哈尔滨工业大学深圳研究生院

复杂网络建模报告

Social Network Analysis of Our Class

姓　　名　　胡文馨

学　　号　　17S151563

报告日期　　20171030

目录

Experimental principle 3

Model thinking 3

1. The original model 3
2. The first improved model 6
3. The second improved model 6
4. The third improved model 9

Network characteristics 11

1.Node-degree distribution 11

2.Average shortest path-length 12

3.Clustering coefficient 13

4.Robustness against intentional attack 14

5.Random attack 18

Experimental principle

Given the desired number of nodes N, the mean degree K , and a special parameter P, satisfying $0 \le \mathrm{P} \le 1$ and $\mathrm{N} \gg \mathrm{K} \gg \ln\mathrm{N} \gg 1$, the model constructs an undirected graph with N nodes and $\frac{NK}{2}$ edges in the following way:

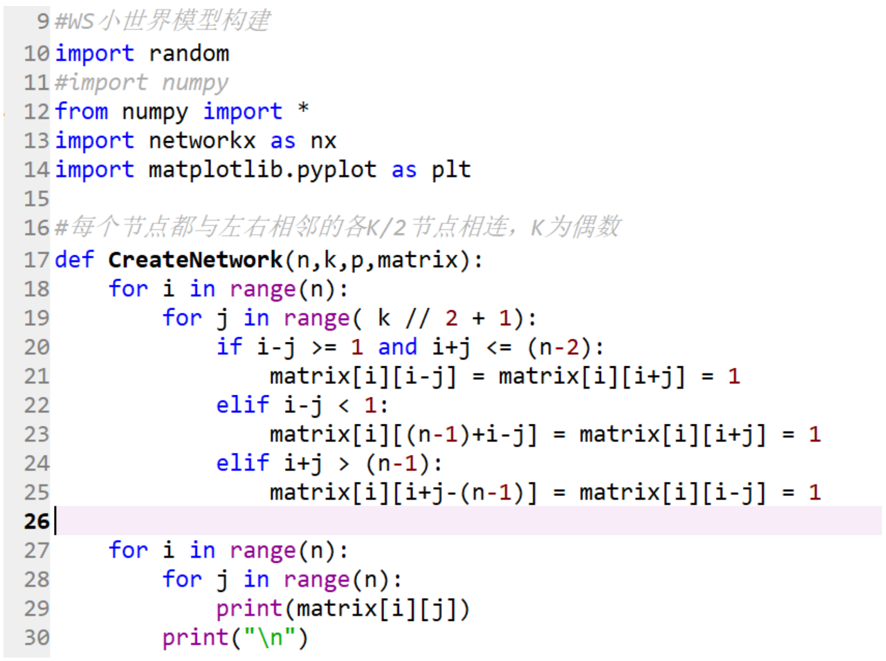
1. Construct a regular ring lattice, a graph with N nodes each connected to K neighbors, K/2 on each side. That is, if the nodes are labeled $n_0 \ldots n_{N-1}$, there is an edge (n_i , n_j) if and only if $0<|i-j|$ mod (N – 1 - $\frac{K}{2}$) $\le \frac{K}{2}$.
2. For every node n_i take every egde (n_i , n_j) with i < j, and rewire it with probability P. Rewiring is done by replacing (n_i , n_j) with (n_i , n_k) where K is chosen with uniform probability from all possible values that avoid self-loops (K $\neq$ i) and link duplication (there is no edge (n_i , n_{k2}) with K2 = K at this point in the algorithm).

Model thinking

1. The original model

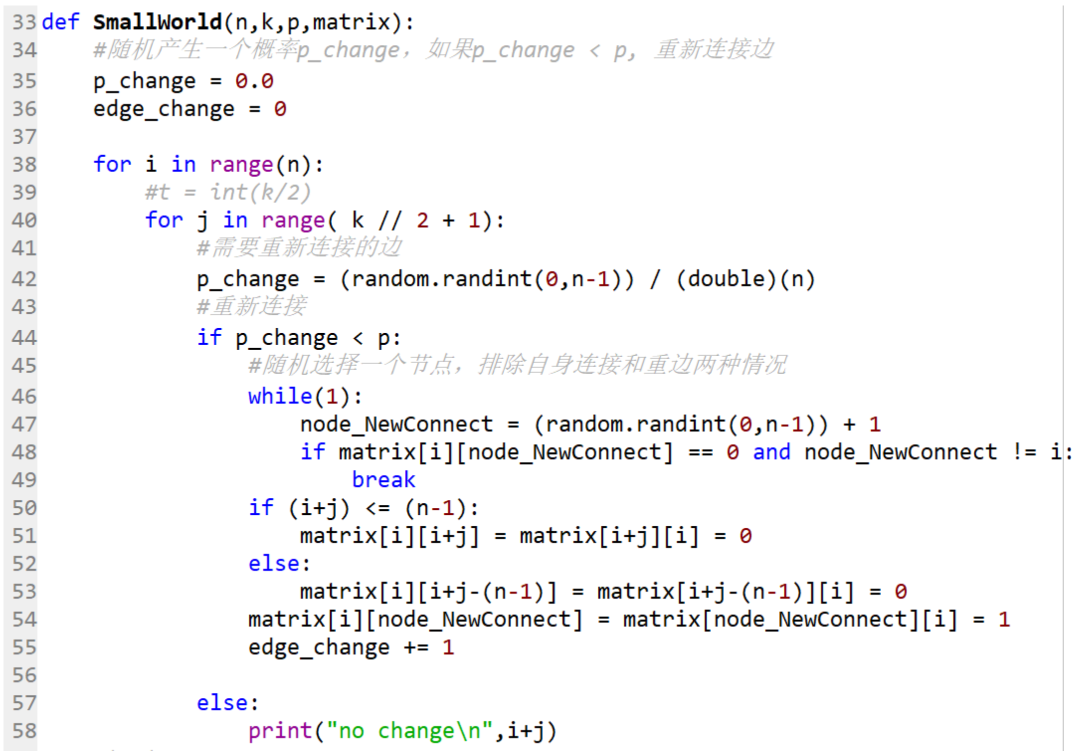
   According to the principle of the algorithm, I try my best to build the model.

   First of all, using two layers for loop nesting to establish a coupled network and the matrix to express the edge.

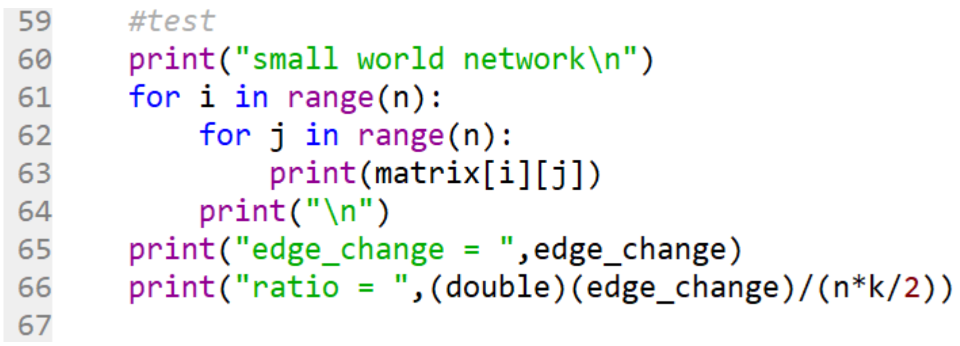


```python
#WS小世界模型构建
import random
#import numpy
from numpy import *
import networkx as nx
import matplotlib.pyplot as plt

#每个节点都与左右相邻的各K/2节点相连，K为偶数
def CreateNetwork(n,k,p,matrix):
    for i in range(n):
        for j in range( k // 2 + 1):
            if i-j >= 1 and i+j <= (n-2):
                matrix[i][i-j] = matrix[i][i+j] = 1
            elif i-j < 1:
                matrix[i][(n-1)+i-j] = matrix[i][i+j] = 1
            elif i+j > (n-1):
                matrix[i][i+j-(n-1)] = matrix[i][i-j] = 1

    for i in range(n):
        for j in range(n):
            print(matrix[i][j])
        print("\n")
```

Second, reconnecting randomly each edge. At the same time, I excluding two conclusions, one is a node connect with itself, the other is two nodes have two even more edges to connect. Through printing the matrix, I find the matrix is symmetrical. And using the test module, we can check clearly the number of changed edges and the ratio.

