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

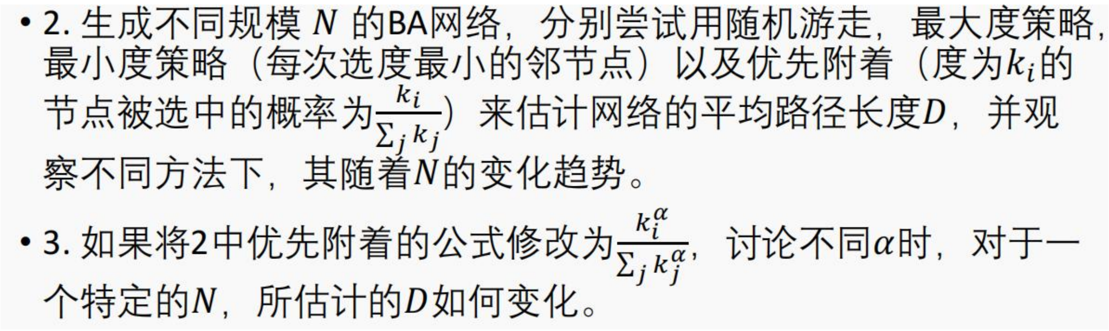
学生姓名：范春

学　　号：21377061

2024 年 4 月 10 日

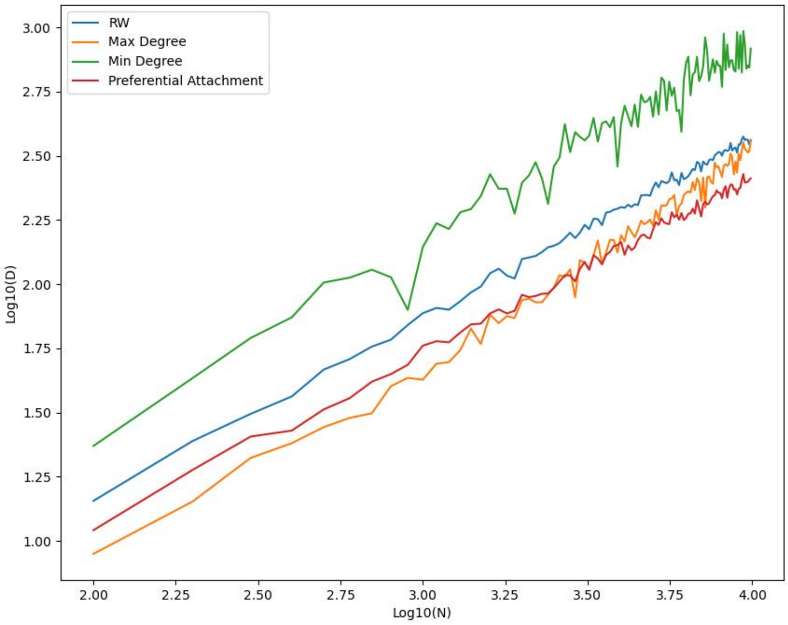
北京航空航天大學
BEIHANG UNIVERSITY

作业要求：

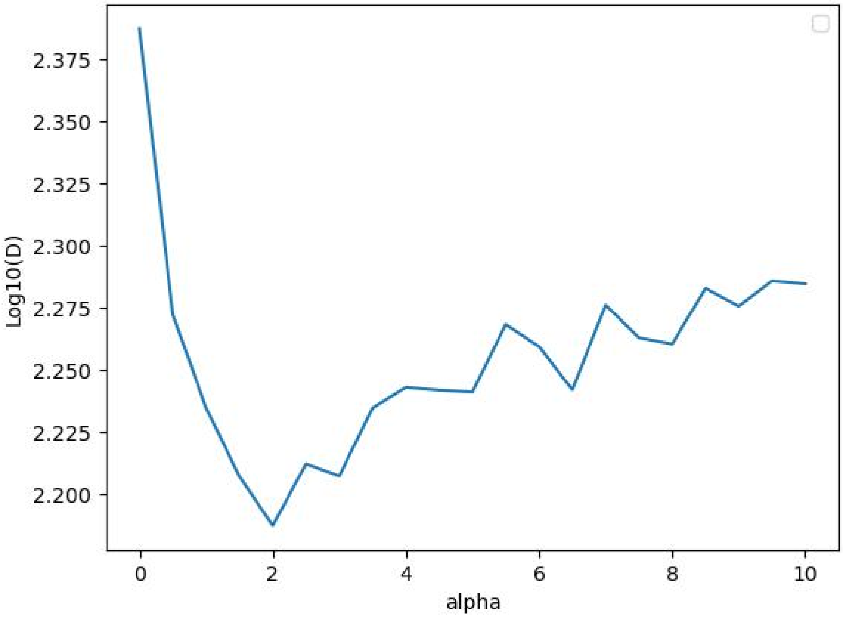


- 2. 生成不同规模 N 的BA网络，分别尝试用随机游走，最大度策略，最小度策略（每次选度最小的邻节点）以及优先附着（度为k_i的节点被选中的概率为$\frac{k_i}{\sum_j k_j}$）来估计网络的平均路径长度D，并观察不同方法下，其随着N的变化趋势。
- 3. 如果将2中优先附着的公式修改为$\frac{k_i^\alpha}{\sum_j k_j^\alpha}$，讨论不同$\alpha$时，对于一个特定的$N$，所估计的$D$如何变化。

Task1:



Task2:



完整代码如下：

Task1:

```python
import networkx as nx
import numpy as np
import itertools
import random
import pandas as pd

# 创建 BA 网络
def create_ba_network(n, m):
    # n: 网络中节点数量
    # m: 附加到每个新节点的现有节点数量
    return nx.barabasi_albert_graph(n, m)

# 最大度策略
def max_degree_path(G, source, target, degrees):
    # 初始化路径
    path = [source]
    current_node = source
    while current_node != target:
        # 获取当前节点的邻居
        neighbors = list(G.neighbors(current_node))

        # 过滤掉已经访问过的邻居
        neighbors = [node for node in neighbors if node not in path]

        # 如果没有邻居或所有邻居都已经检查过则退出循环
        if len(neighbors) == 0:
            break

        # 如果邻居中包含目标节点，则直接返回路径
        elif target in neighbors:
            path.append(target)
            return path

        # 否则选择度最大的邻居作为下一个节点
        else:
            max_degree_neighbor = max(neighbors, key=lambda x: degrees[x])
            path.append(max_degree_neighbor)
            current_node = max_degree_neighbor
    return path

# 最小度策略
def min_degree_path(G, source, target, degrees):
    # 初始化路径
    path = [source]
    current_node = source
    while current_node != target:
        # 获取当前节点的邻居
        neighbors = list(G.neighbors(current_node))

        # 过滤掉已经访问过的邻居
        neighbors = [node for node in neighbors if node not in path]

        # 如果没有邻居或所有邻居都已经检查过则退出循环
        if len(neighbors) == 0:
            break

        # 如果邻居中包含目标节点，则直接返回路径
        elif target in neighbors:
            path.append(target)
            return path

        # 否则选择度最小的邻居作为下一个节点
        else:
            min_degree_neighbor = min(neighbors, key=lambda x: degrees[x])
            path.append(min_degree_neighbor)
            current_node = min_degree_neighbor
    return path

# 随机游走
def RW_path(G, source, target):
    # 初始化路径
    path = [source]
    current_node = source
    while current_node != target:
        # 获取当前节点的邻居
        neighbors = list(G.neighbors(current_node))

        # 过滤掉已经访问过的邻居
        neighbors = [node for node in neighbors if node not in path]

        # 如果没有邻居或所有邻居都已经检查过则退出循环
        if len(neighbors) == 0:
            break

        # 如果邻居中包含目标节点，则直接返回路径
        elif target in neighbors:
            path.append(target)
            return path

