Web 信息处理与应用第二次实验-实验报告

王宇飞 PB14011018

本次实验分为两个部分：社区发现相关算法的实现和影响力最大化的相关内容。

如果需要运行代码，需要把 lab02-dataset 放在代码文件的根目录下，并且将 matlab_bgl-4.0.1 添加至 matlab 的搜索路径。

实验一—社区发现算法的实现与比较

实验内容

实现 spectral clustering 等几个社区发现算法，并比较实验结果。

实验环境

编程语言：matlab

编程环境：matlab 2015b

运行环境：windows10x64，i5-4200H@2.80GHz，8G Ram。

使用工具：matlab，gephi

所使用的第三方库：matlab_bgl_4.0.1

实验过程

1. 实现对应的社区发现算法，包括以下五个算法：
   a) alinkjaccard：该算法由于 matlab 内嵌了相关的实现函数，因此实现起来较为简单。首先利用 pdist 函数计算出每两个点之间的 jaccard 距离，然后再利用 linkage 函数根据 average link 规则计算社群之间的距离，合并对应的社群并且得到相应的层级聚类树，最后再用 cluster 函数对得到的聚类树进行分割最终的得到相应的聚类关系。pdist，linkage 和 cluster 都是 matlab 的内嵌函数。
   b) girvannewman：该算法首先需要计算每条边对应的 betweenness centrality，然后每次去掉 betweenness centrality 最大的边，之后计算图中连通片的个数，当图中连通片的个数小于 k 时不断重复上述的过程，直到图被分为 k 个连通片，即得到 k 个社群。该算法最大的难点是计算 betweenness centrality；如果在 matlab 里实现宽度优先搜索的话，即使是 n^2 时间复杂度的算法也会非常耗时，这会导致这个算法在 Egonet 上不可用。为了解决这个问题，我使用了 matlab_bgl_4.0.1 这个第三方库，其中提供的库函数 betweenness_centrality 调用了 C 实现、经过优化的 mex 函数，可以提高程序的执行效率，也简化了程序的编写流程。另外，matlab_bgl

提供了许多其它非常有用的函数，例如 components，该函数可以非常方便地得到某个结点所属的连通片，并且给出图中总的连通片数目。

c) rcut：该算法首先构造拉普拉斯矩阵 L = D-A 然后调用函数 eigs 求拉普拉斯矩阵 L 最小的 k 个特征值所对应的特征向量。之后将这 k 个特征向量排成 n*k 的矩阵，利用 kmeans 函数对矩阵进行聚类（k 维空间，每一行视作一个数据点）进行聚类即可得到分类的结果。kmeans 函数可以使用 matlab_bgl 中提供的相关函数。

d) ncut：该算法除了将上 rcut 中的拉普拉斯矩阵 L 替换为归一化的拉普拉斯矩阵 D^(-1/2) * (D – A) * D^(-1/2)之外，其余部分流程相同。

e) modularity：构造矩阵 $B = A - dd^T/2m$，其中 d 为度数列向量，m 为矩阵的总边数；然后，调用 eigs 函数求矩阵 B 的最大的 k 个特征值对应的特征向量。之后将这 k 个特征向量排成 n*k 的矩阵，利用 kmeans 函数对矩阵进行聚类（k 维空间，每一行视作一个数据点）进行聚类即可得到分类的结果。kmeans 函数可以使用 matlab_bgl 中提供的相关函数。

2. 实现主函数并进行测试：

a) 对于每个函数，为了测试的方便，我在每个函数的头部都加入了 if(nargin == 0)的语句，用于判断函数是否得到了正确的输入量。如果没有，则打开文件为输入量赋值，这样就可以直接在 matlab 中运行这个函数而不用另写一个脚本调用该函数，简化了测试流程。

