



北京大学

数据结构与算法实验报告

题目： **【FIN】演员的小世界实验报告**

姓 名： 田晨霄

学 号： 1700013239

学 院： 数学科学学院

实验日期： 2020/06/10

目录

1.项目概览 代码目录结构与运行环境.....	4
1.1 代码目录结构	4
1.2 目录及项目运行步骤说明	4
1.3 项目运行环境	4
2.前置准备 项目数据读取与预整理.....	5
2.1 代码与算法分析	5
2.3 数据分析.....	6
3.建立模型 建立无向图模型.....	6
3.1 代码与算法分析	6
3.2 运行结果与数据分析	7
4.实验报告 连通分支数量、规模、电影类别统计.....	8
4.1 代码与算法分析	8
4.2 运行结果	9
4.3 数据分析	9
5.实验报告 连通分支直径计算.....	9
5.1 代码与算法分析	9
5.2 运行结果	11
5.3 数据分析.....	11
6.实验报告 连通分支规模、直径、平均星级作图.....	11

6.1 代码与算法分析	11
6.2 运行结果 1（所有连通分支全部作出的情况）	13
6.3 结果分析 1（所有连通分支全部作出的情况）	14
6.4 运行结果 2（只作前后 20 个分支的情况）	14
■6.4.1 调整后的代码	14
■6.4.2 作图结果 2.....	15
■6.4.3 结果分析 2（只作前后 20 个分支的情况）	16
 7.实验报告 有关周星驰的演艺信息统计	16
7.1 代码与算法分析	16
7.1 运行结果 1（共同出演者总共的统计）	17
7.2 运行结果 2（共同出演者分别逐个统计）	17
 8.自选报告 基于网络排序算法的演员影响力排名.....	18
8.1 自选题目与课内内容的联系与算法简介	18
8.2 代码与算法分析	18
8.3 运行结果	19
■8.3.1 演员影响力排名列表:	19
■8.3.2 PageRank 因子密度分布图:.....	20
■8.3.3 最具影响力前 5 名演员位置子图:	20
8.4 算法发现与分析.....	21
 9.项目展望	21
 10.算法各部运行时间简计	22
 11.附录.....	22

演员的小世界实验报告

【教师】 陈斌老师
【课堂】 数据结构与算法(B)
【学院】 数学科学学院
【姓名】 田晨霄
【学号】 1700013239

1.项目概览 代码目录结构与运行环境

1.1 代码目录结构

名称	日期被修改
constants.bak	2020/6/9 16:51
constants.dat	2020/6/9 16:51
constants.dir	2020/6/9 16:51
draw.py	2020/6/9 16:52
Film.json	2020/6/4 17:04
Graph.py	2020/6/8 22:50
main.py	2020/6/9 16:51
pagerank.py	2020/6/8 15:48

1.2 目录及项目运行步骤说明

■**代码目录结构由:** Graph.py、draw.py、main.py、pagerank.py 四个模块, 以及 Film.json 原始数据和储存了所有连通分支直径、平均星级、规模大小和演员的 pagerank 排名 4 个 list 对象的 constants 系列文件, 共计 8 个文件构成。↵

■**在实际执行项目时:** ↵

我们只需将以上目录文件按原目录关系 copy 到计算机相应位置, 然后用 python 解释器执行 main.py 即可打印出全部项目要求分析的结果和自选挖掘的结论。↵

1.3 项目运行环境

■**原项目于本地运行时:** 环境使用的是 Anaconda Base Environment, 解释器为 spyder 解释器, 但需要提前在 Powershell Prompt 中使用 pip install 命令于根环境中额外安装 matplotlib、brokenaxes 等 Python 包。↵

2.前置准备 项目数据读取与预整理

2.1 代码与算法分析

■对应 main.py 第 16 至 65 行：将所有出现过的演员整理至表格 Actors，与此同时，将各个演员演过的电影、星级、电影种类分别整理至 Movies、Stars、Types 多维表格，注意要保持索引位置一一对应。同时整理含边信息的字典 EdgesValuesDic，其截图如下：

```

16  """ 读取整理原始json文件"""
17  #####
18  import json
19  """ 读取json文件"""
20  file_path = 'Film.json'
21  with open(file_path,"rb") as f:
22      js = json.load(f) #
23      """actor 转为列表"""
24  for i in range(len(js)):
25      js[i]["actor"]=js[i]["actor"].split(",")
26      # if "" in js[i]["actor"]: #注1: 由于数据质量原因, 本应去掉空串考虑顿号等因素, 但
27      #     js[i]["actor"].remove("") 统一标准起见, 本次项目暂时依照原始数据, 不做额外预处理
28      """type 转为列表"""
29  for i in range(len(js)):
30      js[i]["type"]=js[i]["type"].split(",")
31      """依次统计所有actors"""
32  Actors=[] #注2: Actors用于记录所有出现过的演员
33  for i in range(len(js)):
34      for j in range(len(js[i]["actor"])):
35          Actors.append(js[i]["actor"][j])
36  Actors=list(set(Actors)) #注3: 去掉重复演员名
37  """记录演员于列表中位置的字典"""
38  find={}
39  for i in range(len(Actors)): #注4: 一个辅助反查演员于表位置的字典, 减小
40      find[Actors[i]]=i #后续一些算法的复杂度
41  """依次统计所有演员演过的Movies以及记录此电影的星级、种类。"""
42  Movies=[] #注5: 用于整理Actors相应位置演员演过的电影, 为多重列表
43  Stars=[] #注6: 用于记录Movies相应位置的电影星级数, 为多重列表
44  Types=[] #注7: 用于记录Movies相应位置的电影所属种类, 为多重列表

```

```

45  for x in js:
46      temp1=x["actor"]
47      temp2=x["title"]
48      temp3=x["star"]
49      temp4=x["type"]
50  for y in temp1:
51      index=find[y]
52      Movies[index].append(temp2)
53      Stars[index].append(temp3)
54      Types[index].append(temp4)
55  """依次统计所有参与了同一电影演出的演员对, 及其所有合作电影列表"""
56  from collections import defaultdict #注8: 用于字典初始化的value都自动设为空表
57  EdgesValuesDic=defaultdict(list) #注9: 初始化字典, 每个key是共同出演的演员元组, value为电影列表
58  for x in js:
59      temp=x["actor"]
60      if len(temp)>1: #注10: 至少有大于1个演员, 才成对出现
61          for a in range(len(temp)):
62              for b in range(a+1,len(temp)):
63                  if temp[a]!=temp[b]:
64                      co=(min(temp[a],temp[b]),max(temp[a],temp[b]))
65                      EdgesValuesDic[co].append(x["title"])

