第3章

思想引领

节点重要性

在实际应用中有很多关于节点重要性的问题。例如，在各种社交网络（如微信朋友圈、知乎和微博等在线社区）中，哪些是最活跃、最具影响力的人？在疾病传播网络中，哪些是超级传播者？在通信网络和交通网络中，哪些节点承受的流量最大？当你在搜索引擎中输入一个关键词后，搜索引擎如何知道哪些页面对你是最重要的？在论文引用网络中，哪些论文是重要的？如何衡量论文的重要性？

复杂网络的重要节点是指相比网络其他节点而言，能够在更大程度上影响网络的结构与功能的一些特殊节点。网络结构涉及度分布、平均距离、连通性、聚类系数、度相关性等，网络功能涉及网络的抗毁性、传播、同步、控制等。本章首先介绍无向网络中节点的重要性指标，然后介绍有向网络中节点的重要性指标。在众多节点重要性指标中究竟谁好谁劣，如何选择？科研中，如果自己提出一个指标，如何评价它？为了回答这些问题，3.3节将给出节点重要性的衡量标准。

3.1 无向网络节点重要性指标

3.1.1 度中心性

网络中心性最直接的度量是度中心性（Degree Centrality），即一个节点的度越大就意味着这个节点越重要。在一个包含 N 个节点的网络中，节点最大可能的度值为 $N-1$，通常为便于比较而对中心性指标作归一化处理，度为 k_i 的节点的归一化度中心性值定义为

$$\mathrm{DC}_i = \frac{k_i}{N-1} \tag{3-1}$$

【例 3-1】 计算图 3.1 所示线状图、星状图和杠铃图的度中心性。

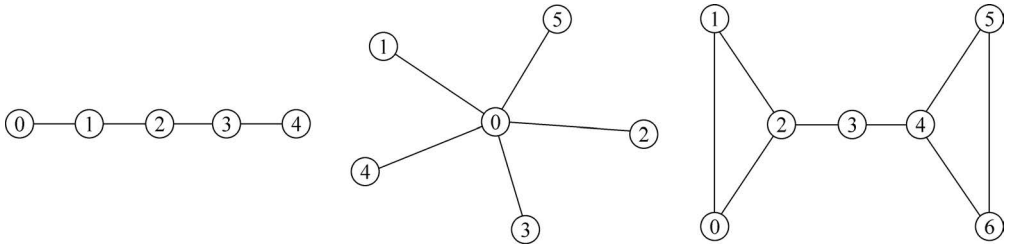


图 3.1 线状图、星状图和杠铃图

(1) 线状图中,除两端节点度值为 1 外,中间节点的度值都为 2,从而度中心性依次为 1/4,2/4,2/4,2/4,1/4。

```
path = nx.path_graph(5)
dc = nx.degree_centrality(path)                    #度中心性
#cc = nx.closeness_centrality(path)                #接近度中心性
#bc = nx.betweenness_centrality(path)              #介数中心性
#ec = nx.eigenvector_centrality(path)              #特征向量中心性
print(dc)
#可视化
plt.figure(figsize=(4,3))
pos = [(0,0),(1,0),(2,0),(3,0),(4,0)]
node_size=[100*100**i for i in dc.values()]
nx.draw(path,pos=pos,with_labels=True,node_color='y',node_size=node_size)
```

运行结果如下:

```
{0: 0.25, 1: 0.5, 2: 0.5, 3: 0.5, 4: 0.25}
```

(2) 星状图,除中心节点度为 5 外,其余节点度值都为 1。

```
star = nx.star_graph(5)
dc = nx.degree_centrality(star)                    #度中心性
#cc = nx.closeness_centrality(path)                #接近度中心性
#bc = nx.betweenness_centrality(path)              #介数中心性
#ec = nx.eigenvector_centrality(path)              #特征向量中心性
print(dc)

plt.figure(figsize=(4,3))
pos = nx.spring_layout(star,seed=60)
node_size=[500*10**i for i in dc.values()]
nx.draw(star,pos=pos,with_labels=True,node_color='y',node_size=node_size)
```

运行结果如下:

```
{0: 1.0, 1: 0.2, 2: 0.2, 3: 0.2, 4: 0.2, 5: 0.2}
```

(3) 杠铃图,该图是由一个中间节点连接的两个 3 节点完全图。

```
barbell = nx.barbell_graph(3,1)
dc = nx.degree_centrality(barbell)                 #度中心性
#cc = nx.closeness_centrality(barbell)             #接近度中心性
#bc = nx.betweenness_centrality(barbell)           #介数中心性
#ec = nx.eigenvector_centrality(barbell)           #特征向量中心性
print(dc)

plt.figure(figsize=(4,3))
pos=[(0,-1),(0,1),(1,0),(2,0),(3,0),(4,1),(4,-1)]
node_size=[100*100**i for i in dc.values()]
nx.draw(barbell,pos=pos,with_labels=True,node_color='y',node_size=node_size)
```

运行结果如下:

```
{0: 0.3333333333333333, 1: 0.3333333333333333, 2: 0.5, 4: 0.5, 5: 0.3333333333333333,
6: 0.3333333333333333, 3: 0.3333333333333333}
```

度中心性如图 3.2 所示。

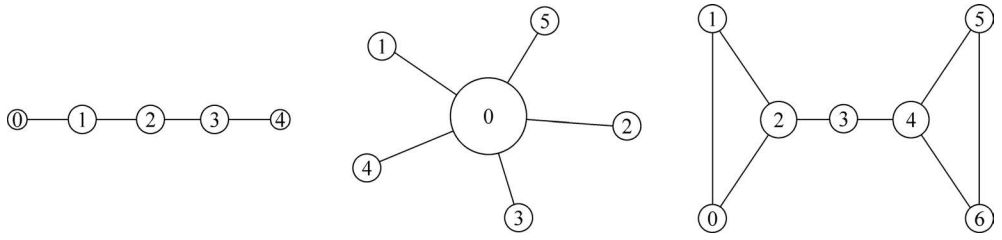


图 3.2　度中心性

nx.degree_centrality(G)：计算节点的度中心性。

nx.in_degree_centrality(G)：计算节点的入度中心性。

nx.out_degree_centrality(G)：计算节点的出度中心性。

3.1.2　接近度中心性

对于网络中的每个节点 i，可以计算该节点到网络中所有节点的距离的平均值，记为 d_i。

$$d_i=\frac{1}{N}\sum_{j=1}^{N}d_{ij}\quad 或\quad d_i=\frac{1}{N-1}\sum_{j=1}^{N}d_{ij} \tag{3-2}$$

其中，d_{ij} 是节点 i 到节点 j 的距离，分母取 N 可认作是包含 $d_{ii}=0$ 的归一化结果，取 $N-1$ 则认作不包含 d_{ii}。这样，就得到网络平均路径长度的另一种计算公式：

$$L=\frac{1}{N}\sum_{i=1}^{N}d_i \tag{3-3}$$

d_i 值的相对大小也在某种程度上反映了节点 i 在网络中的相对重要性：d_i 值越小意味着节点 i 更接近其他节点。把 d_i 的倒数定义为节点 i 的接近中心性(Closeness Centrality)，简称接近数，用记号 CC_i 来表示：

$$CC_i=\frac{1}{d_i} \tag{3-4}$$

接近度中心性的缺点：取值范围较小，区分性不大。在大部分网络中，节点之间的距离一般都比较小(小世界)，并且随着网络规模的增长，该值以对数级速度缓慢增长。这意味着很难通过接近度中心性显著的区分节点的重要性。

