\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）。

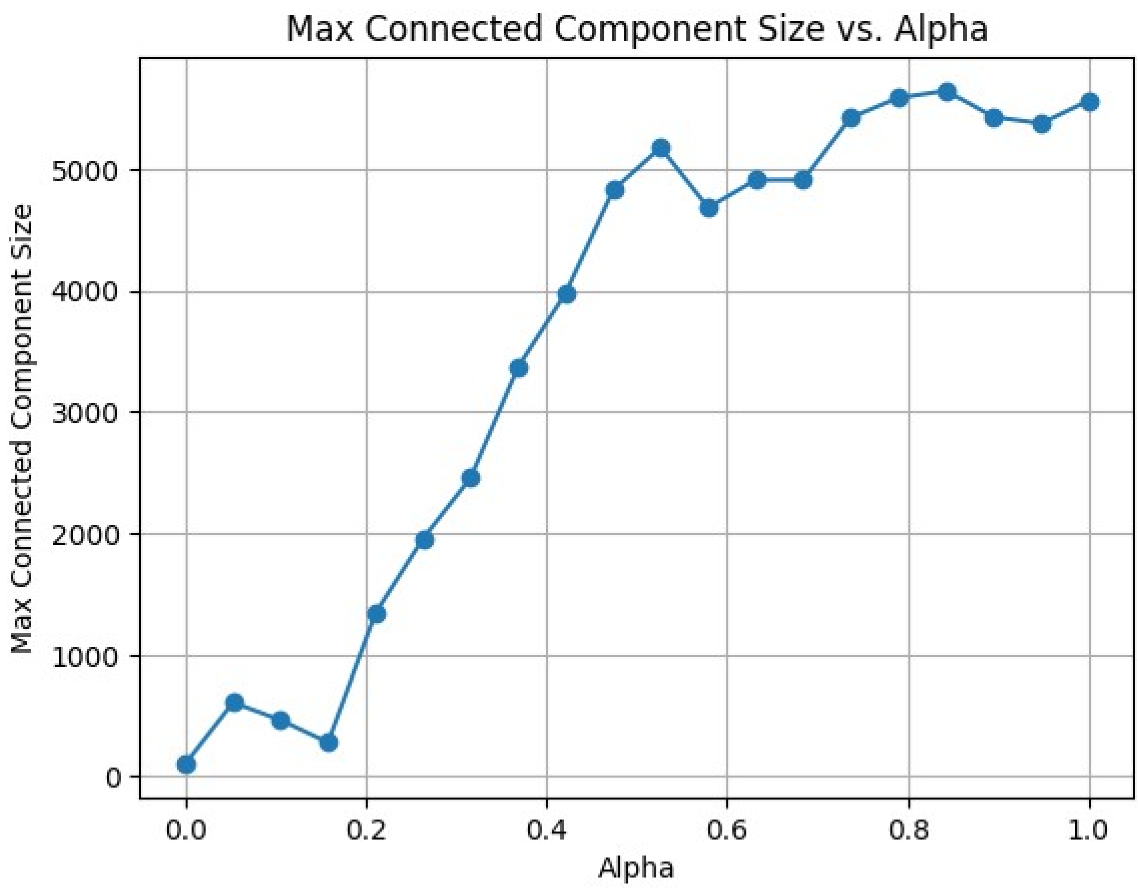
（2）. 尝试讨论不同α时（尤其是α_c附近时），最大连通分量和第二大连通分量随t的变化形态。

