# Assignment1

12110615 董更上

2024 年 3 月 19 日

## 1.

首先，生成一个具有 1000 个节点的网络，这里用到的是 networkx 库。根据题目对网络节点的度的描述和要求，对这 1000 个节点进行配置，然后对其进行可视化，生成的网络结构如图 1 所示。

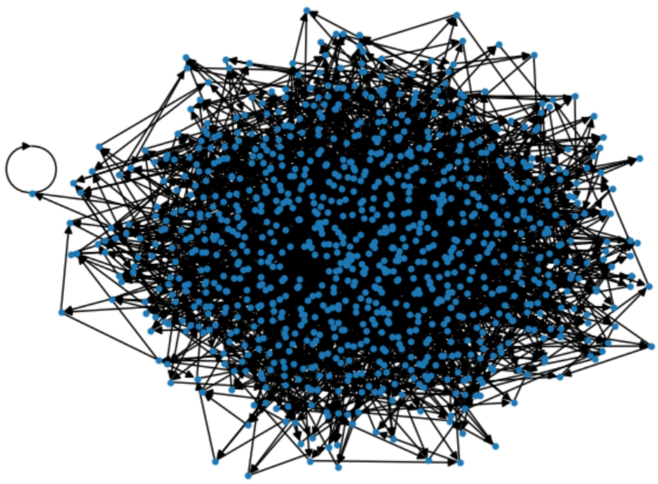

图 1: 网络结构图

然后计算每个节点的 PageRank 值，并对其进行排序，得到排名前十的节点结果如下：

```
Top 10 pages by PageRank:
Page 26: 0.006407657877419998
Page 42: 0.006309533820480254
Page 425: 0.00630924531208407
Page 128: 0.006209825203446937
Page 762: 0.006076218993855884
Page 13: 0.00575856410999918
Page 207: 0.005102287075441184
```

```
Page 529: 0.004891879661096934
Page 455: 0.004639536862101041
Page 617: 0.0044865330310695705
```

接着，为了探究节点的 PageRank 值与其度的相关性，计算 list(degrees.values()) 与 pageranks 两列数据的相关系数，得到的结果为 0.98996。说明网页的重要程度与其关联网页的数量有着极大的关系。

那么 Spam Farm 的网页数量对目标网页 PageRank 有什么影响呢？我接着做了以下实验：设置随机序列 $num\_spam\_pages\_range = range(0, 1001, 50)$ 作为 spam farm 的网页数量，然后随机从原来的 1000 个节点中选取一个节点作为目标节点，令 Spam Farm 节点与目标节点建立双向链接。相关结构如图 2 所示。