【例 3-2】 计算图 3.3 所示线状图、星状图和杠铃图的接近度中心性。

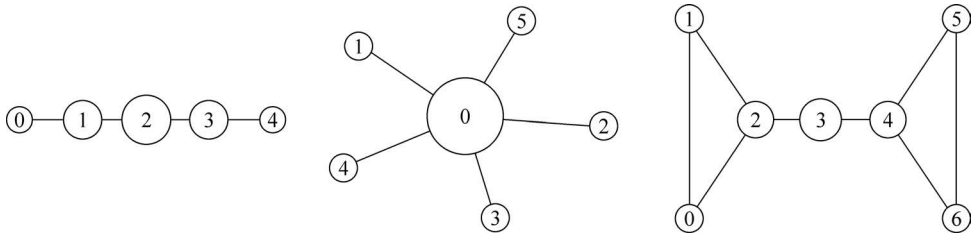


图 3.3　接近度中心性

线状图：{0:0.4,1:0.5714285714285714,2:0.6666666666666666,3:0.5714285714285714,4:0.4}。

星状图：{0:1.0,1:0.5555555555555556,2:0.5555555555555556,3:0.5555555555555556,4:0.5555555555555556,5:0.5555555555555556}。

杠铃图：{0:0.4,1:0.4,2:0.5454545454545454,4:0.5454545454545454,5:0.4,6:0.4,3:0.6}。

nx.closeness_centrality(G[,u,distance,…])：计算节点的接近度中心性。

3.1.3　介数中心性

介数中心性最早由 Freeman 于 1977 年给出，它刻画了节点 i 对于网络中节点对之间沿着最短路径传输信息的控制能力。以经过某个节点的最短路径的数目来刻画节点重要性的指标就称为介数中心性(Betweeness Centrality)，简称介数(BC)。

节点 i 的介数定义为

$$\mathrm{BC}_i = \sum_{s \neq i \neq t} \frac{n_{st}^i}{g_{st}} \tag{3-5}$$

其中，g_{st} 为从节点 s 到节点 t 的最短路径的数目，n_{st}^i 为从节点 s 到节点 t 的 g_{st} 条最短路径中经过节点 i 的最短路径的数目。

对于一个包含 N 个节点的连通网络，节点度的最大可能值为 $N-1$，节点介数的最大可能值是星状网络中的中心节点的介数值：因为所有其他节点对之间的最短路径是唯一的并且都会经过该中心节点，所以该节点的介数就是这些最短路径的数目，即为

$$C_{N-1}^2 = \frac{(N-1)(N-2)}{2} \tag{3-6}$$

因此，一个包含 N 个节点的网络中的节点 i 的归一化介数定义为

$$\mathrm{BC}_i = \frac{1}{(N-1)(N-2)/2} \sum_{s,t} \frac{n_{st}^i}{g_{st}} \tag{3-7}$$

从控制信息传输的角度而言，介数越高的节点其重要性也越大，去除这些节点后对网络传输的影响也越大。介数最高的节点对于网络中信息的流动具有最大的控制力，而接近数最大的节点则对于信息的流动具有最佳的观察视野。一般而言，介数最大的节点并不一定就是接近数最大的节点。

介数中心性的优点：介数的值分布在很大范围内；星状图拥有最大的介数中心性；介数中心性得到的结果比接近度中心性得到的结果更稳定。

【例 3-3】 计算图 3.1 所示线状图、星状图和杠铃图的介数中心性，如图 3.4 所示。

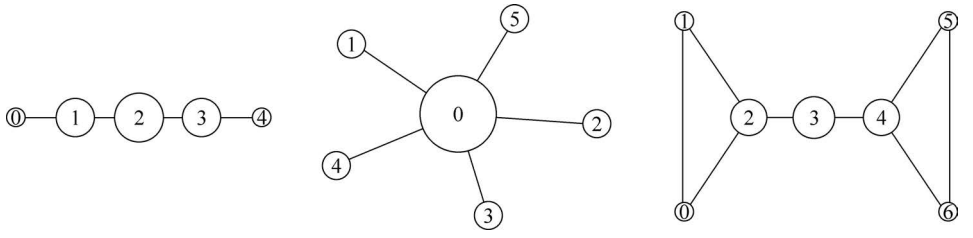


图 3.4　介数中心性

线状图：{0:0.0,1:0.5,2:0.6666666666666666,3:0.5,4:0.0}。

星状图：{0:1.0,1:0.0,2:0.0,3:0.0,4:0.0,5:0.0}。

杠铃图：{0:0.0,1:0.0,2:0.5333333333333333,4:0.5333333333333333,5:0.0,6:0.0,3:0.6}。

nx.betweenness_centrality(G[,k,normalized,…])：计算节点的介数中心性。

nx.edge_betweenness_centrality(G[,k,…])：计算边介数中心性。

3.1.4　特征向量中心性

特征向量中心性(Eigenvector Centrality)的基本想法是：一个节点的重要性既取决于其

邻居节点的数量(即该节点的度),也取决于其邻居节点的重要性。记 x_i 为节点 i 的重要性度量值,那么,应该有

$$x_i = c\sum_{j=1}^{N} a_{ij}x_j \tag{3-8}$$

其中,c 为比例常数,$\boldsymbol{A}=(a_{ij})$是网络的邻接矩阵。记 $\boldsymbol{x}=[x_1,x_2,\cdots,x_N]^{\mathrm{T}}$,则可写成如下矩阵形式:

$$\boldsymbol{x} = c\boldsymbol{A}\boldsymbol{x} \tag{3-9}$$

相应的可以改写为

$$\boldsymbol{A}\boldsymbol{x} = \frac{1}{c}\boldsymbol{x} \tag{3-10}$$

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \tag{3-11}$$

$$x_i = \frac{1}{\lambda_1}\sum_{j=1}^{N} a_{ij}x_j \tag{3-12}$$

对比度中心性仅考虑邻居的数量,特征向量中心性描述一个节点的重要性既取决于邻居的数量也取决于邻居的质量。

有两种方法可用于计算节点的特征向量中心性:①迭代法 $\boldsymbol{x}(k)=c\boldsymbol{A}\boldsymbol{x}(k-1),k=1,2,\cdots$,初值 $x(0)$可随机选择。②矩阵的最大特征值对应的特征向量。

【例 3-4】 计算图 3.1 所示线状图、星状图和杠铃图的特征向量中心性,如图 3.5 所示。

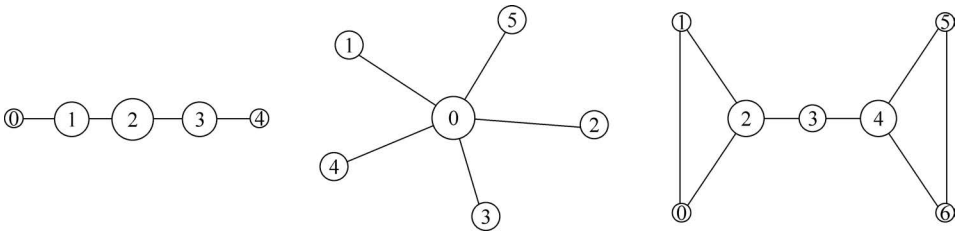


图 3.5 特征向量中心性

线状图:{0:0.2886760320285294,1:0.49999999508235304,2:0.5773493802714443,3:0.4999999950823529,4:0.2886760320285293}。

星状图:{0:0.7071064011232681,1:0.3162279359862123,2:0.3162279359862123,3:0.3162279359862123,4:0.3162279359862123,5:0.3162279359862123}。

