经济管理学院

课　程　报　告

（复杂网络与社会计算）

题　　目：week4 课程作业

课程教师：赵吉昌

学院/专业：信息管理与信息系统

学生姓名：范春

学　　号：21377061

2024 年 3 月 20 日

北京航空航天大學
BEIHANG UNIVERSITY

作业要求：

1. 阅读本周参考资料。

2. 从 https://users.cs.utah.edu/~lifeifei/SpatialDataset.htm 上下载City of Oldenburg (OL) Road Network数据集，其包括了路网节点的坐标和边权，因此，能够在二维平面上可视化（具体使用nx.draw_networkx_nodes(G, pos, node_size=300, node_color='r', node_shape='o')，其中，pos实际上就是一个字典，其中键为节点，值为坐标x和y）该网络。利用该数据集，参照1中的文献，完成如下任务：

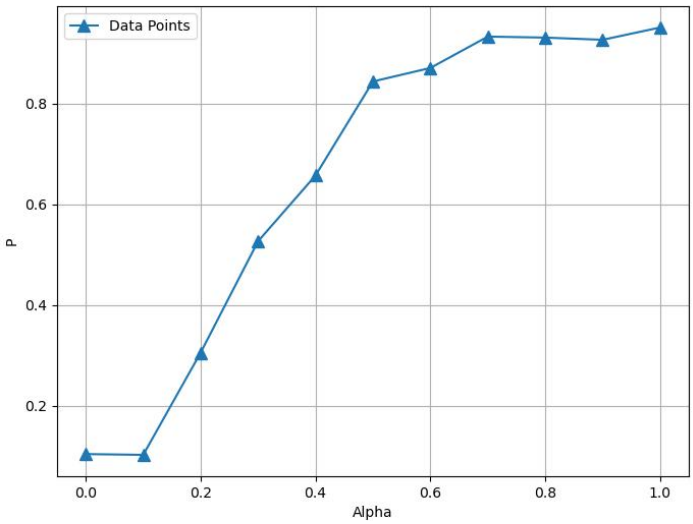
（1）. 实现Motter模型，堵塞网络的空间中心区域（根据平面坐标计算）少量节点，观察不同α下，网络最大连通分量的变化，并讨论是否存在α_c（如级联次数最多的α）。

（2）. 尝试讨论不同α时（尤其是α_c附近时），最大连通分量和第二大连通分量随t的变化形态。

(3). 可视化级联失效（在某个α时，如α_c），用不同的颜色表示初始失效的节点，当步失效的节点，已经失效的节点等，观察传播是否在空间上存在某种模式。

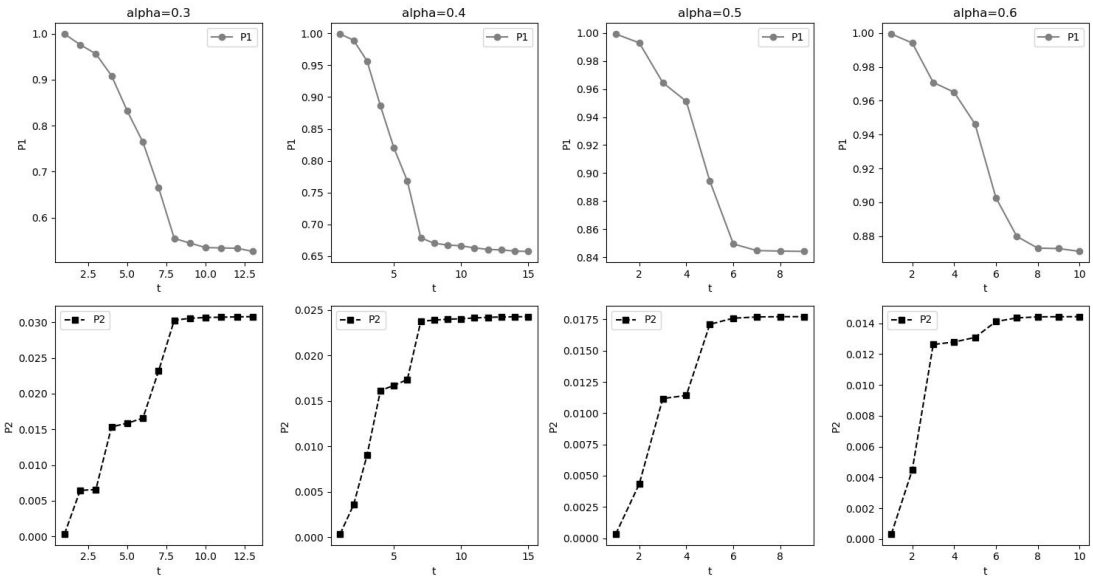
3. 思考级联失效在社会网络中，特别是信息传播过程的潜在应用。比如，信息扩散时，部分用户可能因为收到的信息过载而不再参与后续信息的扩散，这样是否能够描述在信息过载情形下的扩散不畅现象？（引题仅讨论）

