《大数据系统综合实践》任务书

【实验目的】

本实验旨在通过大规模图数据中社区发现算法的设计与性能优化，帮助学生深入理解图计算系统的工作原理和性能优化机制，并学会使用图计算框架进行大规模图数据分析和处理。通过此实验，学生将能够掌握图计算的基本原理，编写比较复杂的图算法程序并进行性能调优。

【实验内容】大规模图数据中社区发现算法的设计与性能优化

一、基本概念

社区发现是大数据社交网络分析中的一个重要领域，它旨在识别网络中具有紧密连接或共享相似特征的节点集合，这些集合被称为“社区”。 如果你仔细观察，你会发现，我们的生活中存在着各种各样的网络，如科研合作网络、演员合作网络、城市交通网络、电力网、以及像 QQ、微博、微信这样的社交网络。

社交网络的核心是参与其中的用户以及用户之间的关系。因此，我们可以采用图模型来为其进行建模，其中的节点表示社交网络中一个个的用户，而边则表示用户与用户之间的关系，如果想对这些关系强度（或亲密度）进行区分的话，我们还可以为每条边赋予一个权重，权值越大表示关系强度越大（或者越亲密）。

如图 1 所示，仔细看就会发现这个图包含一定的结构：其中存在一个一个的节点子集合，在这些子集合的内部边比较多，而子集合与子集合之间边比较少。具体将这些节点子集合圈出来，就会有 3 个内部联系非常紧密的社区。

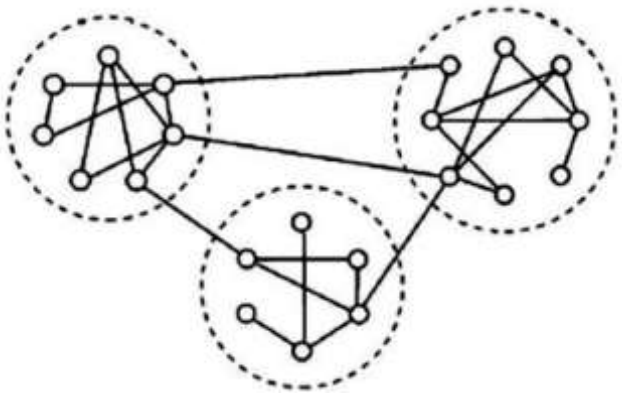

图 1 非重叠型（disjoint）社区

这其实跟我们生活中的一些场景是吻合的。比如，考虑一个创意园区里各员工的认识关系，你会发现一个公司就会对应一个节点子集合，因为同一个公司的人大部分彼此相互认识，而公司与公司之间的人则相互认识的不多。注意，刚才举的例子中，每一个节点只能属于一个子集合，因为每一个员工只属于一家公司，因此各个节点子集合是互不重叠的。但是，在某些场景下，节点子集合之间可能发生重叠，即同一个节点可能同时属于多个子集合。相应的图结构就变成图 2 这个样子了，中间那些涂成彩色的节点表示同时参与了多个社区的关键节点。

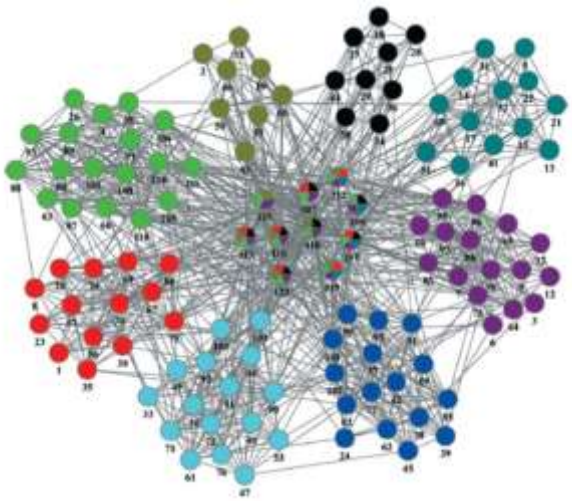

图 2 重叠型（overlapping）社区

那些内部连接比较紧密的节点子集合对应的子图叫做社区（community），各社区节点集合彼此没有交集的称为非重叠型（disjoint）社区，有交集的称为重叠型（overlapping）社区。网络图中包含一个个社区的现象称为社区结构，社区结构是网络中的一个普遍特征。给定一个网络图，找出其社区结构的过程叫做社区发现（community detection）。

直观地说，community detection 的一般目标是要探测网络中的“块”cluster 或是“社团”community。社区划分问题大多基于这样一个假设：同一社区内部的节点连接较为紧密，社区之间的节点连接较为稀疏。因此，社区发现本质上就是网络中结构紧密的节点的聚类。从这个角度来说，这跟聚类算法一样，社区划分问题主要有两种思路：

（1）凝聚方法(agglomerative method)：添加边

（2）分裂方法(divisive method)：移除边

（3）相似性检测方法：同一个社区内的节点，之所以能够聚集在一起，是因为它们有相似性。因此只要我们能够将一个节点很好地表示，成为一个向量，那么同样可以用相似性大法来寻找社区聚集，不过这点上需要向量对节点的描述足够好和足够完备。

