



本科毕业论文（设计）

（主修专业）

基于 Spark 图计算的社会网络分析系统的设计 和实现——顶点分析

Design and Implementation of Social Network Analysis

System based on the Graph Computation Engine of Spark

——Vertex Analysis

姓 名：陈黎栋

学 号：30920122202444

学 院：软件学院

系：软件工程

专 业：软件工程

年 级：2012 级

校内指导教师：毛波 副教授

校外指导老师：岳银亮 副研究员

二〇一六 年 五 月 二十 日

厦门大学本科学位论文诚信承诺书

本人呈交的学位论文是在导师指导下独立完成的研究成果。本人在论文写作中参考其他个人或集体已经发表的研究成果，均在文中以适当方式明确标明，并符合相关法律法规及《厦门大学本科毕业论文规范》。

另外，该学位论文为()课题(组)的研究成果，获得()课题(组)经费或实验室的资助，在()实验室完成。(请在以上括号内填写课题或课题组负责人或实验室名称，未有此项声明内容的，可以不作特别声明。)

学生声明(签名):

年 月 日

该同学呈交的学位论文是在本人指导下独立完成的研究成果。本人已经对学生毕业论文内容进行严格审核，论文写作中参考其他个人或集体已经发表的研究成果，均在文中以适当方式明确标明，并符合相关法律法规及《厦门大学本科毕业论文规范》。

指导教师声明(签名):

年 月 日

致 谢

在毕业设计的整个过程中,我的指导老师毛波老师和岳银亮老师一直不厌其烦的回答我的问题,把他们科研实践的宝贵经验传授给我。他们对待科研极其严谨和朴实,哪怕是毕业设计论文中的每一处标点,他们都仔仔细细的为我检查,使我获益良多。在此,首先向毛波老师和岳银亮老师致以最诚挚的感谢!

毕业设计论文是对本科所学知识的综合应用,因为老师们用心上课,我才能学到许许多多书本及书本以外的东西。感谢教授我编程基础、数据结构和算法的吴清锋、苏劲松、陈海山等老师,尤其感谢苏老师积极引导学生走上科研之路,在机器学习领域执着追求;感谢教授我数据库、操作系统和计算机网络等核心课程的张仲楠、吴清强、黄炜、林坤辉等老师,清强老师清晰的逻辑和仲楠老师的严厉给我留下了深刻的印象,黄炜老师不顾自己利益、真心为学生着想的心意和林坤辉老师严谨的治学态度值得我一生效仿和学习;感谢教授我现代软件工程规范的邱明和王备战老师,真心感谢邱老师模拟真实的软件开发给我们好好上了一课,王老师课上的那些心灵鸡汤培养了我积极和正面的人生态度;感谢教授我选修课程的张海英、苏淑文、赖永炫、高星、洪清启、洪志令、史亮等老师,人数众多,如有遗漏,万勿责怪。

此外,我要感谢我的本科生导师李贵林老师和赵江生、吴克青、陈昉等工程师老师,贵林老师每学期都主动和我进行思想交流,帮助我进步,学院的工程师老师们为我们提供了良好的硬件资源,保障了我们的学习效率。

“自强不息,止于至善”,很幸运能够在“南方之强”厦门大学度过美好的四年,在这期间,我和舍友们相互扶持和包容,最终一起获得了保送研究生资格。

《哈佛幸福课》指出“学术论文的平均读者是7个人,其中包括作者的妈妈”。最后,感谢我的母亲,是你操持这个家,让我可以安心读书学习,感谢我的女朋友,无论我遇到任何事,都给我最大的精神支持,感谢你们,你们是我每一篇文章的忠实读者。

摘 要

随着 SNS (Social Networking Sites, 社交网站) 如 Renren、 Facebook 等的快速发展, SNA (Social Network Analysis, 社会网络分析) 逐渐成为研究的重点。现代 SN (Social Network, 社会网络) 往往都是几百万甚至上千万的超大规模数据集, 因此如何处理大规模的社会网络数据集成为社会网络分析面临的一个较为严峻的挑战。

目前面向海量数据的社会网络分析工具主要基于 Hadoop MapReduce 设计, 如 X-RIME。新兴的 Spark 拥有 Hadoop MapReduce 所具有的优点, 但不同于 MapReduce 的是 Job 中间输出结果可以保存在内存中, 从而不再需要读写 HDFS, 因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的算法。典型的社会网络模型由顶点和边组成, 边的属性依赖于顶点的属性, 因此, 顶点分析是社会网络分析的基础。

本文在 Spark 及图计算引擎 GraphX 的基础上, 设计并实现一套用于顶点分析的社交网络分析系统, 为使用 Spark 进行大规模社交网络的顶点分析提供具体接口, 包括社交网络的图模型构建、顶点重要度估值、顶点间路径的计算、顶点分组等接口, 并对计算结果进行了一定程度上的可视化展现。主要工作包括两个方面: 第一是构建图和数据可视化的实现; 第二是具体的顶点分析接口的实现。

论文从项目背景、系统相关技术、系统功能性需求分析、系统概要设计、系统详细设计和实现、系统功能性测试和总结与展望等部分具体展开。

关键词: 社会网络分析; 图计算; GraphX; 顶点分析

Abstract

With the fast development of Social Networking Sites (SNS) such as Renren, Facebook, etc. Social Network Analysis (SNA) is becoming a hot research area. However, considering a typical SNS consists of millions to tens of millions of ultra-large-scale data sets, how to deal with such large-scale Social Network (SN) data sets becomes a great challenge for SNA.

Designs of Ultra-large-scale SNA tools, such as X-RIME, are mainly based on Hadoop Map Reduce currently. The latest developing technology of Spark not only enjoys the advantages of Hadoop, but also can save the intermediate output results of Job in memory, eliminating the process of reading and writing for HDFS. Therefore, Spark is better suited for algorithms, which need iteration, like Data Mining and Machine Learning. A typical SN model consists of vertexes and edges, and edge attributes depend on vertex attributes. Thus, Vertex Analysis is the base of SNA.

Towards the above statement, this paper designs and implements a social network analysis system, which is based on Spark and its graph compute engine, for vertex analysis. The system provides concrete interfaces to conduct large-scale social network vertex analysis by Spark, including interfaces for building graph models, evaluating importance of vertexes, calculating simple routes between vertexes and clustering vertexes, and visualize the results of analysis to some degree.

This paper consists of the project background, the system-related technologies, the system functional requirements analysis, the summary design of system, the detailed design and implementation of system, the system functional test and Summary and Prospect.

Key words: Social Network Analysis; Graph Compute; GraphX; Vertex Analysis

目 录

第一章 绪论	1
1.1 国内外研究现状	1
1.2 论文研究内容	1
1.3 论文组织结构	2
第二章 系统相关技术	5
2.1 通用并行框架 Spark	5
2.2 图计算框架 GraphX	6
2.3 动态图形组件 GraphStream	6
2.4 函数式编程语言 Scala	6
2.5 本章小结	7
第三章 系统功能性需求分析	9
3.1 系统功能模块	9
3.2 系统用例描述	9
3.3 本章小结	14
第四章 系统概要设计	15
4.1 系统架构	15
4.2 类设计	16
4.2.1 算子子系统	16
4.2.2 可视化对象构建	17
4.2.3 分析演示子系统	18
4.3 功能函数设计	19
4.3.1 图构建	19
4.3.2 整体分析	20
4.3.3 局部分析	21
4.3.4 可视化	22
4.4 本章小结	23
第五章 系统设计与实现	25
5.1 图构建模块	25
5.2 可视化模块	28

5.3 整体分析模块	29
5.3.1 顶点重要程度分析.....	29
5.3.2 顶点分组.....	30
5.4 局部分析模块	31
5.4.1 顶点邻居计算.....	31
5.4.2 顶点到顶点路径计算.....	33
5.5 本章小结	34
第六章 系统测试.....	35
6.1 测试数据	35
6.2 测试环境	35
6.3 图构建和可视化测试	36
6.4 分析模块测试	38
6.4.1 顶点重要程度分析.....	38
6.4.2 顶点分组.....	40
6.4.3 顶点邻居计算.....	42
6.4.4 顶点到顶点的简单路径.....	43
6.5 本章小结	44
第七章 总结与展望.....	45
7.1 论文总结	45
7.2 工作展望	45
参考文献	47

Contents

Chapter 1 Preface	1
1.1 Research Status at Home and Abroad	1
1.2 Research Contents	1
1.3 Structure Arrangements.....	2
Chapter 2 System Related Technologies	5
2.1 General Parallel Framework Spark	5
2.2 Graph Compute Framework Graphx.....	4
2.3 Dynamic Graphic Component Graphstream.....	6
2.4 Functional Programming Language Scala.....	6
2.5 Summary	7
Chapter 3 System Functional Requirements Analysis.....	9
3.1 System Function Modules	9
3.2 Use Case Description	9
3.3 Summary	14
Chapter 4 Summary Design	15
4.1 System Architecture.....	15
4.2 Class Design.....	16
4.2.1 Arithmetic Operators Subsystem	16
4.2.2 Visualization Object Building	17
4.2.3 Analysis and Presentation Subsystem.....	18
4.3 Function Design	19
4.3.1 Building Graphs.....	19
4.3.2 Global Analysis.....	20
4.3.3 Partial Analysis	21
4.3.4 Visualization	22
4.4 Summary	23
Chapter 5 Detailed Design And Implementation.....	25
5.1 Graph-Building Module.....	25
5.2 Visualization Module.....	28
5.3 Global Analysis Module	29
5.3.1 Vertex Importance Evaluation	29
5.3.2 Vertex Cluster	30

5.4	Partial Analysis	31
5.4.1	Neighbors Calculation	31
5.4.2	Simple Routes Calculation	33
5.5	Summary	34
Chapter 6	System Functional Test	35
6.1	Test Data Description	35
6.2	Test Environment Description	35
6.3	Test Of Graph-Building And Visualization	36
6.4	Test Of The Analysis Module	38
6.4.1	Vertex Importance Evaluation	38
6.4.2	Vertex Cluster	40
6.4.3	Neighbors Calculation	42
6.4.4	Simple Routes Calculation	43
6.5	Summary	44
Chapter 7	Conclusions And Feature Works	45
7.1	Conclusions.....	45
7.2	Future Works	45
References	47

第一章 绪论

1.1 国内外研究现状

有关“图论”在“社会（交）网络”^[1]上的应用，国外的论文从类型上主要分为系统工程级应用、图模型建设和算法研究三个方面，从社会网络的规模上可分为对传统社会网络的研究和面向海量数据的社会网络的研究两个方面。系统工程级应用、图模型建设和算法研究三个方面的论文发表数量依次增多，面向海量数据的社会网络的研究的相关论文的数量明显少于传统社会网络的研究的相关论文数量。面向海量数据的社会网络的系统工程级应用的最新论文是 2016 年 3 月发表的《一个用于大规模社交网络服务应用程序的高效图数据处理系统》（《An efficient graph data processing system for large-scale social network service applications》）^[2]。

国内关于“图论”在“社会（交）网络”上的应用的论文数量相对较少，其中比较典型的有：复旦大学的吴文涛在 2010 年的硕士论文中基于图对称理论对社会网络中的实体匿名问题，最短路径查询问题，以及社团挖掘问题进行了深入研究；国防科技大学的刘亮在 2013 年的硕士论文中研究了一种能够同时综合网络拓扑结构和节点属性的统计网络模型——指数随机图模型(ERGM)，并且基于该模型构建人工社会中社会网络。其余的研究集中在“隐私保护”方面，2011 年兰丽辉等人在《小型微型计算机系统》提出用二分图实现社会网络的匿名发布的方案，2013 年东北大学的杨俊在硕士论文中提出了基于图自同构的 AK-Secure 社会网络隐私保护方法^[3]。

