




    """time.sleep(4)
    file_data.close()

    self.__init__()"""









if __name__=="__main__":
    obj1=wireless_sensor_networks()
```

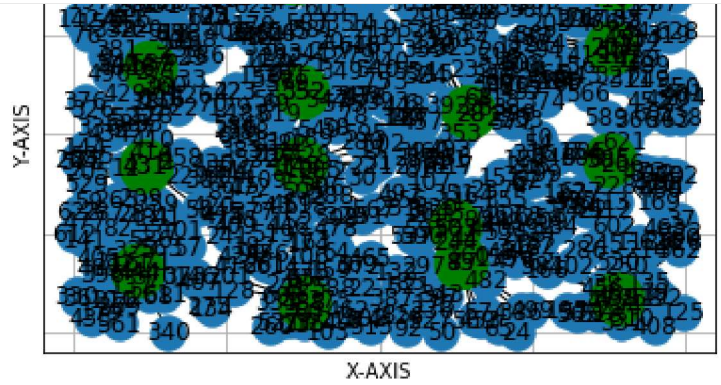
```
PRESS
1 TO CONTINUE
0 TO EXIT AND SAVE :

1
ENTER THE DIMENSION OF THE FIELD AS (Length Breadth [as Int]) :
1000 1000
ENTER THE NUMBER OF NODES THAT IS NEEDED TO BE DEPLOYED [as INT] :
634
1000 1000 634
ENTER TYPE OF CLUSTER CREATION:
                1. AREA WISE CLUSTER CREATION
                2. DEFAULT (NOT KNOWS AS OF NOW) :
1
ENTER THE NUMBER OF CLUSTERS YOU WANT TO CREATE :
16
CHOOSE A DEPLOYMENT TYPE AS OPTIONS -
                1. RANDOM DEPLOYMENT
                2. SPIRAL DEPPLYMENT
                3. SQUARE DEPLYMENT
                4. DEFAULT DEPLOYMENT
1
1
ENTER THE TIME INTERVAL IN UNITS
67
[[1, [774.93, 748.26], [0], [0]], [2, [150.48, 431.66], [0.1056782334384858], [0]], [3, [556.81, 310.73], [0.2113564668769716],
CLUSTER 1 [[36, [4.72, 86.48], [3.6987381703469997], [0]], [57, [185.79, 216.03], [5.917981072555198], [0]], [71, [136.6, 177.26
CLUSTER 2 [[2, [150.48, 431.66], [0.1056782334384858], [0]], [12, [85.4, 413.86], [1.1624605678233437], [0]], [23, [15.36, 434.3
CLUSTER 3 [[9, [51.09, 696.96], [0.8454258675078864], [0]], [27, [124.58, 661.52], [2.7476340694006294], [0]], [53, [121.64, 525
CLUSTER 4 [[25, [226.42, 801.41], [2.536277602523658], [0]], [34, [208.53, 931.96], [3.4873817034700285], [0]], [63, [156.44, 75
CLUSTER 5 [[21, [359.35, 114.77], [2.113564668769716], [0]], [22, [470.99, 114.55], [2.2192429022082014], [0]], [60, [381.69, 54
CLUSTER 6 [[15, [283.2, 368.3], [1.4794952681388012], [0]], [17, [489.52, 462.96], [1.6908517350157728], [0]], [30, [470.67, 392
CLUSTER 7 [[20, [417.1, 733.39], [2.0078864353312302], [0]], [56, [318.01, 590.26], [5.812302839116712], [0]], [58, [429.52, 518
CLUSTER 8 [[14, [474.41, 779.6], [1.3738170347003154], [0]], [26, [341.11, 753.17], [2.641955835962144], [0]], [29, [460.17, 851
CLUSTER 9 [[24, [724.08, 1.19], [2.4305993690851726], [0]], [44, [640.13, 240.52], [4.544164037854885], [0]], [46, [727.51, 196.
CLUSTER 10 [[3, [556.81, 310.73], [0.2113564668769716], [0]], [4, [728.99, 371.64], [0.3170347003154574], [0]], [11, [620.06, 32
CLUSTER 11 [[6, [551.18, 578.52], [0.528391167192429], [0]], [7, [668.1, 733.32], [0.6340694006309148], [0]], [28, [647.52, 555.
CLUSTER 12 [[16, [589.26, 898.14], [1.585173501577287], [0]], [33, [582.38, 994.99], [3.381703470031543], [0]], [40, [724.78, 84
CLUSTER 13 [[19, [951.12, 101.87], [1.9022082018927444], [0]], [38, [873.57, 120.64], [3.910094637223971], [0]], [47, [862.65, 8
CLUSTER 14 [[10, [760.89, 489.75], [0.9511041009463722], [0]], [13, [885.61, 329.27], [1.2681388012618295], [0]], [37, [992.35,
CLUSTER 15 [[1, [774.93, 748.26], [0], [0]], [5, [798.57, 655.11], [0.4227129337539432], [0]], [66, [806.7, 627.82], [6.86908517
CLUSTER 16 [[8, [914.83, 765.75], [0.7397476340694006], [0]], [18, [977.86, 883.54], [1.7965299684542586], [0]], [32, [902.27, 7
```

```
[['CLUSTER 1', [[36, [4.72, 86.48], [3.6987381703469997], [0]], [57, [185.79, 216.03], [5.917981072555198], [0]], [71, [136.6, 1
```

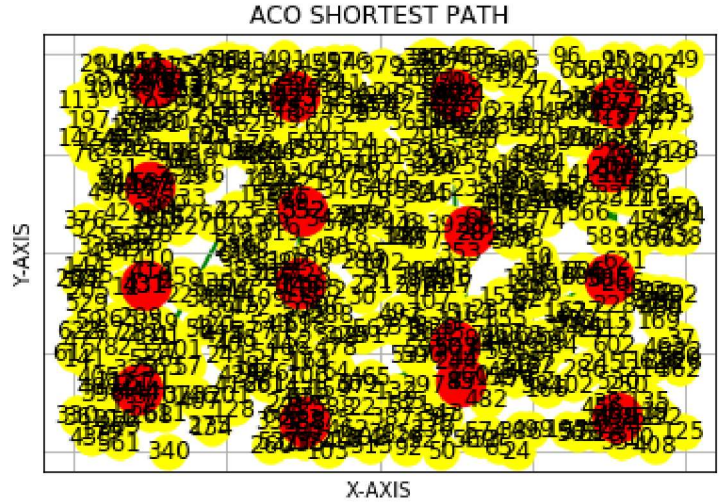




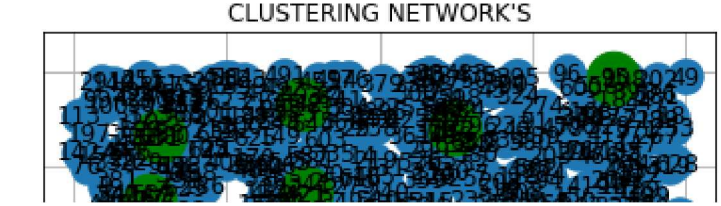
```
##################################
[['CLUSTER 1', [[36, [4.72, 86.48], [3.6987381703469997], [0]], [57, [185.79, 216.03], [5.917981072555198], [0]], [71, [136.6, 1
##########
True
[['CLUSTER 1', [444, [105.11, 155.21], [46.815457413249625], [0]]], ['CLUSTER 2', [431, [117.28, 421.38], [45.441640378549266],
187.4520103919934
##########
True
```

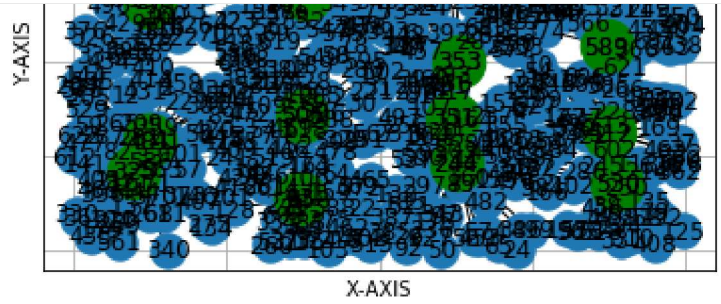



```
shorted_path: ([(0, 1), (1, 2), (2, 3), (3, 11), (11, 9), (9, 8), (8, 10), (10, 15), (15, 12), (12, 14), (14, 13), (13, 4), (4,
1 2400.583098420898
187.4520103919934
0
TOTAL DISTANCE AFTER ITERATION 1 is 2400.583098420898
```

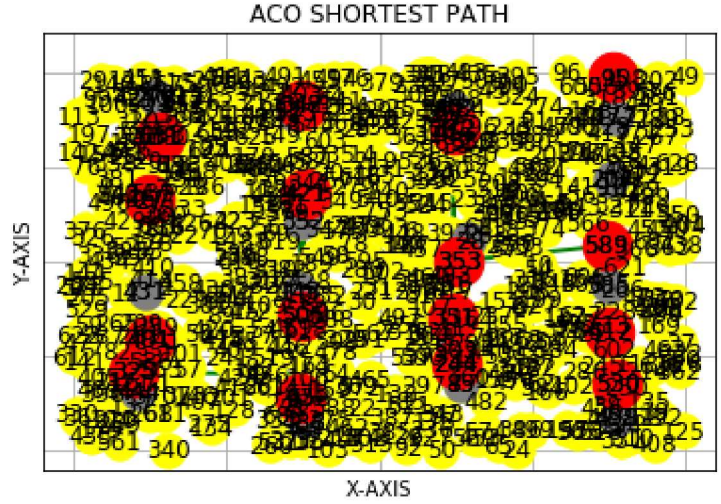



```
[['CLUSTER 1', [325, [97.89, 208.6], [34.23974763406942], [0]]], ['CLUSTER 2', [481, [125.24, 291.41], [50.72555205047372], [0]]
230.42658722465166
##########
True
```

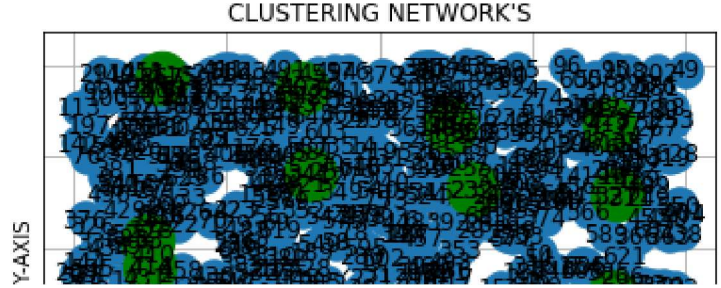


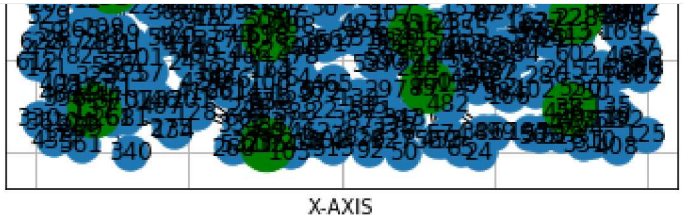



```
shorted_path: ([(0, 1), (1, 2), (2, 3), (3, 4), (4, 7), (7, 5), (5, 6), (6, 9), (9, 11), (11, 8), (8, 10), (10, 14), (14, 13), (
2 2469.748910428705
230.42658722465166
0
TOTAL DISTANCE AFTER ITERATION 2 is 2469.748910428705
```

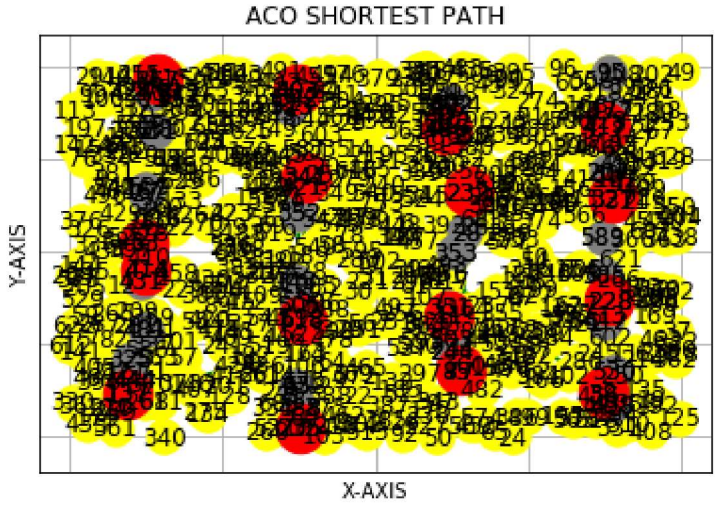