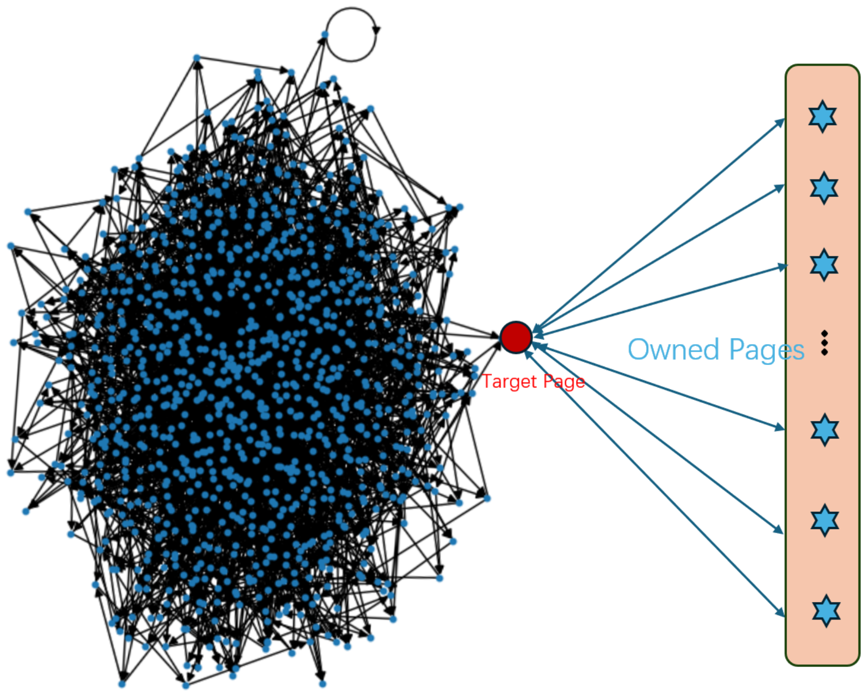


图 2: Spam Farm 架构

然后通过计算，得到了一系列新的 pageranks。而随着 Spam Farm 的网页数量的增加，目标节点的 PageRank 也会稳步上升（图 3）。

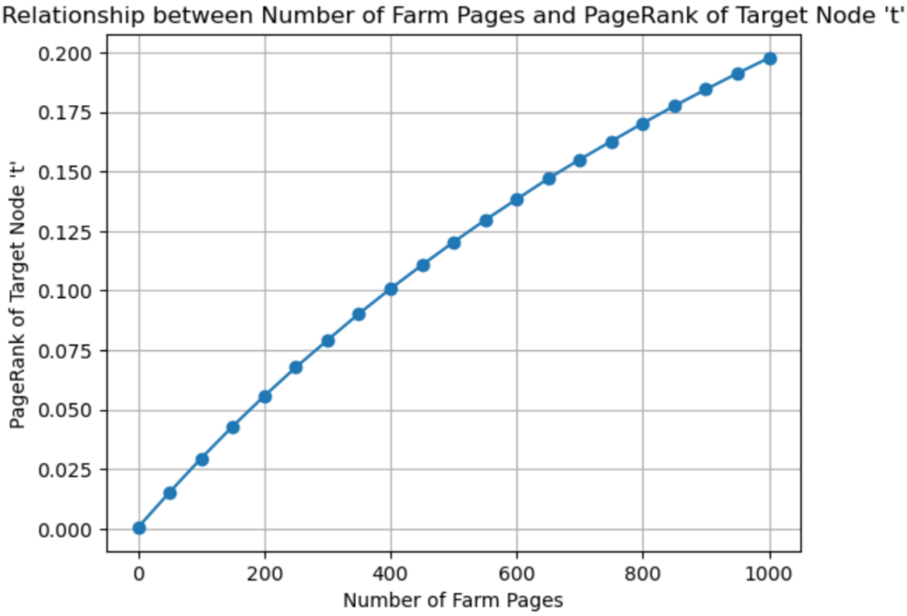


图 3: 目标节点的 PageRank 与 Spam Farm 网页数量的关系

这里结合 spam farm 的原理便很容易理解。假设目标网页 $T$ 的总 PageRank 值为 $y$，则 $y$ 由三部分组成：
Accessible 网页的贡献，设为 $x$；
Web 中所有网页平均分到的贡献：$\frac{(1-\beta)}{n}$，这部分值很小，几乎可以忽略。
Owned 网页的贡献：设有 $m$ 个自有网页，每个自有网页的 PageRank 值为 $\frac{\beta y}{m}+\frac{(1-\beta)}{n}$。
这三部分结合，可以得到：

$$y = x + \beta m\left(\frac{\beta y}{m}+\frac{(1-\beta)}{n}\right) = x + \beta^2 y + \beta(1-\beta)\frac{m}{n}$$

解得：

$$y = \frac{x}{1-\beta^2}+\frac{\beta m}{(1+\beta)n}$$

如果选择 $\beta = 0.85$，那么 $\frac{1}{(1-\beta^2)} = 3.6$，$\frac{\beta}{(1+\beta)} = 0.46$。也就是说，上述结构能够把可达网页的 PageRank 贡献放大到 3.6 倍，并且还可以获得自有网页数目和总网页数目比值 $(m/n)$ 的 46%。对 *num_spam_pages_range* 和 *pageranks* 进行回归拟合，得到的回归系数为 0.000197，这与 0.46 相差甚远，但是，考虑到 n 为 $10^3$ 数量级且随着 $m$ 的数值变化 $n$ 和 $x$ 都会有所改变，所以这个结果还是比较合理的。