1.2 论文研究内容

Social Network 中文译为“社交网络”或“社会网络”，在 Web2.0 技术兴起后发展势头强劲，有着巨大的用户群。其实时产生的庞大信息量具有 4V 特性（Volume, Variety, Velocity, Veracity），是典型的大数据^{[4][5]}。

社会网络分析技术是一项关键技术，是一项多学科交叉技术，也是一项热门的研究^[6]。目前大数据规模的社会网络分析工具主要基于 Hadoop MapReduce 设计^[7]，如 X-RIME^[8]。新兴的 Spark^[9]拥有 Hadoop MapReduce 所具有的优点，但不同于 MapReduce 的是 Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的算法^[10]。典型的社交网络模型由顶点和边组成，边的属性依赖于顶点的属性，因此，顶点分析是社会网络分析的基础。

关于 Spark 图计算引擎 GraphX^{[11][12][13]}的具体应用有资料可查的仅有淘宝的图谱体检平台^[14]，该平台通过对以下指标进行检查：度分布、二跳邻居数、连通图，来对产品和运营的构思进行数据上的预研指导。

本文在 Spark 及图计算引擎 GraphX 的基础上，设计并实现一套用于顶点分析的社交网络分析系统，为使用 Spark 进行大规模社交网络的顶点分析提供具体接口，具体包括社交网络的图模型构建、顶点重要度估值、顶点间路径的计算、顶点分组等接口，并对计算结果进行了一定程度上的可视化。主要工作包括两个方面：第一是图构建和数据可视化的实现；第二是具体的顶点分析接口的实现。

1.3 论文组织结构

本论文从“基于 Spark 图计算的社会网络分析系统”的项目背景、关键技术、系统的设计与实现、测试以及系统的展望等方面进行阐述，共分为七个章节。

第一章为绪论，首先介绍项目的背景以及研究意义，其次概括性的介绍了项目的主要研究内容。

第二章为系统相关技术的概述，主要介绍了大数据技术以及图计算技术，并简要说明了为什么使用 Spark 的图计算引擎。

第三章为系统的功能性需求分析，通过功能模块总图和用例描述等对系统功能进行了介绍。

第四章和第五章为系统的设计与实现，首先阐述了系统的整体架构，然后对系统的关键类和函数进行了宏观设计，之后再分模块具体介绍了各部分的实现，并从程序设计的角度进行了详细的介绍。

第六章为系统的测试与结果说明,对系统进行了深入的测试,对系统的规模、效率进行分析。

第七章为总结与展望,对全文进行了总结,并对“基于 Spark 图计算的社会网络分析系统”的发展前景进行了展望。

第二章 系统相关技术

2.1 通用并行框架 Spark

Spark 是由加州大学伯克利分校 AMP 实验室所开源的类 Hadoop MapReduce 的通用并行框架，Spark 拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是 Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。

Spark 是一种与 Hadoop 相似的开源集群计算环境，但是两者之间还存在一些不同之处，这些有用的不同之处使 Spark 在某些工作负载方面表现得更加优越，换句话说，Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。

Spark 是在 Scala 语言中实现的，它将 Scala 用作其应用程序框架。与 Hadoop 不同，Spark 和 Scala 能够紧密集成，其中的 Scala 可以像操作本地集合对象一样轻松地操作分布式数据集。

尽管创建 Spark 是为了支持分布式数据集上的迭代作业，但是实际上它是对 Hadoop 的补充，可以在 Hadoop 文件系统中并行运行，通过名为 Mesos 的第三方集群框架可以支持此行为。

RDD (Resilient Distributed Datasets)，弹性分布式数据集，是分布式内存的一个抽象概念，RDD 提供了一种高度受限的共享内存模型，即 RDD 是只读的记录分区的集合，只能通过对其他 RDD 执行确定的转换操作(如 map、join 和 group by) 而创建，然而这些限制使得实现容错的开销很低。对开发者而言，RDD 可以看作是 Spark 的一个对象，它本身运行于内存中，如读文件是一个 RDD，对文件计算是一个 RDD，结果集也是一个 RDD，不同的分片、数据之间的依赖、key-value 类型的 map 数据都可以看做 RDD。

2.2 图计算框架 GraphX

Spark GraphX 是一个分布式图处理框架，Spark GraphX 基于 Spark 平台提供对图计算和图挖掘简洁易用的而丰富多彩的接口，极大的方便了大家对分布式图处理的需求。

众所周知，社交网络中人与人之间有很多关系链，例如 Twitter、Facebook、微博和微信等，这些都是大数据产生的地方都需要图计算，现在的图处理基本都是分布式的图处理，而并非单机处理。Spark GraphX 由于底层是基于 Spark 来处理的，所以天然就是一个分布式的图处理系统。

图的分布式或者并行处理其实是把图拆分成很多的子图，然后分别对这些子图进行计算，计算的时候可以分别迭代进行分阶段的计算，即对图进行并行计算。

2.3 动态图形组件 GraphStream

GraphStream 是一个 Java 类库，用于管理动态图形。它由一个面向对象的 API 组成，能够以简便、快速的方式在一张图形中添加边缘和节点，并让它们进行演变^[15]。

2.4 函数式编程语言 Scala

Scala 是一门多范式的编程语言，一种类似 java 的编程语言，设计初衷是实现可伸缩的语言，并集成命令式编程、面向对象编程和函数式编程的各种特性。Scala 有几项关键特性表明了它的面向对象的本质。例如，Scala 中的每个值都是一个对象，包括基本数据类型（即布尔值、数字等）在内，连函数也是对象。另外，类可以被子类化，而且 Scala 还提供了基于 mixin 的组合（mixin-based composition）。

与只支持单继承的语言相比，Scala 具有更广泛意义上的类重用。Scala 允许定义新类的时候重用“一个类中新增的成员定义（即相较于其父类的差异之处）”，Scala 称之为 mixin 类组合。

Scala 还包含了若干函数式语言的关键概念，包括高阶函数（Higher-Order Function）、局部套用（Currying）、嵌套函数（Nested Function）、序列解读（Sequence Comprehensions）等等。

Scala 是静态类型的，这就允许它提供泛型类、内部类、甚至多态方法（Polymorphic Method）。另外值得一提的是，Scala 被特意设计成能够与 Java 和 .NET 互操作。Scala 当前版本还不能在 .NET 上运行，但按照计划将来可以在 .NET 上运行。

Scala 可以与 Java 互操作。它用 scalac 这个编译器把源文件编译成 Java 的 class 文件（即在 JVM 上运行的字节码）。你可以从 Scala 中调用所有的 Java 类库，也同样可以从 Java 应用程序中调用 Scala 的代码。用 David Rupp 的话来说，它也可以访问现存的数之不尽的 Java 类库，这让 Java 类库迁移到 Scala 变得更加容易。

2.5 本章小结

本章介绍了系统的主要技术，包括：通用并行计算框架 Spark、图计算框架 GraphX、动态图形组件 GraphStream 和函数式编程语言 Scala。

第三章 系统功能性需求分析

前两章中，介绍了系统的背景，并介绍了开发中使用的关键技术。本章将描述系统的需求，基于这些需求，系统设计和实现才得以开展。

3.1 系统功能模块

系统功能模块如图 3-1 所示，整个系统分为三大功能块，分别是：图构建模块，可视化模块和顶点分析模块。在顶点分析模块中，用户可以进行顶点重要程度分析、顶点分组（聚类）、顶点邻居计算、顶点到顶点路径计算等多角度的分析。

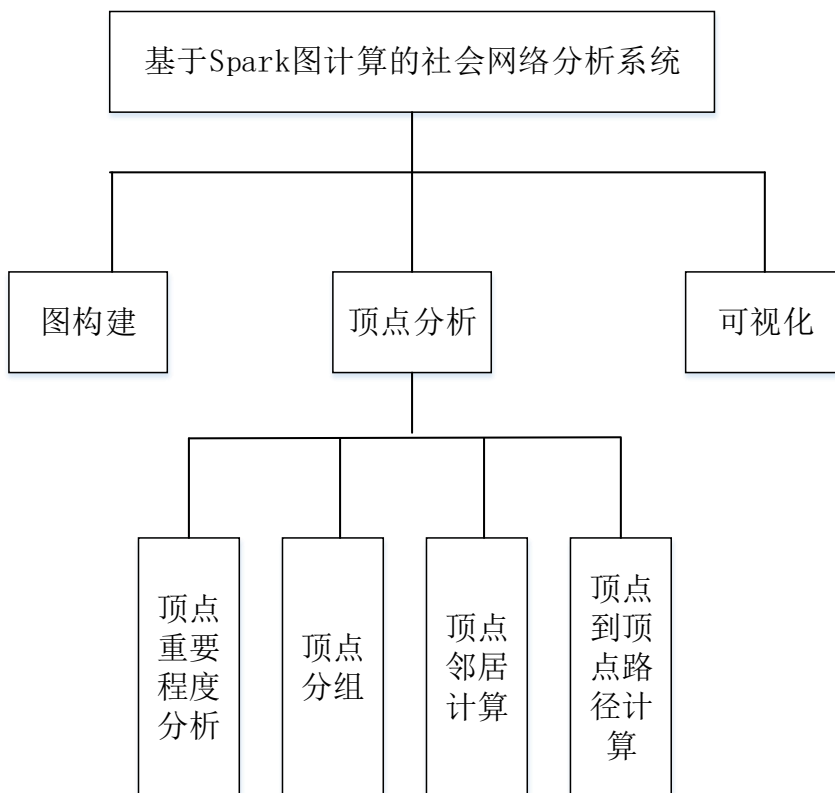


图 3-1 系统功能模块图

3.2 系统用例描述

图构建的用例描述如下表所示：

表 3-1 构建图用例描述

用例编号	0101
用例名称	构建图
用例目标	导入文件系统中的文本文件数据，生成图计算引擎的内部计算单位 Graph。
参与者	程序员、系统
前置条件	1、Spark 工作环境已经启动。 2、用户事先知道文本文件在文件系统中的存储路径。
后置条件	1、生成顶点和边对应的 RDD。 2、生成顶点属性和边属性对应的实体类。 3、生成图计算单位 Graph。
基本流程	1、读取边记录文件，生成对应 RDD。 2、读取顶点记录文件，生成对应 RDD。 3、系统定义边属性对应的实体类。 4、系统定义顶点属性对应的实体类。 5、边 RDD 的第一个元素转化为 ID，其余依次转化为边实体类的属性成员。 6、顶点 RDD 的第一个元素转化为 ID，其余依次转化为顶点实体类的属性成员。 7、构建图。
扩展	

图计算对象可视化的用例描述如下表所示：

表 3-2 图计算对象可视化用例描述

用例编号	0201
用例名称	图计算对象可视化
用例目标	为图计算对象 Graph 创建一个可视化对象 GraphStream ，并在图形界面中显示。
参与者	程序员、系统
前置条件	1、程序员知道图计算对象的名字。
后置条件	1、 GraphStream 对象在可视化对象中显示。
基本流程	1、程序员调用封装好的可视化命令。 2、系统进行相应转化，并把图在图形化界面中显示。
扩展	1a、程序员更改默认样式。

图的可视化对象局部标记的用例描述如下表所示：

表 3-3 图的可视化对象局部标记用例描述

用例编号	0202
用例名称	图的可视化对象局部标记
用例目标	指定子图在原图中以不同的样式显示
参与者	程序员、系统
前置条件	1、程序员知道子图所在的可视化对象的名字和子图对应的对象。
后置条件	1、原 GraphStream 对象的样式更新。
基本流程	1、程序员调用封装好的可视化命令。 2、系统进行相应转化，并把图在图形化界面中显示。

