그래프및네트워크분석 (AAI5005) Homework #1: Network analysis

2024 년 10 월 13 일

2023321342 안민혁

본 보고서에서는 congress_network 데이터셋을 과제의 설명에 따라 분석한다. 제 1 절에서는 unweighted undirected network 로 간주하고, 제 2 절에서는 unweighted directed network 로 간주하여 분석한다. 실험에 사용한 환경은 Ubuntu 18.04.6 LTS, Intel(R) Xeon(R) CPU E5-2698 v4 40 core 2.20GHz, Tesla V100-DGXS (32GB) GPUs 그리고 Python 3.10.13, NumPy 1.26.2 이며 랜덤 seed 를 고정하여 사용하였다. 과제에 대해 누구와 의논하거나 이야기한 바 없음을 알림.

1. undirected network 분석 (undirected.py)

```
import json
import numpy as np
from matplotlib import pyplot as plt
import networkx as nx
```

분석에 사용한 라이브러리는 json, numpy, matplotlib, networkx 임을 밝힘

```
f = open('congress_network/congress_network_data.json')
data = json.load(f)

inList = data[0]['inList']
outList = data[0]['outList']
usernameList = data[0]['usernameList']

# Create an undirected graph
G = nx.Graph()

# Add edges from inList and outList
for i, node in enumerate(usernameList):
    for neighbor in inList[i]:
        G.add_edge(node, usernameList[neighbor])
    for neighbor in outList[i]:
        G.add_edge(node, usernameList[neighbor])
```

다음과 같이 JSON 파일에서 데이터를 읽어와 undirected 그래프를 생성한다. networkx 라이브러리의 Graph() 객체를 사용하여 무방향그래프 G 를 생성하고 usernameList 를 순회하면서 각 노드에 inList 와 outList 에 담긴 이웃 노드와 엣지 정보를 추가한다. 이 때, unweighted network 분석이므로 엣지의 가중치는 고려하지 않았다.

1) 평균 degree, density, degree 분포 등을 조사하라.

```
# 1) Average degree, density, and degree distribution
average_degree = sum(dict(G.degree()).values()) / G.number_of_nodes()
density = nx.density(G)
degree_distribution = [d for n, d in G.degree()]

# Plot the degree distribution
plt.figure(figsize=(8, 6))
plt.hist(degree_distribution, bins=20, color='lightcoral', edgecolor='black')
plt.title('Degree Distribution')
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
plt.savefig('Undirected_Degree_Distribution.png', dpi=300)
```

```
plt.close()
```

- G.degree()는 각 노드의 degree 를 반환하므로 sum / 노드 수로 평균 degree 를 계산하였다.
- nx.density(G)는 G 의 density 를 반환함. Density 는 그래프에서 가능한 최대 엣지 수에 대한 실제 엣지 수의 비율을 나타낸다.
- 평균 degree 는 43.04, Density 는 0.09 로 계산됨된다.
- 리스트 형태로 변환한 뒤, degree 분포를 히스토그램 시각화하였음 (Frequency 분석). 해당 히스토그램은 다음과 같다.

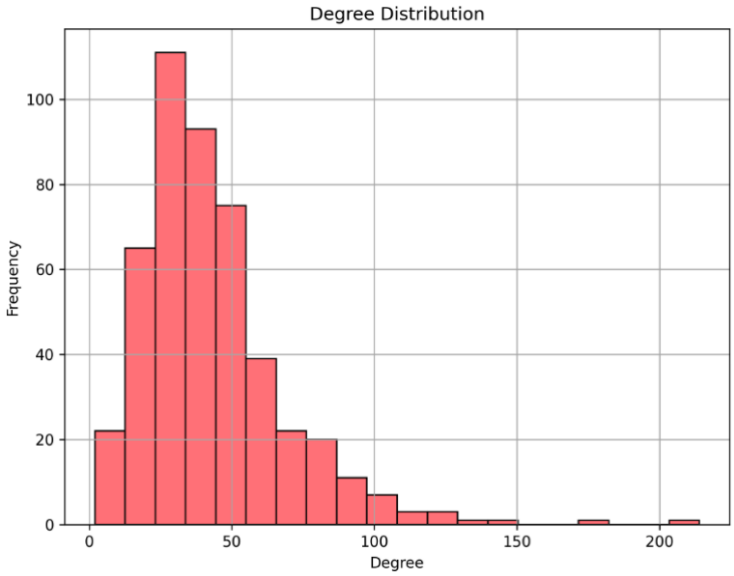


그림 1. Degree 분포 히스토그램

해당 데이터셋에서는 Degree 가 30~40 인 노드 수가 가장 많이 존재함을 확인할 수 있다 (Positive skew 형태).

2) Graph Laplacian 의 eigenvalues 를 조사하고, 수업시간에 다룬 특성들을 확인하라.

```
# 2) Graph Laplacian eigenvalues and eigenvectors
laplacian = nx.laplacian_matrix(G).todense()
eigenvalues, eigenvectors = np.linalg.eig(laplacian)

# Check if eigenvalue 0 exists and its corresponding eigenvector
zero_eigenvalue_index = np.where(np.isclose(eigenvalues, 0))[0]
if zero_eigenvalue_index.size > 0:
    zero_eigenvector = eigenvectors[:, zero_eigenvalue_index[0]]
    print(f"Eigenvector corresponding to eigenvalue 0: {zero_eigenvector}")


# Plot the eigenvalues sorted
plt.figure(figsize=(8, 6))
plt.plot(sorted(eigenvalues), 'o-', label='Eigenvalues')
plt.title('Laplacian Eigenvalues')
plt.xlabel('Index')
plt.ylabel('Eigenvalue')
plt.grid(True)
plt.legend()
plt.show()
plt.savefig('Undirected_Eigenvalues_sorted.png', dpi=300)
plt.close()

# Plot the eigenvalues unsorted
plt.figure(figsize=(8, 6))
```

```
plt.plot(eigenvalues, 'o-', label='Eigenvalues')
plt.title('Laplacian Eigenvalues')
plt.xlabel('Index')
plt.ylabel('Eigenvalue')
plt.grid(True)
plt.legend()
plt.show()
plt.savefig('Undirected_Eigenvalues_unsorted.png', dpi=300)
plt.close()
```

- nx.laplacian_matrix(G)는 네트워크 그래프 G 의 라플라시안 행렬을 반환한다. 이 행렬은 그래프의 연결성을 나타내며, 각 노드의 차수와 인접 노드 간의 관계를 포함한다. 해당 행렬은 [노드 수 X 노드 수] 차원의 행렬이다.
- np.linalg.eig(laplacian)는 라플라시안 행렬의 고유값과 고유벡터를 계산한다. 고유값은 그래프의 구조적 특성을 나타내며, 고유벡터는 각 고유값에 대응하는 방향을 나타낸다..
- 수업시간에 다룬 특성을 확인하기 위해 고유값 0 의 존재 여부를 확인하고 시각화하였다.