```
[['CLUSTER 1', [137, [95.96, 116.86], [14.372239747634046], [0]]], ['CLUSTER 2', [414, [125.14, 455.23], [43.64511041009495], [0
151.21038720934484
##########
True
```

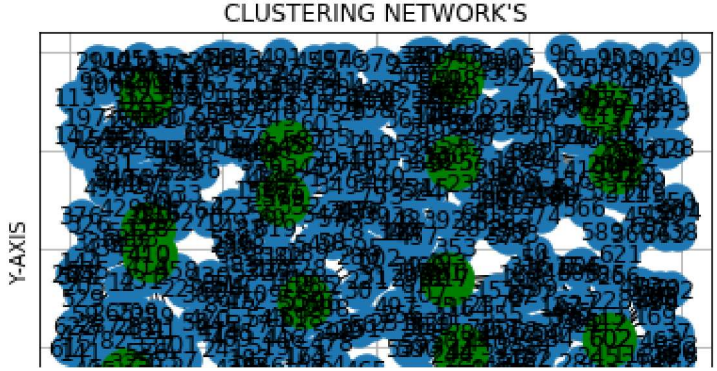


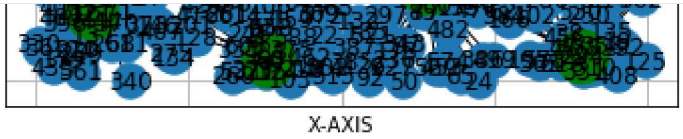



```
shorted_path: ([(0, 2), (2, 1), (1, 3), (3, 7), (7, 4), (4, 5), (5, 6), (6, 12), (12, 15), (15, 13), (13, 14), (14, 10), (10, 8)
3 2404.476856138397
151.21038720934484
0
TOTAL DISTANCE AFTER ITERATION 3 is 2404.476856138397
```

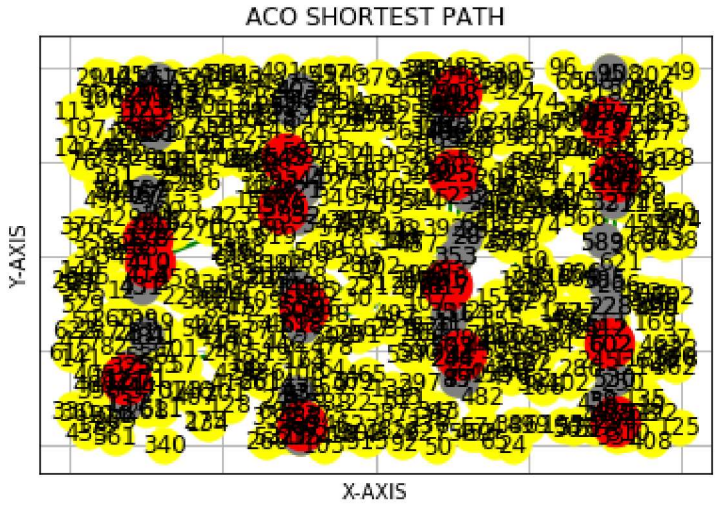



```
[['CLUSTER 1', [121, [92.17, 175.31], [12.681388012618276], [0]]], ['CLUSTER 2', [410, [129.97, 486.46], [43.222397476340994], [
198.06288142910574
##########
True
```

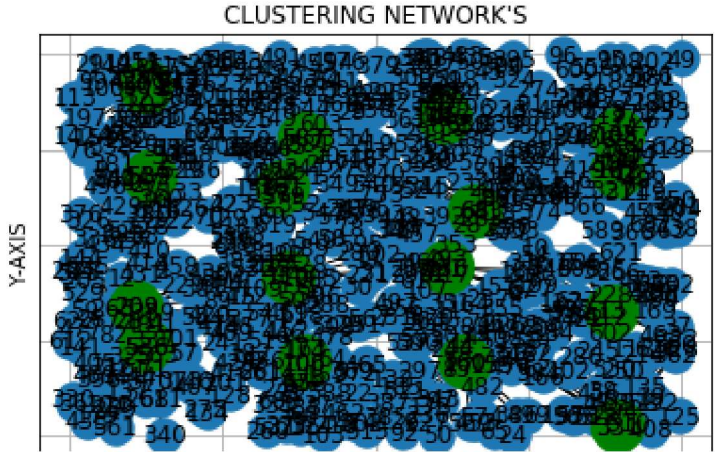


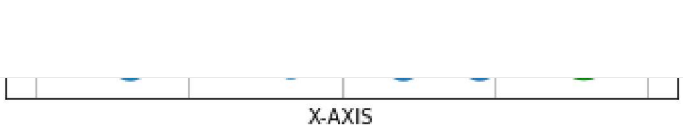



```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 6), (6, 7), (7, 4), (4, 5), (5, 14), (14, 12), (12, 13), (13, 15), (15, 8), (8, 11),
4 2465.01893764193
198.06288142910574
0
TOTAL DISTANCE AFTER ITERATION 4 is 2465.01893764193
```

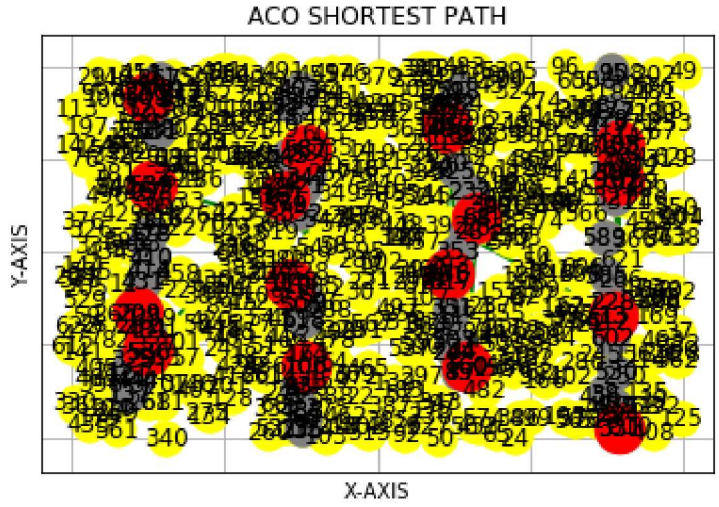



```
[['CLUSTER 1', [536, [124.13, 235.05], [56.537854889590626], [0]]], ['CLUSTER 2', [209, [108.31, 338.05], [21.98107255520501], [
265.81339206292824
##########
True
```

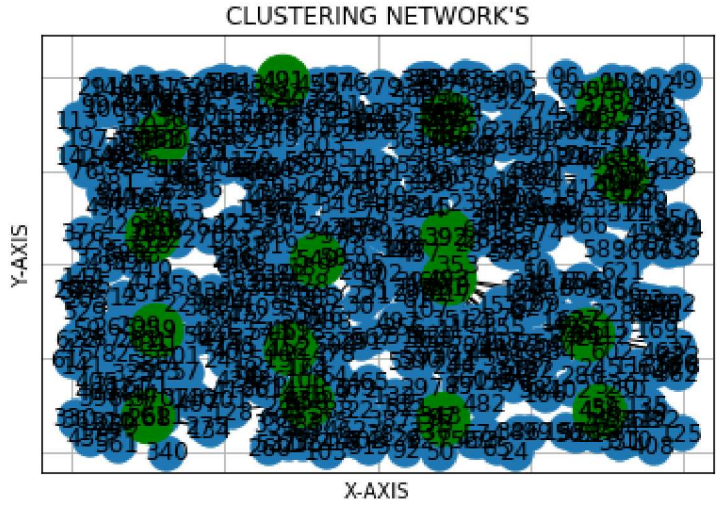





```
shorted_path: ([(0, 3), (3, 1), (1, 2), (2, 9), (9, 11), (11, 8), (8, 10), (10, 14), (14, 12), (12, 15), (15, 13), (13, 7), (7,
5 2498.404058710086
265.81339206292824
0
TOTAL DISTANCE AFTER ITERATION 5 is 2498.404058710086
```

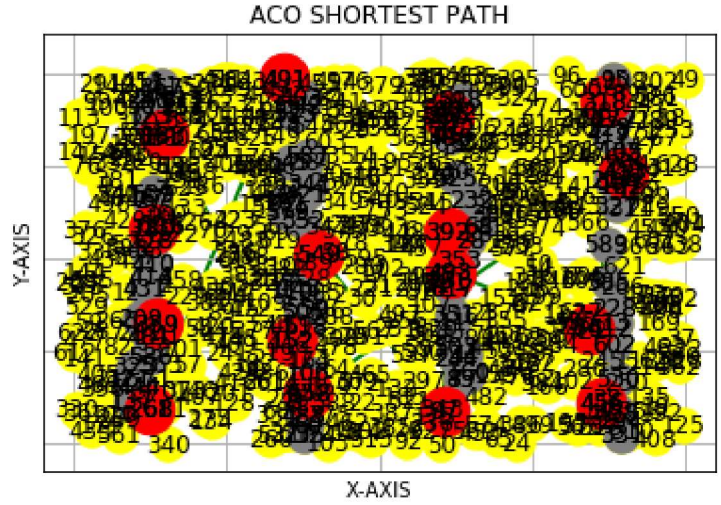



```
[['CLUSTER 1', [268, [125.42, 95.09], [28.21608832807566], [0]]], ['CLUSTER 2', [359, [135.81, 325.5], [37.83280757097805], [0]]
157.3921360805552
##########
True
```

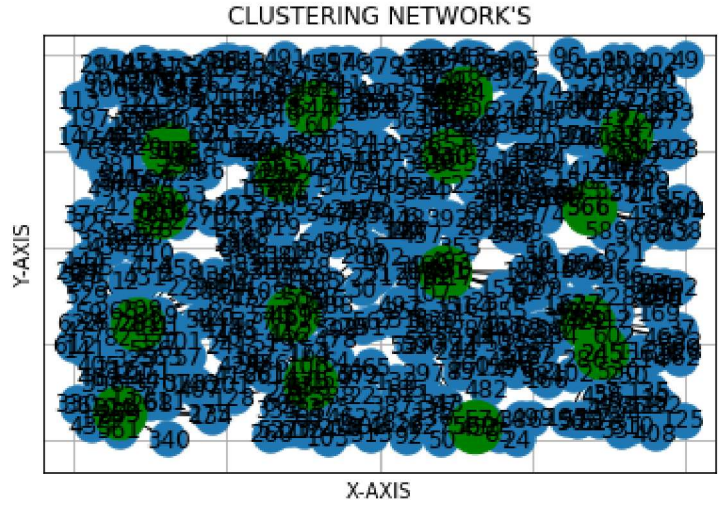



```
shorted_path: ([(0, 2), (2, 1), (1, 3), (3, 4), (4, 14), (14, 15), (15, 12), (12, 13), (13, 9), (9, 11), (11, 10), (10, 8), (8,
```