用例编号	0202
扩展	1a、程序员更改默认样式。

顶点重要程度分析的用例描述如下表所示：

表 3-4 顶点重要程度分析用例描述

用例编号	0301
用例名称	顶点重要程度分析
用例目标	计算一张图中每个顶点的重要程度值
参与者	程序员、系统
前置条件	1、程序员知道目标图对象。
后置条件	1、得到顶点 ID 和顶点重要程度值的二元组列表。
基本流程	1、程序员调用封装好的分析命令。 2、系统进行分析，得到分析后的图对象。 3、在控制台输出分析结果。
扩展	2a、程序对分析结果进行进一步的可视化显示。

顶点分组的用例描述如下表所示：

表 3-5 顶点分组用例描述

用例编号	0302
用例名称	顶点分组
用例目标	对一张图中的所有顶点进行聚类
参与者	程序员、系统
前置条件	1、程序员知道目标图对象。
后置条件	1、得到顶点 ID 和顶点标签的二元组列表。
基本流程	1、程序员调用封装好的分析命令。

用例编号	0302
	2、系统进行分析，得到分析后的图对象。 3、在控制台输出分析结果。
扩展	2a、程序对分析结果进行进一步的可视化显示。

顶点 n 层邻居计算的用例描述如下表所示：

表 3-6 顶点 n 层邻居计算用例描述

用例编号	0303
用例名称	顶点 n 层邻居计算
用例目标	计算图中指定顶点指定阶数的邻居
参与者	程序员、系统
前置条件	1、程序员知道子图所在的可视化对象的名字和子图对应的对象。
后置条件	1、得到顶点的 n 阶邻居集合。
基本流程	1、程序员调用封装好的分析命令。 2、系统进行分析，得到顶点的 n 阶邻居集合。 3、在控制台输出分析结果。
扩展	2a、程序可以对分析结果进行进一步的可视化显示。

顶点到顶点路径计算的用例描述如下表所示：

表 3-7 顶点到顶点路径计算用例描述

用例编号	0304
用例名称	顶点到顶点路径计算
用例目标	计算指定两个顶点之间的简单路径
参与者	程序员、系统
前置条件	1、程序员知道目标图计算对象，源顶点和目标顶点的 ID。

用例编号	0304
后置条件	1、得到所有的简单路径。
基本流程	1、程序员调用封装好的分析命令。 2、系统进行分析，得到所有的简单路径。 3、在控制台输出分析结果。
扩展	2a、程序可以对分析结果进行进一步的可视化显示。

3.3 本章小结

本章主要是系统的需求分析，通过系统功能总图和用例描述相结合的形式，介绍项目的图构建模块、可视化模块、顶点分析模块的功能需求。

第四章 系统概要设计

论文在第二章中介绍了系统的架构设计,在第三章中详细介绍了系统的功能需求,本章将介绍系统关键的类结构和主要的功能函数。

4.1 系统架构

首先,由于现实的数据往往不是直接以图形式呈现的,需要进行数据的清洗、提取,并转换为能为 GraphX 图计算引擎操作的数据,因此有必要在大规模图数据挖掘平台中包含非图数据的计算处理能力;其次,一个平台能够完成对图数据和非图数据的挖掘任务,有助于节约集群资源,简化平台管理;最后,采用两种并行计算框架分别完成图数据计算和非图数据计算需要开发人员了解两种程序的设计模式,影响系统的二次开发能力。因此,本文提出采用 Spark 并行计算框架作为系统的非图数据处理引擎,采用 Spark 的图计算接口 GraphX 作为系统的图数据计算引擎,解决上述问题。系统的架构如图 4-1 所示:

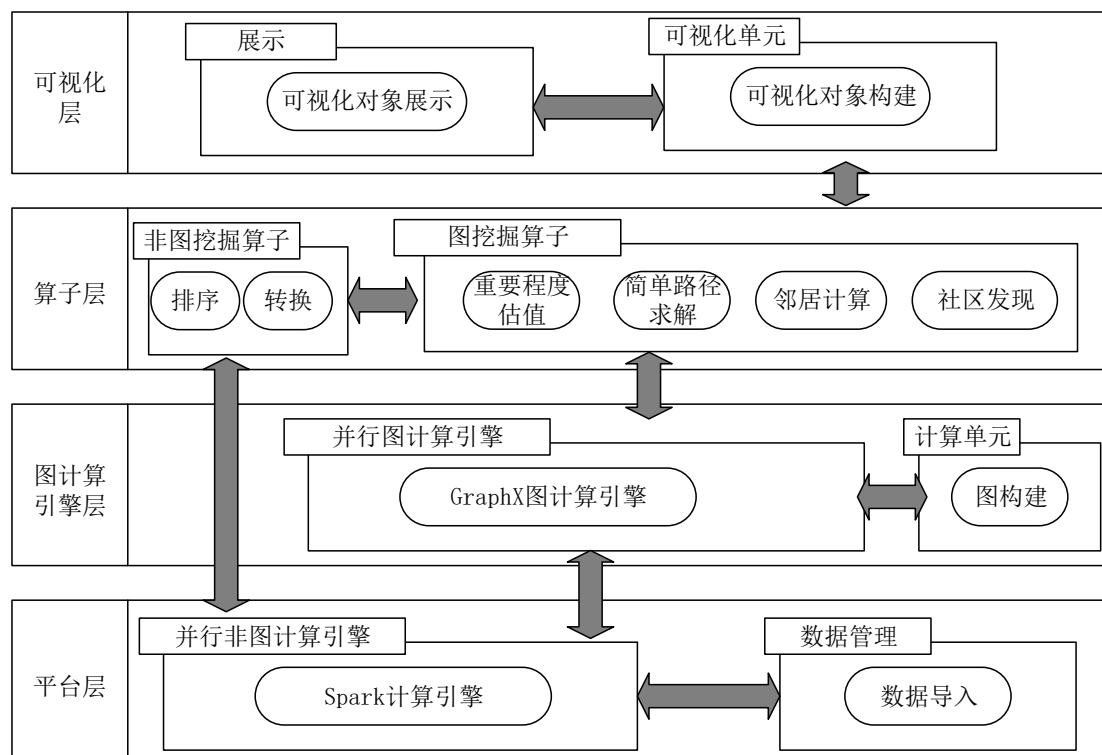


图 4-1 系统架构图

本文系统整体架构包含：提供数据导入、预处理的 Spark 平台层，提供图计算接口的图计算引擎层，提供个性化图数据分析能力的算子层，提供分析结果图形化展示的可视化层。

系统的一个重要可拓展性在于用户可以根据自己的新需求往算子层添加自定义的算子来满足自己的分析需求。

4.2 类设计

整个系统的分析和演示分别基于图计算对象 `Graph` 和可视化图对象 `GraphStream` 两个类，`Graph` 类由图计算框架 `GraphX` 提供，类 `GraphStream` 是动态图形组件 `GraphStream` 自带的可视化图对象类。

本文在第三方框架的基础上，合理应用各种设计模式^[16]来设计系统，减轻各子系统之间的耦合，便于系统未来的扩展和维护。

4.2.1 算子子系统

对于算子层，本文使用“外观模式”进行设计，将算子层和图构建层、可视化层相分离，如图 4-2 所示。本文为算子子系统开发一个外观 `Analysis` 类，对外

提供一个简单的接口，减轻各层之间的耦合。`Analysis` 中的 `analysis` 方法可以对不同的算子进行组合，来满足特定的分析要求。

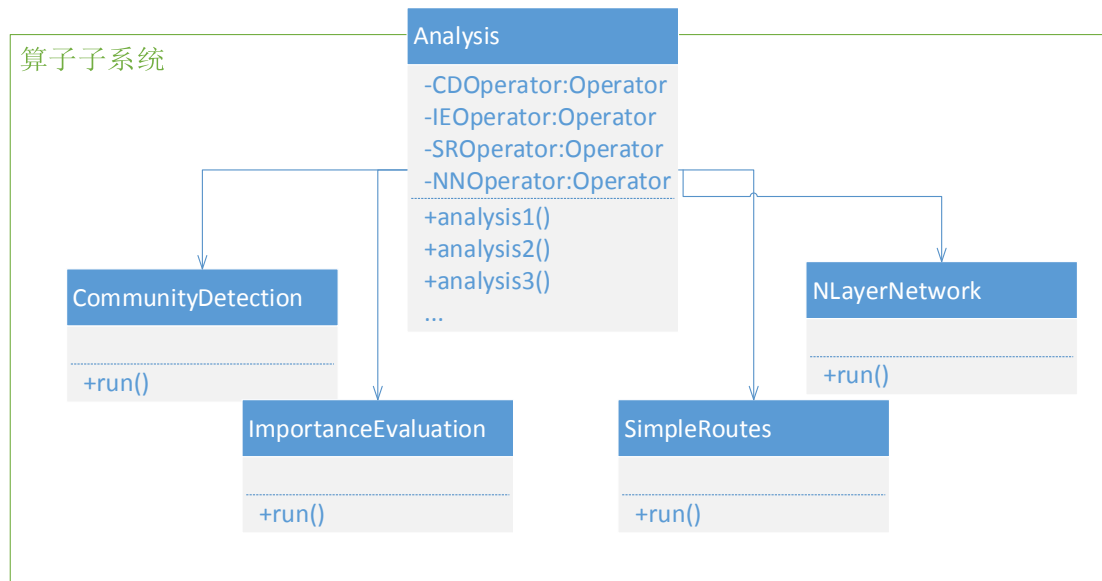


图 4-2 算子子系统类图

4.2.2 可视化对象构建

对一张图进行可视化时往往需要对它的各种属性标签（顶点名称、顶点 ID、边的名称等）进行组合显示，本文使用“建造者模式”将可视化图对象 `GraphStream` 的构建和表示分离，使得同样的构建步骤可以创建不同的表示，如图 4-3。类 `Director` 作为子系统对外的交互接口，用户只需指定具体的建造器，如 `CD_GSBuilder` 就可以得到相应的可视化对象，而具体的建造过程和细节无需知道。