杠铃图:{0:0.3348059077247065,1:0.3348059077247065,2:0.44961739431203446,4:0.44961739431203446,5:0.3348059077247065,6:0.3348059077247065,3:0.3838077827176706}。

nx.eigenvector_centrality(G[,max_iter,tol,…]):计算图 G 的特征向量中心性。

nx.eigenvector_centrality_numpy(G[,weight,…]):计算图 G 的特征向量中心性。

【例 3-5】 综合练习,使用度中心性、接近度中心性、介数中心性和特征向量中心性分析图 3.6 所示风筝网络的节点中心性。

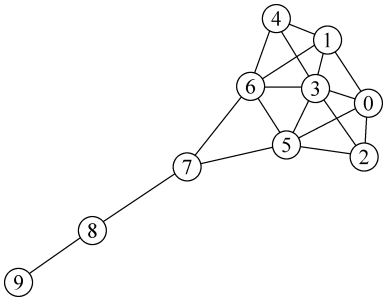


图 3.6 风筝网络

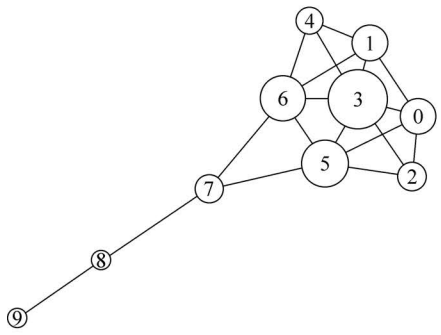
```
kite = nx.krackhardt_kite_graph()
dc = nx.degree_centrality(kite)                    #度中心性
#cc = nx.closeness_centrality(kite)                #接近度中心性
#bc = nx.betweenness_centrality(kite)              #介数中心性
#ec = nx.eigenvector_centrality(kite)              #特征向量中心性
print(dc)

plt.figure(figsize=(4,3))
pos = nx.spring_layout(kite,seed=40)
node_size=[100*100**i for i in dc.values()]
nx.draw(kite,pos=pos,with_labels=True,node_color='y',node_size=node_size)
```

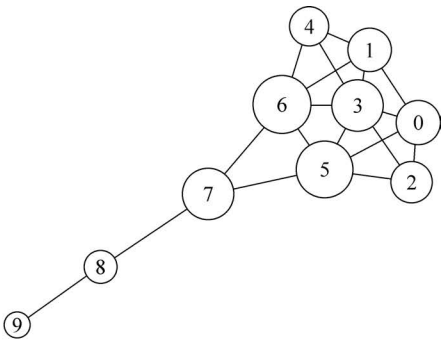
表 3.1 显示节点 3 的度中心性最大，节点 5 和节点 6 的接近度中心性最大，节点 7 的介数中心性最大，节点 3 的特征向量中心性最大，如图 3.7 所示。

表 3.1　风筝网络不同节点重要性指标比较

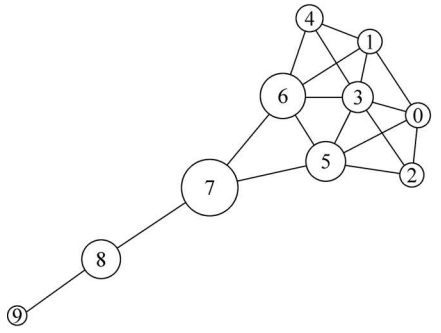
	度中心性	接近度中心性	介数中心性	特征向量中心性
0	0.44	0.53	0.02	0.35
1	0.44	0.53	0.02	0.35
2	0.33	0.5	0.0	0.29
3	**0.67**	0.6	0.1	**0.48**
4	0.33	0.5	0.0	0.29
5	0.56	**0.64**	0.23	0.4
6	0.56	**0.64**	0.23	0.4
7	0.33	0.6	**0.39**	0.2
8	0.22	0.43	0.22	0.05
9	0.11	0.31	0	0.01



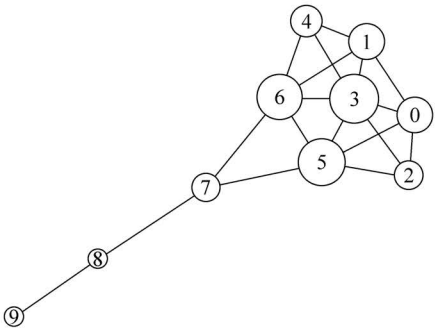
(a) 度中心性



(b) 接近度中心性



(c) 介数中心性



(d) 特征向量中心性

图 3.7　依次为度中心性、接近度中心性、介数中心性和特征向量中心性

nx.eigenvector_centrality(G[,max_iter,tol,…])：计算图 G 的特征向量中心性。

nx.eigenvector_centrality_numpy(G[,weight,…])：计算图 G 的特征向量中心性。

3.1.5 H 指数

信息计量学中用 H 指数衡量学者的贡献：如果一个人在其所有学术文章中最多有 n 篇论文分别被引用了至少 n 次，它的 H 指数就是 n。网络上的 H 指数是指：如果该节点的邻居中最多有 n 个邻居且这些邻居的度至少为 n。也可以定义为满足以下条件的最大 n 值，节点有至少 n 个邻居并且这 n 个邻居的度值都大于或等于 n。

先判断一个序列中任意的 n 是否满足值大于或等于 n 的元素的个数也大于 n。

```
def H_index_n(l,n):
  n_temp = 0
  for i in l:
    if i>=n:
      n_temp +=1
  if n_temp>=n:
    return True
  else:
    return False
```

寻找满足条件的 n 的最大值如下所示。

```
def H_index_list(l):
  results = []
  for n in range(len(l)):
    results.append(H_index_n(l,n))
  return sum(results)-1
```

求一个网络中每个节点的 H 指数如下所示。

```
def H_index_network(network):
  score = {}
  for k in network.nodes:
    degree_list = [j for i,j in network.degree(list(network.neighbors(k)))]
    score[k] = H_index(degree_list)
  return score
```

【例 3-6】 使用 H 指数分析风筝网络中节点的中心性，如图 3.8 所示。

```
kite = nx.krackhardt_kite_graph()
print(H_index_network(kite))
```

运行结果如下：

```
{0: 3, 1: 3, 2: 2, 3: 4, 4: 2, 5: 3, 6: 3, 7: 2, 8: 1, 9: 0}
```

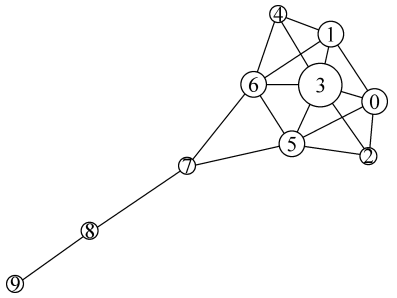


图 3.8 H 指数中心性

```
#使用H指数进行可视化
import matplotlib.pyplot as plt
plt.figure(figsize=(4,3))
pos = nx.spring_layout(kite,seed=40)
```

```
node_size = [10 * 3 ** i for i in list(H_index_network(kite).values())]
nx.draw(kite,pos = pos,with_labels = True,node_size = node_size,node_color = 'y')
```

3.1.6　k-壳分解

k-壳分解可类比剥洋葱，逐渐剥去外部每一层壳。假设网络中不存在度值为 0 的孤立节点。这样从度中心性的角度看，度为 1 的节点就是网络中最不重要的节点。所以该方法先把所有度值为 1 的节点以及与这些节点相连的边都去掉，这时网络中可能又会出现一些新的度值为 1 的节点，我们就再把这些节点及其相连的边去掉，重复这种操作，直至网络中不再有度值为 1 的节点为止，如图 3.9 所示。