```

2.2 运行结果¹

■整理完成了：Actors、Movies、Stars、Types、EdgesValuesDic 5 个列表或者字典对象，输出示例如下：

¹ 实际程序运行时以上内容均未打印。

表 2.2 整理对象输出结果

输出对象名称	对象类型	输出结果
Actors	List	['', '信贵千惠子 Chieko Baishō', 'Eugene Collier', '马克·达赫蒂', '崔成国', '詹姆斯·格雷戈里', '达纳·古尔德', '里卡杜·卡里可', '葛里高利·巴奎特', '钟丽缇', 'Eddie McGee', '孙雪宁', '马克·赖力施', '阿萨·巴特菲尔德', 'Cassidy Paige Bringas', '康文轩', 'Alice Herz Sommer', 'Roger Barclay', '梅根·崔娜', '伊莎贝尔·高德']
Movies	List	['失陷猩球 Beneath the Planet of the Apes'], ['万圣节传说 Tales of Halloween', '一路向南 Southbound'], ['葡萄牙队长 Capitão Falcão'], ['罗密欧与朱丽叶 Romeo & Juliette: De la haine à l'amour'], ['别恋', '哪一天我们会飞 哪一天我们会飞'], ['人族 The Human Race'], ['儿童总动员', '器灵 第一季'], ['诺斯福克 Northfork', '双子的天空 Twin Falls Idaho', '保持冷静 Stay Cool'], ['安德的游戏 Ender's Game', '回到火星 The Space Between Us', '佩小姐的奇幻城堡 Miss Peregrine's Home for Peculiar Children', '雨果 Hugo', '万圣年代 Ten Thousand Saints', '魔法保姆麦克菲2 Nanny McPhee Returns', '穿条纹睡衣的男孩 The Boy in the Striped Pajamas']
Stars	List	[[3.9, 3.9, 5.2, 5.2, 4.0, 5.3, 4.1, 4.1, 5.3, 4.7, 4.9, 5.4, 5.1, 5.1, 5.1, 5.1, 5.5, 5.5, 5.5, 5.5, 5.8, 5.8, 4.3, 4.3, 5.9, 5.6, 5.8, 5.9, 4.5, 4.7, 4.6, 6.0, 5.7, 6.3, 5.7, 6.2, 6.1, 5.7, 6.3, 6.3, 6.3, 6.0, 6.3, 6.3, 6.6, 6.6, 6.6, 6.6, 6.7, 6.5, 6.5, 6.7, 6.4, 6.5, 6.7, 6.7, 6.7, 6.7, 6.6, 6.8, 7.0, 7.0, 6.9, 6.9, 7.2, 7.0, 7.2, 6.9, 6.9, 7.1, 7.1, 7.1, 7.3, 7.1, 7.1, 7.3, 7.5, 7.5, 7.6, 7.5, 7.9, 7.5, 7.9, 7.9, 8.4, 8.3, 7.9, 8.3, 8.3, 7.5, 8.3, 7.9, 7.7, 7.7, 7.5, 8.4, 8.0, 7.7, 8.0, 8.0, 8.0]]
Types	List	['奇幻', '冒险'], ['剧情', '儿童', '传记', '奇幻'], ['剧情', '喜剧', '音乐'], ['喜剧', '家庭', '奇幻'], ['剧情', '战争'], ['剧情', '喜剧', '悬疑', '奇幻'], ['剧情', '儿童', '奇幻'], ['剧情', '纪录片', '短片', '音乐', '传记'], ['剧情', '战争'], ['喜剧', '动画', '奇幻', '冒险'], ['动画', '奇幻']]
EdgesValuesDic	Dictionary	{'Hereditary': {'Brock McKinney', '奥斯丁·R·格兰特': ['遗传厄运 Hereditary'], 'Marilyn Miller', '奥斯丁·R·格兰特': ['遗传厄运 Hereditary'], 'Jason Miyagi', '奥斯丁·R·格兰特': ['遗传厄运 Hereditary'], 'Brock McKinney', 'Rachelle Hardy': ['遗传厄运 Hereditary'], 'Marilyn Miller', 'Rachelle Hardy': ['遗传厄运 Hereditary'], 'Jason Miyagi', 'Rachelle Hardy': ['遗传厄运 Hereditary'], 'Brock McKinney', 'Marilyn Miller': ['遗传厄运 Hereditary'], 'Brock McKinney', 'Jason Miyagi': ['遗传厄运 Hereditary'], 'Jason Miyagi', 'Marilyn Miller': ['遗传厄运 Hereditary']}}

2.3 数据分析

■此节运行结果不在必要的输出中，建立了几个方便后续数据整理和应用的对象，特别是前 4 个表格是按 index 索引一一对应的，一个 index，一个 actor 的姓名字符串、电影列表、星级列表、电影种类二维列表。（因为一个演员可演多部电影，一部又有多个分类）

3.建立模型 建立无向图模型

3.1 代码与算法分析

■对应 main.py 67 行至 73 行以及 Graph.py 7 至 57 行，截图分析如下：

(1) ■Graph.py 模块 7 至 57 行：构建图模型，包括节点和图对象，边属性储存在 edgevalues 字典中：

代码接下页


```

7  class Vertex:
8      def __init__(self, key,value=None):
9          self.id = key
10         self.val=value
11         self.connectedTo = {}
12         self.connectedToValues={}
13     def addNeighbor(self, nbr, weight=0,value=None):
14         self.connectedTo[nbr] = weight
15         self.connectedToValues[nbr] = value
16     def __str__(self):
17         return str(self.id) + 'connectedTo:' + str([x.id for x in self.connectedTo])
18     def getConnections(self):
19         return self.connectedTo.keys()
20     def getId(self):
21         return self.id
22     def getWeight(self, nbr):
23         return self.connectedTo[nbr]
24 class Graph:
25     def __init__(self):
26         self.vertList = {}
27         self.numVertices = 0
28         self.edgevalues={}
29     def addVertex(self, key,value=None):
30         self.numVertices = self.numVertices + 1
31         newVertex = Vertex(key,value)
32         self.vertList[key] = newVertex
33         return newVertex
34
35     def getVertex(self, n ):
36         if n in self.vertList:
37             return self.vertList[n]
38         else:
39             return None
40

```

```

40
41     def __contains__(self, n):
42         return n in self.vertList
43
44     def addEdge(self, f, t, cost = 0,value=None):
45         if f not in self.vertList:
46             self.addVertex(f)
47         if t not in self.vertList:
48             self.addVertex(t)
49         self.vertList[f].addNeighbor(self.vertList[t],cost)
50         self.vertList[t].addNeighbor(self.vertList[f],cost)
51         self.edgevalues[(min(f,t),max(f,t))]=value
52
53     def getVertices(self):
54         return self.vertList.keys()
55
56     def __iter__(self):
57         return iter(self.vertList.values())
58

```

(2) ■ main.py 模块 67 至 73 行：利用之前的 Actors 和 Movies 表格建立无向图模型。

```

67     """ 构建图 """
68     import Graph as G
69     graph=G.Graph()
70     for i in range(len(Actors)):
71         graph.addVertex(Actors[i],Movies[i]) #注11: 遍历Actors表, 以其所演电影为节点附加属性
72     for keys in EdgesValuesDic: #注12: 遍历EdgesValuesDic字典, 以其value为边属性
73         graph.addEdge(keys[0],keys[1],1,EdgesValuesDic[keys])
74