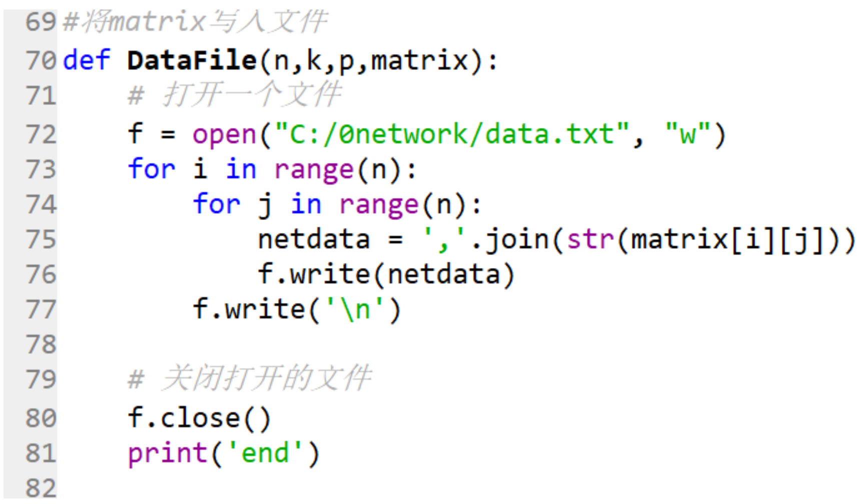


```python
def SmallWorld(n,k,p,matrix):
    #随机产生一个概率p_change, 如果p_change < p, 重新连接边
    p_change = 0.0
    edge_change = 0

    for i in range(n):
        #t = int(k/2)
        for j in range( k // 2 + 1):
            #需要重新连接的边
            p_change = (random.randint(0,n-1)) / (double)(n)
            #重新连接
            if p_change < p:
                #随机选择一个节点，排除自身连接和重边两种情况
                while(1):
                    node_NewConnect = (random.randint(0,n-1)) + 1
                    if matrix[i][node_NewConnect] == 0 and node_NewConnect != i:
                        break
                if (i+j) <= (n-1):
                    matrix[i][i+j] = matrix[i+j][i] = 0
                else:
                    matrix[i][i+j-(n-1)] = matrix[i+j-(n-1)][i] = 0
                matrix[i][node_NewConnect] = matrix[node_NewConnect][i] = 1
                edge_change += 1

            else:
                print("no change\n",i+j)
```



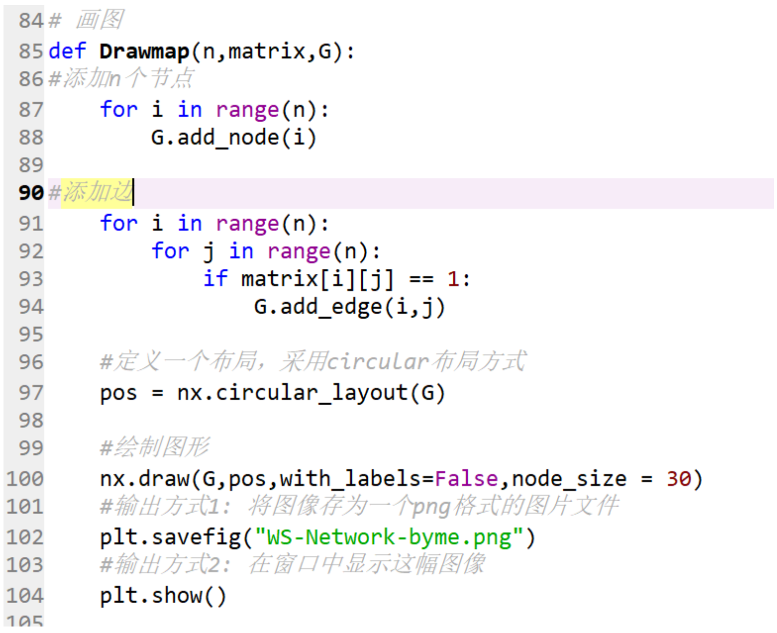
```python
    #test
    print("small world network\n")
    for i in range(n):
        for j in range(n):
            print(matrix[i][j])
        print("\n")
    print("edge_change = ",edge_change)
    print("ratio = ",(double)(edge_change)/(n*k/2))

```

Third, I writing the data to a file to check the matrix and process the file. At that time, I intended to use R language to deal with the graph. But I find the matrix can be translate to the graph in python later. In particularly, I meet too many complex difficults due to I write the algorithm by myself and the style is different with the source code in python networkx.



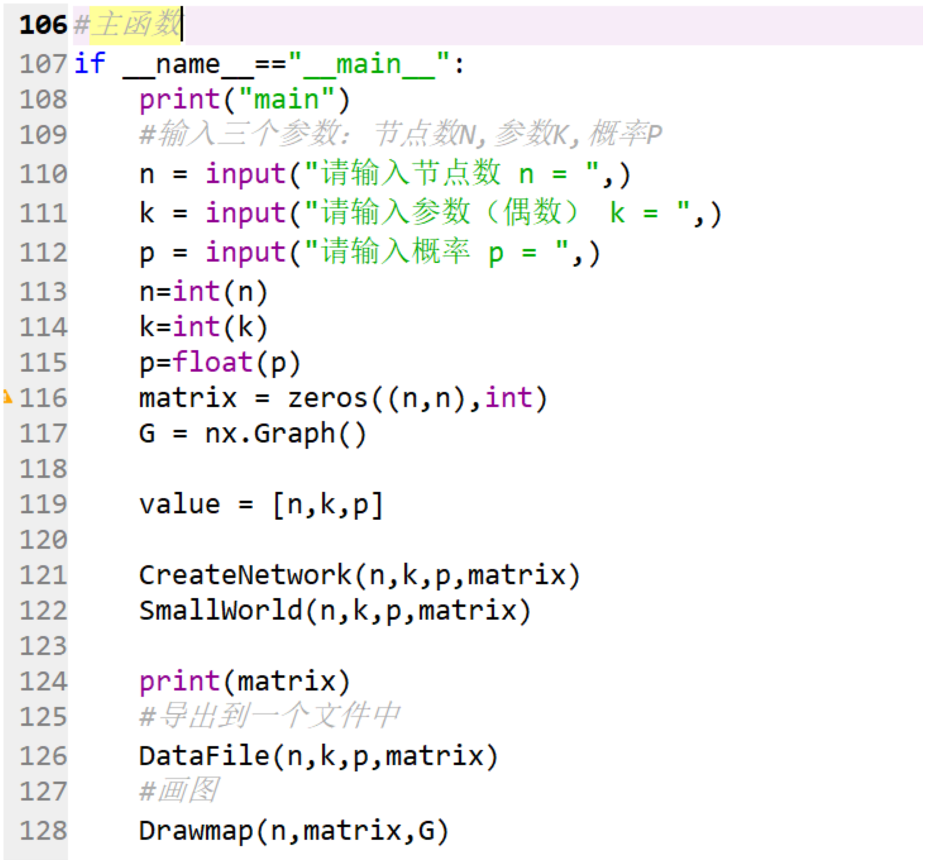
```python
#将matrix写入文件
def DataFile(n,k,p,matrix):
    # 打开一个文件
    f = open("C:/0network/data.txt", "w")
    for i in range(n):
        for j in range(n):
            netdata = ','.join(str(matrix[i][j]))
            f.write(netdata)
        f.write('\n')

    # 关闭打开的文件
    f.close()
    print('end')

```

Fourth, I add the nodes and edges and export the picture.

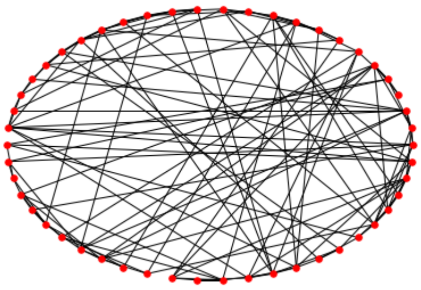


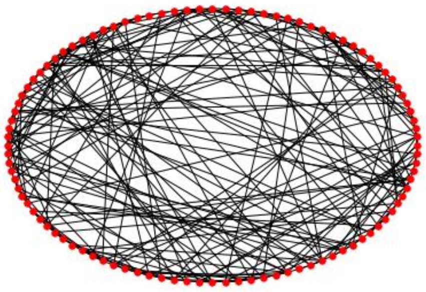
```python
#  画图
def Drawmap(n,matrix,G):
#添加n个节点
    for i in range(n):
        G.add_node(i)

#添加边
    for i in range(n):
        for j in range(n):
            if matrix[i][j] == 1:
                G.add_edge(i,j)

    #定义一个布局，采用circular布局方式
    pos = nx.circular_layout(G)

    #绘制图形
    nx.draw(G,pos,with_labels=False,node_size = 30)
    #输出方式1：将图像存为一个png格式的图片文件
    plt.savefig("WS-Network-byme.png")
    #输出方式2：在窗口中显示这幅图像
    plt.show()
```

Finally, I using the main function to achieve. In particular, it is very important to force conversion type for N,K,P and initialize matrix to zeros.



```python
#主函数
if __name__=="__main__":
    print("main")
    #输入三个参数：节点数N，参数K，概率P
    n = input("请输入节点数 n = ",)
    k = input("请输入参数（偶数） k = ",)
    p = input("请输入概率 p = ",)
    n=int(n)
    k=int(k)
    p=float(p)
    matrix = zeros((n,n),int)
    G = nx.Graph()

    value = [n,k,p]

    CreateNetwork(n,k,p,matrix)
    SmallWorld(n,k,p,matrix)

    print(matrix)
    #导出到一个文件中
    DataFile(n,k,p,matrix)
    #画图
    Drawmap(n,matrix,G)
```