```
shorted_path: ([(0, 2), (2, 1), (1, 5), (5, 4), (4, 14), (14, 15), (15, 12), (12, 15), (15, 9), (9, 11), (11, 10), (10, 8), (8,
6 2348.037228467727
157.3921360805552
0
TOTAL DISTANCE AFTER ITERATION 6 is 2348.037228467727
```

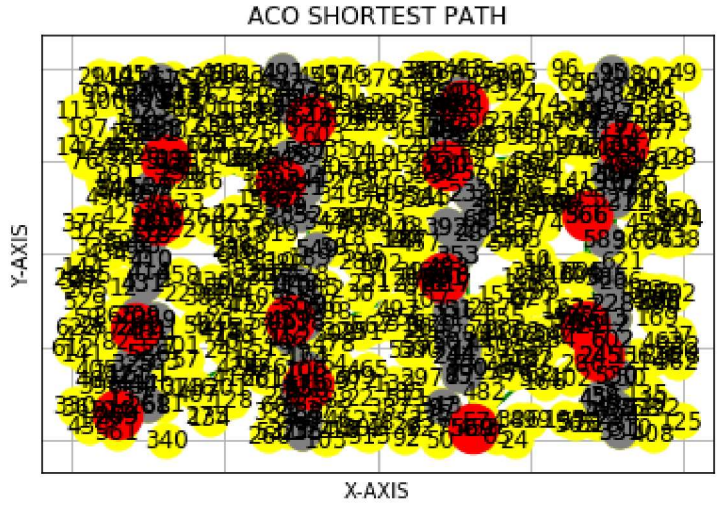



```
[['CLUSTER 1', [269, [74.67, 68.86], [28.321766561514146], [0]]], ['CLUSTER 2', [281, [105.96, 301.74], [29.589905362775973], [0
101.57415271613148
##########
True
```

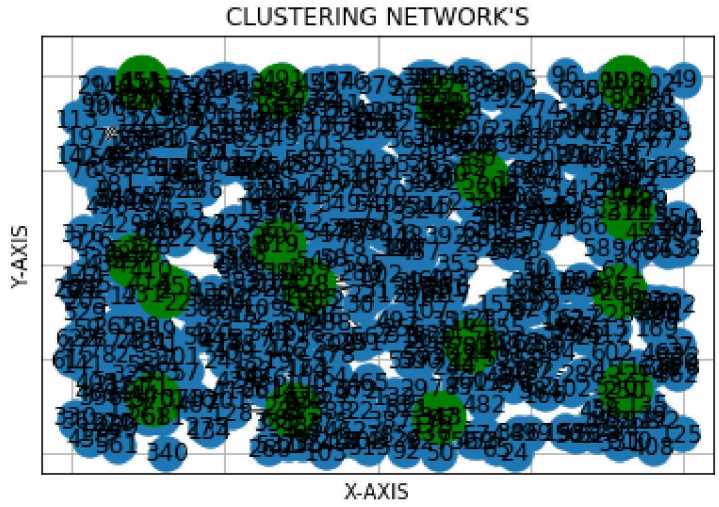



```
shorted_path: ([(0, 1), (1, 2), (2, 3), (3, 6), (6, 5), (5, 4), (4, 7), (7, 11), (11, 8), (8, 13), (13, 14), (14, 12), (12, 15),
7 2442.380441156078
101.57415271613148
```

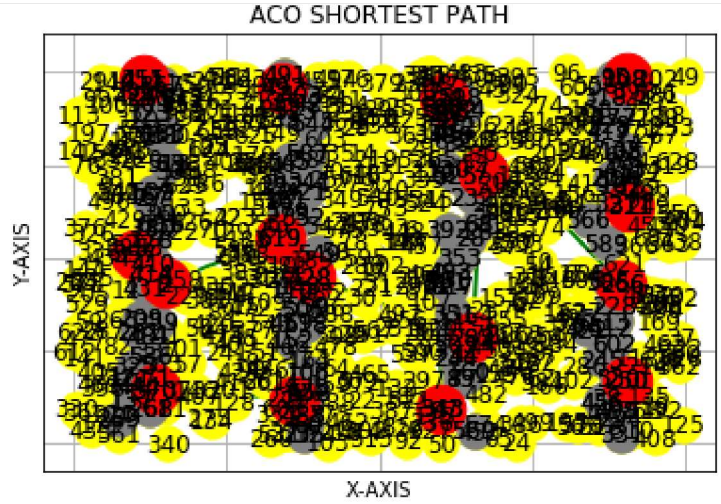
```
0
TOTAL DISTANCE AFTER ITERATION 7 is 2442.380441156078
```



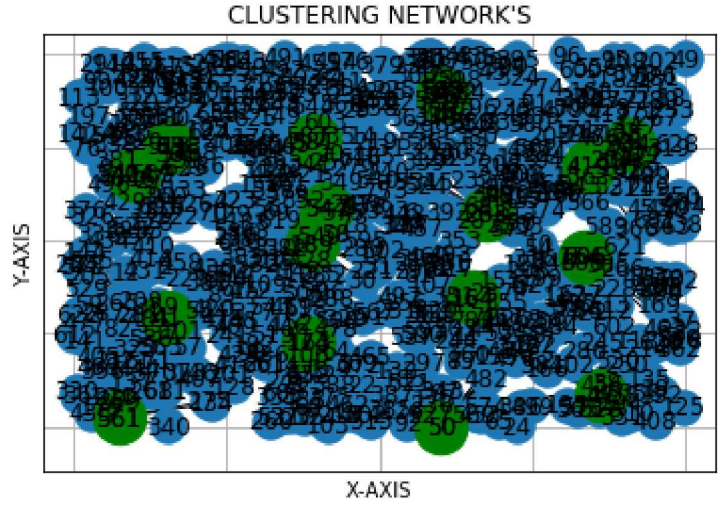

```
[['CLUSTER 1', [570, [132.81, 145.42], [60.13091482649926], [0]]], ['CLUSTER 2', [2, [150.48, 431.66], [0.1056782334384858], [0]
196.94027647995216
##########
True
```



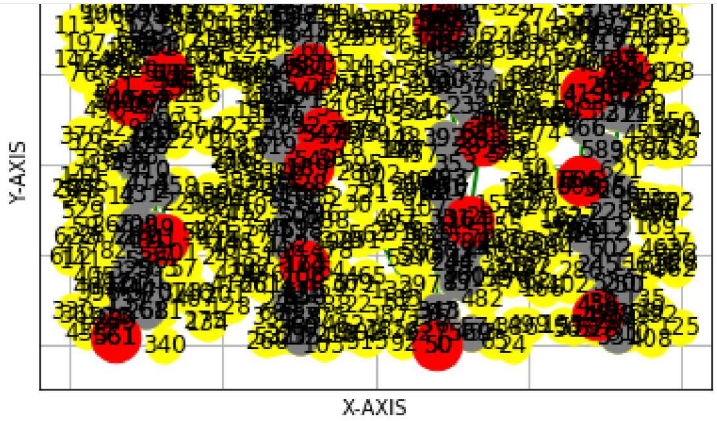

```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 6), (6, 7), (7, 4), (4, 5), (5, 9), (9, 10), (10, 14), (14, 12), (12, 15), (15, 13),
8 2488.6733543055525
196.94027647995216
0
TOTAL DISTANCE AFTER ITERATION 8 is 2488.6733543055525
```



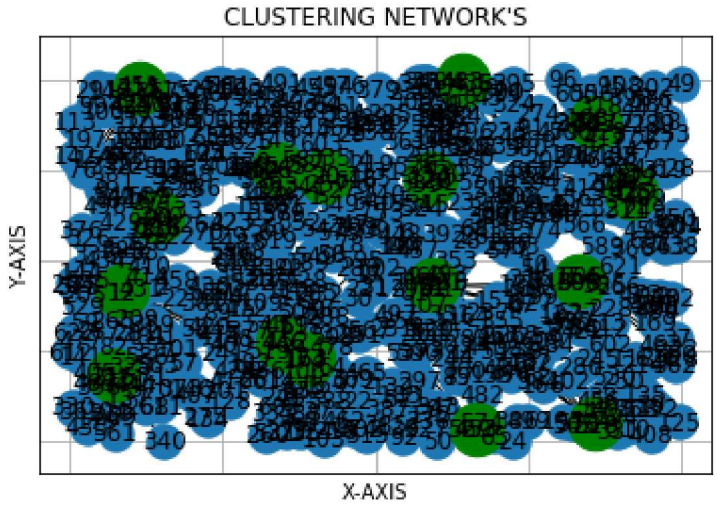

```
[['CLUSTER 1', [561, [74.16, 23.14], [59.179810725552855], [0]]], ['CLUSTER 2', [41, [152.83, 295.97], [4.227129337539428], [0]]
77.68632569506683
##########
True
```



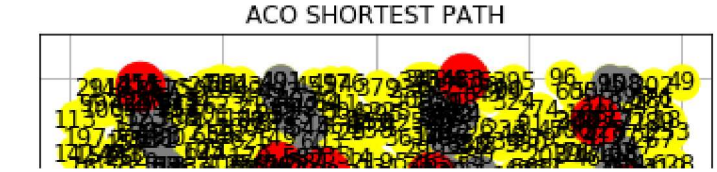

```
shorted_path: ([(0, 2), (2, 1), (1, 3), (3, 4), (4, 5), (5, 7), (7, 6), (6, 8), (8, 11), (11, 9), (9, 10), (10, 13), (13, 14), (
9 2434.614648746067
77.68632569506683
0
TOTAL DISTANCE AFTER ITERATION 9 is 2434.614648746067
```

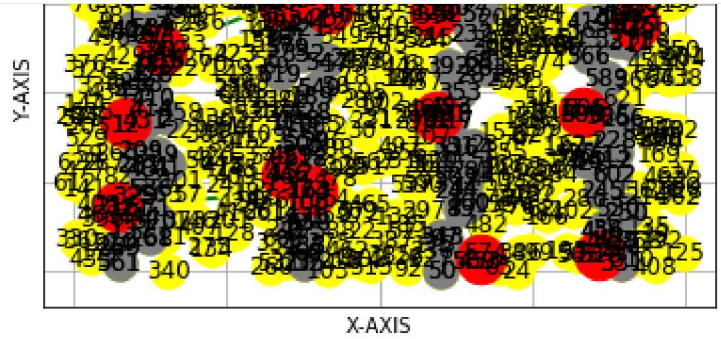



```
[['CLUSTER 1', [216, [71.62, 180.15], [22.72082018927441], [0]]], ['CLUSTER 2', [12, [85.4, 413.86], [1.1624605678233437], [0]]]
193.8645065503224
##########
True
```

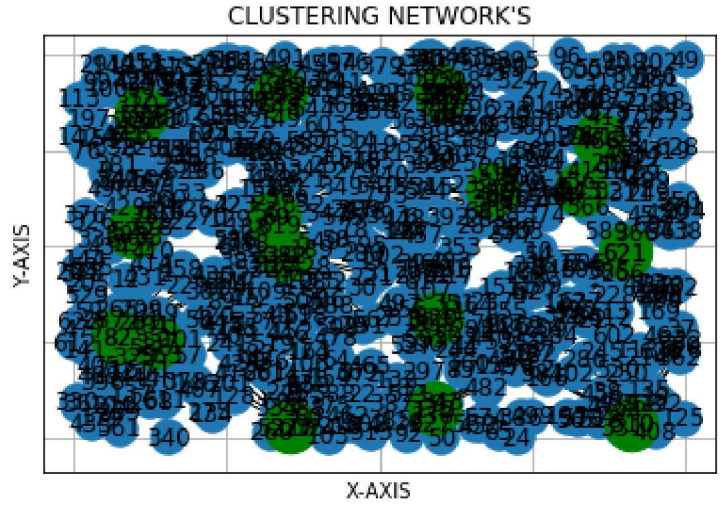



```
shorted_path: ([(0, 1), (1, 3), (3, 2), (2, 7), (7, 5), (5, 6), (6, 10), (10, 9), (9, 8), (8, 13), (13, 12), (12, 15), (15, 14),
10 2588.580615987932
193.8645065503224
0
TOTAL DISTANCE AFTER ITERATION 10 is 2588.580615987932
```

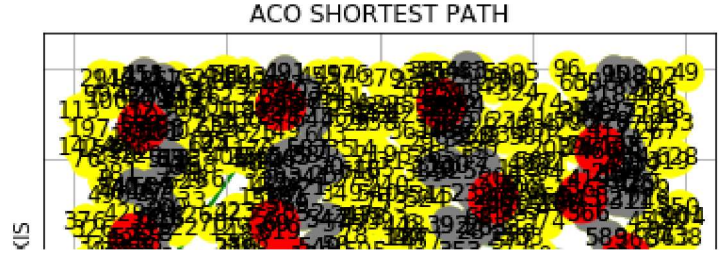


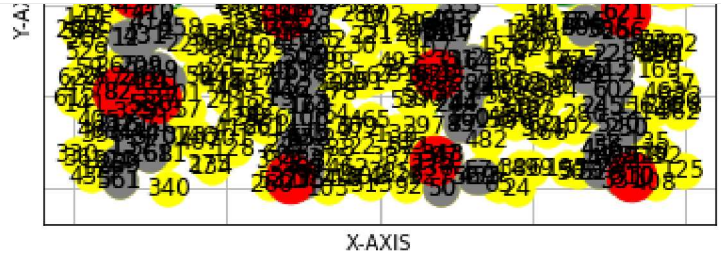