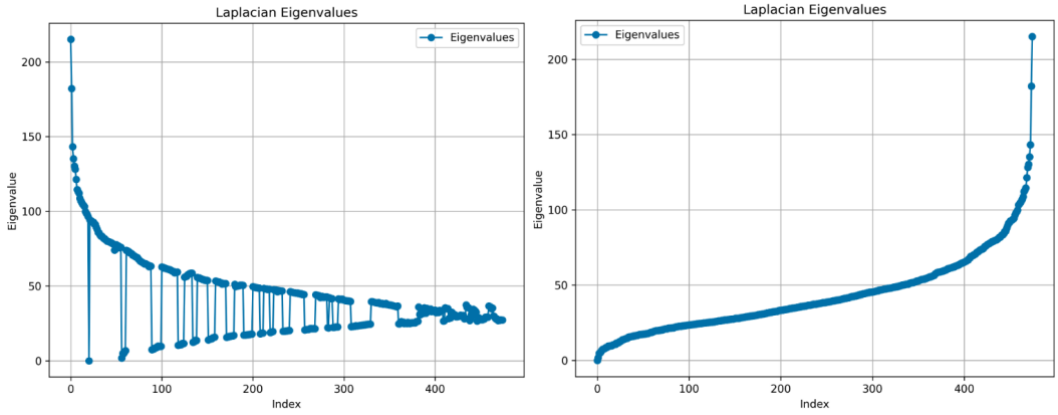


그림 2. Laplacian eigenvalues 시각화 (좌: 정렬 전, 우: 정렬 후)

고유값은 0 에 가까운 양수부터 존재, 200 이상의 범위를 가진다. 대응하는 인덱스의 고유벡터를 출력한 결과, 고유벡터는 [노드 수]의 -0.04588315 로 채워진 벡터로 분석된다. 수업시간에서 배운 내용에 따르면 1 으로 채워진 벡터일 것을 예상했으나 그러하지 못했는데, 해당 원인은 연산 상에서 발생한 양자화 오류라고 예상됨

3) Global clustering coefficient 및 각 node 의 local clustering coefficient 를 조사하라.

```
# 3) Clustering coefficients
global_clustering = nx.transitivity(G)
local_clustering = nx.clustering(G)
# Plot the local clustering coefficients
plt.figure(figsize=(8, 6))
plt.hist(local_clustering.values(), bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Local Clustering Coefficients')
plt.xlabel('Local Clustering Coefficient')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
plt.savefig('Undirected_Local_Clustering_Distribution.png', dpi=300)
plt.close()
```

- nx.transitivity(G)는 그래프 G 의 전역 클러스터링 계수를 계산한다. 전역 클러스터링 계수는 그래프 내의 삼각형(세 개의 노드가 서로 연결된 경우)의 비율을 나타내며, 네트워크의 전체적인 밀집도를 나타낸다. 전역 클러스터링 계수는 0.27 로 계산된다.

- nx.clustering(G)는 그래프 G 의 각 노드에 대한 지역 클러스터링 계수를 계산한다. 지역 클러스터링 계수는 특정 노드의 이웃들이 서로 얼마나 연결되어 있는지를 나타내며, 노드별로 계산된 값이 딕셔너리 형태로 반환된다.
- 노드의 지역 클러스터링 계수 분포를 분석(Frequency 분석)한 뒤 히스토그램 시각화한 결과는 다음과 같다.

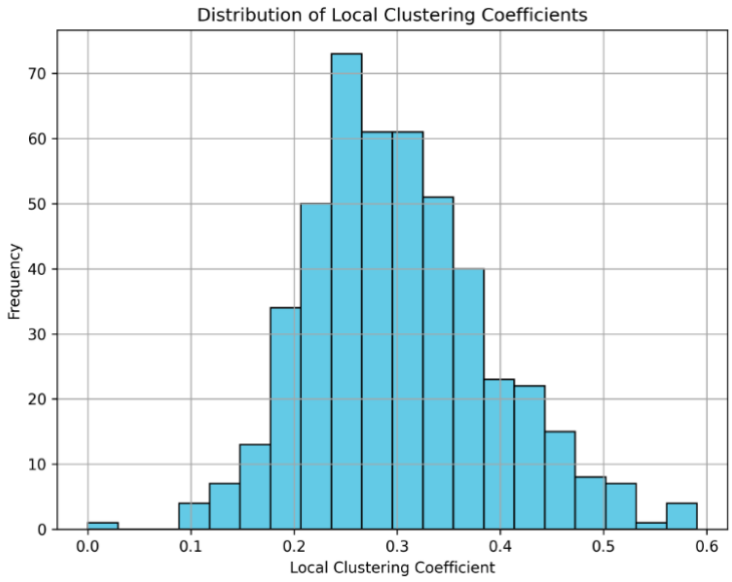


그림 3. Local clustering coefficients 의 분포 히스토그램

시각화 결과 Symmetrical 한 경향를 보이며 0.25 근처에서 가장 높은 빈도수 (peak)를 확인할 수 있다. 이는 대부분의 이웃 노드들이 서로 연결된 비율이 약 25%정도라는 것으로 해석할 수 있다.