## 2.

为了应对 Spam Farm 的作弊行为，TrustRank 这一概念被提出。
Trustrank 的计算步骤如下：

1. 初始化所有节点的 Trustrank 值为 1。

2. 迭代计算每个节点的 Trustrank 值，直到收敛或达到最大迭代次数为止。
3. 对于每个节点，如果它属于信任页面，则其 Trustrank 值保持不变。
4. 如果节点不是信任页面，则其 Trustrank 值由以下方式计算：
   - 将节点的 Trustrank 值初始化为 $\frac{(1-\beta)}{N}$，其中 $N$ 是图中节点的总数，这表示平均分配给每个节点的初始 Trustrank 值。
   - 对于节点的每个邻居，将其 Trustrank 值加上 $\frac{\beta\times\text{trustrank[neighbor]}}{\text{len(list(graph.neighbors(neighbor)))}}$，表示通过邻居节点传递过来的 Trustrank 值。
5. 计算每次迭代后的 Trustrank 值之间的差异，如果差异小于预先设定的容差 tolerance，则停止迭代。
6. 返回最终计算得到的 Trustrank 值。

在 Spam Farm 数量为 500 时，各节点对应的 TrustRank 为 (前 11 个节点)：

```
0: 0.11575965541624894
1: 0.15743536091944468
2: 0.19938644511915665
3: 0.12030224358782396
4: 0.13212101581033076
5: 0.18003629516801795
6: 0.13886847957349077
7: 0.16748264315618477
8: 0.260502783577734
9: 0.08337080653510381
```

如果一个页面的 PageRank 值为 P，TrustRank 为 T，那么网页的 Spam Mass 被定义为：(P - T) / P。Spam Mass 越大，说明此页面被 Spam 的可能性越大。进而就可以通过降低该网页的 PageRank 值来避免作弊行为。在原始 1000 个节点中, 选取 5% 作为 trust pages(去除 target node)。由于选取的 Spam Farm 的页面数量是一系列值，这里不再单独计算 TrustRank 的值。根据 Spam Mass 的定义，计算得到了不同 Spam Farm 的页面数量下对应的目标节点的 Spam Mass 值。结果显示在图 3 中。在 $NumberofFarmPages = 50$ 时, Spam Mass 已经超过了 0.95。可见，通过 TrustRank 算法，可以很容易地检测到 Spam 网页。

