政府新闻社交网络分析（基于百度 LAC 与 NetworkX 库）

崔冠宇[1)]

[1)](中国人民大学 信息学院, 北京 中国 100872)

摘 要 社交网络分析融合了信息学、数学和社会学等多领域的知识，也一直是这些学科领域中的重要问题之一。随着计算机性能的不断提升以及 Python 等数据科学语言的出现，大规模网络的分析难度逐渐降低。本文以从 www.gov.cn 上爬取的政府新闻作为数据集，分别测试了 jieba、pkuseg、thulac 以及 lac(baidu) 库的分词与词性标注效果，综合选择效果较好的库，提取新闻中的实体和人物，统计其中的热门实体，并利用 NetworkX 库建立社交网络图。建立好社交网络后，在基础内容方面，本文进行了图的验证与统计以及 PageRank 影响力的计算；在自选分析部分，本文对社区检测、中心性计算和节点的聚集系数进行了实现与分析。

关键词 社交网络分析；命名实体识别；PageRank；社区检测；中心性；聚集系数

中图法分类号 TP391 DOI 号 10.11897/SP.J.1016.01.2020.00001

Social Network Analysis From Government News (Based on Baidu LAC and NetworkX)

Guanyu Cui[1)]

[1)](Schoool of Information, Renmin University of China, 100872, P.R.C)

Abstract Social network analysis integrates knowledges from multiple fields such as informatics, mathematics, sociology, etc. and it has been one of the most important issues in these disciplines. With the continuous improvements of performance of computers and the emergence of data science programming languages such as Python, the difficulty of analyzing large-scale networks is gradually decreasing. This article uses government news dataset crawled from www.gov.cn to test the word segmentation and part-of-speech tagging effects of jieba, pkuseg, thulac, and lac(baidu) libraries, select the library with best effect, extract the entities and figures in the news, count the popular entities among them, and use the NetworkX library to build a social network graph. After building the social network, in basic analysis part, this article has carried out the verification and statistics of the graph and the calculation of the PageRank of the nodes; In the free-selecting analysis part, this article implements and analyzes the community detection, centrality calculation and the clustering coefficient of the nodes of the graph.

Key words Social Network Analysis; Named Entity Recognation; PageRank; Community Detection; Centrality; Clustering Coefficient

1 引言

社交网络分析跨多学科、多领域，是社会学等学科中十分重要的问题，对于理解人类社会的发展和变化具有重要意义。

本文选取政府新闻数据集，试图提取每条新闻中的人物，并根据人物之间的共现关系构建社交网络图。在构建完社交网络之后，本文对社交网络的基本特征、节点影响力以及社区等结构进行识别和分析，从观察到的结果中获取启示和知识。

同时，本项目的代码也具有一定的通用性，可以作为文本与网络分析的框架，将其开源供其他研究者使用。

收稿日期：2020-12-23；修改日期：2020-12-23 崔冠宇，男，2000 年生，本科生，主要研究领域为集合论、计算理论和计算复杂度. E-mail: cuiguanyu@ruc.edu.cn.

2 研究过程

本文主要分三部分对政府新闻进行分析：

- 数据预处理部分，主要是对海量新闻数据进行分词和实体识别，从而提取出新闻中所含有的

关键人物和机构；

- 基础内容分析部分，主要是统计图的基本特征，以及计算各节点的 PageRank 影响力；
- 自选分析部分，主要是用 Louvain 算法和其它多种算法识别和检测网络中的社区，用近似算法计算各节点的中介中心性，以及计算各节点的聚集系数。

2.1 数据预处理

2.1.1 分词与实体提取

分词与命名实体识别一直是中文文本分析中的巨大挑战。

目前中文分词和词性标注主要有两种实现类别：其一是基于词典和规则匹配的分词，主要有正向最大匹配 (MM)、逆向最大匹配 (RMM) 等方法；其二是基于统计的分词，主要有 N 元文法模型 (N-gram)、隐马尔可夫模型 (HMM) 和条件随机场模型 (CRF) 等方法。

在大数据时代，基于统计的分词占据主流地位，jieba[1]、pkuseg[2]、thulac[3] 和百度的 lac[4] 等 Python 开源包不断出现。在政府新闻的分词与实体提取阶段，我分别实验了上述四种库的分词与词性标注的效果，定性结果如表 1 所示：

表 1 四种中文分词和词性识别库的效果

名称	运行速度[5]	结果准确度[6]
jieba	快 (~ 12 min)	差
pkuseg	较快 (~ 20 min)	一般
thulac	极慢 (~ 7 h)	较好
lac(baidu)	稍慢 (~ 35 min)	好

综合上述实验效果，选择效果较好、运行时间较快的 baidu 的 lac 库进行分词与词性识别。

政府新闻文本文件的分词与实体提取过程如过程 1 所示。

2.1.2 热门实体的统计

由于在分词与实体提取阶段得到了实体-词频字典，因此热门实体统计的方法十分简单，只需要将字典按词频降序排序，然后取前 k 个实体即可。

[1] https://github.com/fxsjy/jieba
[2] https://github.com/lancopku/pkuseg-python
[3] https://github.com/thunlp/THULAC-Python
[4] https://github.com/baidu/lac
[5] 由于首次运行没有记录具体运行时间，部分库的运行时间过长，难以再次运行，因此仅有大约的运行时间。
[6] 由于没有准确标签，因此只能由肉眼观察得到大致分词效果。

过程 1 新闻文本分词与实体提取 `fetch_entities`

输入： 新闻文本文件
输出： 实体-词频字典 `entity_freq`（类型：`dict`，`{str : int}`），新闻编号-人物集合字典 `set_news_people`（类型：`dict`，`{int: set()}`）
打开新闻文本文件
FOR 每一行文件 **DO**
  按制表符分割为若干数据域
  将新闻文本插入列表
END FOR
关闭新闻文件
FOR 新闻列表里的每一条新闻 **DO**
  令 `people` 为一空集合
  分词得到词语和标签列表
  FOR 列表里的每组单词和标签 **DO**
    去掉词语中的标点符号、多余空格等
    IF 标签是人物/机构 **THEN**
      修改实体-词频字典，增加词频
      IF 标签是人物 **THEN**
        将人物加到 `people` 中
      END IF
    END IF
  END FOR
  将 `people` 增加到新闻编号-人物字典中
END FOR
保存处理结果，便于复用
RETURN `entity_freq` 和 `set_news_people`

在政府新闻数据上实验得到的结果如图 1 所示。

```
(2) 热门人物/机构：
出现频率最高的 10 个实体为：
+----+----------+--------+
|    | 实体     | 词频   |
|----+----------+--------|
|  0 | 中国     |  73933 |
|  1 | 新华社   |  48199 |
|  2 | 习近平   |  25009 |
|  3 | 李克强   |  16672 |
|  4 | 北京     |  15578 |
|  5 | 国务院   |  13000 |
|  6 | 一带一路 |   7487 |
|  7 | 上海     |   4655 |
|  8 | 党中央   |   4028 |
|  9 | 王毅     |   3790 |
+----+----------+--------+
```

图 1 词频最高的 k 个热门实体 ($k = 10$)

