WDS

DataScience@Web

机器学习讨论班

2018年暑期

社团发现

杨雨栋

8/24/2018

介绍内容

- 非重叠社团发现

GN算法

Newman快速算法

标签传播算法(LPA)

- 重叠社团发现

SLPA算法

派系过滤算法(CPM)

适应度算法

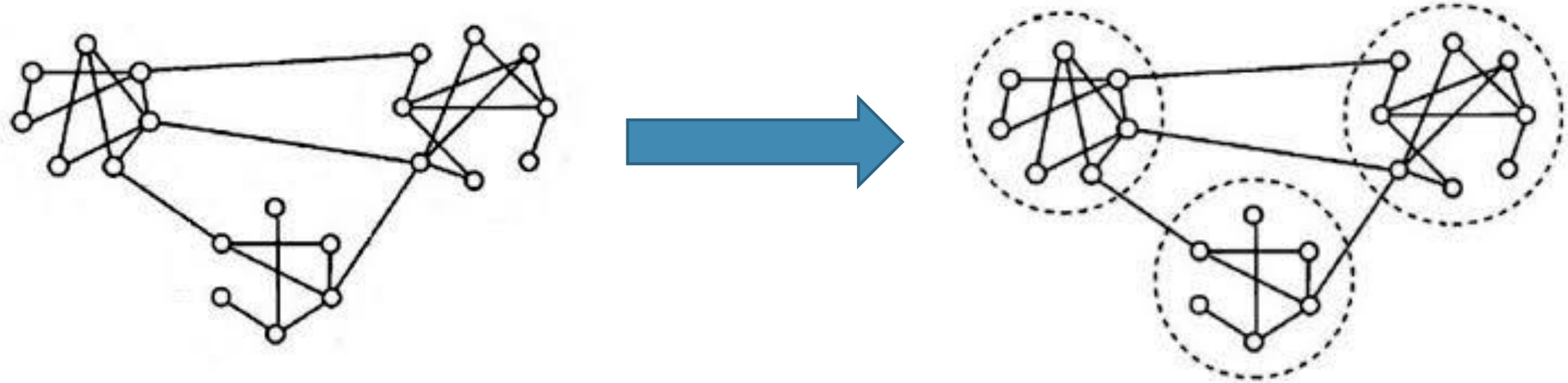
复杂网络

特点：

节点众多

连接关系复杂

社团发现(Community Detection)



