```
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 fi
        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 tex
        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


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

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

```
        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_par
                        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 partic
            y_cordinate = round(random.randrange(0,breadth),2)+round(random.random(),2)  # creates the y cordinate of the part
            # print(x_cordinate,y_cordinate)
            battery=0  #change it later on
            particles.append([id,[x_cordinate,y_cordinate],[time],[battery]])
            time= time_interval
```

```
            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
```

```
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]-cluste
                    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_dis
            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_AC(
    #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_coor
                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 l
            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)
```

```
            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
```

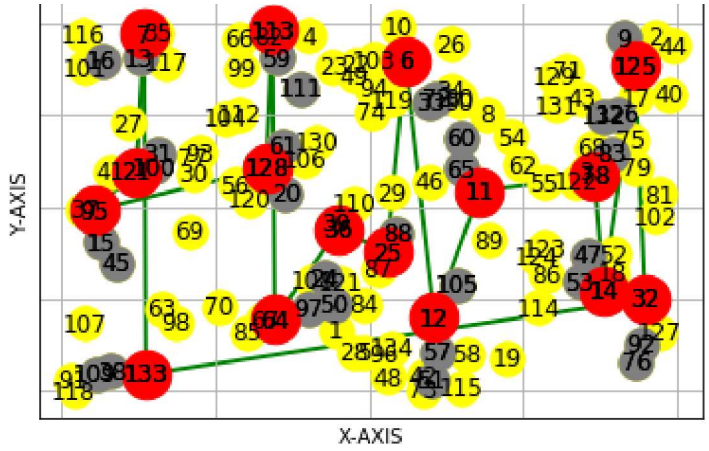
```
        def pick_move(self, pheromone, dist, visited):
            pheromone = np.copy(pheromone)
            pheromone[list(visited)] = 0

            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)
```

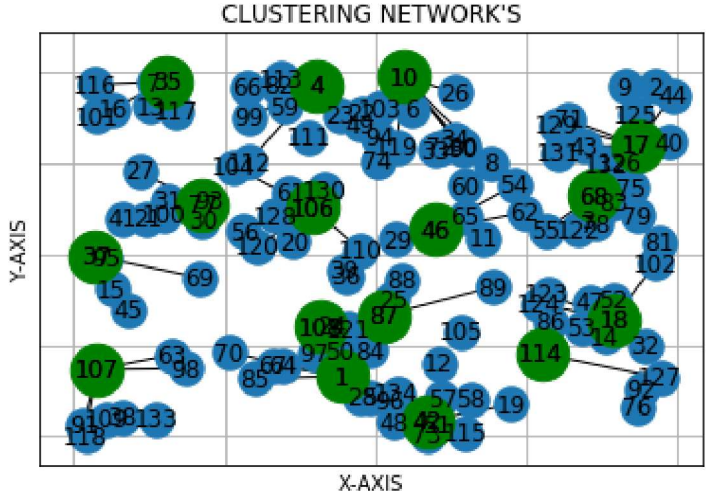
```
# print(simplified_cluster_number)

"""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)
```

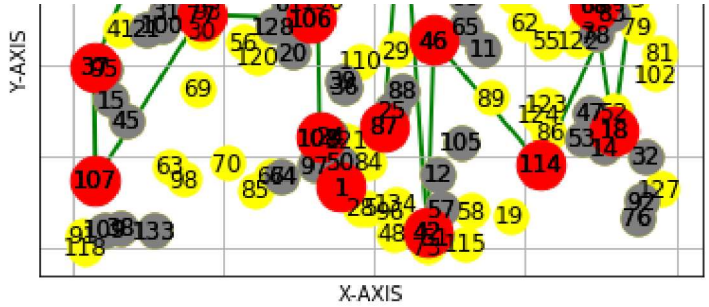
```
    """time.sleep(4)
    file_data.close()

    self.__init__()"""









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




```
[['CLUSTER 1', [107, [38.7, 183.05], [44.298507462686594], [0]]], ['CLUSTER 2', [37, [35.68, 493.91], [15.04477611940299
187.09621188041197
##########
True
```




```
shorted_path: ([(0, 1), (1, 3), (3, 14), (14, 13), (13, 15), (15, 12), (12, 10), (10, 8), (8, 11), (11, 9), (9, 4), (4, 5
4 2728.9471003473177
187.09621188041197
0
TOTAL DISTANCE AFTER ITERATION 4 is 2728.9471003473177
```




```
[['CLUSTER 1', [63, [161.66, 222.96], [25.910447761194046], [0]]], ['CLUSTER 2', [69, [207.28, 435.32], [28.417910447761
275.3999222948329
##########
False
AFTER ALL ITERATION TOTAL DISTANCE IS : 10512.32215310227
```