2.1.3 建立社交网络

关于图的存储，我选择了 NetworkX 这一广受欢迎的 Python 图操作库。

由于要求社交网络图上两人之间的边权为二者的共现新闻条目数，因此需要对分词与实体提

取阶段提取到的新闻序号-人物集合字典进行处理，得到边（以节点对表示）以及边上的权重。整个构建过程如过程 2 所示：

过程 2 社交网络构建 `build_social_network`

输入：新闻编号-人物集合字典（类型：`dict`，`{int: set()}`）
输出：人物社交网络图（类型：`NetworkX Graph`）
令 `edges_dict` 为一空字典（类型：`dict`，`{(str, str):int}`）
FOR 每一条新闻编号对应的人物集合 **DO**
　FOR 集合中的每一对人物 p_1、p_2 **DO**
　　在 `edges_dict` 中增加 (p_1, p_2) 的共现次数
　END FOR
END FOR
利用 NetworkX 库的 `add_edges_from` 函数，从 `edges_dict` 构建社交网络图 `G`
RETURN `G`

2.1.4　社交网络的可视化

得到社交网络后，我研究了如何将大规模的网络可视化。

大规模网络可视化的主要问题是如何合理安排节点的显示位置。当前几种主流的节点分布算法有：ForceAtlas / ForceAtlas 2、OpenOrd 以及 YiFan Hu 算法。它们的主要特点如下：

1. ForceAtlas / ForceAtlas 2 是一类力导向的节点分布算法[1]，这类算法为节点和节点之间的边设置一组吸引力和排斥力，然后模拟物理世界的运行规律，让整个网络在力的作用下趋于稳定。

2. OpenOrd 算法[2] 则是为了克服力导向算法由于复杂度较高（$O(n^2)$）在大规模图上应用困难的问题而设计的算法。它大大降低了算法的复杂度，而且使节点分布过程可并行化。

3. YiFan Hu 算法[3] 是由 YiFan Hu 提出的多级节点分布算法，它在力导向算法的基础上进行综合改进，吸收了多种节点分布算法的优点，在实际的图数据上进行了测试，效果较好。

经过在 Gelphi 软件上的实际实验，我综合选择了速度较快，效果较好的 ForceAtlas 2 算法作为社交网络可视化的节点分布算法。

政府新闻数据构建出的社交网络的可视化结果如图 2 所示。

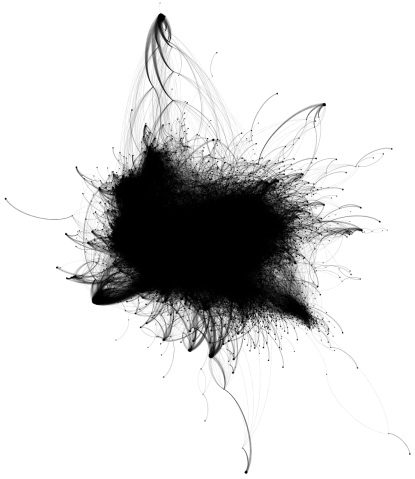


图 2　社交网络可视化结果

2.2　基础内容分析

2.2.1　图的验证

图的验证部分要求查询给定节点的关系最强的 k 个邻居，因此只需要找到该节点的所有邻居形成的列表，将其按边权降序排列，最后取前 k 个邻居返回即可。

在政府新闻社交网络上的实验结果如图 3 所示。

(1) 图的验证：
节点 习近平 的邻居：

	邻居	权重
0	汪洋	290
1	王勇	251
2	王晔	237
3	王毅	228
4	韩正	209
5	王沪宁	204
6	刘鹤	201
7	习主席	163
8	赵乐际	142
9	何立峰	129

图 3　给定节点关系最强的 k 个邻居
(示例节点选择“习近平”，$k = 10$)

2.2.2　图的统计

图的统计部分主要是计算图的几个统计指标，如节点数、边数、连通分量数以及最大连通分量的大小。NetworkX 的图是以邻接表的形式存储的，下面描述邻接表形式存储的图上这四种统计指标的计算方法：

1. 节点数：图的节点数是图的邻接表中节点列表

的长度。

2. 边数：对于有向图，图的边数是各节点的邻居列表的长度之和；对于无向图，则是各节点的邻居列表的长度之和除以二；

3. 连通分量数：对于无向图，一般通过 DFS 或是 BFS 遍历的方式获得各连通分量包含的节点，具体如算法 1 所示。

算法 1 DFS 获取连通分量

```
输入：  无向图 G
输出：  图 G 的各连通分量
  令 visited 为一空集合
  令 s 为一空栈
  令 comps 为一空集合
  WHILE |visited| < |V(G)| DO
    v ← V(G) − visited 中随机一节点
    s.push(v)
    令 c 为一空集合
    WHILE s 不空 DO
      n ← s.top()
      s.pop()
      visited ← visited ∪ {n}
      c ← c ∪ {n}
      FOR n 的所有邻居节点 m DO
        IF m 不在 visited 中 THEN
          s.push(m)
        END IF
      END FOR
    END WHILE
    comps ← comps ∪ {c}
  END WHILE
  RETURN comps
```

运行算法后，*comps* 集合中保存的即为各连通分量的节点集合，图 G 的连通分量个数即为集合 *comps* 的元素个数。

另外，用 BFS 算法也可以得到相同的结果。只是需要将 DFS 算法中的栈改为队列，压栈操作改为入队操作，获取栈顶元素操作改为获取队首元素操作，弹栈操作改为出队操作即可。

4. 最大连通分量的大小：对上一步得到的连通分量集合中的每一个连通分量求节点数，并取它们的最大值即可。

NetworkX 库实现了上述四种统计指标的前三种，对于最大连通分量的节点个数，则可以利用第三步得到的连通分量集合求得。

政府新闻社交网络图的统计结果如表 2 所示。

表 2 政府新闻社交网络的若干统计特征

节点数	边数	连通分量数	最大连通分量节点数
22571	268226	117	22228

2.2.3 影响力计算

为了衡量一个网页的重要程度，Page 等人于 1999 年提出了 PageRank 的概念[4]。

在图 G 中，某节点 v 的原始 PageRank 值 $PR(v)$ 定义为

$$PR(v) = \sum_{u \to v} \frac{PR(u)}{OutDeg(u)} \tag{1}$$

即每个节点会将自身的 PageRank 值按出度平均传递给所有出边指向的节点。当上述过程达到平衡时，各个节点的 PageRank 值将不再发生变化。一个简单的网络如图 4 所示。

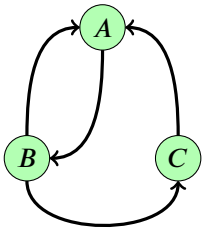


图 4 小型网络上的 PageRank 示例 1

从各节点的 PageRank 值均为 1/3 开始迭代，结果如表 3 所示。

表 3 图 4 所示网络的 PageRank 值迭代结果

	A	B	C
初始	1/3	1/3	1/3
结果	2/5	2/5	1/5

但是当图中有一些“自私”的、没有出边的顶点时，原始 PageRank 的表现不佳，不符合我们的直觉。

比如删除上图中节点 C 到节点 A 的边，得到图 5。

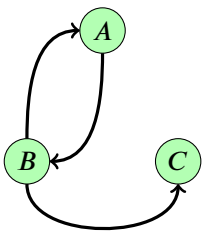


图 5 小型网络上的 PageRank 示例 2

此时，再次从各节点的 PageRank 值均为 1/3 开

始迭代，结果如表 4 所示。

表 4　图 5 所示网络的 PageRank 值迭代结果

	A	B	C
初始	1/3	1/3	1/3
结果	0	0	0

从上面的结果可以看出，那些“自私”的节点就像一个个黑洞，贪婪地吸收着周围节点的 PageRank 值，并且不再释放它们。

为了消除这些“黑洞节点”的影响，Page 等人在[4] 中又提出了随机冲浪模型下的 PageRank，它的定义式为

$$PR(v) = \frac{1-\alpha}{n} + \alpha \sum_{u \to v} \frac{PR(u)}{OutDeg(u)} \tag{2}$$