特点：社团内部的连接相对稠密，社团之间的连接相对稀疏。

应用场景

社交网络

- 信息传播
- 精准营销

万维网

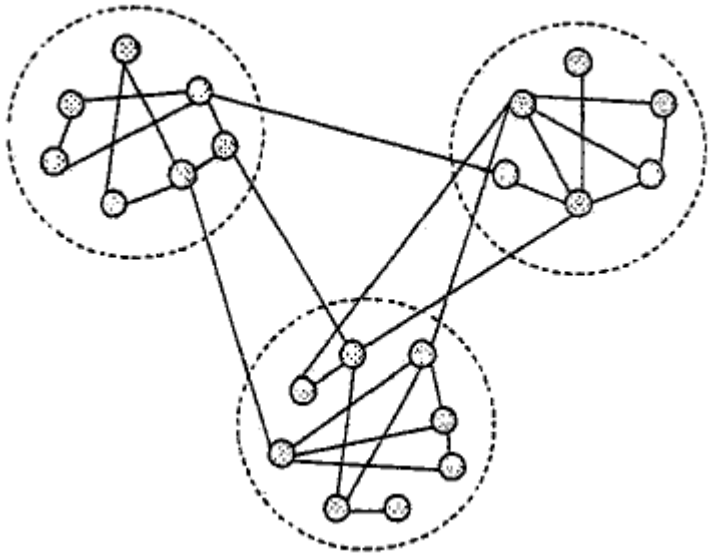
- 搜索性能
- 话题追踪
- 信息过滤

电路网络

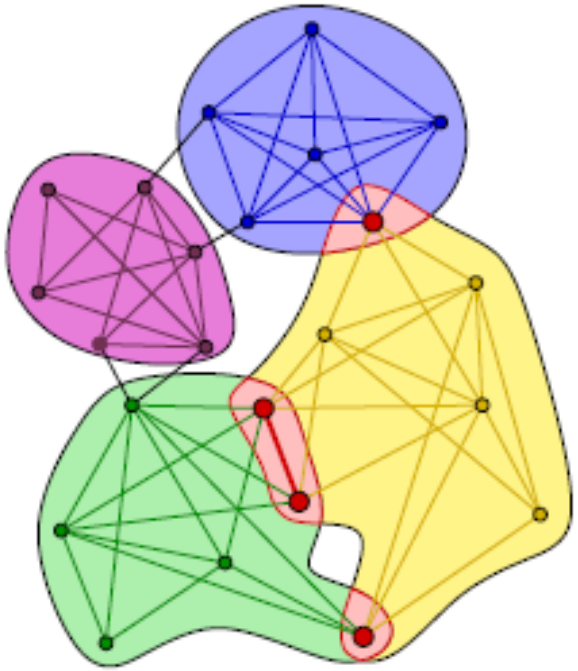
- 功能结构

社团发现分类

非重叠网络

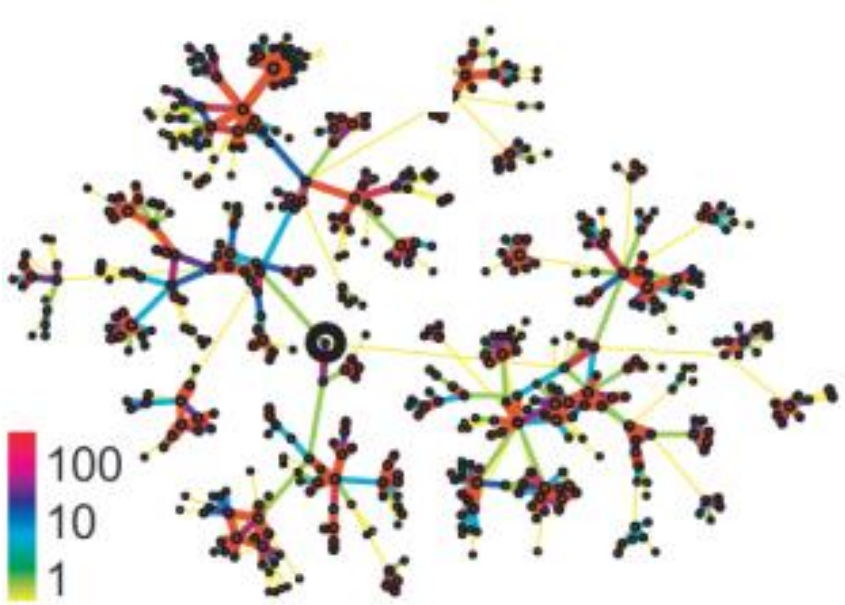


重叠网络

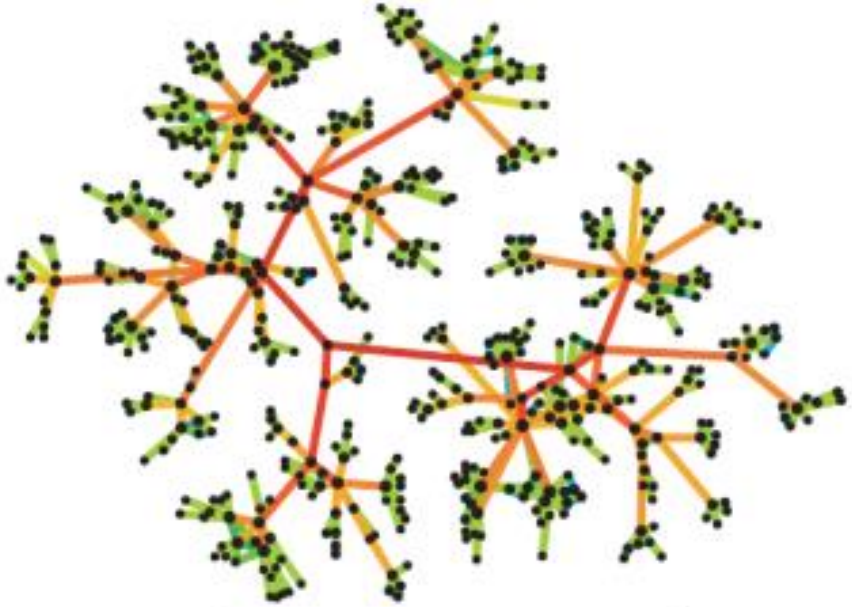


G-N算法

- 边介数(edge betweenness)：网络中通过某条边的最短路径的数目。



Edge strengths (call volume) in a real network



Edge betweenness in a real network

Girvan-Newman算法基本流程

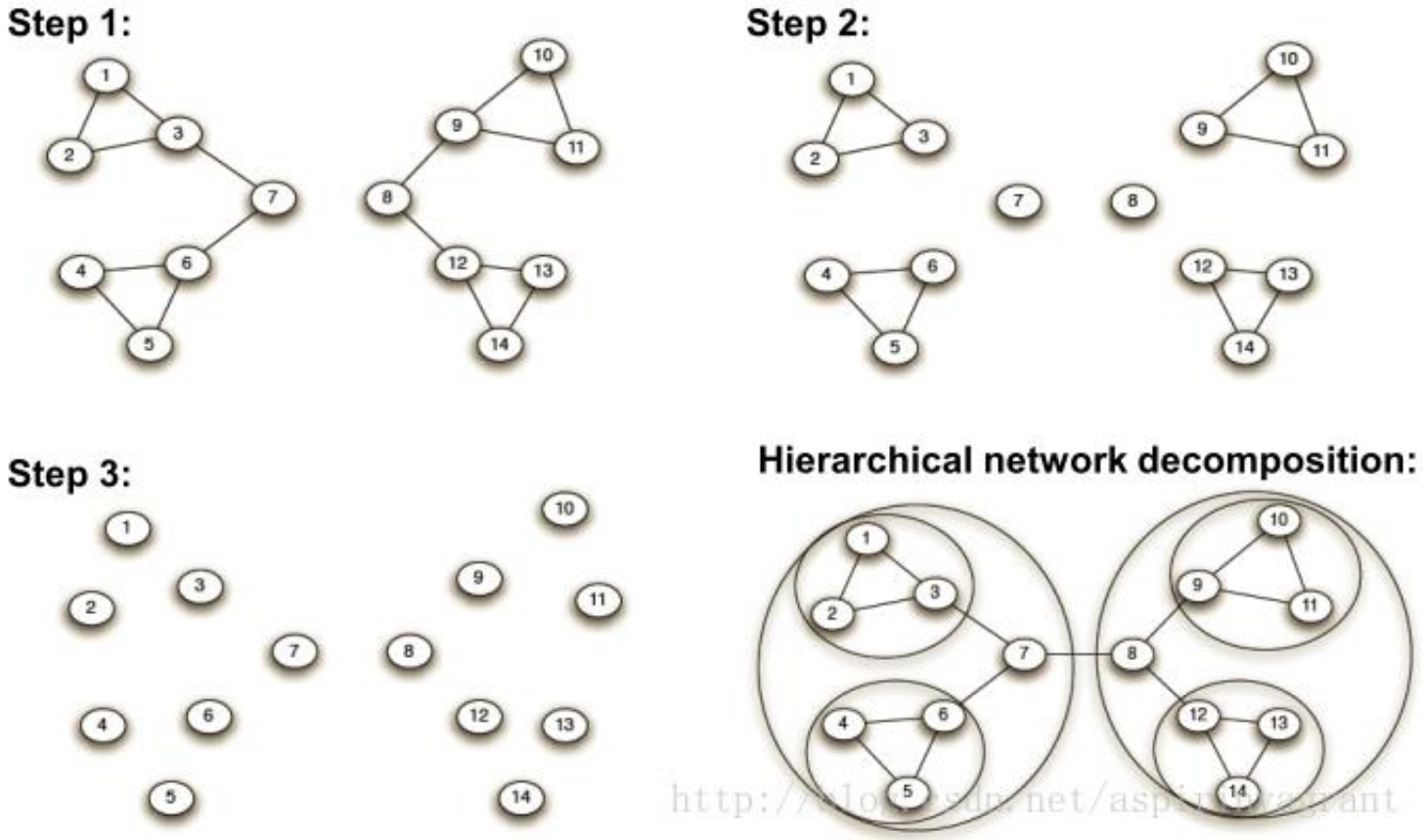
1. 计算网络中所有边的边介数；

2. 删除边介数最大的边；

3. 重新计算剩下的边的边介数

4. 重复步骤2和步骤3，直到每个节点成为一个独立的社区为止，即网络中没有边存在。

G-N算法



一个问题

划分到多少社团为止？

模块度(Modularity)