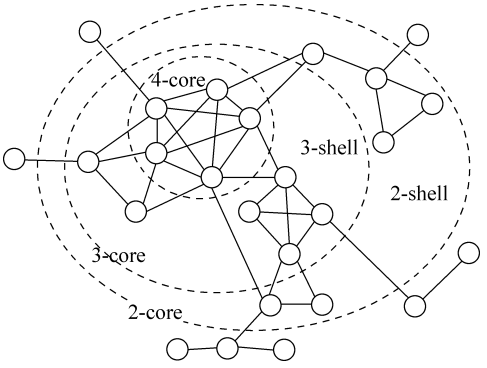


图 3.9　k-核和 k-壳

以此类推，按照同样的方法再去掉网络中度值为 2 的节点，一直进行下去，直到所有节点都被去除。为了更好地描述这一过程，引入 k-壳、k-核和 k-皮的概念。

0-壳(0-shell)：网络中度为 0 的孤立节点。

1-壳(1-shell)：所有被去除的度为 1 的节点以及它们之间的连边。

2-壳(2-shell)：重复把网络中度值为 2 的节点及其相连的边去掉直至不再有度值为 2 的节点为止。

以此类推，可以进一步得到指标更高的壳，直至网络中的每个节点最后都被划分到相应的 k-壳中，就得到了网络的 k-壳分解。

网络中的每个节点对应唯一的 k-壳指标 k_s，并且 k_s 壳中所包含的节点的度值必然满足 $k \geqslant k_s$。

在得到一个网络的 k-壳分解之后，我们把所有 $k_s \geqslant k$ 的 k-壳的并集称为网络的 k-核(k-core)，把指标 $k_s \leqslant k$ 的 k-壳的并集称为网络的 k-皮(k-crust)。

k-核的一个等价定义是：它是一个网络中所有度值不小于 k 的节点组成的连通片。基于这一定义，我们可以按照如下方法得到 k-核。

首先去除网络中度值小于 k 的所有节点及其连边；如果在剩下的节点中仍然有度值小于 k 的节点，那么就继续去除这些节点，直至网络中剩下的节点的度值都不小于 k。依次取 $k=1,2,3,\cdots$，对原始网络重复这种去除操作，就得到了该网络的 k-核分解(k-core decomposition)。对于一个连通网络，1-核实际上就是整个网络，$(k+1)$-核一定是 k-核的子集。k-核：k-壳往内的所有点，即 k-壳指标大于或等于 k；k-皮：k-壳往外的所有点，即 k-壳指标小于或等于 k；属于 k-核但不属于$(k+1)$-核的所有节点就是 k-壳中的节点。

【例 3-7】 使用 k-壳分解分析图 3.10 所示网络的节点中心性。

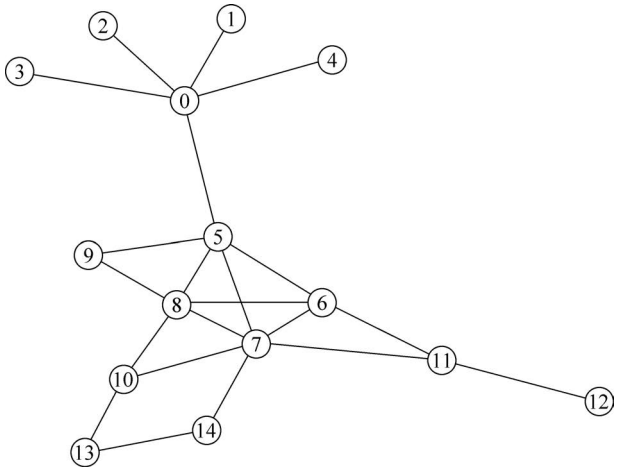


图 3.10 k-壳分解示例

```
#构建网络
edge_list = [(0,1),(0,2),(0,3),(0,4),(0,5),(5,6),(5,7),(5,8),(6,7),(6,8),(7,8),(5,9),\
 (8,9),(7,10),(8,10),(6,11),(7,11),(11,12),(10,13),(7,14),(13,14)]
kgraph = nx.Graph()
kgraph.add_edges_from(edge_list)
#可视化
pos = nx.spring_layout(kgraph,seed = 20)
nx.draw_networkx_nodes(kgraph,pos = pos,nodelist = nx.k_shell(kgraph,1),node_color = 'yellow')
nx.draw_networkx_nodes(kgraph,pos = pos,nodelist = nx.k_shell(kgraph,2),node_color = 'lime')
nx.draw_networkx_nodes(kgraph,pos = pos,nodelist = nx.k_shell(kgraph,3),node_color =
'deepskyblue')
nx.draw_networkx_edges(kgraph,pos = pos)
nx.draw_networkx_labels(kgraph,pos = pos)
plt.axis('off')
plt.show()
```

k-壳分解如图 3.11 所示。

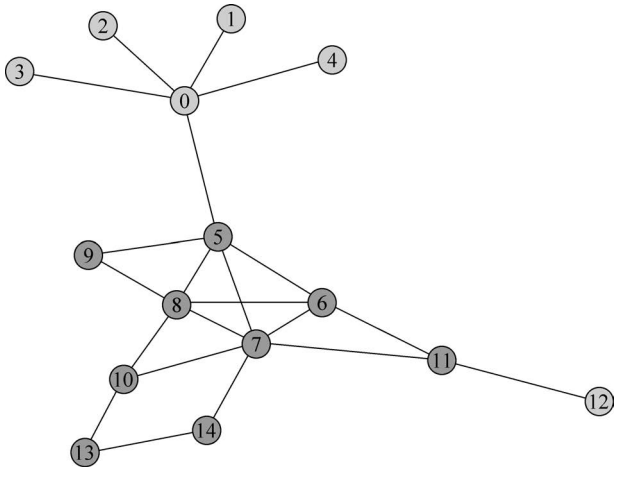


图 3.11 k-壳分解：1-壳黄色，2-壳绿色，3-壳蓝色

计算各个节点的核数如下所示。

```
>>> print(nx.core_number(kgraph))
```

运行结果如下：

```
{0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 3, 6: 3, 7: 3, 8: 3, 9: 2, 10: 2, 11: 2, 12: 1, 13: 2, 14: 2}
```

计算 k-核如下所示。

```
1  for i in range(1,4):
2    print(list(nx.k_core(kgraph,i)))
```

运行结果如下。

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[5, 6, 7, 8, 9, 10, 11, 13, 14]
[8, 5, 6, 7]
```

计算 k-壳如下所示。

```
1  for i in range(1,4):
2    print(list(nx.k_shell(kgraph,i)))
```

运行结果如下：

```
[0, 1, 2, 3, 4, 12]
[9, 10, 11, 13, 14]
[8, 5, 6, 7]
```

计算 k-皮如下所示。

```
1  for i in range(1,4):
2    print(list(nx.k_crust(kgraph,i)))
```

运行结果如下：

```
[0, 1, 2, 3, 4, 12]
[0, 1, 2, 3, 4, 9, 10, 11, 12, 13, 14]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

nx.core_number(G)：返回每个节点的核数。

nx.k_core(G[,k,core_number])：返回图 G 的 k-core。

nx.k_shell(G[,k,core_number])：返回图 G 的 k-shell。

nx.k_crust(G[,k,core_number])：返回图 G 的 k-crust。

需要说明的是，还有很多节点的重要性指标（如 11.3 节的基于随机游走的若干指标），此处不再一一介绍。此外，后续会有新的中心性指标被不断提出。

3.2　有向网络节点重要性指标

在实际应用中，有两个重要的有向网络：引文网络和 WWW 网络。在论文引用网络中，一篇论文的出度是它的参考文献的数量，而入度是该论文的他引次数。显然，即使一篇论文的出度很大，即参考文献数量很多，也不能反映该论文是否一定重要，否则每个人都可以轻而易

