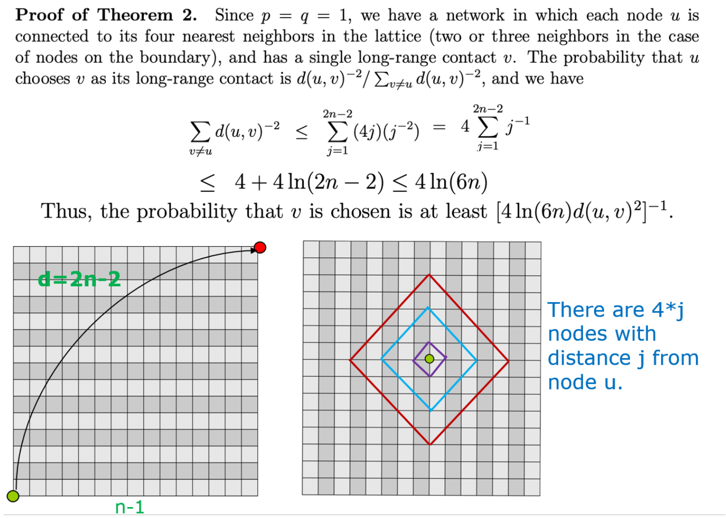
Assignment

12110615 董更上

2024 年 4 月 3 日

1.


Proof of Theorem 2. Since $p = q = 1$, we have a network in which each node u is connected to its four nearest neighbors in the lattice (two or three neighbors in the case of nodes on the boundary), and has a single long-range contact v. The probability that u chooses v as its long-range contact is $d(u,v)^{-2}/\sum_{v\neq u} d(u,v)^{-2}$, and we have

$$\sum_{v\neq u} d(u,v)^{-2} \le \sum_{j=1}^{2n-2}(4j)(j^{-2}) = 4\sum_{j=1}^{2n-2} j^{-1}$$
$$\le 4+4\ln(2n-2) \le 4\ln(6n)$$

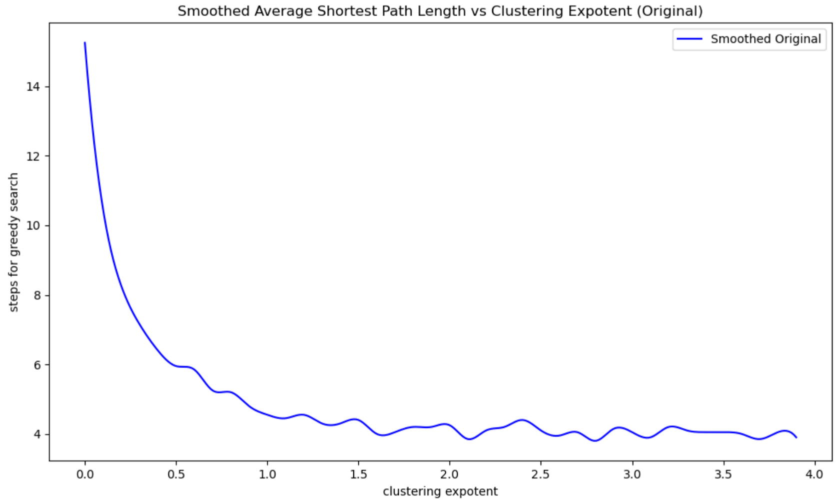
Thus, the probability that v is chosen is at least $[4\ln(6n)d(u,v)^2]^{-1}$.

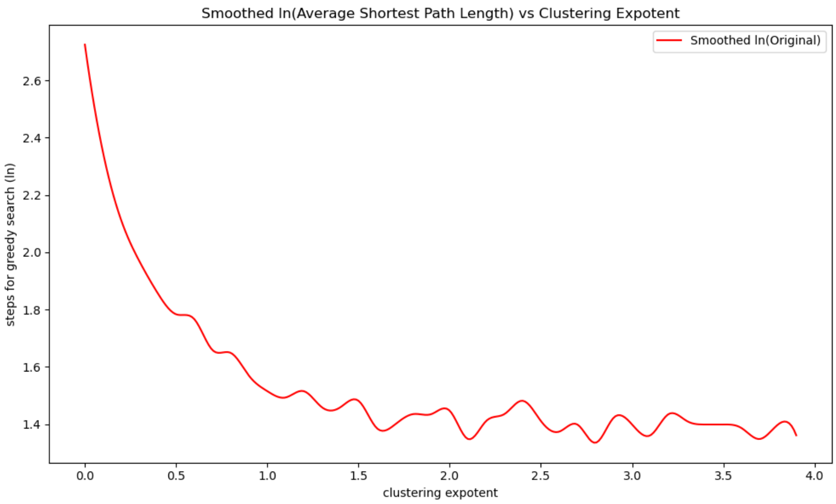
图 1: 定理 2

定理二中提到的是二维图中 p=q=1（p=1 代表节点 p 与欧氏距离为 1 的节点都有连接；q=1 代表一个节点有 1 个长程节点）的场景，在本次作业的模拟中，仍然选择 p=q=1, 但是将网络迁移到一维的链中. 链与二维图不同的是，不同距离对应的节点数从 4j 变为了 2, 同时对于长度为 n 的链，距离的最大种类数为（n-1). 那么类似地，对于节点 u, v 被选做 u 的长程节点的概率为 $d(u,v)^{-r}/\sum_{i!=u} d(u,v)^{-r} \ge d(u,v)^{-r}/\sum_{j=1}^{n-1} 2(\ j^{-r}\ )$. (1)

那么在代码实现中，我们先定义一个节点数为 num_nodes 的链，然后让链中的节点与前后节点相连. 然后，定义函数 generate_long_range_edges, 这个函数取 clustering exponent r 作为参数，在其中定义刚刚不等式（1）得到的归一化常数 normalization_constant（不等式右侧的分母），然后根据长程边的分布概率生成新的长程边添加到原来的链中. 至此，我们已经完成了图的构建. 为了探究节点之间平均步程与 r 之间的关系，接着定义函数 average_shortest_path_length, 它通过在生成的链中随机选取 num_pairs 个点对，计算点对之间的平均距离并返回.