第一版：$Q = \sum_i (e_{ii} - a_i^2) = Tre - \|e^2\|$

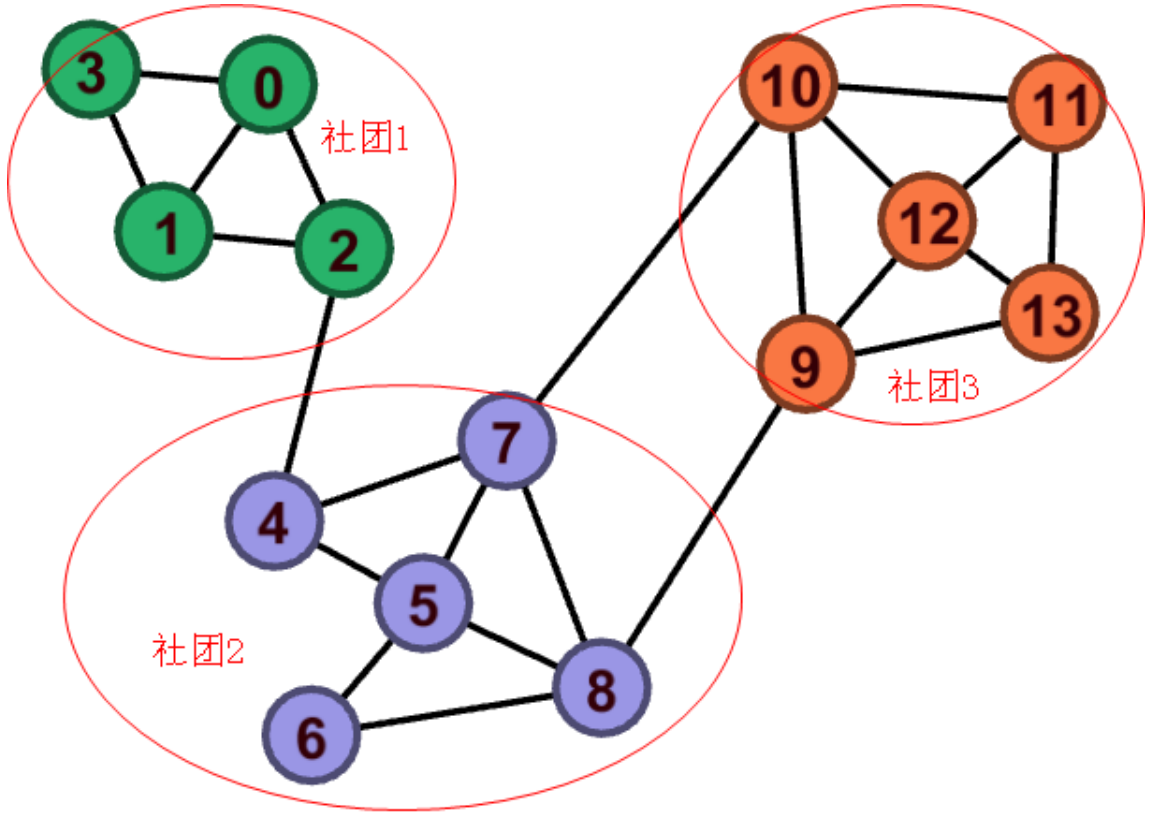
第二版：$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(i,j)$

第一版模块度Q

$$Q = \sum_i \left(e_{ii} - a_i^2\right) = Tre - \|e^2\|$$

$e_{ii} = \frac{e_i}{m}$ 社团i内边占比

$a_i = \frac{kc_i}{2m}$ 社团i内节点度数和占所有节点度数和的比例



第一版模块度Q

	1	2	3
1	$\frac{5}{23}$	$\frac{1}{2}\times\frac{1}{23}$	$\frac{1}{2}\times\frac{0}{23}$
2	$\frac{1}{2}\times\frac{1}{23}$	$\frac{7}{23}$	$\frac{1}{2}\times\frac{2}{23}$
3	$\frac{1}{2}\times\frac{0}{23}$	$\frac{1}{2}\times\frac{2}{23}$	$\frac{8}{23}$

$$Q=\frac{5}{23}+\frac{7}{23}+\frac{8}{23}-(\frac{11}{46})^2-(\frac{17}{46})^2-(\frac{18}{46})^2=\frac{553}{1058}=0.52268431$$