```
[['CLUSTER 1', [337, [134.59, 248.09], [35.50788643533129], [0]]], ['CLUSTER 2', [82, [71.74, 264.31], [8.559936908517338], [0]]
282.24655214900326
##########
True
```

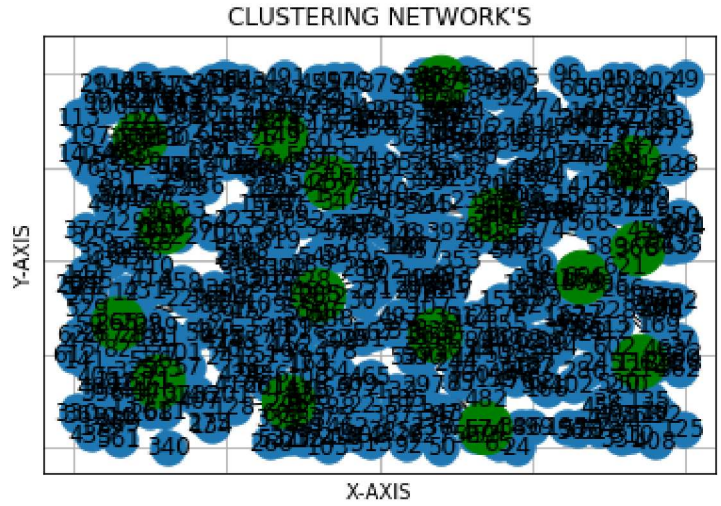



```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 7), (7, 5), (5, 4), (4, 8), (8, 9), (9, 11), (11, 10), (10, 14), (14, 15), (15, 12),
11 2660.444647178095
282.24655214900326
0
TOTAL DISTANCE AFTER ITERATION 11 is 2660.444647178095
```

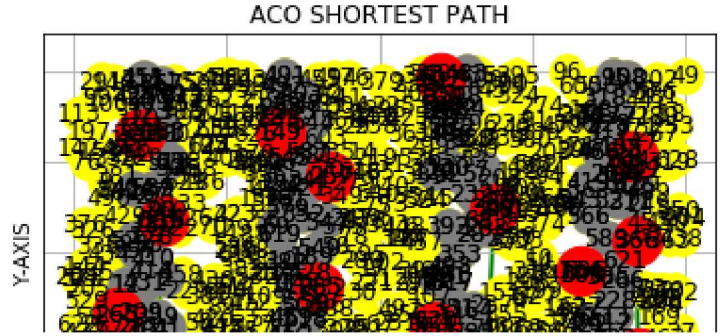


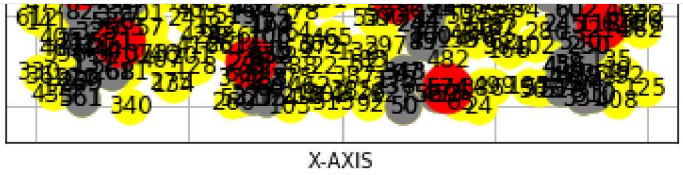



```
[['CLUSTER 1', [71, [136.6, 177.26], [7.397476340693996], [0]]], ['CLUSTER 2', [265, [71.02, 335.68], [27.899053627760203], [0]]
223.7871032923926
##########
True
```

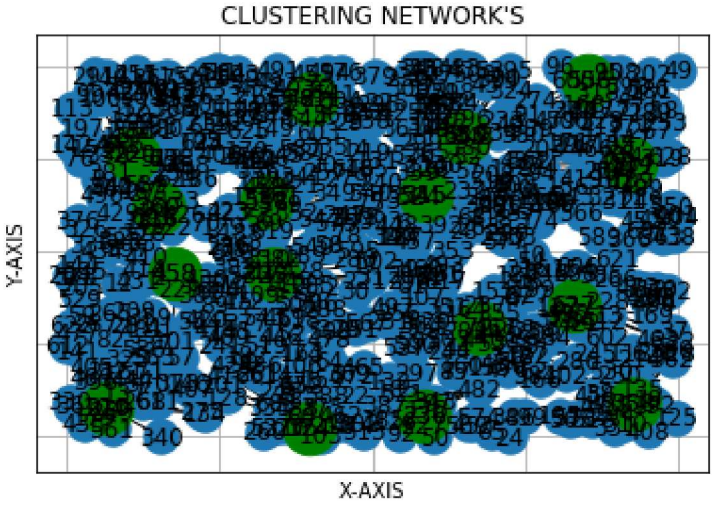



```
shorted_path: ([(0, 2), (2, 7), (7, 4), (4, 5), (5, 6), (6, 11), (11, 9), (9, 13), (13, 12), (12, 14), (14, 15), (15, 10), (10,
12 2661.9478533661036
223.7871032923926
0
TOTAL DISTANCE AFTER ITERATION 12 is 2661.9478533661036
```

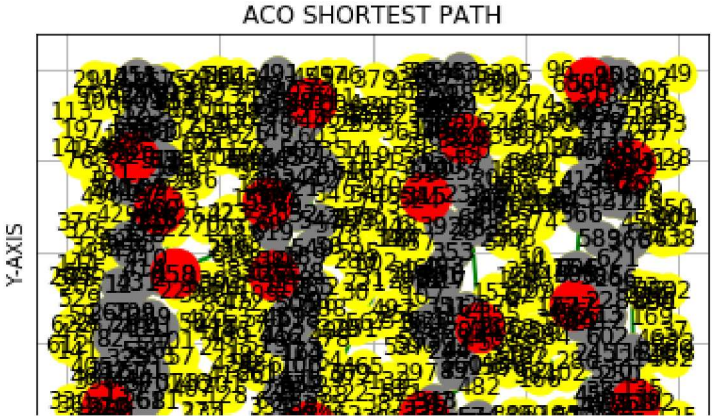


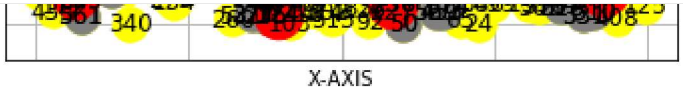



```
[['CLUSTER 1', [210, [65.65, 80.85], [22.086750788643496], [0]]], ['CLUSTER 2', [458, [175.59, 443.38], [48.29495268138847], [0]
104.14722751950721
##########
True
```

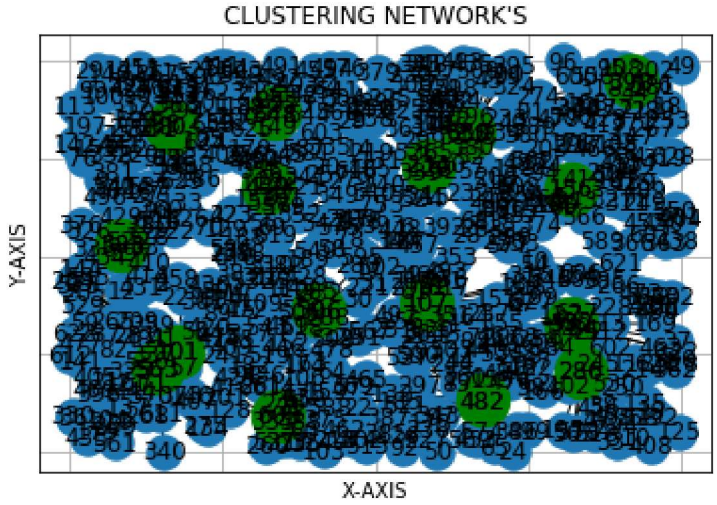



```
shorted_path: ([(0, 6), (6, 5), (5, 7), (7, 4), (4, 10), (10, 8), (8, 13), (13, 15), (15, 14), (14, 12), (12, 9), (9, 11), (11,
13 2560.183637021614
104.14722751950721
0
TOTAL DISTANCE AFTER ITERATION 13 is 2560.183637021614
```

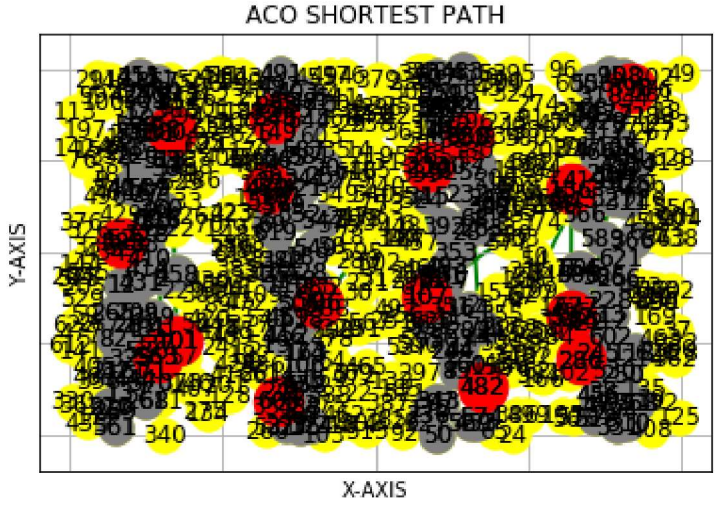




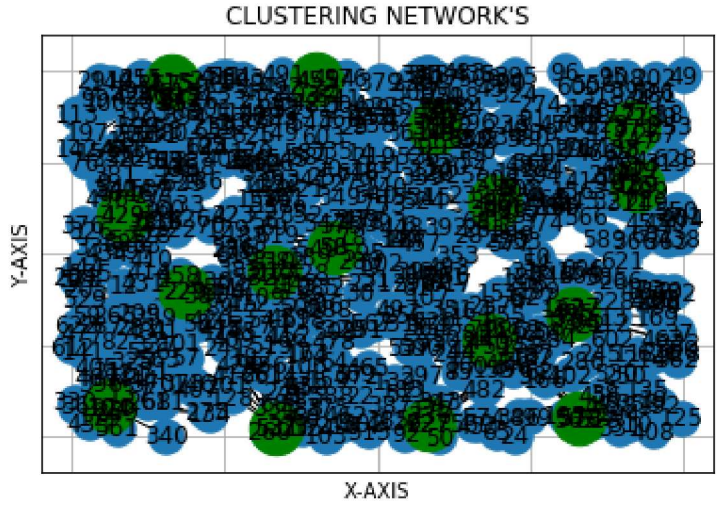

```
[['CLUSTER 1', [563, [142.22, 214.66], [59.39116719242983], [0]]], ['CLUSTER 2', [101, [175.33, 257.61], [10.567823343848564], [
257.49843494670023
##########
True
```



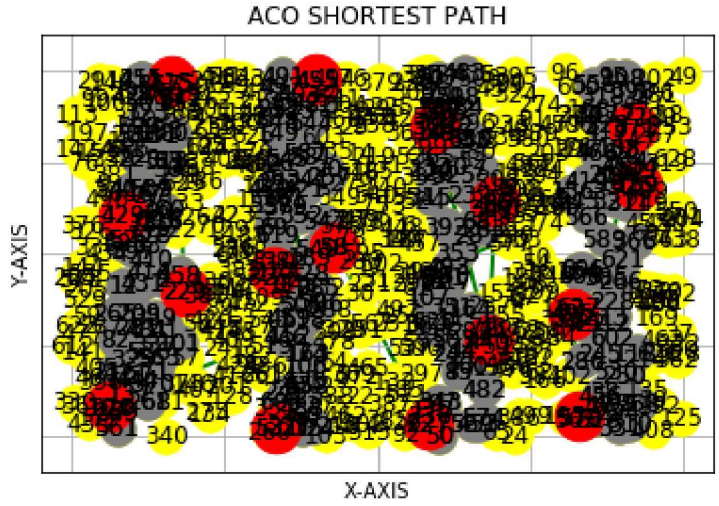