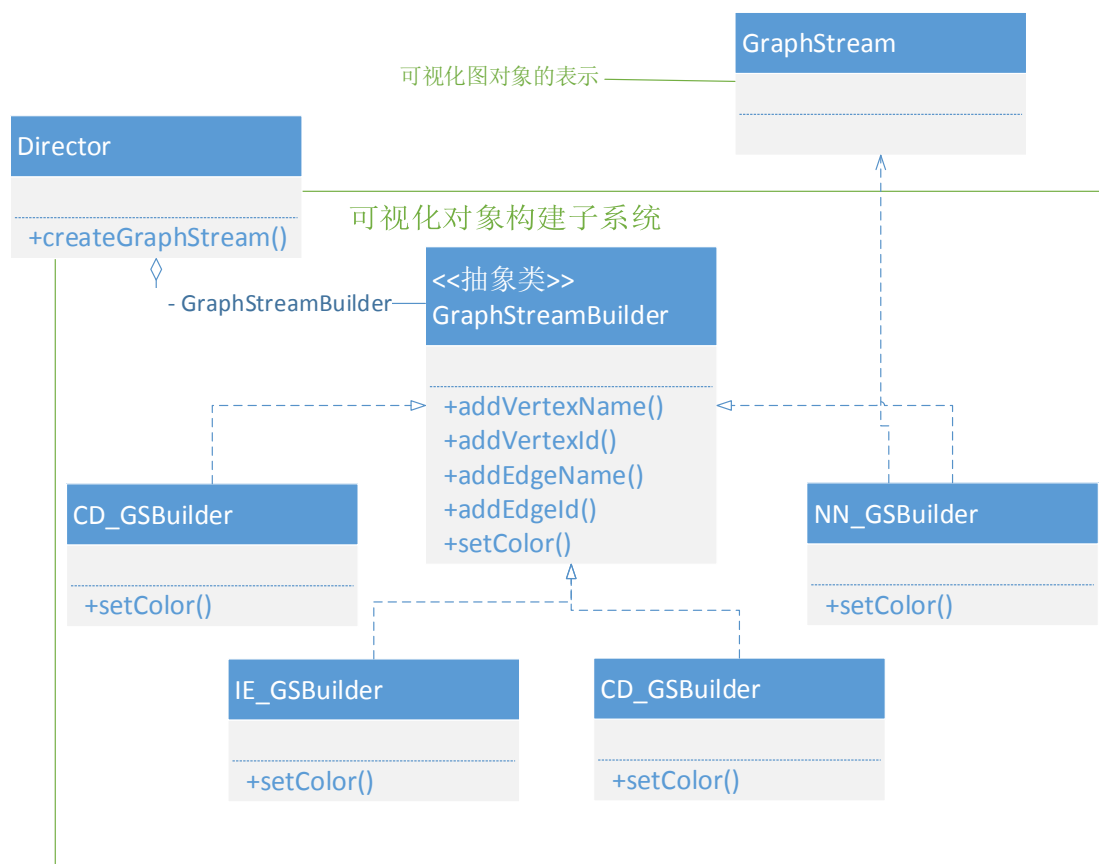


图 4-3 可视化对象构建子系统

策略模式是定义一系列算法的方法，从概念上来看，所有这些算法完成的都是相同的工作，只是实现不同，它可以以相同的方式调用所有的算法，减少各种算法类之间的耦合。由于不同的分析结果需要在可视化图对象上用不同的着色效果来显示，本文使用策略模式来设计顶点的着色功能：在抽象类 `GraphStreamBuilder` 中定义了该算法的接口——`setColor()` 函数，在具体实现类（`CD_GSBuilder` 等）中给出了 `setColor` 的不同的具体的实现。

4.2.3 分析演示子系统

本文使用模板方法定义了分析演示的算法骨架：构建 `Graph`、显示 `Graph`、分析 `Graph`、显示新的 `Graph`，将一些步骤的具体实现延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重新定义该算法的某些特定步骤，如 `analysis()` 和 `displayGraph()`，并且通过把不变的行为 `createGraph()`、`displayInitGraph()` 搬移到超类 `Template`，去除子类中的重复代码。分析演示子系统类图如图 4-4 所示。

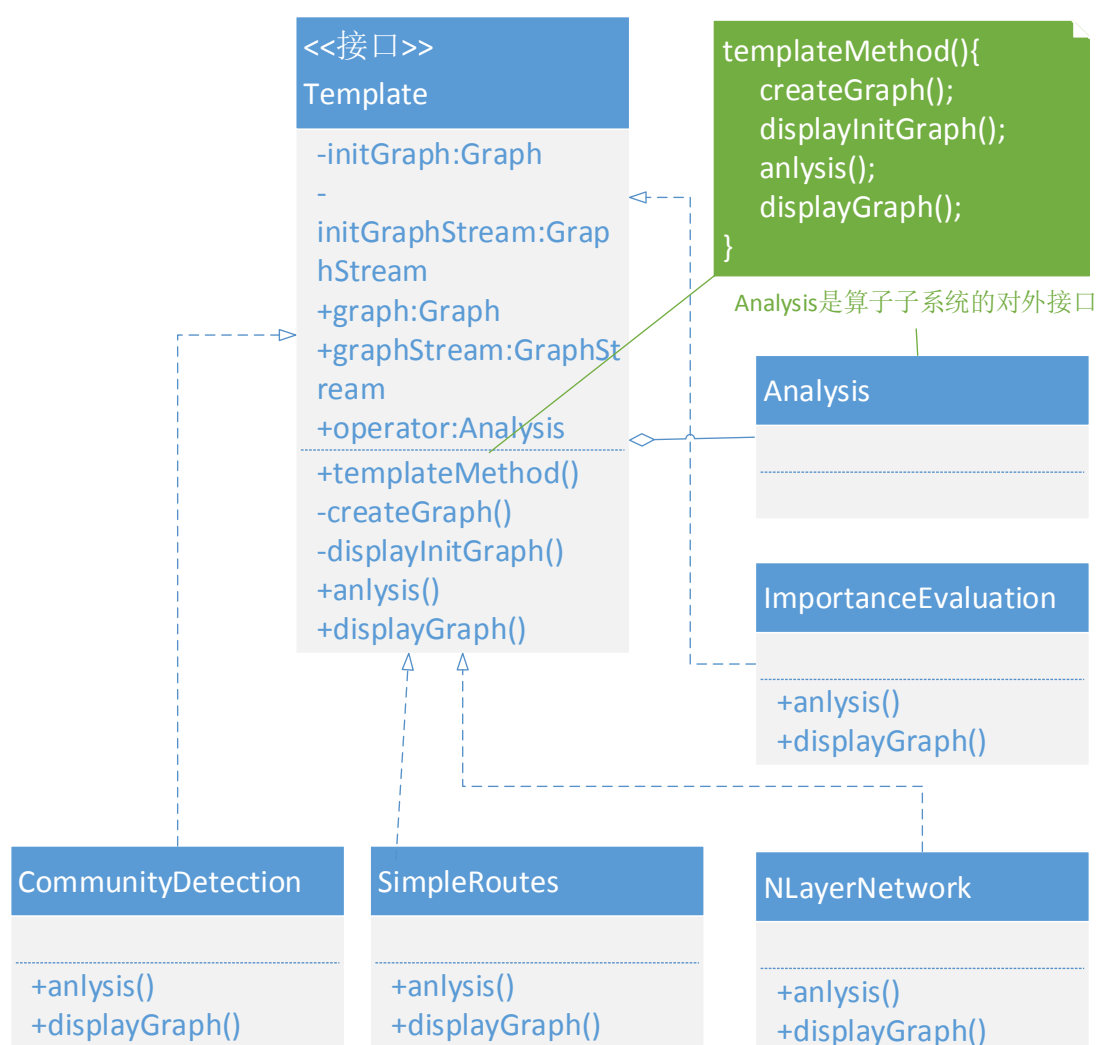


图 4-4 分析演示子系统

4.3 功能函数设计

本小节讨论系统核心功能函数的概要设计。

4.3.1 图构建

1、createGraph

【函数原型】

```
def createGraph(vertexFilePath:String, edgeFilePath:String): Graph[VD,ED]
```

【功能】给定文本文件数据，构建图计算对象

【参数说明】

- 1) vertexFilePath 顶点文件路径

2) edgeFilePath 边文件路径

【返回值说明】一个图计算对象

【技术方案】先将文本数据读取为 RDD，再将 RDD 转化为 Graph。

4.3.2 整体分析

1、sortByImportance

【函数原型】

```
def sortByImportance(graph:Graph[VD,ED], tol:Double = 0.0001) : Array[(VertexId,VD)]
```

```
def sortByImportance(graph:Graph[VD,ED], tol:Double = 0.0001, num:Int) : Array[(VertexId,VD)]
```

【功能】把所有的顶点按照重要程度的降序排序，重要程度由 PageRank 的大小决定，并打印和返回指定数量的顶点信息。

【参数说明】

- 1) graph 需要做顶点重要程度排序的图计算对象
- 2) tol PageRank 算法的容错度，默认是 0.0001
- 3) num 返回值中包含顶点的数量

【返回值说明】返回的是一个按照重要程度排序的点的 array

【技术方案】调用 util 包中的 PageRank 类中的 run 函数得到一个 Graph，每个点的值都是它的 PageRank 值。再由这个 Graph 得到 vertexRDD，把 vertexRDD 转化为相应的 Array[(VertexId,Double)]，最后对这个 Array 进行排序。

2、vertexesCluster

【函数原型】

```
def vertexesCluster(graph:Graph[Person,Link], num:Int) : (Graph[VertexId,Link], Array[(VertexId,Long)])
```

【功能】利用半监督的聚类算法——标签传播算法对一个图的顶点做聚类，输出不同标签对应的顶点数量和每个顶点对应的标签。

【参数说明】

- 1) `graph` 需要做顶点聚类的图
- 2) `num` 算法的迭代次数

【返回值说明】二元组：以顶点最后的标签为顶点属性的 `Graph` 和统计不同标签对应的顶点数量的 `Array`。

【技术方案】具体使用常用的社区发现算法 `LPA` 来进行聚类，先得到一个以顶点最后的标签为顶点属性的 `Graph`，再根据这个 `Graph` 统计标签对应的顶点数量。

4.3.3 局部分析

1、`subgraphWithVertexes`

【函数原型】

```
def subgraphWithVertexes( graph:Graph[Person,Link], set:Set[Long] ) : Graph[Person,Link]
```

【功能】使用指定的顶点从原图中抠取出一张子图

【参数说明】

- 1) `graph` 需要操作的图，指定的点所在的图
- 2) `set` 包含指定顶点 ID 的集合

【返回值说明】返回指定点集构成的子图

【技术方案】调用 `subgraph` 操作进行过滤

2、`nLayerNetwork`

【函数原型】

```
def nLayerNetwork(graph:Graph[Person,Link], n:Int, srcId:Long): Graph[Person,Link]
```

【功能】从原图中取出从指定顶点发散出的 `n` 层网络，指定点属于第 0 层。

【参数说明】

- 1) `graph` 需要操作的图，指定的点所在的图
- 2) `n` 网络的层数
- 3) `srcId` 指定顶点的 ID

【返回值说明】 返回指定点集构成的子图

【技术方案】 循环调用 GraphX 内置的 collectNeighborId 接口进行迭代计算，得到 n 层网络的顶点集，再调用 subgraphWithVertexes 从原图中过滤出子图。

4.3.4 可视化

1、displayGraph

【函数原型】

```
def displayGraph(graph:Graph[Person,Link],name:String):SingleGraph
```

【功能】 根据图计算对象生成一个可视化对象，并在图形界面中显示

【参数说明】

- 1) graph 需要操作的图计算对象
- 2) name 可视化对象的命名

【返回值说明】 返回可视化对象

【技术方案】 图计算对象的边和顶点一一转化为可视化对象的边和顶点，最后显示。

2、subgraphMark

【函数原型】

```
def subgraphMark(graphStream:SingleGraph,subgraph:Graph[Person,Link])
```

【功能】 在可视化对象上标记一个子图，用不同的样式显示

【参数说明】

- 1) graphStream 被标记的原图的可视化对象
- 2) subgraph 子图对应的图计算对象

【返回值说明】 返回可视化对象

【技术方案】 一一查找子图的顶点和边在原图中的对应对象，并修改样式。

4.4 本章小结

本章使用各种设计模式来进行类设计，减轻各子系统之间的耦合，便于系统未来的扩展和维护，并对核心的功能函数给出了宏观设计。

第五章 系统设计与实现

上一章介绍了系统的概要设计，本章在上一章的基础上针对图构建、可视化、整体分析和局部分析四个模块进一步介绍系统的详细设计和实现。

5.1 图构建模块

本模块意在将文本数据转化为可以被 GraphX 图计算引擎所直接接受的分析单元 Graph，该过程分两步走：先将文本数据转化为 Spark 平台能够接收的数据格式 RDD；再用 RDD 构造生成 Graph。