4) Covariance 를 통해 assortative/dissortative mixing by degree 를 확인하라.

```
# 4) Assortative mixing by degree
assortativity = nx.degree_assortativity_coefficient(G)

# Calculate covariance of degrees between connected nodes
degree_dict = dict(G.degree())
degree_pairs = [(degree_dict[u], degree_dict[v]) for u, v in G.edges()]
degrees_u, degrees_v = zip(*degree_pairs)

# Calculate covariance
covariance = np.cov(degrees_u, degrees_v, bias=True)[0][1]

# Print the results
print(f"Assortativity (Degree Mixing): {assortativity}")
print(f"Covariance of Degrees: {covariance}")

# Interpretation
if covariance > 0:
    print("The network exhibits assortative mixing by degree.")
elif covariance < 0:
    print("The network exhibits dissortative mixing by degree.")
else:
    print("The network shows no particular preference in degree mixing.")
```

- np.cov(degrees_u, degrees_v, bias=True)는 연결된 노드 쌍의 차수 간의 공분산을 계산한다. 공분산은 두 변수 간의 상관관계를 나타내며, 양수일 경우 두 변수는 같은 방향으로 변화하는 경향이 있다. 공분산의 부호에 따라 mixing by degree 를 해석할 수 있으며 양수일 경우 assortative, 음수일 경우 dissortative

mixing, 0 인 경우 특별한 경향이 없음을 나타낸다. 동일 변수끼리의 상관관계는 1331, 812 로 측정되며 두 변수의 공분산 분석 결과는 -35.77082598 임. 즉, dissortative 로 해석된다.
- nx.degree_assortativity_coefficient(G)는 그래프 G 의 차수에 따른 mixing by degree 계수를 계산한다. 이 계수는 -1 에서 1 사이의 값을 가지며, 1 에 가까울수록 유사한 차수를 가진 노드들이 서로 연결되는 경향이 강하다는 것을 의미한다(assortative). 반대로 -1 에 가까울수록 서로 다른 차수를 가진 노드들이 연결되는 경향이 강하다(dissortative**)**. 해당 값 출력 결과, -0.07846534685468998 으로 dissortative 로 결론낼 수 있다.

5) Clique 을 찾아보라.

```
# 5) Find cliques
cliques = list(nx.find_cliques(G))
```

- nx.find_cliques(G)는 그래프 G 에서 모든 최대 클리크를 찾는다. 최대 클리크는 더 이상 노드를 추가할 수 없는 완전 subgraph 를 의미하며 Clique 의 수를 측정한 결과 21979 개로 측정된다.

6) 네트워크를 visualize 하고 관찰하라.

```
# 6) Visualize the network with additional information
plt.figure(figsize=(18, 10))

pos = nx.spring_layout(G, seed=1)
# Node color based on local clustering coefficient
node_colors = [local_clustering[node] for node in G.nodes()]

# Node size based on degree
node_sizes = [G.degree(node) * 6 for node in G.nodes()]

# Edge color based on weight (assuming all edges have weight 1 for simplicity)
edge_colors = ['gray' for _ in G.edges()]

# Create a ScalarMappable for the colorbar
sm = plt.cm.ScalarMappable(cmap=plt.cm.viridis, norm=plt.Normalize(vmin=min(node_colors),
vmax=max(node_colors)))
sm.set_array([])

# Draw the graph
nx.draw(G, pos, node_color=node_colors, node_size=node_sizes, edge_color=edge_colors,
       with_labels=False, font_size=6, cmap=plt.cm.viridis, width=0.4)

# Add colorbar with a reduced size
cb = plt.colorbar(sm, ax=plt.gca(), label='Local Clustering Coefficient', shrink=0.8)

# Add text annotations
plt.text(0.05, 0.95, f'Average Degree: {average_degree:.2f}', transform=plt.gca().transAxes,
fontsize=30)
plt.text(0.05, 0.90, f'Density: {density:.2f}', transform=plt.gca().transAxes, fontsize=30)
plt.text(0.05, 0.85, f'Global Clustering: {global_clustering:.2f}',
transform=plt.gca().transAxes, fontsize=30)
plt.text(0.05, 0.80, f'Assortativity: {assortativity:.2f}', transform=plt.gca().transAxes,
fontsize=30)
plt.text(0.05, 0.75, f'Number of Cliques: {len(cliques)}', transform=plt.gca().transAxes,
fontsize=30)
# plt.colorbar(sm, ax=ax)
plt.title("Network Visualization with Properties")
plt.tight_layout()  # Adjust layout to make room for the colorbar
plt.savefig('Undirected_Visualization_with_Properties.png', dpi=300)
plt.close()
```

- network 라이브러리의 draw() 함수와 matplotlib 을 이용해 그래프를 시각화할 수 있다. spring_layout() 함수에 따라 노드의 위치를 결정하였고, Seed=1 로 고정하여 시각화 재현이 동일하게 이루어질 수 있도록 하였다. Degree 에 따라 노드 크기를 결정하였으며 Local clustering coefficient 값에 따라 노드 색상을 연속적으로 지정해주었다. 해당 결과는 다음과 같다.

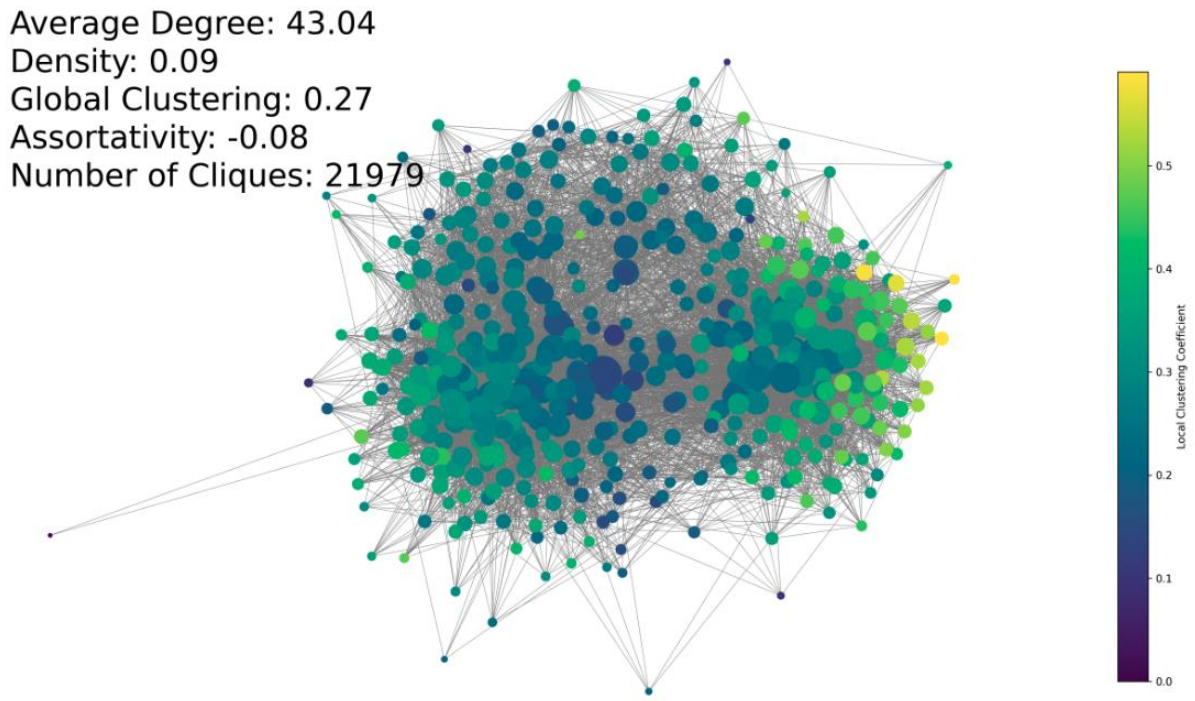


그림 4. 그래프 시각화 결과.