下面介绍几个基本概念：

1. 点度中心性：节点的度，其意义在于评价该点单独的价值。

2. 紧密中心度（clossness centrality）

公式：$\frac{|V|-1}{\sum_{i=v} d_{vi}}$ 某点到其他各点距离之和的平均值的倒数。

意义：某点到达其他点的难易程度，越大说明越在中心，越能更快到达其他

节点。

3. 节点介数中心性（betweenness centrality）

公式：$\sum_{i \neq j \neq v} \frac{g_{ivj}}{g_{ij}}$

g_{ij} 为节点 i 到 j 的最短路径数量，g_{ivj} 为上面得到的 g_{ij} 中过节点 v 的路径数。

意义：表明该点在网络中的重要程度，判断是不是一个交通枢纽。

4. 边介数(betweenness)：

网络中任意两个节点通过此边的最短路径的数目。

二、典型社区发现算法

GN 算法是社区发现领域的第一个算法，或者说它开启了这个领域的研究。下面我们来分别介绍这个领域及其算法是如何演化的。

1、GN 算法（Girvan Newman）

GN 算法基本思想：在一个网络之中，通过社区内部边的最短路径相对较少（边介数小），而通过社区之间边的最短路径的数目则相对较多（边介数大）。GN 算法是一个基于删除边的算法，本质是基于聚类中的分裂思想，在原理上是使用边介数作为相似度的度量方法。把一个大的网络，以边介数为衡量，每次都选择边介数高的边删去，以实现网络分裂。

算法过程如下：

（1）计算每一条边的边介数；

（2）删除边界数最大的边；

（3）重新计算网络中剩下边的边阶数;

（4）重复(3)和(4)步骤，直到网络中的任一顶点作为一个社区为止。

这种方法的缺点：（1）计算复杂度高，在每次删除边了之后都会重复计算剩余节点的边介数；（2）算法的终止条件缺乏一个衡量的目标，不能控制最后可以分裂成多少个社区;如图 3 所示,GN 算法相当于是一棵自顶向下的层次树,划分社区就是层次分裂的过程。

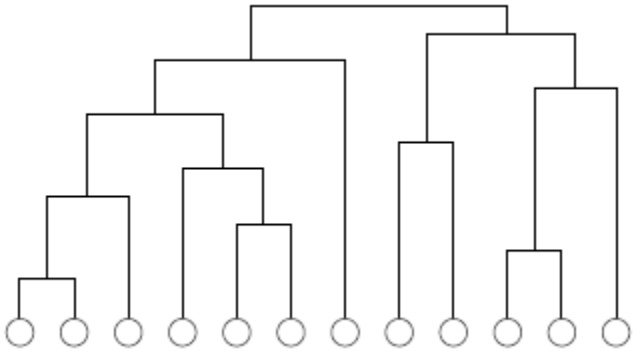

图 3　层次分裂过程

解决的办法:引入模块度 Q,模块度是用来衡量社区划分好坏的程度的概念。整个算法的思想反过来了，不再是从顶层分裂，而是从底层合并聚类，直到最终形成一个大的网络。每次是根据计算合并后使得模块度 Q 的增量最大的社区进行合并，直到收敛。也就是说，是基于增加边而不是删除边了。这种引入模块度 Q 来度量社区划分质量的思想，有点像梯度下降算法，是通过迭代计算来获得定义的目标函数的最优解的。

举例如下:

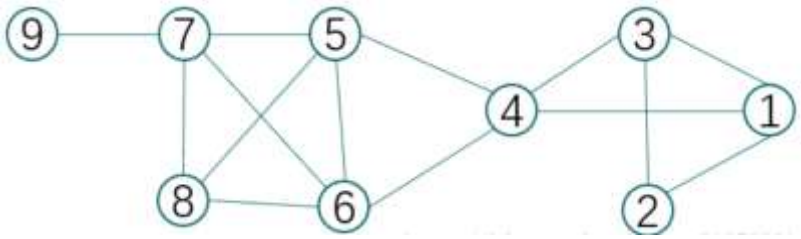

图 4　GN 算法示例

第一步：从节点 s 开始，执行一个广度优先搜索，确定一个节点的最短路径数量；

第二步：计算边介数，每个节点都要进行一遍广度优先搜索，计算边介数，也就是说上面的一、二两步要执行九次。然后将这九次所有点介数加起来除以 2，得到最终点介数。同样边介数的计算也一样，将九个二叉树的边介数加起来除以 2 得到最终边介数。如图 6，红色数字为边介数，黄色数字为节点介数。