其中 n 是图 G 中节点总数；α 是一个 $[0,1]$ 间的常数，表示按原始方式均分给出边指向节点的比例。一般将 α 设置为 0.85 左右，这样既保证了网络结构对 PageRank 的主导作用，又在一定程度上抑制了“黑洞节点”对 PageRank 的影响。

PageRank 根据网络中节点之间的链接关系，较为成功地定量化了节点的重要程度。对于政府新闻数据构建的社交网络，选用节点的 PageRank 作为重要度的衡量，利用 NetworkX 库提供的 `pagerank` 函数，计算出各节点的 PageRank 值，然后取重要性最高的前 k 个节点。实验结果如图 6 所示。

(3) 影响力计算：

	节点	影响力
0	习近平	0.0193907
1	李克强	0.00890095
2	王毅	0.00552646
3	何立峰	0.0025932
4	杨洁篪	0.00257405
5	丁薛祥	0.00225653
6	韩正	0.00209851
7	汪洋	0.00209689
8	王沪宁	0.00191154
9	刘鹤	0.00170081
10	王晔	0.00167909
11	鞠鹏	0.00159992
12	王勇	0.0015968
13	孙春兰	0.00152717
14	栗战书	0.00150208
15	习主席	0.00150189
16	肖捷	0.00148292
17	赵乐际	0.00127501
18	胡春华	0.00125153
19	郭声琨	0.00122214

图 6　PageRank 影响力最高的 k 个节点 ($k = 20$)

2.3　自选分析

2.3.1　社区挖掘

网络中的社区结构是指网络中的部分节点，这些节点之间的联系相对比较紧密，但它们与外部的联系相对稀疏。图的社区挖掘，或称社区检测和社区发现，是指在给定的图中，找到一组较好的社区划分。

为了衡量一组划分的好坏，Newman 等人在[5] 中提出了“模块度”(modularity) 的概念。(无向) 图 G 上一组划分的模块度 Q 定义为：

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{3}$$

其中 A_{ij} 表示节点 i 到节点 j 的边权；$k_i = \sum_j A_{ij}$ 表示以 i 为起点的出边的权重和；c_i 表示节点 i 所属的社区编号；$\delta(x,y)$ 是等同函数，当且仅当 $x = y$ 时 $\delta(x,y) = 1$，否则 $\delta(x,y) = 0$；$m = \frac{1}{2}\sum_{i,j} A_{ij}$ 是图 G 的边权之和。

基于模块度的社区发现算法的目标是找到一种社区划分策略，使得在这种划分之下的模块度尽可能高。但是由于 n 个顶点的图的划分共有第 n 个贝尔数 B_n 种，根据 Berend & Tassa 2010 年的结果[6]，

$$B_n < \left(\frac{0.792n}{\ln(n+1)} \right)^n,$$

即 B_n 是以指数级增长的，因而获得模块度最大的社区划分也是一个极为困难的问题。又根据 Brandes 等人 2007 年的成果[7]，找寻最大化模块度的划分是 NP 困难问题，因此只能应用贪心算法来获得近似解。

Clauset 等人在[8] 中给出了一种基于模块度增益的贪心社区划分算法；Blondel 等人提出的层次化社区发现算法（现在被称为 Louvain 算法）[9] 是对普通贪心算法的改进，将第一次划分得到的社区抽象成超级节点，重复多次进行社区划分，直到不再有模块度增益为止。

除上述两种算法之外，Raghavan 等人在[10] 中提出了名叫“异步标记传播”(asynchronous label propagation) 的算法，它能以接近线性时间检测图中的社区。

NetworkX 库实现了 Clauset 等人提出的贪心社区划分算法 `greedy_modularity_communities` 以及异步标记传播算法 `asyn_lpa_communities`；python-

louvain 库[5]实现了 Louvain 算法。

根据我的实验，普通的贪心社区划分算法 `greedy_modularity_communities` 由于时间复杂度较高，不能在较短时间内计算出结果。其他几种方法的结果如图 7、8 和 9 所示：

```
(3) 社区挖掘：
在 louvain 社区发现模式下，共划分出 169 个社区
模块度为：0.5834180233132118

前 5 个社区为：

社区 1：
大小：6677
代表元：曾金华

社区 2：
大小：4325
代表元：习近平

社区 3：
大小：3109
代表元：徐昱

社区 4：
大小：1580
代表元：李刚

社区 5：
大小：1065
代表元：吴刚
```

图 7 Louvain 算法得到的前 5 个社区

```
在 asyn_lpa 社区发现模式下，共划分出 1275 个社区
模块度为：0.2371932795034038

前 5 个社区为：

社区 1：
大小：13561
代表元：习近平

社区 2：
大小：1323
代表元：吴刚

社区 3：
大小：618
代表元：刘满仓

社区 4：
大小：122
代表元：李安摄

社区 5：
大小：102
代表元：巴赫
```

图 8 异步标记传播算法得到的前 5 个社区

```
在 lpa 社区发现模式下，共划分出 1187 个社区
模块度为：0.10631262993841155

前 5 个社区为：

社区 1：
大小：16002
代表元：习近平

社区 2：
大小：527
代表元：刘满仓

社区 3：
大小：75
代表元：陈某

社区 4：
大小：66
代表元：蒋华栋

社区 5：
大小：66
代表元：周斌
```

图 9 标记传播算法得到的前 5 个社区

[5]https://github.com/taynaud/python-louvain

2.3.2 中心性计算

中心性 (centrality) 是图中的节点或边在图的整体结构上的重要性指标。较为基础与常见的中心性指标有：度中心性 (degree centrality)、接近中心性 (closeness centrality)、中介中心性 (betweeness centrality) 以及特征向量中心性 (eigenvector centrality) 等。

下面着重介绍节点的中介中心性概念。节点的中介中心性，是用图中各个节点对之间的最短路径中通过该节点的路径的比例之和来反映节点的重要程度（类似地，还有边的中介中心性）。它的计算方法如公式 4 所示：

$$c_B(v) = \sum_{s,t \in V(G)} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{4}$$

其中 $V(G)$ 是图 G 的顶点集，$\sigma(s,t)$ 表示节点 s 与节点 t 之间的最短路径的条数，而 $\sigma(s,t|v)$ 则表示上述路径中经过 t 的条数。当 $s=t$ 时，认为 $\sigma(s,t)=1$；当 $v=s$ 或 $v=t$ 时，认为 $\sigma(s,t|v)=0$。