文本数据的格式如表格 5-1、5-2 所示：

表格 5-1 顶点文本数据格式

顶点 ID	属性 1	属性 2	...	属性 n
-------	------	------	-----	------

表格 5-2 边文本数据格式

边的 ID	属性 1	属性 2	...	属性 n
-------	------	------	-----	------

在本功能的实现中，主要涉及对文本字符串的读取和切割，以及对切割后的数据的一系列转化。流程如图 5-1 所示。

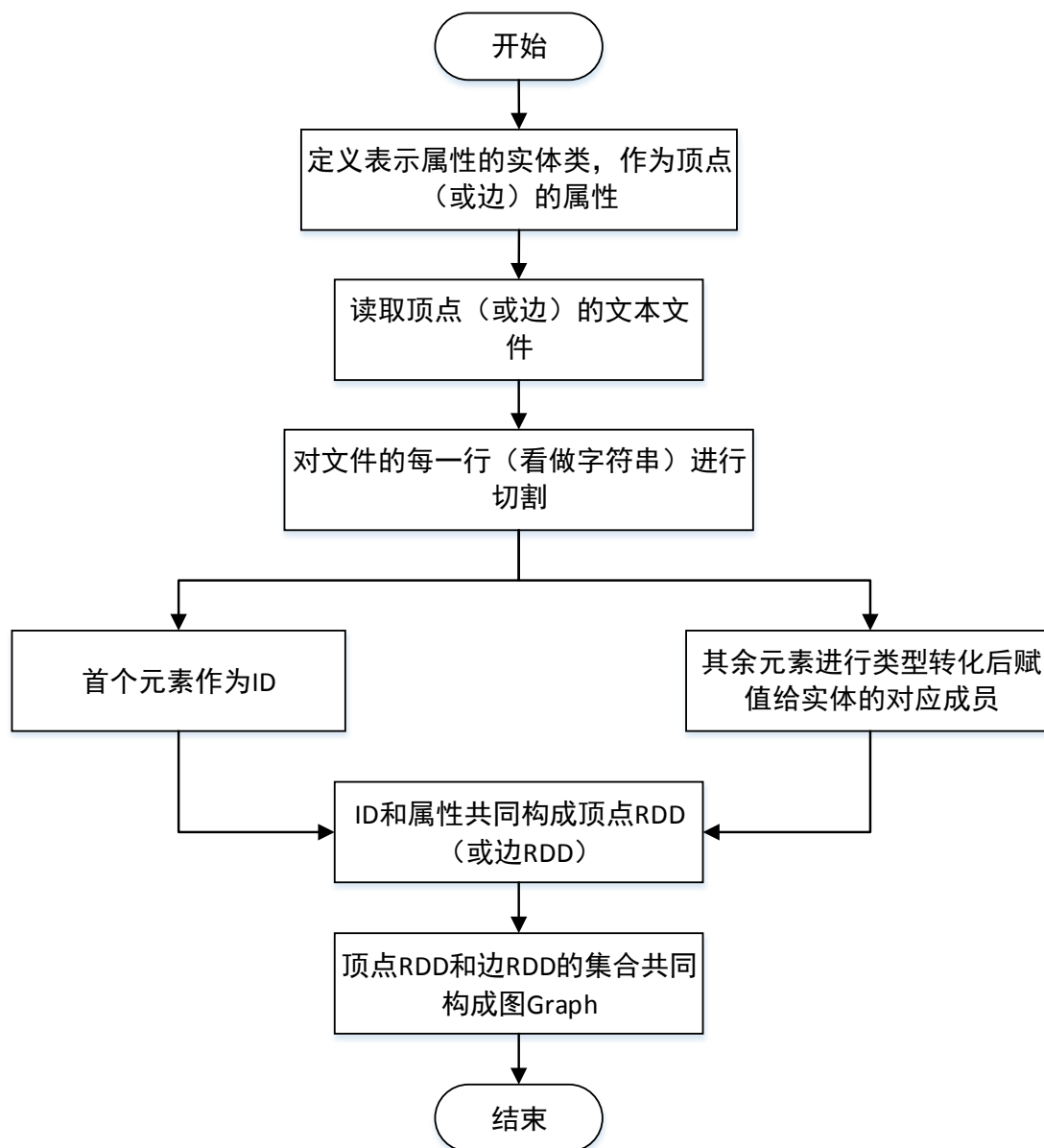


图 5-1 构建图流程图

首先系统读取文件系统中的文本文件，每个文件读取为一个数组，文件的每一行以字符串的形式存储为数组的一个元素。然后对每个字符串进行切割产生若干元素，第一个元素作为 ID，之后的每个元素进行相应的类型转化赋值给对应实体类的属性成员。这里的实体相当于顶点（或边）的属性，与顶点（或边）的 ID 共同构成顶点（或边）。最后顶点的集合和边的集合构造成为一个图。部分重要代码如下图所示：

```
/**
 * create a graph from files which have specified form
 * @param vertexFilePath file path of vertexs.csv
 * @param edgeFilePath file path of edges.csv
 * @return
 */
def createGraph(sc:SparkContext ,vertexFilePath:String,
edgeFilePath:String): Graph[Person,Link] ={
    val vertices = sc.textFile(vertexFilePath)
    val links= sc.textFile(edgeFilePath)
    //构建边、顶点 RDD
    links.foreach(println)
    val verticesRDD: RDD[(VertexId,Person)] = vertices map { line
=>
        val row = line split ','
        (row(0).toLong,Person(row(1),row(2)))
    }

    val linksRDD:RDD[Edge[Link]] = links map { line =>
        val row = line split ','
        Edge(row(0).toLong, row(1).toLong, Link(row(4), row(5)))
    }
    //构建图
    val social: Graph[Person,Link] = Graph(verticesRDD, linksRDD)
    println("Create a Graph successfully!")
    return social
}
```

5.2 可视化模块

本模块利用第三方工具包 `GraphStream` 对图进行可视化展示，本小节将讲解展示一个可视化图对象的核心步骤。

在 `GraphStream` 中，所有的可视化对象均继承自 `Graph` 这个类，为了避免和 `GraphX` 中的已有的 `Graph` 类的类名冲突，首先将 `GraphStream` 中的 `Graph` 重命名为 `GraphStream`，代码如下所示：

```
import org.graphstream.graph.{Graph => GraphStream}
```

可视化显示核心代码如下所示：

```
// 1、生成原始的可视化对象
val graphStream:SingleGraph = new SingleGraph(name);

// 2、设置图形可视化的全局属性
graphStream.addAttribute("ui.stylesheet","url(./style/stylesheet.css)")//
设置样式文件

graphStream.addAttribute("ui.quality") //保证更慢但质量更好的渲染
graphStream.addAttribute("ui.antialias") //保证图形的抗锯齿效果

//3、 将 GraphX 对象的顶点加载到 GraphStream 中，并设置要展示
的属性

for ((id,person:Person) <- graph.vertices.collect()) {
    val node =
graphStream.addNode(id.toString).asInstanceOf[SingleNode]

    node.addAttribute("ui.label",id  + "\n"+person.name)
}

// 4、将 GraphX 对象的顶点加载到 GraphStream 中，并设置要展示
的属性

for (Edge(x,y,link:Link) <- graph.edges.collect()) {

    val edge = graphStream.addEdge(x.toString ++ y.toString,
        x.toString, y.toString,
```



```

        true).

        asInstanceOf[AbstractEdge]

    }

    //5、显示

    graphStream.display()

```

5.3 整体分析模块

5.3.1 顶点重要程度分析

本模块利用 PageRank[17]算法对所有顶点的重要程度进行估值。

PageRank 算法针对于 Web 系统而设计，其基本思想是：指向某页面的链接将增加该页面的 PageRank 值。受该算法启发，可以将社会网络中的节点模拟成 Web 中的页面，而将社会网络中的边模拟成 Web 中的超链接[18]。因此，可以采用类似 PageRank 的算法来计算节点的影响力。

GraphX 引擎集成了 PageRank 算子，进行 PageRank 计算并按降序排列的核心代码如下：

```

val ranks = graph.pageRank(0.0001).vertices.sortBy(_._2,false) //false 表示
降序排列

```

在这段代码中，我们调用 GraphX 内置的 PageRank 算法进行计算，并通过的 RDD 的 sortBy 接口进行排序。

该部分进行可视化显示核心代码如下：

```

//顶点样式

node {

    fill-mode: dyn-plain;

    fill-color: green, red;

}

//将每个顶点的 PageRank 乘以系数 0.7

val x:java.lang.Double = ranks.lookup(id)(0) * 0.7

```

```
//为每个顶点设置显示颜色
node.setAttribute("ui.color",x)
```

在这段代码中，我们将顶点颜色填充为 green 和 red 的混合色，x 与顶点的 PageRank 值成正比，x 的值越大，顶点的颜色越偏向于红色。

5.3.2 顶点分组

本模块利用标签传播算法^[19]对图中的所有顶点进行聚类，将相同标签的顶点分为一组。

标签传播算法（LPA）由 Zhu 等人于 2002 年提出^[19]，它是一种基于图的半监督学习方法，其基本思路是用已标记节点的标签信息去预测未标记节点的标签信息。利用样本间的关系建立关系完全图模型，在完全图中，节点包括已标注和未标注数据，其边表示两个节点的相似度，节点的标签按相似度传递给其他节点。标签数据就像是一个源头，可以对无标签数据进行标注，节点的相似度越大，标签越容易传播。由于该算法简单易实现，算法执行时间短，复杂度低且分类效果好，引起了国内外学者的关注，并将其广泛地应用到多媒体信息分类、虚拟社区挖掘等领域中。

针对在大规模网络结构中进行聚类时，需要事先知道聚类数目、规模等先验信息，或者聚类算法的计算代价比较大的情况，Raghavan 等人提出了基于标签传播的简单快速算法^[19]。在算法中，初始化时每个节点被赋予唯一的数字标签，在之后的每步中，将所有节点以随机序列排列，根据邻接节点当前出现频率最高的标签依次确定随机序列中节点的标签。如此迭代，在几乎线性的时间内实现了节点聚类。核心代码如下所示：

```
val labelGraph = LabelPropagation.run(graph,n)
```

本模块使用顶点的 ID 作为它们的初始标签，聚类后在同一组的顶点具有相同的标签。进行顶点分组的核心代码如下所示，其中 n 是迭代次数：

```
//顶点样式
node {
    fill-mode: dyn-plain;
    fill-color: green, red;
```

```
}  
  
val x:java.lang.Double = (label - min)/delt.toDouble//为每个顶点设置显示颜色  
  
node.setAttribute("ui.color",x)
```

在该部分的可视化中，用颜色区别不同顶点的组别，采用统一的样式方案，不同的是 x 的计算方式， x 的计算公式为： $x = (\text{label} - \text{minID}) / (\text{maxID} - \text{minID})$ ，其中 label 是进行迭代后当前顶点的标签， minID 和 maxID 是迭代前的最小标签和最大标签。

5.4 局部分析模块

局部分析模块对指定 ID 的顶点进行相关分析。

5.4.1 顶点邻居计算

本模块通过反复迭代对给定的顶点求解他的 N 层网络。首先通过 GraphX 引擎提供的 `collectNeighborIds` 接口计算出每个顶点的一级邻居，然后遍历一级邻居的邻居即可得到二级邻居，以此类推，循环迭代，直到求得第 n 层邻居，在这个过程中，每求得一层邻居都加入到最后的结果集中。为了提高计算效率，系统对重复计算进行了规避，具体算法流程如图 5-2 所示。

