Project

1 网络构建

我选择生成了一个无标度网络，无标度网络是一种具有"幂律"度分布的网络，即少数节点拥有大量的连接，而大多数节点则只有少数连接，这种网络模拟了许多现实世界中的复杂网络，如社交网络、互联网、引用网络等。

其构建的步骤为：

1. 初始化网络：先构建一个小的完全图；
2. 加入节点：每次新增一个节点，这个节点会与网络中已有的节点连接；
3. 优先连接：新的节点更有可能连接到度更高的节点上去；

节点：代表个体；

边：代表个体之间的关系或者连接；

```
def create_scale_free_network(n, m):

    edges = []  # 网络中的边列表
    nodes = list(range(m))  # 初始节点集

    # 初始完全图
    for i in range(m):
        for j in range(i + 1, m):
            edges.append((i, j))

    # 逐步添加节点
    for new_node in range(m, n):
        target_nodes = set()
        while len(target_nodes) < m:
            # 根据度数优先选择已有节点
            potential_target = random.choices(nodes, weights=[degree_count(n,
edges, node) for node in nodes])[0]
            target_nodes.add(potential_target)

        # 将新节点连接到目标节点
        for target in target_nodes:
            edges.append((new_node, target))
        nodes.append(new_node)

    return edges
```

2 进行网络分析

计算网络的节点度分布

把图的每条边都遍历一遍即可

```
def degree_distribution(n, edges):
```

```
    degree_dict = {i: 0 for i in range(n)}
    for edge in edges:
        degree_dict[edge[0]] += 1
        degree_dict[edge[1]] += 1

    degree_freq = {}
    for degree in degree_dict.values():
        if degree not in degree_freq:
            degree_freq[degree] = 0
        degree_freq[degree] += 1

    return degree_freq
```

计算平均最短路径长度

使用广度优先搜索找到每对节点之间的最短路径，然后求和即可得到平均最短路径长度。

```
def bfs_shortest_path(n, edges, start):

    distances = {i: float('inf') for i in range(n)}
    distances[start] = 0
    queue = deque([start])

    while queue:
        node = queue.popleft()
        for edge in edges:
            if node in edge:
                neighbor = edge[0] if edge[1] == node else edge[1]
                if distances[neighbor] == float('inf'):
                    distances[neighbor] = distances[node] + 1
                    queue.append(neighbor)
    return distances

def average_shortest_path_length(n, edges):

    total_path_length = 0
    num_pairs = 0

    for node in range(n):
        distances = bfs_shortest_path(n, edges, node)
        for target, distance in distances.items():
            if distance != float('inf') and target != node:
                total_path_length += distance
                num_pairs += 1

    return total_path_length / num_pairs if num_pairs > 0 else float('inf')
```

计算聚类系数

步骤：

1. 找出当前节点的邻居；
2. 计算邻居之间的边数；
3. 最后计算聚类系数即可；

```
def clustering_coefficient(n, edges):
    clustering_coeffs = []
```

```
    for node in range(n):

        neighbors = set()
        for edge in edges:
            if edge[0] == node:
                neighbors.add(edge[1])
            elif edge[1] == node:
                neighbors.add(edge[0])


        if len(neighbors) < 2:
            clustering_coeffs.append(0)
            continue

        links_between_neighbors = 0
        for neighbor1 in neighbors:
            for neighbor2 in neighbors:
                if neighbor1 != neighbor2 and (neighbor1, neighbor2) in edges or
(neighbor2, neighbor1) in edges:
                    links_between_neighbors += 1


        clustering_coeffs.append(links_between_neighbors / (len(neighbors) *
(len(neighbors) - 1)))

    return sum(clustering_coeffs) / n
```