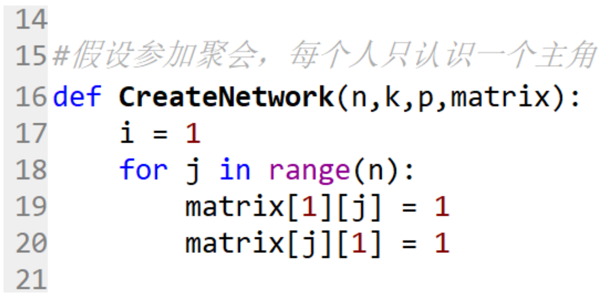




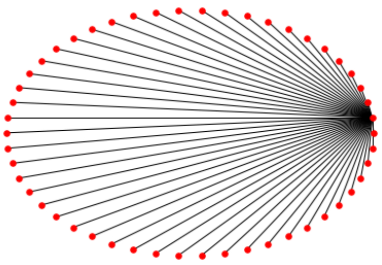
N = 50   K = 4   P = 0.4   N = 100   K = 4   P = 0.4

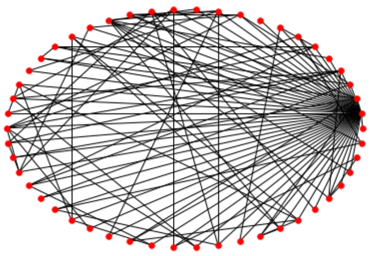
2. The first improved model

The first improved model is to change the initial connecting way. Hypothesis you take part in a birthday party, everyone must know the host and they maybe do not know other people. Extremly everone only know the host, so the host is the social center.



```

#假设参加聚会，每个人只认识一个主角
def CreateNetwork(n,k,p,matrix):
    i = 1
    for j in range(n):
        matrix[1][j] = 1
        matrix[j][1] = 1

```





N = 50   K = 4   P = 0.4   N = 50   K = 4   P = 0.4

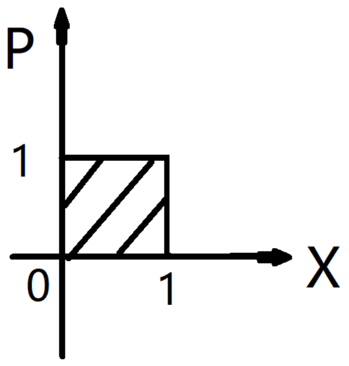
Edge change = 65

change ratio = 0.65

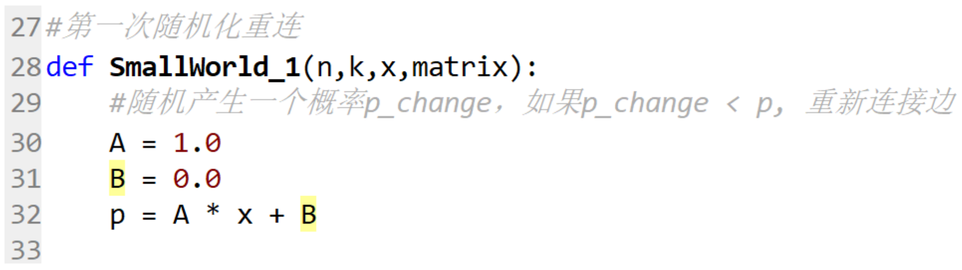
As the pictures show, if every node connects to a node except itself, the other node will have many chances to know more than one person after random reconnection.

3. The second improved model

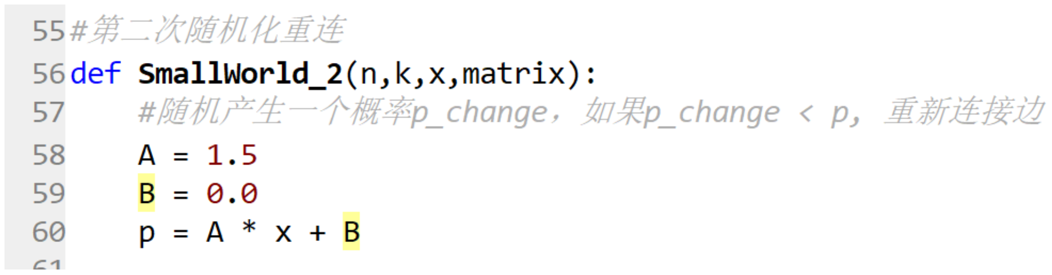
The second improved model is to change the fixed probability. If p = 0, then the network is completely regular. If p = 1, then the network is completely random. However, in the reality, p is a complex and fluctuating between the random and the regular. So I define p is the number of linear changes P = A * X + B. Setting x in the range of [0,1], p also in the range of [0,1]. I conclude that A determines the change speed in randomness and B determines the size of the initial randomness.



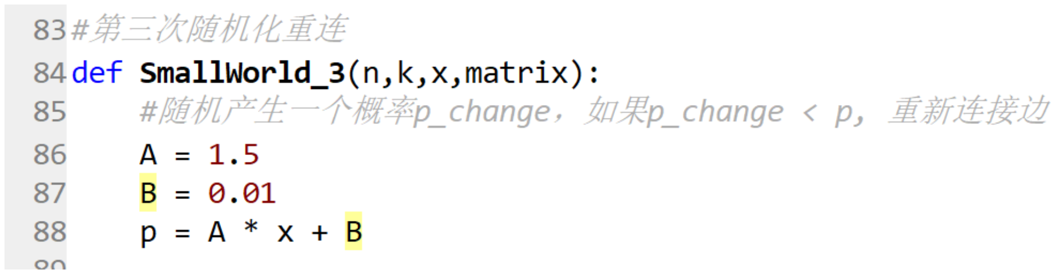
According to the principle, I simulate two situations. The first situation is on the basis of the original randomized reconnection, the second situation is randomized reconnection three times respectively.



```
#第一次随机化重连
def SmallWorld_1(n,k,x,matrix):
    #随机产生一个概率p_change，如果p_change < p, 重新连接边
    A = 1.0
    B = 0.0
    p = A * x + B
```

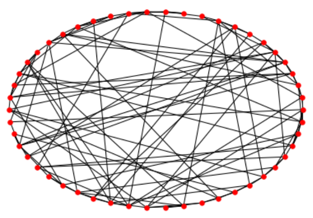


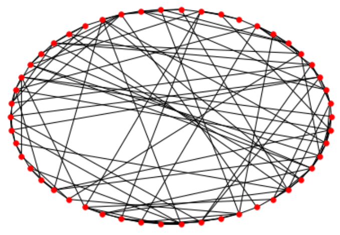
```
#第二次随机化重连
def SmallWorld_2(n,k,x,matrix):
    #随机产生一个概率p_change，如果p_change < p, 重新连接边
    A = 1.5
    B = 0.0
    p = A * x + B
```



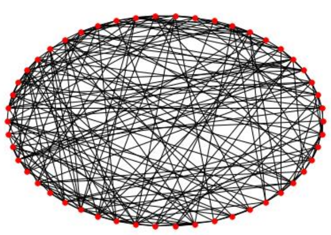
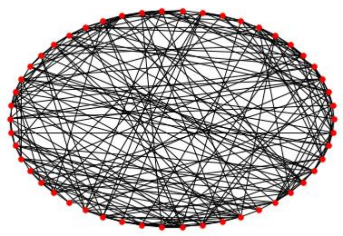
```
#第三次随机化重连
def SmallWorld_3(n,k,x,matrix):
    #随机产生一个概率p_change，如果p_change < p, 重新连接边
    A = 1.5
    B = 0.01
    p = A * x + B
```

The first situation: on the basis of the original randomized reconnection

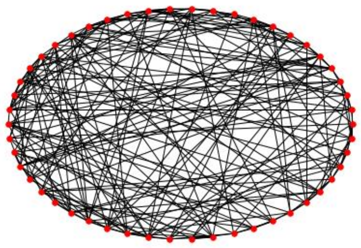
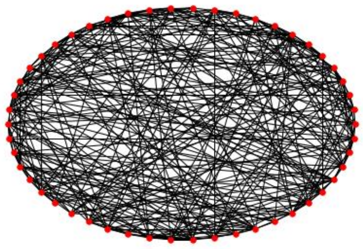




A = 1.0 B = 0.0 X= 0.4 P = A * X + B = 0.4 N = 50 K = 4

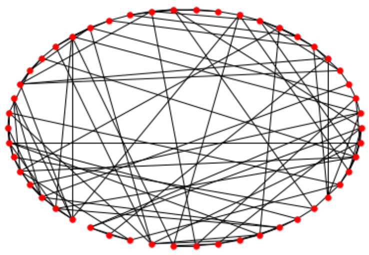
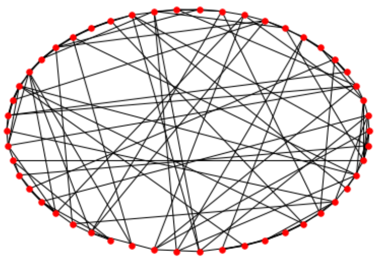