b) 对于主函数，为了使主要部分的代码保持简洁，我将常用的操作都进行了包装，例如：

i. saveresult 函数：用于保存某一方法生成的 clustering 向量至文件；

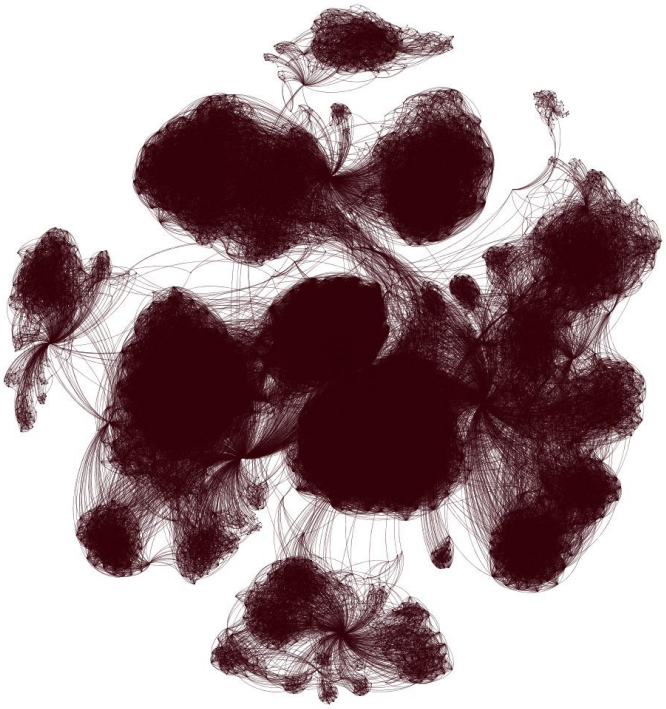
ii. callfunc：根据方法的名称调用对应的方法；

iii. visualization：用于生成在 Gephi 导入数据所用的 csv 文件。该函数会生成两个文件，分别代表边和结点；只有当 VIS = true 的时候，该函数才会运行。csv 文件是使用逗号分隔的纯文本文件，用于导入数据时，结点文件中需要有 Class 和 ID 项，边文件中则需要由 Source 和 Target 项（对应结点的 ID）；也可以根据需要增加其它项。

所有的结果都会被保存在./result 目录下。大部分方法的速度都非常快，但 girvannewman 方法在 Egonet 上需要非常长的时间才能得到结果（大概在三到四个小时左右）。

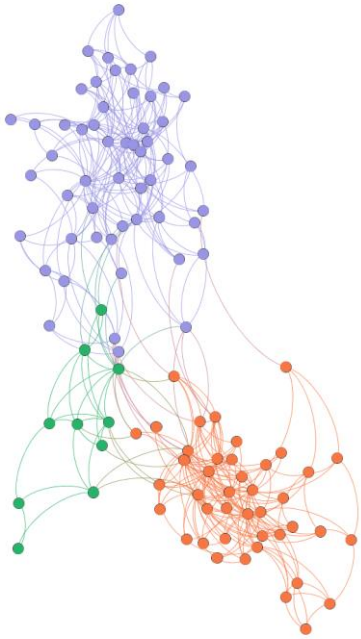
3. Egonet 中 k 的选择与可视化：

a) Egonet 中 k 选择可以有很多方法，例如可以在不同 k 的情况下分别运行 modularity 函数来判断 modularity 值的变化趋势，来决定最终使用哪个 k 值；也可以使用其它聚类评价标准来评价每个 k 的在同一方法上得到的结果来做出选择。但我使用的方法是直接将 Egonet 对应的边信息导入到 Gephi 中，利用 ForceAtlas 2 算法进行布局，可以得到如下的结果：