        # 否则随机选取一个邻居作为下一个节点
        else:
            rw_node = random.choice(neighbors)
            path.append(rw_node)
            current_node = rw_node
    return path

#优先附着
def preferential_attachment_path(G, source, target,degrees):
    # 初始化路径
    path = [source]
    current_node = source
    while current_node != target:
        # 获取当前节点的邻居
        neighbors = list(G.neighbors(current_node))

        # 过滤掉已经访问过的邻居
        neighbors = [node for node in neighbors if node not in path]

        # 如果没有邻居或所有邻居都已经检查过则退出循环
        if len(neighbors) == 0:
            break

        # 如果邻居中包含目标节点，则直接返回路径
        elif target in neighbors:
            path.append(target)
            return path

        # 否则随机选取一个邻居作为下一个节点
        else:
            # 计算每个邻居节点被选中的概率（度数占比）
            degree = [degrees[node] for node in neighbors]
            total_degree = sum(degree)
            probabilities = [deg / total_degree for deg in degree]

            pa_node = np.random.choice(neighbors, p=probabilities)
            path.append(pa_node)
            current_node = pa_node
    return path

# 估计平均路径长度
```

```python
def estimate_average_path_length(G):
    # 随机获取一千个节点对
    node_pairs = random.sample(list(itertools.combinations(G.nodes(), 2)),1000)
    degrees = dict(G.degree())

    # 初始化路径长度列表
    path_lengths1 = []
    path_lengths2 = []
    path_lengths3 = []
    path_lengths4 = []

    # 遍历所有节点对
    for source, target in node_pairs:
        path1 = RW_path(G,source,target)
        path2 = max_degree_path(G,source,target,degrees)
        path3 = min_degree_path(G,source,target,degrees)
        path4 = preferential_attachment_path(G, source, target,degrees)

        path_lengths1.append(len(path1) - 1)
        path_lengths2.append(len(path2) - 1)
        path_lengths3.append(len(path3) - 1)
        path_lengths4.append(len(path4) - 1)

    # 计算平均路径长度
    average_path_length1 = np.mean(path_lengths1)
    average_path_length2 = np.mean(path_lengths2)
    average_path_length3 = np.mean(path_lengths3)
    average_path_length4 = np.mean(path_lengths4)
    return average_path_length1, average_path_length2, average_path_length3, average_path_length4

def main():
    m = 3
    data = []
    for i in range(1,100):
        print(i)
        n = i*100
        G = create_ba_network(n,m)
        RW, max_degree, min_degree, preferential_attachment = estimate_average_path_length(G)
        print(RW,max_degree,min_degree,preferential_attachment)
        data.append([RW, max_degree, min_degree, preferential_attachment])

    df = pd.DataFrame(data, columns=['RW', 'max_degree', 'min_degree', 'preferential_attachment'])
    df.to_excel("C:\\Users\\范春\\Desktop\\week7\\results1.xlsx",index=False)

if __name__=="__main__":
    main()
```