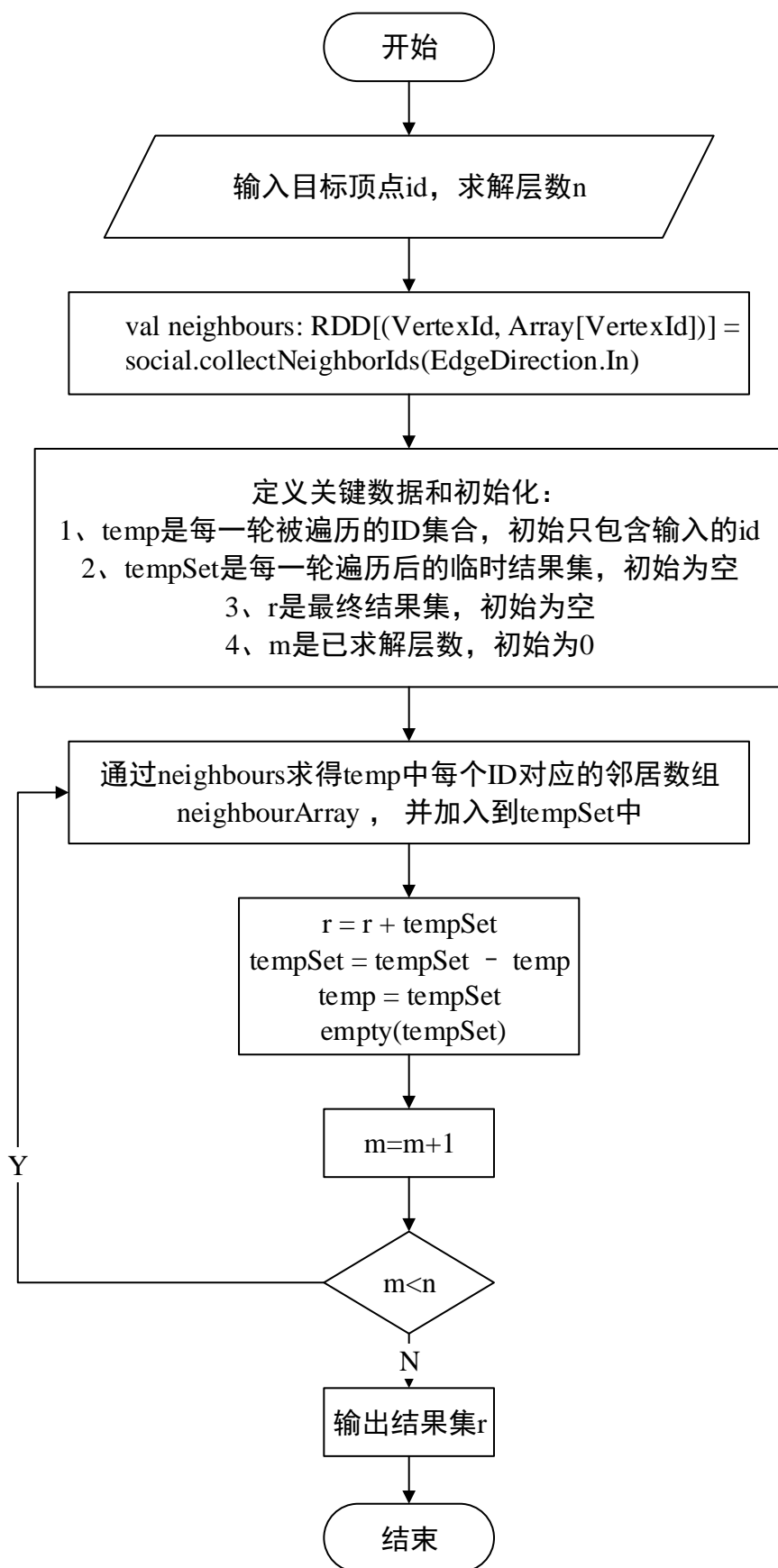


图 5-2 求解 n 层邻居流程图

该模块的可视化部分，用不一样的颜色对计算结果涉及的顶点进行标记。

5.4.2 顶点到顶点路径计算

本模块使用深度优先搜索算法计算给定的两个顶点之间的所有简单路径（如果一条路径上的顶点除了起点和终点可以相同外，其它顶点均不相同，则称此路径为一条简单路径）^[20]。

该部分深度优先算法的递归函数伪代码如下所示：

```
创建存储临时路径的栈 S;  
创建存储最终结果的列表 R;  
Function SimpleRoutesBetween(起点, 终点){  
    S.push(起点)  
    while(起点还有没有遍历过的邻居节点){  
        取下一个邻居节点;  
        if (邻居节点是终点) {  
            S.push(邻居节点);  
            将 S.toArray()加入 R;  
            S.pop();  
            直接结束本次循环;  
        }  
        Else If (当前邻居节点不构成环路) {  
            SimpleRoutesBetween(当前邻居节点, 终点);  
        }  
    }  
    S.pop()//回溯前还原现场  
}
```

该模块的可视化部分，用不一样的颜色对计算结果涉及的顶点进行标记。

5.5 本章小结

本章对系统的图构建、可视化、整体分析、局部分析等模块的具体实现思路 and 实现方法进行了详细介绍，对于关键的算法和步骤给出了流程图，部分重要代码予以展示。

第六章 系统测试

本章对系统进行功能性测试，先对基础的图构建功能给出测试，再对四个主要分析模块的分析结果在控制台进行输出，并给出可视化的展示。

6.1 测试数据

本文采用的测试数据集是“智器云”（一款大数据可视化分析软件，<http://www.zqykj.cn/>）提供的社会关系网络样例数据。测试数据集包含顶点 69 个，边 84 条。

6.2 测试环境

系统测试是在个人笔记本电脑上进行的，系统的测试环境情况如表 6-1 所示。

表格 6-1 系统测试环境说明

参数	配置描述
CPU	2.2 GHz Intel Core i7, 4 cores
L2 Cache (per Core)	256 KB
L3 Cache	6 MB
Memory	16 GB 1600 MHz DDR3
硬盘	220 GB
操作系统	OS X EI Captitan Version 10.11.3
集成开发环境	IntelliJ IDEA 15
Spark	Stand-alone Model

6.3 图构建和可视化测试

图 6-1 对图构建的可视化结果进行了展示，图中绿色圆点对应人物的实体，人物旁边的文字标注对应人物的姓名，如 Wang Jie、Wang Feng，紫红色线段用于连接两个人物，表示两个人物有一定强度的联系。



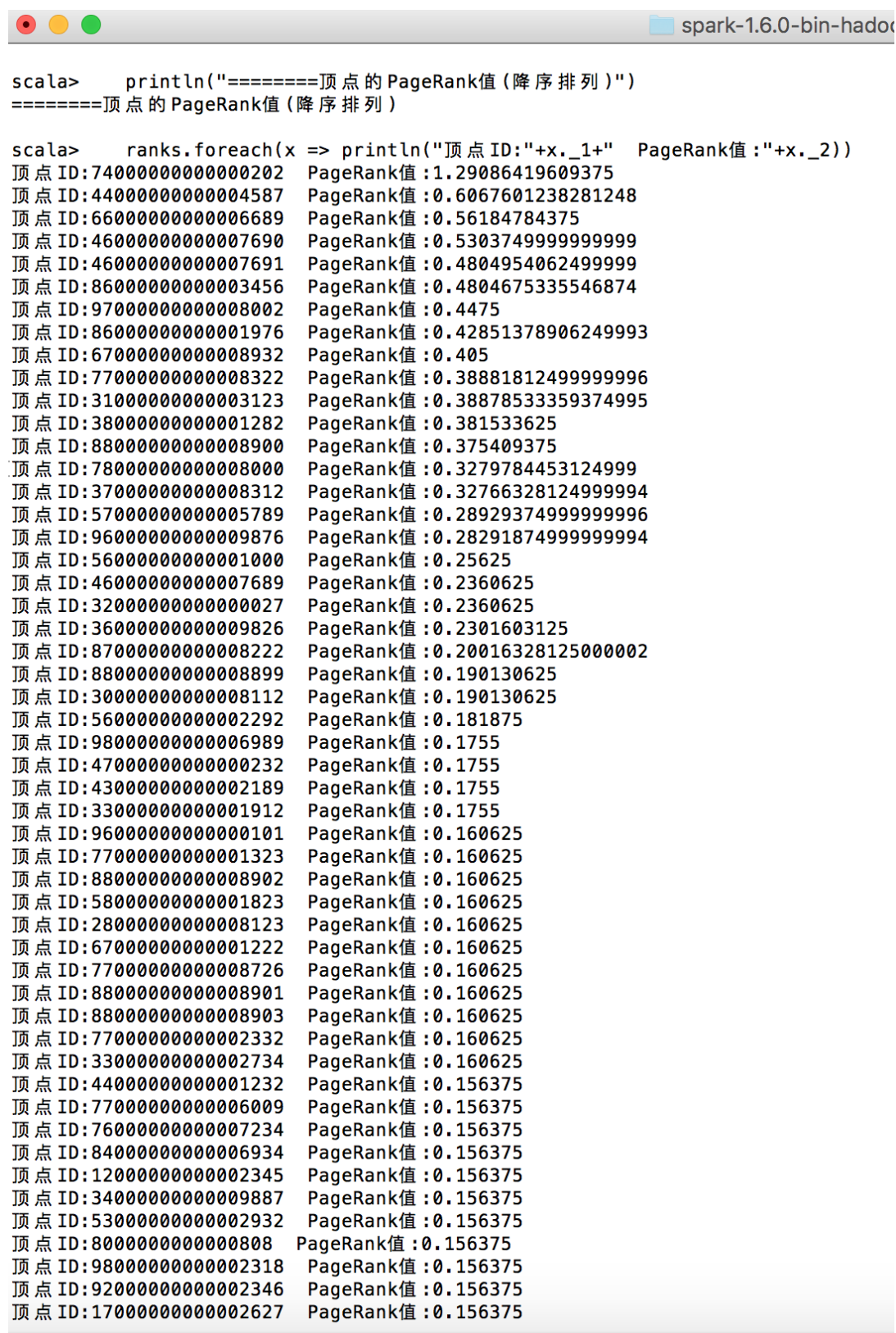
图 6-1 带有人物姓名的图构建可视化结果

系统可以根据需要对顶点或边的属性进行定制化的显示，图 6-2，6-3 中不仅显示了人物的姓名，还分别显示了人物的 ID 和人物的关系。人物的关系有 cousin（表亲关系）、partner（合作关系）、schoolmate（同学）等等。

37

图 6-3 帶有人物姓名和关系的图构建可视化结果

6.4.1 顶点重要程度分析



```

scala> println("====顶点 PageRank 值 (降序排列)")
====顶点 PageRank 值 (降序排列)

scala> ranks.foreach(x => println("顶点 ID:" + x._1 + " PageRank 值:" + x._2))
顶点 ID:7400000000000202 PageRank 值:1.29086419609375
顶点 ID:44000000000004587 PageRank 值:0.6067601238281248
顶点 ID:66000000000006689 PageRank 值:0.56184784375
顶点 ID:46000000000007690 PageRank 值:0.5303749999999999
顶点 ID:46000000000007691 PageRank 值:0.4804954062499999
顶点 ID:86000000000003456 PageRank 值:0.4804675335546874
顶点 ID:97000000000008002 PageRank 值:0.4475
顶点 ID:86000000000001976 PageRank 值:0.42851378906249993
顶点 ID:67000000000008932 PageRank 值:0.405
顶点 ID:77000000000008322 PageRank 值:0.38881812499999996
顶点 ID:31000000000003123 PageRank 值:0.38878533359374995
顶点 ID:38000000000001282 PageRank 值:0.381533625
顶点 ID:88000000000008900 PageRank 值:0.375409375
顶点 ID:78000000000008000 PageRank 值:0.3279784453124999
顶点 ID:37000000000008312 PageRank 值:0.32766328124999994
顶点 ID:57000000000005789 PageRank 值:0.28929374999999996
顶点 ID:96000000000009876 PageRank 值:0.28291874999999994
顶点 ID:56000000000001000 PageRank 值:0.25625
顶点 ID:46000000000007689 PageRank 值:0.2360625
顶点 ID:32000000000000027 PageRank 值:0.2360625
顶点 ID:36000000000009826 PageRank 值:0.2301603125
顶点 ID:87000000000008222 PageRank 值:0.20016328125000002
顶点 ID:88000000000008899 PageRank 值:0.190130625
顶点 ID:30000000000008112 PageRank 值:0.190130625
顶点 ID:56000000000002292 PageRank 值:0.181875
顶点 ID:98000000000006989 PageRank 值:0.1755
顶点 ID:47000000000000232 PageRank 值:0.1755
顶点 ID:43000000000002189 PageRank 值:0.1755
顶点 ID:33000000000001912 PageRank 值:0.1755
顶点 ID:96000000000000101 PageRank 值:0.160625
顶点 ID:77000000000001323 PageRank 值:0.160625
顶点 ID:88000000000008902 PageRank 值:0.160625
顶点 ID:58000000000001823 PageRank 值:0.160625
顶点 ID:28000000000008123 PageRank 值:0.160625
顶点 ID:67000000000001222 PageRank 值:0.160625
顶点 ID:77000000000008726 PageRank 值:0.160625
顶点 ID:88000000000008901 PageRank 值:0.160625
顶点 ID:88000000000008903 PageRank 值:0.160625
顶点 ID:77000000000002332 PageRank 值:0.160625
顶点 ID:33000000000002734 PageRank 值:0.160625
顶点 ID:44000000000001232 PageRank 值:0.156375
顶点 ID:77000000000006009 PageRank 值:0.156375
顶点 ID:76000000000007234 PageRank 值:0.156375
顶点 ID:84000000000006934 PageRank 值:0.156375
顶点 ID:12000000000002345 PageRank 值:0.156375
顶点 ID:34000000000009887 PageRank 值:0.156375
顶点 ID:53000000000002932 PageRank 值:0.156375
顶点 ID:8000000000000808 PageRank 值:0.156375
顶点 ID:98000000000002318 PageRank 值:0.156375
顶点 ID:92000000000002346 PageRank 值:0.156375
顶点 ID:17000000000002627 PageRank 值:0.156375