举地写出重要的文章了。评价一篇论文是否重要更为合理的标准应该是它的入度(即他引次数)。当然,更进一步的考虑是,一篇论文是否重要不仅要看有多少别人的论文引用它,还要看其中有多少重要的论文引用它。当你在 Google、百度或者 Bing 等搜索引擎网站上输入一个关键词后,搜索引擎就会基于某种排序算法对与该关键词有关的网页按照某种重要性指标进行排序。搜索引擎领域的两个算法是 Cornell 大学的 Kleinberg 提出的 HITS(Hyperlink-Induced Topic Search)算法以及经典 Google 创始人 Page 和 Brin 提出的 PageRank 算法。接下来,将分别介绍这两个算法。

3.2.1 HITS 算法

HITS 算法的基本思想是:每个节点的重要性有两个刻画指标:权威性(Authority)和枢纽性(Hub)。

权威中心性。一个网页的权威值由指向该页面的其他页面的枢纽值来刻画:如果一个页面被多个具有高枢纽值的页面所指向,那么该页面就具有高的权威值。举例:重要论文、官方网站等。

枢纽中心性。一个网页的枢纽值由它所指向的页面的权威值来刻画:如果一个页面指向多个具有权威值的页面,那么该网页就具有高的枢纽值。举例:综述性论文、导航网站等。

HITS 算法。

(1) 初始步:设定网络中所有节点的权威值和枢纽值的初始值 $x_i(0), y_i(0), i=1,2,\cdots,N$。

(2) 迭代过程:在第 k 步($k\geqslant 1$)进行如下 3 种操作。

① 权威值校正规则:每个节点的权威值校正为指向它的节点的枢纽值之和,即

$$x'_i(k)=\sum_{j=1}^{N}a_{ji}y_j(k-1),\quad i=1,2,\cdots,N \tag{3-13}$$

② 枢纽值校正规则:每个节点的枢纽值校正为它所指向的节点的权威值之和,即

$$y'_i(k)=\sum_{j=1}^{N}a_{ij}x'_j(k),\quad i=1,2,\cdots,N \tag{3-14}$$

③ 归一化:

$$x_i(k)=\frac{x'_i(k)}{\|x'(k)\|},\quad y_i(k)=\frac{y'_i(k)}{\|y'(k)\|},\quad i=1,2,\cdots,N \tag{3-15}$$

【例 3-8】 运用 HITS 算法分析图 3.12 所示网络中节点的权威性和枢纽性。

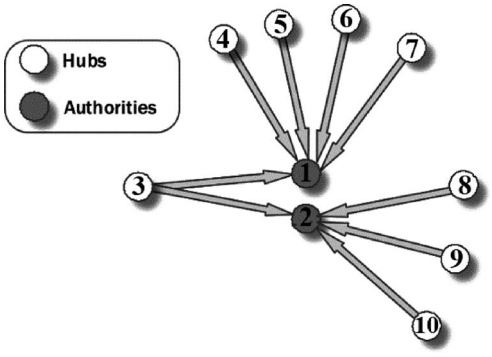


图 3.12 HITS 实例图

表 3.2 显示，节点 1 和节点 2 的权威性最大，节点 3 的枢纽性最大。

表 3.2　HITS 算法执行过程

		1	2	3	4	5	6	7	8	9	10
初值	A	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	H	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
第一次迭代	A	0.5	0.4	0	0	0	0	0	0	0	0
	H	0	0	0.9	0.5	0.5	0.5	0.5	0.4	0.4	0.4
归一化	A	0.56	0.44	0	0	0	0	0	0	0	0
	H	0	0	0.22	0.12	0.12	0.12	0.12	0.1	0.1	0.1
第二次迭代	A	0.7	0.52	0	0	0	0	0	0	0	0
	H	0	0	1.22	0.7	0.7	0.7	0.7	0.52	0.52	0.52
归一化	A	0.57	0.43	0	0	0	0	0	0	0	0
	H	0	0	0.22	0.13	0.13	0.13	0.13	0.09	0.09	0.09
第三次迭代	A	0.74	0.49	0	0	0	0	0	0	0	0
	H	0	0	1.23	0.74	0.74	0.74	0.74	0.49	0.49	0.49
归一化	A	**0.6**	**0.4**	0	0	0	0	0	0	0	0
	H	0	0	**0.22**	0.13	0.13	0.13	0.13	0.09	0.09	0.09

```
1  #建网
2  ha = nx.DiGraph()
3  node_list = list(range(1,11))
4  edge_list = [(3,1),(3,2),(4,1),(5,1),(6,1),(7,1),(8,2),(9,2),(10,2)]
5  ha.add_nodes_from(node_list)
6  ha.add_edges_from(edge_list)
7  #可视化
8  plt.figure(figsize=(4,3))
9  pos = nx.spring_layout(ha,seed=20)
10 node_color = ['lime','lime','deepskyblue','yellow','yellow','yellow','yellow','yellow','yellow',
   'yellow']
11 nx.draw(ha,pos=pos,with_labels=True,node_color=node_color)
```

HITS 结果展示如图 3.13 所示。

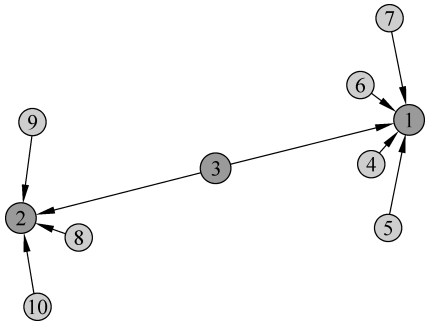


图 3.13　HITS 结果展示

```
1  >>> print(nx.hits(ha))
```

运行结果如下：

```
({3: 0.21654236465910043, 1: 0.0, 2: 0.0, 4: 0.1338305413635981, 5: 0.1338305413635981, 6:
0.1338305413635981, 7: 0.1338305413635981, 8: 0.08271182329550232, 9:
0.08271182329550232, 10: 0.08271182329550232}, {3: 0.0, 1: 0.6180339887498949, 2:
0.3819660112501051, 4: -1.4922561934464395e-17, 5: -2.0118284120836512e-17, 6:
1.6094643021610798e-17, 7: -1.1148380522407496e-17, 8: 2.827242258312061e-17, 9:
-5.154961167288202e-18, 10: 5.189517421663977e-17})
```

nx.hits(G[, max_iter, tol, nstart, normalized])：返回节点的 HITS 中心值和权威值。

3.2.2 PageRank 算法

PageRank 算法的基本想法是：WWW 上一个页面的重要性取决于指向它的其他页面的数量和质量。被有重要影响的节点指向的节点，其从重要节点获得的中心性会因为与其他节点共享而被稀释。

1. 基本的 PageRank 算法

基本的 PageRank 算法。

(1) 初始步：给定所有节点的初始 PageRank 值(简称 PR 值) $PR_i(0)$，$i=1,2,\cdots,N$，满足：

$$\sum_{i=1}^{N} PR_i(0) = 1 \tag{3-16}$$

(2) 基本的 PageRank 校正规则：把每个节点在第 $k-1$ 步时的 PR 值平分给它所指向的节点。也就是说，如果节点 i 的出度为 k_i^{out}，那么节点 i 所指向的每个节点分得的 PR 值为 $PR_i(k-1)/k_i^{out}$。如果一个节点的出度为 0，那么它就始终把 PR 值留给自己。每个节点的新的 PR 值校正为它所分得的 PR 值之和，即有

$$PR_i(k) = \sum_{j=1}^{N} a_{ji} \frac{PR_j(k-1)}{k_j^{out}}, \quad i = 1,2,\cdots,N \tag{3-17}$$