囿于计算机性能,本实验选取 num_nodes = 20000, num_pairs = 20, 同时选取 r = np.arrange(0,4,0.1), 得到了平均步程与 clustering exponent 的 39 组对应关系。关系曲线如下：

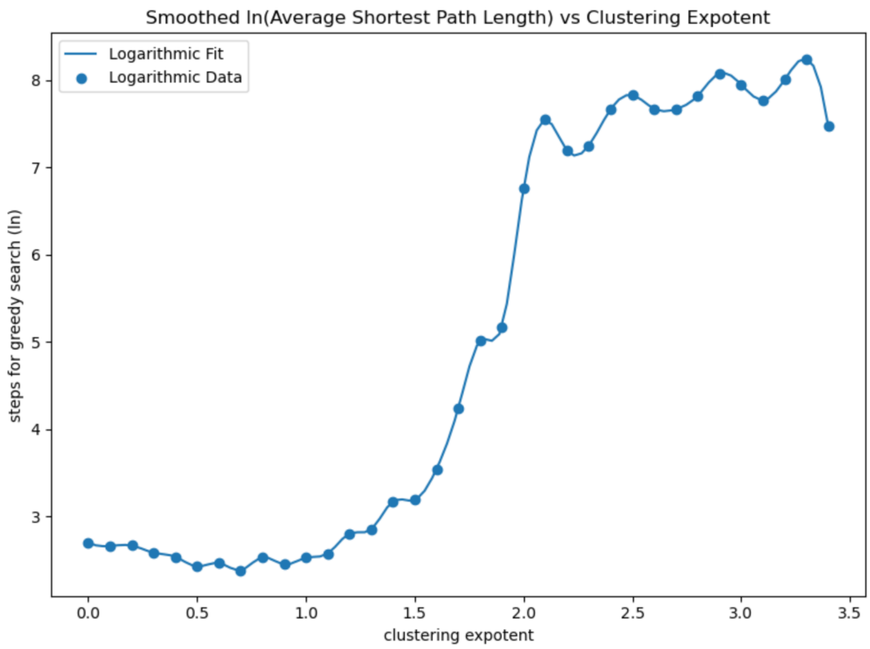




可以看到，平均步程在 r=1.3 时已经趋近最短 (在二维世界中这个数值接近于 2，这里可以猜测一维世界中这个数值接近 1，在这里的最低点，信息传递效率最高，建立的 clustering exponent 模型也最有可能接近现实情况)。但是，遗憾的是，并没有发现预期中的步程回升的现象。经过思考和小范围讨论仍未找到原因。希望在课程中与同学们交流讨论后可以找到问题所在。

************************************** 分割线 **

截止作业提交前，终于发现了问题所在，原来是在进行 for 循环对 r 遍历时没有对 chain 进行初始化，导致链中的边一直增多，并且平均步程趋于收敛。更新代码后关系曲线如下：



所使用代码如下:

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

num_nodes = 10000

def generate_long_range_edges(r):
    edges = []
    normalization_constant = sum([2*(j**-r) for j in range(1, num_nodes)])

    for i in range(num_nodes):
        for j in range(i+1, num_nodes):
            distance = abs(i - j)
            edge_prob = (distance**-r) / normalization_constant

            if random.random() < edge_prob:
                edges.append((i, j))
    return edges

def average_shortest_path_length(chain, num_pairs=15):
    total_path_length = 0
```

```
    for _ in range(num_pairs):
        source = random.choice(list(chain.nodes))
        target = random.choice(list(chain.nodes))
        path_length = nx.shortest_path_length(chain, source=source, target=target)
        total_path_length += path_length

    average_path_length = total_path_length / num_pairs
    return average_path_length

data = []

chain = nx.Graph()

for i in np.arange(0, 3.5, 0.1):
    chain.clear()
    for j in range(num_nodes - 1):
        chain.add_edge(j, j+1)

    long_range_edges = generate_long_range_edges(r=i)
    chain.add_edges_from(long_range_edges)
    average_path_length = average_shortest_path_length(chain)
    data.append((i, average_path_length))
    print(f"r={i}, average shortest path length={average_path_length}")

print(data)


# %%
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

data = np.array(data)

x = data[:, 0]
y = data[:, 1]

y_log = np.log(y)
```

```
plt.figure(figsize=(8, 6))

f_log = interp1d(x, y_log, kind='cubic')

x_new = np.linspace(min(x), max(x), 100)

plt.plot(x_new, f_log(x_new), label='Logarithmic Fit')
plt.scatter(x, y_log, label='Logarithmic Data')
plt.xlabel('clustering expotent')
plt.ylabel('steps for greedy search (ln)')
plt.title('Smoothed ln(Average Shortest Path Length) vs Clustering Expotent')
plt.legend()

plt.tight_layout()
plt.show()
```