```

图 6-4 顶点 PageRank 值输出结果

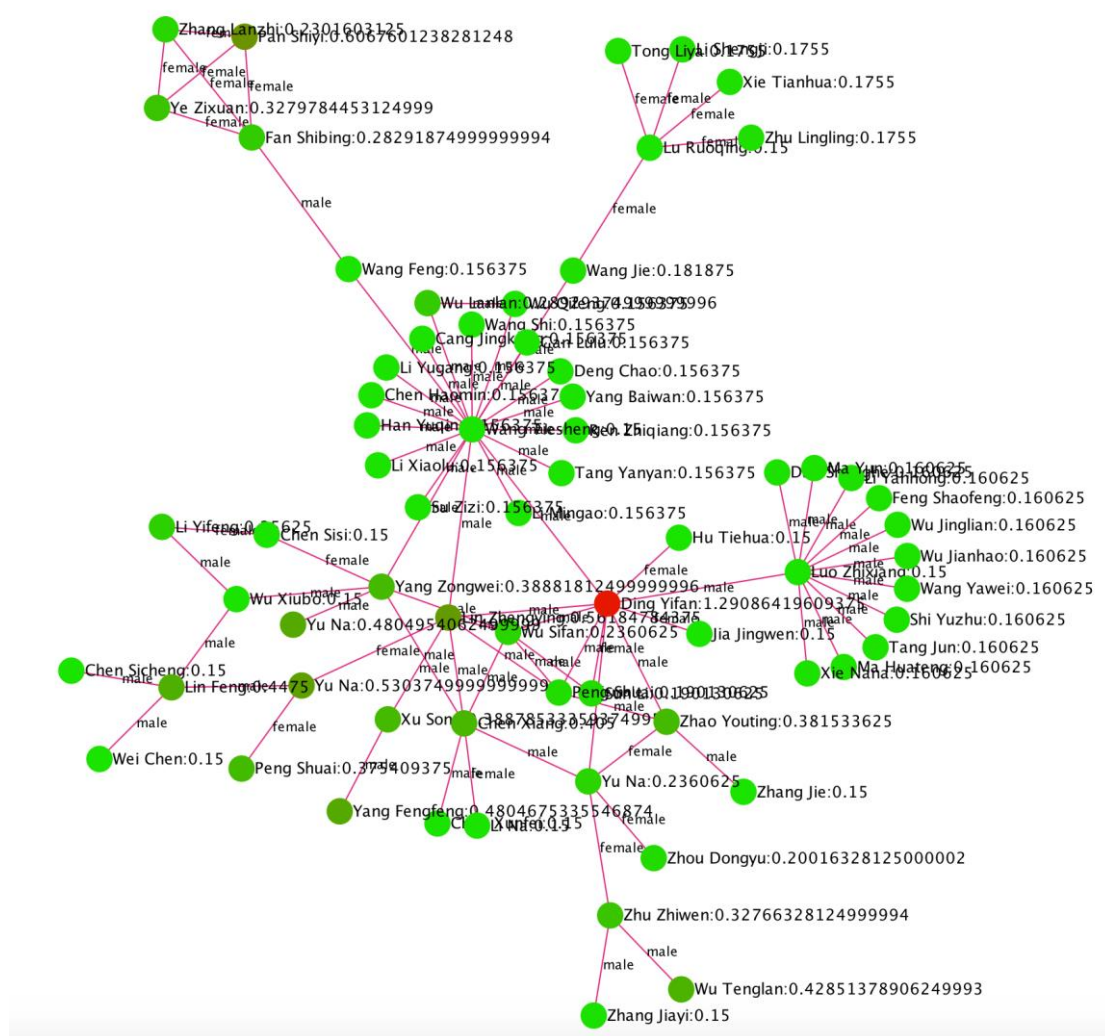



图 6-5 顶点重要程度可视化结果

6.4.2 顶点分组

图 6-6 显示了顶点分组测试的控制台输出结果，输出结果为两栏，左边一栏为顶点的 ID，右边一栏为使用标签传播算法分组后顶点的标签。



```

scala> v.foreach(x => println("顶点 ID:"+x._1+" 标签:"+x._2))
顶点 ID:23000000000002371 标签:23000000000002371
顶点 ID:44000000000001232 标签:84000000000006934
顶点 ID:88000000000008899 标签:32000000000000027
顶点 ID:77000000000006009 标签:84000000000006934
顶点 ID:37000000000008312 标签:37000000000008312
顶点 ID:45000000000002721 标签:40000000000001381
顶点 ID:30000000000008112 标签:32000000000000027
顶点 ID:28000000000008123 标签:88000000000008903
顶点 ID:9600000000000101 标签:88000000000008903
顶点 ID:8000000000000808 标签:84000000000006934
顶点 ID:78000000000001982 标签:32000000000000027
顶点 ID:67000000000001222 标签:88000000000008903
顶点 ID:76000000000007234 标签:84000000000006934
顶点 ID:77000000000008726 标签:88000000000008903
顶点 ID:66000000000006689 标签:32000000000000027
顶点 ID:54000000000002771 标签:23000000000002371
顶点 ID:40000000000001381 标签:40000000000001381
顶点 ID:84000000000006934 标签:84000000000006934
顶点 ID:83000000000004422 标签:83000000000004422
顶点 ID:78000000000008000 标签:96000000000009876
顶点 ID:39000000000003992 标签:32000000000000027
顶点 ID:64000000000003255 标签:7400000000000202
顶点 ID:36000000000009826 标签:96000000000009876
顶点 ID:98000000000002318 标签:84000000000006934
顶点 ID:87000000000008222 标签:67000000000008932
顶点 ID:67000000000008932 标签:67000000000008932
顶点 ID:86000000000003456 标签:32000000000000027
顶点 ID:46000000000007691 标签:23000000000002371
顶点 ID:57000000000005789 标签:84000000000006934
顶点 ID:21000000000002332 标签:21000000000002332
顶点 ID:88000000000008901 标签:88000000000008903
顶点 ID:92000000000002346 标签:84000000000006934
顶点 ID:33000000000001912 标签:43000000000002189
顶点 ID:12000000000002345 标签:84000000000006934
顶点 ID:17000000000002627 标签:84000000000006934
顶点 ID:88000000000008903 标签:88000000000008903
顶点 ID:77000000000008322 标签:56000000000001000
顶点 ID:57000000000002223 标签:84000000000006934
顶点 ID:56000000000001000 标签:56000000000001000
顶点 ID:46000000000007690 标签:32000000000000027

```

图 6-6 顶点分组控制台输出结果

图 6-7 为顶点分组测试的可视化测试结果，其中相同颜色的顶点表示在同一个分组。

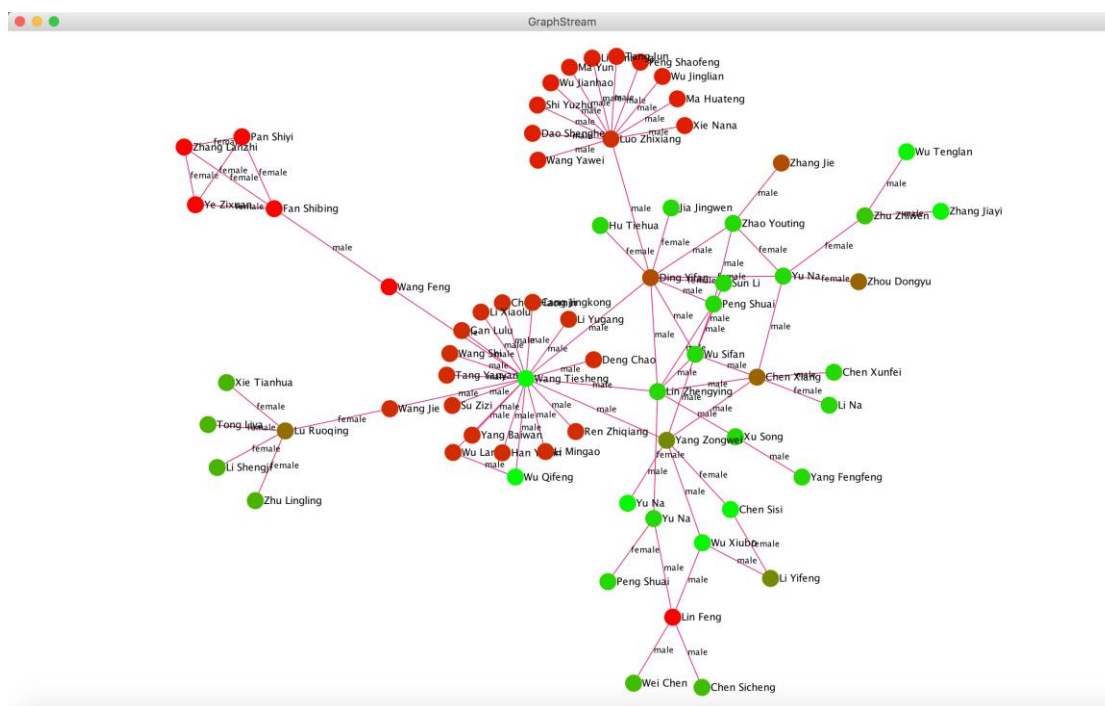


图 6-7 顶点分组可视化结果

6.4.3 顶点邻居计算

本文计算了人物 Liu Zhengying 的 2 层邻居组（2 层邻居组：节点的邻居和节点的邻居的邻居），如图 6-8 所示，被计算节点用蓝色圆圈标记，结算结果用加深的绿色标记显示。