【例 3-9】 运用 PageRank 算法分析图 3.14 所示有向图中节点的重要性。

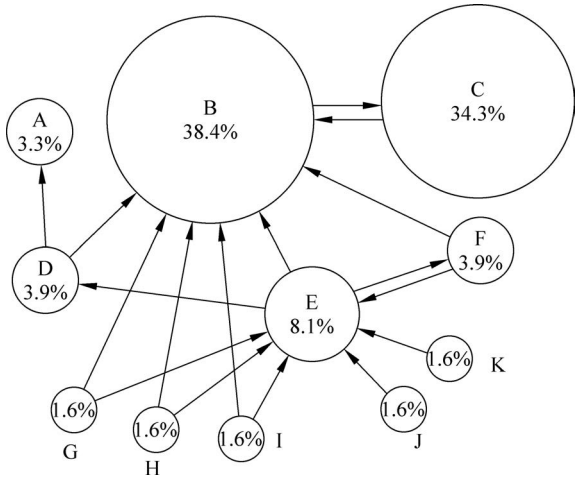


图 3.14 PageRank 示例图

从表 3.3 可以看出，节点 B 的 PR 值最大，节点 C 次之，紧接着是节点 A 和节点 E，接下来是节点 D 和节点 F，其余节点 PR 值为 0。

表 3.3　PageRank 算法执行过程

	A	B	C	D	E	F	G	H	I	J	K
0	0.09	0.1	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09
1	0.045	0.345	0.1	0.03	0.36	0.03	0	0	0	0	0
2	0.015	0.25	0.345	0.12	0.015	0.12	0	0	0	0	0
3	0.0075	0.485	0.25	0.005	0.06	0.005	0	0	0	0	0
4	0.0025	0.275	0.485	0.02	0.0025	0.02	0	0	0	0	0
5	0.01	**0.506**	0.275	0.00125	0.01	0.00125	0	0	0	0	0

```
#建网
node_list = ['A','B','C','D','E','F','G','H','I','J','K']
edge_list = [('D','A'),('C','B'),('D','B'),('E','B'),('F','B'),('G','B'),('G','B'),('H','B'),('I','B'),('B',
'C'),('E','D'),('F','E'),('G','E'),('H','E'),('I','E'),('J','E'),('K','E'),('E','F')]
pg = nx.DiGraph()
pg.add_nodes_from(node_list)
pg.add_edges_from(edge_list)
pg_score = nx.pagerank(pg)
print(pg_score)
#绘图
node_size = [100 * 1000 ** i for i in pg_score.values()]
pos = nx.spring_layout(pg,seed = 160)
nx.draw(pg,pos = pos,with_labels = True,node_size = node_size,\
    node_color = list(pg_score.values()),cmap = 'tab20_r')
```

运行结果如下：

```
{'A': 0.03278149315934399, 'B': 0.38439863456604384, 'C': 0.3429125997558898, 'D':
0.039087092099966095, 'E': 0.08088569323449774, 'F': 0.039087092099966095, 'G':
0.016169479016858404, 'H': 0.016169479016858404, 'I': 0.016169479016858404, 'J':
0.016169479016858404, 'K': 0.016169479016858404}
```

显然，所得结果与表 3.3 并不相同，后面将对基本的 PageRank 算法进行修正。PageRank 分析结果的可视化展示如图 3.15 所示，节点的大小与节点的重要性成比例，越重要的节点圆圈的大小越大。

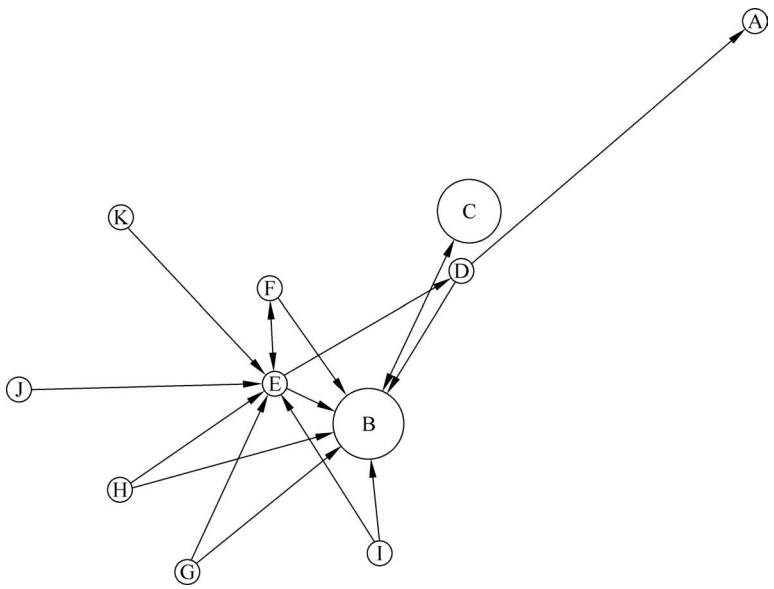


图 3.15　PageRank 结果展示

2. PageRank 算法与随机游走的等价性

运用随机游走的观点解释基本的 PageRank 算法。首先，完全随机地选择一个初始节点；然后，从当前节点出发，从该节点指出去的边中随机选择一条边并沿着该边到达另一个节点。可以证明：随机游走 k 步后位于节点 i 的概率等于应用基本 PageRank 算法 k 步后所得到的节点 i 的 PR 值。

从而可以使用随机游走描述 PageRank 算法，如使用转移矩阵的转置与当前 PR 值相乘可以得到转移后的 PR 值

$$\mathrm{PR}(t+1)=\bar{\boldsymbol{A}}^{\mathrm{T}}\mathrm{PR}(t) \tag{3-18}$$

其中，$\mathrm{PR}(t)$表示第 t 次迭代后的 PR 值，$\bar{\boldsymbol{A}}=(\bar{a}_{ij})_{N\times N}$ 表示转移矩阵，$\bar{a}_{ij}$ 表示从页面 i 转移到页面 j 的概率。

$$\bar{a}_{ij}=\begin{cases}1/k_i^{\mathrm{out}}, & \text{如果有节点 } i \text{ 指向节点 } j \text{ 的边}\\ 0, & \text{否则}\end{cases} \tag{3-19}$$

多次相乘后可以得到最终的平稳分布(详细分析参照 11.2 节)：

$$\mathrm{PR}(n)=\bar{\boldsymbol{A}}^{\mathrm{T}}\mathrm{PR}(n) \tag{3-20}$$

上述随机游走规则的缺陷在于：一旦到达某个出度为零的节点，就会永远停留在该节点而无法再走出来。出度为零的节点也称为悬挂节点(Dangling node)，这些节点的存在会使基本的 PageRank 算法失效。

【例 3-10】 一个只包含两个节点和一条边的例子如图 3.16 所示。

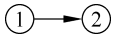


图 3.16 PageRank 中的悬挂节点

转移矩阵可以表示为

$$\bar{\boldsymbol{A}}=\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

初始 PR 值取为 $\mathrm{PR}(0)=[1/2 \quad 1/2]^{\mathrm{T}}$，

第一转移后的结果

$$\mathrm{PR}(1)=\bar{\boldsymbol{A}}^{\mathrm{T}}\mathrm{PR}(0)=\begin{bmatrix}0\\ 1/2\end{bmatrix}$$

第二次转移后的结果

$$\mathrm{PR}(2)=\bar{\boldsymbol{A}}^{\mathrm{T}}\mathrm{PR}(1)=\begin{bmatrix}0\\ 0\end{bmatrix}$$

