

经济管理学院

# 课程报告

(复杂网络与社会计算)

题目: week7 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 4 月 10 日

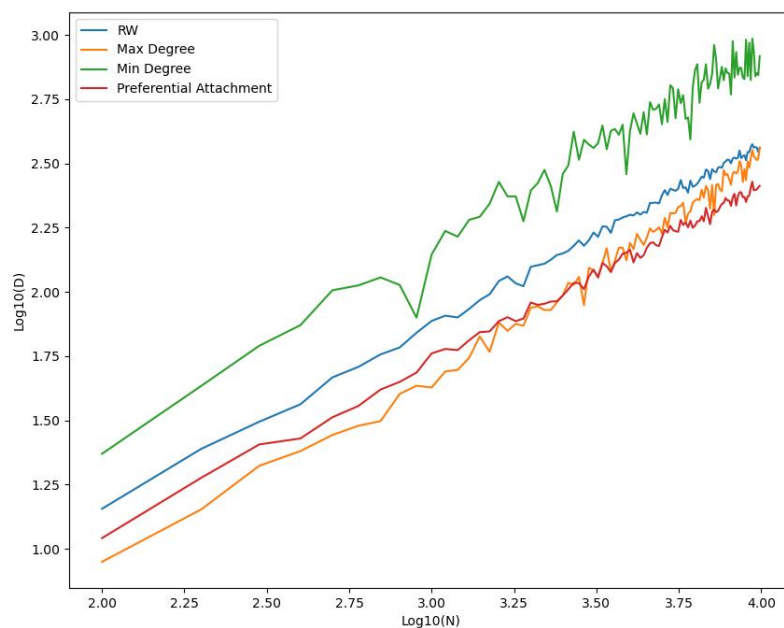


北京航空航天大学  
BEIHANG UNIVERSITY

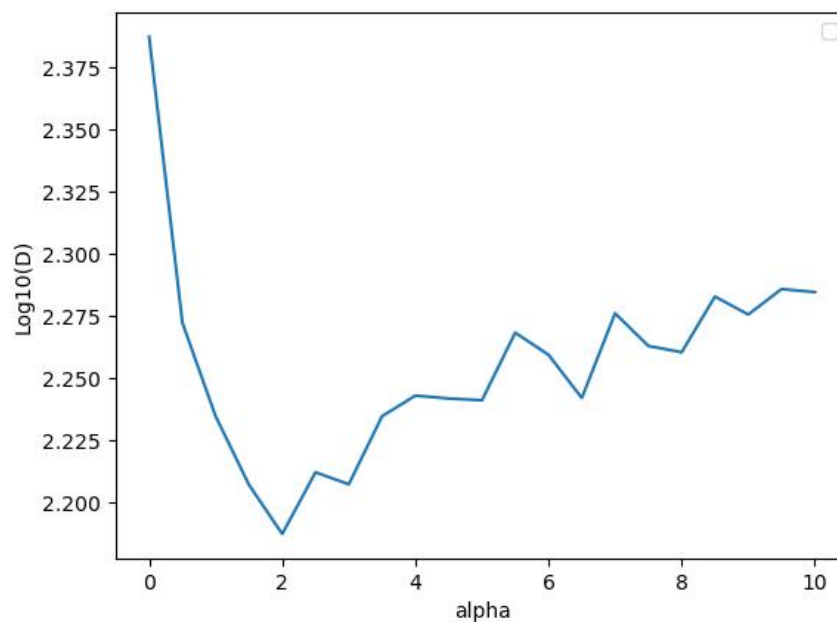
作业要求：

- 2. 生成不同规模  $N$  的BA网络，分别尝试用随机游走，最大度策略，最小度策略（每次选度最小的邻节点）以及优先附着（度为  $k_i$  的节点被选中的概率为  $\frac{k_i}{\sum_j k_j}$ ）来估计网络的平均路径长度  $D$ ，并观察不同方法下，其随着  $N$  的变化趋势。
- 3. 如果将2中优先附着的公式修改为  $\frac{k_i^\alpha}{\sum_j k_j^\alpha}$ ，讨论不同  $\alpha$  时，对于一个特定的  $N$ ，所估计的  $D$  如何变化。

Task1:



Task2:



完整代码如下：

Task1:

```
1. import networkx as nx
2. import numpy as np
3. import itertools
4. import random
5. import pandas as pd
6.
7. # 创建 BA 网络
8. def create_ba_network(n, m):
9.     # n: 网络中节点数量
10.    # m: 附加到每个新节点的现有节点数量
11.    return nx.barabasi_albert_graph(n, m)
12.
13. # 最大度策略
14. def max_degree_path(G, source, target, degrees):
15.     # 初始化路径
16.     path = [source]
17.     current_node = source
18.     while current_node != target:
19.         # 获取当前节点的邻居
20.         neighbors = list(G.neighbors(current_node))
21.
22.         # 过滤掉已经访问过的邻居
23.         neighbors = [node for node in neighbors if node not in path]
24.
25.         # 如果没有邻居或所有邻居都已经检查过则退出循环
26.         if len(neighbors) == 0:
27.             break
28.
29.         # 如果邻居中包含目标节点，则直接返回路径
30.         elif target in neighbors:
31.             path.append(target)
32.             return path
33.
34.         # 否则选择度最大的邻居作为下一个节点
35.         else:
36.             max_degree_neighbor = max(neighbors, key=lambda x: degrees[x])
37.             path.append(max_degree_neighbor)
38.             current_node = max_degree_neighbor
39.     return path
40.
41. # 最小度策略
42. def min_degree_path(G, source, target, degrees):
```

```

43.     # 初始化路径
44.     path = [source]
45.     current_node = source
46.     while current_node != target:
47.         # 获取当前节点的邻居
48.         neighbors = list(G.neighbors(current_node))
49.
50.         # 过滤掉已经访问过的邻居
51.         neighbors = [node for node in neighbors if node not in path]
52.
53.         # 如果没有邻居或所有邻居都已经检查过则退出循环
54.         if len(neighbors) == 0:
55.             break
56.
57.         # 如果邻居中包含目标节点，则直接返回路径
58.         elif target in neighbors:
59.             path.append(target)
60.             return path
61.
62.         # 否则选择度最小的邻居作为下一个节点
63.         else:
64.             min_degree_neighbor = min(neighbors, key=lambda x: degrees[x])
65.             path.append(min_degree_neighbor)
66.             current_node = min_degree_neighbor
67.     return path
68.
69. # 随机游走
70. def RW_path(G, source, target):
71.     # 初始化路径
72.     path = [source]
73.     current_node = source
74.     while current_node != target:
75.         # 获取当前节点的邻居
76.         neighbors = list(G.neighbors(current_node))
77.
78.         # 过滤掉已经访问过的邻居
79.         neighbors = [node for node in neighbors if node not in path]
80.
81.         # 如果没有邻居或所有邻居都已经检查过则退出循环
82.         if len(neighbors) == 0:
83.             break
84.
85.         # 如果邻居中包含目标节点，则直接返回路径
86.         elif target in neighbors:

```

```

87.         path.append(target)
88.         return path
89.
90.         # 否则随机选取一个邻居作为下一个节点
91.     else:
92.         rw_node = random.choice(neighbors)
93.         path.append(rw_node)
94.         current_node = rw_node
95.     return path
96.
97. #优先附着
98. def preferential_attachment_path(G, source, target,degrees):
99.     # 初始化路径
100.    path = [source]
101.    current_node = source
102.    while current_node != target:
103.        # 获取当前节点的邻居
104.        neighbors = list(G.neighbors(current_node))
105.
106.        # 过滤掉已经访问过的邻居
107.        neighbors = [node for node in neighbors if node not in path]
108.
109.        # 如果没有邻居或所有邻居都已经检查过则退出循环
110.        if len(neighbors) == 0:
111.            break
112.
113.        # 如果邻居中包含目标节点，则直接返回路径
114.        elif target in neighbors:
115.            path.append(target)
116.            return path
117.
118.        # 否则随机选取一个邻居作为下一个节点
119.    else:
120.        # 计算每个邻居节点被选中的概率（度数占比）
121.        degree = [degrees[node] for node in neighbors]
122.        total_degree = sum(degree)
123.        probabilities = [deg / total_degree for deg in degree]
124.
125.        pa_node = np.random.choice(neighbors, p=probabilities)
126.        path.append(pa_node)
127.        current_node = pa_node
128.    return path
129.
130. # 估计平均路径长度