```
shorted_path: ([(0, 3), (3, 1), (1, 10), (10, 9), (9, 14), (14, 13), (13, 12), (12, 15), (15, 8), (8, 11), (11, 5), (5, 4), (4,
14 2618.9522413972945
257.49843494670023
0
TOTAL DISTANCE AFTER ITERATION 14 is 2618.9522413972945
```



```
[['CLUSTER 1', [104, [62.21, 81.47], [10.884858044164021], [0]]], ['CLUSTER 2', [229, [185.59, 397.94], [24.094637223974722], [0
102.50582910254421
##########
True
```

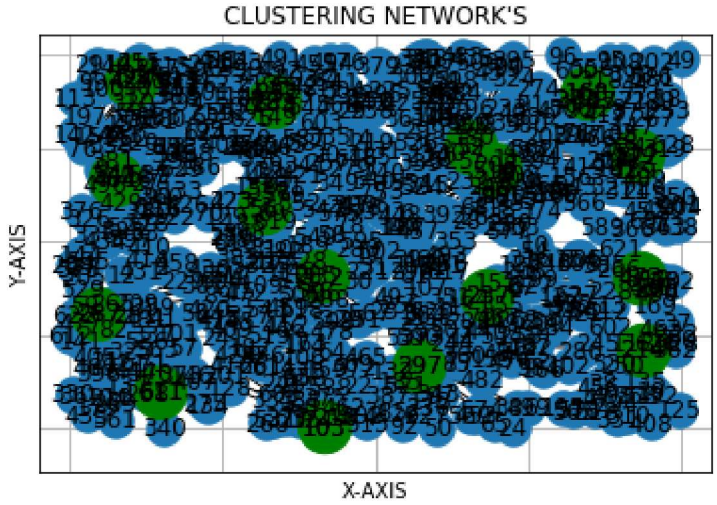



```
shorted_path: ([(0, 2), (2, 3), (3, 1), (1, 5), (5, 4), (4, 7), (7, 6), (6, 8), (8, 11), (11, 9), (9, 10), (10, 13), (13, 12), (
15 2546.662844680717
102.50582910254421
0
TOTAL DISTANCE AFTER ITERATION 15 is 2546.662844680717
```

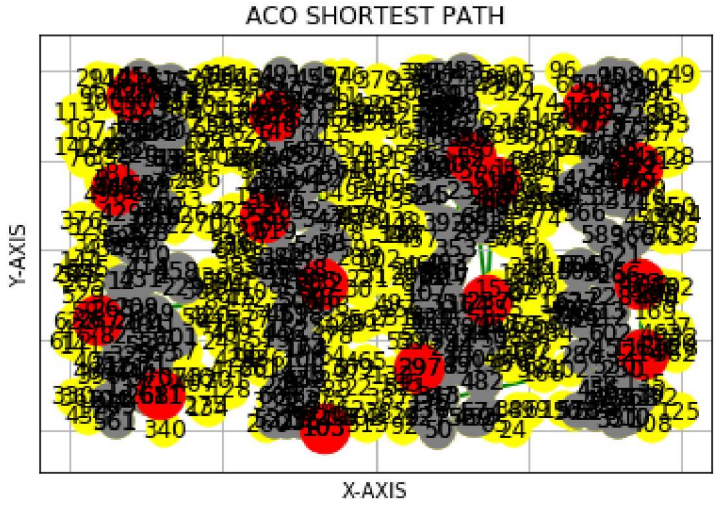



```
[['CLUSTER 1', [611, [147.76, 97.03], [64.46372239747728], [0]]], ['CLUSTER 2', [287, [46.4, 306.77], [30.223974763406886], [0]]
176.77058154568593
```

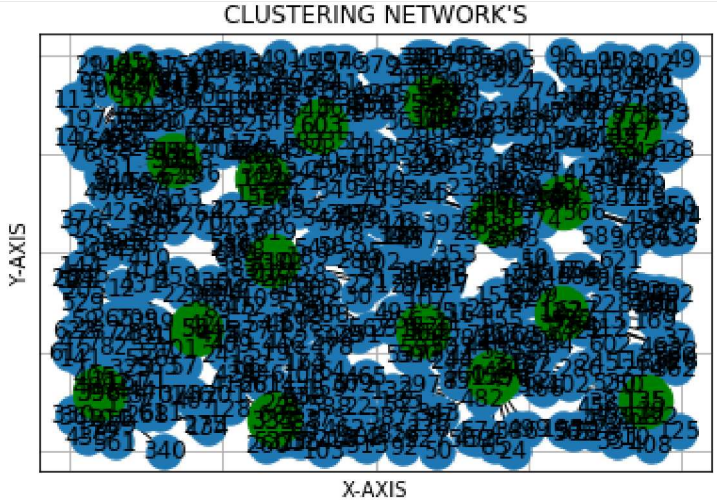
```
##########
True
```



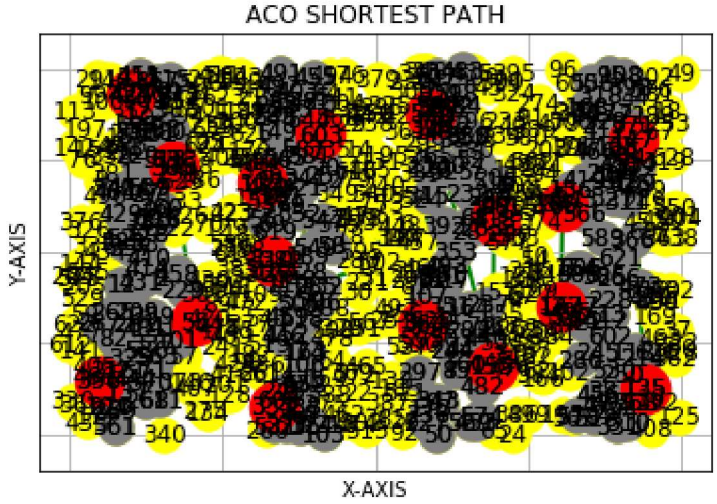

```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 5), (5, 4), (4, 12), (12, 13), (13, 14), (14, 15), (15, 10), (10, 9), (9, 11), (11,
16 2696.8425811520465
176.77058154568593
0
TOTAL DISTANCE AFTER ITERATION 16 is 2696.8425811520465
```



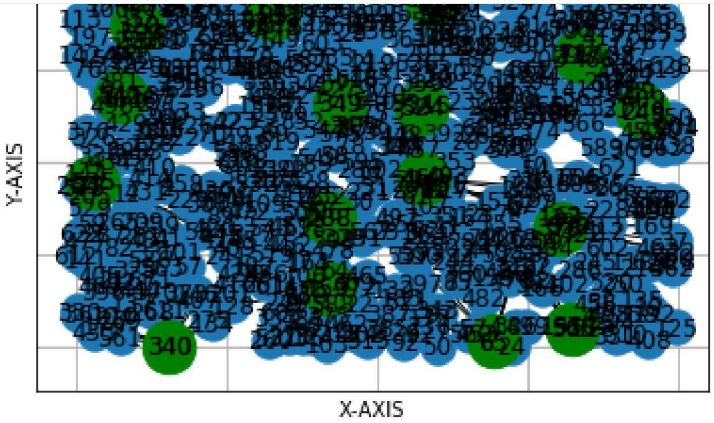

```
[['CLUSTER 1', [396, [46.94, 148.26], [41.742902208202146], [0]]], ['CLUSTER 2', [55, [204.68, 310.9], [5.706624605678226], [0]]
155.5133151855493
##########
True
```



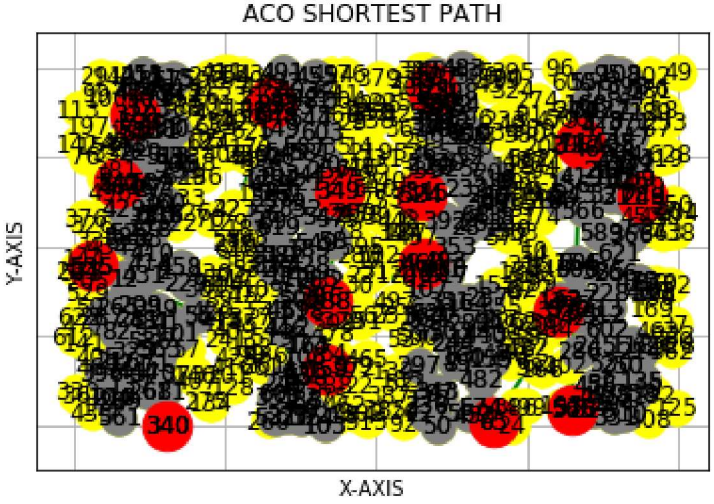

```
shorted_path: ([(0, 3), (3, 6), (6, 5), (5, 4), (4, 7), (7, 9), (9, 11), (11, 8), (8, 10), (10, 15), (15, 12), (12, 13), (13, 14
17 2690.2082830268478
155.5133151855493
0
TOTAL DISTANCE AFTER ITERATION 17 is 2690.2082830268478
```



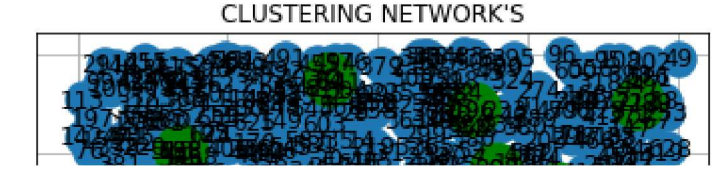

```
[['CLUSTER 1', [340, [154.93, 0.55], [35.824921135646754], [0]]], ['CLUSTER 2', [535, [29.34, 446.73], [56.43217665615214], [0]]
154.93097624426176
##########
True
```

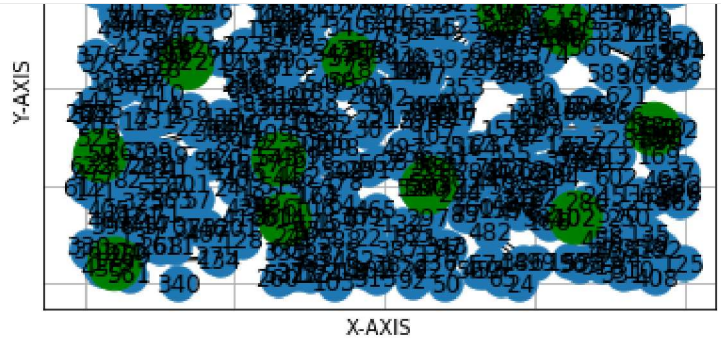



```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 4), (4, 5), (5, 6), (6, 10), (10, 9), (9, 11), (11, 13), (13, 12), (12, 15), (15, 14
18 2734.08881848657
154.93097624426176
0
TOTAL DISTANCE AFTER ITERATION 18 is 2734.08881848657
```

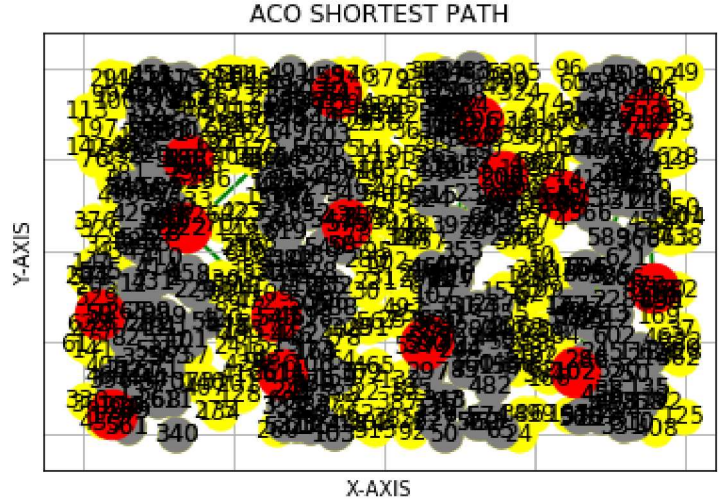



