分类号_____ 密级 _____

UDC _____

学 位 论 文

支持 why-not 问题解释的社团发现系统

作者姓名： 高静

指导教师： 王斌 副教授

东北大学计算机科学与工程学院

申请学位级别： 硕士 学科类别： 专业学位

学科专业名称： 计算机技术

论文提交日期： 2017 年 12 月 论文答辩日期：2017 年 12 月

学位授予日期： 2018 年 1 月 答辩委员会主席：乔建中

评阅人：杨晓春、石祥滨

东北大学

2017 年 12 月

A Thesis in Computer Technology

A Community Discovery System for Explaining Why-not Problems

By Gao Jing

Supervisor: Associate Professor Wang Bin

Northeastern University

December 2017

独创性声明

本人声明，所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：

日　　期：

学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人同意东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

作者和导师同意网上交流的时间为作者获得学位后：

半年□　　一年□　　一年半□　　两年□

学位论文作者签名：　　导师签名：

签字日期：　　签字日期：

摘 要

网络结构广泛存在于自然界和现实生活之中，近期研究发现很多领域的问题都能用复杂网络来表示。而随着对复杂网络更深入的研究，可以发现很多实际网络具有共同的结构特性—社团结构。研究网络中的社团结构可以在网络的局部结构上对节点与边的结构和功能有更好的理解，也可以以社团为基本单位来更好的理解整个网络的动态性。why-not 问题是查询处理中的一个重要问题，对 why-not 问题的查询结果中缺失数据作出解释已经成为数据库社区的热点研究内容。合理解释 why-not 问题，不仅仅能够提高数据库的可用性，在实际生活中也有很多重要的应用。但是目前的社团发现算法并不能解决 why-not 问题的解释。

综合上述问题，本文设计并实现一个支持 why-not 问题解释的社团发现系统。本文首先对社团发现算法的相关理论以及关键技术进行了系统综述和分析，并基于此，设计并实现了一个支持 why-not 问题解释的社团发现系统。该系统主要包括数据采集，数据处理，why-not 问题解释以及结果可视化展示，其中的核心问题是支持 why-not 问题解释的社团发现算法。基于经典的 pSCAN 社团发现算法，系统地探究了密度阈值和相似度阈值对 pSCAN 算法聚类结果的影响，并根据用户的 why-not 问题可以自适应地调节 pSCAN 算法的密度阈值和相似度阈值，以最小的修改代价对 pSCAN 算法中的 why-not 问题作出解释。

最后，本文对该系统进行部分展示，并在两个真实现有的数据集和一个从网络爬取的数据集上测试了该系统的性能。实验结果表明，本文提出的支持 why-not 问题解释的社团发现系统能有效地解释 pSCAN 算法中的 why-not 问题。

关键词：社团发现；pSCAN 算法；why-not 问题；密度阈值；相似度阈值

Abstract

Network structure is widely used in nature and real life. Through most of the current research, recently，it is found that many problems in the field can be expressed in complex networks. With more in-depth study of complex networks, it is found that many actual networks have common structural features - community structure. The study of the community structure in the network can have a better understanding of the structure and function of the nodes and edges on the local structure of the network. The community can also be used as a basic unit to better understand the dynamics of the entire network. And the Why-not problem is an important problem in query results. It has become a hot research topic in database community to explain the missing data in why-not query results. The rational explaination of the why-not problem not only is able to improve the availability of the database, but also has many important applications in real life. However, the community discovery system based on the current community discovery algorithm can not solve the why-not problem which the expected data points appearing in the community structure don't appear in the excepted clusters.

To sum up the above questions, this thesis aims to design and implement a community discovery system for explaining why-not problems. Firstly, the related theories and key technologies of community discovery algorithm are systematically reviewed and analyzed. Based on this, a community discovery system for explaining why-not problems is designed and implemented. The system mainly includes data acquisition, data processing, why-not problem explaination and result visualization display. The core problem is community search algorithm for explaining why-not problems. The classic pSCAN community discovery algorithm based on systematically explore the effect of density threshold and similarity threshold on the clustering results of pSCAN algorithm, density threshold and similarity threshold, and can adaptively adjust the pSCAN algorithm according to the why-not user, with the minimum cost of modification on pSCAN algorithm in why-not to explain.

Finally, the system is partly displayed, and the performance of the system is tested on two real data sets and a data set crawling from the network. The experimental results show that the community discovery system for explaining why-not problems in this thesis can effectively explain the why-not problem in the pSCAN algorithm.

Key words: Community Discovery; pSCAN Algorithms; Why-not Questions; Density Threshold; Similarity Threshold

目 录

第1章 绪 论

社团发现就是在将用户看成节点的用边来表示节点之间联系的网络结构中，能够将或疏或密的联系找出来并加以聚类的过程。而社团发现算法是如今的研究热点内容，其在现实生活中也有着非常重要的作用。但是现在提出的一系列社团发现算法还是不能够解决某一社团中为什么没有期望出现的节点，也就是 why-not 问题。所以本文将致力于解决 why-not 问题的社团发现系统的设计与实现。本章首先介绍该研究的背景，接着提出本文的研究内容，最后给出本文的组织结构。

1.1 研究背景

网络结构广泛存在于自然界和现实生活之中[1]，例如现有的技术相关网络，比如Internet、路网、社交及通信网络等；再比如自然界中存在的食物链关系网络、疾病基因网络等。虽然这些网络结构分别来自不同的领域，有着不同的应用背景，但是在结构上却表现出了非常相似的特征，那就是这些网络结构中的节点之间的连接关系模式不是单纯随机的，也不是单纯的有规则的，而是具有以下特点的：“小世界性”(Small-World)[2]、“高聚集系数”(High Clustering Coefficient)[1]、“自相似性”(Self-Similarity)[4]、“无尺度特性”(Power-Law)[3]等。当然，也存在着网络的连接结构会随时间变化，或者具有权值和方向这样的状况。随着研究的深入，“复杂网络”(Complex Networks)[1,5,6]这一名词应运而生。钱学森给出的复杂网络定义，复杂网络表示的就是具有自组织，自相似，吸引子、小世界、无标度中部分或全部性质的网络[29]。

通过现在的大多研究发现很多领域的问题都能用复杂网络来表示。而随着对复杂网络更深入的研究，可以发现很多实际网络都有共同的结构特性—社团结构。社团结构中的整个网络是由若干个“群”或者“团”构成的。团内部的节点之间有着紧密的联系，而两个团之间联系则比较稀疏[7]。研究网络中的社团结构可以在网络的局部结构上对节点与边的结构和功能有更好的理解[8]。也可以以社团为基本单位来更好的理解整个网络的动态性。最重要的是，对于真实世界中规模庞大并且链接稀疏的数据，为其提供了不需要依赖数据的其他属性仅仅依靠链接关系而聚类的方法。

社团结构已经被认为是复杂网络中最重要的拓扑结构属性之一，因为他揭示了复杂网络的隐藏规律和行为特征[9,10]。目前，社团结构的发现已经应用到社交网络分类，社区挖掘和搜索引擎等领域中。如今已经提出了很多社团发现算法，根据可以解决的网络

结构类型分为两类非重叠社团发现算法和重叠社团发现算法。非重叠社团发现算法指的是被聚出的社团之间是互不重叠的，每个节点属于且仅属于一个社团。根据采用的社团发现策略非重叠社团发现算法大致分为以下四种：基于模块度优化的[13]、基于谱分析的、基于信息论的[11,12]非重叠社团发现算法等；顾名思义重叠社团发现算法就是社团之间是有重叠的，某些节点同时具有多个社团的特性，相应的重叠社团发现算法可划分为以下几种：基于团渗透改进的[10]、基于种子扩散思想的、基于混合概率模型的、基于边聚类的重叠社团发现算法等。对于非重叠社团的划分算法虽然比较成熟，但是它能处理的问题却比较理想化。对于重叠社团发现算法来说虽然提出了划分算法，但是效果其实并不如人意。非重叠社团发现算法中一个划分算法称为 SCAN[14]算法，是由机器学习里的基于密度的聚类算法 DBSCAN[17]改进而来的非重叠社团发现算法，具有线性时间复杂度。它并未基于上述的四个分类，但是相较于介绍过的算法而言，SCAN 算法最大的亮点在于能发现社团中的桥节点(hub)和离群点(outlier)，pSCAN[15]算法则是通过 SCAN 算法的优化。SCAN 算法和 pSCAN 算法可以将网络结构中的拓扑结构表现的更全面。