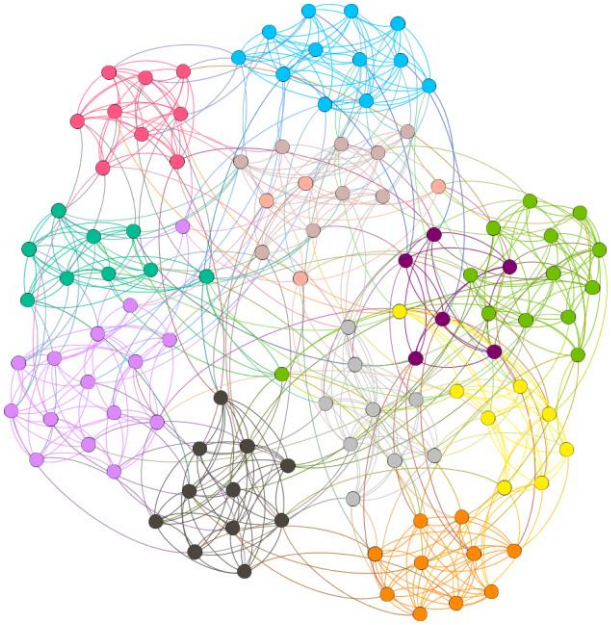


从图中可以看出，大致的聚类数量在 13~15 个左右。我最终选择了 k=14 来对 Egonet 进行聚类；实际结果和预期结果符合得很好，可参见下文的结果。

b) 将上文中得到的 csv 文件导入到 Gephi 中绘图以生成可视化结果，得到结果如下：



alinkjaccard on polbooks，Yifan Hu 布局算法



Girvannewman on football，ForceAtlas 2 布局算法

alinkjaccard on DBLP，ForceAtlas 2 布局算法

alinkjaccard on Egonet，ForceAtlas2 布局算法

可以看到分类算法和布局算法的结果是互相印证的。

实验结果与分析

利用助教提供的 evaluation 函数可以得到下表：

polbooks 与 football 数据集在五个算法下的 NMI 与 ACC 值						
数据集	算法	alinkjaccard	girvannewman	ncut	rcut	modularity
polbooks	NMI	0.4318	0.4388	0.4257	0.4851	0.1605
	ACC	0.7714	0.781	0.7619	0.8	0.5429
football	NMI	0.2633	0.2637	0.269	0.2633	0.2751
	ACC	0.1478	0.1478	0.1826	0.1304	0.1826

对于 polbooks，不同方法得到的 NMI 与 ACC 都还比较高（除了 modularity

的 NMI 明显偏低之外），但是对于 football 数据集，不同方法得到的 NMI 和 ACC 都很低。这可能是由于 groundtruth 本身和聚类算法的基本假设不相符引起的，譬如说在 alinkjaccard 算法中假设平均距离较近的类可能属于同一个社群，但是事实上可能并不是这样；这个问题在数据集本身比较小的时候会更加明显。

另外，我对数据进行了可视化，并且布局算法的结果和聚类得到的结果相一致。图片在上文中已经给出。

实验总结、问题与附加讨论

1. 五个算法中，girvannewman 的复杂度最高，其在 Egonet 上的运行时间超过了三个小时，主要是因为计算每条边的 betwenness centrality 是一个非常耗时的过程，并且这个操作需要重复进行直至数据集被正确地分类。并且，girvannewman 的结果和 alinkjaccard 较为相似，而 alinkjaccard 在 Egonet 上生成结果在一秒钟之内就能够完成。因此有理由认为，alinkjaccard 算法在一定程度上优于 girvannewman 算法。
2. 在大矩阵上计算特征值某些情况下会得不到收敛的结果，这时候 matlab 会显示 warning，并且 eigs 函数在稀疏矩阵和密集矩阵上的行为是不一样的。因此，我在程序中使用了一个 try-catch 表达式来尽量避免这种情况。
3. 关于附加讨论：上述的算法事实上存在一些问题，例如说不能够自行确定 k 值等。这个问题可以采用特定的评价标准来解决，即在同一数据集上使用不同的 k 运行得到结果并且选择评价最高的结果；但是普适的聚类评价方法不一定总是存在。在本是实验中，我使用了 Gephi 来生成可视化结果，并且注意到可视化的结果在一定程度上是可以和聚类算法本身向印证的（虽然布局算法的原理和聚类算法不完全相同）。那么，是否能够通过可视化的结果来确定聚类算法的分类效果，进而确定可行的评价标准？进一步地，是否能够根据布局算法的原理来进行聚类，或者是用其来辅助聚类过程的进行？当然这只是猜想，需要阅读文献以及进行更多的实验来验证。

实验二—影响力最大化

实验内容

利用 Degree Centrality、Closeness Centrality 等标准选取最具影响力的节点集合。将这些点作为种子节点，在独立级联模型(Independent Cascade Model)下进行模拟传播实验。比较不同方法选择种子节点的优劣。