A = 1.5 B = 0.0 X= 0.4 P = A * X + B = 0.6 N = 50 K = 4

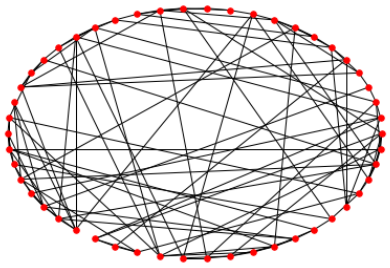
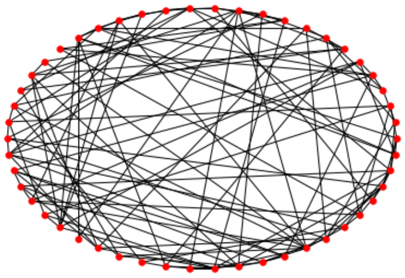



A = 1.5 B = 0.2 X= 0.4 P = A * X + B = 0.8 N = 50 K = 4

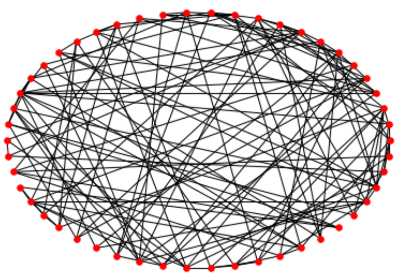
The second situation: randomized reconnection three times respectively

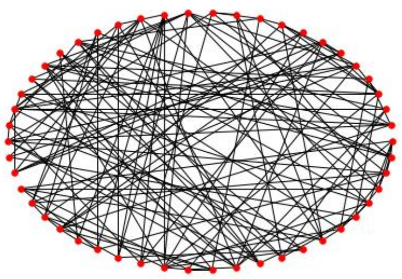



A = 1.0 B = 0.0 X= 0.4 P = A * X + B = 0.4 N = 50 K = 4

A = 1.5 B = 0.0 X= 0.4 P = A * X + B = 0.6 N = 50 K = 4



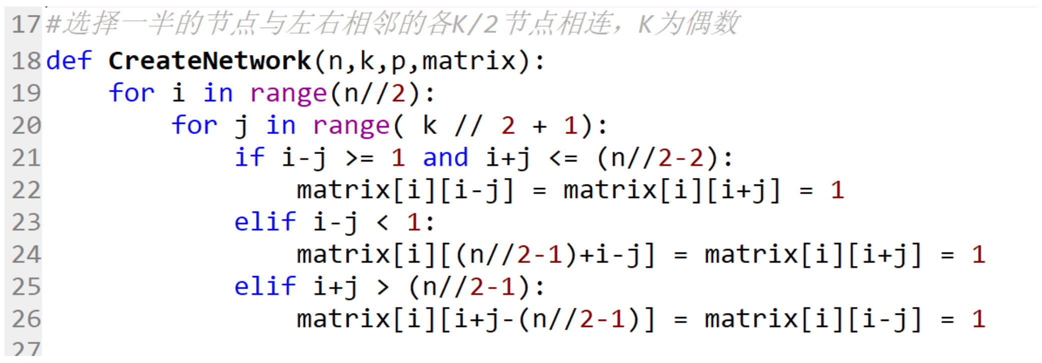


A = 1.5 B = 0.2 X= 0.4 P = A * X + B = 0.8 N = 50 K = 4

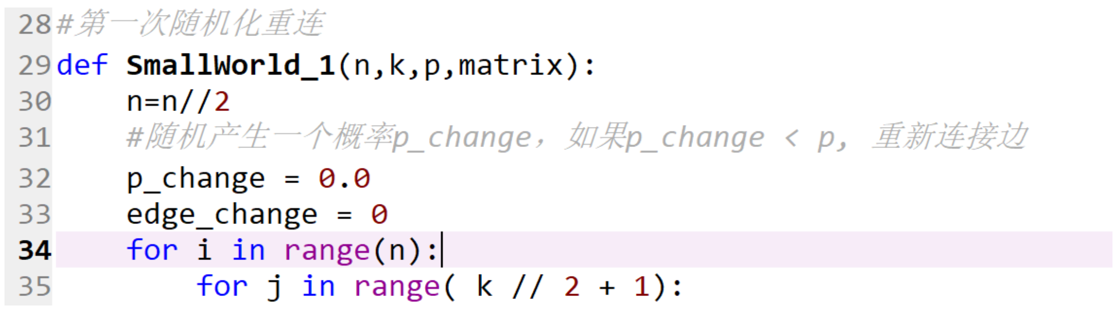
As the pictures show, if we random reconnect on the basis of the original, the connection will grow more and more with fast speed. If we random reconnection three times respectively, it means that we choose different p in different time. And the p is affected by A and B, A is the speed of change, B is the initial random value.

4. The third improved model

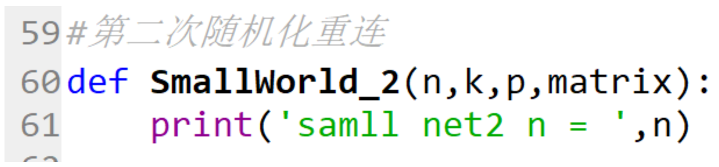
One of the constraints of the WS small world model is closed. In reality, the scale of network is constantly expanding. For example, hypothesis graduate sophomore can form a network. Graduate freshmen are newly added discrete nodes. And graduate sophomore and graduate freshmen form a new network now. At this time, the old network will affect the new network. How to deal with the hierarchical structure. I decide to change the WS model to grow. I set the half of the nodes form the old network, and the all of nodes form the new network.



```
#选择一半的节点与左右相邻的各K/2节点相连，K为偶数
def CreateNetwork(n,k,p,matrix):
    for i in range(n//2):
        for j in range( k // 2 + 1):
            if i-j >= 1 and i+j <= (n//2-2):
                matrix[i][i-j] = matrix[i][i+j] = 1
            elif i-j < 1:
                matrix[i][(n//2-1)+i-j] = matrix[i][i+j] = 1
            elif i+j > (n//2-1):
                matrix[i][i+j-(n//2-1)] = matrix[i][i-j] = 1
```



```
#第一次随机化重连
def SmallWorld_1(n,k,p,matrix):
    n=n//2
    #随机产生一个概率p_change，如果p_change < p, 重新连接边
    p_change = 0.0
    edge_change = 0
    for i in range(n):
        for j in range( k // 2 + 1):
```



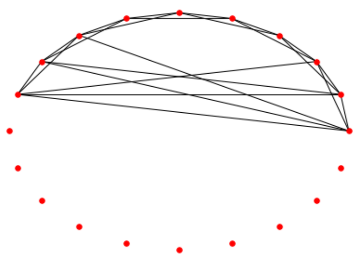
```python
#第二次随机化重连
def SmallWorld_2(n,k,p,matrix):
    print('samll net2 n = ',n)
```

First of all, the nodes are divided into two layers. And then according to the principle, I simulate two situations.

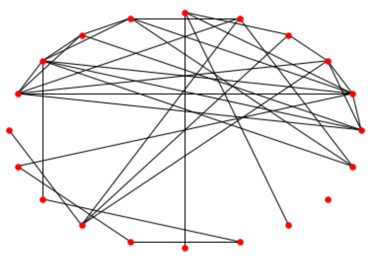
For the first situation, the first layer of the old network is a nearest neighbor coupled network. Then adding to the second layer of discrete nodes, all of nodes together with the random re-connected. So the new network is a small world network.

For the second situation, the first layer of the old network is a small world network which had been random re-connected. And then leaving the previous edges, adding to the second layer of discrete nodes and all of nodes together with the random re-connected. So the new network is also a small world network.

The first situation:

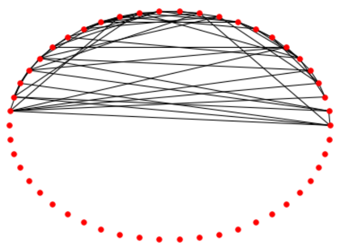


N = 25 K = 4 P = 0.4

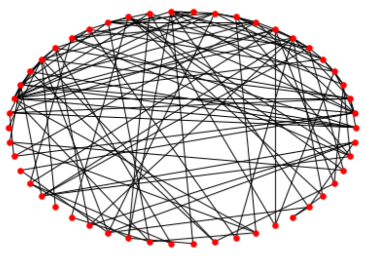


N = 25 K = 4 P = 0.4

The second situation:



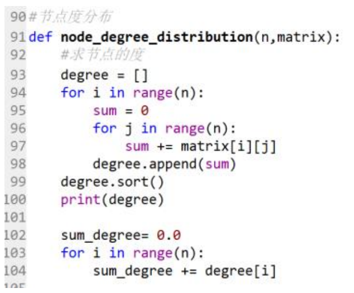
N = 50 K = 4 P = 0.4



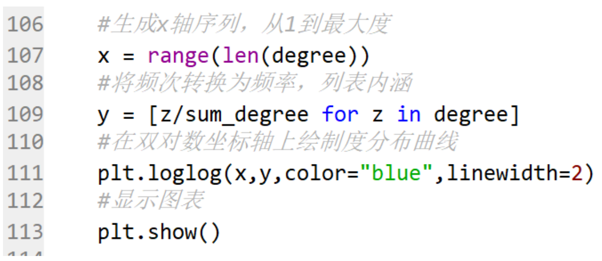
N = 50 K = 4 P = 0.4

Network characteristics

1.Node-degree distribution



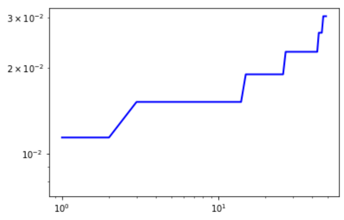
```
#节点度分布
def node_degree_distribution(n,matrix):
    #求节点的度
    degree = []
    for i in range(n):
        sum = 0
        for j in range(n):
            sum += matrix[i][j]
        degree.append(sum)
    degree.sort()
    print(degree)

    sum_degree= 0.0
    for i in range(n):
        sum_degree += degree[i]
```



```
    #生成x轴序列，从1到最大度
    x = range(len(degree))
    #将频次转换为频率，列表内涵
    y = [z/sum_degree for z in degree]
    #在双对数坐标轴上绘制度分布曲线
    plt.loglog(x,y,color="blue",linewidth=2)
    #显示图表
    plt.show()
```

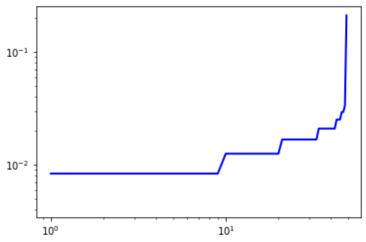
1. The original model：

degree = [2, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 8, 8, 8]



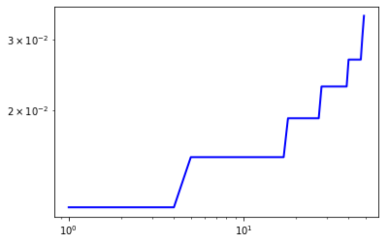
2. The first improved model：

degree = [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7, 7, 8, 50]



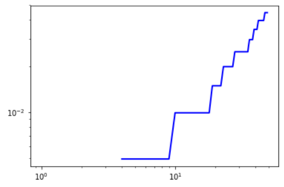
3. The second improved model：

degree = [3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 8, 9]

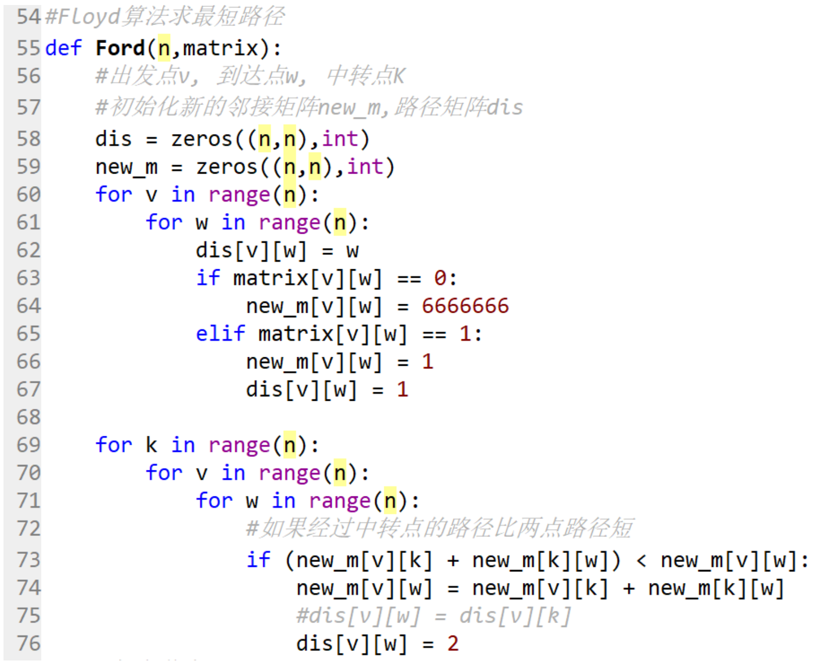


4. The third improved model：

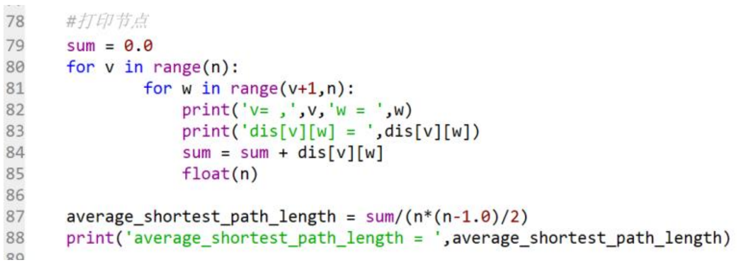
degree = [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 8, 8, 9, 9, 9]



2.Average shortest path-length



```python
#Floyd算法求最短路径
def Ford(n,matrix):
    #出发点v, 到达点w, 中转点K
    #初始化新的邻接矩阵new_m,路径矩阵dis
    dis = zeros((n,n),int)
    new_m = zeros((n,n),int)
    for v in range(n):
        for w in range(n):
            dis[v][w] = w
            if matrix[v][w] == 0:
                new_m[v][w] = 6666666
            elif matrix[v][w] == 1:
                new_m[v][w] = 1
                dis[v][w] = 1

    for k in range(n):
        for v in range(n):
            for w in range(n):
                #如果经过中转点的路径比两点路径短
                if (new_m[v][k] + new_m[k][w]) < new_m[v][w]:
                    new_m[v][w] = new_m[v][k] + new_m[k][w]
                    #dis[v][w] = dis[v][k]
                    dis[v][w] = 2
```



```python
    #打印节点
    sum = 0.0
    for v in range(n):
            for w in range(v+1,n):
                print('v= ,',v,'w = ',w)
                print('dis[v][w] = ',dis[v][w])
                sum = sum + dis[v][w]
                float(n)

    average_shortest_path_length = sum/(n*(n-1.0)/2)
    print('average_shortest_path_length = ',average_shortest_path_length)
```

1. The original model：
   average_shortest_path_length = 1.90530612245

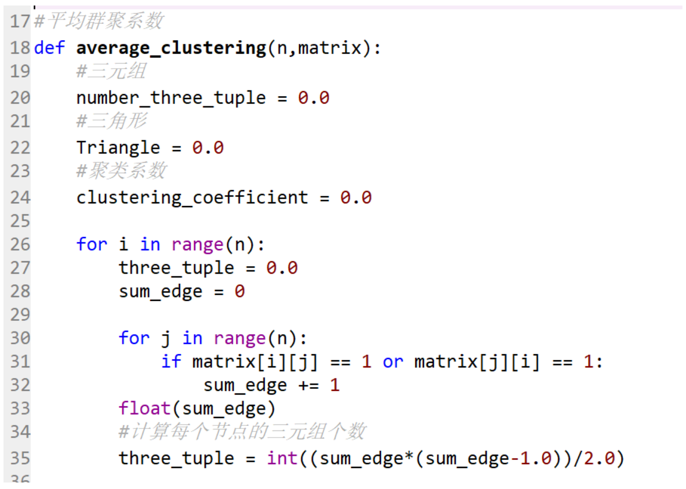
2. The first improved model：
   average_shortest_path_length = 1.90285714286

3. The second improved model：
   average_shortest_path_length = 1.90612244898

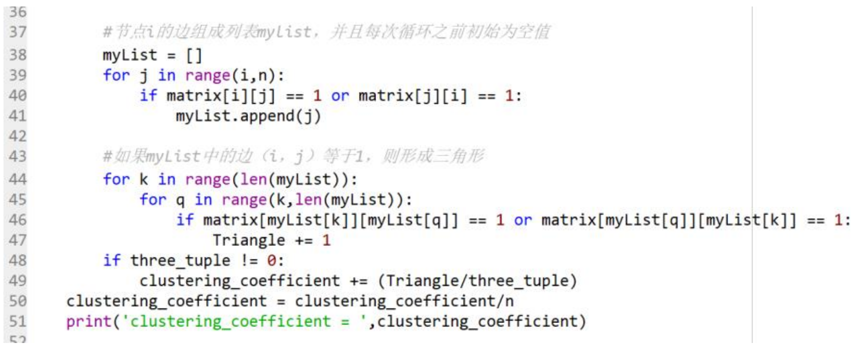
4. The third improved model：

average_shortest_path_length =   10.2857142857

3.Clustering coefficient



```python
#平均群聚系数
def average_clustering(n,matrix):
    #三元组
    number_three_tuple = 0.0
    #三角形
    Triangle = 0.0
    #聚类系数
    clustering_coefficient = 0.0

    for i in range(n):
        three_tuple = 0.0
        sum_edge = 0

        for j in range(n):
            if matrix[i][j] == 1 or matrix[j][i] == 1:
                sum_edge += 1
        float(sum_edge)
        #计算每个节点的三元组个数
        three_tuple = int((sum_edge*(sum_edge-1.0))/2.0)
```



```python

        #节点i的边组成列表myList，并且每次循环之前初始为空值
        myList = []
        for j in range(i,n):
            if matrix[i][j] == 1 or matrix[j][i] == 1:
                myList.append(j)

        #如果myList中的边（i，j）等于1，则形成三角形
        for k in range(len(myList)):
            for q in range(k,len(myList)):
                if matrix[myList[k]][myList[q]] == 1 or matrix[myList[q]][myList[k]] == 1:
                    Triangle += 1
        if three_tuple != 0:
            clustering_coefficient += (Triangle/three_tuple)
    clustering_coefficient = clustering_coefficient/n
    print('clustering_coefficient = ',clustering_coefficient)
```