(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）

2 (1) Motter模型，查找α_c

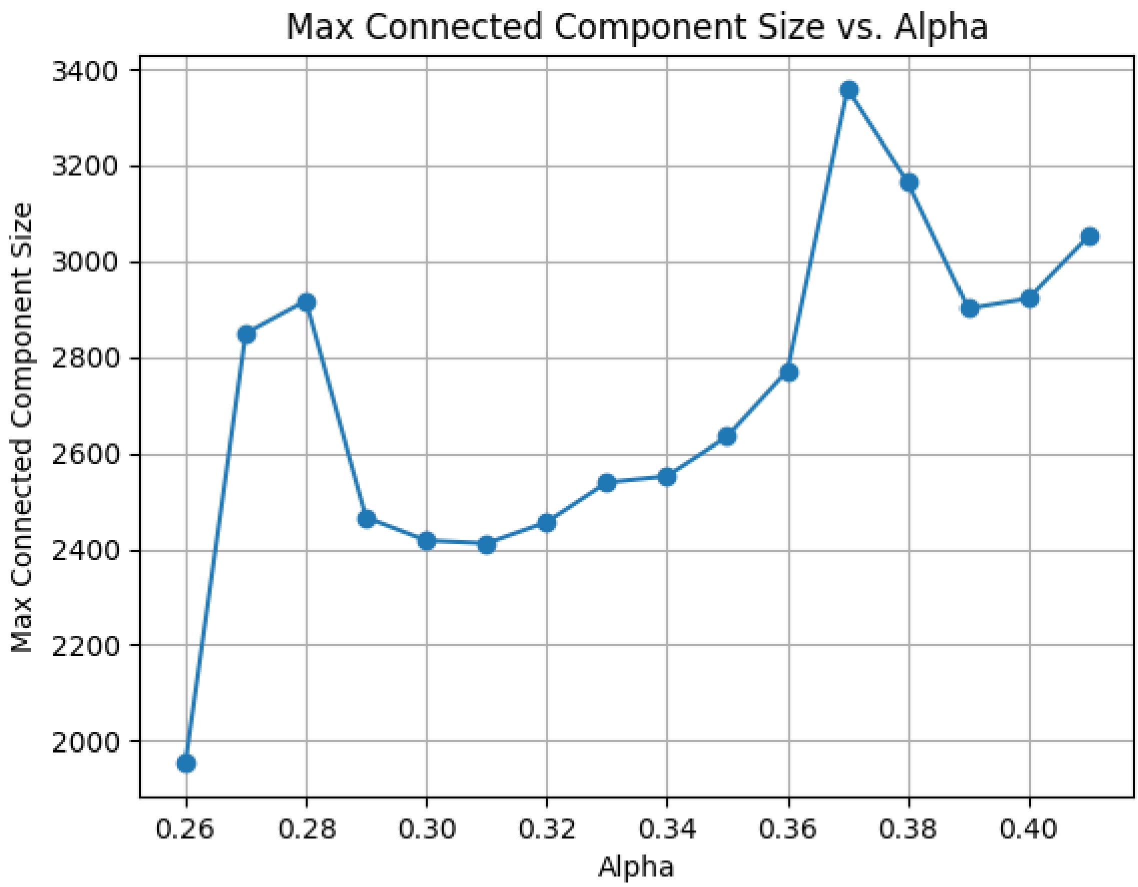
利用平面坐标，通过计算距离其他节点距离之和排序得到空间的中心区域，移除了中心区域的20个节点，查找α在`（0,1）`,步长为0.05时，网络最大连通分量的变化。