```
[['CLUSTER 1', [179, [47.0, 53.98], [18.810725552050442], [0]]], ['CLUSTER 2', [59, [26.09, 330.28], [6.129337539432169], [0]]],
71.57402042640891
##########
True
```

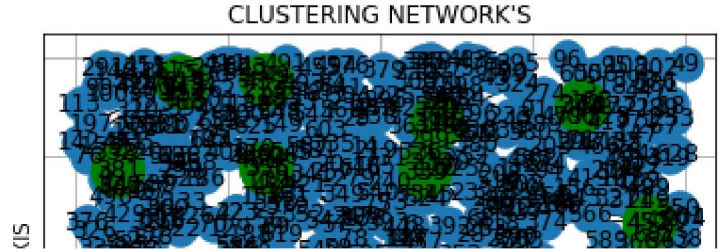


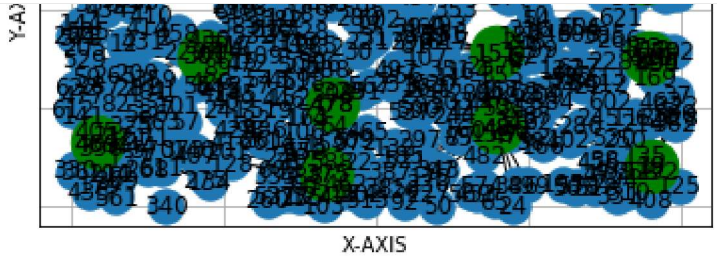



```
shorted_path: ([(0, 1), (1, 7), (7, 6), (6, 8), (8, 9), (9, 11), (11, 14), (14, 13), (13, 15), (15, 12), (12, 10), (10, 4), (4,
19 2687.3599939341807
71.57402042640891
0
TOTAL DISTANCE AFTER ITERATION 19 is 2687.3599939341807
```

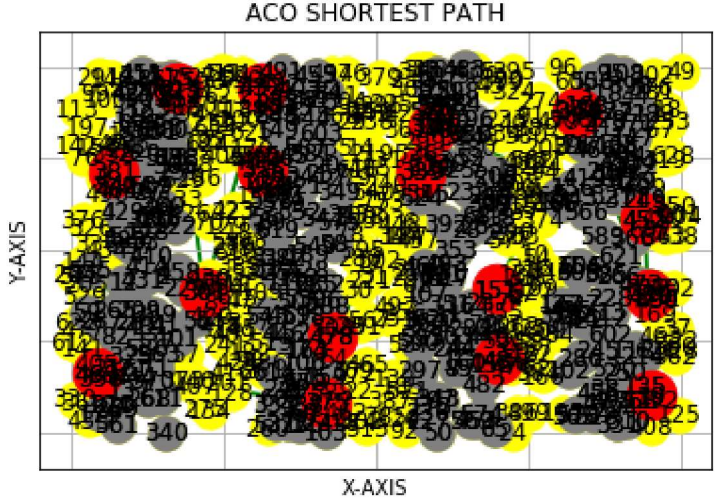



```
[['CLUSTER 1', [469, [41.73, 162.36], [49.457413249211854], [0]]], ['CLUSTER 2', [301, [218.1, 385.44], [31.703470031545685], [0
167.63699621503602
##########
True
```

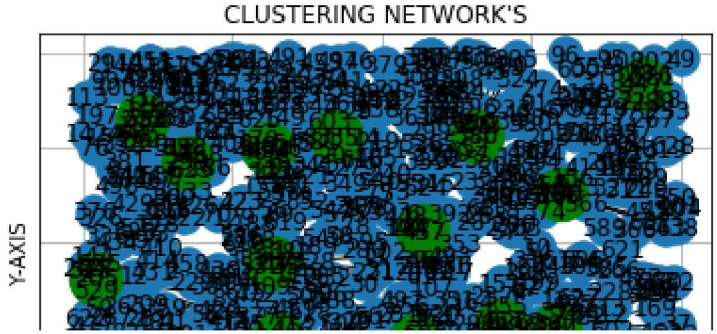


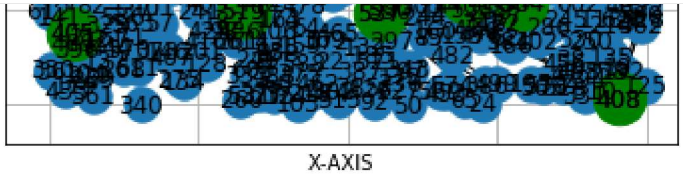



```
shorted_path: ([(0, 2), (2, 3), (3, 1), (1, 7), (7, 6), (6, 8), (8, 9), (9, 15), (15, 13), (13, 12), (12, 14), (14, 11), (11, 10
20 2744.1078326888355
167.63699621503602
0
TOTAL DISTANCE AFTER ITERATION 20 is 2744.1078326888355
```

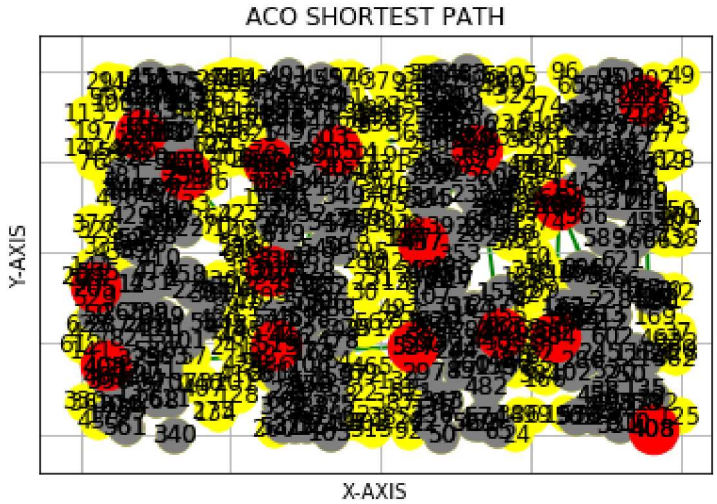



```
[['CLUSTER 1', [405, [41.46, 190.22], [42.69400630914855], [0]]], ['CLUSTER 2', [205, [22.98, 406.07], [21.558359621451068], [0]
194.68584951146295
##########
True
```

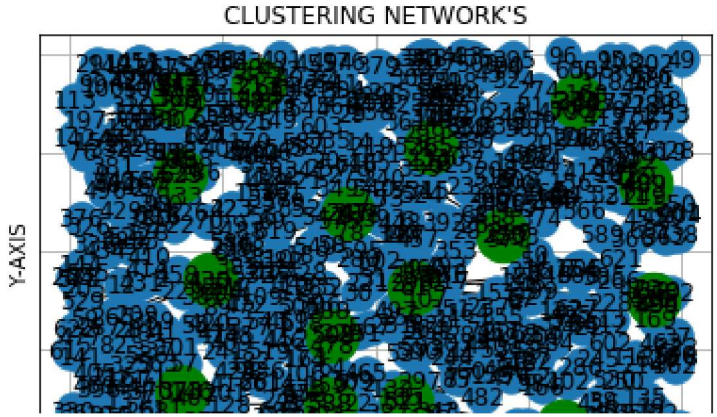


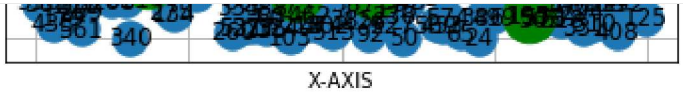



```
shorted_path: ([(0, 1), (1, 3), (3, 2), (2, 5), (5, 6), (6, 4), (4, 7), (7, 15), (15, 12), (12, 14), (14, 13), (13, 9), (9, 11),
21 2836.5216368381466
194.68584951146295
0
TOTAL DISTANCE AFTER ITERATION 21 is 2836.5216368381466
```

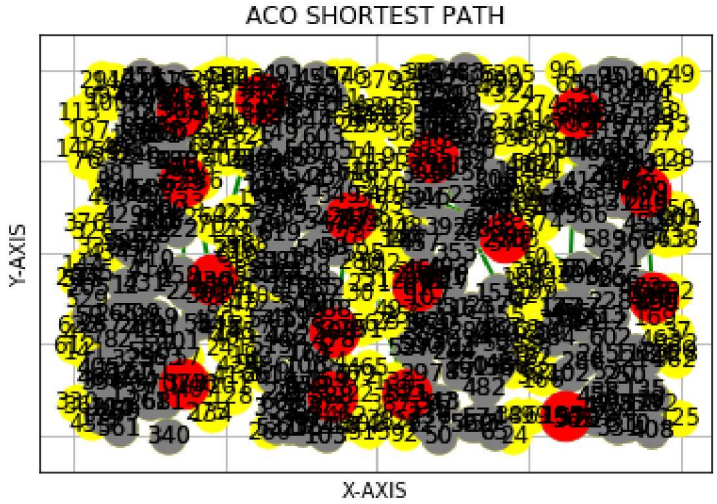



```
[['CLUSTER 1', [373, [182.67, 144.52], [39.312302839116896], [0]]], ['CLUSTER 2', [430, [227.47, 426.99], [45.33596214511078], [
232.92565187200827
##########
True
```

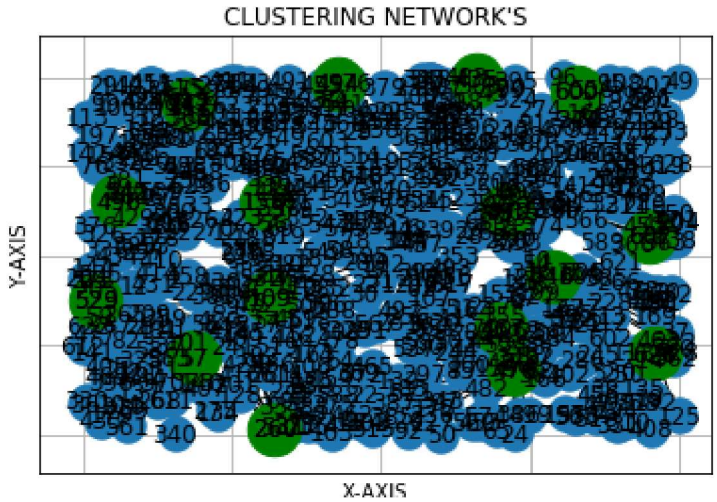




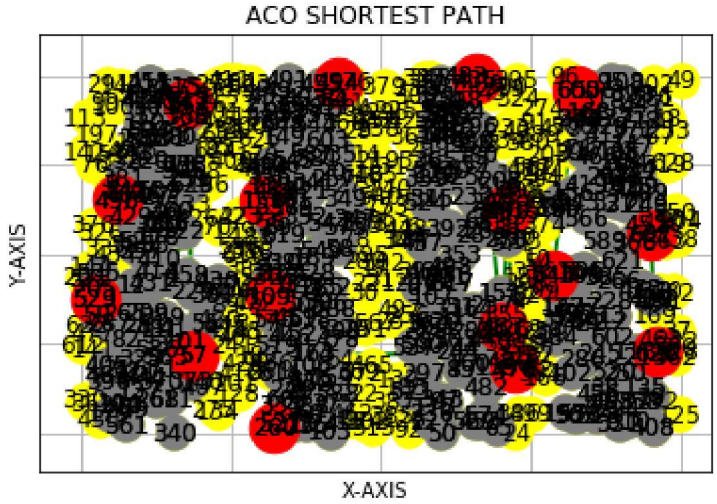