1. The original model：

   clustering_coefficient =   14.654111111111111

2. The first improved model：

   clustering_coefficient =   36.61819727891157

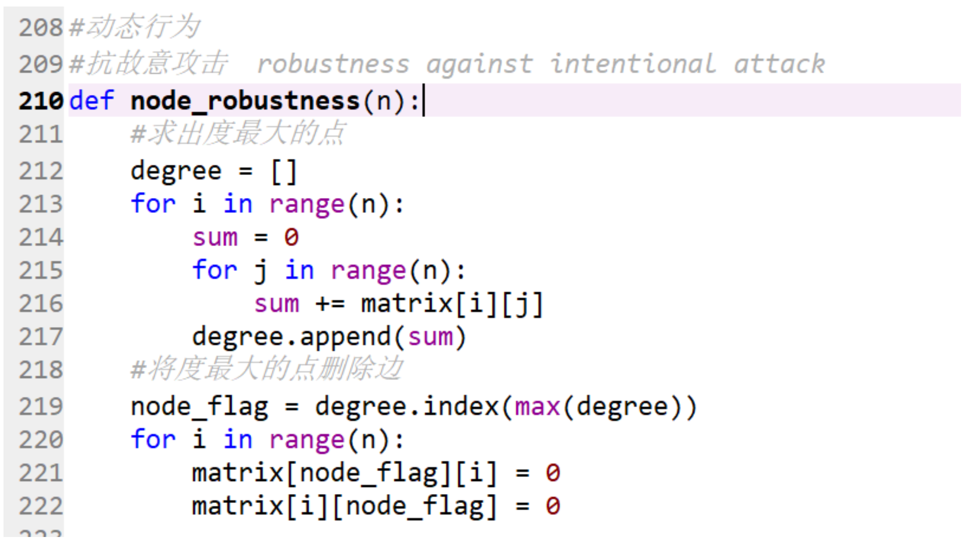
3. The second improved model：

   clustering_coefficient =   16.23684126984127

4. The third improved model：

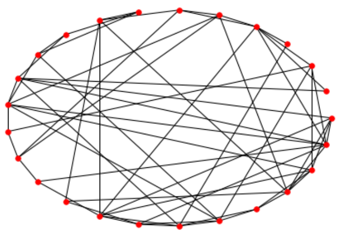
   clustering_coefficient =   17.105698412698413

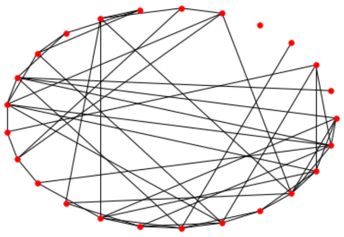
4.Robustness against intentional attack



```
#动态行为
#抗敌意攻击  robustness against intentional attack
def node_robustness(n):
    #求出度最大的点
    degree = []
    for i in range(n):
        sum = 0
        for j in range(n):
            sum += matrix[i][j]
        degree.append(sum)
    #将度最大的点删除边
    node_flag = degree.index(max(degree))
    for i in range(n):
        matrix[node_flag][i] = 0
        matrix[i][node_flag] = 0
```

1. The original model：

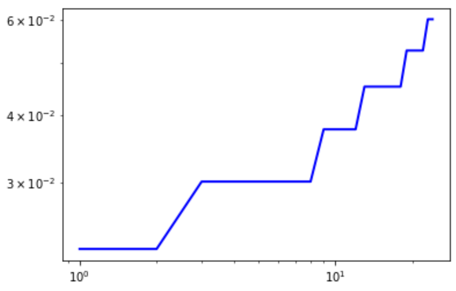


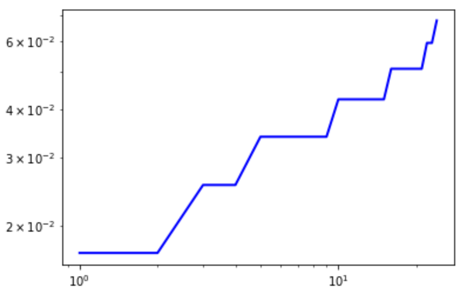


clustering_coefficient = 7.104952380952382

average_shortest_path_length = 1.80333333333

degree = [3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8]



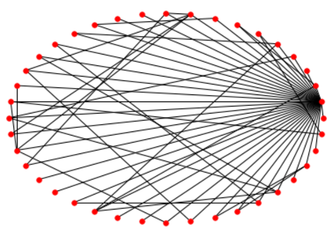
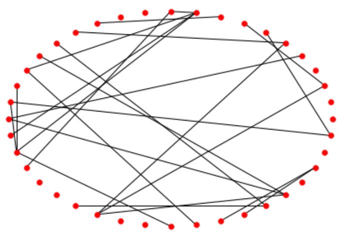


clustering_coefficient = 7.199238095238095

average_shortest_path_length = 2.68666666667

degree = [0, 2, 2, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 8]

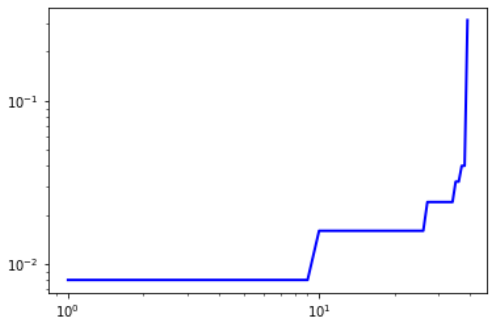
2. The first improved model：

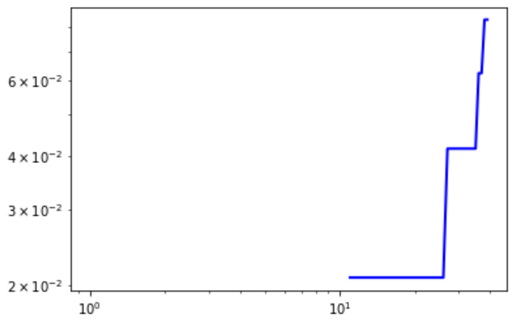



clustering_coefficient = 30.807058029689614

average_shortest_path_length = 1.92051282051

degree = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 5, 5, 39]





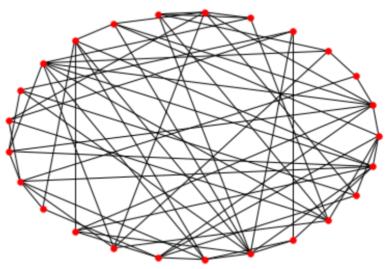
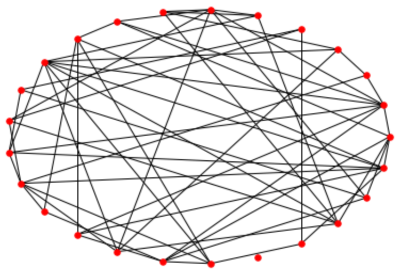
clustering_coefficient = 0.0

average_shortest_path_length = 23.1282051282

degree = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4, 4]

3. The second improved model：

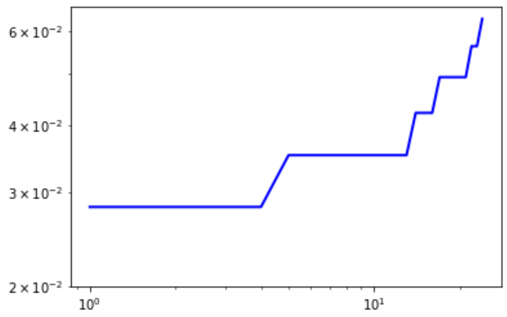
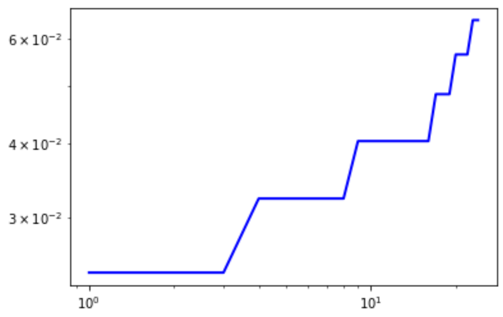
The first situation:

clustering_coefficient = 4.718444444444445

average_shortest_path_length = 1.77333333333

degree = [3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8, 9]