随着α的增大，节点的阈值增大，网络最大连通分量总体呈现上升趋势，当α增大到0.5左右时，增加速度放缓。

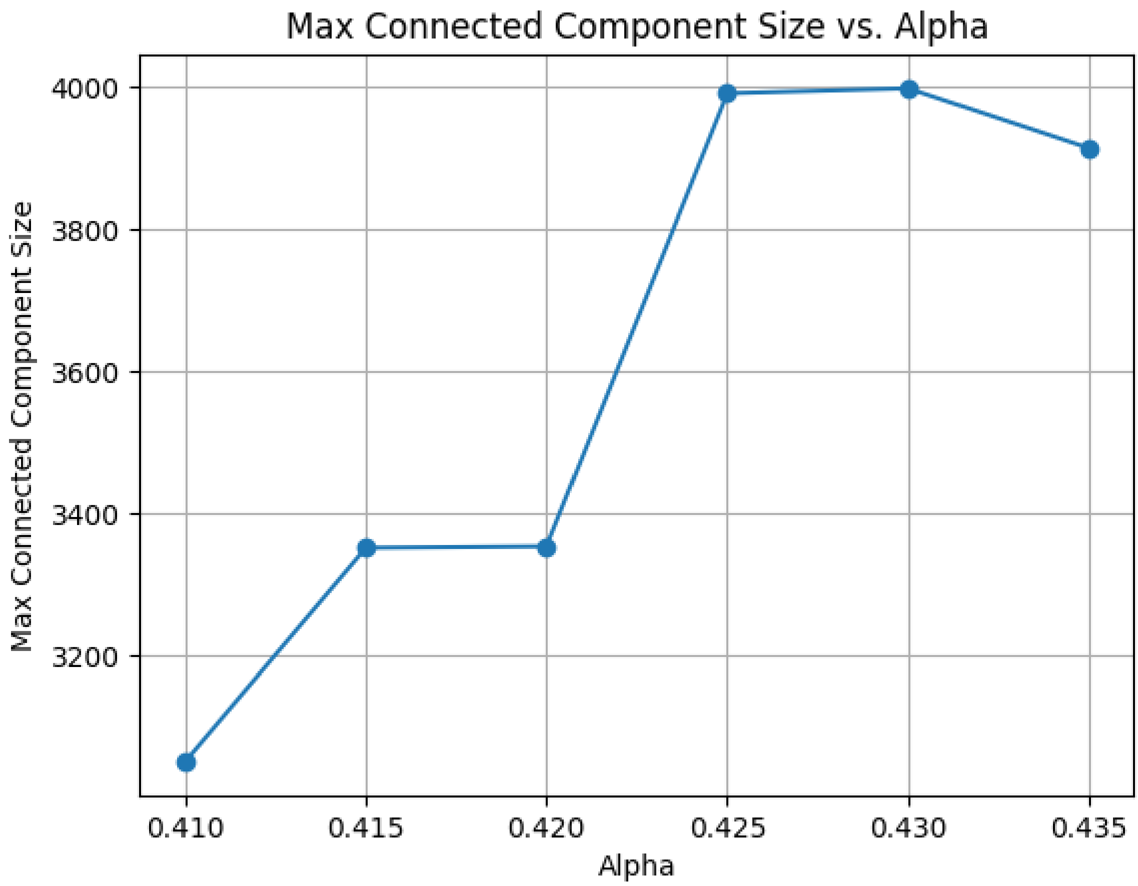
α	$iterations$	α	$iterations$
0	2	0.526	10
0.052	6	0.578	10
0.105	7	0.631	11
0.157	6	0.684	11
0.211	7	0.736	10
0.263	10	0.789	13
0.315	14	0.842	10
0.368	14	0.894	11
0.421	12	0.947	12
0.473	10	1.0	10

根据结果，α_c应该存在于0.26--0.42之间，在此区间上缩小步长为0.01



α	$iterations$	α	$iterations$
0.26	10	0.34	14
0.27	12	0.35	13
0.28	12	0.36	12
0.29	13	0.37	14
0.30	13	0.38	15
0.31	14	0.39	15
0.32	14	0.40	15
0.33	14	0.41	16