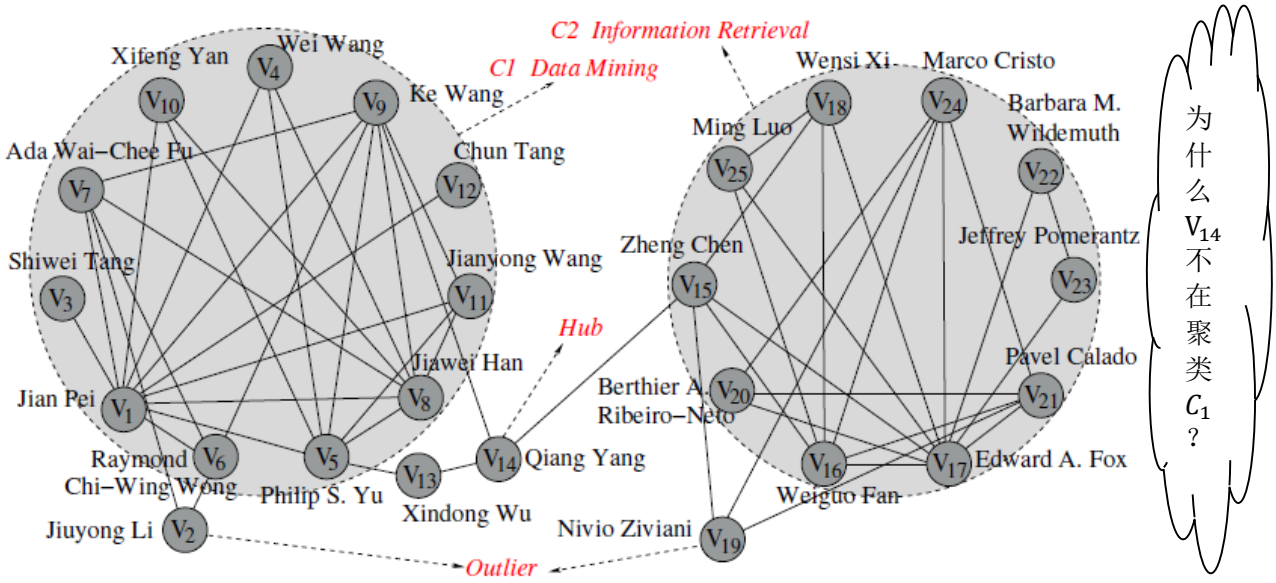


图 1.1 why-not 问题现象

Fig. 1.1 The phenomenon of why-not problem

社团发现可以看作是图聚类问题。聚类是人的直观感觉，即将紧密相连的数据作为一类，对于社团发现算法来讲，它的输入参数都是人为根据经验进行确定的，这样就导致社团分类并不灵活，有人就会提出如图 1.1 这样的问题：为什么期望的某个对象点没有被划分到期望社团中？所以提出这样的构想：是否存在一种解决方法，可以解决这类问题，通过修改社团算法的查询参数，让期望对象点能够被划分到期望社团聚类中。

数据库在过去几十年的发展中，其性能方面有很大的提高。虽然性能的提高使得同构或者异构的数据库可以进行各种复杂查询，但是其可用性还是有很大程度不能满足用户的需求。很多用户对于数据库不能够解释为什么数据库查询结果中没有得到预期的答案感到很失望。就会提出类似图 1.1 中的问题，为什么某些期望的数据没有出现在查询结果集中呢？类似于这样的问题被称为 why-not 问题查询[16]。Why-not 问题查询首次被数据库专家 Jagadish[16]提出，起初被应用到数据库社区中。这个查询是想知道为什么一些期望的数据没有出现在查询结果中，怎样做能够使得期望的数据出现在查询结果集中[19]。因此为 why-not 问题查询提供解释就可以为用户对他们在数据库中的到的查询结果提供有效的解释和说明。就像 Jagadish 等指出的，数据库不能对用户在查询结果中没有得到预期答案的情况作出解释，这使很多用户感到失望[16]。用户在遇到这种情况时，可能会产生这样的想法，为什么自己期望的元组没有出现在查询结果集中？而这类问题就被称为 why-not 问题。那么若数据库能够针对 why-not 问题给出合理的解释，则会帮助用户更好的理解原始的查询，以及可以知道如何去改变查询使得数据的可用性得到提高。因此如何帮助用户高效的解释 why-not 问题成为当前数据库管理系统方面研究的热点内容。

图 1.1 提出的社团发现相关的问题属于社团发现算法中的 why-not 问题，而这个 why-not 问题在现实生活中很普遍。社团发现这一过程，很多用户对于社团发现的结果集中出现的某个对象点未出现在期望社团中的现象很失望。并且目前并没有能够解决这一问题的研究。所以本文将社团发现算法与 why-not 问题查询相结合，设计并实现一个能够支持 why-not 问题解释的社团发现系统，解决社团发现中出现的 why-not 问题，通过精炼社团发现算法的社团查询参数，使得社团发现结果更加符合用户的要求。

综上所述，社团结构是复杂网络中拓扑结构属性之一，它是网络中的一群阶段或边的集合，一般意义上也可被称为团、群或模块，其特征是社团内部的节点间连接相对紧密，而社团之间的连接相对稀疏。在处理现实世界中大规模并且链接稀疏的数据时，提供仅仅只依靠链接关系不需要依赖数据的其他属性就能划分社团的方法。SCAN 聚类算法是由基于密度的聚类算法 DBSCAN 改进来的，属于非重叠社团发现算法的一种，具有着其他非重叠社团发现算法没有的优势，即可以分辨出社团中的离群点和桥节点。另外，why-not 问题研究能够对查询结果中缺失的数据作出解释，是数据库社区中热门研究方向，能够在很大程度上提高聚类结果的可用性。所以本文将社团发现算法与 why-not 问题解释相结合，完成一个支持 why-not 问题解释的社团发现系统。

1.2 本文研究内容

尽管社团发现算法的相关技术研究目前已经相对成熟，但是还存在着本文第 1 章中如图 1.1 所提出的问题，即 why-not 问题。所以本小节主要介绍本文的主要研究内容，将社团发现算法与 why-not 问题解释相结合解决社团发现中的 why-not 问题，进而实现 SWPI(Supporting Why-not Problem Interpretation)社团发现系统，即支持 why-not 问题解释的社团发现系统。本文的主要工作内容包括以下几个方面：

(1) 对系统在功能、性能、可行性以及系统架构方面进行需求分析，并制定系统的设计方案。

(2) 对系统进行总体设计、详细设计，并对每个模块进行详细的设计与说明。

(3) 实现数据采集功能，通过一定的数据采集技术将数据从指定的数据源采集过来。

(4) 实现采集后数据处理功能，将获取出来的图数据里含有不需要的数据信息进行过滤，将需要的数据信息留下并用一定的数据格式进行存储。

(5) 实现社团发现算法的 why-not 问题解释，此部分是本系统的主要研究内容。

(6) 将社团发现的聚类效果可视化的显示出来。

1.3 本文组织结构

根据上述研究内容，本文共分六章：

第 1 章为绪论，介绍要解决问题的研究背景，提出本文的研究内容和面临的挑战。

第 2 章为背景知识与相关工作，介绍了 pSCAN 算法以及 SCAN 算法的具体思想，并给出了基于 pSCAN 算法的 why-not 问题的定义。

第 3 章介绍系统的分析，主要从需求分析和功能分析以及界面分析三个方面进行，然后进行了系统的设计，从总体设计和详细设计两个方面进行介绍。

第 4 章因为一共四个功能模块其中三个功能模块可以通过现有的技术进行实现，所以第 4 章主要进行 why-not 问题解释功能模块的算法介绍

第 5 章为聚类系统的实验与实现分析，首先对 pSCAN 聚类中的 why-not 问题算法进行实验分析。通过在两个真实的现成的数据集和一个爬成的数据集上进行大量实验，测试本文所提出的支持 why-not 问题解释的社团发现算法的时间效率，然后然后进行了系统测试，主要进行 why-not 解释问题的功能测试。

第 6 章为总结与展望，对本文所做的工作进行总结，并提出未来的研究内容。

第2章 相关理论与关键技术

本章主要对社团发现算法的相关理论以及关键技术进行系统综述和分析，包括网络爬虫技术，社团发现算法的相关研究，why-not 问题解释研究现状、以及能够实现结果可视化展示的可视化工具。

2.1 网络爬虫技术

2.1.1 现有爬虫技术

本社团发现系统所用数据利用网络爬虫技术来获得。现如今的网络爬虫按照系统的结构与实现技术可以分为四种类型：聚焦网络爬虫、通用网络爬虫、增量式网络爬虫、深层网络爬虫。