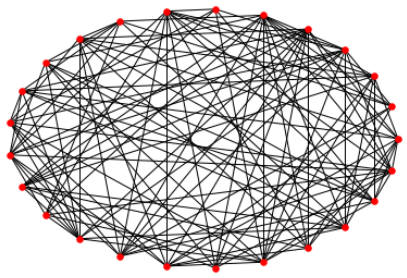
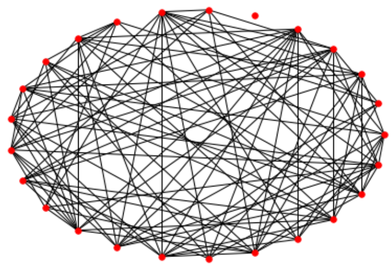



clustering_coefficient = 5.230476190476192

average_shortest_path_length = 3.27666666667

degree = [0, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8]

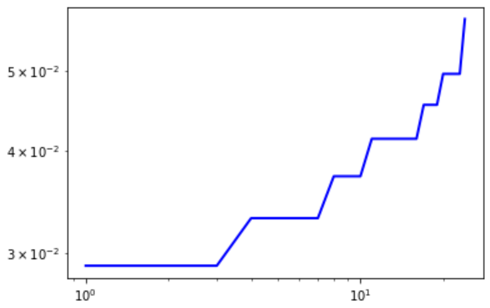
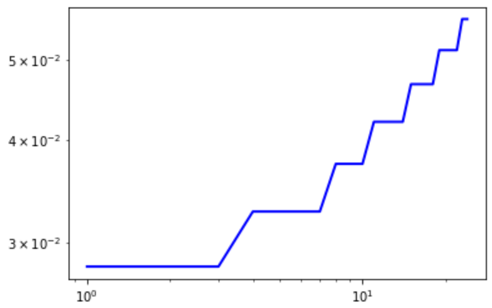
The second situation:

clustering_coefficient = 3.3337595737595738

average_shortest_path_length = 1.59666666667

degree = [7, 7, 7, 7, 8, 8, 8, 8, 8, 9, 9, 9, 10, 10, 10, 10, 10, 10, 11, 11, 11, 12, 12, 12, 12, 14]

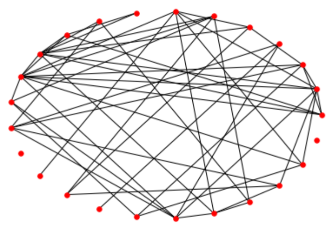
clustering_coefficient = 3.1082799422799416

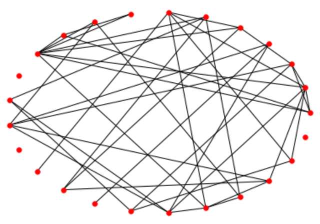
average_shortest_path_length = 2.51666666667

degree = [0, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 11, 11, 11, 11, 12, 12]

4. The third improved model：

The first situation:

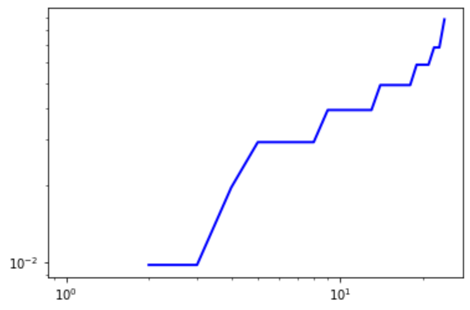


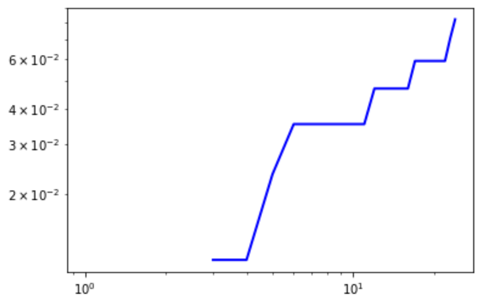


clustering_coefficient = 5.927873015873016

average_shortest_path_length = 4.66333333333

degree = [0, 0, 1, 1, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 7, 7, 9]



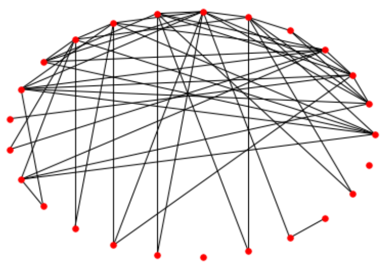


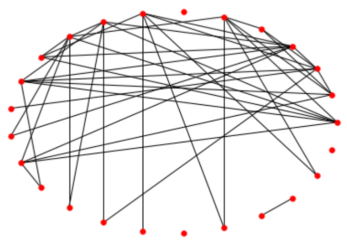
clustering_coefficient = 4.927301587301587

average_shortest_path_length = 5.6

degree = [0, 0, 0, 1, 1, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 7]

The second situation:

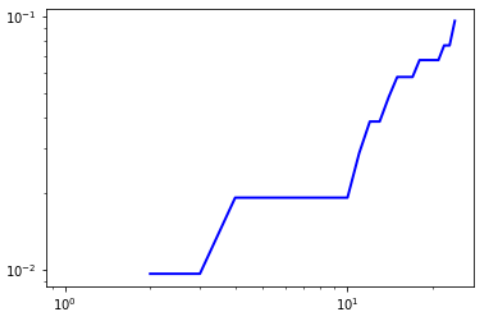


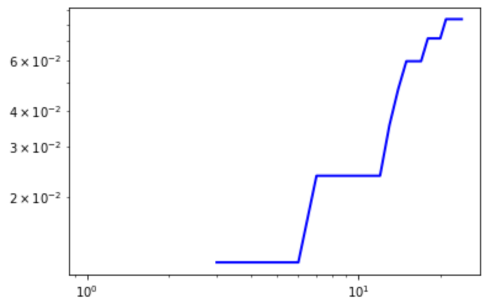


clustering_coefficient = 15.26015873015873

average_shortest_path_length = 4.92333333333

degree = [0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7, 7, 7, 8, 8, 10]



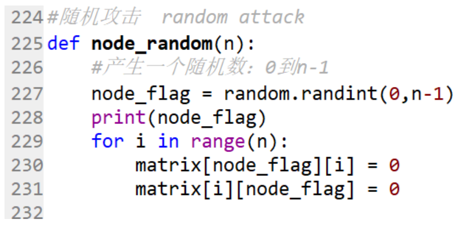


clustering_coefficient = 9.709142857142858

average_shortest_path_length = 8.32666666667

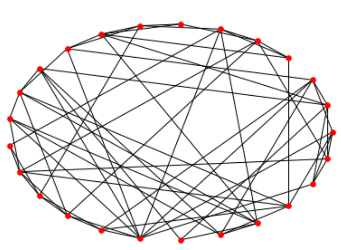
degree = [0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7]

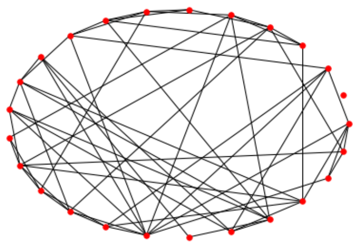
5.Random attack



```
#随机攻击  random attack
def node_random(n):
    #产生一个随机数：0到n-1
    node_flag = random.randint(0,n-1)
    print(node_flag)
    for i in range(n):
        matrix[node_flag][i] = 0
        matrix[i][node_flag] = 0
```

1. The original model：

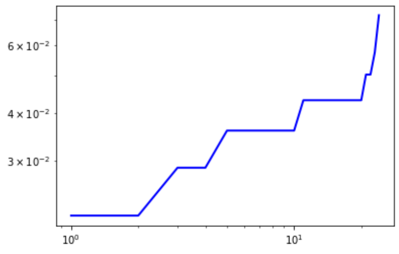


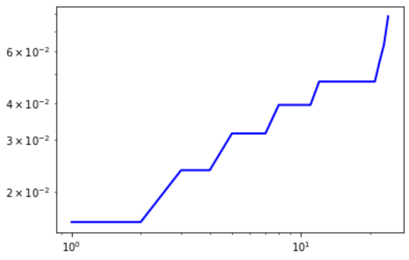


clustering_coefficient = 6.540825396825397

average_shortest_path_length = 1.78333333333

degree = [3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 8, 10]



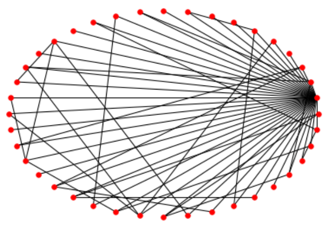


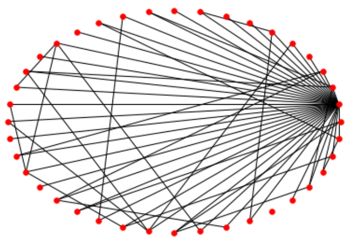
clustering_coefficient = 9.18336507936508

average_shortest_path_length = 2.64333333333

degree = [0, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 10]