α_c在0.39-0.41上增长，在0.41-0.43之间以0.05步长计算



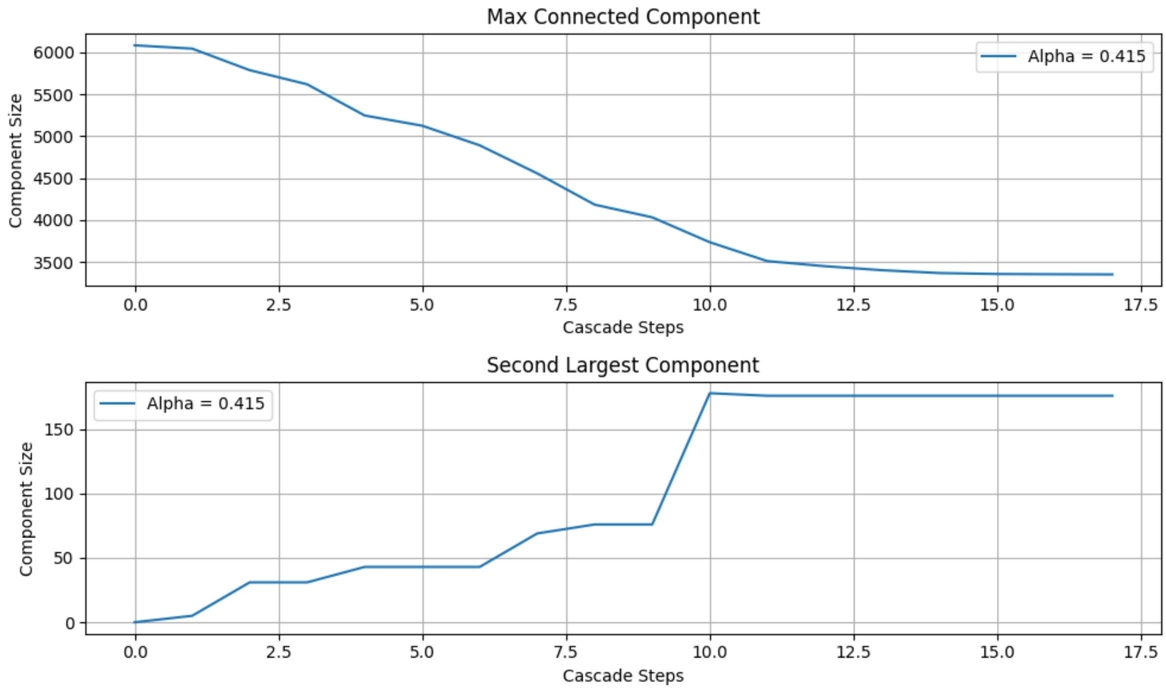
α	$iterations$	α	$iterations$
0.41	16	0.425	12
0.415	18	0.43	11
0.42	19	0.435	12