```

3.2 运行结果与数据分析

■ 运行结果构建了一个以演员为节点, 所演电影列表为附加属性; 有共同出现为边, 以共同出演电影列

表为边附加属性的自定义的无向图 Graph 对象。

4.实验报告 连通分支数量、规模、电影类别统计²

4.1 代码与算法分析

(1) ■ **main.py 83 至 121 行**: 调用 Graph.py 的 find_all_subs 方法, 统计连通分支数、规模、电影类别:

```

83 """求连通分支数目、每个分支所含演员数目、前20和后20规模的连通分支所含电影类别的前三名"""
84 from collections import Counter
85 start=time.time()
86 def counter(arr): #注13:统计一个列表中出现的最多的三种元素及其出线次数
87     return Counter(arr).most_common(3)
88 def movietypes(form): #注14:form输入的为每个连通分支的演员列表, 返回电影种类统计列表 (去重的)
89     temp=[]
90     for s in form:
91         k=Movies[find[s]]
92         for t in k:
93             temp.append(t)
94     temp=list(set(temp))
95     types=[]
96     for t in temp:
97         for k in js[findmovies[t]]["type"]:
98             types.append(k)
99     return types
100 s=graph.find_all_subs()#注15:调用Graph.py模块中的find_all_subs()函数遍历寻找所有连通分支
101 print("连通分支数目: ",len(s))
102 w=sorted(s,key=(lambda x:len(x)))#注16:按连通分支大小, 从小到大排序
103 print("每个连通分支所含的演员数: ")
104 print([len(x) for x in w])
105 mostthreetypesfirst={} #注17:用于记录前20最大的分支的电影种类
106 mostthreetypeslast={} #注18:用于记录后20最小的分支的电影种类
107 print("前20和后20规模的连通分支所含电影类别的前三名: ")
108 print([len(x) for x in w])
109 tobesort=[]
110 for i in range(len(w)):
111     if len(w[i])>1:
112         break
113     tobesort.append(w[i])
114 tobesort.sort(key=(lambda ele:ele[0])) #新增调整: 后20并列表中,字符串升序取前20。
115 print([len(x) for x in w])
116 for i in range(20):
117     mostthreetypeslast["倒数第"+str(i+1)+"大小的分支"]=counter(movietypes(tobesort[i]))
118 for i in range(20):
119     mostthreetypesfirst["正数第"+str(i+1)+"大小的分支"]=counter(movietypes(w[-1-i]))
120 print(mostthreetypesfirst)
121 print(mostthreetypeslast)

```

(2) ■ **Graph.py 第 59 至 77 行**: 定义 find_all_subs 方法, 寻找所有连通分支, 返回为多重列表。

■ **用基于深度遍历 DFS 算法**: 用栈非递归实现的分离连通分支的算法,总运行和统计时间只需 2.61s。

```

59 def dfs(self,start): #注19:深度遍历dfs
60     visited, stack = set(), [start]
61     while stack:
62         vertex = stack.pop()
63         if vertex not in visited:
64             visited.add(vertex)
65             stack.extend(set(change(list(self.vertList[vertex].connectedTo.keys())) - visited))
66     return visited
67
68 def find_all_subs(self):#注20: 寻找所有连通分支, 并返回为多重列表
69     temp=set(list(self.vertList.keys()))
70     allsubs=[]
71     while len(temp)>0:
72         cur=list(temp)[0]
73         temp1=self.dfs(cur)
74         allsubs.append(list(temp1))
75         s=temp-temp1
76         temp=s
77     return allsubs

```

² 全部输出可参看提交内容的“全部输出内容.txt”文档。

■第 78 至 116 行：定义基于弗洛伊德算法寻找任意子图的直径的方法，时间复杂度为 $O(N^3)$

■采用了基于动态规划思想的弗洛伊德算法，为节约时间，讨论了连通分支大小为 1, 2, 3 的情况再进行弗洛伊德算法。↵

■算法效率较高，除去最大分支，其他所有分支求直径的本地运行时间为 0.18s。↵

(2) main.py 部分：

■第 104 行至 117 行：调用算法求直径列表，可考虑 `shelve` 储存(但事实上时间成本很小，也可不储存)，以方便下次调用直径列表：

```

78  def change(form):                                #注21: 辅助函数，将节点列表转化为对应的节点id列表
79      for i in range(len(form)):
80          form[i]=form[i].id
81      return form
82  def haskey(x,d):                                  #注22: 辅助函数，查找字典里面是否有key
83      if x in d.keys():
84          return True
85      else:
86          return False
87  def find_diam(graph,form,EdgesvaluesDic):#注23:动态规划版本的弗洛伊德算法，复杂度 $O(n^3)$ 
88      if len(form)==1:
89          return 0
90      if len(form)==2:
91          return 1
92      form.sort()
93      if len(form)==3:
94          temp=[(form[0],form[1]),(form[1],form[2]),(form[0],form[2])]
95          if [haskey(x,EdgesvaluesDic) for x in temp]==[True,True,True]:
96              return 1
97          else:
98              return 2
99      vertex,inf,dis=len(form),99999999,[]
100     dis=[[inf for j in range(vertex)] for i in range(vertex)]
101     for i in range(vertex):
102         dis[i][i]=0
103         for j in range(i+1,vertex):
104             temp=form[i],form[j]
105             if haskey(temp,EdgesvaluesDic)==True:
106                 dis[i][j]=1
107                 dis[j][i]=1
108         for k in range(vertex):
109             for i in range(vertex):
110                 for j in range(vertex):
111                     if dis[i][j] > dis[i][k] + dis[k][j]:
112                         dis[i][j]= dis[i][k] + dis[k][j]
113     return max(max(dis))
114  if __name__=="__main__":
115     a=Vertex("ds")
116     print(a.id)

```

(代码接下页)

```

104     """求直径"""
105     import shelve
106     Dias=[]
107     j=0
108     for i in range(len(w)-1): ##注24:最大的不计算
109         j=j+1
110         print(j)
111         Dias.append(G.find__diam(w[i],graph.edgevalues))
112     Dias.append(-1)
113     with shelve.open ('constants') as h: #注25: 储存算好的直径表于constants中
114         h["Dias"]=Dias
115     with shelve.open ('constants') as h: #注26: 从constants中提取直径表
116         Dias=h["Dias"]
117     print(Dias)

```

5.2 运行结果

表尾部分示例（及最大连通分支的直径“-1”所在表的位置）：

```

1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 3, 2, 1, 1, 1, 2, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 4, 1, 2, 1, 1, 1, 1, 3, 3, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
2, 1, 1, 3, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1,
3, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 3, 2, 2, 1, 1, 2, 2, 1, 1, 1, 3, 1, 1, 1, 2, 1, 2, 1,
2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 1,
2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 3, 4, 1, 1, 3, 2, 1, 2, 1, 1, 1, 1, 1, 2, 3,
1, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 3, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 3, 2, 2, 1, 4, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1,
1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 3, 1, 1, 1, 2, 1, 3, 1, 1, 1, 3, 3, 1, 1, 1, 2, 2, 1, 1, 1, 1,
1, 1, 2, 1, 2, 3, 1, 1, 3, 3, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 2, 2, 3, 2, 2, 2, 2, 2, 2, 3, 4, 2, 1, 3, 1, 3, 5, 4, 3, -1]