第二版模块度Q

- $Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(i,j)$

A_{ij}: 结点i和结点j之间边的数目

$\frac{k_i k_j}{2m}$: 随机放置边的情况下，结点i和结点j之间边数的期望值

$\delta(i,j)$: 节点i,j是否在同一社团

期望边数

随机连边满足条件：

1.每个结点度不变，最终总边数也不会变。因此随机连边后，图中的总边数期望值等于原来真实图中的边数。

$$\sum_{i,j} P_{ij} = \sum_{i,j} A_{ij} = 2m$$

2.对每个节点来说，它的度不会变。

$$\sum_{j} P_{ij} = k_i$$

P_{ij}:点i和点j之间连边的概率

$$P_{ij} = f(k_i)f(k_j)$$

$$\sum_j P_{ij} = \sum_j f(k_i)f(k_j) = f(k_i)\sum_j f(k_j) = k_i$$

设$f(k_i) = Ck_i$

$$P_{ij} = C^2 k_i k_j$$

$$\sum_{ij} P_{ij} = C^2 \sum_{ij} k_i k_j = 2m$$

$$\sum_{ij} k_i k_j = (k_1 + k_2 + \cdots + k_n)^2 = (2m)^2$$

$$\therefore C^2(2m)^2 = 2m$$

$$C = \frac{1}{\sqrt{2m}}$$

$$P_{ij} = C^2 k_i k_j = \frac{k_i k_j}{2m}$$

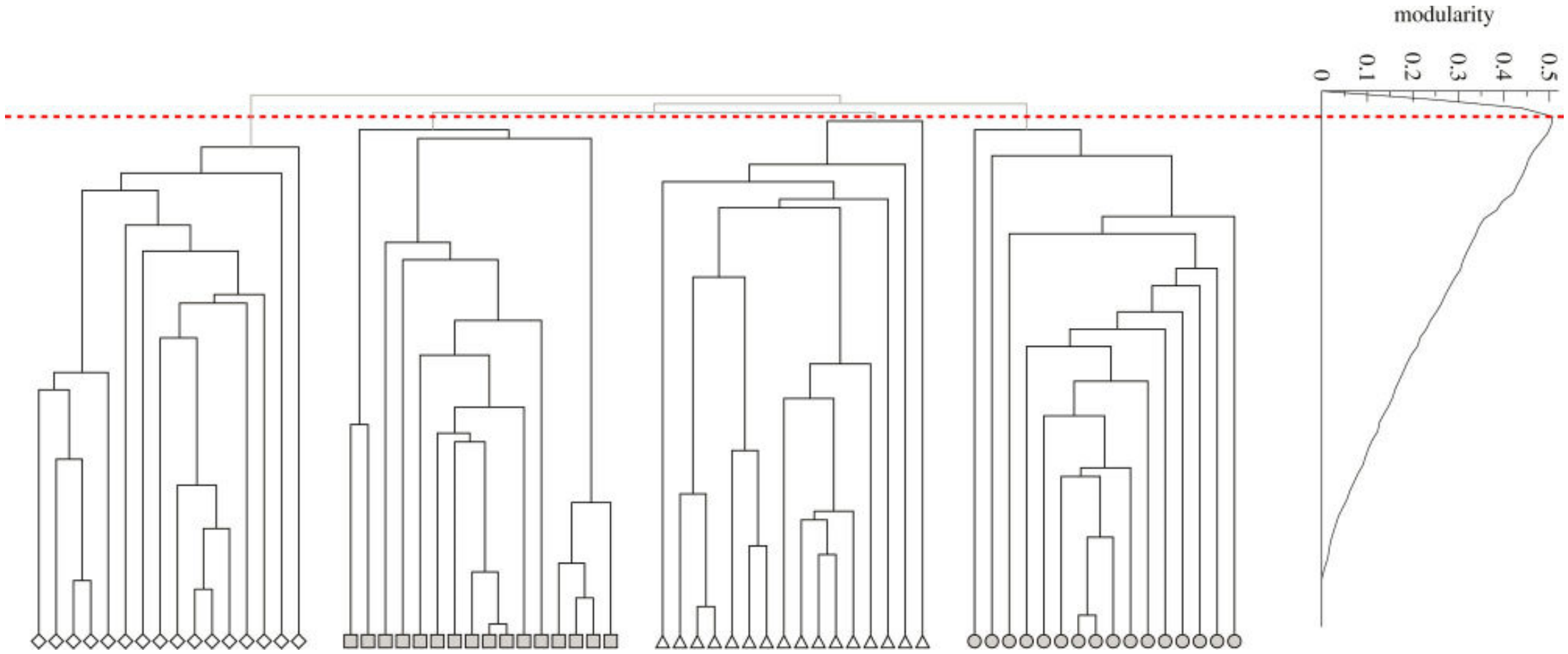
改进的G-N算法

1. 计算当前网络的边介数和Q值，并存储Q值和当前网络中社团分割情况。

2. 去除边介数最高的边。

3. 计算当前网络的Q值，如果此Q值比原来的大，则将现在的Q值和网络中社团分割情况存储更新，否则，进行下一次网络分割。

4. 所有边分割完毕，返回当前的Q值和社团分割情况。


modularity
0
0.1
0.2
0.3
0.4
0.5

G-N算法优缺点

■ 优点：

1. 在知道划分社团数量时，可以较为精确的给出社团的划分结果；

2. 可以给出不同程度的社团划分结果，可以揭示关于网络层次结构的信息；

■ 缺点：

在计算边介数的时候可能会有很多重复计算最短路径的情况，时间复杂度太高，只适用于中小型规模的网络；

Newman快速算法

- **基本思想：**首先将网络中的每个顶点设为一个单独社区，然后选出使得模块度Q的增值最大的社区对进行合并。如果网络中的顶点属于同一个社区，则停止合并过程。

- 设网络有n个节点，m条边，每一步合并对应的社区数目为r，组成一个r*r矩阵e.

- 矩阵元素e_{ij}:社区i中的节点与社区j中节点之间连边的数目在网络总变数的百分比。

Newman快速算法

1. 初始化网络，网络有n 个社区，初始化e_{ij}和a_i:

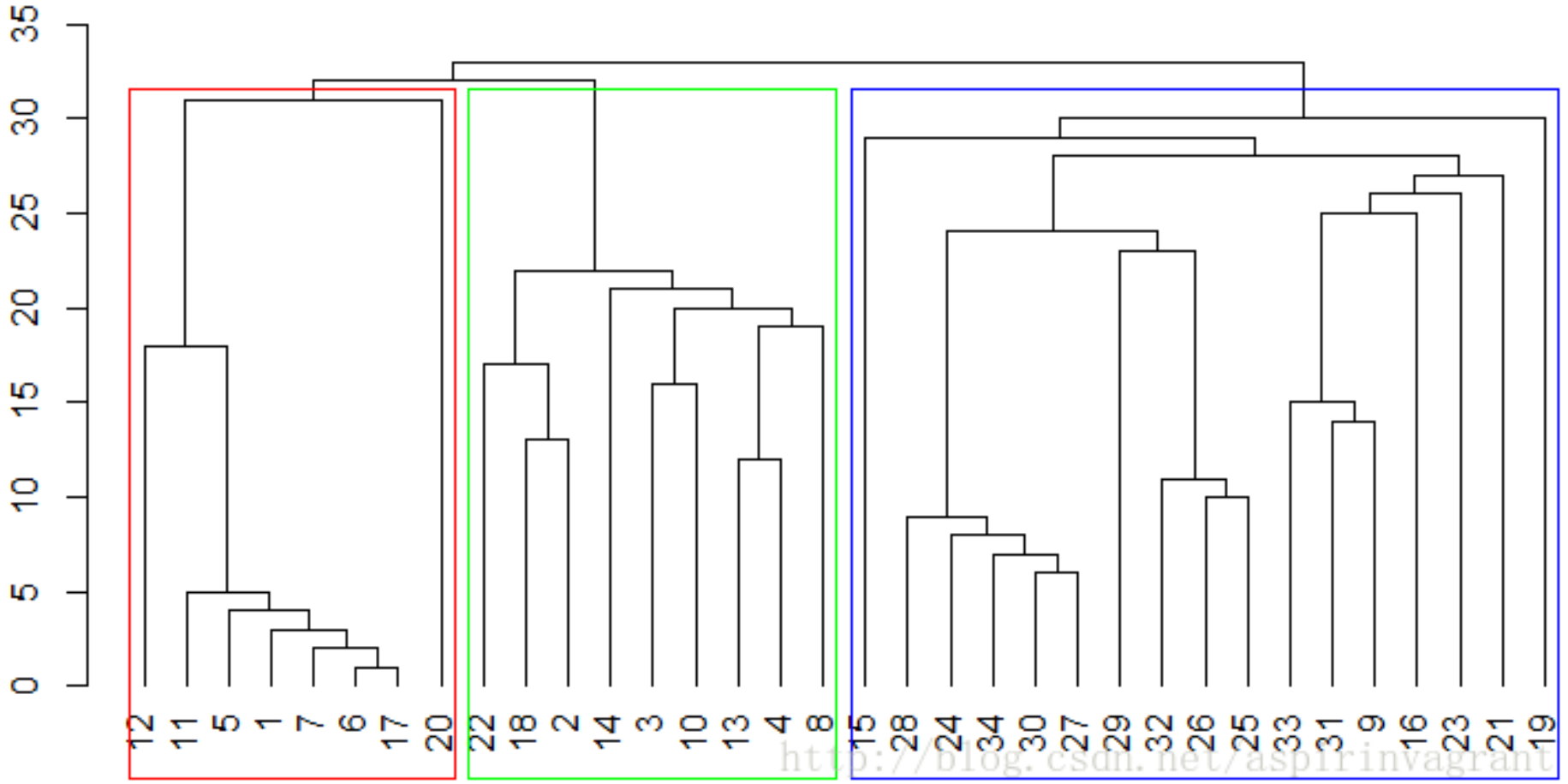
$$e_{ij} = \begin{cases} \frac{1}{2m}, & \text{节点}i\text{和节点}j\text{有连边} \\ 0, & \text{其他} \end{cases} \qquad a_i = \frac{k_i}{2m}$$

2. 依次按照ΔQ的最大方向进行合并有边相连的社区对，并计算合并后的模块度增量ΔQ:

$$\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$$

3. 合并社区对以后修改对社区对称矩阵e 和社区i和j对应的行列。

4. 重复执行步骤(2)和(3)，不断合并社区，直至整个网络合并成一个社区为止。


0
5
10
15
20
25
30
35
12
11
5
1
7
6
17
20
22
18
2
14
3
10
13
4
8
15
28
24
34
30
27
29
32
26
25
33
31
9
16
23
21
19

时间复杂度：$O((m+n)*n)$

Newman快速算法

■ 优点：

算法复杂度对于稀疏网络约为$O(n^2)$，可用于规模较大的复杂网络。

■ 缺点：

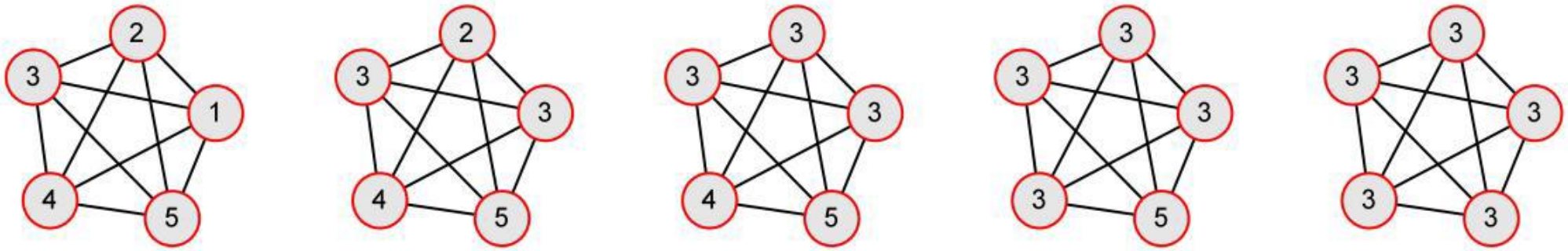
总是先合并大社团，即节点数目多的社团，会使Q值无法达到最高。

标签传播算法(LPA)