因此，推测α_c为 `0.42`

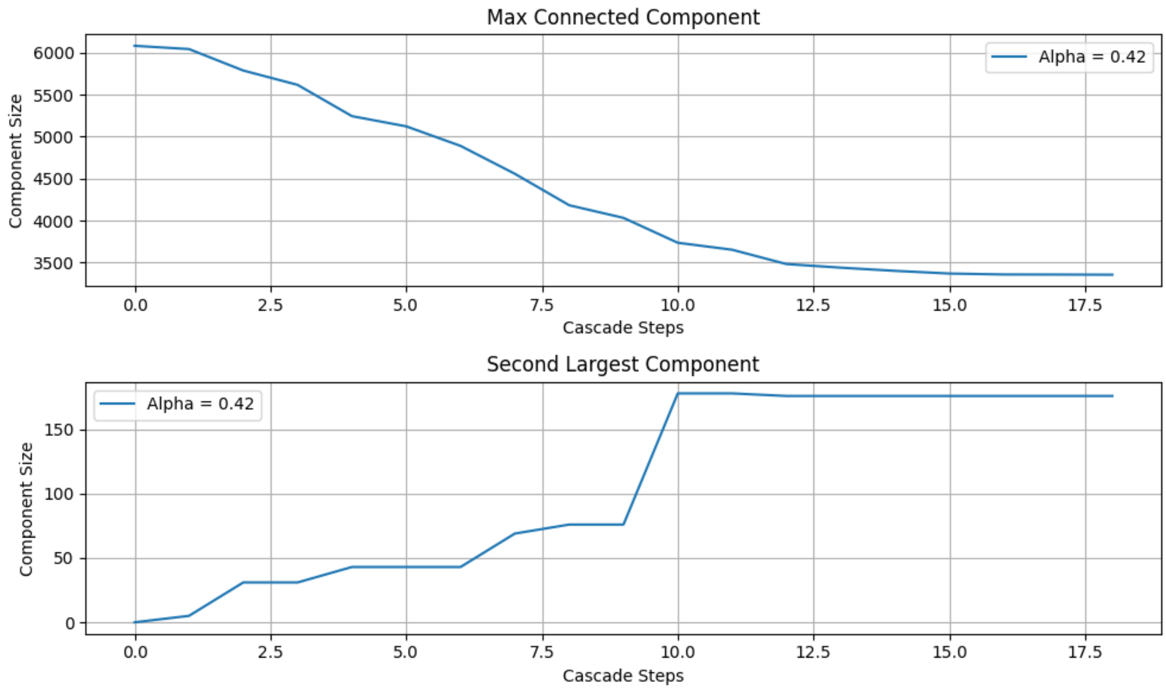
2.(2)最大连通分量和第二大连通分量随t的变化形态

取α分别为0.415，0.42，0.425

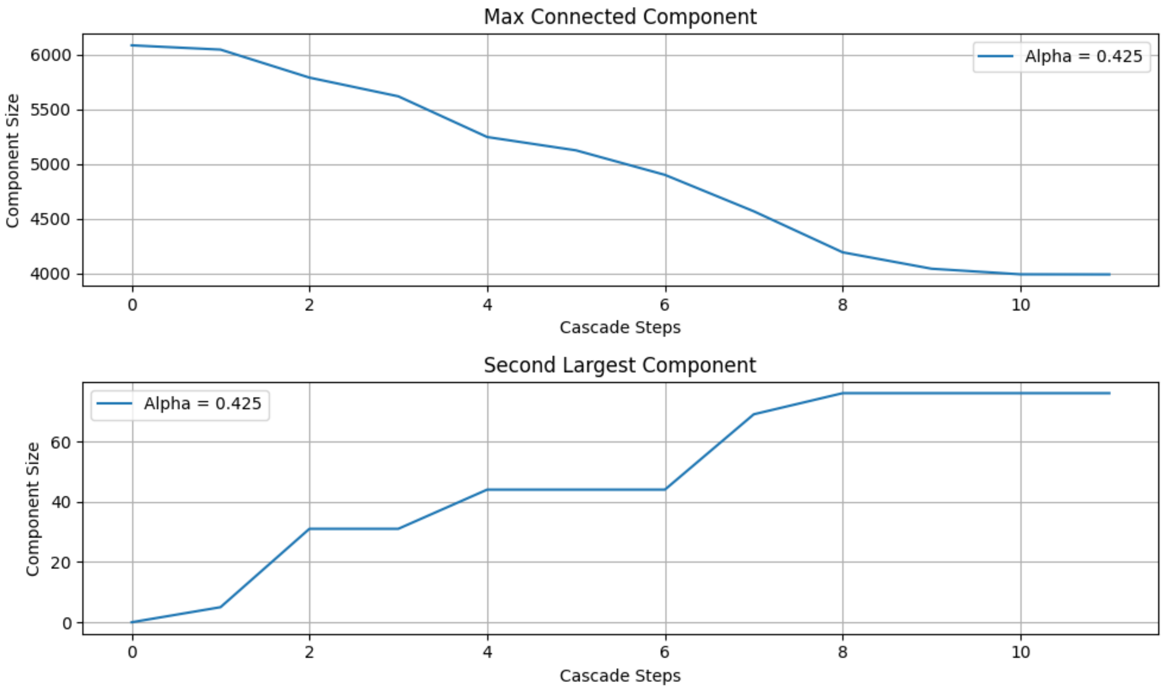
- α=0.415



- α=0.42



- α=0.425



第二大连通分量整体呈上升趋势，最大连通分量整体呈下降趋势。第二大连通分量均在某一个步骤时出现了明显上升，此时的最大连通分量接近于最终值，之后基本维持不变。

2.（3）可视化级联失效

当α=0.42时，使用红色表示初始失效的节点，即中心删去的20个节点，黄色表示已经失效的节点，蓝色表示当前步失效的节点，灰色表示现存的节点。

传播在空间上呈现逐层向外扩散的现象，由里层逐渐向外围扩展，中心区域节点较为密集，失效的节点较多。

3.信息传播应用

通过级联失效，也可以判断网络中哪些节点起着关键的信息传播作用，如果该节点失效，则整个网络会受到巨大影响。比如社交网络中意见领袖，对于特定的网络信息传播起着重大作用，可以模拟失效过程发现意见领袖。

```
# 读取节点数据
def read_nodes_data(file_path):
    nodes_data = {}
    with open(file_path, 'r') as f:
        for line in f:
            node_id, x, y = line.split()
            nodes_data[int(node_id)] = (float(x), float(y))
    return nodes_data

# 读取边数据
def read_edges_data(file_path):
    edges_data = []
    with open(file_path, 'r') as f:
        for line in f:
            edge_id, source, target, weight = line.split()
            edges_data.append((int(edge_id), int(source), int(target),
float(weight)))
    return edges_data

# 测试读取数据函数
```

```python
nodes_file = "/update_documents/nodes.txt"
edges_file = "/update_documents/edges.txt"
nodes_data = read_nodes_data(nodes_file)
edges_data = read_edges_data(edges_file)

import networkx as nx

# 计算节点的初始负载
def initial_load(nodes_data, edges_data):
    G = nx.Graph()
    G.add_weighted_edges_from([(source, target, weight) for _, source, target, weight in edges_data])
    initial_loads = nx.betweenness_centrality(G)
    return {node: initial_loads[node] for node in nodes_data}

# 计算节点的负载容量
def capacity(nodes_data, initial_loads, alpha):
    capacities = {}
    for node, load in initial_loads.items():
        capacities[node] = (1 + alpha) * load
    return capacities

import numpy as np
from scipy.spatial import distance

# 计算节点之间的欧几里得距离
def calculate_distances(nodes_data):
    nodes_coords = np.array([coord for _, coord in nodes_data.items()])
    distances = distance.cdist(nodes_coords, nodes_coords, 'euclidean')
    return distances

# 根据平面坐标计算空间中心区域
def calculate_center_region(nodes_data):
    distances = calculate_distances(nodes_data)
    center_region = []
    MOD = 10000
    for i, node_i in nodes_data.items():
        count = 0
        for j, node_j in nodes_data.items():
            count += distances[i,j]