```

注：利用 C++ 语言，在 2 个小时内可以较高效率的求出最大连通分支的直径数值为 17，但 python 运行时于表尾暂记其值为 -1。

4 1 3 1 3 5 4 3 17

C:\Users\田晨霄\Desktop\大学courses\数据结构与算法B

5.3 数据分析

■这组数据从一定程度上再次验证了六度分隔理论，即光靠电影演艺这个渠道联系上的人群中，除了最大的连通分支外，其余都在 5 人之内就能认识（次大直径是不超过 5 的）。

■绝大多数小连通分支的直径甚至为 0 或 1，虽排除数据质量的问题，但也足见六度分隔理论的合理性。

6.实验报告 连通分支规模、直径、平均星级作图

6.1 代码与算法分析

(1) draw.py 模块

■定义 4 种不同功能的作图函数（见 6.2 运行结果），在 draw.py 内作其中 1 种 y 轴截断图：

```

7  from brokenaxes import brokenaxes
8  import matplotlib.pyplot as plt
9  def draw(namelist=None, lenlist=None, dialist=None, starlist=None):
10     x=[0 for i in range(len(lenlist))]
11     y=[0 for i in range(len(lenlist))]
12     z=[0 for i in range(len(lenlist))]
13     width =0.9
14     x[0]=x[0]
15     y[0]=y[0]+width/9
16     z[0]=z[0]+2*width/9
17     for i in range(len(x)-1):
18         x[i+1] = x[i] + width/3
19         y[i+1] = y[i] + width/3
20         z[i+1] = z[i] + width/3
21     plt.bar(x, dialist, width=width/9, label='Dias Change Function:  $\text{math.log}(\text{dias}+1.25,4)$ ',f
22     plt.bar(y, lenlist, width=width/9, label="Lens Change Function:  $\text{math.log}(\text{math.log}(\text{math.L$ 
23     plt.bar(z, starlist, width=width/9, label='Stars Change Function:  $\text{math.log}(\text{stars},3.5)$ ',f
24     plt.xticks(rotation=-45)
25     plt.legend()
26     plt.show()
27 def singledraw(namelist=None, form=None, l=None, color=None):#注28:分别作单个信息的柱状图
28     x =list(0 for i in range(len(form)))
29     width =0.001
30     for i in range(len(x)-1):
31         x[i+1] = x[i] + width+0.00005
32     plt.bar(x,form, width=width, label=l,tick_label = namelist,fc = color)
33     plt.xticks(rotation=-45)
34     plt.legend()
35     plt.show()

36 def originaldraw(form=None, labels=None, k=True):
37     x =list(0 for i in range(len(form)))
38     width =1
39     for i in range(len(x)-1):
40         x[i+1] = x[i] + width
41     if k==True:
42         bax = brokenaxes(ylims=((0,50),(84680,84690)),hspace=0.05, despine=False)
43     else:
44         bax = brokenaxes(despine=False)
45     bax.plot(x,form,label=labels)
46     bax.legend(loc=3)
47     bax.set_xlabel('The Position Of Connected Component Based On Scale')
48     bax.set_ylabel('Value')
49 def originaldrawall(form1=None, form2=None, form3=None, labels=None, k=None):
50     x =list(0 for i in range(len(form1)))
51     width =1
52     for i in range(len(x)-1):
53         x[i+1] = x[i] + width
54     if k==True:
55         bax = brokenaxes(ylims=((0,50),(84680,84690)),hspace=0.05, despine=False)
56     else:
57         bax = brokenaxes(despine=False)
58     bax.plot(x,form1,label=labels[0])
59     bax.plot(x,form2,label=labels[1])
60     bax.plot(x,form3,label=labels[2])
61     bax.legend(loc=3)
62     bax.set_xlabel('The Position Of Connected Component Based On Scale')
63     bax.set_ylabel('Value')
64
65 if __name__=="__main__":
66     import shelve
67     with shelve.open('constants') as h:
68         Lens=h["L"]
69         Dias=h['Dias']
70         starlist=h["S"]
71     originaldraw(Lens,"lens",True)
72     originaldraw(Dias,"dias",False)
73     originaldraw(starlist,"stars",False)
74     originaldrawall(Lens,Dias,starlist,["lens","dias","stars"],True)

```

- (2) ■ main.py 第 118 至 148 行：作出对数变换数据的非截断的条形图：

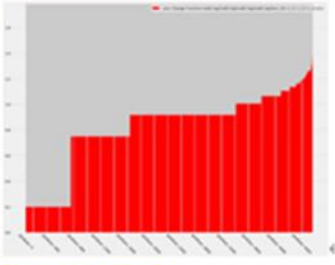
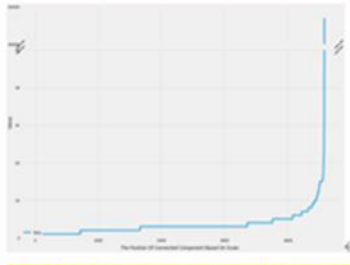
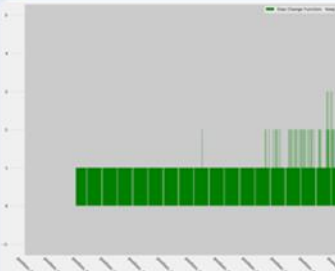
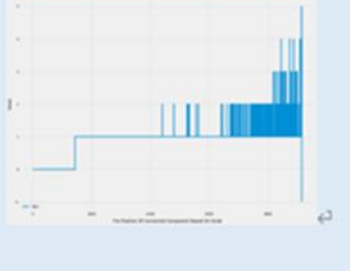

```

118 """做条形图"""
119 import matplotlib
120 import draw as d
121 import math
122 lenlist2=[math.log(math.log(math.log(len(x),10)+1,2)+1,2)+0.5 for x in w]
123 lenlist1=[math.log(math.log(math.log(len(x),10)+1,2)+1,2)+0.2 for x in w]
124 namelist=[]
125 for i in range(len(w)):
126     if i%400==0:
127         namelist.append("position: "+str(i))
128     else:
129         namelist.append("")
130 starlist=[]
131 for i in range(len(w)):          #注32: 统计每个分支的平均星级
132     temp1=w[i]
133     S=0
134     L=0
135     for j in range(len(temp1)):
136         temp2=Stars[find[temp1[j]]]
137         L=L+len(temp2)
138         S=S+sum(temp2)
139     starlist.append(round(S/L,2))
140 print(starlist)
141 with shelve.open('constants') as h      #注33: 储存数据
142     h["L"]=[len(x) for x in w]
143     h["S"]=starlist                    #注34: 作图
144 d.singledraw(namelist,lenlist1,"Lens Change Function:math.Log(math.Log(math.Log(Ler
145 d.singledraw(namelist,Dias,"Dias Change Function: Keep Original Data","g")
146 d.singledraw(namelist,[math.log(x+1.25,4) for x in Dias],"Dias Change Function; math.Log(x+1
147 d.singledraw(namelist,starlist,"Average Stars Change Function : Keep Original Data","y")
148 d.draw(namelist,lenlist2,[math.log(x+1.25,4) for x in Dias],[math.log(x,3.5) for x in starli