```
shorted_path: ([(0, 2), (2, 3), (3, 1), (1, 4), (4, 5), (5, 6), (6, 8), (8, 9), (9, 11), (11, 12), (12, 15), (15, 14), (14, 13),
22 2431.914604277181
232.92565187200827
0
TOTAL DISTANCE AFTER ITERATION 22 is 2431.914604277181
```



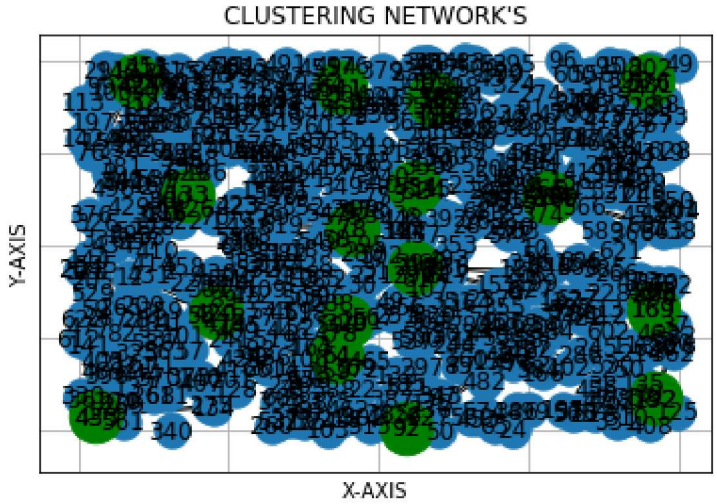

```
[['CLUSTER 1', [57, [185.79, 216.03], [5.917981072555198], [0]]], ['CLUSTER 2', [529, [21.75, 374.49], [55.7981072555212], [0]]]
284.9331237325699
##########
True
```



```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 6), (6, 5), (5, 4), (4, 7), (7, 11), (11, 9), (9, 10), (10, 8), (8, 13), (13, 15), (
23 2936.1027205709165
284.9331237325699
0
TOTAL DISTANCE AFTER ITERATION 23 is 2936.1027205709165
```

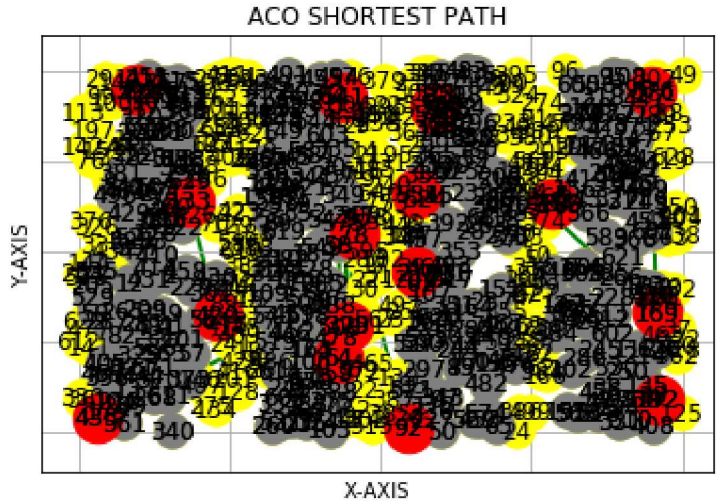



```
[['CLUSTER 1', [433, [29.53, 37.44], [45.652996845426244], [0]]], ['CLUSTER 2', [420, [229.32, 320.31], [44.279179810725886], [0
47.68411160963366
##########
True
```

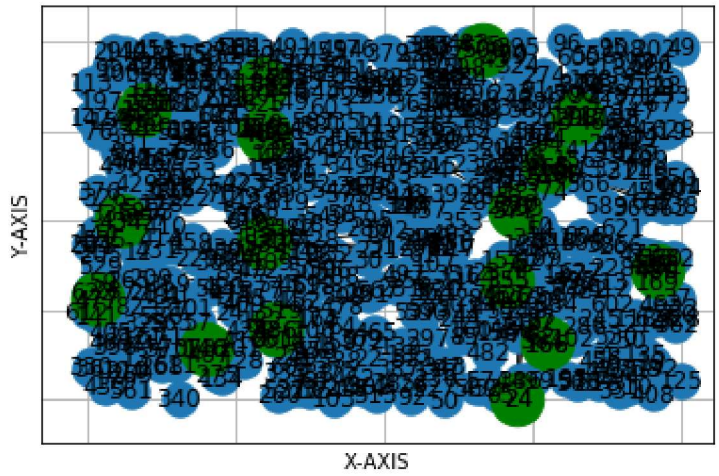



```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 4), (4, 7), (7, 5), (5, 6), (6, 8), (8, 10), (10, 9), (9, 11), (11, 13), (13, 12), (
24 2680.1012366109135
```

```
47.68411160963366
0
TOTAL DISTANCE AFTER ITERATION 24 is 2680.1012366109135
```

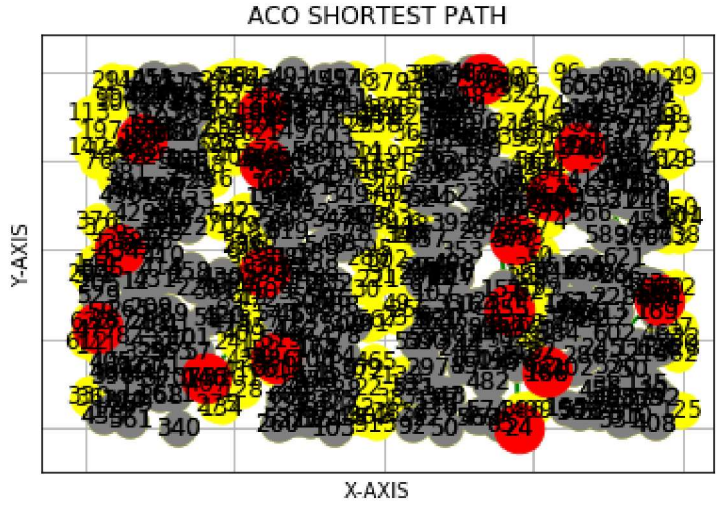



```
[['CLUSTER 1', [146, [200.61, 143.76], [15.323343848580416], [0]]], ['CLUSTER 2', [477, [18.54, 281.47], [50.30283911671977], [0
246.80216712986945
##########
True
```

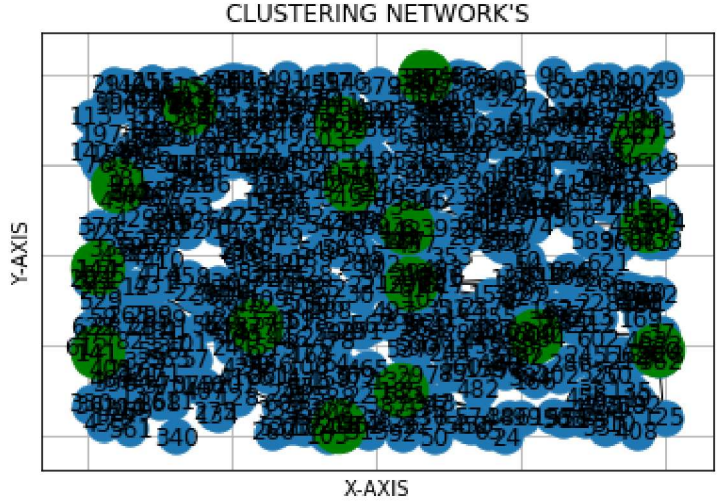



```
shorted_path: ([(0, 2), (2, 1), (1, 3), (3, 7), (7, 6), (6, 5), (5, 4), (4, 12), (12, 14), (14, 15), (15, 13), (13, 8), (8, 10),
25 2907.3904262936203
246.80216712986945
0
```

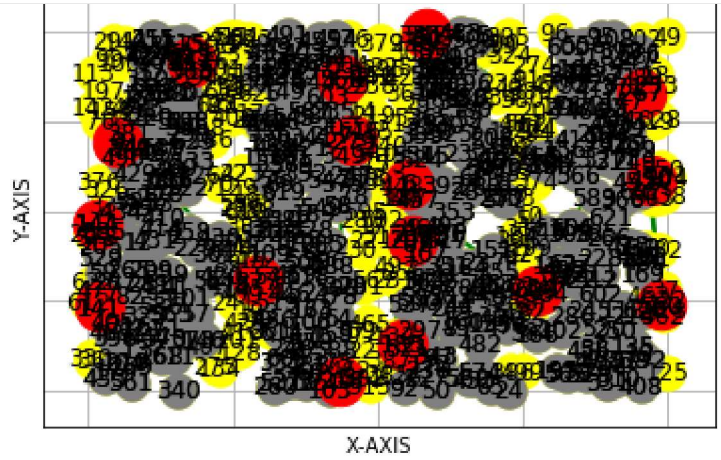
```
TOTAL DISTANCE AFTER ITERATION 25 is 2907.3904262936203
```



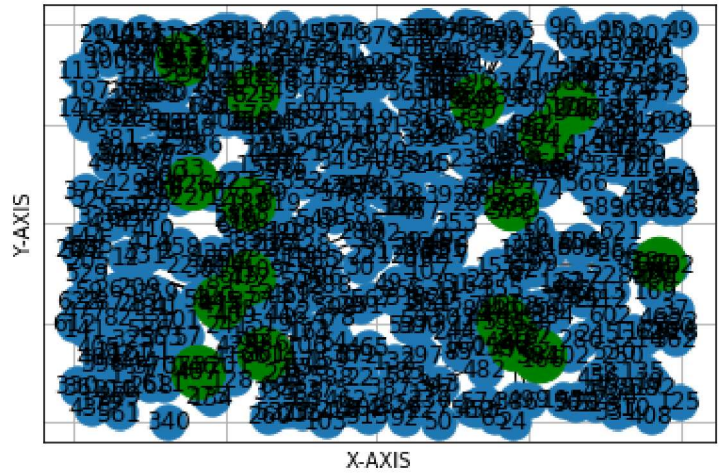

```
[['CLUSTER 1', [141, [19.09, 232.97], [14.794952681387988], [0]]], ['CLUSTER 2', [144, [17.72, 467.13], [15.111987381703445], [0
233.75082673650587
##########
True
```



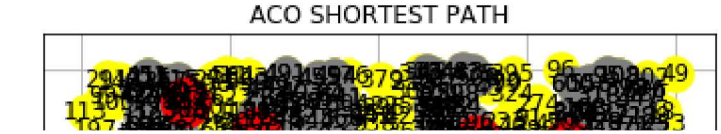

```
shorted_path: ([(0, 1), (1, 2), (2, 5), (5, 4), (4, 7), (7, 6), (6, 8), (8, 10), (10, 9), (9, 11), (11, 14), (14, 12), (12, 15),
26 2976.335188246649
233.75082673650587
0
TOTAL DISTANCE AFTER ITERATION 26 is 2976.335188246649
```

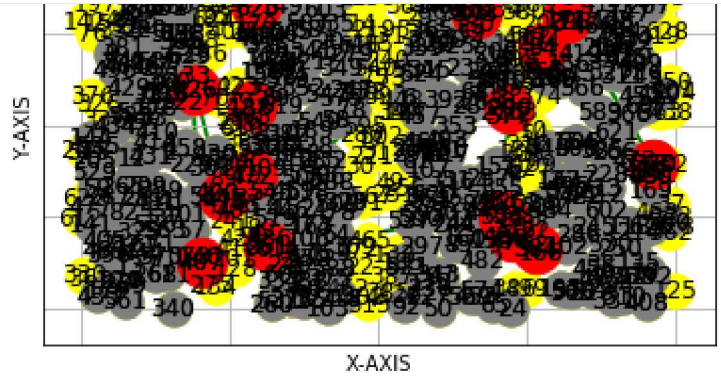



```
[['CLUSTER 1', [407, [204.28, 128.06], [42.90536277602553], [0]]], ['CLUSTER 2', [445, [239.78, 305.55], [46.921135646688114], [
241.10097884496446
##########
True
```

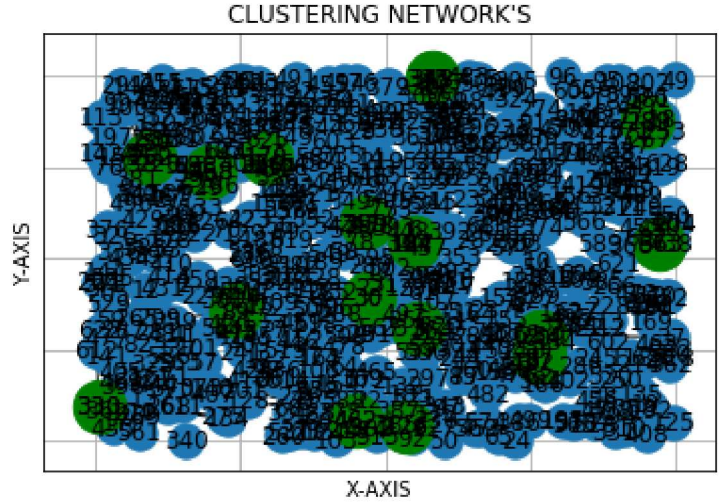