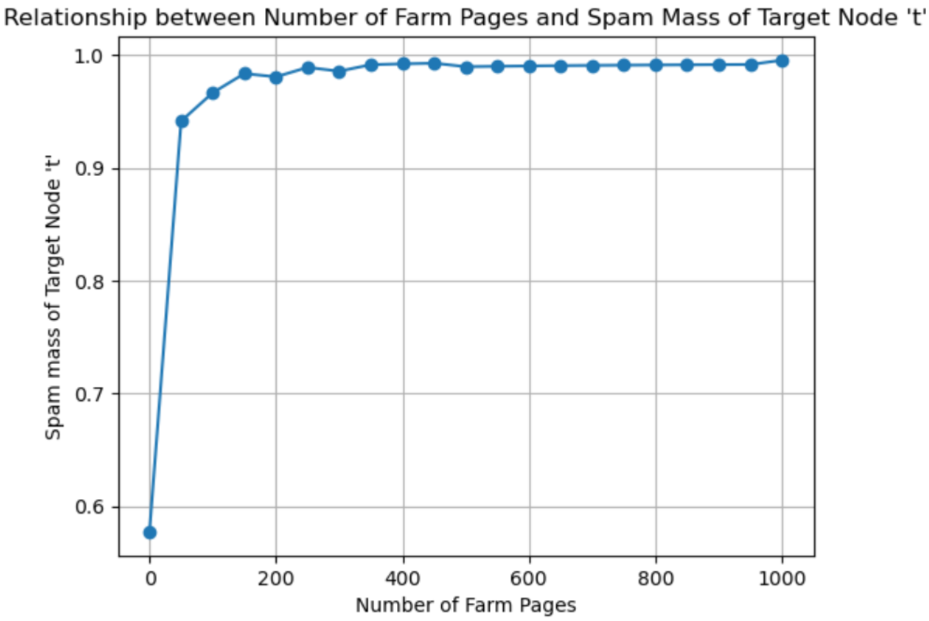


图 4: 目标节点的 Spam Mass 与 Spam Farm 网页数量的关系

## 参考资料

https://blog.csdn.net/weixin_43378396/article/details/90322422

## A 附录：相关代码

```
# %%
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

# Generate directed graph



def generate_graph(n, min_degree, max_degree, power):
    degrees = np.random.zipf(power, n)
    degrees = np.clip(degrees, min_degree, max_degree)
    G = nx.directed_configuration_model(degrees, degrees, create_using=nx.DiGraph)
    G = nx.DiGraph(G)
    return G

# Define PageRank algorithm
def page_rank(G, alpha=0.85, max_iter=100, tol=1.0e-6):
```

```
    # initialize PageRank values
    pagerank = {node: 1 / len(G) for node in G.nodes()}

    # Calculate PageRank values by iteration
    for _ in range(max_iter):
        pagerank_new = {}

        for node in G.nodes():
            pr = (1 - alpha) / len(G)
            for neighbor in G.predecessors(node):
                pr += alpha * pagerank[neighbor] / len(list(G.neighbors(neighbor)))
            pagerank_new[node] = pr

        diff = max(abs(pagerank_new[node] - pagerank[node]) for node in G.nodes())
        if diff < tol:
            break

        pagerank = pagerank_new

    return pagerank


# Spam Farm with num_spam_pages
def spam_farm(G, num_spam_pages):
    spam_nodes = np.random.choice(G.nodes(), num_spam_pages, replace=False)
    return spam_nodes

# Plot graph
def plot_graph(G):
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=False, node_size=10)
    plt.show()

# Generate graph with 1000 nodes, min_degree=2, max_degree=20, power=2.5
graph = generate_graph(n=1000, min_degree=2, max_degree=20, power=2.5)

plot_graph(graph)

pagerank = page_rank(graph, alpha=0.85)
```

```
# Top 10 pages by PageRank
sorted_pagerank = sorted(pagerank.items(), key=lambda x: x[1], reverse=True)
print("Top 10 pages by PageRank:")
for i, (page, rank) in enumerate(sorted_pagerank[:10]):
    print(f"Page {page}: {rank}")

# Calculate the correlation between node PageRank and node degree
degrees = dict(graph.degree())
pageranks = np.array([pagerank[node] for node in graph.nodes()])
correlation = np.corrcoef(list(degrees.values()), pageranks)[0, 1]
print("\n")
print("Correlation between node importance (PageRank) and node degree:", correlation)

# %%
# Chose a random target node
np.random.seed(42)
target_node = np.random.choice(list(graph.nodes()))
print("Target node", target_node)

# define the range of number of spam pages
num_spam_pages_range = list(range(0, 1001, 50))

def compute_pagerank(graph, target_node, spam_farm_pages):
    # Add owned pages to the graph
    graph_spam_farm = graph.copy()

    edges = [(page, target_node) for page in spam_farm_pages]
    edges.extend([(target_node, page) for page in spam_farm_pages])  # Add reverse edges
    graph_spam_farm.add_edges_from(edges)

    pagerank = nx.pagerank(graph_spam_farm)

    return pagerank[target_node]

# Calculate the PageRank of the target node for different number of spam pages
pageranks = []
for num_spam_pages in num_spam_pages_range:
    # Generate spam farm
```

```
    spam_farm_pages = list(range(num_spam_pages))
    # Calculate PageRank of the target node
    pagerank_t = compute_pagerank(graph, target_node, spam_farm_pages)
    pageranks.append(pagerank_t)

# Plot the relationship between the number of spam pages and the PageRank of the target node
plt.plot(num_spam_pages_range, pageranks, marker='o')
plt.xlabel("Number of Farm Pages")
plt.ylabel("PageRank of Target Node 't'")
