Complex-Network-Community-Analysis

问题描述

1. 阅读本周参考资料。
2. 参照1中的实验思路，利用LFR benchmark，生成一定规模的网络，并通过调整μ来测试不同社团发现算法性能的变化。具体地，发现算法应至少包括CNM和Louvain，评价指标则至少包括模块度，Coverage, Performance， Rand index和NMI。
3. 复杂网络分析工具Gephi支持基于社团的可视化。请利用其将2中某个μ下的发现结果进行可视化。

实现代码

```
import os
import networkx as nx
import community as community_louvain
from networkx.algorithms.community import greedy_modularity_communities
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from itertools import combinations

# Function to calculate coverage and performance
def calculate_coverage_performance(G, communities):
    intra_edges, total_edges = 0, G.number_of_edges()
    for community in communities:
        intra_edges += sum(1 for _ in combinations(community, 2) if
G.has_edge(*_))
    inter_edges = total_edges - intra_edges
    return intra_edges / total_edges, 1 - (inter_edges / total_edges)

# Function to add community to graph nodes
def add_community_to_graph(G, partition):
    for node, community_id in partition.items():
        G.nodes[node]['community'] = str(community_id)
    return G

# Function to save graph with communities to .net file
def save_graph_with_communities(G, path):
    nx.write_gexf(G, path)


def process_community_detection(algorithm_name, G, save_filename, mu):
    if algorithm_name == 'CNM':
        communities = list(greedy_modularity_communities(G))
        partition = {node: idx for idx, comm in enumerate(communities) for node in
comm}
    elif algorithm_name == 'Louvain':
        partition = community_louvain.best_partition(G)
        communities = {}
        for node, community_id in partition.items():
```

```
                communities.setdefault(community_id, []).append(node)
            communities = list(communities.values())
        else:
            raise ValueError("Unsupported algorithm name")

        # Calculate metrics
        modularity = nx.algorithms.community.quality.modularity(G, communities)
        coverage, performance = calculate_coverage_performance(G, communities)

        # Add community info to graph
        G_with_community = add_community_to_graph(G.copy(), partition)

        # Save graph with community info
        save_graph_with_communities(G_with_community, os.path.join(network_dir,
save_filename))

        return {
            'Modularity': modularity,
            'Coverage': coverage,
            'Performance': performance,
        }

# Function to evaluate community detection algorithms
def evaluate_community_detection(G, true_partitions, mu):
    results = {}
    node_to_community = {node: idx for idx, community in
enumerate(true_partitions) for node in community}
    true_labels = [node_to_community[node] for node in G.nodes()]

    # Process CNM algorithm
    cnm_results = process_community_detection('CNM', G,
f'network_cnm_mu_{mu}.gexf', mu)
    cnm_predicted_labels = [G.nodes[node]['community'] for node in G.nodes()]
    cnm_results['Rand Index'] = adjusted_rand_score(true_labels,
cnm_predicted_labels)
    cnm_results['NMI'] = normalized_mutual_info_score(true_labels,
cnm_predicted_labels)
    results['CNM'] = cnm_results

    # Process Louvain algorithm
    louvain_results = process_community_detection('Louvain', G,
f'network_louvain_mu_{mu}.gexf', mu)
    louvain_predicted_labels = [G.nodes[node]['community'] for node in G.nodes()]
    louvain_results['Rand Index'] = adjusted_rand_score(true_labels,
louvain_predicted_labels)
    louvain_results['NMI'] = normalized_mutual_info_score(true_labels,
louvain_predicted_labels)
    results['Louvain'] = louvain_results

    # Save results to Markdown file
    save_results_to_file(results, mu)

def save_results_to_file(results, mu):
    with open('results.md', 'a') as f:
```

```
        f.write(f'\n## Mu = {mu}\n')
        f.write('| Algorithm | Modularity | Coverage | Performance | Rand Index |
NMI |\n')
        f.write('|-----------|------------|----------|-------------|------------|-
----|\n')
        for algo, metrics in results.items():
            f.write(f'| {algo} | {metrics["Modularity"]:.4f} |
{metrics["Coverage"]:.4f} | {metrics["Performance"]:.4f} | {metrics["Rand
Index"]:.4f} | {metrics["NMI"]:.4f} |\n')

if __name__ == "__main__":
    # Create Network directory if it doesn't exist
    network_dir = 'Network'
    if not os.path.exists(network_dir):
        os.makedirs(network_dir)

    # Main example of generating LFR benchmark graph and evaluating it
    mu_values = [0.1, 0.3, 0.5]  # Example mixing parameter values
    for mu in mu_values:
        G = nx.LFR_benchmark_graph(n=1000, tau1=3, tau2=1.5,average_degree = 5,
min_degree=None, max_degree=None, min_community=20,max_community=30, mu=mu,
seed=42)
        true_partitions = {frozenset(G.nodes[v]['community']) for v in G}
        print(f"Evaluating for mu={mu}")
        evaluate_community_detection(G, true_partitions, mu)
```