NetworkX 库实现了节点中介中心性函数 `betweeness_centrality` 以及边中介中心性函数 `edge_betweeness_centrality`。

经过我的实验，节点中心性函数在所有节点上运行时由于复杂度较高无法得到结果，因此我使用了采样节点数这一参数，即按一定的采样率随机选择部分节点，在这些节点上计算各节点的中介中心性。实验结果如图 10 所示。

(4) （中介）中心性计算：

	节点	中心性
0	习近平	0.455978
1	李克强	0.123842
2	王毅	0.0231852
3	牟宇	0.00962656
4	李刚	0.00861066
5	何立峰	0.0084709
6	陶明	0.0084694
7	徐昱	0.00812207
8	赵萍	0.00785266
9	胡浩	0.0075667

图 10 中介中心性最高的 k 个节点 ($k=10$，采样率 `sample_ratio` $=0.02$)

2.3.3 节点的聚集系数计算

节点的聚集系数 (clustering coefficient) 反映了与之相邻的顶点之间互相连接的程度。某节点的聚集系数定义为它的邻居节点相互之间存在的边数

除以邻居节点之间最多可能的边数，即

$$cc(v) = \frac{\sum_{s,t\in N(v),(s,t)\in E(G)} 1}{\frac{|N(v)|(|N(v)-1|)}{2}} = \frac{2\sum_{s,t\in N(v),(s,t)\in E(G)} 1}{|N(v)|(|N(v)|-1)} \tag{5}$$

计算一个节点的聚集系数的方法如算法 2 所示。

算法 2 计算节点的聚集系数

输入：　图 G，节点 v
输出：　v 的聚集系数
　令 $N(v)$ 为 v 的邻居集合
　$count \leftarrow 0$
　FOR $N(v)$ 中的任意两顶点 s 和 t **DO**
　　IF $(s,t) \in E(G)$ **THEN**
　　$count \leftarrow count + 1$
　　END IF
　END FOR
　RETURN $2 \times count/(|N(v)| \times (|N(v)|-1))$

在政府新闻社交网络图上运行的结果如图 11 所示。

```
(5) 节点的聚集系数计算：
+----+--------+------------+
|    | 节点   |   聚集系数 |
+----+--------+------------+
|  0 | 陈星杰 |          1 |
|  1 | 龚静毅 |          1 |
|  2 | 徐仰辉 |          1 |
|  3 | 张秀清 |          1 |
|  4 | 姬兆亮 |          1 |
|  5 | 杜旭亮 |          1 |
|  6 | 蔡宇   |          1 |
|  7 | 彭林鹏 |          1 |
|  8 | 郑襄段 |          1 |
|  9 | 熙楠   |          1 |
+----+--------+------------+
```

图 11　聚集系数最高的 k 个节点 ($k = 10$)

3　结果分析与论述

3.1　结果分析

3.1.1　数据预处理部分

数据预处理部分主要是对文本进行预处理，然后建立社交网络图。

四种中文文本分析库的实体提取输出文件及社交网络邻接表输出文件可以在 `./data` 目录下找到。

简单浏览四种库的输出文件，可以看出 jieba 库的效果最差，把一些动词、形容词等错分为人物和机构；pkuseg 和 thulac 库的效果相较于 jieba 有了一定的提升，但效果还是令我不太满意，而且 thulac 库分词速度太慢，难以在大规模数据上使用；baidu 的 LAC 库的分词速度相对可以接受，效果也有了提升，因而我选择 baidu 的库进行分词。

观察热门实体提取部分，可以看到词频最高的实体是“中国”，符合我的预想；词频第二高的实体是“新华社”，这可能是由于新闻正文开头经常出现“新华社电”等字样造成的；接下来的两个人物分别是“习近平”和“李克强”，因为二者是中国的重要政治人物，因此词频高是可以预知的；后面的高频词也是符合我们日常经验的。

在社交网络构建与可视化部分，我自学了 Gephi 软件的使用。执行分布算法后得到图 2，可以很明显地看出，图中有部分节点结合很紧密，甚至还出现了很密集的“核心”，而其它大部分节点的连接相对稀疏。

我还尝试了 Gephi 的度分布统计、聚集系数分布统计等功能，结果分别如图 12、图 13 所示。从

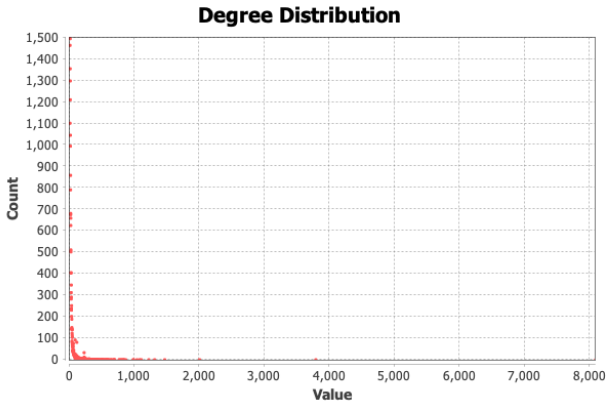


图 12　节点的度分布（平均值：23.767）

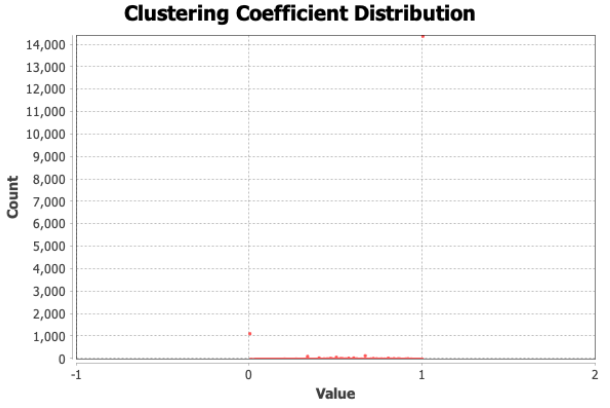


图 13　节点的聚集系数分布（平均值：0.833）

图的度统计数据来看，节点的度十分符合“二八定律”或是幂律分布，即绝大多数顶点的度都较小，仅有少数节点的度很大。

3.1.2 基础内容部分

基础内容部分主要对图的一些基本指标进行统计和分析。

在图的验证，即与某节点关系最强的 k 个邻居的部分中，我选择了节点“习近平”的邻居进行了观察。如图 3 所示，与习近平共现次数最多的人物多为我们熟悉的政治人物（如汪洋、王毅、王沪宁等），符合我们的经验和认知。

在图的统计部分，根据边数与节点数的值，可以使用公式 6 计算出该（无向）图的密度

$$\mathscr{D} = \frac{m}{\frac{n(n-1)}{2}} = \frac{2m}{n(n-1)} \tag{6}$$

其中 m 是图的边数，n 是图的顶点数。代入政府新闻社交网络图的数据 $m = 268226$，$n = 22571$，算得图密度 $\mathscr{D} = 0.105\%$，因此该图十分稀疏。

同时观察节点数、连通分量数与最大连通分量包含的节点数，可以发现图中存在一个超大的连通分量，占比超过总节点数的 98.48%，还可以推断出，其他较少的节点存在于许多小规模的连通分量中。