聚焦网络爬虫是指可以选择性的对预先定义好的主题相关网页进行爬取的网络爬虫。它具有的特点就是只需要爬取与主题相关的页面，这样在很大的程度上节省了硬件及网络资源，保存的页面因为数量少是的更新更快。达到能够满足一些特定人群对特定领域信息的需求[18]。

通用网络爬虫主要的思想就是其爬行对象是从一些种子 URL 扩充到整个 web 的，主要为门户网站搜索引擎与大型 web 服务提供商提供服务，因为爬行页面数据量过大刷新页面太多，所以对硬件存储空间要求比较高，以为它能够为搜索引擎搜索广泛主题，这使得他具有更好的应用价值。

增量式网络爬虫能够将已经下载的网页进行增量式更新或者它只爬取新产生或已发生变化的网页信息。也就是说这种网络爬虫只会在需要的时候爬行新产生或发生更新的网页，这样会减少数据量下载，进而减少时间和空间消耗，算法实现难度也有所增加。

深层网络爬虫就是对深层的其内容不能通过静态链接获取的，隐藏在表单背后只有在用户提交特定的关键之后才能获得的 web 网页进行网页爬取。2000 年 Bright Planet 指出：深层网络中可访问信息容量是表层网络的几百倍，是互联网上最大、发展最快的新型信息资源[19]。

本文中采用通用网络爬虫类技术，因为通用网络爬虫具有较强的应用价值。本文采用 Scrapy 框架，它是由 Python 编写的爬虫框架，因为 Scrapy 比较流水化，各部分的分工更加清晰，Scrapy 的基础学起来更容易。

2.1.2 Scrapy 框架介绍

（1） Scrapy 组件

1) 引擎：用来处理整个系统的数据流处理，触发事务(框架核心)；

2) 调度器：用来接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回。可以想像成一个 URL（抓取网页的网址或者说是链接）的优先队列，
由它来决定下一个要抓取的网址是什么，同时去除重复的网址；

3) 下载器：用于下载网页内容，并将网页内容返回给蜘蛛(Scrapy 下载器是建立在 twisted 这个高效的异步模型上的)；

4) 爬虫：爬虫是主要干活的，用于从特定的网页中提取自己需要的信息，即所谓的实体。用户也可以从中提取出链接，让 Scrapy 继续抓取下一个页面；

5) 项目管道：负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，
将被发送到项目管道，并经过几个特定的次序处理数据；

6) 下载器中间件：位于 Scrapy 引擎和下载器之间的框架，主要是处理 Scrapy 引擎与下载器之间的请求及响应；

7) 爬虫中间件：介于 Scrapy 引擎和爬虫之间的框架，主要工作是处理蜘蛛的响应输入和请求输出；

8) 调度中间件：介于 Scrapy 引擎和调度之间的中间件，从 Scrapy 引擎发送到调度的请求和响应。

（2） Scrapy 的运行流程

首先引擎先从调度器中取出一个链接(URL)用于接下来的抓取，其次引擎把取出来的 URL 封装成一个请求(Request)传给下载器，这样再由下载器把资源下载下来，并封装成应答包(Response)。然后爬虫解析 Response，解析出来的是实体（Item），那么交给实体管道进行进一步的处理，若解析出的是链接（URL），则把 URL 交给调度器等待抓取。

2.2 社团发现算法

在第 1 章中已介绍过社团结构的定义，所以在本小节将介绍现有的社团发现算法以及他们的优缺点，选择出本社团发现系统将使用的社团发现算法。进而详细介绍所选择的社团发现算法。有多种分为非重叠社团发现算法和重叠社团发现算法。

2.2.1 现有社团发现算法

识别网络中的社团结构，对于分析网络特征和了解网络结构都很重要。目前来讲，社团结构发现已经被用于很多领域，如社交网络、web 社区挖掘以及搜索引擎等。因此，关于社团结构的快速有效地识别发现方法以及其内在的动力演化机制的这两方面研究，对揭示网络局部功能与其自身拓扑结构之间相互影响的关系都具有着很重要的作用。

目前有很多算法用于社团结构的发现，俗称社团发现算法。这些算法大致可分为两类：非重叠社团发现算法和重叠社团发现算法。杨博等人对这些社团发现算法进行了很好的综述[19]。例如，谱方法是基于优化的方法，它将网络聚类问题转化成二次型优化问题，通过计算特殊矩阵的特征向量来优化预定义的“截(cut)”。而 GN 算法[21]则是一种启发式算法，它的启发式规则为社团间连接的边介数应大于社团内连接的边介数。杨楠等人提出基于流的社区特征和马尔可夫图形聚类算法(MCL)的团结合的方法去查找潜在的 Web 社区[20]。将镜像或近似镜像页面的删除放在图形聚类之后，大大减少了比较的代价。然后，在聚类簇的基础上，使用判定每个团内元素的筛选算法产生可能的社区候选集合。

重叠社团发现算法研究相对于非重叠社团发现算法并不成熟，算法实现的效率并不高。非重叠社团发现算法的研究已经相对成熟，但是对网络结构的划分并不彻底，而 pSCAN 算法虽是非重叠社团发现算法的一种，但其最大的亮点就在于可以发现社团结构中的桥节点和离群点。并且其主要的思想是，在考虑两个节点之间关系时，考虑的不仅是他们之间的直接联系，还会考虑他们的邻居节点，也就是说节点是根据它们共享邻居方式去进行社团划分。因其存在的优势，本文我们选择 pSCAN 社团发现算法作为社团发现系统的实现算法。因为 pSCAN 算法是对 SCAN 算法的优化，所以为了更好的理解 pSCAN 社团发现算法，需要先介绍 SCAN 聚类算法，然后再进行 pSCAN 社团发现算法的介绍。

2.2.2 SCAN 聚类算法

SCAN 算法是由机器学习里的基于密度的聚类算法 DBSCAN 改进而来的一种非重叠社团发现算法，处理基于图结构的结构聚类，具有线性时间复杂度。其一大亮点在于能发现社团中桥节点(hub)和离群点(outlier)。主要思想在于，在考虑两点之间的关系的时候，不仅考虑它们的直接链接，而是利用它们的邻居节点来作为聚类的标准。

(1) SCAN 算法相关定义

SCAN 聚类算法关注的是无加权值的无向图$G=(V,E)$ [文献]，在这里 V 是点的集合，E 是边的集合。而|V|，|E|分别用 n，m 表示点的个数和边的个数。利用$(u,v)\in E$ 表示 u 到 v 之间的边，u(或 v)也可以别说成 v(或 u)的邻居。基于此，下面进行 SCAN 算法相关定义的介绍：

定义 2.1 节点相似度($\sigma(u,v)$)[15]。 两个节点共同邻居的数目与两个节点邻居数目的几何平均数的比值（这里的邻居均包含节点自身）。

$$\sigma(u,v)=\frac{|N[u]\cap N[v]|}{\sqrt{d[u]\cdot d[v]}} \tag{2.1}$$

其中，$N[u]$表示节点 u 及其相邻节点所组成的集合，而$d[u]$ 即为$|N[u]|$。

定义 2.2 ε - 邻居[15]。给定一个相似度阈值$0<\varepsilon\leq 1$，点 u 的 ε - 邻居定义为与其相似度不小于 ε 的节点所组成的集合，表示为$N_\varepsilon[u]$。

$$N_\varepsilon[u]=\{v\in N[u]|\sigma(u,v)\geq\varepsilon\} \tag{2.2}$$

定义 2.3 核节点[15]。是指 ε - 邻居的数目大于 μ 的节点。

$$CORE_{\varepsilon,\mu}(v)\Leftrightarrow|N_\varepsilon[u]|\geq\mu \tag{2.3}$$

定义 2.4 直接可达[15]。节点 u 是核节点 v 的 ε -邻居，那么称从 v 直接可达 u。

$$DirREACH_{\varepsilon,\mu}(u,v)\Leftrightarrow CORE_{\varepsilon,\mu}(v)\cap w\in N_\varepsilon[u] \tag{2.4}$$

定义 2.5 可达[15]。 节点 u 可达 v，当且仅当存在一个节点链$u_1,\cdots,u_n\in V,u_1=u,u_n=v,$ 使得u_{i+1}是从u_i直接可达的。

$$REACH_{\varepsilon,\mu}(u,v)\Leftrightarrow\exists u_1,\cdots,u_n\in V:u_1=u\cap u_n=v\cap\forall i\in$$

$$\{1,\cdots,u_n,\}:DirREACH_{\varepsilon,\mu}(u_i,u_{i+1}) \tag{2.5}$$

定义 2.6 相连[15]。若核节点 o 可达节点 u 和节点 v，则称节点 u 和节点 v 相连.