- 由拉加万（Usha Nandini Raghavan）等人提出
- 基于局部相似性的方法
- 用于不重叠社区发现

- 基本思想：将一个节点的邻居节点的标签中数量最多的标签作为该节点自身的标签。给每个节点添加标签以代表它所属的社区，并通过标签的传播形成同一标签的社区结构。

传播过程



输入：网络 $G(V,E)$；聚类方法。
输出：$G(V,E)$ 的社区结构。
聚类过程：
初始化网络中所有节点的标签 $C_x(0)=x$；
/* $C_x(t)=a$ 表示节点 x 在第 t 次迭代后的标签为 a */
初始化 $t=1$；
迭代聚类划分社区
while 存在需要更新标签的节点
将节点随机排序放到 X 中；
对所有 X 中的节点，更新其标签值 $C_x(t)=f(C_{x_1}(t),\cdots,C_{x_m}(t),C_{x_{m+1}}(t-1),\cdots,C_{x_n}(t-1))$；
/* $C_{x_i}(t)$ 表示节点 x 的邻接点 i 在第 t 次迭代时的标签值，f 函数用来返回在邻居节点中出现频率最高的标签值 */
$t=t+1$；
end while

标签传播算法

■ 优点：

1.时间复杂度接近线性，收敛周期短，适合大规模复杂网络。

2.无需事先指定社团个数和大小。

■ 缺点：

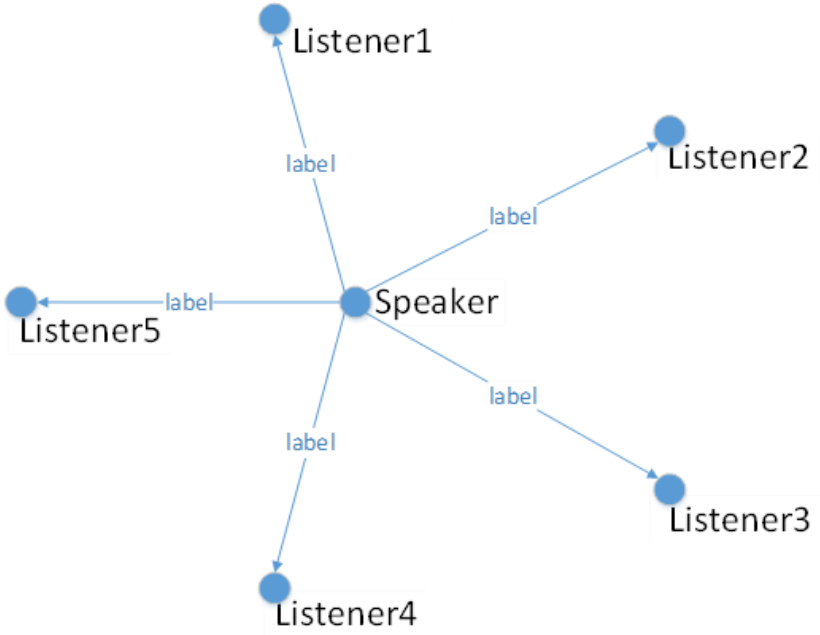
1.算法具有不确定性，多次运行结果不同。

2.可能产生怪兽(monster)社团。

SLPA(Speaker-listener Label Propagation Algorithm)

- Listener：标签传播时接收邻居节点标签的节点。

- Speaker：标签传播时Listener节点的所有邻接节点。



Algorithm 1 : SLPA(T, r)

```
[n,Nodes]=loadnetwork();
Stage 1: initialization
for i = 1 : n do
    Nodes(i).Mem=i;
Stage 2: evolution
for t = 1 : T do
    Nodes.ShuffleOrder();
    for i = 1 : n do
        Listener=Nodes(i);
        Speakers=Nodes(i).getNbs();
        for j = 1 : Speakers.len  do
            LabelList(j)= Speakers(j).speakerRule();
        w=Listener.listenerRule(LabelList);
        Listener.Mem.add(w);
Stage 3: post-processing
for i = 1 : n do
    remove Nodes(i) labels seen with probability < r;
```