在节点的 PageRank 影响力部分，我计算了影响力最高的 20 个节点，结果如图 6 所示。可以看到，在政府新闻社交网络中，影响力最高的两人分别是习近平与李克强，PageRank 值分别为 0.0194 和 0.0090。他们分别代表党中央和国务院，可以称得上是中国政治领域最有影响力的人，因此实验结果符合我的预期。虽然二者都很有影响力，但是习近平主席的 PageRank 值为李克强总理的两倍以上，这个发现也比较新奇，有些出乎我的意料。

3.1.3 自选分析部分

自选分析部分主要是对图的深层结构进行计算和分析。

在社区检测部分，我尝试了三种社区发现算法，它们的结果如图 7、图 8 及图 9 所示。观察三种结果，我发现仅从模块度的角度来看，Louvain 算法得到的社区划分较好，并且各社区的大小相对较为均衡；而另两种算法的模块度都较低，而且划分出的社区的大小相差较大。

另外，我还尝试在 Gephi 软件中运行社区检测算法，并且筛选出原图中 PageRank 值最高的 96 个节点进行可视化，其结果如图 14 所示（大图可参见附录B）。

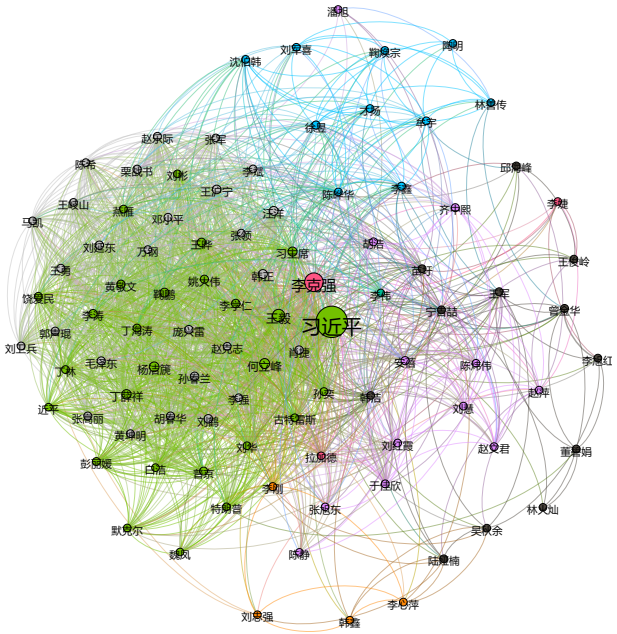


图 14 运行社区检测算法后，将原图中 PageRank 最高的 96 个节点可视化

在节点中心性计算部分，我使用了随机采样的近似算法，得到中介中心性最高的节点如图 10 所示。可以看到，习近平主席和李克强总理仍然位于前两位，而且前者的中介中心性达到了惊人的 0.456。这一现象说明在政府新闻社交网络中，任意两个人之间的最短路径中，有超过 45% 将会经过习近平主席，从这一现象可以看出习近平主席在中国政治生活中的重要性。

在节点聚集系数计算部分，也有很有意思的结果。在图 11 所示的计算结果中，竟然所有节点的聚集系数都为 1，而且这些节点并不是著名人物。结合之前关于图上连通分量的分析，我猜想这些节点可能处于众多小连通分量中，甚至有可能是一些单节点。

3.2 研究体会与收获

通过本次探究政府新闻数据的作业，我不仅实践了课上学习的图的分析指标和分析方法，还在课外自己学习了文本分析、词性标注以及图可视化技术，获得了新知识与新见解。

除此之外，这次作业更是锻炼了我的文献阅读能力以及论文撰写能力。在产出一篇研究论文的基础上，我还从头到尾亲自体验了 Python 模块的开发，并且为自己的代码撰写了用户手册，方便真实用户的理解和使用。

3.3　不足之处

当然我的研究也不是尽善尽美的，我觉得研究过程中还有下面的一些缺憾：

- 分词与实体识别效果仍没有达到最好，偶尔会有一些专有名词（如“郑襄高铁”）的一部分会被错误的当成人物，而且有些人物与其别名（如“习近平”与“习主席”）会被认为是不同的人物。后续研究时可以考虑如何识别合并相同人物的不同表达。

- 大规模网络的可视化效果不佳，不美观。后续研究时可以考虑使用更高级的图数据库，如 Neo4j 等。

- 节点的中介中心性部分由于网络规模过大，只能使用采样的近似算法。后续可以考虑研究其它降低计算复杂度的方法，或是研究节点或边的其它中心性特征。

- 节点的聚集系数计算效果不好，很难得到明显的结论。后续处理时可以考虑在图的超大连通分量上进行计算，避免单节点等特殊情况的干扰。

致　谢　（致谢内容略）

参考文献

[1] JACOMY M, VENTURINI T, HEYMANN S, et al. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software[J]. PloS one, 2014, 9(6):e98679.

[2] MARTIN S, BROWN W M, KLAVANS R, et al. Openord: an open-source toolbox for large graph layout[C]//Visualization and Data Analysis 2011: volume 7868. [S.l.]: International Society for Optics and Photonics, 2011: 786806.

[3] HU Y. Efficient, high-quality force-directed graph drawing[J]. Mathematica Journal, 2005, 10(1):37-71.

[4] PAGE L, BRIN S, MOTWANI R, et al. The pagerank citation ranking: Bringing order to the web.[R]. [S.l.]: Stanford InfoLab, 1999.

[5] NEWMAN M E. Analysis of weighted networks[J]. Physical review E, 2004, 70(5):056131.

[6] BEREND D, TASSA T. Improved bounds on bell numbers and on moments of sums of random variables[J]. Probability and Mathematical Statistics, 2010, 30(2):185-205.

[7] BRANDES U, DELLING D, GAERTLER M, et al. On modularity clustering[J]. IEEE transactions on knowledge and data engineering, 2007, 20(2):172-188.

[8] CLAUSET A, NEWMAN M E J, MOORE C. Finding community structure in very large networks[J/OL]. Physical Review E, 2004, 70 (6). http://dx.doi.org/10.1103/PhysRevE.70.066111. DOI: 10.1103/physreve.70.066111.

[9] BLONDEL V D, GUILLAUME J L, LAMBIOTTE R, et al. Fast unfolding of communities in large networks[J]. Journal of statistical mechanics: theory and experiment, 2008, 2008(10):P10008.

[10] RAGHAVAN U N, ALBERT R, KUMARA S. Near linear time algorithm to detect community structures in large-scale networks[J/OL]. Physical Review E, 2007, 76(3). http://dx.doi.org/10.1103/PhysRevE.76.036106. DOI: 10.1103/physreve.76.036106.