크기가 큰 노드들은 주로 시각화 결과의 중심에 위치되는 것으로 분석된다. 이는 연결성이 큰 노드들이 중심이 되어 중앙에 배치된다는 것으로 해석할 수 있다. 또, local 클러스터링 계수가 큰 (노란색) 값의 노드들이 주로 비슷한 위치에 군집되어 나타나 있는 것도 확인할 수 있다 (오른쪽).

2. directed network 분석 (directed.py)

```
import pandas as pd
import seaborn as sns
```

2 번 과제를 수행하기 위해 pandas 와 seaborn 라이브러리를 추가로 불러왔다. pandas 와 seaborn 은 데이터 분석과 heatmap 시각화를 위함이다.

```
# Create a directed graph
G = nx.DiGraph()
```

1 번의 undirected 분석과는 다르게, 이번에는 위와 같이 DiGraph() 객체를 이용해 directed 그래프 G 를 생성하였으며, 노드와 엣지 상태 연결은 1 번과 동일한 방법임을 밝히며 구현 방법은 생략한다.

1) 각 node 의 centrality 를 계산하라.(eigenvector centrality, Katz centrality, PageRank, closeness centrality, betweenness centrality 등) 어떤 node 가 가장 큰/작은 centrality 를 갖는지, centrality 의 분포는 어떠한지 등을 분석하라.

```
# 1) Calculate centralities without weights
eigenvector_centrality = nx.eigenvector_centrality_numpy(G)
katz_centrality = nx.katz_centrality_numpy(G)
pagerank = nx.pagerank(G)
closeness_centrality = nx.closeness_centrality(G)
betweenness_centrality = nx.betweenness_centrality(G)
```

```
# Combine centralities into a DataFrame
centralities = pd.DataFrame({
    'Eigenvector': eigenvector_centrality,
    'Katz': katz_centrality,
    'PageRank': pagerank,
    'Closeness': closeness_centrality,
    'Betweenness': betweenness_centrality
})

# Analyze centrality distribution
print("Centrality Distribution:")
print(centralities.describe())

# Plot centrality distribution
centralities.plot(kind='box', subplots=True, layout=(3, 2), figsize=(12, 8))
plt.tight_layout()
plt.show()
plt.savefig('Directed_centralities.png', dpi=300)
plt.close()
correlation_matrix = centralities.corr()
print("Centrality Correlation Matrix:")
print(correlation_matrix)
```

- eigenvector_centrality: 노드의 중요성을 측정하여, 중요한 노드와 연결된 노드가 더 중요하다고 평가한다.
katz_centrality: 노드의 직접적인 연결뿐만 아니라 간접적인 연결도 고려한다.
pagerank: 웹 페이지의 중요성을 평가하는 알고리즘으로, 노드의 중요성을 측정한다.
closeness_centrality: 노드가 다른 모든 노드에 얼마나 가까운지를 측정한다.
betweenness_centrality: 노드가 다른 노드 쌍 간의 최단 경로에 얼마나 자주 등장하는지를 측정한다.
이러한 중심성 지표는 네트워크 내에서 중요한 노드나 영향력 있는 노드를 식별하는 데 사용된다.
- 위와 같이 pandas 라이브러리와 seaborn 을 이용하여 각 centrality 를 Botplot 한 뒤 비교 분석하였으며 그 결과는 다음과 같다.

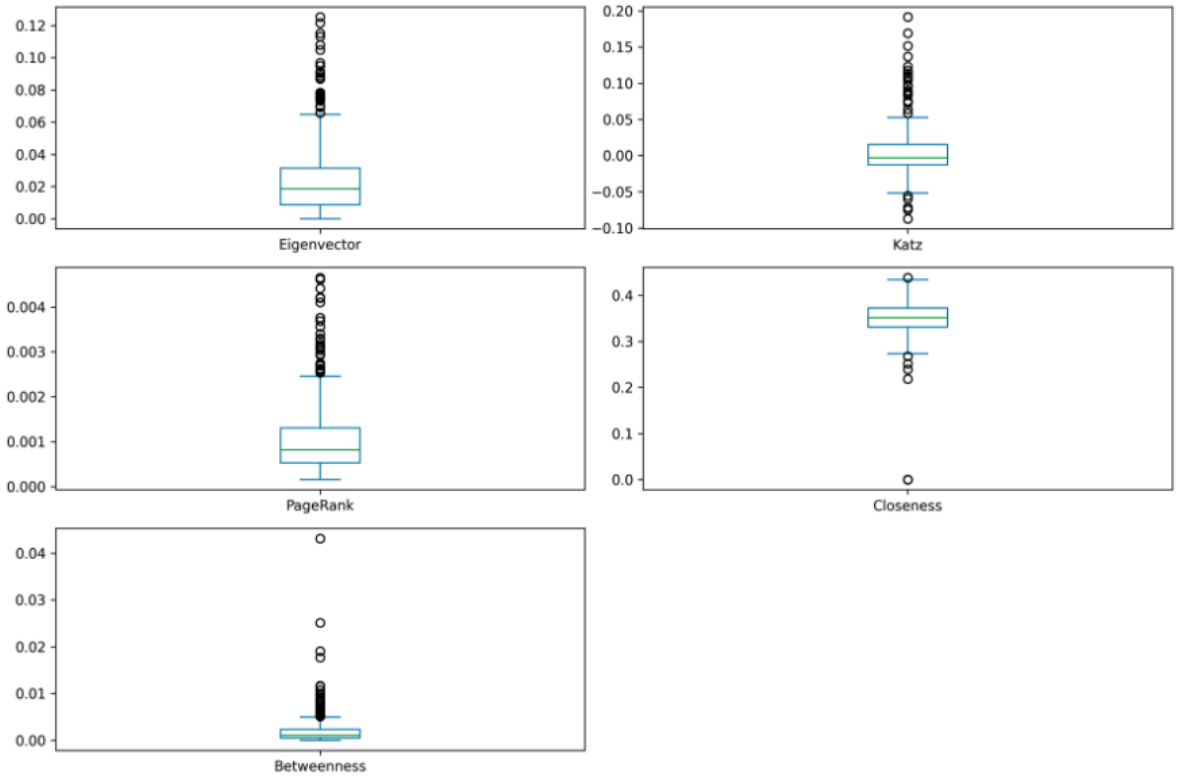


그림 5. Centrality 별 Box plot 결과.