任务 2（1）：



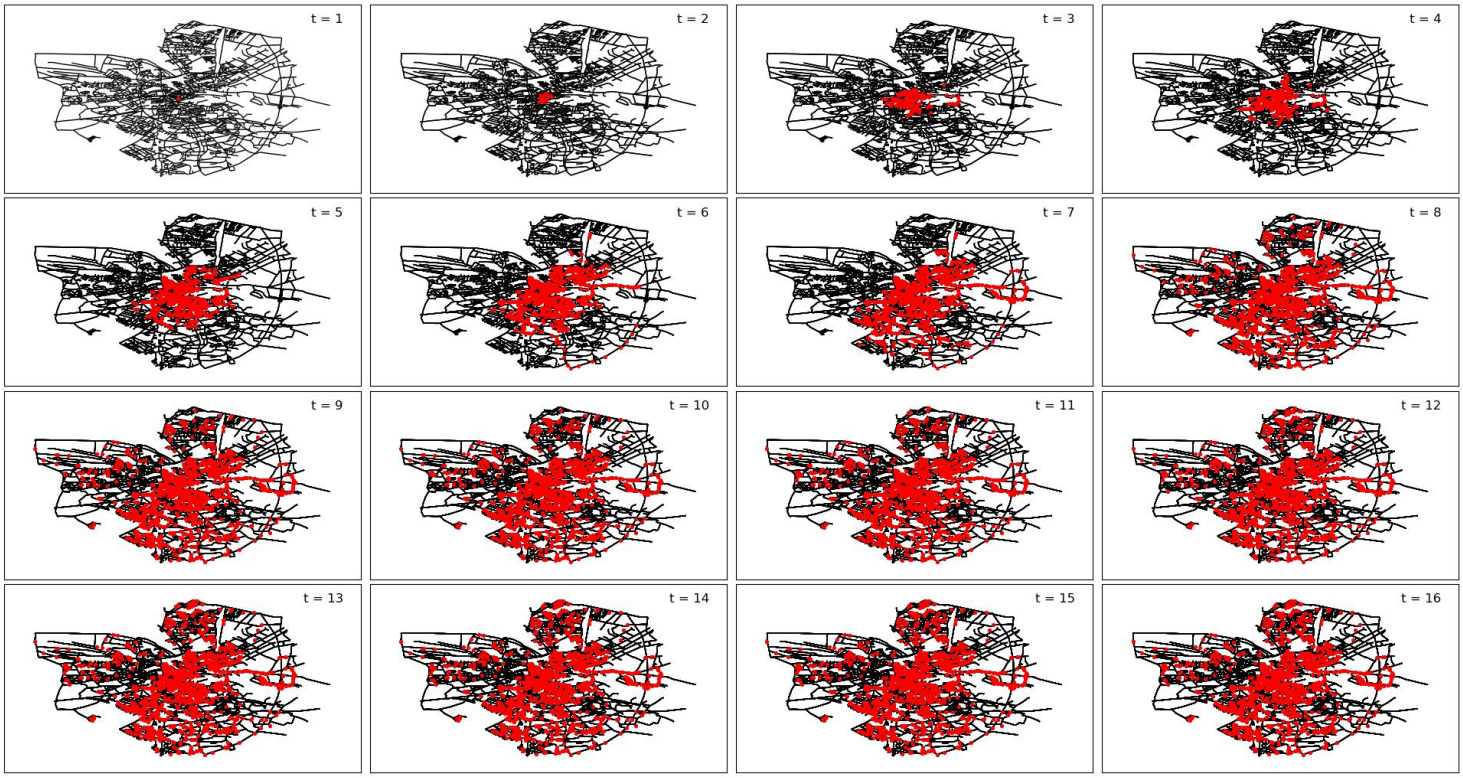
由运行结果可知，级联次数最多的 alpha 为 0.4，因此本次作业将 alpha_c 定位 0.4。由上图可知，随着 alpha 的增大，网络最大连通分量占整个网络的比例也在上升，并且呈现初期平稳、然后猛然陡增、最后趋于平稳的变化趋势。

任务 2（2）：



由于 alpha_c=0.4，所以分别绘制了 alpha 为 0.3、0.4、0.5、0.6 时网络的最大连通分量和第二大连通分量随 t 的变化情况。由上图我们可以发现，最大连通分量随着 t 的增加而减小，但第二大连通分量则相反，并且最大连通分量和第二大连通分量趋于平稳的拐点一致。

任务 2（3）：



上图为 alpha 为 0.4 时随着时间的变化时效节点的分布情况。可以发现传播过程是以最初失效节点为中心向四周扩散，并且扩散的范围速度逐渐增大，最终基本覆盖整个网络。

任务 3：

信息扩散时，部分用户可能因为收到的信息过载而不再参与后续信息的扩散，我认为这在一定程度上可以描述在信息过载情形下的扩散不畅现象。因为当一些用户由于信息过载而不再参与信息扩散时，表明这些点就失效了，由于级联失效，它会导致更多的点失效，从而阻碍信息的传播，虽然这不利于信息的扩散，但为我们进行舆情控制提供了思路。