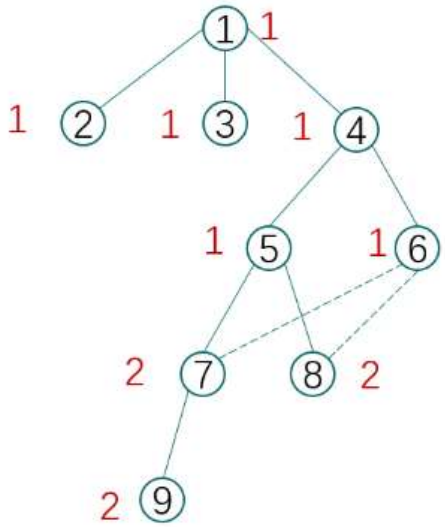

图 5 广度优先搜索

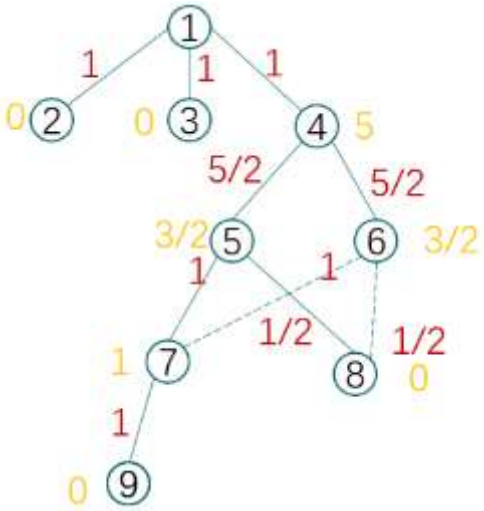

图 6 计算边介数

	1	2	3	4	5	6	7	8	9
1	0	4	1	9	0	0	0	0	0
2	4	0	4	0	0	0	0	0	0
3	1	4	0	9	0	0	0	0	0
4	9	0	9	0	10	10	0	0	0
5	0	0	0	10	0	1	6	3	0
6	0	0	0	10	1	0	6	3	0
7	0	0	0	0	6	6	0	2	8
8	0	0	0	0	3	3	2	0	0
9	0	0	0	0	0	0	8	0	0

图 7 边介数

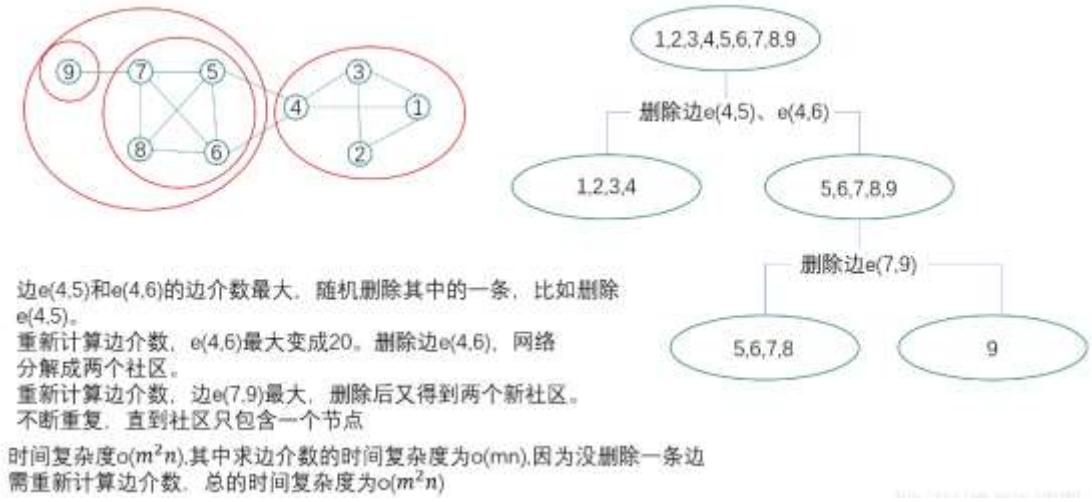

图 8 计算流程示例

【实验要求】

本实验要求在给定服务器平台，以及数据集上实现社区发现算法，调试并获得最高的性能。在 Linux 环境下，采用 Spark GraphX 或 Pregel 或其它框架设计社区发现算法。所开发的算法能够充分利用多核资源，以完成给定格式的图数据中三角形的计数。开发环境如下：

- 操作系统：Linux ubuntu 14.04 或 16.04
- 编译器：gcc 或者 g++ 4.8 以上
- Make：GNU make 4.0 以上
- 框架：Spark GraphX、Pregel、或你所熟悉擅长的其它框架

数据集	说明
中等规模数据集 soc-LiveJournal1	顶点数：4.8 million、边数：69 million 来源：https://snap.stanford.edu/data/soc-LiveJournal1.html

小数据集 cit-HepPh	顶点数：34546，边数：421578 来源：https://snap.stanford.edu/data/cit-HepPh.html

1）本实验有功能和性能两方面的要求，首先需要能够正确地统计出所给数据集中的所有社区，其次需要不断优化你所设计的算法（并行优化、存储优化、通信优化等），使得在最终的测试机器上执行时间最短，最终的成绩和你的算法优化效果密切相关。(注：完成本阶段即可得 85 分。)

2）在 GPU+CPU 环境下，通过 GPU 加速来实现社区发现算法。(注：完成本阶段加 15 分。)

实验完成后，需要撰写一份详细描述你的算法实现和优化机制的文档，以班级为单位于 10 月 31 日前交东五楼 206 房间。