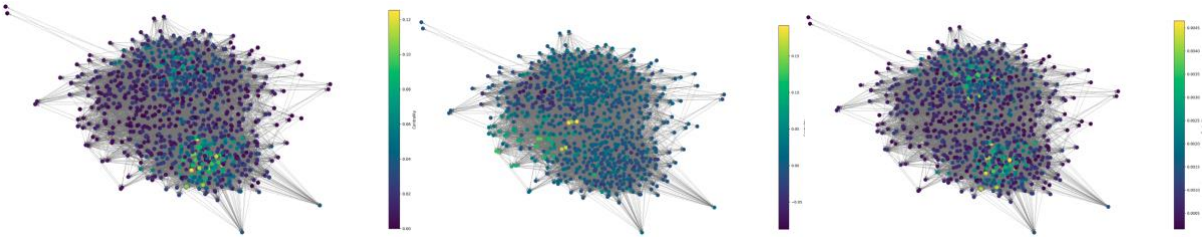
- Eigenvector centrality: -0.00~0.12 정도의 범위, outlier 가 다수 발견됨.
- Katz centrality: -0.1~0.2 정도의 범위, outlier 가 다수 발견됨, 음수 값을 포함하는 것이 관찰됨
- PageRank centrality: 0~0.005 정도의 범위, outlier 가 다수 발견됨, Eigenvector 와 분포가 유사하며 가장 작은 범위가 관찰됨.
- Closeness centrality: 0~0.45 정도의 범위, outlier 가 가장 적은 편이며 Median 에 조밀하게 분포. 가장 큰 값의 범위를 관찰됨
- Betweenness centrality: 0~0.04 정도의 범위, Median 에 가장 조밀하게 분포하는 것이 관찰됨

구체적인 수치는 다음과 같다.

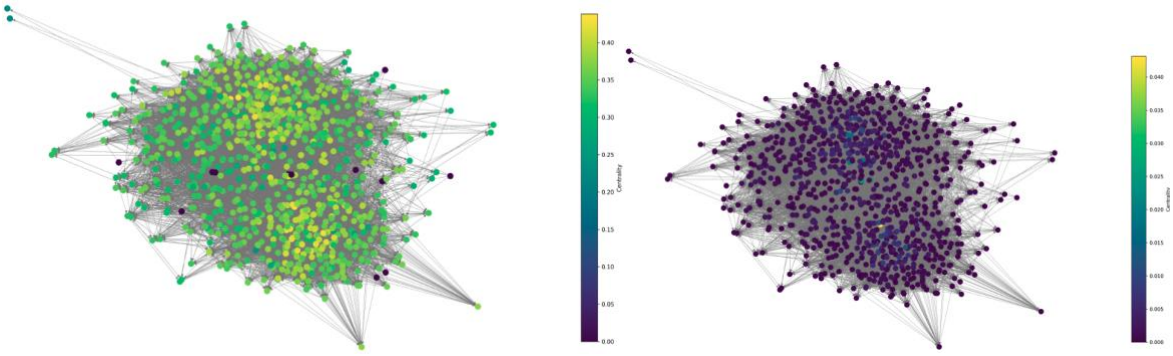
Centrality Distribution:

	Eigenvector	Katz	PageRank	Closeness	Betweenness
count	9.500000e+02	950.000000	950.000000	950.000000	950.000000
mean	2.401101e-02	0.003762	0.001053	0.348091	0.001937
std	2.183128e-02	0.032242	0.000792	0.050159	0.003099
min	-1.553735e-18	-0.087310	0.000158	0.000000	0.000000
25%	8.692955e-03	-0.012586	0.000529	0.331355	0.000466
50%	1.850998e-02	-0.002887	0.000818	0.351351	0.001062
75%	3.138799e-02	0.015170	0.001303	0.372522	0.002293
max	1.253289e-01	0.191599	0.004651	0.438337	0.043155

- 다음은 각 centrality 를 기반으로 한 시각화이다.



(a) Eigenvector (b) Katz (c) PageRank



(d) Closeness (e) Betweenness

그림 6. Centrality 계수 6 종에 따른 그래프 시각화

(a)~(e) 모두 특정 밝은 노드를 중심으로 색상 변화가 연속적으로 나타나며 군집이 형성되는 것이 관찰되며, betweenness 의 경우 그것이 가장 덜 뚜렷하게 나타나는 것이 관찰된다. Katz 의 경우 군집이 더 많이 관찰되기도 하였다.

2) Centrality 간의 correlation 을 조사하여 어떤 centrality 척도들이 서로 유사하거나 다른지 분석하라.

```
# 2) Correlation between centralities
# Plot correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Centrality Correlation Matrix')
plt.tight_layout()
plt.show()
plt.savefig('Directed_Centrality_Correlation_Matrix.png', dpi=300)
plt.close()
```