$$CONNECT_{\varepsilon,\mu}(u,v)\Leftrightarrow\exists o\in V:REACH_{\varepsilon,\mu}(o,u)\cap REACH_{\varepsilon,\mu}(o,v) \tag{2.6}$$

定义 2.7 桥节点（hub）。与至少两个聚类相邻的孤立节点。

定义 2.8 离群点（outlier）。只与一个聚类相邻或不与任何聚类相邻的孤立节点。

定义 2.9 聚类（cluster）。 一个聚类 C 是有至少两个点的 V 集合的子集且满足：

1) 最大性：如果一个核心点$u\in C$，那么从 u 可达所有节点都属于 C。

2) 连通性：对于任意聚类 C 中的两个点v_1和v_2，都存在一点 o，使得v_1和v_2与 o 相连

算法 2.2 SCAN 聚类算法

输入：结构图$G=(V,E)$，相似性阈值ε，密度阈值μ

输出：聚类 *C* 的集合

```
SCAN((V, E), ε, μ)
    for each unclassified vertex v ∈ V do
        if CORE ε, μ(v) then
            generate new clusterID;
            insert all x ∈ N_ε(v) into queue Q;
            while Q ≠ 0 do
                y = first vertex in Q;
                R = {x ∈ V | DirREACH_{ε,μ}(y, x)}
                for each x ∈ R do
                    if x is unclassified or non-member then
                        assign current clusterID to x;
                    if x is unclassified then
                        insert x into queue Q;
                remove y from Q;
        else
            label v as non-member;
    end for.
    for each non-member vertex v do
        if (∃ x, y ∈ N(v) ( x.clusterID ≠ y.clusterID) then
            label v as hub
        else
            label v as outlier;
      end for.
end SCAN.
```

(2) SCAN 算法执行过程及分析

因为 SCAN 算法是由 DBSCAN 算法演变过来，应用到结构图上的聚类算法。因此其思想与 DBSCAN 类似，给出该算法描述：

1) 对于每个未分配社团的节点 v，计算点 v 与其邻居节点的结构相似性，检查 v 是否是核节点。若是，则将其直接可达节点分配到一个聚类中（聚类标号记为该节点的值），同时也将其 ε-邻居放进队列中，重复进行 1 步骤。

2) 若 v 不是，则将其标志为 non-member。

3) 最后检查所有的 non-menber 节点，若其相邻节点存在于两个及以上的聚类中，则将其标为桥节点(hub)节点，否则标为离群点(outlier)。算法 2.2 给出了 SCAN 聚类算法的伪代码。

因 SCAN 算法是 DBSCAN 的变形，所以很大程度上也具有着 DBSCAN 算法的优缺点。但在应用上来说，其能够实现结构图的聚类，在线性时间内以基于密度的思想完成社团发现聚类，并能够发现社团内的桥节点和离群点，这也是其特有的一大优点。但是 SCAN 算法存在着不足，他会将每两个点的结构相似性都算一遍，并且还会出现重复计算的缺点，这使得算法的复杂度变得很高。因此，出现了 pSCAN 算法能够解决以上问题。

2.2.3 pSCAN 聚类算法

pSCAN 算法也是处理结构图上的聚类。他为了解决 SCAN 算法出现的结构相似性全计算以及重复计算而使得 SCAN 算法在大图上没有扩展性的这一缺陷，提出了两步模式。在此模式中，第一步，将核心点进行聚类；第二步，将非核心点分配给其邻居是核心点，且要满足该非核心点与其要分配的核心点之间要满足结构相似性的核心点所在的聚类里。在此模式的基础上，提出了剪枝算法—pruned SCAN，以减少结构相似性的全计算以及重复计算，即 pSCAN 算法。

(1) pSCAN **算法相关定义**

pSCAN 算法是在 SCAN 算法已给出的相关定义基础上新引进了两个新的定义分别为相似的度以及有效的度。

定义 2.10 相似的度(similar-degree)[15]**。**表示为 $sd(u)$，是 u 节点与其邻居节点能确定的结构相似性的节点个数。

定义 2.11 有效的度(effective-degree)[15]**。**表示为 $ed(u)$，是 u 点的度减去 u 与其邻居节点能确定的非结构相似性的节点个数。

(2) pSCAN **算法思想**

pSCAN 算法通过以上提出的两个定义，以及两步模式来提高算法的执行效率。其具体思想就是根据两步模式。首先进行第一步，遍历所有点中未访问的点按照 SCAN 算法的思想先将核心点进行聚类；第二步，对核心点聚类完毕后，根据核心点集将与其相连的非核心点通过判断是否节点相似而收入所属聚类。先判断若该节点是核心点，并且 $sd(u) \geq \mu$，则将其直接可达节点分配到一个聚类中（聚类标号记为该节点的

值），并且，同时也将其 ε-邻居放进队列中，重复进行 1 步骤直到所有的核心点都聚完。 对于每个已经聚好的聚类，由核心点向外延伸遇到未聚类的非核心点，判断其两点之间若是符合结构相似性则拉入相应核心点所在聚类，并标记此非核心点以核心所在聚类的标号，即聚类非核心点。具体算法实现如算法 2.3 所示。

算法 2.3 pSCAN 聚类算法

输入： A graph $G = (V, E)$, and parameters $0 < \varepsilon \leq 1$ and $\mu \geq 2$

输出：The set C of clusters in G

```
1.  /* Step-1: cluster core vertices */
2.  Initialize a disjoint-set data structure with all vertices in V;
3.  for each vertex u ∈ V do
4.      sd(u) ← 0; /* Initialize similar-degree */;
5       ed(u) ← d[u]; /* Initialize effective-degree */;
6.  end for
7.  for each vertex u ∈ V in non-increasing order w.r.t. ed(u) do
8.      CheckCore(u); /* Check if u is core vertex */;
9.      if sd(u) ≥ μ then ClusterCore(u);
10. end for
11. Cc ← the set of subsets of core vertices in the disjoint-set data structure;
12. /* Step-2: cluster non-core vertices */
13. ClusterNoncore();
14. return C;
```

首先将图中的所有的点进行初始化，然后将所有的点的 *sd(u)*以及 *ed(u)*进行初始化(2-5 行)，对于图中的以 *ed(u)*的值降序排列的每个节点，检查该点是否是核心节点进行对于每个未分配社团的节点 *u*，进行计算点 *v* 与其邻居节点的结构相似性，检查 *v* 是否是核节点(6,7 行)，而检查一个点是否为核心点的的伪代码如算法 2.4 所示。进行核心点的判断之后，在判断 *u* 节点的 *sd(u)*是否大于等于μ，然后将其进行核心点的聚类(8,9 行)，核心点的聚类算法如算法 2.5 所示。最后将确定的非核心点符合条件的分配到适合聚类中(10,11 行)。最终将聚类集合返回(第 12 行)。

算法 2.4 描述的就是检查节点 *u* 是否是核心点的算法。该算法通过 *sd(u)*以及 *ed(u)* 这两个值判断节点 *u* 是否为核心点。应用到的判断策略就是，若 $sd(u) \geq \mu$，则节点 *u* 一定是核心点；若 $ed(u) < \mu$，则节点 *u* 一定不是核心点。

算法 2.4 CheckCore(u)

```
1.  if ed(u) ≥ μ and sd(u) < μ then
2.      ed(u) ← d[u]; sd(u) ← 0;
3.      for each vertex v ∈ N[u] do
4.          Compute σ(u, v);
5.          if σ(u, v) ≥ ε then sd(u) ← sd(u) + 1;
6.          else ed(u) ← ed(u) - 1;
7.          if vertex v has not been explored then
8.              if σ(u, v) ≥ ε then sd(v) ← sd(v) + 1;
9.              else ed(v) ← ed(v) - 1;
10.         if ed(u) < μ or sd(u) ≥ μ then break;
11.     end for
12. Mark v as explored;
```

如果节点 u 没有明确其是核心点并且也没有明确其一定不是核心点，那么将其 $ed(u)$赋值为 $d[u]$同时将 $sd(u)$ 赋值为 0(1,2 行)。计算节点 u 与其所有邻居节点之间的节点相似度，判断如果其节点相似度大于等于相似度阈值 ε，节点 u 的 $sd(u)$值加 1，否则说明节点 u 与某邻居节点 v 结构不相似，则节点 u 的 $ed(u)$值减 1(3-6 行)。并且判断如果 v 节点没有被访问过，相应的将节点 v 的 $ed(v)$与 $sd(v)$值进行与 u 同样的处理(7-9 行)。在遇到 $ed(u) < \mu$ 或者 $sd(u) \geq \mu$ 时说明能够确定节点 u 是核心点或者其一定不是核心点，能够确定时就跳出循环(第 10 行)。最后将节点 v 设为被访问。