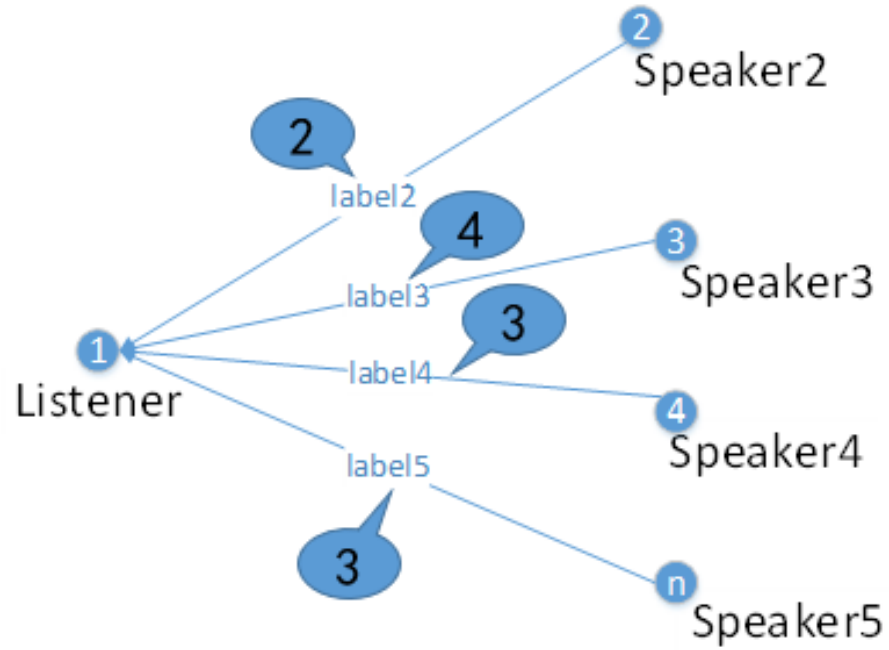
初始化每个节点

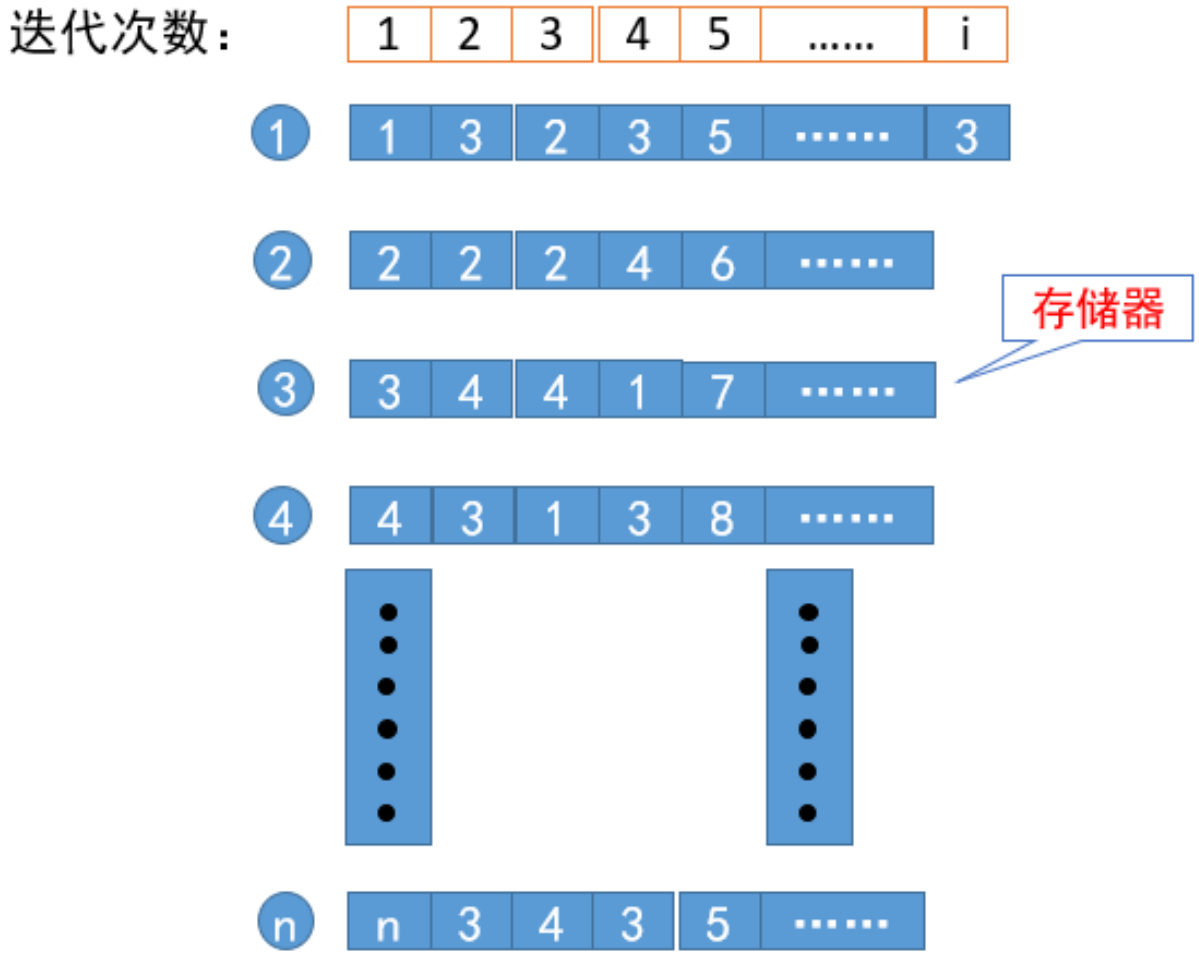
迭代T次

循环将一个节点作为Listener
将Listener的邻居节点作为Speakers

Speakers将自身最多的标签发送给Listener
Listener的标签从接收到的标签中按自定的规则选取

统计每一个节点的得到的所有标签数量
保存比例超过阈值r的标签


2
Speaker2
2
label2
4
3
Speaker3
label3
3
1
label4
Listener
4
Speaker4
label5
3
n
Speaker5


迭代次数：
1 2 3 4 5 i
1 1 3 2 3 5 3
2 2 2 2 4 6 .....
存储器
3 3 4 4 1 7 .....
4 4 3 1 3 8 .....
n n 3 4 3 5 .....

SLPA算法优缺点

■ 优点：

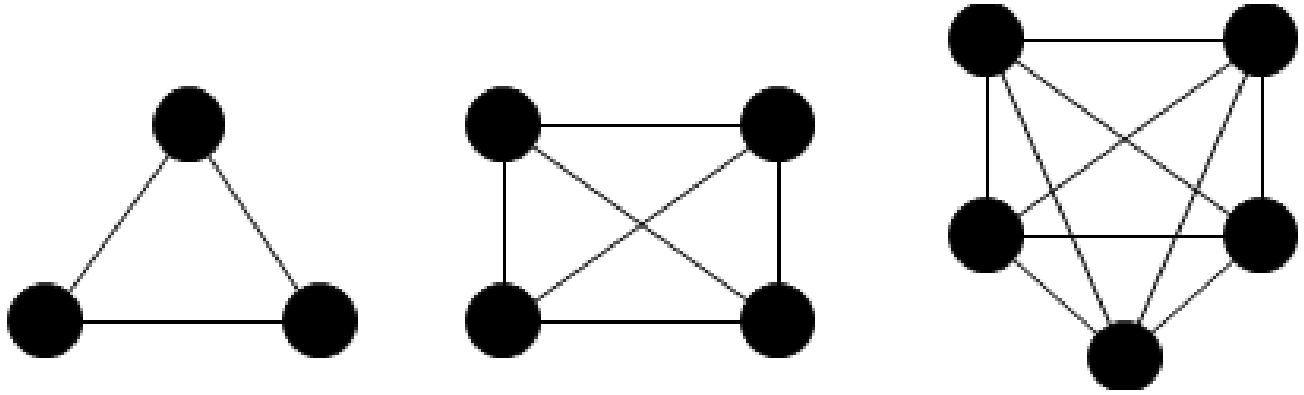
1. 能够检测重叠社团
2. 算法结果较LPA更加稳定
3. 具有线性的时间复杂度
4. 算法过程可以很容易地进行修改，以适应不同的规则。