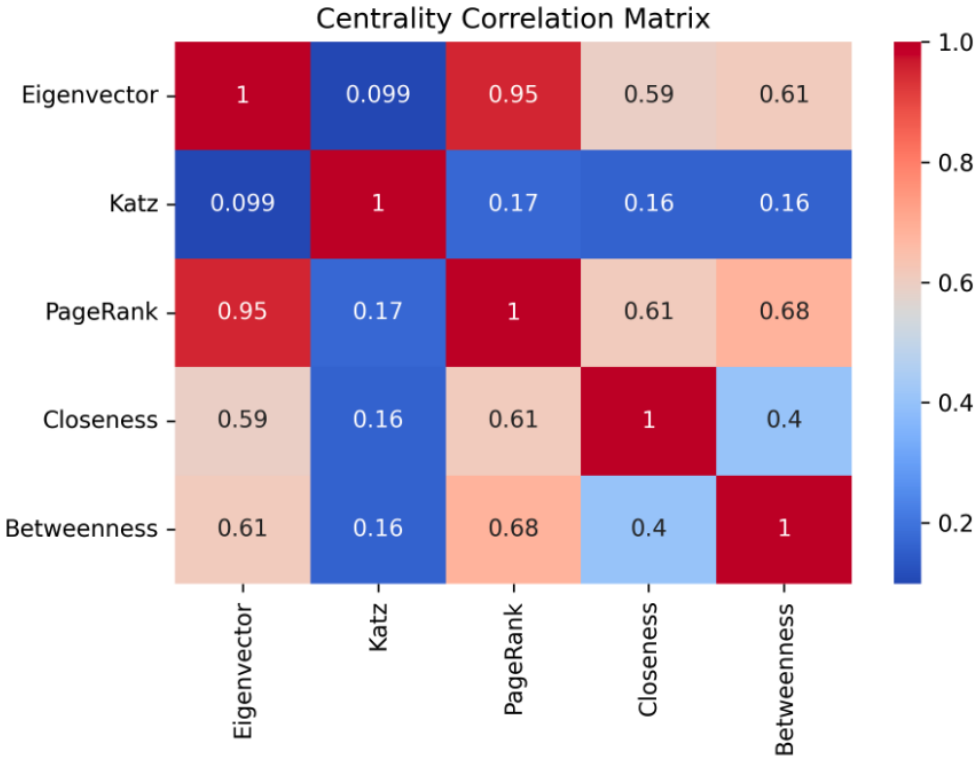


그림 6. Centrality 5 종에 대한 correlation matrix 결과

- 그림 6 은 Centrality 5 종에 대한 correlation matrix 결과이다. Eigenvector-PageRank 간 유사도가 가장 높게 측정되었으며, Katz-PageRank 간 유사도가 가장 낮게 측정되었다. Katz centrality 같은 경우 다른 4 종과 대체로 correlation 이 낮게 측정되는 뚜렷한 특징이 관찰된다.

3) Reciprocity 를 조사하라.

```
# 3) Reciprocity
reciprocity = nx.reciprocity(G)
print(f"Reciprocity of the network: {reciprocity}")
```

- reciprocity() 함수를 통해 계산할 수 있으며, 높은 reciprocity 은 네트워크가 쌍방향 상호작용을 많이 포함하고 있음을 나타내며, 이는 social network 등 분석에서 중요한 특성일 수 있다. 계산된 결과는 0.4615847693581157 이다.

그래프및네트워크분석 (AAI5005) Homework #2: Scale-free network

2024 년 11 월 17 일

2023321342 안민혁

본 보고서에서는 Barabasi-Albert 모델과 변형된 모델을 만들어서 특성을 분석해본다. 실험에 사용한 PC 환경은 macOS Sequoia, Apple M2, 8GB 메모리며 라이브러리 환경은 Python 3.12.1, NumPy 2.1.1, 실험의 reproducibility 를 위해 랜덤 시드=2024 로 고정했을 밝힘. 실험에 사용한 소스코드는 hw2.py 임. 그 누구와도 과제에 대해 의논하지 않았음을 밝힘.

1. Barabasi-Albert (BA) 모델

1-1. BA 모델 알고리즘 정의

```
# 초기 complete graph 생성 (인접 리스트)
def create_complete_graph(n):
    adj_list = defaultdict(set)
    for i in range(n):
        for j in range(i + 1, n):
            adj_list[i].add(j)
            adj_list[j].add(i)
    return adj_list
```

알고리즘 1. 초기 complete graph 생성

알고리즘 1 과 같이 초기 undirected network 를 생성한다. 이 때, 초기 complete graph 의 노드 수는 10 으로 설정했으며, 기존 코드나 함수를 사용하지 않는데, 계산 효율성을 위해, 인접행렬 대신 인접 리스트를 활용하여 생성한다.

```python
# 네트워크 생성 함수
def preferential_attachment_network(n_total, c):
    adj_list = create_complete_graph(n_initial)  # 초기 complete graph 생성
    degrees = np.array([len(adj_list[i]) for i in range(n_initial)])  # node degree

    for new_node in range(n_initial, n_total):
        probs = degrees / degrees.sum()  # 연결 확률 (노드 degree 에 비례)
        targets = np.random.choice(range(new_node), size=c, replace=False, p=probs)  #
선택된 기존 노드들

        # 선택된 노드와 연결
        for target in targets:
            adj_list[new_node].add(target)
            adj_list[target].add(new_node)

        # degree 업데이트
        degrees = np.append(degrees, c)  # 새 노드의 degree 추가
        degrees[targets] += 1  # 연결된 기존 노드의 degree 증가

    return adj_list
```

알고리즘 2. Perferential attachment

알고리즘 2 는 알고리즘 1 에 의해 생성된 초기 undirected network 의 기존 노드 degree 를 바탕으로 새로운 node 를 연결하는 알고리즘이다. 가령, 10 개의 노드를 가진 그래프는 각 노드의 degree 가 각각 [9,9,9,9,9,9,9,9,9,9]이므로, 새로운 노드와 연결된 확률은 1/9 이다. 그 중 c 개를 선택하고 새로운 노드와 연결한다. 연결 뒤, 연결 상태를 업데이트한다.

1-2. 분석 알고리즘 정의

```
# MLE 기반 exponent 추정
def estimate_alpha(x):
    xmin = x.min()
    x = x[x >= xmin]
    n = len(x)
    sum_log_ratios = sum(np.log(x / (xmin-0.5)))
    alpha = 1 + n / sum_log_ratios
    err = (alpha - 1) / n ** 0.5
    return alpha, err

# degree 분포 분석
        degrees = np.array([len(neighbors) for neighbors in adj_list.values()])
        alpha, err = estimate_alpha(degrees)
        print(f"Degree Distribution Power-law Exponent (c={c}): {alpha} += {err}")


# degree 분포 그래프
        plt.hist(degrees, bins=range(1, max(degrees) + 1), density=True,
edgecolor='black')
        plt.title(f"Degree Distribution (c={c}, n={n_total})")
        plt.xlabel("Degree $k$")
        plt.ylabel("Fraction of vertices $P_k$ having degree k")
        plt.yscale('log')
        plt.xscale('log')
        plt.ylim(top=1)
```

알고리즘 3. Degree distribution 분포 분석 알고리즘

알고리즘 3 은 그래프에 대해 degree distribution 을 조사하고 power-law 의 alpha 를 추정하는 알고리즘이다. degree distribution 에 대해 exponent 를 MLE 로 추정하는 식은 다음과 같다.

$$\alpha = 1 + n\left(\Sigma_i^n \ln\left(\frac{k_i}{k_{min} - \frac{1}{2}}\right)\right)^{-1}, error = \pm\frac{\alpha - 1}{\sqrt{n}}$$