经过两轮迭代之后，网络中两个节点的 PR 值全部稳定为零。没有出度的节点像黑洞一样，吸尽了网络中的 PR 值。因此，需要对算法进行适当修正。

3. PageRank 算法的随机性修正

假设一旦到达一个出度为零的页面，就以相同概率 $1/N$ 随机地访问网络中的任一页面。也就是说，把转移矩阵 $\bar{\boldsymbol{A}}$ 中的全零行替换为每个元素均为 $1/N$ 的行。

$$\bar{a}_{ij}=\begin{cases}1/k_i^{\mathrm{out}}, & \text{如果 } k_i^{\mathrm{out}}>0 \text{ 且有从节点 } i \text{ 指向节点 } j \text{ 的边}\\ 0, & \text{如果 } k_i^{\mathrm{out}}>0 \text{ 且没有从节点 } i \text{ 指向节点 } j \text{ 的边}\\ 1/N, & \text{如果 } k_i^{\mathrm{out}}=0\end{cases}$$

对于例 3-10 而言，随机修正后的转移矩阵为

$$\overline{\boldsymbol{A}}=\begin{bmatrix}0 & 1\\ \frac{1}{2} & \frac{1}{2}\end{bmatrix}$$

初始 PR 值仍取为 $\mathrm{PR}(0)=[1/2 \quad 1/2]^{\mathrm{T}}$，可以求得稳态 PR 值为 $\mathrm{PR}^{*}=[1/3 \quad 2/3]^{\mathrm{T}}$。

4. 最终的 PageRank 算法

虽然引入了随机性修正，但基本的 PageRank 算法仍然可能存在收敛性问题，有时会出现周期解。因此还需要进一步修正：从当前页面出发，不管该页面是否为悬挂页面，都允许以一定概率随机选取网络中的任一页面作为下一步要浏览的页面。

针对一般的有向网络，应使用如下的修正规则：完全随机地选择一个初始节点。如果当前所在节点的出度大于零，那么以概率 $s(0<s<1)$ 在指出去的边中随机选择一条边并沿着该边到达下一个节点，以概率 $1-s$ 在整个网络上完全随机选择一个节点作为下一步要到达的节点。如果当前所在节点的出度等于零，那么完全随机选择一个节点作为下一步要到达的节点。

PageRank 算法

(1) 初始步：给定所有节点的初始 PageRank 值(简称 PR 值)$\mathrm{PR}_i(0)$，$i=1,2,\cdots,N$，满足 $\sum_{i=1}^{N}\mathrm{PR}_i(0)=1$。

(2) 修正的 PageRank 校正规则(简称 PageRank 校正规则)：给定一个标度常数 $s\in(0,1)$。首先按照基本的 PageRank 校正规则计算各个节点的 PR 值，然后把每个节点的 PR 值通过比例因子 s 进行缩减。这样，所有节点的 PR 值之和也就缩减为 s，再把 $1-s$ 平均分给每个节点的 PR 值，以保持网络总的 PR 值为 1。即有

$$\mathrm{PR}_i(k)=s\sum_{j=1}^{N}a_{ji}\frac{\mathrm{PR}_j(k-1)}{k_j^{\mathrm{out}}}+(1-s)\frac{1}{N},\quad i=1,2,\cdots,N \tag{3-21}$$

5. 常数 s 的取值

关于标度常数 s 的取值需要考虑到收敛性和有效性之间的折中：如果 $s=1$，那么算法会无法收敛，s 越接近 1 算法收敛速度越慢；s 越接近 0 算法收敛速度越快，如果 $s=0$，那么算法一步就收敛到所有节点均具有相同 PR 值的状态，但收敛值缺乏有效的意义。Page 和 Brin 当初提出 PageRank 算法时，建议取 $s=0.85$。

nx.pagerank(G[,alpha,personalization,…])：返回节点的 PageRank 值。

3.3　节点重要性衡量标准

衡量节点重要性的方法多种多样，如度中心性、接近度中心性、介数中心性和特征向量中心性等。诚然，各项指标有自身的适用范围和局限性，但研究者仍希望通过网络的某些动力学过程评价节点中心性的有效性，如网络的鲁棒性，网络上的疾病传播、免疫、同步和控制等。本节先介绍移除关键节点对网络鲁棒性的影响，疾病传播、免疫和同步等将在后续章节介绍。

网络的鲁棒性是指移除一定量的节点后观测网络连通性的变化。移除节点的方式一般分为两种：随机移除节点和依据某一中心性指标的降序排列依次移除节点。

3.3.1　静态鲁棒性

这里使用的观测量是，最大连通子图的相对大小，即找到网络中的最大连通片，然后用最

大连通片中节点的数量除以网络的总节点数。

【例 3-11】 分别用随机移除节点，依据度中心性、接近度中心性、介数中心性和特征向量中心性测试空手道俱乐部网络的鲁棒性。

```
import networkx as nx
import matplotlib.pyplot as plt
from random import choice
from random import sample
import numpy as np

def one_times_robustness_random(network,n):
    N = network.number_of_nodes()
    components_size = [len(i) for i in nx.connected_components(network)]
    max_components_size = [max(components_size)]
    for i in range(N//n-1): # -1 是为了防止恰好全部移除时求最大连通子图报错
        selected_nodes = sample(list(network.nodes),n)
        network.remove_nodes_from(selected_nodes)
        components_size = [len(i) for i in nx.connected_components(network)]
        max_components_size.append(max(components_size))
    return max_components_size
#随机移除每次都不一样，需要多次平均以保证结果的可靠性
def N_times_robuatness_random(network,n):
    N = network.number_of_nodes()
    max_components_sizes = []
    for i in range(20):                    #平均的次数
        network1 = network.copy()
        max_components_sizes.append(one_times_robustness_random(network1,n))
    return np.array(max_components_sizes).mean(axis=0)/N

def selected_robustness(network,remove_list,n):
    network1 = network.copy()
    N = network1.number_of_nodes()
    components_size = [len(i) for i in nx.connected_components(network1)]
    max_components_size = [max(components_size)]
    for i in range(N//n-1):
        network1.remove_nodes_from(remove_list[i*n:(i+1)*n])
        components_size = [len(i) for i in nx.connected_components(network1)]
        max_components_size.append(max(components_size))
    return np.array(max_components_size)/N

za = nx.karate_club_graph()
N = za.number_of_nodes()
n = 1                                      #每次移除的节点数量
x = np.arange(0,N,n)

remove_list_dc = [i[0] for i in sorted(nx.degree_centrality(za).items(), key=lambda x:
x[1],reverse=True)]
remove_list_cc = [i[0] for i in sorted(nx.closeness_centrality(za).items(),key=lambda x:
x[1],reverse=True)]
remove_list_bc = [i[0] for i in sorted(nx.betweenness_centrality(za).items(),key=lambda
x:x[1],reverse=True)]
remove_list_ec = [i[0] for i in sorted(nx.eigenvector_centrality(za).items(),key=lambda
x:x[1],reverse=True)]

plt.plot(x/N,N_times_robuatness_random(za,n),marker='o',label='random')
plt.plot(x/N,selected_robustness(za,remove_list_dc,n),marker='s',label='dc')
plt.plot(x/N,selected_robustness(za,remove_list_cc,n),marker='^',label='cc')
plt.plot(x/N,selected_robustness(za,remove_list_bc,n),marker='*',label='bc')
plt.plot(x/N,selected_robustness(za,remove_list_ec,n),marker='x',label='ec')
```

```
plt.rcParams['font.sans-serif'] = ['SimSun']
plt.xlabel('移除节点比率')
plt.ylabel('最大连通子图的相对大小')

plt.legend()
plt.show()
```

空手道俱乐部网络的鲁棒性如图 3.17 所示，可以看出依据某一中心性指标对网络连通程度的伤害性强于随机移除节点。各指标对网络的伤害则各不相同，会因具体的网络而异。