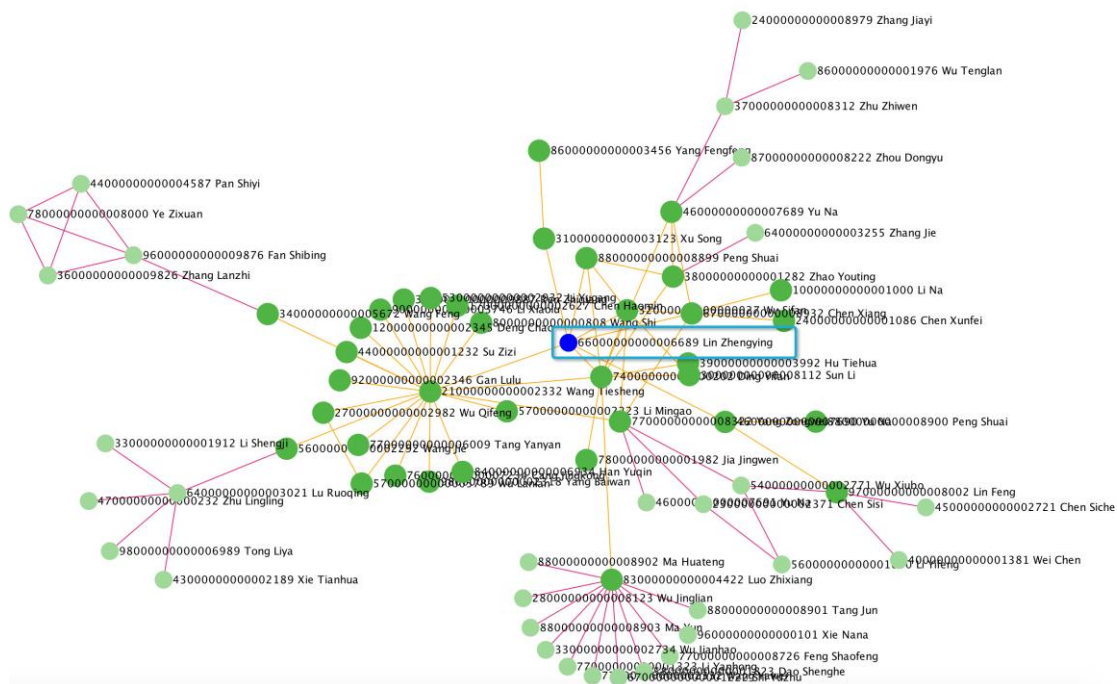


图 6-8 节点邻居计算可视化结果

6.4.4 顶点到顶点的简单路径

本小节对“顶点到顶点的简单路径”模块的功能进行了测试，图 6-9 显示了 ID 为 78000000000008000 的点和 ID 为 96000000000009876 的点之间的所有简单路径，一共是 5 条。

```
scala> simpleRoutesBetween(78000000000008000L,96000000000009876L)
78000000000008000-->36000000000009826-->96000000000009876
78000000000008000-->36000000000009826-->44000000000004587-->96000000000009876
78000000000008000-->44000000000004587-->36000000000009826-->96000000000009876
78000000000008000-->44000000000004587-->96000000000009876
78000000000008000-->96000000000009876
res5: List[Array[Long]] = List(Array(78000000000008000, 36000000000009826, 96000000000009876), Array(
78000000000008000, 36000000000009826, 44000000000004587, 96000000000009876), Array(78000000000008000,
44000000000004587, 36000000000009826, 96000000000009876), Array(78000000000008000, 44000000000004587
, 96000000000009876), Array(78000000000008000, 96000000000009876))
```

图 6-9 两点间简单路径的输出结果

可视化测试结果如图 6-10 所示，其中用深绿色标记的点是所有简单路径上涉及的点。

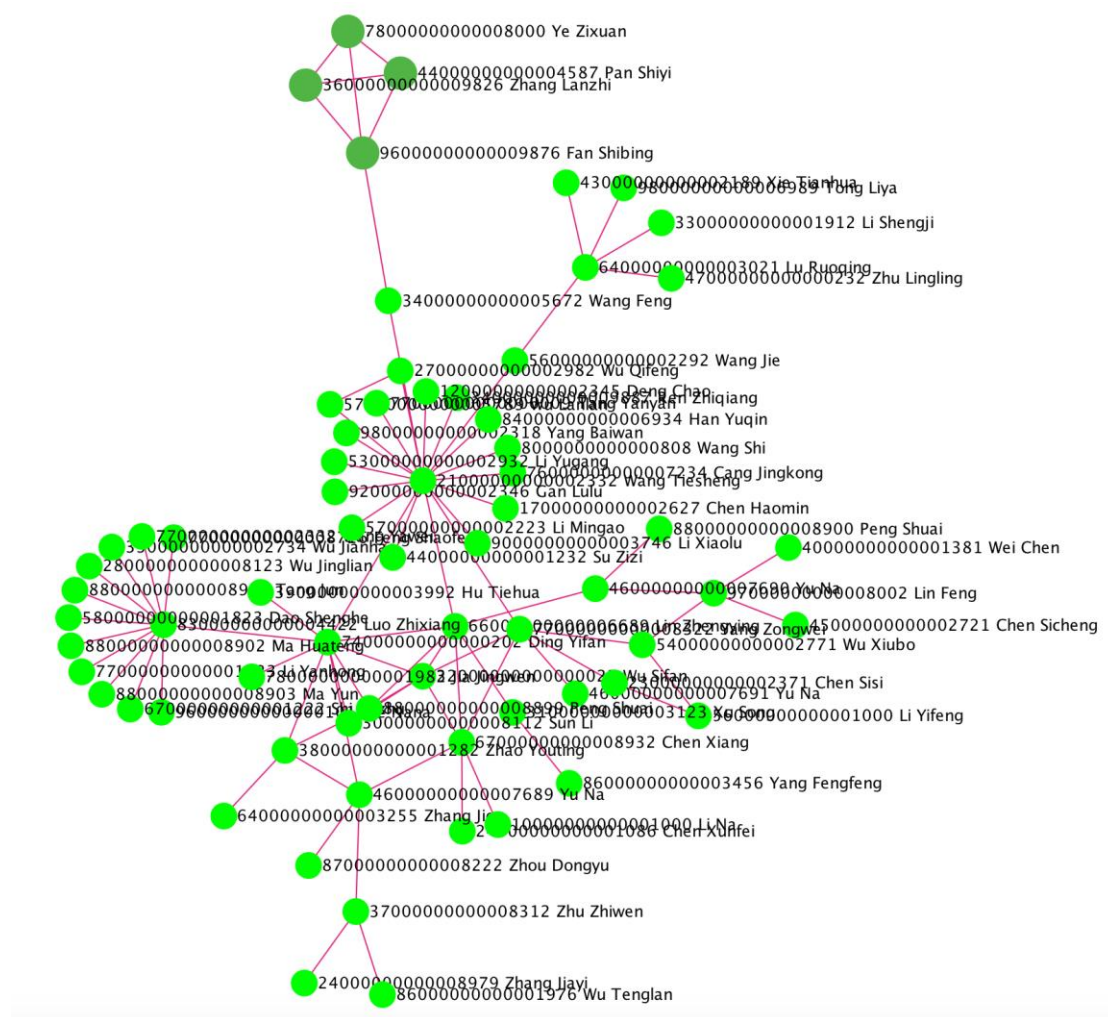


图 6-10 简单路径可视化结果

6.5 本章小结

本章对系统的测试进行了说明，通过测试数据集以及测试环境的说明，并用图片和表格的方式展示测试的结果，最后对这些测试结果进行分析说明，充分验证了系统功能的正确性。

第七章 总结与展望

7.1 论文总结

论文先从项目背景出发，介绍了图计算在社会网络分析方面的应用现状，然后详细介绍 Spark、GraphX、GraphStream、Scala 四种系统相关技术，在此基础上，论文用系统功能总图对系统的总体功能做出描述，并对主要需求作出了用例描述。再次，论文分概要设计和详细设计两个层次对系统的设计和实现进行了讲解。论文还对开发的系统进行了测试，通过测试结果对系统进行分析。最后论文进行了总结，并对下一步的工作计划进行了阐述说明。

7.2 工作展望

工作中还存在许多不足，有待进一步改善：

1. 将项目代码进行开源，编写使用手册。
2. 获取大规模的实际数据，并将系统部署到大规模集群上，测试系统进行大数据处理的性能。
3. 丰富算子的内容。
4. 进一步重构代码，提高可扩展性，方便后期维护。
5. 优化机器学习算法，提高分析的准确率。

关于图模型的 GraphX 实现工作还有很多可以做，可以分享。仅就商用系统而言，这些模型应用于用户网络的社区发现、用户影响力、能量传播、标签传播等，可以提升用户黏性和活跃度；而应用到推荐领域的标签推理、人群划分、年龄段预测、商品交易时序跳转，则可以提升推荐的丰富度和准确性。

我在中国科学院信息工程研究所攻读硕士期间，将在研究生导师的指导下进一步深入研究这个课题，完成以上列出的有待改进的工作。

参考文献

- [1]维基百科: Social Network[EB/OL].(2016-04-13). <https://zh.wikipedia.org/wiki/%E7%A4%B%E4%BC%9A%E7%BD%91%E7%BB%9C>
- [2]Zhou W, Han J, Gao Y, et al. An efficient graph data processing system for large - scale social network service applications[J]. Concurrency and Computation: Practice and Experience, 2014.
- [3]杨俊. 基于图自同构的 AK-Secure 社会网络隐私保护方法[D]. 东北大学,2013.
- [4]程学旗, 靳小龙, 王元卓, 等. 大数据系统和分析技术综述[J]. 软件学报, 2014, 25(9): 1889-1908.
- [5]王强,李俊杰,陈小军,黄哲学,陈国良. 大数据分析平台建设与应用综述.集成技术, 2016(3), 5(2):2-18.
- [6]易成岐, 鲍媛媛, 薛一波. 社会网络大数据分析框架及其关键技术[J]. 中兴通讯技术, 2014(1):5-10.
- [7]邓波, 张玉超, 金松昌,等. 基于 MapReduce 并行架构的大数据社会网络社团挖掘方法[C] // 中国计算机学会 ccf 大数据学术会议. 2013.
- [8]杨寅. X-RIME: 基于 Hadoop 的大规模社会网络分析.中国科技论文在线 (<http://www.paper.edu.cn>)
- [9]Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster Computing with Working Sets[J]. HotCloud, 2010, 10: 10-10.
- [10]Apache Spark[EB/OL]. <http://spark.apache.org/>.
- [11]Xin R S, Gonzalez J E, Franklin M J, et al. GraphX: a resilient distributed graph system on Spark[C]// International Workshop on Graph Data Management Experiences and Systems. ACM, 2013:1-6.
- [12] Gonzalez J E, Xin R S, Dave A, et al. Graphx: Graph processing in a distributed dataflow framework[C]//11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 599-613.
- [13]Xin R S, Crankshaw D, Dave A, et al. GraphX: Unifying data-parallel and graph-parallel analytics[J]. arXiv preprint arXiv:1402.2394, 2014.

- [14]黄明, 吴炜. 快刀初试:Spark GraphX 在淘宝的实践[J]. 程序员, 2014(8):98-103.
- [15]Dutot A, Guinand F, Olivier D, et al. Graphstream: A tool for bridging the gap between complex systems and dynamic graphs[C]//Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007). 2007.
- [16]Erich Gamma, Richard Helm, Ralph Johnson. 设计模式: 可复用面向对象软件的基础 (第 1 版) [M]. 北京: 机械工业出版社, 2007.
- [17]李稚楹, 杨武, 谢治军. PageRank 算法研究综述[J]. 计算机科学, 2011(10), 38(10A):186-188.
- [18]仇丽青, 陈卓艳. 基于 Pagerank 的社会网络关键节点发现算法[J]. 软件导刊, 2008(8), 13(8):48-49.
- [19]Xie J, Szymanski B K. Community detection using a neighborhood strength driven label propagation algorithm[C]//Network Science Workshop (NSW), 2011 IEEE. IEEE, 2011: 188-195.
- [20]Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. 算法导论 (第二版) [M]. 北京: 机械工业出版社, 2006.