算法 2.5 ClusterCore(u)

```
1.  N'[u] ← {v ∈ N[u] | σ(u, v) has been computed};
2.  for each vertex v ∈ N'[u] do
3.      if sd(v) ≥ μ and σ(u, v) ≥ ε then union(u, v);
4.  end for
5.  for each vertex v ∈ N[u]\ N'[u] do
6.      if find-subset(u) , find-subset(v) and ed(v) ≥ μ then
7.          Compute σ(u, v);
8.          if vertex v has not been explored then
9.              if σ(u, v) ≥ ε then sd(v) ← sd(v) + 1;
10.             else ed(v) ← ed(v) - 1;
11.         if sd(v) ≥ μ and σ(u, v) ≥ ε then union(u, v);
12.     end for
```

算法 2.5 描述的就是如何对节点 u 进行聚类的算法。如果已经计算过节点 u 与其邻居节点 v 之间的节点相似度，则将节点 v 并入$N'[u]$的集合(第 1 行)。对于计算过节点相似度的 u 的每一个邻居节点 v，如果节点 v 是核心点并且节点 u 与节点 v 满足节点相似，则将节点 v 以及边(u,v)加入聚类中(2,3 行)。对于没有在$N'[u]$集合中的节点 u 的邻居节点 v，如果节点 v 还没有被聚到 u 所在聚类中并且并没有确定节点 v 不是核心点，那么计算节点 u，v 的节点相似度(4-6 行)。如果 u 与 v 满足节点相似 $sd(v)$的值加 1，否则 $ed(v)$的值减 1(7-9 行)。最后再进行判断如果节点 v 是核心点并且节点 u 与节点 v 满足节点相似，则将节点 v 以及边(u,v)加入聚类中，核心点聚类的算法结束。

算法 2.6 描述的是对非核心点的聚类过程，简单的讲就是利用与非核心点相连的核心点，判断他们之间是否为节点相似，如果节点相似，那么该非核心点就可以被聚到与其节点相似的核心点所在的聚类中。其具体伪代码描述如算法 2.6 所示。

算法 2.6 ClusterNoncore()

输入： 核心点聚类集合 Cc

输出： 聚类集合 C

```
1.  C ← ∅;
2.  for each Cc ∈ Cc do
3.      C ← Cc
4.      for each u ∈ Cc do
5.          for each v ∈ N[u] do
6.              if sd(v) < μ and v ∉ C then
7.                  Compute σ(u, v) if it has not been computed;
8.                  if σ(u, v) ≥ ε then C ← C ∪ {v};
9.          end for
10.     end for
11.     C ← 𝑪 ∪ {C}
```

先将最终的聚类结果集合 C 赋值为空(第 1 行)。对于核心点聚类集合中的每一个子集合 Cc，先将 Cc 赋值给集合 C 中(2,3 行)。对于核心点聚类子集合 Cc 中的任意核心点 u，节点 u 的每一个邻居节点 v 进行判断，如果节点 v 不是核心点并且节点没有被加入到集合 C 中，且若节点相似度$\sigma(u, v)$未被计算则计算，计算完成后若节点 u 与 v 节点相似则将点 v 放入集合 C 中(4-8 行)。将新聚好的非核心点聚类集合并入到结果集合 $\boldsymbol{C}$ 中(第 9 行)。

2.3 Why-not 问题相关研究

数据的溯源/世系探索了查询结果中数据的产生，及其随着时间的推移而变化的过程。他可以使用户了解到数据出现在查询结果中的原因以及过程。但是大部分溯源技术只能解释 why 问题，即查询结果为什么会出现在结果集中，却无法说明为什么查询结果中未出现某些期望数据的问题，这也就是本文的研究方向—why-not 问题。

目前，why-not 问题查询的解释模型有多种。文献[16,22-24]阐述了不同数据和查询集中存在的 why-not 问题：在文献[22]中，Chapman 等人提出了能够识别导致 why-not 元组被过滤掉的操作的方法。Tran 和 Chan 在 Chapman 的基础上，提出了从用户的反馈中收集 why-not 元组，再执行查询重写操作，解释了 SPJA(选择-投影-连接-联合-聚合)查询中的 why-not 问题[23]；在文献[25]中，Huang 等人将对原始数据集的修改作为研究重点，再通过修改原始数据中的元组值，使得 why-not 元组出现在 SPJA 查询结果集中，完成了 SPJA 查询中的 why-not 问题解释。文献[24]中，He 等人为解释 top-k 查询上的 why-not 问题，根据用户的偏好去修改 k 值和权值。文献[24]解决了反向 skyline 查询中的 why-not 问题。在文献[27]中，Islam 等人提出了一种叫做 FlexIQ 的 SPJ(选择-投影-连接)查询重写框架，它着重考虑了用户反馈的 why 和 why-not 问题。在文献[28]中实现了一个基于因果关系分析查询处理的框架模型。以上的 why-not 问题解释框架，主要是基于数据库的， 而在文献[27]中提出了新的 why-not 解释方法，该文献在文献[26]的算法基础上将空间关键字与 why-not 结合起来，采用分别修改查询点，修改阈值 K，修改阈值 K 与权重向量集 W，权重向量集 W 和查询点来解决空间关键字的 why-not 问题。

但是目前并没有社团发现中的图聚类 why-not 问题解释的相关研究，所以本文将社团发现算法与 why-not 问题解释结合起来，实现一个支持 why-not 问题解释的社团发现系统。

2.4 社团发现可视化工具

Gephi 是基于模块化的思想设计的，其真正的体现了高内聚低耦合的思想。每个模块负责不同的职责，比如有专门负责图形结构的 Graph，有专门用于布局的 Layout。其最大的特点就是它强大的显示功能，能够把网络图现实的很漂亮。所以本系统的可视化部分基于 Gephi 进行开发，因为本系统需要把可视化结果内嵌到网页中，所以 Gephi 软件需要安装 sigmajs 插件。

2.5 本章小结

本章主要介绍了网络爬虫技术、社团发现算法、pSCAN 聚类算法、why-not 问题以及社团发现可视化工具相关方面的研究及涉及到的理论知识和关键技术。针对本文研究的内容，首先介绍了网络爬虫技术，选定本系统数据采集的所用到的网络爬虫技术，然后介绍关于社团发现算法的相关研究，并选定 pSCAN 算法作为本系统社团发现算法的实现算法。最后说明了本系统能够将社团发现结果可视化的可视化工具。

第3章 系统分析与设计

本章首先对支持 why-not 问题解释的社团发现系统的分析进行论述。先对系统进行需求分析，并确定该系统的功能需求和性能需求。再对系统进行界面和可行性分析，之后确定本系统的架构。然后以需求分析和可行性作为基础，确定设计目标，结合实际开发中的软硬件环境，对系统中已经确定的功能模块进行设计，并对各模块的具体内容进行说明。

3.1 系统分析

本小节将先对支持 why-not 问题解释的社团发现系统进行需求分析，首先，确定该系统的功能需求和性能需求。其次，分析系统的界面与可行性，之后确定本系统的框架。

3.1.1 需求分析

(1) 功能需求

本系统的目标是能够开发一个支持 why-not 问题解释的社团发现系统。首先，本系统需要获取数据源，所以需要数据的采集。其次，因为采集出来的数据会含有无用的信息，使得采集数据并不符合系统需求的数据格式，所以本系统需要进行数据处理，获取得到本系统需要的数据。然后，最核心就是实现对社团发现算法进行 why-not 问题解释功能。最后，为使社团发现结果更加直观和容易理解，社团发现的聚类结果将进行可视化展现。

通过上述功能总结分析可以得出本系统的需求概述主要包括以下几个方面：

1) 数据采集

采集数据通过现有的网络爬虫工具，采集需要的系统数据并将采集结果存储到文数据库中。

2) 数据处理

由于采集到的数据中含有很多无用的数据信息，所以对其进行处理留下有用的信息，并将其处理成本系统需要的文本文件格式。

3) 支持 why-not 问题解释的社团发现算法

本功能模块能够实现社团聚类，并且支持用户的个性化参数调节，为使用户期望的数据点出现在期望的社团中，选取出合适的输入参数，这也是本系统最大的研究点和亮点。

4) 结果可视化

为满足使聚类结果显示更清晰直观，将聚类的结果利用现有的可视化工具进行可视化展示。

(2) 性能需求分析

支持 why-not 问题解释的社团发现系统在功能上进行分析的同时，也从性能上进行了详尽分析。本系统主要从可靠性与稳定性、扩展性、可维护性等方面进行性能分析，分析结果如下：