        count /= MOD
        center_region.append([count,i])
    return center_region

# 实现Motter模型，首先移除中心区域的节点，然后根据级联反应移除节点
def motter_model_with_cascading_and_center_removal(nodes_data, edges_data, alpha):
    # 计算初始负载
    initial_loads = initial_load(nodes_data, edges_data)
    # 计算负载容量
    capacities = capacity(nodes_data, initial_loads, alpha)
    # 计算空间中心区域
    center_region = calculate_center_region(nodes_data)
    # 移除中心区域的20个节点
    nodes_to_remove = sorted(center_region,key=lambda x:x[0])[:20]
    # 构建图
    G = nx.Graph()
```

```
    G.add_weighted_edges_from([(source, target, weight) for _, source, target, weight in edges_data])
    center_node = [node[1] for node in nodes_to_remove]
    G.remove_nodes_from(center_node)
    # 迭代次数
    iteration = 0
    while True:
        iteration += 1
        # 计算剩余节点的新负载
        current_loads = nx.betweenness_centrality(G)
        # 移除负载超过负载容量的节点
        overloaded_nodes = [node for node in G.nodes if current_loads[node] > capacities[node]]
        if not overloaded_nodes:
            break
        if len(overloaded_nodes) == len(G.nodes):
            break
        # 移除策略：根据级联反应移除节点
        nodes_to_remove = []
        for node in overloaded_nodes:
            nodes_to_remove.append(node)
        for node_to_remove in nodes_to_remove:
            G.remove_node(node_to_remove)
    return G, iteration

import matplotlib.pyplot as plt

# 模拟不同α值下的网络最大连通分量的变化
def simulate_alpha_range(nodes_data, edges_data, alpha_range):
    max_connected_components = []
    for alpha in alpha_range:
        G, iterations = motter_model_with_cascading_and_center_removal(nodes_data, edges_data, alpha)
        max_connected_component = max(len(component) for component in nx.connected_components(G))
        max_connected_components.append(max_connected_component)
        print("Alpha:", alpha, "Max Connected Component Size:", max_connected_component, "Iterations:", iterations)
    return max_connected_components

# 设置α值范围
alpha_range = np.linspace(0, 1, 20)
# 模拟不同α值下的网络最大连通分量的变化
max_connected_components = simulate_alpha_range(nodes_data, edges_data, alpha_range)

# 绘制图表
plt.plot(alpha_range, max_connected_components, marker='o')
plt.title('Max Connected Component Size vs. Alpha')
plt.xlabel('Alpha')
plt.ylabel('Max Connected Component Size')