```

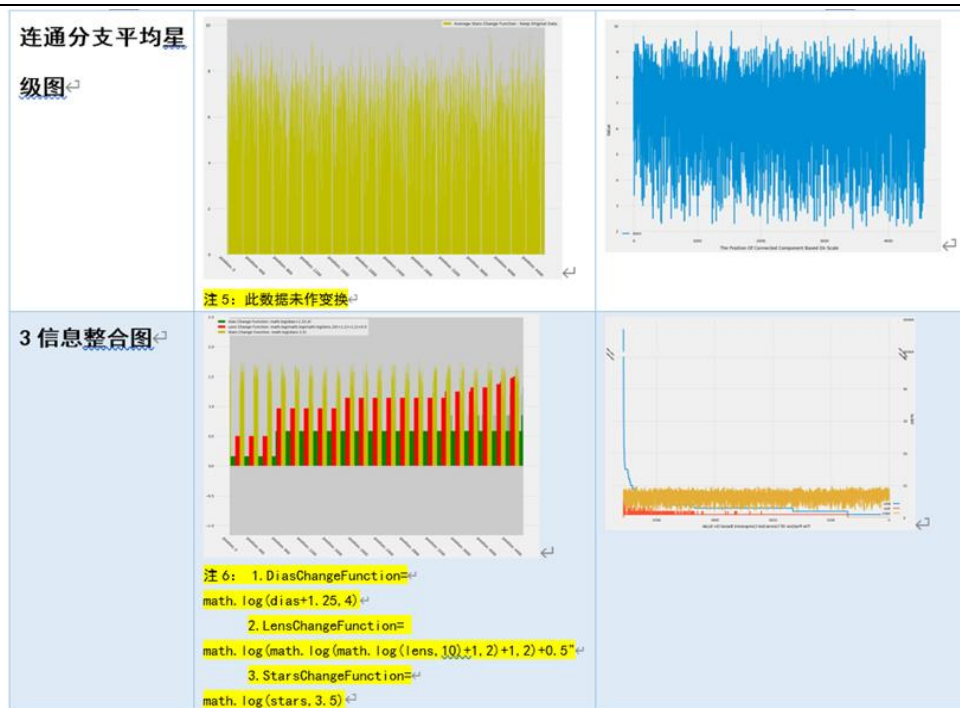
6.2 运行结果 1 (所有连通分支全部作出的情况)⁴

表 6.2 作图运行结果

作图对象	作图种类	
注 1: 两种方法克服数据量过大的问题	对数变换后的柱状图	坐标截断后的折线图
连通分支规模图	 <p>注 2: 所作对数型单调变换: $\text{math.log}(\text{math.log}(\text{math.log}(\text{math.log}(\text{len}(x),10)+1,2)+1,2)+0.5)$</p>	 <p>注 4: 截断位置为 $((0, 50), (84680, 84690))$, 下都同。</p>
连通分支直径图	 <p>注 3: 此数据未作变换</p>	

表格接下页

⁴ 所有原输出图储存在“所有图片程序输出材料”文件夹，另外本次作图作出的是所有的连通分支的图。



6.3 结果分析 1(所有连通分支全部作出的情况)

6.3 结果分析

(包括 10 页图表和 11 页续表)

■ 连通分支规模图

(见本页表格): 按连通分支规模从小到大画出, 可见最大的一个规模直线上升至 8 万有余, 其余的缓慢一个阶梯一个阶梯的平台式上升。

■ 连通分支直径图

(见本页表格): 同按由小到大连通分支画出, 绝大多数直径都是 0 或者 1, 直到较大的连通分支才出现 3、4、5 等较大的直径。

■ 连通分支平均星级图

(接下页续表): 按连通分支从小到大画出, 无明显规律, 随机分布。

■ 信息整合图

(接下页续表): 按连通分支从小到大画出, 规模、直径总体而言都是平台式上升, 特别是规模, 而直径大多数都是 1 或者 0 少数大连通分支可到达 3、4、5; 星级依然随机分布。

6.4 运行结果 2 (只作前后 20 个分支的情况)

6.4.1 调整后的代码

在主实验报告和整合 py 程序中: 所作的是全部连通分支的对数变换柱形图, 但由于第 3 题的实际操作要求更新, 下也只作出前 20 和后 20 分支的对数变换后的柱形图, 对应代码 74 至 91 行:


```

57 def originaldrawall(form1=None,form2=None,form3=None,labels=None,k=None):
58     x=list(0 for i in range(len(form1))) #注30:用坐标截断的方法同时作信息的折线图
59     width=1
60     for i in range(len(x)-1):
61         x[i+1]=x[i]+width
62     if k==True:
63         bax=brokenaxes(ylims=((0,50),(84680,84690)),hspace=0.05, despine=False)
64     else:
65         bax=brokenaxes(despine=False)
66     bax.plot(x,form1,label=labels[0])
67     bax.plot(x,form2,label=labels[1])
68     bax.plot(x,form3,label=labels[2])
69     bax.legend(loc=3)
70     bax.set_xlabel('The Position Of Connected Component Based On Scale')
71     bax.set_ylabel('Value')
72
73 if __name__=="__main__":
74     import shelve
75     import math
76     with shelve.open('constants') as h:
77         Lens=h['L']
78         Dias=h['Dias']
79         starlist=h['S']
80     Lens=Lens[:20]+Lens[-20:] #补注:只画前后20个分支的相关信息条形图的作图程序。
81     Dias=Dias[:20]+Dias[-20:]
82     starlist=starlist[:20]+starlist[-20:]
83     namelist=[str(i) for i in range(20)]+[str(20-i) for i in range(20)]
84     lens=[math.log(math.log(math.log(math.log(x,10)+1,2)+1,2)+1,2)+0.2 for x in Lens]
85     singledraw(namelist,lens,"Lens: Log change function: math.Log(math.Log(math.Log(x,10)+1,2)+1,2)+0.2")
86     singledraw(namelist,Dias,"Dias: Keep original data ","g")
87     singledraw(namelist,starlist,"stars:Keep original data","y")
88     Lens=[math.log(math.log(math.log(x,10)+1,2)+1,2)+0.5 for x in Lens]
89     Dias=[math.log(x+1.25,4) for x in Dias]
90     starlist=[math.log(stars,3.5) for stars in starlist]
91     draw(namelist,Lens,Dias,starlist)