1) 可靠性与稳定性

系统的可靠和稳定是保证其正常运行的基础，如果系统不够稳定，那么工作人员就无法正常使用，这是大家都不愿意看到的。本系统在进行数据采集时不可能时刻的去监控整个采集过程，所以我们使用现有的成熟的网络爬虫工具，来确保数据采集过程的顺利进行。而其他的功能部分并不能影响系统的可靠性以及稳定性。

2) 可扩展性

社团结构是多种多样的，所以进行数据采集时数据源会因为社团结构的不同而不同，why-not 问题解释能解释的数据集也应该更全面。因此，综合考量，本系统应该具有较好的扩展性，以适应未来系统不同的应用需要，这也是本系统的一个重要考虑方面。

3) 可维护性

维护性对于一个系统来说非常重要因为他是一个系统能够长期应用的重要前提，如果系统可维护性差，那么系统出现故障时不能及时修复，这会给用户带来很大的不便甚至是影响，同时也会给系统维护人员带来额外的工作压力。所以系统需要严格按照软件设计流程进行设计，完成系统设计文档和用户使用说明的编写，详细说明整个系统的开发流程，这样给维护人员的工作带来极大的便利，工作负担会大大减少。

3.1.2 系统界面分析

本系统面对的直接对象是有社团发现需要的用户。这需要我们在设计系统界面的时候要求尽量简洁美观，这样使用户能在最短时间内了解系统。用户对一个软件进行评价的时候，第一评价印象就是用户操作界面是否简洁。因此对主页面的设计

会直接影响软件开发的成功与否。标准的系统其操作界面通常要具备以下几个特征：

(1) 一个美观和简洁的页面。除了要有简洁易用的系统界面之外，还需给系统适当的地方提供一些帮助性的文档，方便用户操作。

(2) 方便使用。系统界面上的操作应该是简单的，要避免多个界面的频繁切换。

(3) 系统需要有简单易懂的提示信息。这就需要系统界面不仅简洁，还需要在遇到操作或其他错误时，能够弹出提示信息提示。

3.1.3 可行性分析

在系统功能方面，本章的 3.1 小节介绍了功能需求，本系统具有以下四个方面的功能：数据采集、数据处理、why-not 问题解释功能、可视化功能。对于数据采集功能来说，主要应用到网络爬虫技术，因为目前的网络爬虫技术相对成熟，并且存在着多种爬虫工具并且其爬行效果很好，所以对于数据采集功能在现有的网络爬虫工具基础上去实现。对于数据处理功能，就是对爬行数据中的无用信息进行处理并转换成特定的存储格式，利用简单的代码就可以实现。对于 wht-not 问题解释功能虽然现实生活中没有现成的技术，但是有相关的理论支撑，所以该功能也可以实现，并且具有着很好的应用价值，对于结果可视化部分，同网络爬虫技术一样，社团结构的可视化工具也存在很多，相应的选择当前使用的可视化工具完成结果的可视化，所以综上所述，该系统技术上是可行的。

3.1.4 系统架构分析

系统体系结构位于系统需求分析和系统设计之间，它是系统设计和开发的关键，系统的体系结构设计关系到系统的应用场景，以及对系统后期的维护和优化也起着重要的作用。现在比较常见的系统体系结构主要有 B/S 结构和 C/S 结构。B/S 结构目前的应用很广泛，它在降低开发难度的同时，提高了系统的升级与维护效率，并且不必进行无用软件的安装，对用户使用的软硬件要求相应的就比较低用的软硬件条件限制。综上分析本系统采用 B/S 结构，下面将对 B/S 结构尽进行介绍。

B/S 结构即为浏览器/服务器模式，系统功能的核心部分在服务器上实现，web 浏览器作为客户端，是一种能够利用多种脚本和控件技术来进行访问的软件体系结构。浏览器上主要进行页面显示和逻辑交互，核心算法运行在在服务端。这种结构模式就是一种 MVC 设计模式，即模型、视图、控制器。模型主要用来操作数据

库，视图主要是界面显示，控制器主要处理用户交互的部分，他负责从视图读取数据，控制用户输入，并向数据库发送数据。B/S 架构的特点体现在：

(a) 系统的维护和升级方便。因为客户端与服务端是分离的，系统的核心实现逻辑主要在服务端，维护人员只需管理服务端即可，不需要去修改浏览器内容。

(b) 成本低。这种结构的软件系统只在一台服务器上安装，其他计算机直接通过访问浏览器访问系统即可。

(c) 开发简单，共享性强。MVC 设计模式有着明显的层次架构，所以这使系统分工更明确，降低了开发难度。

B/S 结构图如下图所示：

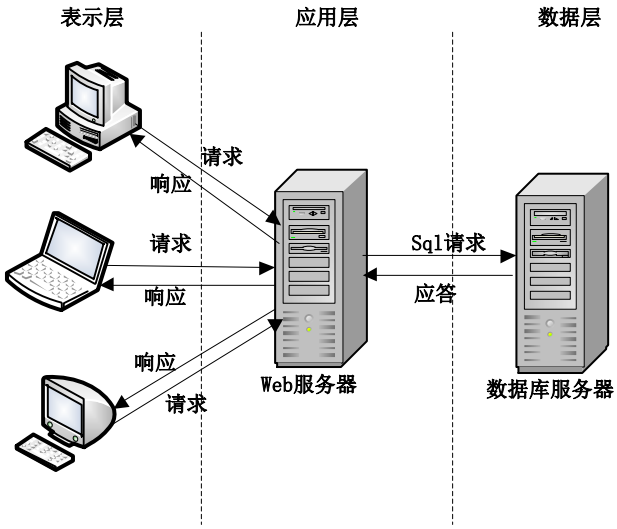


图 3.1 B/S 架构图

Fig. 3.1 Browser/Server architecture

3.2 系统设计

本小节主要对支持 why-not 问题解释的社团发现系统进行设计，主要对已经确定的各功能模块进行设计，并对各功能模块进行具体的内容介绍。

3.2.1 总体设计

本系统目的是建立一个支持 why-not 问题解释的社团发现系统平台，通过该平台实现对不同社团结构的数据源的社团发现，以及能够根据用户提出的 why-not 问题实现个性化调参，即重新确定一个聚类参数使得用户期望的对象点能够被划分到期望聚类中。根据本章 3.1 小节的系统分析，能够确定本系统在结构上包括确定数据源、原始数据的采集、数据处理、why-not 问题解释的社团发现、聚类结果可视

化。本系统的数据源包含通过网络爬虫技术相应的网站进行数据爬行微博好友网络、还有现有的已经爬好的 ca-GrQc 数据集以及 DBLP 论文作者网络等。数据处理就是将社团结构数据进行处理，只需要节点和各个节点之间的联系，将其余不需要的信息全都处理掉，然后将数据处理成需要的数据格式并以文本文件的格式进行存储，在运行 why-not 问题解释功能时会有读取上一步数据处理生成的数据进行功能实现，最后利用将的处的聚类结构用现有可视化工具展示出来。

(1) 系统架构设计

通过第 1、2 章所涉及到的相关技术介绍以及需求分析，从整体上可以确定本系统的基本架构。本系统采用 MVC 设计模式，主要分为三层即，应用层、服务层、数据层，每个层次之间通过接口的调用实现连接，相互之间又保持着各自的独立性。本系统的系统架构图如图 3.2 所示。