plt.title("Relationship between Number of Farm Pages and PageRank of Target Node 't'")
plt.grid(True)
plt.show()

# %%
from sklearn.linear_model import LinearRegression

x = np.array(num_spam_pages_range).reshape(-1, 1)
y = np.array(pageranks)
model = LinearRegression()
model.fit(x, y)
slope = model.coef_[0]
print("Slope of regression line:", slope)

# %%
def compute_trustrank_1(graph, trust_pages, damping_factor=0.85, max_iterations=100, toleran
    # Trustrank computation logic here
    trustrank = {node: 1 for node in graph.nodes()}
    for _ in range(max_iterations):
        trustrank_new = {}
        for node in graph.nodes():
            if node in trust_pages:
                trustrank_new[node] = trustrank[node]
            else:
                trustrank_new[node] = (1 - damping_factor) / len(graph.nodes())
                for neighbor in graph.neighbors(node):
                    trustrank_new[node] += damping_factor * trustrank[neighbor] / len(list(g
        diff = max(abs(trustrank_new[node] - trustrank[node]) for node in graph.nodes())
        if diff < tolerance:
            break
```

```
        trustrank = trustrank_new
    return trustrank

# Compute PageRank
def compute_pagerank(graph_spam_farm, trust_pages):
    # Calculate PageRank with trust pages
    pagerank_with_trust = nx.pagerank(graph_spam_farm, personalization=dict.fromkeys(trust_p
    # Calculate PageRank without trust pages
    pagerank_without_trust = nx.pagerank(graph_spam_farm)
    return pagerank_with_trust, pagerank_without_trust

accessible_pages = list(graph.nodes())
accessible_pages.remove(target_node)
num_trust_pages = int(len(accessible_pages) * 0.05)
trust_pages = np.random.choice(accessible_pages, num_trust_pages, replace=False)

spam_masses = []
for num_spam_pages in num_spam_pages_range:
    # Combine the graph
    graph_spam_farm = graph.copy()
    spam_farm_pages = list(range(1000))
    spam_farm_pages = range(len(graph_spam_farm.nodes()), len(graph_spam_farm.nodes()) + nu
    edges = [(page, target_node) for page in spam_farm_pages]
    edges.extend([(target_node, page) for page in spam_farm_pages])  # Add reverse edges
    graph_spam_farm.add_edges_from(edges)

    trustrank = compute_trustrank_1(graph_spam_farm, trust_pages)
    # Calculate PageRank with and without trust pages
    pagerank_with_trust, pagerank_without_trust = compute_pagerank(graph_spam_farm, trust_pa
    # Calculate Spam mass for the target node
    spam_mass = (pagerank_without_trust[target_node] - pagerank_with_trust[target_node]) / p
    spam_masses.append(spam_mass)


# %%
plt.plot(num_spam_pages_range, spam_masses, marker='o')
plt.xlabel("Number of Farm Pages")
plt.ylabel("Spam mass of Target Node 't'")
plt.title("Relationship between Number of Farm Pages and Spam Mass of Target Node 't'")
```

```
plt.grid(True)
plt.show()

# %%
# Calculate the PageRank of the target node for number of spam pages = 500
num_spam_pages_range = 500

# Create a copy of the graph
graph_spam_farm = graph.copy()

# Add 500 nodes to the graph connected to the target_node
spam_farm_pages = range(len(graph_spam_farm.nodes()), len(graph_spam_farm.nodes()) + num_spa
edges = [(page, target_node) for page in spam_farm_pages]
edges.extend([(target_node, page) for page in spam_farm_pages])  # Add reverse edges
graph_spam_farm.add_edges_from(edges)

trustrank = compute_trustrank_1(graph_spam_farm, trust_pages)
# print("Trustrank:", trustrank)
count = 0
for key, value in trustrank.items():
    if count < 10:
        print(f"{key}: {value}")
        count += 1
    else:
        break
```