实验环境

编程语言：matlab

编程环境：matlab 2015b

运行环境：windows10x64，i5-4200H@2.80GHz，8G Ram。

使用工具：matlab，gephi

所使用的第三方库：matlab_bgl_4.0.1

实验过程

1. 首先实现从输入文件中读取数据并且建图的函数。该函数首先需要从文件中读取第一行，获得图中节点的数目和边的数目，然后利用 fscanf 读取所有边的信息，并且根据信息构建对应的稀疏矩阵。
2. 实现种子节点生成的函数，包括如下四个方法：
   a) random：如果图中有 m 个节点，那么只需要在 1~m 内随机生成 k 个正整数返回即可。这里需要注意的是，生成的节点编号不能重复，比较方便的方法是使用 randperm 函数，然后取结果的前 k 项即可。
   b) degree_centrality：将输入的矩阵利用 logical 转化为二值矩阵，然后对每一行分别求和即可得到每个节点对应的度数，然后选择最大的 k 个即可。可以利用 sort 函数进行排序之后得到返回 index 的前 k 个作为结果；事实上也可以使用复杂度为 O(n)的算法选择前 k 个数，但是由于图中的节点比较少，所以直接使用 sort 是更加方便的选择。
   c) closeness_centrality：使用 all_shortest_paths 计算所有点对之间的最短距离，然后对距离矩阵求和并且根据规则计算 closeness_centrality 即可。这里需要注意的问题是，距离矩阵中如果某一结点到另一节点不可达，那么他们之间的距离是无穷大，这个结果将会导致节点计算出的 closeness centrality 为 0，这并不符合要求（如果图中有一个节点和所有节点之间都没有边，那么所有节点的 closeness centrality 都为 0）。解决这一问题较为简单的方法是将距离矩阵中的 inf 替换为一个足够大的值（例如矩阵中节点的数目），这样在一定程度上可以避免这个问题。求出各节点的 closeness centrality 之后同样利用 sort 即可得到前 k 大的值对应的节点 index，然后返回 index 即可。
   d) greedy：~~由于助教没有给出 f(S)要怎么计算，我给助教发了邮件询问也没有人回我~~，我简单地认为 f(S)为 S 中节点在 ICM 一轮传播中可以激活的最大节点数量，即 f(S) = S∪{S 的所有相邻节点}。对每个节点去模拟 ICM 传播过程来确定最大影响力集合并不可取，因为在本次实验中图较密且边的分布较为平均，基本在图中任取一个节点该节点的最大影响力集合就能覆盖超过图中 90%的节点，这会使得 greedy 算法的结果在 k>1 时失去意义。如果采用上述的定义方式，只需要在每次循环时将所有指向 S 和 S 邻接点的边都从图中去除，然后在子图中寻找度数最大的节点加入 S 中即可。这样定义的 f(S)是满足子模性要求的。

      Update：助教回了我邮件，表示计算 f(S)时应该将 S 作为 seed 来执行一次 ICM，结果集合中的结点数就是 f(S)的大小。这种方法时间复杂度非常高，并且由于上述分析的原因，基本上在图中随便选一个节点它的影响力集合都会覆盖图中超过 9/10 的节点，在这种情况下 greedy 算法的结果基本上就是完全随机的（相比于随机算法有更大可能选到没什么边连接的节点）。而且，f(S∪{u})的结果可能会比 f(S)更小，这也给实现带来了麻烦。不过我仍然按照规定实现了 ICM 算法，同时将我之前的函数命名为 greedy_mine 保存在 greedy.m 文件中供参考。

3. 实现 ICM 传播函数，算法的具体流程如下：
   a) 将 N_nxt 集合初始化为 seed，表示下一轮中将要执行激活操作的节点；
   b) 对于 N_nxt 中的每一个节点，依次产生随机数并判断其相邻节点是否应该在本轮传播中被激活；若满足激活条件，且该相邻节点没有被访问过，则将该节点加入下一轮需要执行操作节点的列表 Q 中；
   c) 将 N_nxt 中的所有节点加入全部激活节点列表 N 中，并且令 N_nxt = Q，返回第二步，直至算法完成了 T 轮的传播。