进一步将其可视化：

1. 计算每个节点的局部聚类函数；
2. 计算聚类系数；
3. 然后绘制其散点图和条形图；

```
def calculate_local_clustering_coefficients(n, edges):
    local_clustering_coeffs = []

    for node in range(n):
        neighbors = set()
        for edge in edges:
            if edge[0] == node:
                neighbors.add(edge[1])
            elif edge[1] == node:
                neighbors.add(edge[0])

        if len(neighbors) < 2:
            local_clustering_coeffs.append(0)
            continue

        links_between_neighbors = 0
        neighbors_list = list(neighbors)
        for i in range(len(neighbors_list)):
            for j in range(i + 1, len(neighbors_list)):
                if (neighbors_list[i], neighbors_list[j]) in edges or
(neighbors_list[j], neighbors_list[i]) in edges:
                    links_between_neighbors += 1
```

```
        local_clustering_coeffs.append(links_between_neighbors / (len(neighbors)
* (len(neighbors) - 1) / 2))

    return local_clustering_coeffs

def plot_local_clustering_coefficients(local_clustering_coefficients):
    nodes = list(range(len(local_clustering_coefficients)))
    clustering_values = local_clustering_coefficients

    plt.figure(figsize=(8, 6))
    plt.scatter(nodes, clustering_values, color='purple', alpha=0.7)
    plt.title("Local Clustering Coefficients of Nodes")
    plt.xlabel("Node")
    plt.ylabel("Local Clustering Coefficient")
    plt.ylim(0, 1)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

def plot_average_clustering_coefficient(avg_clustering_coefficient):

    plt.figure(figsize=(6, 4))
    plt.bar(['Network'], [avg_clustering_coefficient], color='skyblue',
edgecolor='black')
    plt.title("Average Clustering Coefficient of the Network")
    plt.ylabel("Clustering Coefficient")
    plt.ylim(0, 1)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```

3 模拟动态行为

模拟了两种攻击：随机攻击和故意攻击。

模拟网络的随机攻击

随机攻击通过随机打乱节点顺序，然后移除节点和相关的边。

```
def simulate_random_attack(n, edges):

    nodes = list(range(n))
    random.shuffle(nodes)
    sizes = []

    for node in nodes:
        edges = [(u, v) for u, v in edges if u != node and v != node]
        if not edges:
            break
        largest_cc_size = get_largest_connected_component_size(n, edges)
        sizes.append(largest_cc_size)

    return sizes
```

模拟网络的故意攻击

故意攻击先将度数从大到小排序，然后从头开始移除对应的节点和相关的边

```
def simulate_targeted_attack(n, edges):
    nodes = list(range(n))
    degrees = {node: degree_count(n, edges, node) for node in nodes}
    nodes_sorted_by_degree = sorted(nodes, key=lambda x: degrees[x],
reverse=True)
    sizes = []

    for node in nodes_sorted_by_degree:
        edges = [(u, v) for u, v in edges if u != node and v != node]
        if not edges:
            break
        largest_cc_size = get_largest_connected_component_size(n, edges)
        sizes.append(largest_cc_size)

    return sizes
```

计算最大连通子图的大小

我们通过计算出当前图中最大连通子图的大小来衡量网络被攻击之后的连通性和造成的影响，通过建立一个栈来进行深度优先搜索，每次记录走过最长的路径并记录在`largest_size`中，然后暴力遍历每个节点即可得出最大联通子图的大小。

```
def get_largest_connected_component_size(n, edges):
    visited = [False] * n

    def dfs(node):
        stack = [node]
        size = 0
        while stack:
            current = stack.pop()
            if not visited[current]:
                visited[current] = True
                size += 1
                for edge in edges:
                    if edge[0] == current and not visited[edge[1]]:
                        stack.append(edge[1])
                    elif edge[1] == current and not visited[edge[0]]:
                        stack.append(edge[0])
        return size