```

```

131. def estimate_average_path_length(G):
132.     # 随机获取一千个节点对
133.     node_pairs = random.sample(list(itertools.combinations(G.nodes(), 2)),1000)
134.     degrees = dict(G.degree())
135.
136.     # 初始化路径长度列表
137.     path_lengths1 = []
138.     path_lengths2 = []
139.     path_lengths3 = []
140.     path_lengths4 = []
141.
142.     # 遍历所有节点对
143.     for source, target in node_pairs:
144.         path1 = RW_path(G,source,target)
145.         path2 = max_degree_path(G,source,target,degrees)
146.         path3 = min_degree_path(G,source,target,degrees)
147.         path4 = preferential_attachment_path(G, source, target,degrees)
148.
149.         path_lengths1.append(len(path1) - 1)
150.         path_lengths2.append(len(path2) - 1)
151.         path_lengths3.append(len(path3) - 1)
152.         path_lengths4.append(len(path4) - 1)
153.
154.     # 计算平均路径长度
155.     average_path_length1 = np.mean(path_lengths1)
156.     average_path_length2 = np.mean(path_lengths2)
157.     average_path_length3 = np.mean(path_lengths3)
158.     average_path_length4 = np.mean(path_lengths4)
159.     return average_path_length1, average_path_length2, average_path_length3, average_path_length4
160.
161. def main():
162.     m = 3
163.     data = []
164.     for i in range(1,100):
165.         print(i)
166.         n = i*100
167.         G = create_ba_network(n,m)
168.         RW, max_degree, min_degree, preferential_attachment = estimate_average_path_length(G)
169.         print(RW,max_degree,min_degree,preferential_attachment)
170.         data.append([RW, max_degree, min_degree, preferential_attachment])
171.
172.     df = pd.DataFrame(data, columns=['RW', 'max_degree', 'min_degree', 'preferential_attachment'])

```

```
173.     df.to_excel("C:\\Users\\范春\\Desktop\\week7\\results1.xlsx",index=False)
174.
175. if __name__=="__main__":
176.     main()
```

## Task2:

```
1.  import networkx as nx
2.  import numpy as np
3.  import itertools
4.  import random
5.  import pandas as pd
6.
7.  # 创建 BA 网络
8.  def create_ba_network(n, m):
9.      # n: 网络中节点数量
10.     # m: 附加到每个新节点的现有节点数量
11.     return nx.barabasi_albert_graph(n, m)
12.
13. # 优先附着
14. def preferential_attachment_path(G, source, target,degrees,alpha):
15.     # 初始化路径
16.     path = [source]
17.     current_node = source
18.     while current_node != target:
19.         # 获取当前节点的邻居
20.         neighbors = list(G.neighbors(current_node))
21.
22.         # 过滤掉已经访问过的邻居
23.         neighbors = [node for node in neighbors if node not in path]
24.
25.         # 如果没有邻居或所有邻居都已经检查过则退出循环
26.         if len(neighbors) == 0:
27.             break
28.
29.         # 如果邻居中包含目标节点，则直接返回路径
30.         elif target in neighbors:
31.             path.append(target)
32.             return path
33.
34.         # 否则随机选取一个邻居作为下一个节点
35.         else:
36.             # 计算每个邻居节点被选中的概率（度数占比）
37.             degree = [degrees[node]**alpha for node in neighbors]
38.             total_degree = sum(degree)
```

```

39.         probabilities = [deg / total_degree for deg in degree]
40.
41.         pa_node = np.random.choice(neighbors, p=probabilities)
42.         path.append(pa_node)
43.         current_node = pa_node
44.     return path
45.
46. # 估计平均路径长度
47. def estimate_average_path_length(G, alpha):
48.     # 随机获取一千个节点对
49.     node_pairs = random.sample(list(itertools.combinations(G.nodes(), 2)), 1000)
50.     degrees = dict(G.degree())
51.
52.     # 初始化路径长度列表
53.     path_lengths = []
54.
55.     # 遍历所有节点对
56.     for source, target in node_pairs:
57.         path = preferential_attachment_path(G, source, target, degrees, alpha)
58.         path_lengths.append(len(path) - 1)
59.
60.     # 计算平均路径长度
61.     average_path_length = np.mean(path_lengths)
62.     return average_path_length
63.
64. def main():
65.     m = 3
66.     n = 5000
67.     G = create_ba_network(n, m)
68.     results = []
69.     for item in np.arange(0, 10.25, 0.5):
70.         print(item)
71.         average_path_length = estimate_average_path_length(G, item)
72.         results.append((item, average_path_length))
73.
74.     df = pd.DataFrame(results, columns=['item', 'average_path_length'])
75.
76.     df.to_excel("C:\\Users\\范春\\Desktop\\week7\\results2.xlsx", index=False)
77.
78. if __name__ == "__main__":
79.     main()

```