plt.grid(True)
plt.show()



# 模拟最大连通分量和第二大连通分量随级联步骤的变化
def simulate_cascading(nodes_data, edges_data, alpha):
```

```python
    # 计算初始负载
    initial_loads = initial_load(nodes_data, edges_data)
    # 计算负载容量
    capacities = capacity(nodes_data, initial_loads, alpha)
    # 计算空间中心区域
    center_region = calculate_center_region(nodes_data)
    # 移除中心区域的20个节点
    nodes_to_remove = sorted(center_region,key=lambda x:x[0])[:20]
    # 构建图
    G = nx.Graph()
    G.add_weighted_edges_from([(source, target, weight) for _, source, target, weight in edges_data])
    center_node = [node[1] for node in nodes_to_remove]
    G.remove_nodes_from(center_node)
    max_connected_component_sizes = []
    second_largest_component_sizes = []
    sizes = sorted([len(component) for component in nx.connected_components(G)], reverse=True)
    max_connected_component_sizes.append(sizes[0])
    second_largest_component_sizes.append(sizes[1] if len(sizes) > 1 else 0)
    iteration = 0
    while True:
        iteration += 1
        overloaded_nodes = []
        current_loads = nx.betweenness_centrality(G)
        for node in G.nodes:
            if current_loads[node] > capacities[node]:
                overloaded_nodes.append(node)
        if not overloaded_nodes:
            break
        if len(overloaded_nodes) == len(G.nodes):
            break
        for node_to_remove in overloaded_nodes:
            G.remove_node(node_to_remove)
        sizes = sorted([len(component) for component in nx.connected_components(G)], reverse=True)
        max_connected_component_sizes.append(sizes[0])
        second_largest_component_sizes.append(sizes[1] if len(sizes) > 1 else 0)
    return max_connected_component_sizes, second_largest_component_sizes,iteration

# 设置α值
alphas = [0.42, 0.415, 0.425]
plt.figure(figsize=(10, 6))
for i, alpha in enumerate(alphas):
    max_connected_component_sizes, second_largest_component_sizes,iteration = simulate_cascading(nodes_data, edges_data, alpha)
    # 设置级联步骤范围
    cascade_steps = range(0, iteration)
    plt.subplot(2, 1, 1)
    plt.plot(cascade_steps, max_connected_component_sizes, label='Alpha = {}'.format(alpha))
    plt.title('Max Connected Component')
    plt.xlabel('Cascade Steps')
    plt.ylabel('Component Size')
    plt.legend()
    plt.grid(True)