2. The first improved model：

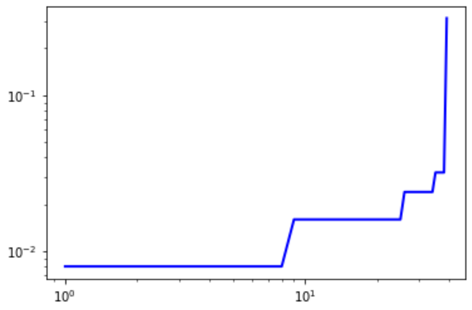


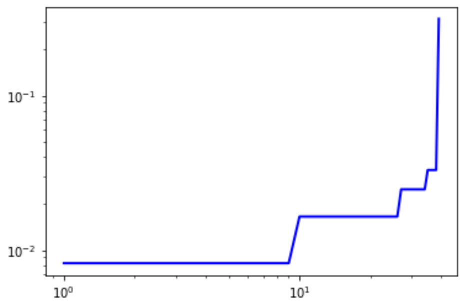


clustering_coefficient = 29.993724696356274

average_shortest_path_length = 1.92051282051

degree = [1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 39]





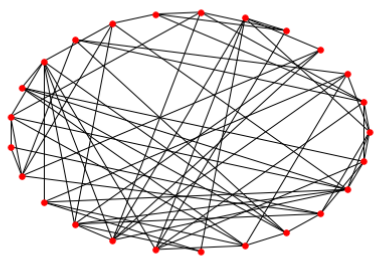
clustering_coefficient = 28.51876481744903

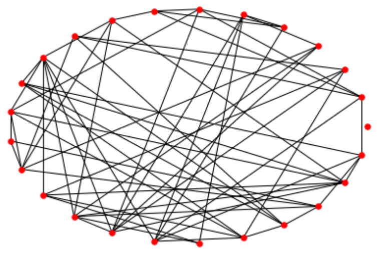
average_shortest_path_length = 3.54230769231

degree = [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 38]

3. The second improved model：

The first situation:

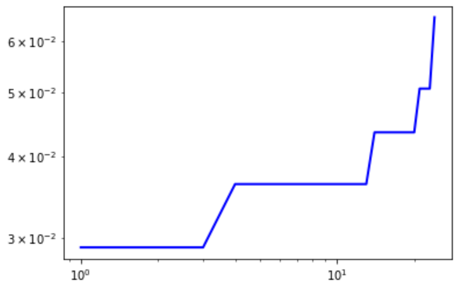


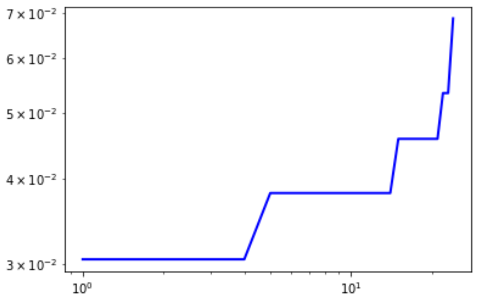


clustering_coefficient = 4.247999999999999

average_shortest_path_length = 1.78

degree = [4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 9]



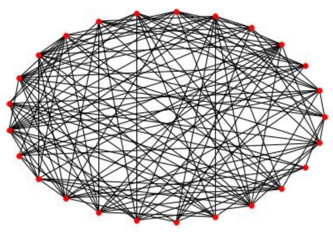
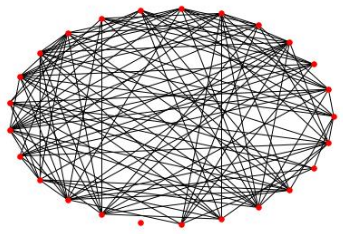


clustering_coefficient = 3.973904761904762

average_shortest_path_length = 2.64

degree = [0, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 9]

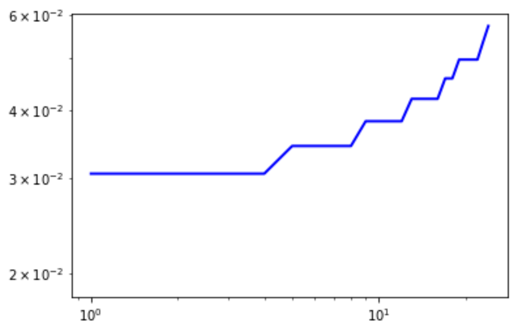
The second situation:

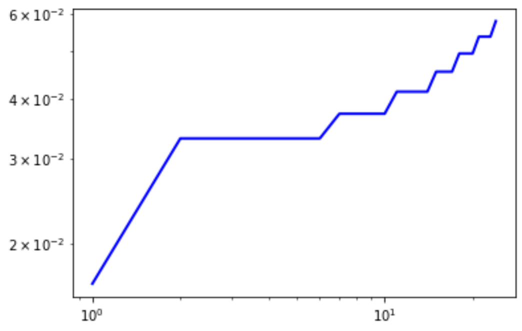



clustering_coefficient = 3.704933066933067

average_shortest_path_length = 1.56333333333

degree = [5, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 11, 11, 11, 11, 12, 12, 13, 13, 13, 13, 14, 15]





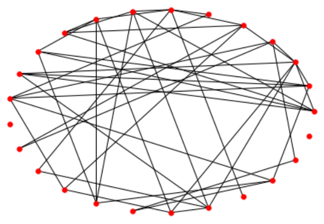
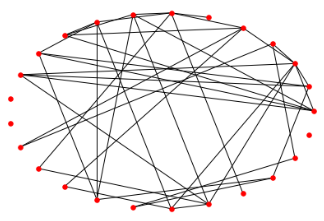
clustering_coefficient = 3.5414398934398936

average_shortest_path_length = 2.94666666667

degree = [0, 4, 8, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 11, 11, 11, 12, 12, 12, 13, 13, 13, 14]

4. The third improved model：

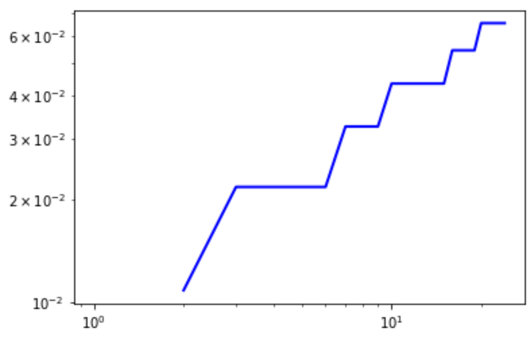
The first situation:

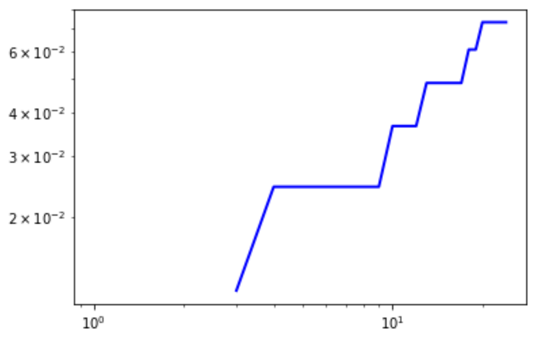



clustering_coefficient = 8.981142857142858

average_shortest_path_length = 4.63666666667

degree = [0, 0, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6]



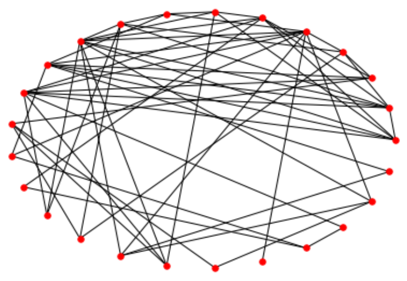


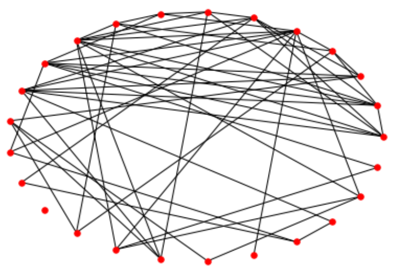
clustering_coefficient = 10.160571428571428

average_shortest_path_length = 5.60333333333

degree = [0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 6, 6, 6, 6, 6]

The second situation:

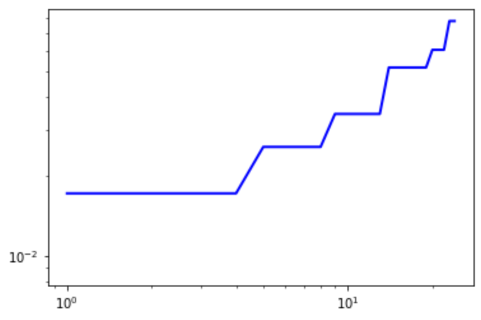


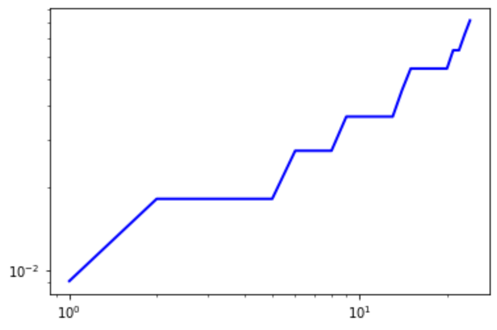


clustering_coefficient = 12.889428571428573

average_shortest_path_length = 1.80666666667

degree = [1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 7, 7, 7, 9, 9]





clustering_coefficient = 12.09873015873016

average_shortest_path_length = 3.00666666667

degree = [0, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 5, 6, 6, 6, 6, 6, 6, 7, 7, 8, 9]