```

6.4.2 作图结果 2



6.4.3 结果分析 2（只作前后 20 个分支的情况）

简析：由 6.4.2 节运行结果图可见，三个信息的变化趋势，由小分支到大分支，与主报告全部作出的情况完全一致（参见主报告或北大网盘提交全部输出图片文件夹）。

7.实验报告 有关周星驰的演艺信息统计

7.1 代码与算法分析

■main.py 第 163 行至第 200 行：直接利用 js 表、find 字典和 Stars 表遍历即可得单人演员的统计；遍历 js 表，可得所有共同出演者的总信息：

```

163  """特定演员个人统计"""
164  start=time.time()
165  index=find["周星驰"]
166  print("平均星级为: ",round(sum(Stars[index])/len(Stars[index]),2))#周星驰平均星级
167  company=[]
168  for y in js:
169      if "周星驰" in y["actor"]:
170          company=company+y["actor"]
171  company=list(set(company))
172  print("共同出演者人数 含周星驰本人 不去空串: ",len(company))
173  recordnum={}
174  recordstar={}
175  recordtypes={}
176  temp=set(company)
177  totalmovies=0
178  totalstars=0
179  totaltypes=[]
180  for m in js:
181      if temp & set(m["actor"]) !=set():
182          totalmovies+=1
183          totalstars+=m["star"]
184          totaltypes+=m["type"]
185  print("共同出演者总电影数: ",totalmovies)
186  print("共同出演者总平均星级: ",round(totalstars/totalmovies,2))
187  print("共同出演者电影所属类别前3名: ",counter(totaltypes))
188  "周星驰共同出演者各自分别统计"
189  for x in company:
190      recordnum[x]=len(Movies[find[x]])
191      recordstar[x]=round(sum(Stars[find[x]]/len(Stars[find[x]]),2)
192      temp1=Types[find[x]]
193      temp2=[]
194      for i in range(len(temp1)):
195          for j in range(len(temp1[i])):
196              temp2.append(temp1[i][j])
197      recordtypes[x]=counter(temp2)
198  print("各自所演的电影数: ",recordnum)
199  print("各自所演电影的平均星级数: ",recordstar)
200  print("各自所演电影种类的前3名: ",recordtypes)

```

7.1 运行结果 1 (共同出演者总共的统计)

```
共同出演者人数 含周星驰本人 不去空串: 302
共同出演者总电影数: 3132
共同出演者总平均星级: 6.26
共同出演者电影所属类别前3名: [('动作', 1291), ('剧情', 1179), ('喜剧', 1054)]
```

7.2 运行结果 2 (共同出演者分别逐个统计)

(1) 下展示周星驰平均星级数, 共同出演者数目, 以及周星驰部分共同出演者们各自所演电影数:

注 1: 共同出演者各自所演电影数为部分截屏, 注意到有一个空串演员, 其来源是数据质量缺陷。

```
共同出演者人数 含周星驰本人 不去空串: 302
各自所演的电影数: {'梁家树': 1, '袁和平': 8, '张学友': 64, '甄子丹': 56, '江欣燕': 14, '杜琪峰': 2, '黄韵诗': 9, '林小楼': 9, '吴倩莲': 31, '恬妞': 26, '苑琼丹': 55, '徐克': 14, '彭丹': 11, '冯淬帆': 66, '关之琳': 49, '陈辉虹': 17, '李连杰': 47, '萧芳芳': 14, '梁荣忠': 12, '叶德嫻': 38, '郑佩佩': 62, '刘洵': 34, '朱咪咪': 13, '吴君如': 106, '汤镇业': 27, '郑家生': 1, '柏安妮': 15, '袁洁莹': 29, '林雪': 142, '黄美棋': 4, '张莽': 1, '谷峰': 117, '潘健君': 3, '方季韦': 1, '曾江': 41, '曾志伟': 171, '许英秀': 5, '葛民辉': 40, '廖启智': 44, '梅小惠': 8, '江希文': 8, '梁克逊': 2, '释彦能': 20, '林正英': 60, '赵薇': 27,
```

(2) 周星驰共同出演者们的平均星级数:

注 2: 以下为部分截屏内容

```
各自所演电影的平均星级数: {'曾志伟': 6.34, '': 7.26, '梁思浩': 7.45, '葛民辉': 6.74, '叶子楣': 6.42, '关海山': 6.4, '赵雷': 6.81, '谢贤': 6.76, '吴大维': 6.65, '王晶': 5.93, '黄百鸣': 6.64, '李连杰': 6.68, '龚嘉玲': 6.3, '郑祖': 7.2, '吴宇森': 7.5, '陈宝莲': 5.76, '沈殿霞': 6.7, '蔡少芬': 6.47, '陈松伶': 5.93, '姜大卫': 6.75, '李卉': 7.4, '林青霞': 6.82, '单立文': 6.52, '赵薇': 6.37, '林子祥': 6.79, '米奇': 6.08, '孙佳君': 6.61, '黄锦江': 6.76, '张学友': 6.88, '尹扬明': 6.55, '黄秋生': 6.44, '陈观泰': 6.42, '汪禹': 6.54, '袁信义': 6.88, '宣萱': 6.96, '江约诚': 8.9, '卢宛茵': 6.4, '田启文': 6.63, '历苏': 7.48, '张敏': 6.79, '赵志凌': 6.5, '黄一飞': 6.18, '杨宝玲': 6.6, '周志辉': 7.73, '梁家树': 6.9, '江欣燕': 6.66, '巩俐': 7.13, '陈欣健': 6.5, '莫美林': 7.4, '刘家辉': 6.04, '伍咏薇': 6.33, '梅艳芳': 7.0, '李嘉欣': 6.82, '周比利': 6.11, '梁克逊': 7.15, '张卫健': 6.22, '麦嘉': 6.83, '何佩仪': 6.5, '彭丹': 4.86, '吴启华': 6.24, '何超仪': 6.16, '元秋': 5.12, '林正英': 6.63, '杜琪峰': 7.0, '李丽珍': 6.55, '张雨绮':
```

(3) 周星驰共同出演者们的电影种类前 3 名:

注 3: 以下为部分截屏内容

```
各自所演电影种类的前3名: {'梁家树': [('喜剧', 1), ('动作', 1)], '袁和平': [('动作', 7), ('剧情', 3), ('喜剧', 3)], '张学友': [('喜剧', 32), ('动作', 28), ('剧情', 24)], '甄子丹': [('动作', 48), ('剧情', 20), ('喜剧', 13)], '江欣燕': [('喜剧', 10), ('动作', 5), ('恐怖', 4)], '杜琪峰': [('喜剧', 2), ('剧情', 1)], '黄韵诗': [('喜剧', 9), ('爱情', 3), ('动作', 2)], '林小楼': [('动作', 5), ('喜剧', 3), ('剧情', 2)], '吴倩莲': [('剧情', 16), ('爱情', 13), ('喜剧', 11)], '恬妞': [('动作', 11), ('剧情', 11), ('喜剧', 8)], '苑琼丹': [('喜剧', 35), ('爱情', 19), ('动作', 13)], '徐克': [('喜剧', 9), ('动作', 6), ('剧情', 4)], '彭丹': [('动作', 3), ('剧情', 3), ('犯罪', 2)], '冯淬帆': [('喜剧', 47), ('动作', 22),
```