    largest_size = 0
    for node in range(n):
        if not visited[node]:
            component_size = dfs(node)
            if component_size > largest_size:
                largest_size = component_size
    return largest_size
```

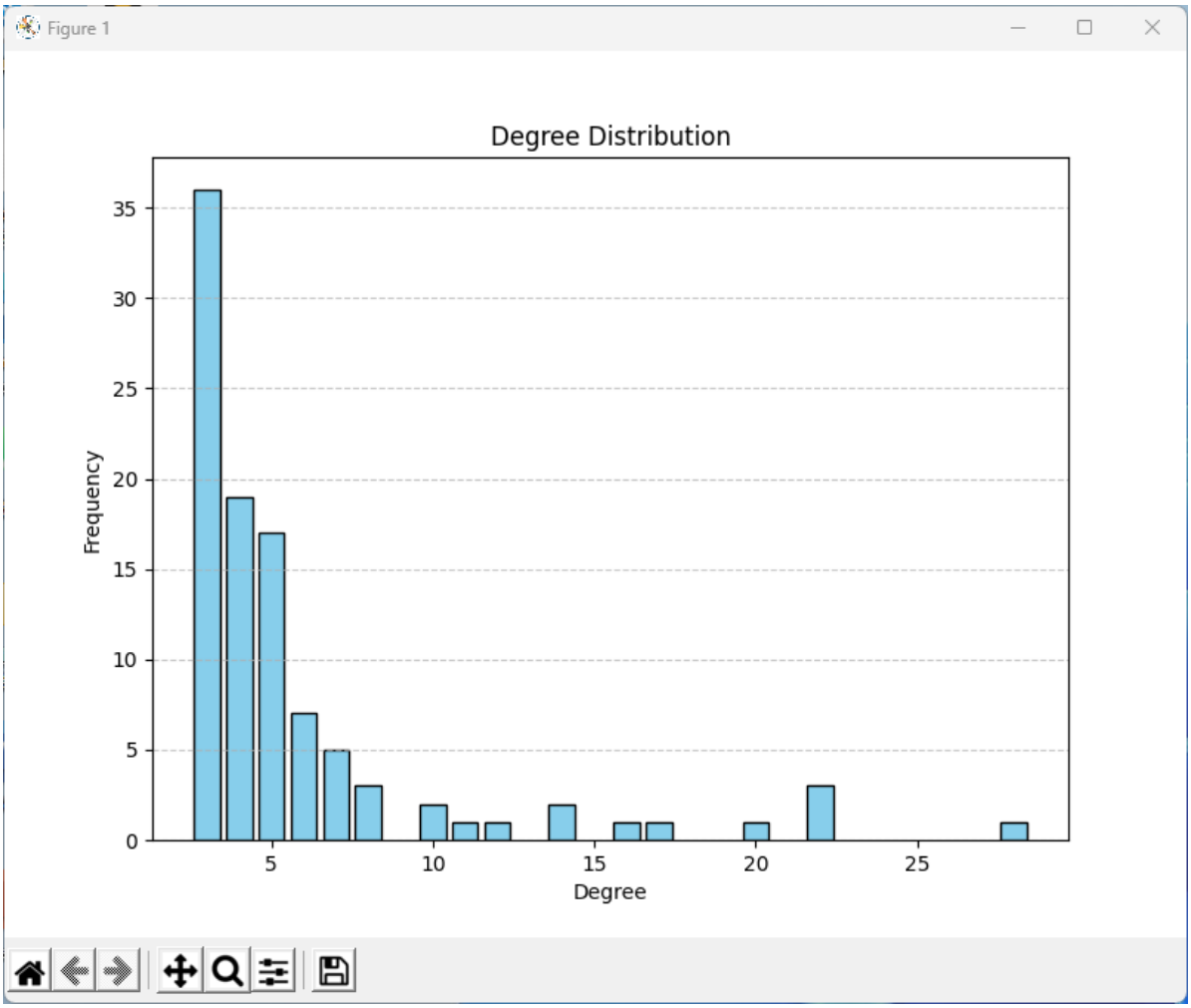
4 计算构建图中的节点核心度

计算节点的核心度将其按度的大小排列即可，先移除度数小的节点，然后更新核心度，在移除当前节点并更新其邻居的度数，记得每次便利更新队列。