Task2:

```
import networkx as nx
import numpy as np
import itertools
import random
import pandas as pd

# 创建 BA 网络
def create_ba_network(n, m):
    # n: 网络中节点数量
    # m: 附加到每个新节点的现有节点数量
    return nx.barabasi_albert_graph(n, m)

#优先附着
def preferential_attachment_path(G, source, target,degrees,alpha):
    # 初始化路径
    path = [source]
    current_node = source
    while current_node != target:
        # 获取当前节点的邻居
        neighbors = list(G.neighbors(current_node))

        # 过滤掉已经访问过的邻居
        neighbors = [node for node in neighbors if node not in path]

        # 如果没有邻居或所有邻居都已经检查过则退出循环
        if len(neighbors) == 0:
            break

        # 如果邻居中包含目标节点，则直接返回路径
        elif target in neighbors:
            path.append(target)
            return path

        # 否则随机选取一个邻居作为下一个节点
        else:
            # 计算每个邻居节点被选中的概率（度数占比）
            degree = [degrees[node]**alpha for node in neighbors]
            total_degree = sum(degree)
            probabilities = [deg / total_degree for deg in degree]

            pa_node = np.random.choice(neighbors, p=probabilities)
            path.append(pa_node)
            current_node = pa_node
    return path

# 估计平均路径长度
def estimate_average_path_length(G, alpha):
    # 随机获取一千个节点对
    node_pairs = random.sample(list(itertools.combinations(G.nodes(), 2)),1000)
    degrees = dict(G.degree())

    # 初始化路径长度列表
    path_lengths = []

    # 遍历所有节点对
    for source, target in node_pairs:
        path = preferential_attachment_path(G, source, target,degrees, alpha)
        path_lengths.append(len(path) - 1)

    # 计算平均路径长度
    average_path_length = np.mean(path_lengths)
    return average_path_length

def main():
    m = 3
    n = 5000
    G = create_ba_network(n,m)
    results = []
    for item in np.arange(0,10.25,0.5):
        print(item)
        average_path_length = estimate_average_path_length(G, item)
        results.append((item, average_path_length))

    df = pd.DataFrame(results, columns=['item', 'average_path_length'])

    df.to_excel("C:\\Users\\范春\\Desktop\\week7\\results2.xlsx", index=False)

if __name__=="__main__":
    main()
```