■ 缺点：

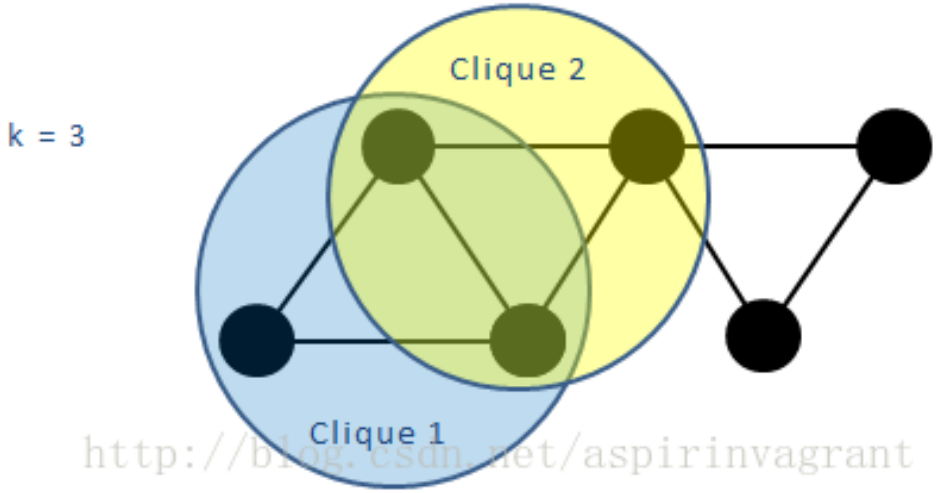
1. 无法有效避免怪兽社团产生，发现的社区的节点数量不太均衡。

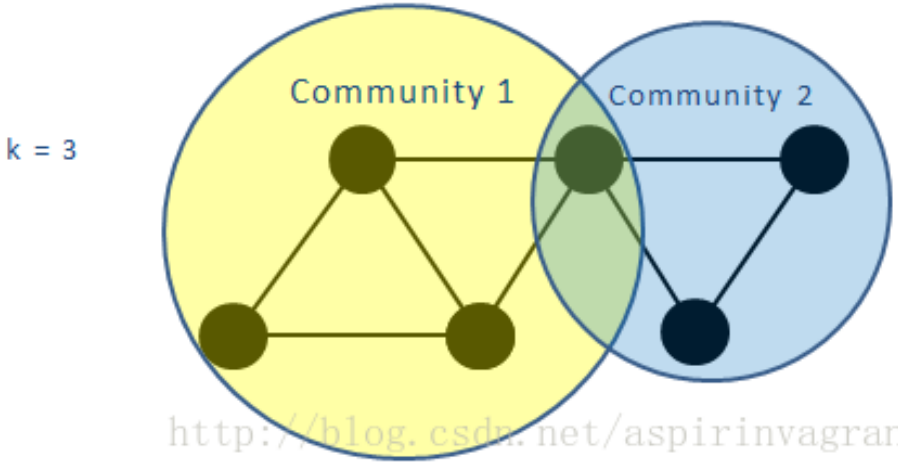
派系过滤算法(CPM)

- 用于发现重叠社区。
- 派系：任意两点都相连的顶点的集合，即完全子图。



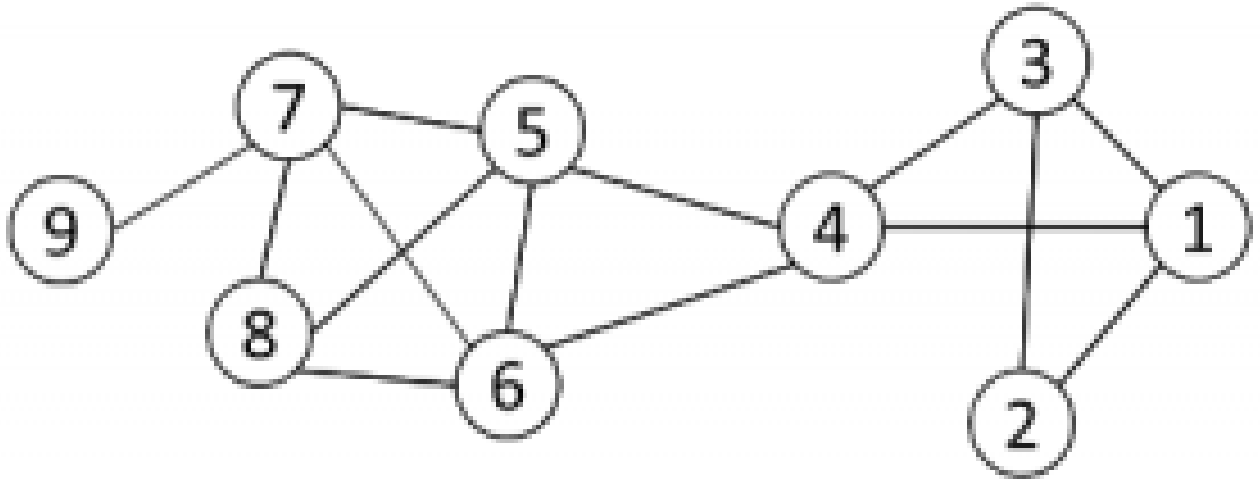
3-clique　　4-clique　　5-clique





k-派系相邻：两个不同的k-派系共享k-1个节点，认为他们相邻。

k-派系连通：一个k-派系可以通过若干个相邻的k-派系到达另一个k-派系，则称这两个k-派系彼此联通。



- 第一步： 找到网络中大小为k的完全子图。图中k=3的完全子图有
  $\{1,2,3\},\{1,3,4\},\{4,5,6\},\{5,6,7\},\{5,6,8\},\{5,7,8\},\{6,7,8\}$

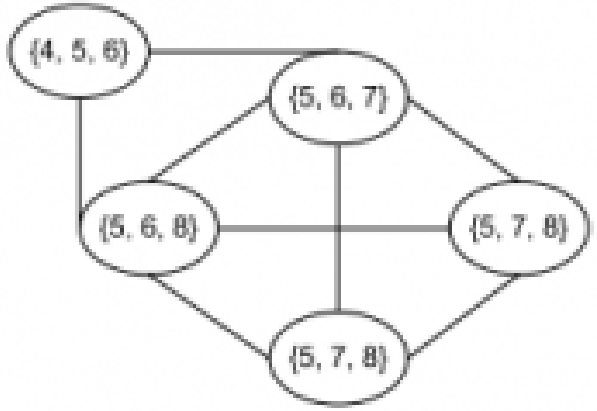
- 第二步：将完全子图定义为节点，构建一个重叠矩阵。
- 在这个对称的矩阵中，每一行（列）代表了一个派系，矩阵中的非对角线元素代表两个连通派系中共享的结点的数目。对角线元素代表派系的规模。将小于k-1的非对角线元素置为0，小于k的对角线元素置为1，得到k-派系连接矩阵，每个连通部分构成了一个k-派系社区。

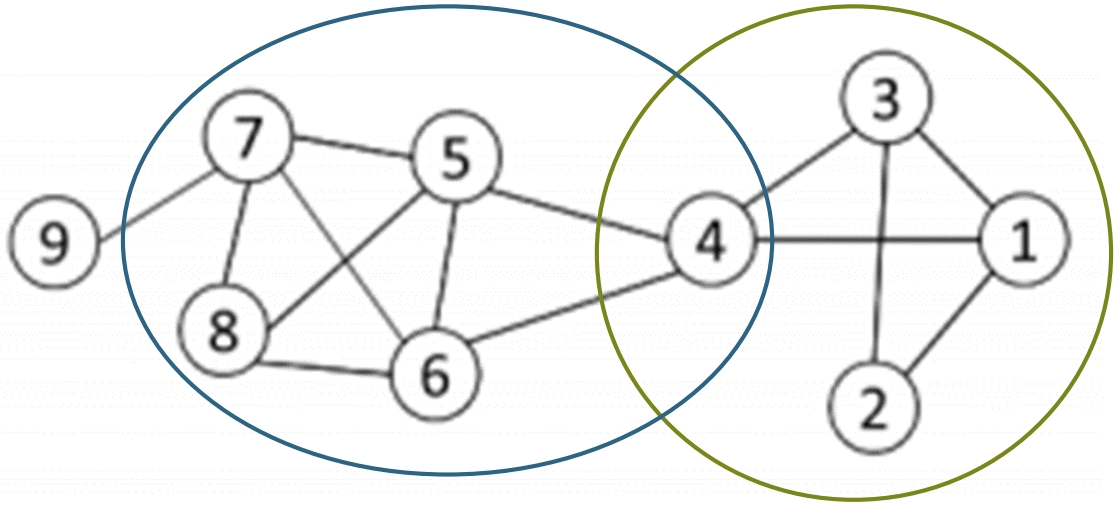
$$a = \begin{bmatrix} 3 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 2 & 2 & 1 & 1 \\ 0 & 0 & 2 & 3 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 & 3 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 3 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 & 3 \end{bmatrix}$$