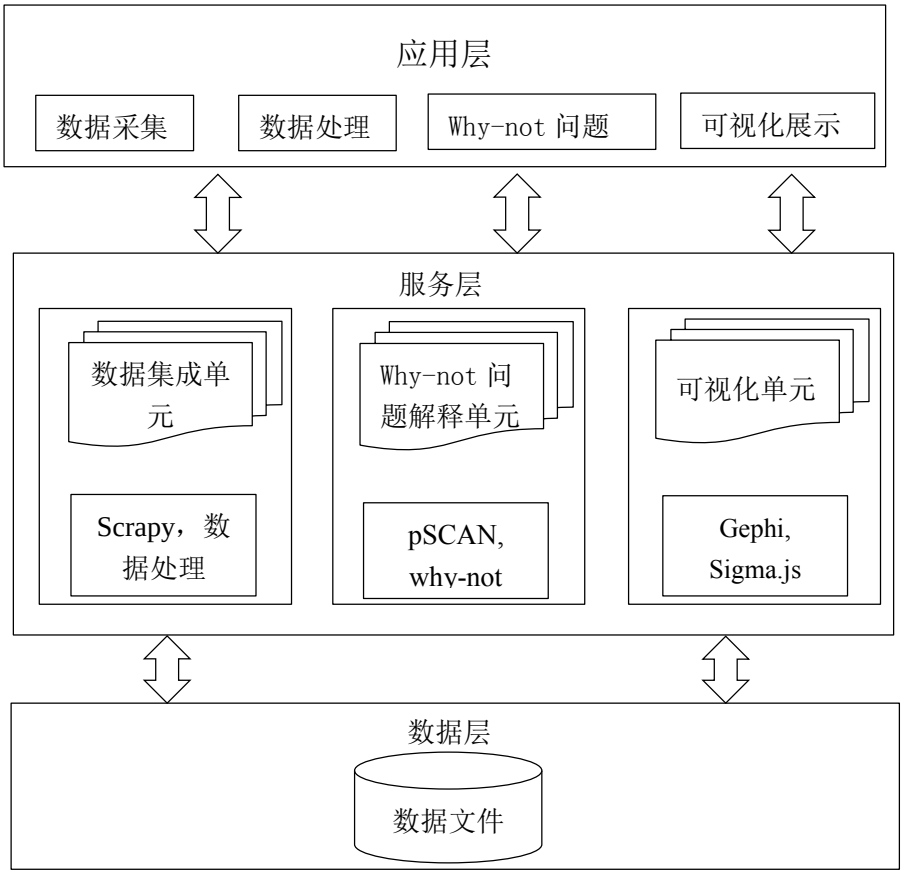


图 3.2 系统体系结构图

Fig. 3.2 The physical architecture diagram of system

通过对系统体系结构图的分析，对各层进行简要介绍：

应用层：应用层就是用户与系统对话的窗口。本系统的客户端是以浏览器的方式进行访问，用户无需登录系统，直接从浏览器打开进入用户界面之后，选择相应的模块进行操作，就可以获取所需要的结果，其具体实现逻辑是由服务层来实现，用户不需要对其进行了解。

服务层：服务层则是实现各功能模块的具体实现技术，所有功能的核心实现方法都是在服务层完成的，它能够连接应用层和数据层实现这两层之间的交互，通过接收应用层提交的请求，在数据层获取满足要求的数据在服务层进行处理后，将结果返回到应用层。据处理将其转化成适合的数据存储格式。并且将聚类完成的结果也以非格式化文件存储

数据层：所有要进行操作的数据都被存储在这一层，本层的数据由网络爬虫技术获取的。分别获得微博好友网络、DBLP 论文作者网络的社团结构都将存储到数据层。

(2) 功能模块设计

本系统可分为了四个主要功能模块，分别是数据采集、数据处理、支持 why-not 问题的社团发现、结果可视化。而数据处理与支持 why-not 问题的社团发现都包含二级功能模块。本系统的功能模块图如下所示：

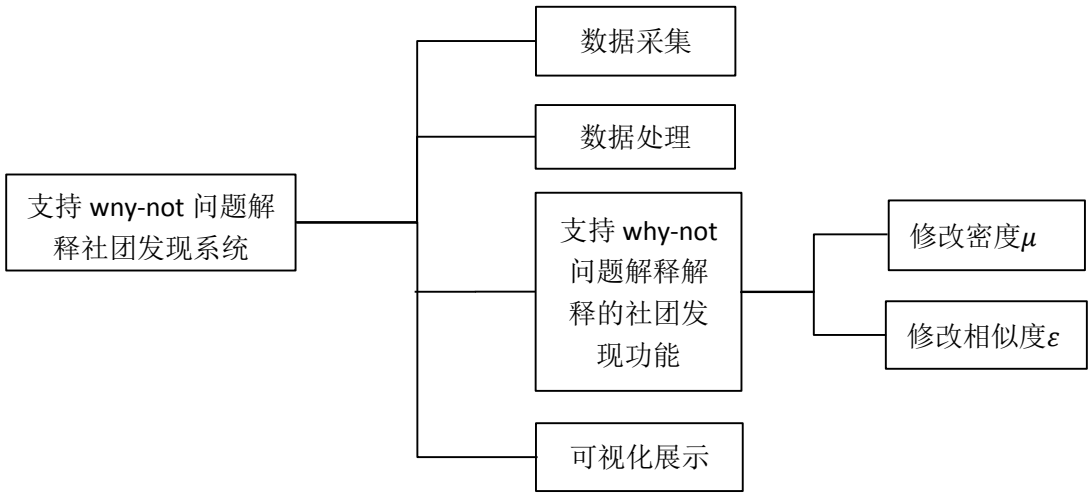


图 3.3 系统功能模块结构图

Fig.3.3 The function module diagram of system

根据系统功能模块结构，下边对四个一级功能模块进行介绍如下：

数据采集模块：采集的数据源可以是任意的网络，然后将利用现有网络爬虫工具采集到的数据存储到的数据数据库中，还有一部分数据不是经过网络爬出的，而是由

某些实验室提供的存储一定属性信息的图结构的非格式化文本数据。

数据处理：对于从网络中爬出来的数据即结构化数据将无用的信息处理掉，然后转化成具有一定格式的文本数据；对于非格式化的文本数据也进行，转化成的数据会作为下一功能的输入。

支持 why-not 问题解释的社团发现模块：实现社团发现算法，能够自适应的调节参数以此实现对 why-not 问题的解释功能。

结果可视化模块：利用现有可视化工具，将聚类结果进行可视化展示。

3.2.2 详细设计

⑴ 数据采集模块

本功能模块采用网络爬虫技术 Scrapy 框架，一个由 Python 语言开发的 web 抓取框架，进行数据的采集。Scrapy 用途广泛，任何人都可以根据需求方便的进行修改。并且它也提供了多种类型的网络爬虫的基类，比如 BaseSpider、sitemap 爬虫等。Scrapy 也因其实现语言是 Python，所以实现起来更加的方便。采集用户需要的信息存储到文本文件中。具体实现过程为用户通过在浏览器提交的数据采集请求，服务器端的数据采集模块接收请求后创建任务，运行数据采集模块的基于 Scrapy 框架的网络爬虫功能，将数据存储到文本文件中。任务结束后，会通过系统采集模块返回给浏览器数据收集完毕的提示信息。数据采集结构如图 3.4 所示。

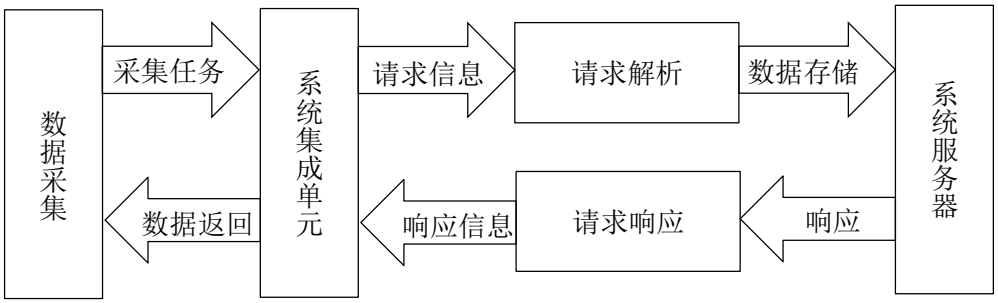


图 3.4 系统数据采集结构图

Fig3.4 The data acquisition architecture diagram of system

在本部分有的用户有可能并不需要从网上爬取数据，而是直接使用自己提供的社团结构进行社团发现。因此可以提供用户自己的数据文件但要符合一定的文本结构。Scrapy 框架通过编写简单数据爬取代码，包括通需要提取的数据结构定义，与 Scrapy 的请求相关联的请求及响应相关联的框架，以及对提取数据进行处理操作等等的相关代码就可以基于框架进行用户所需信息数据的爬取。在数据采集模块的实

现过程中，主要就是利用 Scrapy 数据爬取框架，进行数据的爬取，然后将提取的数据存储到服务器系统中。数据采集的流程图如图 3.5 所示。

新建采集任务就是在浏览器界面点击数据采集功能时，则后台建立数据采集任务，进而运行基于 Scrapy 框架的数据爬取代码，如果爬取成功，则将爬取成功后的数据文件存储到系统服务器中。若未成功返回给用户未成功信息。