附录 A 程序代码

政府新闻分析——gov_news_analysis.py

```
import os
import re, string
# 分词
import jieba
import jieba.posseg as pseg
import thulac
import pkuseg
from LAC import LAC
# 网络分析
import networkx as nx
import networkx.algorithms.community as nx_comm
import community
# 表格相关
import pandas as pd
from tabulate import tabulate

def has_chinese(string):
    for ch in string:
        if u'\u4e00' <= ch <= u'\u9fff':
            return True
    return False

# 从新闻文件中获得实体信息
# @param news_file: 需要提取的新闻文件
# @param save_file: 保存文件
# @param mode: 分词提取使用模式 in (jieba, pkuseg, thulac)
# @return entity_freq, set_news_people: 实体-词频字典 {entity: freq}, 新闻编号-人物集合字典 {count: set{people}}
def fetch_entities(news_file, save_file = 'default_entity_file.txt', mode = 'jieba'):
    if not os.path.exists(news_file):
        print('打不开新闻文件! 退出中...')
        return None

    # 文本
    texts = []
    with open(news_file) as f:
        # 去掉标题行
        headline = f.readline()
        # 读入每一行
        for line in f:
            # 五个数据域
            url, date, meta, title, text = line.split(sep = '\t')
            texts.append(text)
    f.close()
    del f, headline, line, url, date, meta, title, text

    # 保留标签
    # nr    人名
    # nrfg  人名(jieba)
    # nrt   外国人名(jieba)
    # np    人名(thulac)
    # PER   人名(baidu)
    # ns    地名
    # LOC   地名(baidu)
    # nt    机构团体
    # ni    机构名(thulac)
    # ORG   机构名(baidu)
    # nz    其他专名
    tags = ['nr', 'nrfg', 'nrt', 'np', 'PER', 'ns', 'LOC', 'nt', 'ni', 'ORG', 'nz']

    # 各人/地实体出现次数
    entity_freq = {}
    # 每条新闻出现的人
    set_news_people = {}

    # 第几条新闻
    text_count = 0
    # 共几条新闻
    text_total = len(texts)

    # 不同模式的执行语句不同
    if mode == 'jieba':
        cut_func_str = 'pseg.cut(text)'
    elif mode == 'pkuseg':
        seg = pkuseg.pkuseg(postag = True)
        seg
        cut_func_str = 'seg.cut(text)'
    elif mode == 'thulac':
        thu1 = thulac.thulac()
        thu1
        cut_func_str = 'thu1.cut(text)'
    elif mode == 'baidu':
        lac = LAC(mode = 'lac')
        lac
        cut_func_str = 'lac.run(text)'

    for text in texts:
        print('正处理新闻', text_count, '/', text_total, '...')
        # 去掉特殊转义字符
        text = re.sub(r'[\t\r\n]', ' ', text)
        # 去掉多余空格
        text = re.sub(r'[ ]+', ' ', text)
        text = text.strip()
```

```
        # 人实体
        people = set()
        # 分词加标签
        words_tags = eval(cut_func_str)
        # baidu 模式需要单独处理一下
        if mode == 'baidu':
            words_tags = [ (words_tags[0][i], words_tags[1][i]) for i in range( len( words_tags[0]) ) ]
        # 对每个词
        for word, tag in words_tags:
            # 去掉空格和除·以外的标点符号
            word = re.sub(r'[~!@#$%^&*()_+{}|:"<>?`=-\[\]\\;,./～！ ￥…* （） —+ 「」： “” 《》 〈〉 ？ 【】 、 ； ‘’ ， 。 ]', '', word)
            word = word.replace(' ', '')
            # 如果是人/地 (去掉单字)
            if tag in tags and len(word) > 1:
                # 增加词频
                if word in entity_freq:
                    entity_freq[word] += 1
                else:
                    entity_freq[word] = 1
                # 如果是人物
                if tag in tags[:5] and has_chinese(word):
                    people.add(word)
        print(people)
        # 新闻序号 -> 人集合
        set_news_people[text_count] = people
        text_count += 1
    print('保存处理结果...')
    #保存
    f = open(save_file, 'w')

    f.write(str(entity_freq) + '\n')
    f.write(str(set_news_people) + '\n')
    f.close()
    print('新闻实体处理完成!')
    return entity_freq, set_news_people

# 统计出现频率最高的实体
# @param entity_freq: 实体-词频字典
# @param top_k: 前 k 个
# @param return_tabulate: 是否返回表格
# @return 词频前 k 高的实体的表格 或 dict
def most_frequently_entities_topk(entity_freq, top_k = 10, return_tabulate = True):
    # 按值降序
    entity_freq = sorted(entity_freq.items(), key = lambda kv: kv[1], reverse = True)
    # 前 top_k 个
    topk_entities_list = entity_freq[: top_k]
    topk_entities_dict = {k : v for k, v in topk_entities_list}
    # 构建表格并返回
    if return_tabulate:
        df = pd.DataFrame(topk_entities_dict.items(), columns = ['实体', '词频'])
        return tabulate(df, headers = 'keys', tablefmt = 'psql')
    else:
        return topk_entities_dict

# 从之前获取的集合建立人物社交图
# @param set_news_people: 新闻序号-人物集合字典/列表
# @return G: 社交网络图
def build_social_network(set_news_people):
    print('社交网络构建中...')
    # 计算每条边的权重 {(p1, p2) : {'weight': xxx}}
    edges_dict = {}
    for i in range(len(set_news_people)):
        # 把人物实体集合转成列表
        people = list(set_news_people[i])
        # 选定两个人，增加他们之间的权重
        for i in range(len(people)):
            for j in range(i + 1, len(people)):
                if (people[i], people[j]) not in edges_dict.keys():
                    edges_dict[(people[i], people[j])] = {'weight': 1}
                else:
                    edges_dict[(people[i], people[j])]['weight'] += 1
    # 把字典形式的边转化成列表 [(p1, p2, {'weight': xxx})]
    edges = [(*k, v) for k, v in edges_dict.items()]
    # 建图
    G = nx.Graph()
    G.add_edges_from(edges)
    return G

# 做各种预处理工作
# @param news_file: 新闻文本
# @param entity_file: 实体文件
# @param network_file: 社交网络文件
# @return G: 社交网络图
def preprocess(news_file, entity_file = 'default_entity_file.txt', network_file = '
```

```
    default_network_file.txt', cut_mode = 'jieba'):

    print('(1) 分词与实体提取:')
    # 如果实体文件不存在
    if not os.path.exists(entity_file):
        entity_freq, set_news_people =
    fetch_entities(news_file, entity_file, mode =
    cut_mode)
    else:
        #读取预处理文件
        print('利用已经存在的文件', entity_file, '
    ...')
        f = open(entity_file, 'r')
        l = f.readline()
        entity_freq = eval(l)
        l = f.readline()
        set_news_people = eval(l)
        f.close()
    print('分词与实体提取完成!')

    print('')
    print('(2) 热门人物/机构:')
    # 获取出现频率最高的若干实体
    k = 10
    print('出现频率最高的', k, '个实体为:')
    print(most_frequently_entities_topk(entity_freq
    , k))

    print('')
    print('(3) 社交网络构建:')
    # 如果网络文件不存在
    if not os.path.exists(network_file):
        G = build_social_network(set_news_people)
        nx.write_weighted_edgelist(G, network_file)
    else:
        # 读取预处理文件
        print('利用已经存在的文件', network_file, '
    ...')
        G = nx.read_weighted_edgelist(network_file)
    print('社交网络构建完成!')
    return G

# 找寻某节点权重最高的邻居
# @param graph: 图
# @param node: 节点
# @param top_k: 权重前 k 大
# @param return_tabulate: 是否返回表格
# @return 权重前 k 大的邻居的表格 或 dict
def strongest_neighbors_topk(graph, node, top_k =
    10, return_tabulate = True):
    # 没有该节点就不返回
    if node not in graph:
        print('图内没有该节点!')
        return None
    print('节点', node, '的邻居:')
    # 按 weight 降序排序邻接表的边
    # 并取前k个
    neighbor = sorted(graph[node].items(),
                      key = lambda kv: kv[1]['weight'
    ],
                      reverse = True)
    topk_neighbor_list = neighbor[:top_k]
    # 变成 {'neighbor': weight} 的简单字典
    topk_neighbor_dict = {k : v['weight'] for k, v
    in topk_neighbor_list}
    # 构建表格并返回
    if return_tabulate:
        df = pd.DataFrame(topk_neighbor_dict.items
    (), columns = ['邻居', '权重'])
        return tabulate(df, headers = 'keys',
    tablefmt = 'psql')
    else:
        return topk_neighbor_dict

# 图的统计数据
# @param graph: 图
# @return nodes, edges, components, largest_comp:
    图的节点数、边数、连通分量数以及最大连通分量节
    点数
def graph_statistics(graph):
    nodes = nx.number_of_nodes(graph)
    edges = nx.number_of_edges(graph)
    components = nx.number_connected_components(
    graph)
    largest_comp = len( max(nx.connected_components
    (graph), key = len) )
    print('节点个数:', nodes)
    print('边数:', edges)
    print('连通分量个数:', components)
    print('最大连通分量的大小:', largest_comp)
    return nodes, edges, components, largest_comp

# 影响力(pagerank) top_k
# @param graph: 图
# @param return_tabulate: 是否返回表格
# @return 节点影响力 top_k 表格 或字典
def pagerank_influence_topk(graph, top_k = 20,
    return_tabulate = True):
    # pagerank
    influence = sorted(nx.pagerank(graph).items(),
    key = lambda kv: kv[1], reverse = True)
    topk_influence_list = influence[:top_k]
```

```
    topk_influence_dict = {k : v for k, v in topk_influence_list}
    # 构建表格并返回
    if return_tabulate:
        df = pd.DataFrame(topk_influence_dict.items(), columns = ['节点', '影响力'])
        return tabulate(df, headers = 'keys', tablefmt = 'psql')
    else:
        return topk_influence_dict

# 基础分析
# @param graph: 社交网络图
def basic_analysis(graph):
    # 输入 A，输出与 A 关系最强的 10 个邻居
    print('(1) 图的验证:')
    node = input('请输入节点名: ')
    # node = '习近平'
    print(strongest_neighbors_topk(graph, node = node))

    print('')
    # 图的节点个数、边数、连通分量个数、最大连通分量大小
    print('(2) 图的统计:')
    graph_statistics(graph)

    print('')
    # PageRank 最高的20个人
    print('(3) 影响力计算:')
    print(pagerank_influence_topk(graph, top_k = 20))

# (1) 小世界理论验证
# (未实现)
def smallworld_validate(graph):
    print('暂未实现')

# (2) 三元闭包验证
# (未实现)
def ternary_closure_validate(graph):
    print('暂未实现')

# (3) 社区挖掘
# @param graph: 社交网络图
# @param top_k: 打印前 k 个社区
# @param print_all: 是否打印社区内所有节点
# @param mode: 社区发现模式 (louvain/asyn_lpa/lpa)
# @return louvain 模式下返回 partition: {节点:所属社区}，其他两种模式返回社区生成器 partition_gen
def community_detection(graph, top_k = 5, print_all = False, mode = 'louvain'):
    # 先算所有节点的 pagerank
    pagerank_all = pagerank_influence_topk(graph, graph.number_of_nodes(), return_tabulate = False)
    # louvain 算法
    if mode == 'louvain':
        partition = community.best_partition(graph)
        # 社区节点列表的列表
        comms = []
        for com in set(partition.values()):
            com_nodes_list = [node for node in partition.keys()
                              if partition[node] == com]
            # 社区内节点按 pagerank 降序排列
            com_nodes_list = sorted(com_nodes_list, key = pagerank_all.get, reverse = True)
            comms.append(com_nodes_list)
        # 所有社区按节点数目降序排列
        comms = sorted(comms, key = len, reverse = True)
        print('在', mode, '社区发现模式下，共划分出', len(comms), '个社区')
        print('模块度为:', nx_comm.modularity(graph, comms))
        print()
        print('前', top_k, '个社区为:')
        print()
        # 打印前k个社区
        for i in range(top_k):
            if print_all:
                print('社区', i + 1, ':')
                print('大小:', len(comms[i]))
                print('节点:', comms[i])
            else:
                print('社区', i + 1, ':')
                print('大小:', len(comms[i]))
                print('代表元:', comms[i][0])
            print()
        return partition
    # Label propagation
    elif mode == 'asyn_lpa' or mode == 'lpa':
        if mode == 'asyn_lpa':
            partition_gen = nx_comm.asyn_lpa_communities(graph)
        else:
            partition_gen = nx_comm.label_propagation_communities(graph)
        comms = []
        for com in partition_gen:
```

```
            com_nodes_list = list(com)
            com_nodes_list = sorted(com_nodes_list, key = pagerank_all.get, reverse = True)
            comms.append(com_nodes_list)
        comms = sorted(comms, key = len, reverse = True)
        print('在', mode, '社区发现模式下，共划分出', len(comms), '个社区')
        print('模块度为:', nx_comm.modularity(graph, comms))
        print()
        print('前', top_k, '个社区为:')
        print()
        for i in range(top_k):
            if print_all:
                print('社区', i + 1, ':')
                print('大小:', len(comms[i]))
                print('节点:', comms[i])
            else:
                print('社区', i + 1, ':')
                print('大小:', len(comms[i]))
                print('代表元:', comms[i][0])
            print()
        return partition_gen
    else:
        print('社区发现模式不存在!')
        return None

# (4) 中介中心性计算
# @param graph: 社交网络图
# @param sample_ratio: 近似算法的采样比
# @param top_k: 前k
# @param return_tabulate: 是否返回表格
# @return 中心性 top_k 表格或字典
def centrality_topk(graph, sample_ratio = 0.05, top_k = 10, return_tabulate = True):
    centrality = sorted(
            nx.betweenness_centrality(graph, k = int(sample_ratio * graph.number_of_nodes()) ).items(),
            key = lambda kv: kv[1], reverse = True)
    topk_centrality_list = centrality[:top_k]
    topk_centrality_dict = {k : v for k, v in topk_centrality_list}
    # 构建表格并返回
    if return_tabulate:
        df = pd.DataFrame(topk_centrality_dict.items(), columns = ['节点', '中心性'])
        return tabulate(df, headers = 'keys', tablefmt = 'psql')
    else:
        return topk_centrality_dict

# (5) 聚集系数计算
# @param graph: 社交网络图
# @param top_k: 前k
# @param return_tabulate: 是否返回表格
# @return 聚集系数 top_k 表格 或字典
def clustering_coefficient_topk(graph, top_k = 10, return_tabulate = True):
    cc = sorted(nx.clustering(graph).items(), key = lambda kv: kv[1], reverse = True)
    topk_cc_list = cc[:top_k]
    topk_cc_dict = {k : v for k, v in topk_cc_list}
    # 构建表格并返回
    if return_tabulate:
        df = pd.DataFrame(topk_cc_dict.items(), columns = ['节点', '聚集系数'])
        return tabulate(df, headers = 'keys', tablefmt = 'psql')
    else:
        return topk_cc_dict

# (6) 结构洞挖掘
# (未实现)
def structural_holes_detection(graph):
    print('暂未实现')

# 可选分析
# @param graph: 社交网络图
def optional_analysis(graph):
    # 六选三
    print('(1) 小世界理论验证:')
    smallworld_validate(graph)

    print('')
    print('(2) 三元闭包验证:')
    ternary_closure_validate(graph)

    print('')
    print('(3) 社区挖掘:')
    community_detection(graph, top_k = 5, print_all = False, mode = 'louvain')
    community_detection(graph, top_k = 5, print_all = False, mode = 'asyn_lpa')
    community_detection(graph, top_k = 5, print_all = False, mode = 'lpa')

    print('')
    print('(4) (中介) 中心性计算:')
    print(centrality_topk(graph, sample_ratio = 0.02, top_k = 10))
```

```
431
432     print('')
433     print('(5) 节点的聚集系数计算:')
434     print(clustering_coefficient_topk(graph, top_k
        = 10))
435
436     print('')
437     print('(6) 结构洞挖掘:')
438     structural_holes_detection(graph)
439
440 if __name__ == '__main__':
441     print('一、数据预处理:')
442     G = preprocess(news_file = './data/gov_news.txt
        ', entity_file = './data/entities_data_baidu.
        txt',
443         network_file = './data/network_data_baidu.
        txt', cut_mode = 'baidu')
444     nx.write_gexf(G, 'graph.gexf')
445     print('')
446     print('二、基础分析:')
447     basic_analysis(G)
448     print('')
449     print('三、可选分析:')
450     optional_analysis(G)
```