8.自选报告 基于网络排序算法的演员影响力排名

8.1 自选题目与课内内容的联系与算法简介

在课堂上我们曾多次了解到图模型在互联网种有着重要的应用，而其中最具代表性的算法就是网络排序算法，是谷歌公司搜索引擎的“镇宅之宝”，可用于评估网页的影响力，其背后的思想其实抛开随机过程中的马氏收敛等比较深入的数学，其他的想法在课内基本都有体现，比如用 PageRank 因子作为一个类似评估函数的东西在 2048 对抗游戏中已有过具体应用。

$$PR(A) = (1 - d) + d(PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n))$$

公式 8.1 PR 评估因子（定义本质是递推的定义）

注 1 PR:节点的 PR 值 D:阻尼系数，即为用户浏览到 1 网页后的浏览概率 C(T):有向图出度，无向图的度

另外要综合评价一个节点的影响力，那势必会需要考虑它周围的连接情况和权重，于是定会应用到遍历算法，由此可见这种算法与课内的内容是有在应用和原理上多层次的联系的。

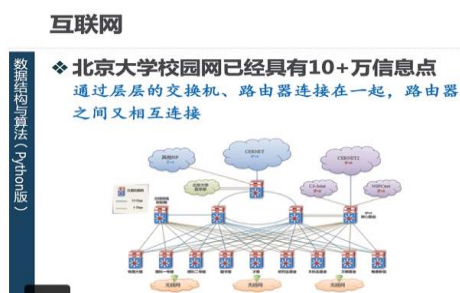


图 8.1.1



图 8.1.2

8.2 代码与算法分析

(1) main.py 第 178 行至第 187 行：调用 pagerank.py 内的评估函数和作图函数，给所有演员影响力一个排名，并画出前 5 名演员的相对位置子图：

```
174 """图算法发现"""
175 Edgelist=[]
176 for keys in EdgesValuesDic:
177     Edgelist.append([keys[0],keys[1],len(EdgesValuesDic[keys])])
178 import pagerank as p
179 h=p.pagerank(Edgelist)
180 with shelve.open('constants') as t:    #注36:存储排名结果
181     t["rank"]=h
182 with shelve.open('constants') as t:
183     h=t["rank"]
184 print(h)
185 g=p.setup(Edgelist)
186 for x in h[:5]:
187     p.show_rankfirst_subgraph(x[0],g)#注37:作出前5名演员的相对位置子图
```

5 更加详细的原理阐释见附录参考文献。

(2) `pagerank.py`: 定义评估函数和作图函数, 这里使用了 `networkx`、`warnings`、`matplotlib` 等包:

```

7 import networkx as nx
8 import matplotlib.pyplot as plt
9 import warnings
10 def setup(edgelist): #注38: 构建图的函数
11     G = nx.Graph()
12     for x in edgelist:
13         G.add_edge(x[0],x[1],weight=x[2])
14     return G
15 def pagerank(edgelist):
16     G = nx.Graph() #注39: pagerank评估演员影响力, 给出排名, 边的权重这里设为其附带电影列表的长度
17     for x in edgelist:
18         G.add_edge(x[0],x[1],weight=x[2])
19     fb=G
20     pos = nx.spring_layout(fb)
21     warnings.filterwarnings('ignore')
22
23     plt.style.use('fivethirtyeight')
24     plt.rcParams['figure.figsize'] = (20, 15)
25     plt.axis('off')
26     nx.draw_networkx(fb, pos, with_labels = False, node_size = 35)
27     plt.show()
28     pagerank = nx.pagerank(fb)
29     import operator
30     sorted_pagerank = sorted(pagerank.items(), key=operator.itemgetter(1),reverse=True)
31     print(sorted_pagerank)
32     return [sorted_pagerank,G]
33 def show_rankfirst_subgraph(s,G):
34     fb=G #注40: 实现作出排名靠前的演员的相对位置图功能的函数
35     first_degree_connected_nodes = list(fb.neighbors(s))
36     second_degree_connected_nodes = []
37     for x in first_degree_connected_nodes:
38         second_degree_connected_nodes+=list(fb.neighbors(x))
39     second_degree_connected_nodes.remove(s)
40     second_degree_connected_nodes = list(set(second_degree_connected_nodes))
41     subgraph_Mannheim= nx.subgraph(fb,first_degree_connected_nodes+second_degree_connected_nodes)
42     pos = nx.spring_layout(subgraph_Mannheim)
43     warnings.filterwarnings('ignore')
44     node_color = ['yellow' if v == s else 'red' for v in subgraph_Mannheim]
45     node_size = [1000 if v == s else 35 for v in subgraph_Mannheim]
46     plt.style.use('fivethirtyeight')

```

(3) `if __name__ == "__main__"` 下的代码: 是一个验证算法合理性和有效性的测试, 调用时不执行:

```

48     plt.axis('off')
49     nx.draw_networkx(subgraph_Mannheim, pos, with_labels = False, node_color=node_color,node_size=node_size)
50     plt.show()
51 if __name__ == "__main__":
52     edgelist = [['Mannheim', 'Frankfurt', 85], ['Mannheim', 'Karlsruhe', 80], ['Erfurt', 'Wurzburg', 75]]
53     G = nx.Graph()
54     for x in edgelist:
55         G.add_edge(x[0], x[1], weight=1)
56     fb=G
57     pagerank(edgelist)
58     show_rankfirst_subgraph("Mannheim",fb)

```

8.3 运行结果

8.3.1 演员影响力排名列表:

按照 PageRank 方法评估出的影响力最大的几位演员: 其中有曾志伟、成龙、林雪、刘德华、尼古拉斯凯奇、汤姆汉克斯、摩根弗里曼、塞缪尔杰克逊等著名演员, 搜索过他们所演出的电影后, 发现确实大致符合排名, 比如曾志伟确实比刘德华、成龙等演的电影在数目上多一些, 说明 Pagerank 算法应用在评估演员影响力上是合理的: ←

```
[('曾志伟', 0.0002855844671308413), ('林雪', 0.00025468293914414564), ('成龙', 0.000227126682065527), ('刘德华', 0.00022501109959239095), ('任达华', 0.00020724799914995432), ('黄秋生', 0.00020184373068607986), ('罗伯特·德尼罗', 0.00019717831637477612), ('塞缪尔·杰克逊', 0.0001903147642995772), ('吴君如', 0.00018220742340324827), ('迈克尔·凯恩', 0.00017928568216321757), ('梁家辉', 0.000168702699369888), ('洪金宝', 0.00016764042344328596), ('竹中直人', 0.0001673294486602648), ('热拉尔·德帕迪约', 0.00016701785290846994), ('尼古拉斯·凯奇', 0.00016675083143828694), ('詹姆斯·弗兰科', 0.00016643239045895217), ('威廉·达福', 0.0001611967302545424), ('连姆·尼森', 0.000155142696556792), ('约翰·赫特', 0.0001542968421515547), ('布鲁斯·威利斯', 0.00015336313963934038), ('摩根·弗里曼', 0.0001522130760601525), ('午马', 0.00014913834412677183), ('汤姆·汉克斯', 0.0001470829222936232), ('谷峰', 0.00014586881701459707),
```