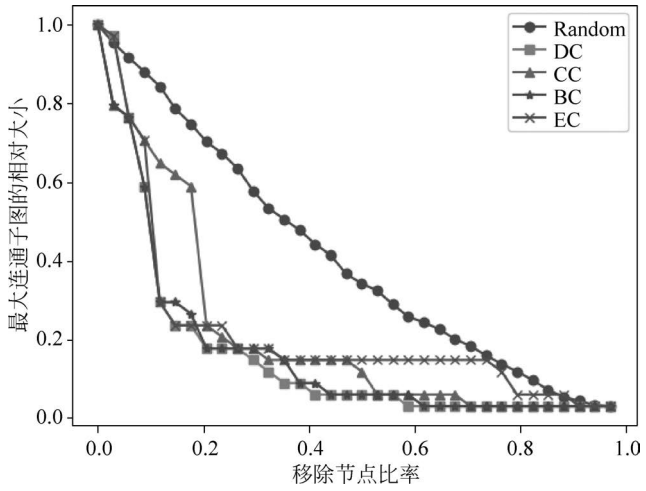


图 3.17　空手道俱乐部网络的鲁棒性

《悲惨世界》人物关系网络的鲁棒性如图 3.18 所示，显然依据某一指标的选择性失效对网络结构的破坏性强于随机移除。曲线的变化规律不同于图 3.17。

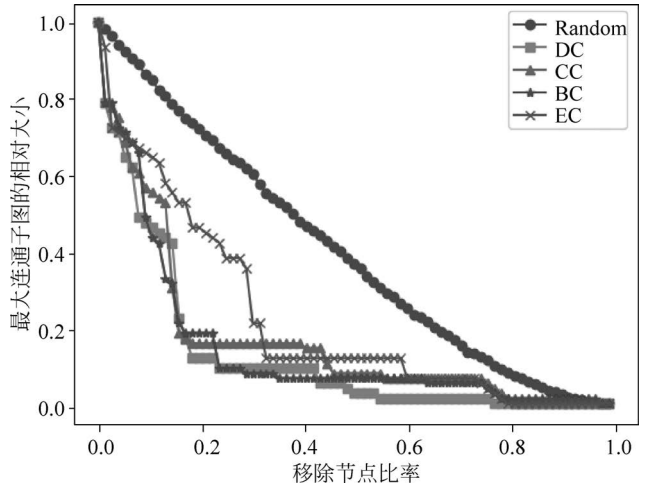


图 3.18　《悲惨世界》人物关系网络的鲁棒性

如果要比较网络鲁棒性的整体效果，则可以比较鲁棒性曲线与 x 轴和 y 轴所围成面积的大小。面积越小对网络连通性的伤害越大，中心性指标越好。可通过以下代码进行近似求解。

```
print(sum(N_times_robuatness_random(za,n))*(n/N),sum(selected_robustness(za,remove_list_
dc,n))*(n/N),sum(selected_robustness(za,remove_list_cc,n))*(n/N),sum(selected_robustness
(za,remove_list_bc,n))*(n/N),sum(selected_robustness(za,remove_list_ec,n))*(n/N))
```

从表 3.4 可以看出，无论是对于空手道俱乐部网络还是《悲惨世界》人物共现网络，度中心性表现最好，随机移除节点表现最差，但这一结论并不具有一般性，在实际研究中需要具体问题具体分析。

表 3.4　不同攻击策略的比较

	Random	DC	CC	BC	EC
karate_club	0.4263	**0.1713**	0.2223	0.1747	0.2266
les_miserables	0.4169	**0.1417**	0.1801	0.1594	0.2312

3.3.2　动态鲁棒性

有时也会考虑网络的动态鲁棒性。动态鲁棒性是指在考查某一中心性指标时，每次移除节点后重新计算网络中该指标的值，以此作为下次移除节点的依据。

【例 3-12】　比较《悲惨世界》人物共现网络中动态指标和静态指标的差异。

```
def selected_robustness_dynamic_dc(network,n):
    network1 = network.copy()
    N = network1.number_of_nodes()
    remove_list_dc = [i[0] for i in sorted(nx.degree_centrality(network1).items(),key =
    lambda x:x[1], reverse = True)]
    components_size = [len(i) for i in nx.connected_components(network1)]
    max_components_size = [max(components_size)]
    for i in range(N//n - 1):
        network1.remove_nodes_from(remove_list_dc[0:n])
        components_size = [len(i) for i in nx.connected_components(network1)]
        max_components_size.append(max(components_size))
        remove_list_dc = [i[0] for i in sorted(nx.degree_centrality(network1).items(), key =
        lambda x:x[1],reverse = True)]
    return np.array(max_components_size)/N
```

《悲惨世界》人物共现网络中静态鲁棒性和动态鲁棒性的比较如图 3.19 所示，显然，动态指标对网络连通性的破坏要强于静态指标。继续比较曲线与 x 轴和 y 轴围成面积的大小，如表 3.5 所示。

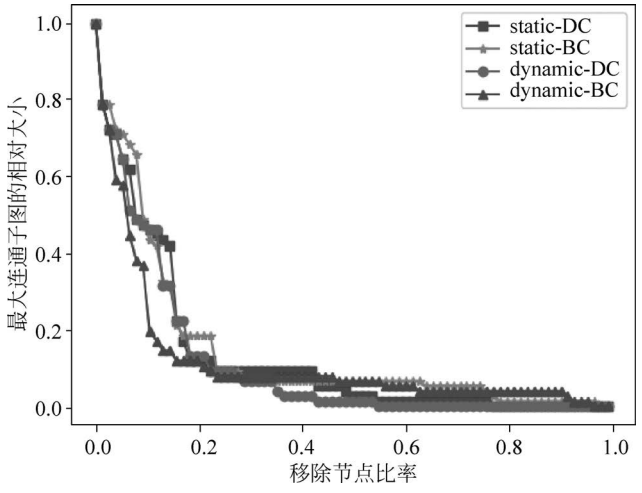


图 3.19　静态鲁棒性和动态鲁棒性

表 3.5　静态鲁棒性和动态鲁棒性的比较

	度 中 心 性	介数中心性
静态	0.1417	0.1594
动态	0.1258	0.1321

3.3.3　级联失效模型

有时还会分析网络的级联失效鲁棒性，每个节点同时包含负载和容量两个属性(初始时可依据某些假定规则进行初始化，如负载与节点的度值呈指数关系，而容量则和负载存在线性关系)，如果负载超过了容量节点就会失效，失效后节点上的负载不会消失而要被分配到它的邻居节点上，然后判断邻居节点是否负载超过容量从而引发进一步的失效，这种将失效逐层传递的过程被称为级联失效。接下来介绍一种基于度值的级联失效模型。

假设网络中任意节点 i 的初始负载 L_i 由式(3-22)确定：

$$L_i = k_i^{\alpha} \tag{3-22}$$

其中，α 为初始负载调节参数，k_i 为节点 i 的度值。

节点 i 的容量 C_i 与初始负载 L_i 线性相关，表达式如式(3-23)所示。

$$C_i = (1+\beta)L_i \tag{3-23}$$

其中，β 表示复杂容忍参数。当节点 i 的负载大于其容量时，该节点将会失效，它的负载将会按照一定比例再分配给邻居节点。节点 i 分配给它的邻居节点 j 的负载 $\Delta L_{i\to j}$ 为

$$\Delta L_{i\to j} = \frac{k_j^{\alpha}}{\sum\limits_{m\in\tau_i} k_m^{\alpha}} L_i \tag{3-24}$$

其中，τ_i 表示节点 i 的邻居节点集合。