- 第三步：将重叠矩阵变成社团邻接矩阵，其中重叠矩阵中对角线小于k，非对角线小于k-1的元素全置为0。

$$\begin{bmatrix} 3 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 2 & 2 & 1 & 1 \\ 0 & 0 & 2 & 3 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 & 3 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 3 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 & 3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

■ 第四步：根据社团邻接矩阵画出派系图，相连的派系看作一个社团。





派系过滤算法特点

- 需要输入参数值k，即寻找的最小派系范围。k取值越小，则能够发现的社团越大，社团结构越稀疏。一般k值为4-6。
- CPM算法基于完全子图，因此适用于完全子图较多的网络，即边密集的网络。
- 可以发现重叠社团。
- 无法分配完全子图以外的节点。

基于适应度的重叠社区发现算法

适应度函数:

$$f_G = \frac{k_{in}^G}{(k_{in}^G + k_{out}^G)^\alpha}$$

G：划分的一个社区

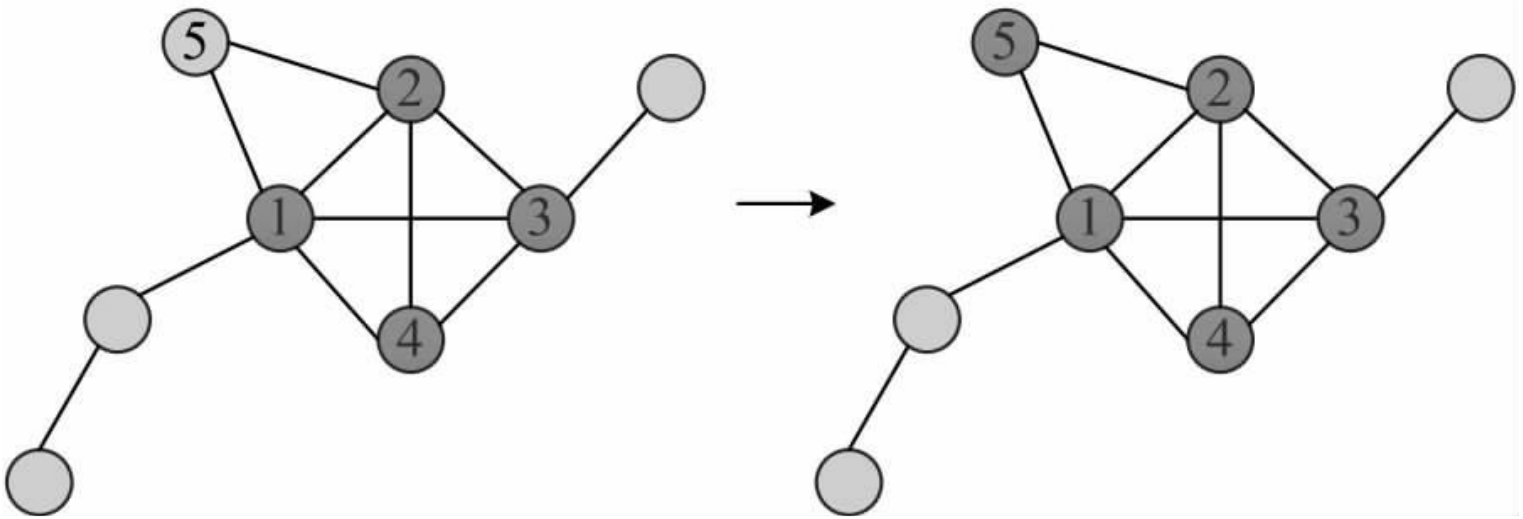
k_{in}^G：社团G内部边权重总和

k_{out}^G：社团G外部部边权重总和

α：一个控制社区规模的参数

■ 对于网络中的任一节点A，对社区G的适应度为：

$$f_G^A = f_{G+\{A\}} - f_{G-\{A\}}$$



$$f_G^5 = \frac{8}{(8+2)} - \frac{6}{(6+4)} > 0$$

自然社区形成

1. 对子图G的所有邻接点循环操作，计算每个邻接点对G的适应度函数值；

2. 将适应度值最大的点加入子图G形成更大的子图G' ；

3.重新计算子图G'中的所有点的适应度值；

4. 如果其中一个节点的适应度值变为负数，则将该节点从子图G'中剔除，形成新的子图G"；

5. 如果第四步发生，则进行第三步；否则，以子图G“ 从第一步开始重复，知道社区所有邻接点对子图的适应度都为负数为止。

适应度算法

1. 随机选取一个节点A；
2. 探测获取包含节点A的自然社区；
3. 随机选取一个不属于任何社区的节点B；
4. 探测节点B的自然社区，无论其周围节点是否属于其他社区；
5. 重复开始第三步，知道所有节点都被分配到至少一个社区。

适应度算法优缺点

■ 优点：

1.可以快速发现社团，能够用于局部社团发现。

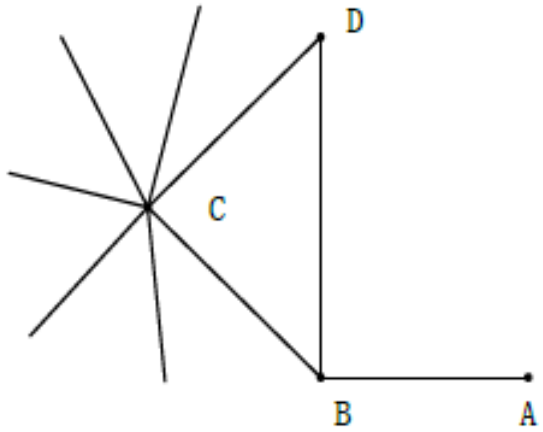
2.能够发现重叠社团。

3.通过设置参数，可以发现不同层次不同规模的社团。

■ 缺点：

1.不可靠，从不同的点开始可能导致不同的结果。

2.可能出现节点反复进入自然社区的死循环。



代码实践

- GN算法：https://github.com/sikasjc/CommunityDetection
- FN算法：https://download.csdn.net/download/qq993298526/8171639
- CPM算法：https://download.csdn.net/download/yujianhengxing/2431529
- LPA算法：https://blog.csdn.net/nwpuwyk/article/details/47426909
- SLPA/GANXiS算法： https://download.csdn.net/download/u014472643/10456631
- 适应度算法：https://github.com/zzz24512653/CommunityDetection/blob/master/algorithm/LFM.py

其他社团发现算法

■ 非重叠社团发现：

图分割方法：KL算法

谱平分方法

基于模块度优化算法：CNM算法、Louvain算法

■ 重叠社团发现：

COPRAE算法(标签传播)

谱聚类算法

非负矩阵分解方法

谢 谢