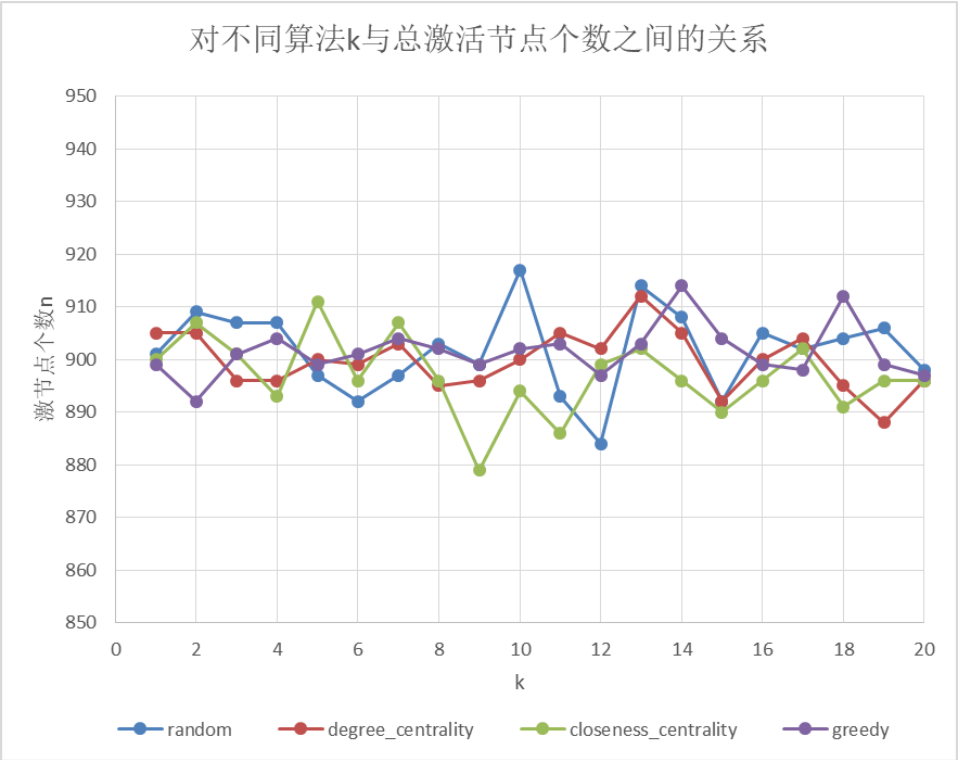
   我实现的 ICM 传播函数在主函数内。
4. 构建主函数并完成测试。对于每个方法产生的 seed 集合，分别调用 ICM 算法得到结果，并且按照要求输出即可。

实验结果与分析

实验得到的结果将会被保存在./results 文件夹下。各个算法得到的总激活节点数和 k 的关系如下：

k	random	degree centrality	closeness centrality	greedy
1	901	905	900	899
2	909	905	907	892
3	907	896	901	901
4	907	896	893	904
5	897	900	911	899
6	892	899	896	901
7	897	903	907	904
8	903	895	896	902
9	899	896	879	899
10	917	900	894	902
11	893	905	886	903
12	884	902	899	897
13	914	912	902	903
14	908	905	896	914
15	892	892	890	904
16	905	900	896	899
17	902	904	902	898
18	904	895	891	912
19	906	888	896	899
20	898	896	896	897

作图可以得到：



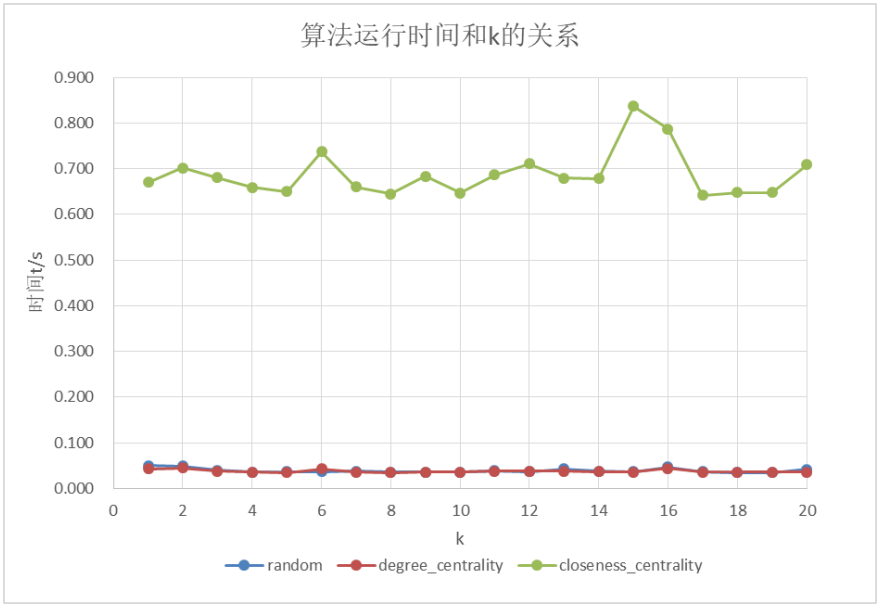
从图中可以看出，基本上不同的 seeds 和 k 产生的结果没有什么不同，原因在上文 greedy 算法的分析中有提到，主要是数据集的问题。我询问过助教，助教表示**他们在批改实验报告的时候会注意这个问题**，所以我这里就不再对算法进行进一步的性能分析了。