```

```

```python
def calculate_core_number(n, edges):
    degree = {i: 0 for i in range(n)}
    for u, v in edges:
        degree[u] += 1
        degree[v] += 1

    core_number = {i: 0 for i in range(n)}

    nodes = sorted(degree.keys(), key=lambda x: degree[x])

    while nodes:
        node = nodes.pop(0)
        current_degree = degree[node]

        core_number[node] = current_degree

        for u, v in edges:
            if u == node or v == node:
                neighbor = v if u == node else u
                if degree[neighbor] > current_degree:
                    degree[neighbor] -= 1

        nodes = sorted(nodes, key=lambda x: degree[x])

    return core_number
```

Result



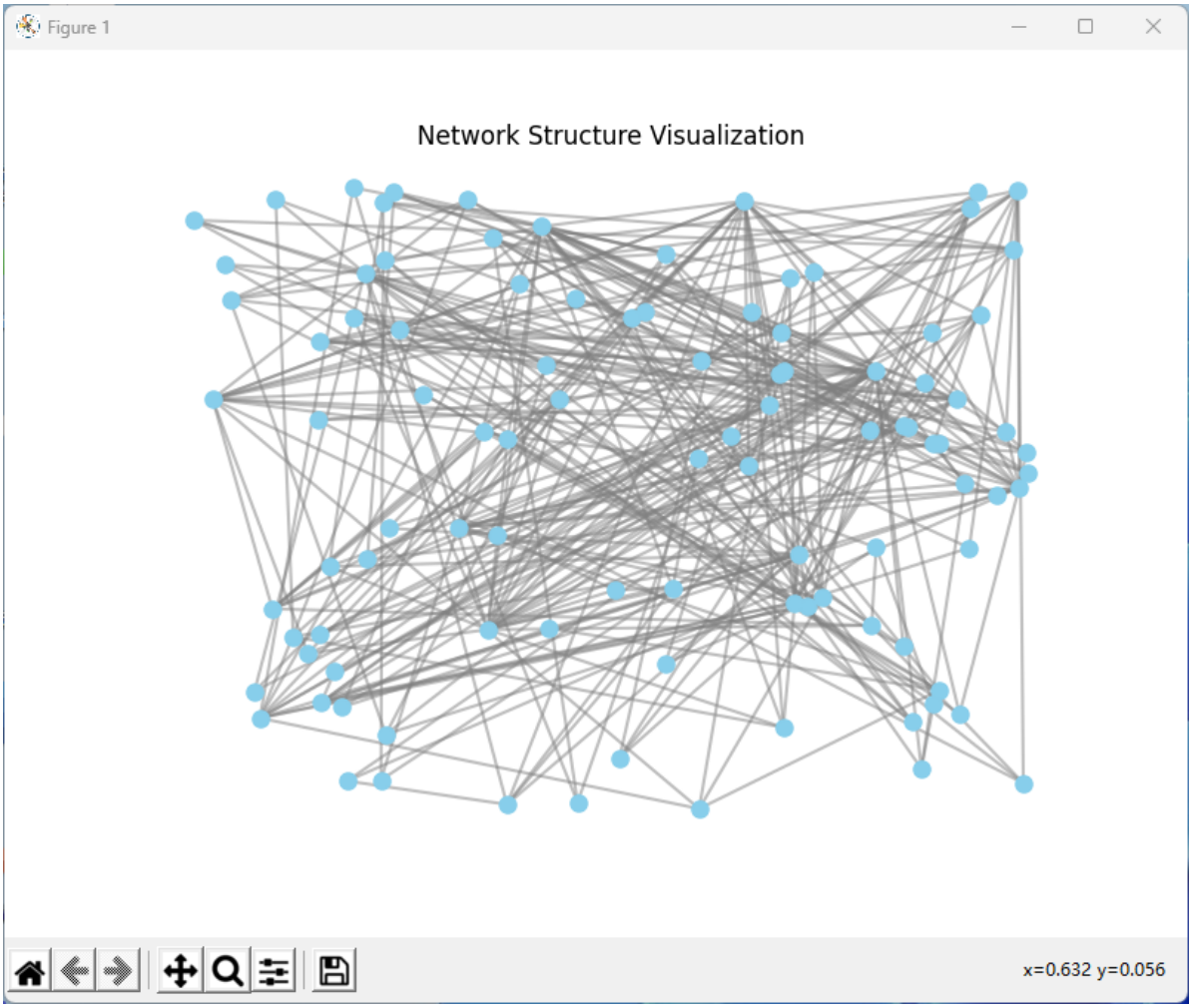


Figure 1
Network Structure Visualization
x=0.632 y=0.056

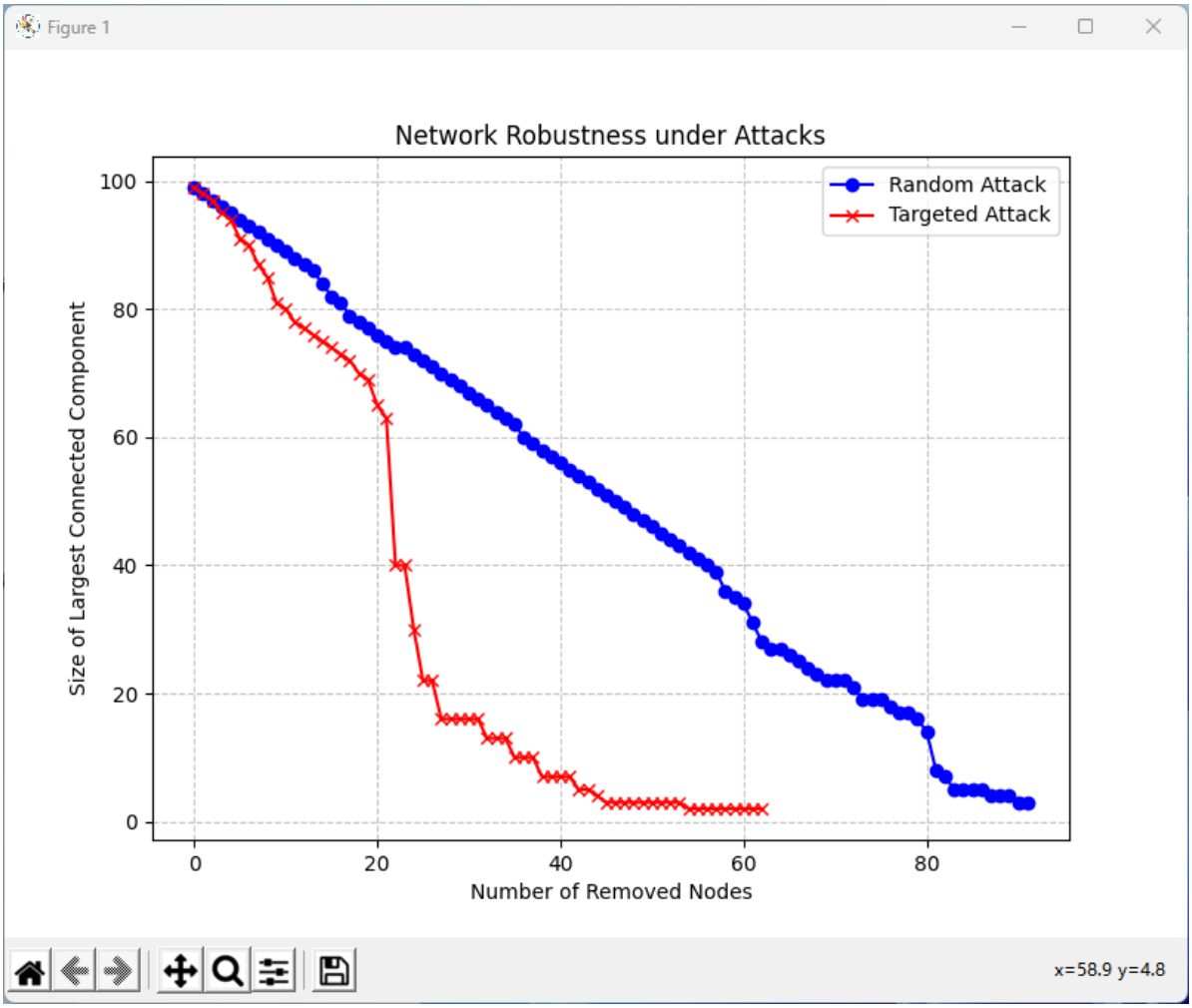


Figure 1
Network Robustness under Attacks
Random Attack