附录 B 社区可视化结果

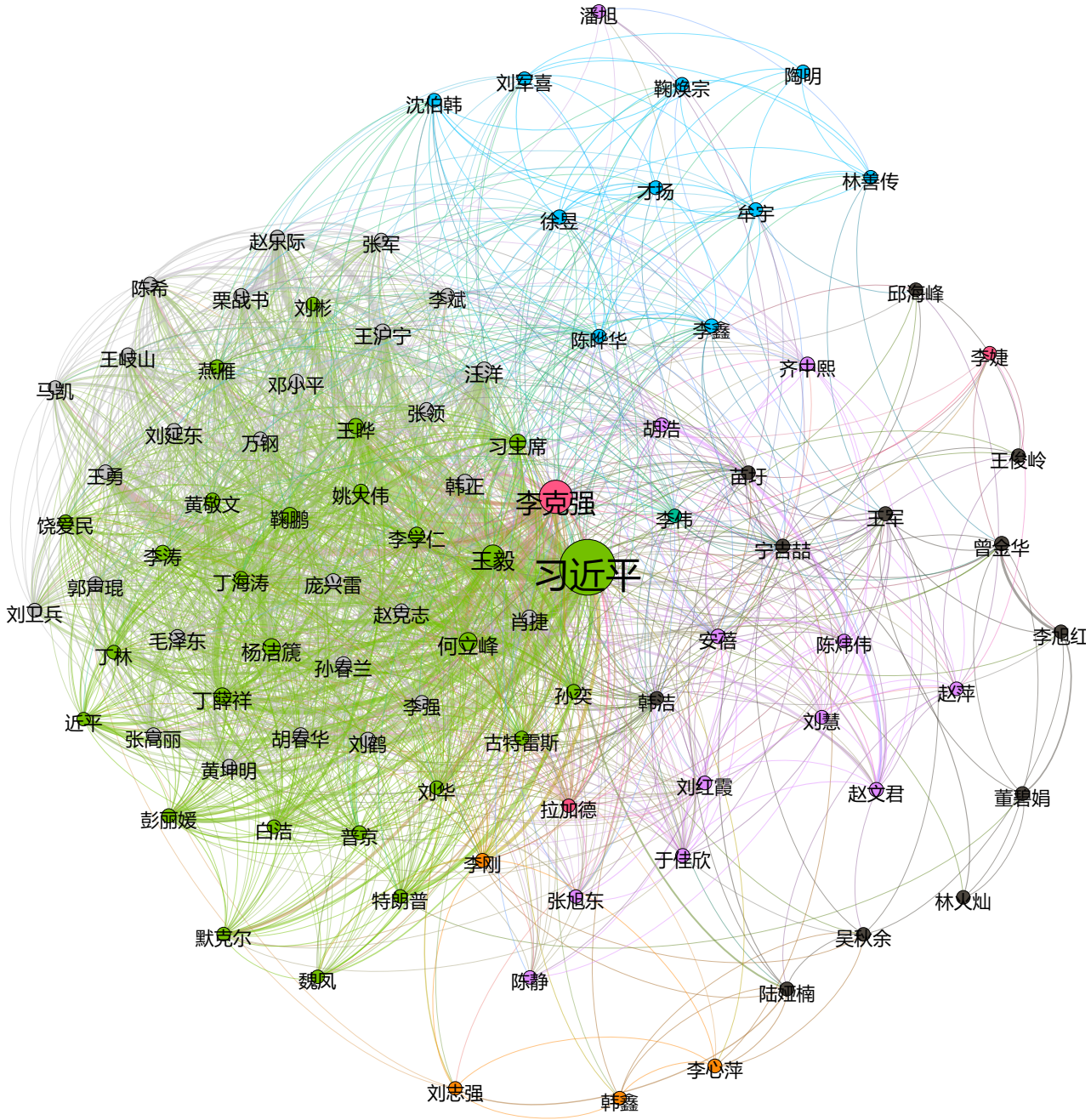
运行社区检测算法后，选取 PageRank 最高的 96 个节点进行可视化，每种颜色代表原图的一个社区。（见下页）

Guanyu Cui, Undergraduate. His research interests include Set Theory, Theory of Computation and Computational Complexity.

Background

（略）

（此页留空）


潘旭
刘幸喜
鞠焕宗
陶明
沈伯韩
才扬
林鲁传
华宇
徐昱
赵尔际
张军
陈希
栗战书
刘彬
李斌
邱海峰
王沪宁
陈晔华
李鑫
王岐山
燕雁
邓小平
汪洋
齐中熙
李婕
马凯
张领
刘延东
万钢
王晔
胡浩
习主席
王俊岭
王勇
苗圩
韩正
姚大伟
李克强
黄敬文
鞠鹏
李伟
王军
饶爱民
李学仁
宁吉喆
曾金华
李涛
王毅
习近平
丁海涛
庞兴雷
郭声琨
刘卫兵
赵克志
肖捷
李旭红
毛泽东
何立峰
安蓓
陈炜伟
丁林
杨洁篪
孙春兰
赵萍
丁薛祥
孙奕
韩浩
李强
刘慧
近平
张高丽
胡春华
刘鹤
古特雷斯
黄坤明
刘红霞
赵文君
董碧娟
刘华
拉加德
彭丽媛
白浩
普京
于佳欣
李刚
特朗普
张旭东
林火灿
默克尔
吴秋余
魏凤
陈静
陆娅楠
李心萍
刘思强
韩鑫