```

```python
    plt.subplot(2, 1, 2)
    plt.plot(cascade_steps, second_largest_component_sizes, label='Alpha = {}'.format(alpha))
    plt.title('Second Largest Component')
    plt.xlabel('Cascade Steps')
    plt.ylabel('Component Size')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.savefig('/home/ubuntu/generate_documents/alpha_{}.png'.format(alpha))
    plt.clf()



def vision_model_state(nodes_data, edges_data,alpha,):
    # 设置初始失效的节点、当前步失效的节点和已经失效的节点的颜色
    initial_failure_color = 'red'
    current_failure_color = 'blue'
    failed_color = 'gray'
    # 设置节点状态对应的颜色
    colors = {'initial_failure': 'red', 'current_failure': 'blue', 'failed': 'yellow', 'not_failed': 'gray'}

    # 获取节点坐标
    node_coordinates = np.array([coord[1:] for coord in nodes_data.values()])

    # 初始化节点状态字典，初始状态为未失效
    node_status = {node: 'not_failed' for node in nodes_data.keys()}
    # 计算初始负载
    initial_loads = initial_load(nodes_data, edges_data)
    # 计算负载容量
    capacities = capacity(nodes_data, initial_loads, alpha)
    # 计算空间中心区域
    center_region = calculate_center_region(nodes_data, center_threshold)
    # 移除中心区域的20个节点
    nodes_to_remove = sorted(center_region,key=lambda x:x[0])[:20]
    # 构建图
    initial_G = nx.Graph()
    initial_G.add_weighted_edges_from([(source, target, weight) for _, source, target, weight in edges_data])
    G = initial_G.copy()
    center_node = [node[1] for node in nodes_to_remove]
    G.remove_nodes_from(center_node)
    # 模拟级联失效过程，并可视化
    initial_failure_nodes = center_node
    for i in range(1, 20):
        overloaded_nodes = []
        current_failure_nodes = []
        current_loads = nx.betweenness_centrality(G)
        for node in G.nodes:
            if current_loads[node] > capacities[node]:
                overloaded_nodes.append(node)
        if not overloaded_nodes:
            break
        for node_to_remove in overloaded_nodes:
            node_status[node_to_remove] = 'failed'
            G.remove_node(node_to_remove)
```

```python
        if i == 1:
            initial_failure_nodes = overloaded_nodes
        else:
            current_failure_nodes = overloaded_nodes
        # 可视化
        plt.figure(figsize=(8, 6))
        pos = {node:pos for node,pos in nodes_data.items()}
        nx.draw(initial_G, pos=pos, node_color=[colors['initial_failure'] if node in initial_failure_nodes else colors['current_failure'] if node in current_failure_nodes else colors['failed'] if node_status[node] == 'failed' else colors['not_failed'] for node in initial_G.nodes()], with_labels=False, node_size=0.5)

        plt.title('Cascade Step {}'.format(i))
        plt.grid(False)
        plt.axis('off')
        plt.savefig('/generate_documents/cascade_step/cascade_step_{}.png'.format(i))
        plt.close()

    # 可视化最终状态
    plt.figure(figsize=(8, 6))
    pos = {node:pos for node,pos in nodes_data.items()}
    nx.draw(initial_G, pos=pos, node_color=[colors['initial_failure'] if node in initial_failure_nodes else colors['failed'] if node_status[node] == 'failed' else colors['not_failed'] for node in initial_G.nodes()], with_labels=False, node_size=0.5)

    plt.title('Final State')
    plt.grid(False)
    plt.axis('off')
    plt.savefig('/generate_documents/cascade_step/final_state.png')
    plt.close()

vision_model_state(nodes_data, edges_data,0.42,)

```