Targeted Attack
Size of Largest Connected Component
Number of Removed Nodes
0
20
40
60
80
100
x=58.9 y=4.8

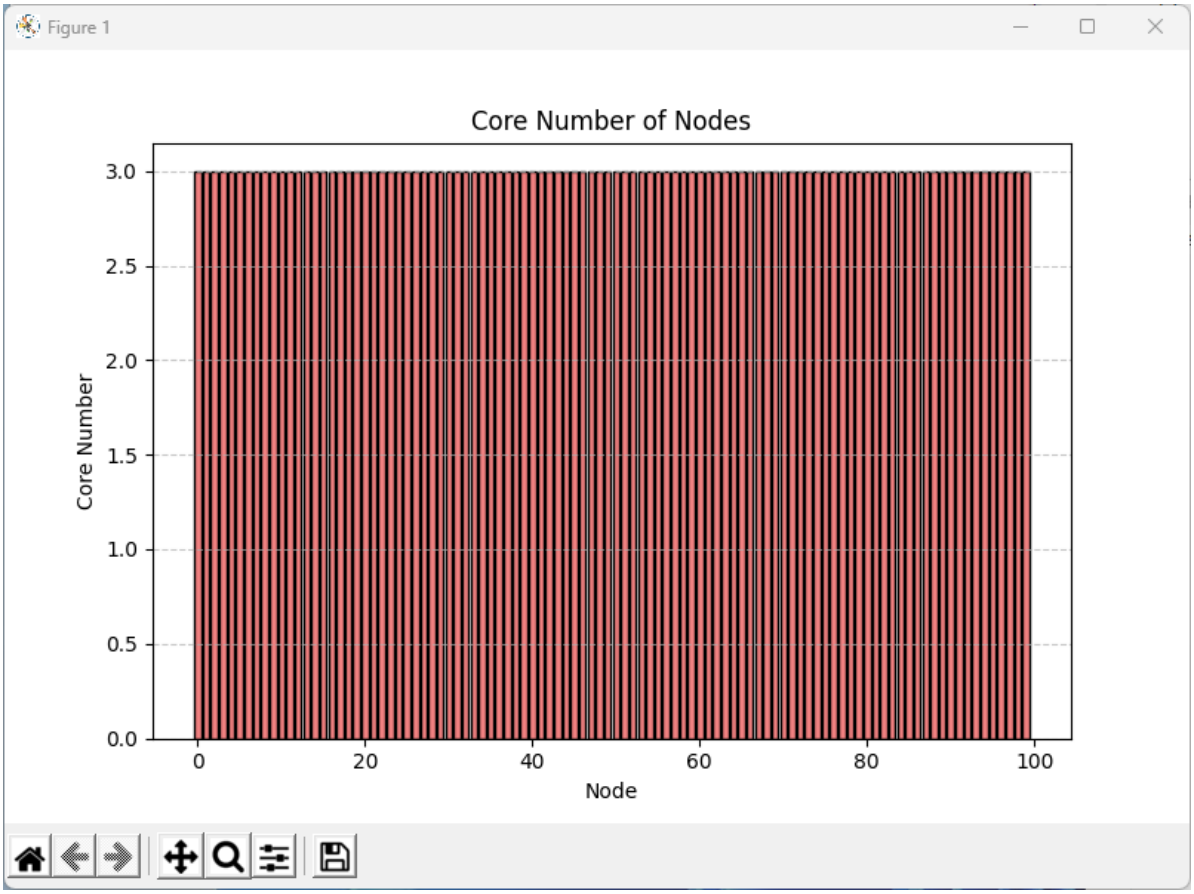


Figure 1
Core Number of Nodes
Core Number
Node
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

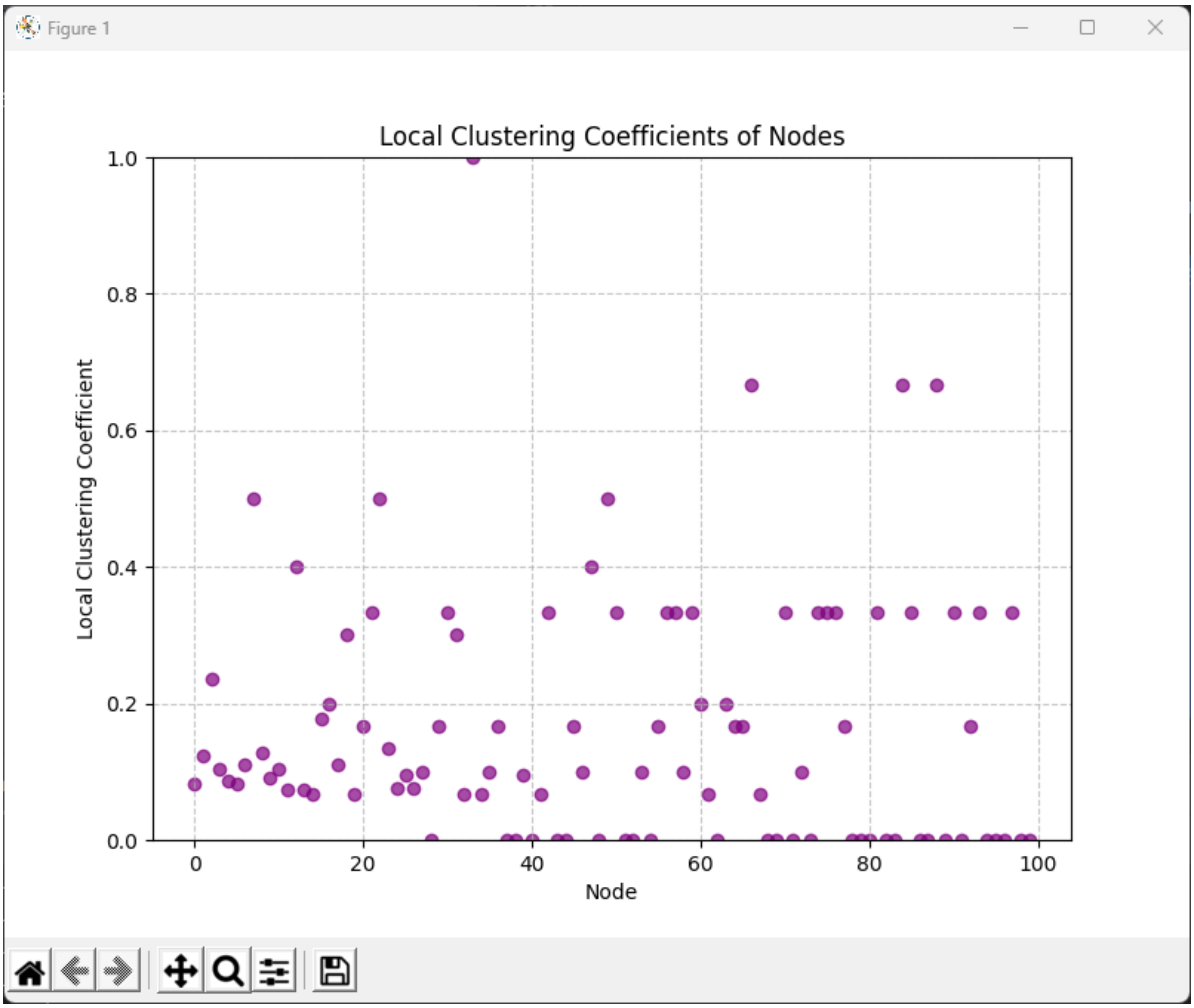
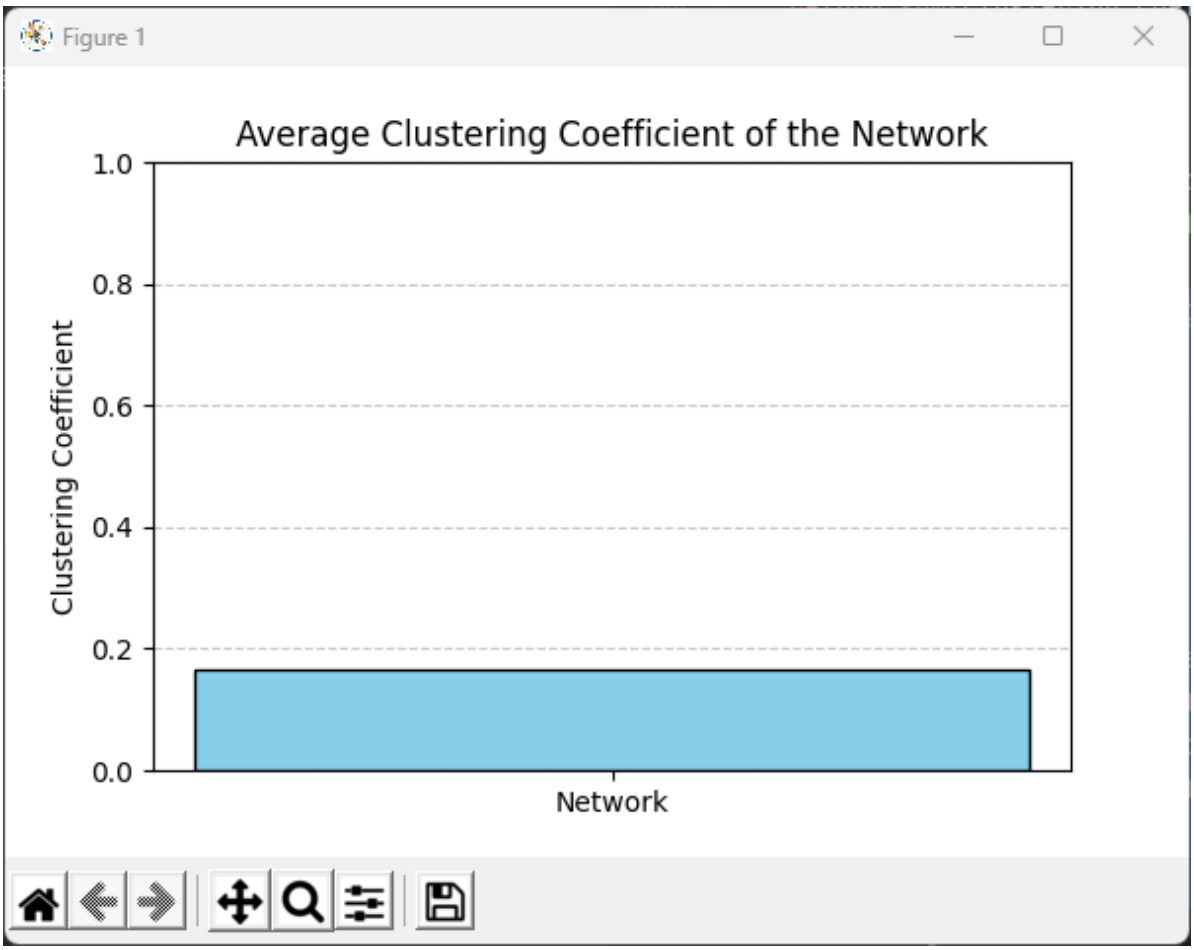


Figure 1
Local Clustering Coefficients of Nodes
Local Clustering Coefficient
Node
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100




```
Degree Distribution: {22: 3, 28: 1, 17: 1, 16: 1, 5: 17, 11: 1, 10: 2, 7: 5, 8:
3, 14: 2, 6: 7, 3: 36, 4: 19, 20: 1, 12: 1}
Total distance 25490
Average Shortest Path Length: 2.5747474747474746
Average Clustering Coefficient: 0.1717951702293807
Random Attack Sizes: [99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 84,
82, 81, 79, 78, 77, 76, 75, 74, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63,
62, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42,
41, 40, 39, 36, 35, 34, 31, 28, 27, 27, 26, 25, 24, 23, 22, 22, 22, 21, 19, 19,
19, 18, 17, 17, 16, 14, 8, 7, 5, 5, 5, 5, 4, 4, 4, 3, 3]
Targeted Attack Sizes: [99, 98, 97, 95, 94, 91, 90, 87, 85, 81, 80, 78, 77, 76,
75, 74, 73, 72, 70, 69, 65, 63, 40, 40, 30, 22, 22, 16, 16, 16, 16, 16, 13, 13,
13, 10, 10, 10, 7, 7, 7, 7, 5, 5, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2,
2, 2, 2]
```