■8.3.2 PageRank 因子密度分布图:

同时我们顺便输出全无向图模型的 PageRank 因子相对密度分布图:

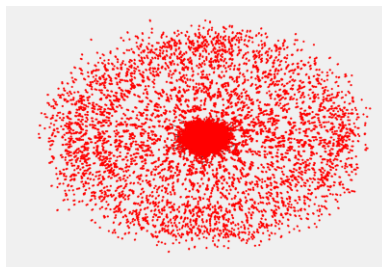


图 8.3.0 意义简析:

此图(图 8.3.0)为所有演员节点所构建的无向图的 PageRank 因子分布密度图,中心部分位置密度最高,与其他节点的连接关系最为密切,对应了最大的 8 万规模大小的连通分支。

图 8.3.0 PageRank 因子相对密度分布图

■8.3.3 最具影响力前 5 名演员位置子图:

下画出前 5 名演员的相对位置子图(黄点为演员在图中所在位置):

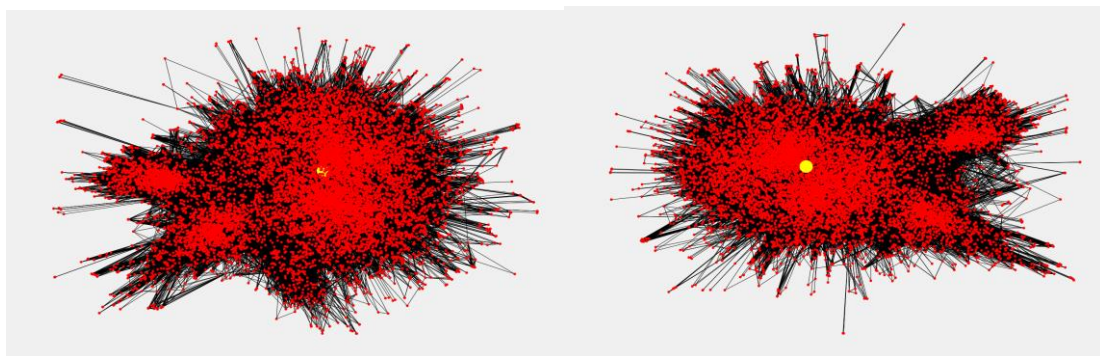


图 8.3.1 曾志伟

图 8.3.2 林雪

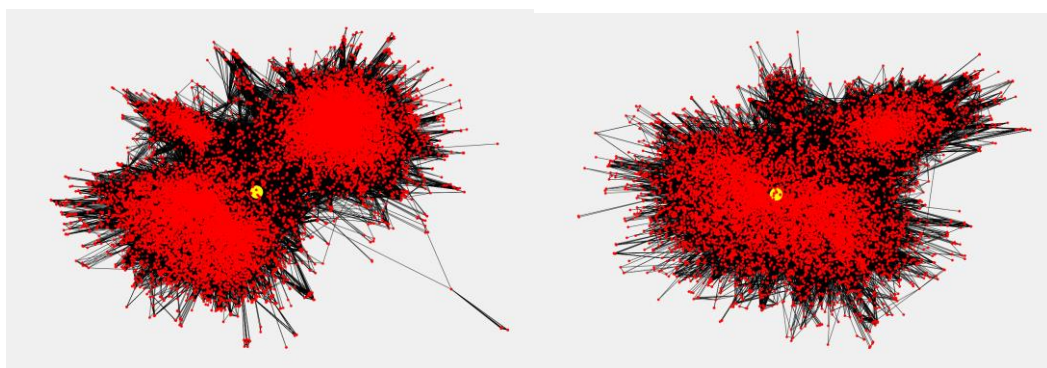


图 8.3.3 成龙

图 8.3.4 刘德华

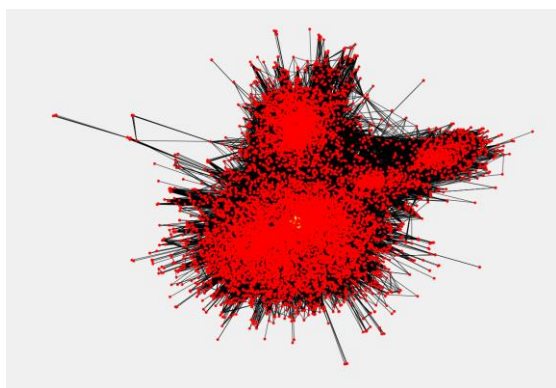


图 8.3.5 任达华

8.4 算法发现与分析

■从 PageRank 因子排名上看,我们可以发现,如果以合作电影列表的长度为边的权重,以演员为节点,当且仅当合作过的演员有边建立无向图模型:那么根据网排序算法得出的前 5 名最有影响力的演员确实基本符合他们所演电影数目的排名。这再次验证了这种排序思想的合理性。

■从直观上看,入选前 5 名的演员在相对位置子图中,也自然处于全图重心的位置。

■结合现实分析结果,可以领悟或验证一条做事的哲理:我们注意到有趣的是,成龙、刘德华等虽然在 PageRank 排名上没有前林雪、曾志伟高,但事实上,前者的知名度和获奖情况是远不逊于后两位的,而且前两者演艺的平均星级要高于比如曾志伟的 6.34,这告诉我们,电影作品、科研成果的质量也很重要,在现实中有时一部极好的作品或科研成果比许多一般的作品影响力还要大。

9.项目展望

- (1) 目前数据质量仍存在一定的问題,主要是演员姓名中有非“,”的符号,需要爬取更好的数据源。
- (2) 算法速度的再优化,特别是如何更加快速的求出最大规模的连通分支的直径。
- (3) 如何对演员排名的评估加入星级因素,即电影质量而不仅仅是数量,或可以体现在边的权重问題上,综合更多的因素,而不仅是共同出演的电影列表的长度。
- (4) 将此次用到的统计方法应用到其他感兴趣的统计对象上,例如不同学者间合作论文的情况统计。

10.算法各部运行时间简计

表格 10.1 算法各部运行时间简计

步骤名称	本地运行时间	时间单位
加载整理数据	checktime1 加载整理数据时间: 4.81824517250061	s
构建无向图	3.5938363075256348	s
统计连通分支数目、规模、前三电影种类	2.6054437160491943	s
统计连通分支直径	checktime4 求直径:0.18159270286560059	s
作出对数变换后的柱状图与截断折线图	105.16236615180969	s
周星驰演绎信息统计	0.3280518054962158	s
演员 pagerank 影响力排名	337.23	min
作出前 5 名演员相对位置子图	18.32	min

11.附录

- (1) PageRank 算法原理详细介绍: <https://blog.csdn.net/leadai/article/details/81230557>
- (2) PageRank 算法简介: <https://www.cnblogs.com/rubinorth/p/5799848.html>