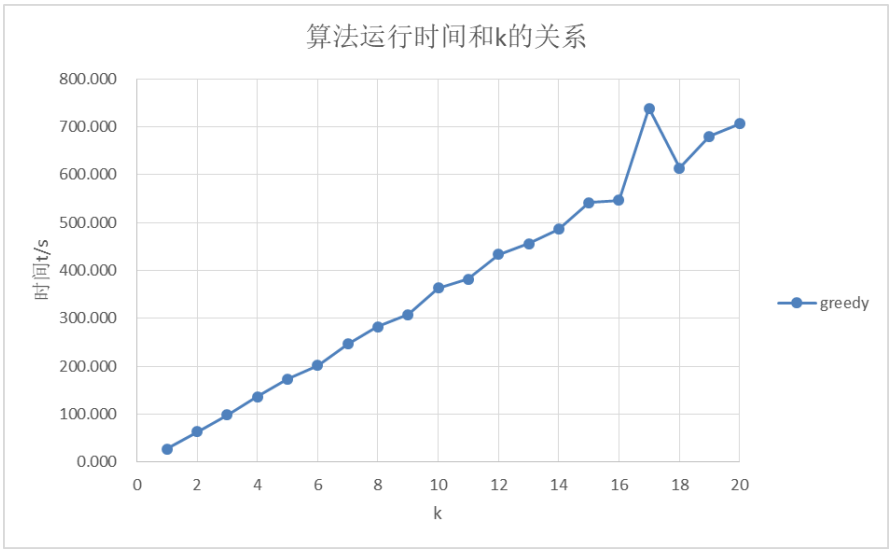
各个算法计算 seeds 所花费的时间如下：

k	random	degree centrality	closeness centrality	greedy
1	0.050	0.043	0.670	27.556
2	0.049	0.045	0.701	63.028
3	0.040	0.038	0.680	98.560
4	0.036	0.036	0.659	136.201
5	0.037	0.035	0.650	173.781
6	0.037	0.043	0.738	201.693
7	0.038	0.036	0.660	246.555
8	0.036	0.035	0.645	282.779
9	0.036	0.035	0.684	307.921
10	0.035	0.036	0.647	363.231
11	0.039	0.038	0.686	382.142
12	0.037	0.038	0.710	433.695
13	0.043	0.038	0.680	455.786
14	0.038	0.037	0.679	486.382
15	0.037	0.036	0.837	541.762
16	0.047	0.044	0.787	546.934
17	0.037	0.036	0.642	738.489

18	0.035	0.035	0.648	614.442
19	0.035	0.036	0.648	680.006
20	0.042	0.036	0.709	706.163

作图可以得到：





事实上，通过分析，random 算法的时间复杂度是 O(k)，degree_centrality 为 O(n)（考虑稀疏图，每个节点连接的边有限），closeness_centrality 的是时间复杂度取决于计算最短路使用的算法和数据结构，大致是 O(n*E)或者 O(n^3)的；greedy 若采用规定的实现，则复杂度是 O(k*V*(E+V))的，但考虑到 ICM 不方便向量化，这一部分会非常慢。上述结果基本上和理论分析是符合的。

实验总结

各个算法本身的实现较为简单，试验中遇到的大部分问题都是数据集导致的。助教所提供的数据集应该是随机生成的，之后的试验中可以考虑更换为从实际场景中获取的数据集。