主要代码如下：

```
import networkx as nx
import math

#返回网络节点的位置信息
def read_node_coordinates():
    nodes = []
    with open("C:\\Users\\范春\\Desktop\\week4\\Nodes.txt",'r') as file:
        for line in file:
            data = line.strip().split()
            node = data[0]
            x = float(data[1])
            y = float(data[2])
            nodes.append((node,x,y))
    return nodes
# 计算节点之间的欧氏距离
def euclidean_distance(node1, node2):
    return math.sqrt((node1[1] - node2[1])**2 + (node1[2] - node2[2])**2)
# 计算网络空间的中心位置
def calculate_center_position(nodes):
    total_x = sum(node[1] for node in nodes)
    total_y = sum(node[2] for node in nodes)
    center_x = total_x / len(nodes)
    center_y = total_y / len(nodes)
    return center_x, center_y
# 计算网络空间中心区域的少量节点
def calculate_center_nodes(num_center_nodes):
    nodes = read_node_coordinates()
    center_x, center_y = calculate_center_position(nodes)

    center_nodes = []
    for node in nodes:
        distance = euclidean_distance((node[0], node[1], node[2]), ('center', center_x, center_y))
        center_nodes.append((node[0], distance))

    center_nodes.sort(key=lambda x: x[1])  # 按距离排序
    center_nodes = center_nodes[:num_center_nodes]  # 选择距离最近的少量节点

    return center_nodes

#生成加权网络
def network(filePath):
    G = nx.Graph()
    with open(filePath,'r') as f:
```

```
        lines = f.readlines()
        for line in lines:
            row = line.strip().split()
            G.add_edge(int(row[1]),int(row[2]),weight=float(row[3]))
    return G
#移除网络空间中心区域的 3 个节点
def remove_center_node(G):
    center_nodes = calculate_center_nodes(3)
    remove_nodes = [int(node[0]) for node in center_nodes]
    G.remove_nodes_from(remove_nodes)
    return remove_nodes
#计算最大连通分支和第二连通分支占网络规模的比值
def component_ratio(G):
    results = []
    ccs = sorted(nx.connected_components(G), key=len,reverse=True)
    largest_cc = ccs[0]
    giant_component_size = len(largest_cc)
    network_size = len(G.nodes())
    giant_component_ratio = giant_component_size / network_size
    results.append(giant_component_ratio)
    if (len(ccs)>1):
        second_cc = ccs[1]
        second_component_size = len(second_cc)/network_size
        results.append(second_component_size)
    else:
        print("该网络连通，无第二大连通分支！")
    return results
def Motter(G):
    L0 = nx.betweenness_centrality(G) #介数
    initial_removed_nodes = remove_center_node(G)
    results = [] #存储最大连通分量随着 alpha 变化而变化的情况

    for i in range(0,11):
        print(i)
        G_copy = G.copy()
        alpha = 0.1*i
        load_capacity = {node:(1+alpha)*load for node,load in L0.items()}

        t = 0
        removed_nodes = [initial_removed_nodes] #存储特定 alpha 下每一轮次删除的节点情况
        ratios_t =[]
        ratios = component_ratio(G_copy)
        ratios.append(t)
        ratios_t.append(ratios)
```

```
        while len(G_copy.nodes())>0:
            t +=1
            Li = nx.betweenness_centrality(G_copy)
            overloaded_nodes = [node for node in G_copy.nodes() if Li[node] > load_capacity[node]]
            if len(overloaded_nodes)==0:
                break
            else:
                G_copy.remove_nodes_from(overloaded_nodes)
                removed_nodes.append(overloaded_nodes)
            ratios = component_ratio(G_copy)
            ratios.append(t)
            ratios_t.append(ratios)

        result = component_ratio(G_copy)
        result.append(alpha)
        results.append(result)
        with open(f"C:\\Users\\范春\\Desktop\\week4\\{alpha:.1f}_remove_nodes.txt","w") as f:
            for node in removed_nodes:
                f.write(f"{node}\n")
         with open(f"C:\\Users\\范春\\Desktop\\week4\\{alpha:.1f}_ratio.txt","w") as f:
            for item in ratios_t:
                if (len(item)==2):
                    f.write(f"{item[1]}\t{item[0]}\n")
                else:
                    f.write(f"{item[2]}\t{item[0]}\t{item[1]}\n")

    with open (f"C:\\Users\\范春\\Desktop\\week4\\result.txt","w") as f:
        for result in results:
            if (len(result)==2):
                f.write(f"{result[1]:.1f}\t{result[0]}\n")
            else:
                f.write(f"{result[2]:.1f}\t{result[0]}\t{result[1]}\n")
def main():
    filePath = "C:\\Users\\范春\\Desktop\\week4\\Edges.txt"
    G = network(filePath)
    Motter(G)
if __name__=="__main__":
    main()
```