```
shorted_path: ([(0, 3), (3, 2), (2, 1), (1, 6), (6, 5), (5, 7), (7, 4), (4, 9), (9, 10), (10, 8), (8, 12), (12, 14), (14, 15), (
27 2464.414404522955
241.10097884496446
0
TOTAL DISTANCE AFTER ITERATION 27 is 2464.414404522955
```

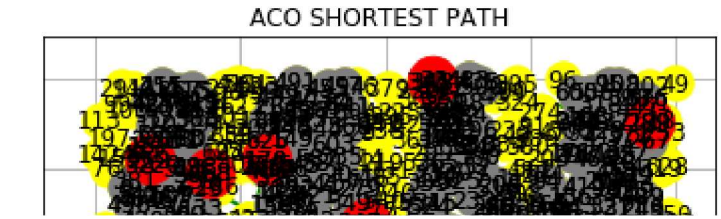


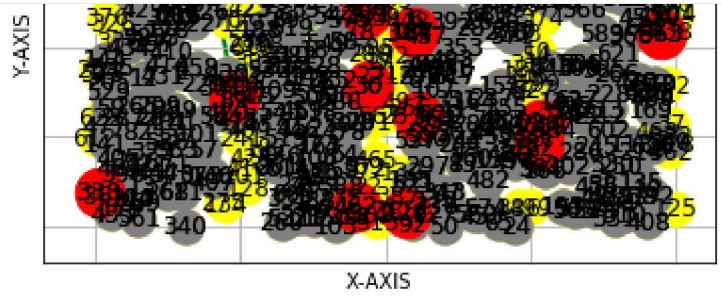



```
[['CLUSTER 1', [330, [5.83, 94.88], [34.76813880126186], [0]]], ['CLUSTER 2', [89, [240.03, 356.76], [9.299684542586737], [0]]],
95.05894644903235
##########
True
```

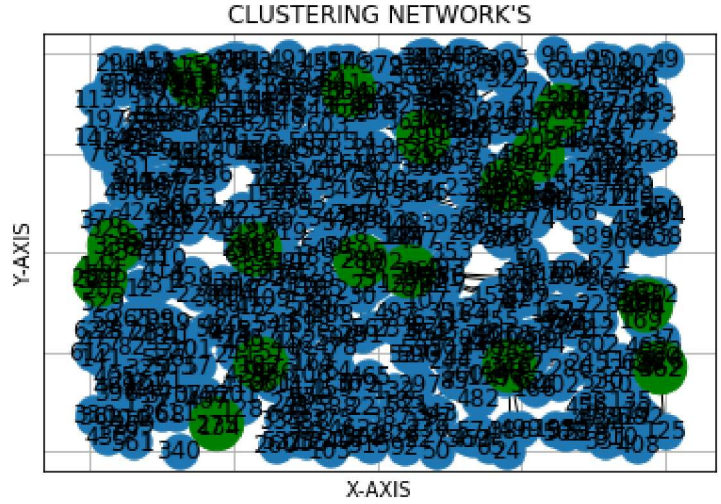



```
shorted_path: ([(0, 2), (2, 1), (1, 7), (7, 4), (4, 6), (6, 5), (5, 10), (10, 9), (9, 8), (8, 11), (11, 15), (15, 14), (14, 13),
28 2886.829655658512
95.05894644903235
0
TOTAL DISTANCE AFTER ITERATION 28 is 2886.829655658512
```

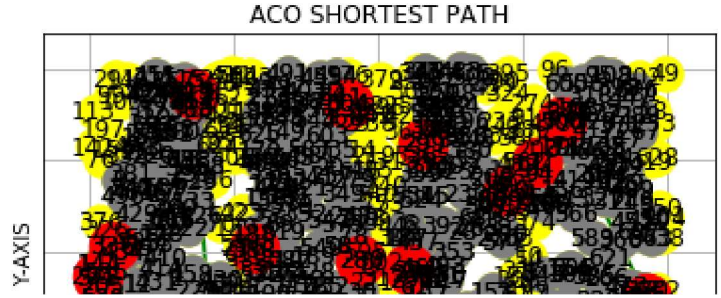


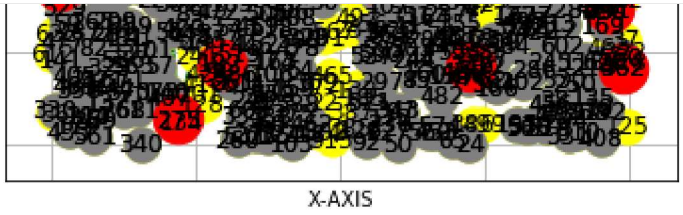



```
[['CLUSTER 1', [275, [219.47, 69.16], [28.95583596214506], [0]]], ['CLUSTER 2', [23, [15.36, 434.32], [2.324921135646687], [0]]]
230.10907522303418
##########
True
```

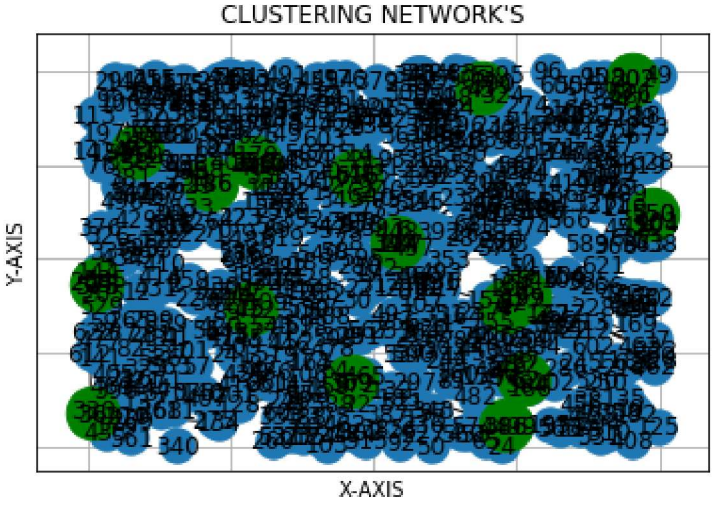



```
shorted_path: ([(0, 6), (6, 4), (4, 7), (7, 5), (5, 9), (9, 11), (11, 10), (10, 8), (8, 14), (14, 15), (15, 13), (13, 12), (12,
29 2987.8255218505697
230.10907522303418
0
TOTAL DISTANCE AFTER ITERATION 29 is 2987.8255218505697
```

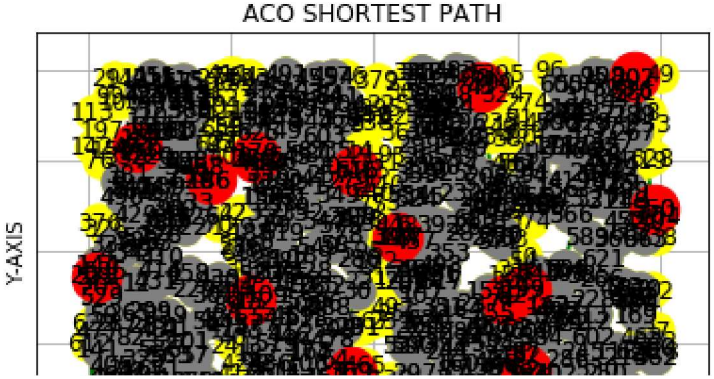


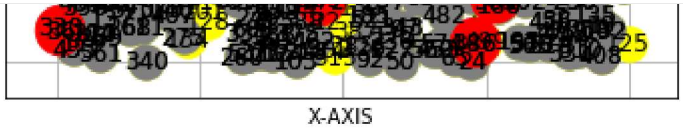



```
[['CLUSTER 1', [36, [4.72, 86.48], [3.6987381703469997], [0]]], ['CLUSTER 2', [573, [13.96, 430.01], [60.447949526814725], [0]]]
86.60871087829446
##########
True
```

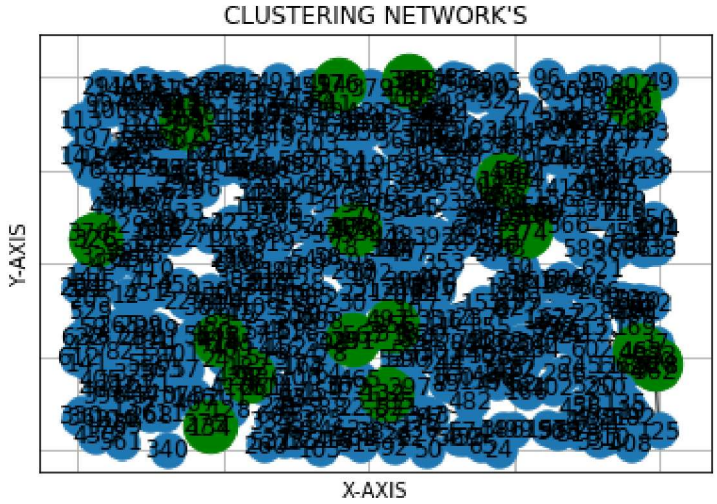



```
shorted_path: ([(0, 1), (1, 2), (2, 7), (7, 5), (5, 12), (12, 13), (13, 14), (14, 15), (15, 8), (8, 9), (9, 11), (11, 10), (10,
30 2888.929453263077
86.60871087829446
0
TOTAL DISTANCE AFTER ITERATION 30 is 2888.929453263077
```

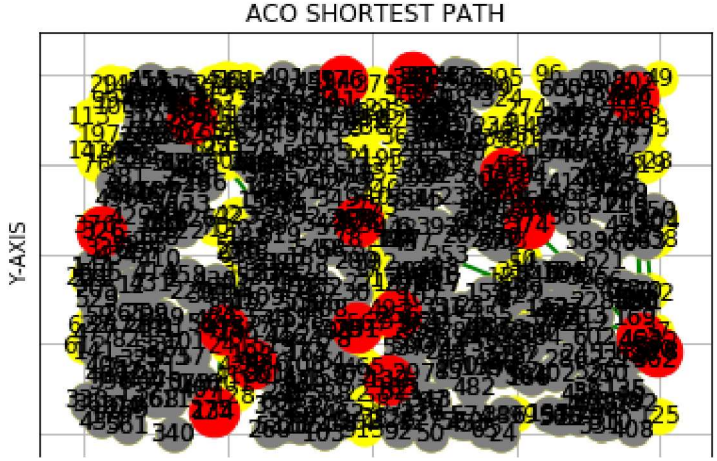


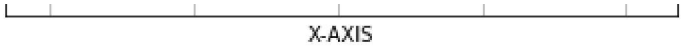



```
[['CLUSTER 1', [134, [226.9, 61.84], [14.055205047318589], [0]]], ['CLUSTER 2', [475, [244.57, 293.59], [50.09148264984279], [0]
235.1760948736074
##########
True
```




```
shorted_path: ([(0, 1), (1, 4), (4, 7), (7, 5), (5, 6), (6, 13), (13, 15), (15, 12), (12, 14), (14, 10), (10, 11), (11, 9), (9,
31 2994.3350393347628
235.1760948736074
0
TOTAL DISTANCE AFTER ITERATION 31 is 2994.3350393347628
```






```
[['CLUSTER 1', [612, [0.66, 247.57], [64.56940063091577], [0]]], ['CLUSTER 2', [622, [8.24, 303.15], [65.62618296530059], [0]]],
247.57087974961834
##########
False
AFTER ALL ITERATION TOTAL DISTANCE IS : 82132.01677040297
```