结果与讨论

不同μ

Mu = 0.1

Algorithm	Modularity	Coverage	Performance	Rand Index	NMI
CNM	0.9106	0.8939	0.8939	0.0064	0.7019
Louvain	0.9064	0.8958	0.8958	0.0064	0.7019

Mu = 0.3

Algorithm	Modularity	Coverage	Performance	Rand Index	NMI
CNM	0.5970	0.6412	0.6412	0.0053	0.7013
Louvain	0.6017	0.6252	0.6252	0.0053	0.7013

Mu = 0.5

Algorithm	Modularity	Coverage	Performance	Rand Index	NMI

Algorithm	Modularity	Coverage	Performance	Rand Index	NMI
CNM	0.5412	0.5934	0.5934	0.0053	0.7013
Louvain	0.5511	0.6036	0.6036	0.0053	0.7013

模块度是衡量网络中社区分割质量的指标。在 μ=0.1 时，CNM 和 Louvain 算法的模块度非常接近，都超过了 0.9，说明这两种算法都能很好地识别出社区结构。然而随着 μ 的增加，模块度有所下降，尤其是在 μ=0.5 时，模块度明显下降，表明社区结构变得模糊，社区之间的边界不那么明显了。

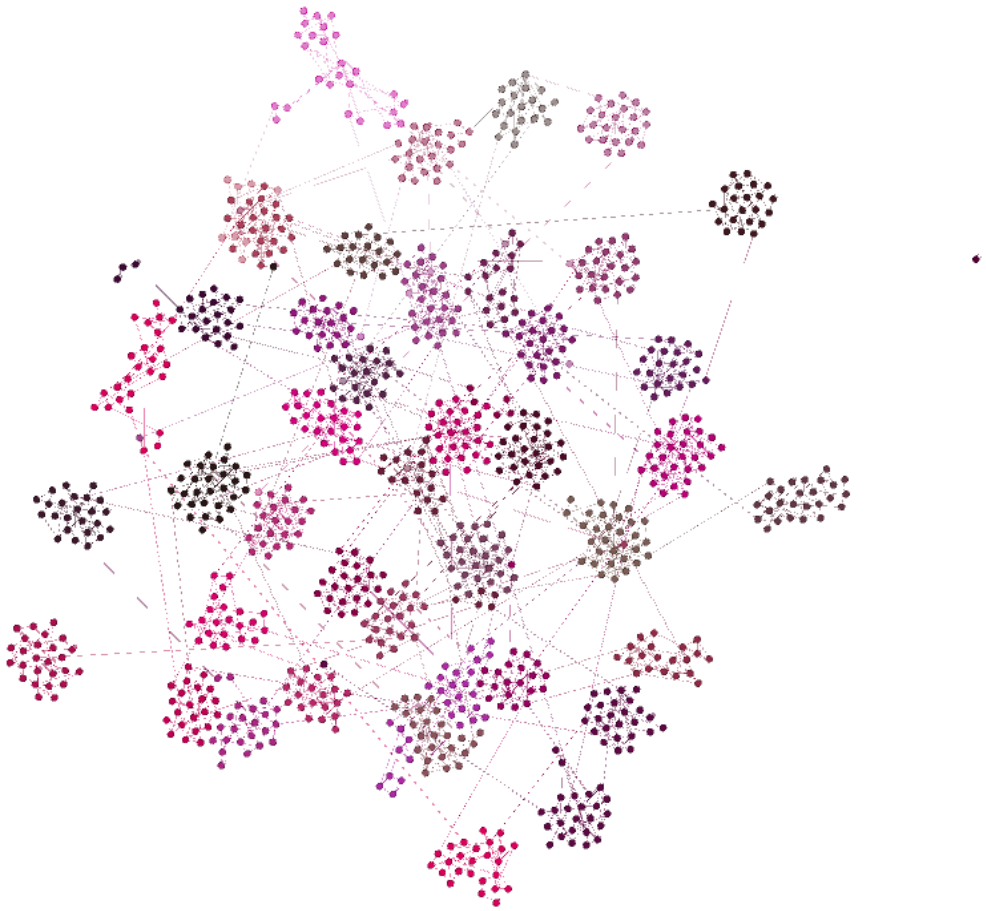
覆盖率和性能在这里得到的数值是一样的，这可能是因为在网络的结构上，社区内部边的数目和社区间边的数目呈现出某种对称性。随着 μ 的增加，这两个指标都下降，表明社区内部的联系相比于整个网络的联系减弱了。