알고리즘 4 는 clustering coefficient 를 계산하는 알고리즘이다. 특정 노드에 대한 이웃 노드끼리 연결되어있는 경우 해당 coefficient 값이 증가한다.

```python
# calculate clustering coefficient 계산 함수
def calculate_clustering_coefficient(adj_list):
    triangles = 0
    triplets = 0

    for _, neighbors in adj_list.items():
        k = len(neighbors)
        if k < 2:
            continue

        # 이웃 노드 간의 연결 확인
        neighbors = list(neighbors)
        for i in range(k):
            for j in range(i + 1, k):
                if neighbors[j] in adj_list[neighbors[i]]:
                    triangles += 1
        triplets += k * (k - 1)

    return triangles / triplets if triplets > 0 else 0
```

알고리즘 4. Clustering coefficient 계산 알고리즘

```python
# diameter 계산 함수 (BFS 기반)
def calculate_diameter(adj_list):
    def bfs(node):
        visited = {node}
        queue = deque([(node, 0)])
        max_dist = 0
        while queue:
            current, dist = queue.popleft()
            max_dist = max(max_dist, dist)
            for neighbor in adj_list[current]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append((neighbor, dist + 1))
        return max_dist

    # 가장 먼 거리의 노드 쌍 찾기
    max_diameter = 0
    for node in adj_list:
        max_diameter = max(max_diameter, bfs(node))
    return max_diameter
```

알고리즘 5. Diameter 알고리즘

알고리즘 5 는 Diameter 를 조사하기 위한 알고리즘이다. 앞서, 1-1 절에서 그래프를 인접리스트를 활용해 표현했다고 밝혔다. 따라서 노드와 노드끼리의 거리는 Breath-first search (BFS)를 이용하여 구할 수 있고, 그 중 거리가 가장 긴 값을 반환한다.

1-3 n 과 c 의 변화에 따른 분석

해당 섹션에서는 1-1, 1-2 에서 정의한 알고리즘을 바탕으로 n 과 c 를 변화해가며 결과를 관찰해보겠다.

표 1. n 과 c 변화에 따른 값 관찰 표

n	c	$\alpha \pm error$	Clustering coefficient	Diameter
5000	1	1.942±0.0133	0.0030	17
	2	2.398±0.0197	0.0033	8
	3	2.589±0.0224	0.0036	7
	4	2.678±0.0237	0.0044	6
	5	2.736±0.0245	0.0050	5
10000	1	1.942±0.0133	0.0030	17
	2	2.398±0.0197	0.0033	8
	3	2.589±0.0224	0.0036	7
	4	2.678±0.0237	0.0044	6
	5	2.736±0.0245	0.0050	5
15000	1	1.940±0.0076	0.0009	23
	2	2.396±0.0114	0.0010	9
	3	2.579±0.0128	0.0014	7
	4	2.679±0.0137	0.0016	6
	5	2.740±0.0142	0.0019	6

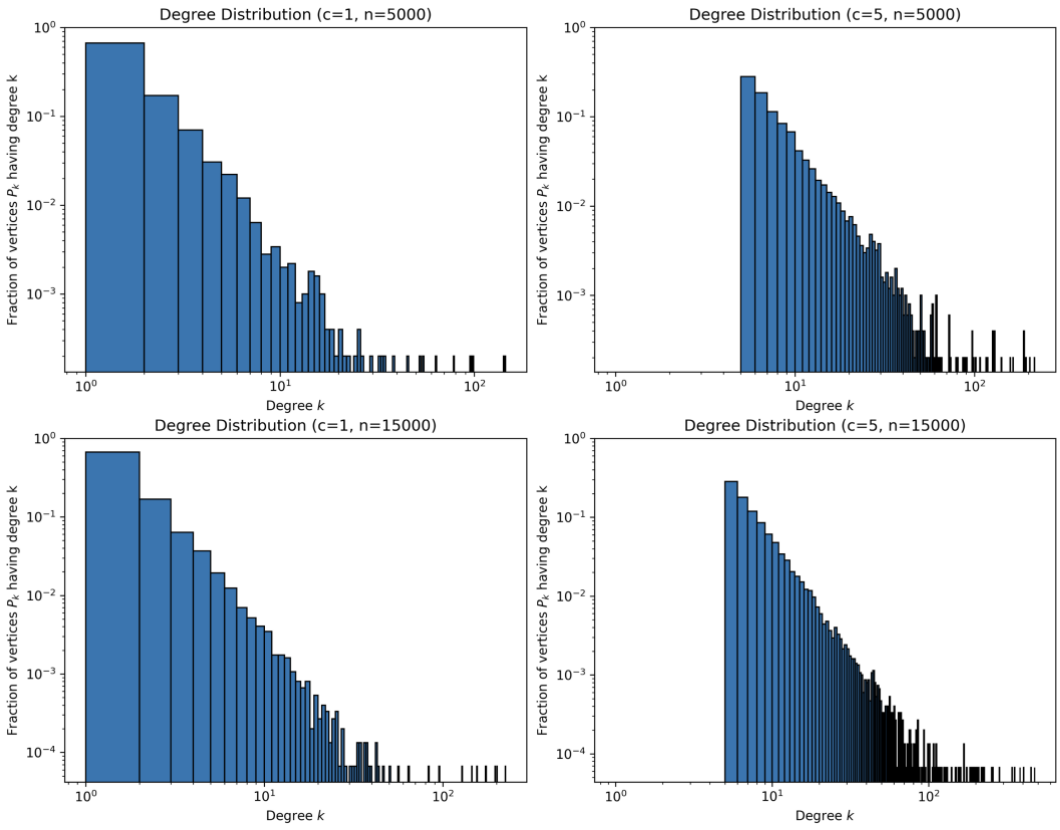


그림 7. Degree distribution 시각화

표 1 에 따르면 α, clustering coefficient 는 c 가 증가하면 할수록 증가하며, diameter 는 감소하는 경향이 보인다. 반면, c 를 고정하고 n 를 기준으로 변화를 살펴보면 n 이 변화함에 따라 α는 거의 변화하지 않으며, clustering coefficient 는 느리게 감소, diameter 는 느리게 증가하는 것을 보인다. 이론적으로는, BA 모델에서 $n = \{5000, 10000, 15000\}$에 따라 $\alpha = 3$ 정도의 값을 가지며, 클러스터링 계수는 $n^{-0.75}$ =(0.0016, 0.0010, 0.0007)로 감소하고 지름은 $\frac{\log(n)}{\log(\log n)}$ =(6.51, 6.64, 6.72)로 증가한다. 실험의 결과와 이론 예측값이 동일하진 않지만, 예측 가능한 범위에 있거나, 변화 양상이 이론과 일치하다는 것을 확인했다. 그림 2 는 $n = \{5000, 15000\}$, 그리고 $c = \{1, 5\}$일 일 때의 degree distribution 을 시각화한 것이다. n 의 변화와 관계없이 distribution 이 거의 유사한 (α가 유사한) scale-free 의 성질을 확인할 수 있다.

2. 변형된 BA 모델

2-1. 무작위 edge 추가 알고리즘

```
# edge 추가 과정
      for _ in range(c):  # 무작위 노드 선택
          random_node = random.choice(list(adj_list.keys()))
          neighbors = list(adj_list[random_node])
          if len(neighbors) < 2:
              continue

          # 무작위로 이웃 중 두 개의 노드를 선택하고 p 확률로 연결
          node_a, node_b = random.sample(neighbors, 2)
          if node_b not in adj_list[node_a] and random.random() < p:
              adj_list[node_a].add(node_b)
              adj_list[node_b].add(node_a)

              # 엣지 추가 후 degree 업데이트
              degrees[node_a] += 1
              degrees[node_b] += 1
```

알고리즘 6. p 확률로 이웃간 edge 추가 알고리즘

알고리즘 6 은 무작위 노드를 선택한 뒤, 이웃 노드를 무작위로 2 개 고르고 p 확률로 edge 를 연결시킨다. 이는 triangle 형태로 연결시키는 것을 의미하고 clustering coefficient 를 증가시킨다. edge 연결이 성공한 경우 degree 를 업데이트한다.

2-2. $n = 10000$ 및 $c = \{1, 2, 3, 4, 5\}$일 때, 변형 방법 적용 및 $p = \{0.1, 0.3, 0.5\}$ 에 따른 분석

BA 모델은 clustering coefficient 가 낮은 편이다. 새로운 노드를 추가할 때, degree 가 큰 노드끼리와 연결할 가능성이 높은데, 이는 기존노드들과 triangle 형태로 연결되기 보단, 방사형으로 연결된 가능성이 높기 때문이다. 표 2 는 변형 방법 (알고리즘 6)을 적용한 후 비교 표이다. 변형 방법의 경우, c 에 관계없이 p 의 확률이 증가함에 따라 α는 감소하는 경향을 보였다. 이는 변형 방법이 degree 에 상관 없이 edge 를 전체적으로 추가하기 때문에 degree 가 높은 노드들이 증가하기도 하며, 분포가 smoothing 시키기 때문이다, 이는 그림 2 에서 확인할 수 있다. c 와 p 는 동일하게 clustering coefficient 를 증가시키며, c=5, p=0.5 일 때 최대

0.0189 의 값을 기록했다. 또, 변형방법을 증가한 경우, p 가 낮게 설정된 경우 diameter 가 되려 증가하기도 하며 p 가 어느정도 높게 설정된 경우 감소하기도 했다. p 가 낮은 경우, 새로운 에지가 추가되면서 기존에 도달하지 못했던 경로들이 연결되어 더 긴 경로를 갖는 노드 쌍이 생겨났기 때문이며, p 가 충분히 높게 설정된 경우, 최단 경로를 줄이는 edge 가 많이 생겨났기 때문이라고 추측된다.

표 2. 기존 방법과 변형 방법과의 비교 표

방법	c	p	$\alpha \pm error$	Clustering coefficient	Diameter
기존	1	-	1.942±0.0133	0.0030	17
변형	1	0.1	1.927±0.0092	0.0043	19
	1	0.3	1.900±0.0090	0.0092	19
	1	0.5	1.880±0.0088	0.0110	17
기존	2	-	2.398±0.0197	0.0033	8
변형	2	0.1	2.345±0.0134	0.0110	8
	2	0.3	2.262±0.0126	0.0132	8
	2	0.5	2.220±0.0120	0.0152	7
기존	3	-	2.589±0.0224	0.0036	7
변형	3	0.1	2.509±0.0150	0.0076	7
	3	0.3	2.394±0.0139	0.0127	6
	3	0.5	2.319±0.0131	0.0171	6
기존	4		2.678±0.0237	0.0044	6
변형	4	0.1	2.577±0.0157	0.0076	6
	4	0.3	2.442±0.0144	0.0136	6
	4	0.5	2.359±0.0135	0.0182	5
기존	5	-	2.736±0.0245	0.0050	5
변형	5	0.1	2.624±0.0162	0.0080	6
	5	0.3	2.487±0.0148	0.0140	5
	5	0.5	2.379±0.0137	**0.0189**	5

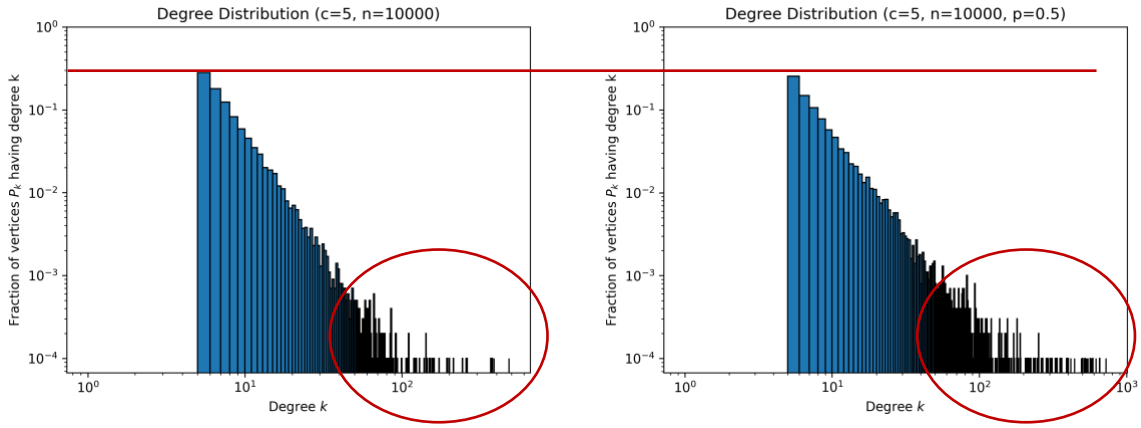


그림 8. 비교방법 전/후 ($p = 0.5$) degree distribution 비교 plot