归一化互信息指数也是衡量聚类结果相似度的一个指标，值在0到1之间。从表中看到，NMI的值保持在0.7左右，这表示算法检测到的社区结构与实际的社区结构有一定程度的相似性。

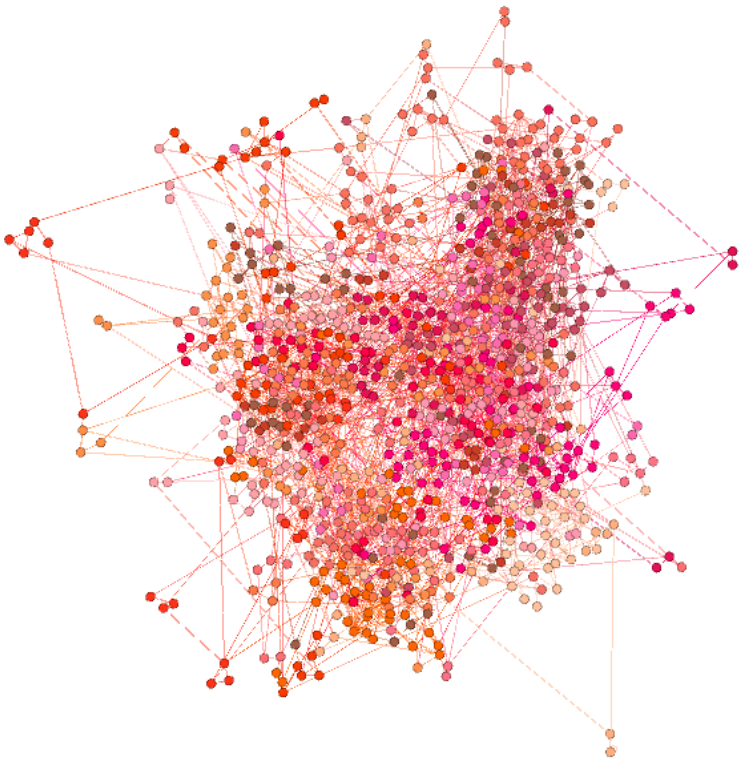
随着 μ 的增加，社区结构变得越来越模糊，所有的性能指标都有所下降。NMI的值表明尽管社区边界变得模糊，算法依然能在一定程度上检测出社区结构，但是这种检测的准确性是有限的。

可视化社区发现

louvain_mu_0.1



louvain_mu_0.3



上方两张图片是在不同混合参数 μ 值下通过 Louvain 方法检测到的 LFR 基准图的群落结构。基于图片结果进行讨论：

当 μ = 0.1 时，群落结构定义明确，具有紧密的节点簇。这与 0.9102 的高模块化得分相一致，表明与社区间链接相比，社区内链接更强。可视化可能显示具有密集内部连接和组之间连接较少的不同组。而当 μ = 0.3 时，虽然群落仍然可见，但不如 μ = 0.1 时那么清晰。这与较低的模块化得分 0.6087 一致，表明随着混合参数的增加，由于社区间连接的增加，算法更难以区分社区。

两个 μ 值之间的可视化差异表明，随着混合参数的增长，检测清晰的群落结构变得越来越困难，凸显了 Louvain 方法在较低混合水平下的有效性以及随着网络结构变得更加同质化而面临的困难。