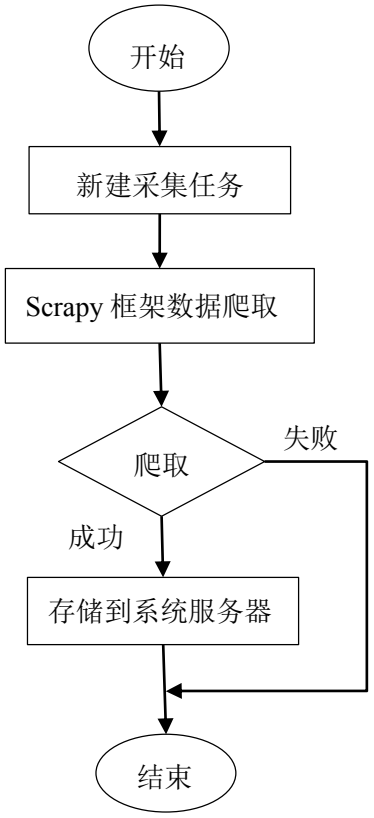


图 3.5 系统数据采集流程图

Fig. 3.5 The data collection flowchart of system

⑵ 数据处理模块

在完成上一步的数据采集之后，因为数据中有可能有无用的信息，或数据格式不符合系统的要求。用户通过在浏览器提交的数据处理请求，服务器端的数据处理模块接收请求后创建任务，运行数据处理程序，将数据格式进行规范化。任务结束后，会通过系统处理模块返回给浏览器数据处理完毕的提示信息。数据处理结构如图 3.6 所示。

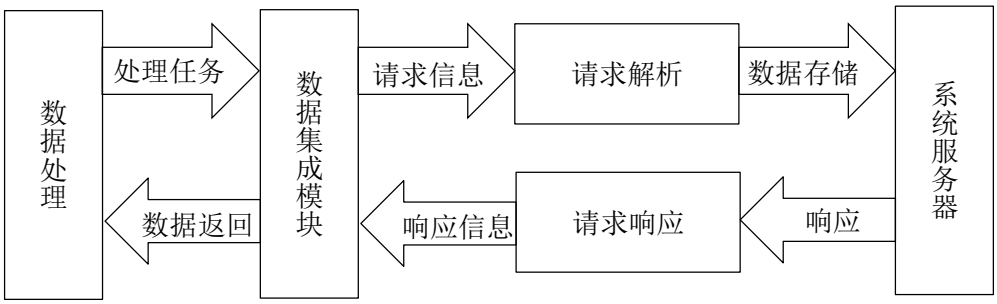


图 3.6 系统数据处理结构图

Fig. 3.6 The data processing architecture diagram of system

(3) why-not 问题解释模块

本部分功能实现社团发现以及对社团发现中出现的 why-not 问题进行解释的功能。因为社团发现算法主要有两个参数即密度阈值参数μ和相似度阈值参数ε，所以该模块从这两方面对本系统进行 why-not 问题解释。用户通过在浏览器点击支持 why-not 问题解释的社团发现简称解释功能模块，出现二级功能菜单，在二级菜单中可以选择是进行社团发现功能还是 why-not 解释功能，在提交相应的功能请求后。服务器端的社团发现模块接收请求后创建任务，运行解释算法程序，得出的是修改后的对应参数直接现实在浏览器界面，若想查看修改参数之后的结果，则可运行社团发现算法得出的文件进行展示。

对于社团发现模块的算法已经在第二章进行预先的介绍，而对于解释功是本系统的研究重点，所以社团发现算法的 why-not 问题解释的实现将在第 4 章进行详细的介绍。

(4) 结果可视化模块

该模块实现将社团结果利用可视化工具可视化的展现出来。选用的可视化工具为 Gephi 安装 Sigmajs Exporter 插件，就可以将网络图内嵌到网页中。可视化的效果如图 3.7 所示。

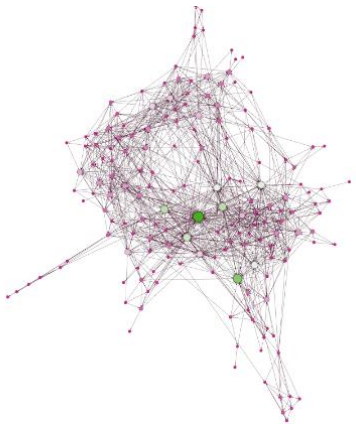


图 3.7 可视化效果图

Fig. 3.7 Diagram of visual effect

3.3 本章小结

本章首先对系统进行了分析。先对系统进行了需求分析，包括功能需求分析和性能需求分析两个方面。然后分析了系统的技术可行性，确保系统在技术上能够实现，接着对系统的架构进行了分析和说明。另外，又对系统的设计进行介绍，主要介绍了系统的总体设计和详细设计。而再详细设计部分主要分了四个模块分别是数据采集、数据处理、why-not 问题解释、结果可视化，因为数据采集、数据处理、结果可视化这三个功能都是基于现有的且成熟的工具去实现，而 why-not 问题解释功能下的支持 why-not 问题解释的社团发现算法则在第 4 章进行具体的介绍。

第 4 章 基于 why-not 问题解释的自适应的社团发现算法

在第 3 章已经对支持 why-not 问题解释的社团发现系统其他三个功能模块已经进行了详细的设计和说明。因为，本章将主要介绍社团发现算法的 why-not 问题解释算法的具体实现。对 pSCAN 聚类算法而言，确定其输入参数是一个很重要的问题。该聚类算法中的两个参数——密度阈值μ，相似度阈值 ε对聚类结果都有很大影响，首先进行 pSCAN 算法的 why-not 问题定义。其次将对密度阈值μ以及相似度阈值ε进行分析，探索其对聚类结果的影响，通过其对聚类结果的影响，找出如何使得 why-not 对象点 w 属于期望聚类 C_{target} 的基本算法，最后再分别对算法进行优化。

4.1 pSCAN 算法的 why-not 问题定义

在上文已经介绍了 pSCAN 聚类算法的算法思想。那么问题出现了，当我进行 pSCAN 聚类时会出现一个问题我期望的某个数据应该会出现在聚类结果中的某个聚类里，但却未出现，这就是 pSCAN 算法中的 why-not 问题的提出。接下来将给出 pSCAN 算法的 Why-not 问题的定义：

定义 4.1 pSCAN 算法的 why-not 问题。 给定一个结构图$G = (V, E)$，在 G 上执行 pSCAN 聚类查询 $Q(\varepsilon, \mu)$，Q 的聚类结果为 Rc，且 Rc 包含 n 个簇 C_1，C_2，…，C_n，假设 w 为 why-not 对象，C_{target} 为期望簇，pSCAN 聚类的 why-not 问题为:

(1)为什么 w 不属于期望簇 C_{target};

(2)怎样用最小的代价修改修改原来的聚类查询 $Q(\varepsilon, \mu)$，使得 w 能够出现在期望簇 C_{target} 中。

根据定义 4.1，本文需要从两方面回答 pSCAN 算法的 why-not 问题，即（1）给出为什么缺失数据没有出现在结果集中的原因；（2）对如何修改原始 pSCAN 算法提供建议，使得在代价最小的情况下使结果集中包括缺失数据。

4.2 修改密度阈值参数μ

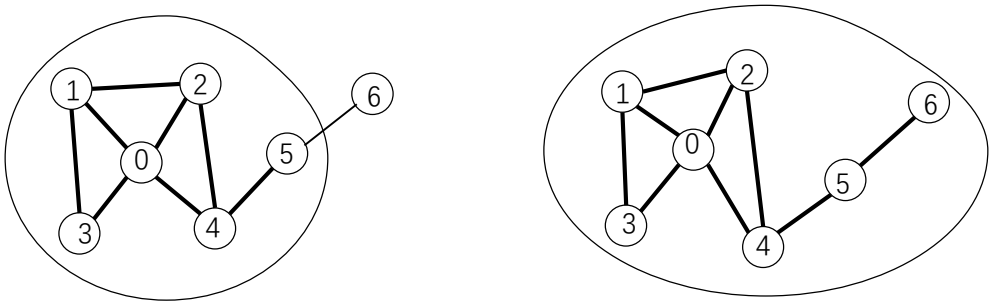
4.2.1 密度阈值μ对聚类结果的影响

pSCAN 聚类参数必须由用户来确定，μ是期望聚类的最小簇的大小。由经验法则，μ的最小值 1 没有任何意义，因为这种情况下每个点自己已经是一个聚类了。然

而，比较大的μ值对于有噪声的数据集而言通常会更好一些，也会产生更有意思的聚类。数据集越大，μ的值也应该更大一些。

pSCAN 算法聚类参数中的密度阈值参数μ在相似度阈值ε一定的情况下，决定了一个点是不是核心点。一个点与其邻居节点的结点相似度满足大于等于ε的点的数量 *count* 与μ值的大小比较，能够判断该点是不是核心点，由于ε值不变，即满足结点相似性度大于等于ε点的数量 *count* 不变，改变μ值的大小使得 $count \geq \varepsilon$，从而使得该点成为核心点。pSCAN 算法过程中只对核心点的邻居节点进行扩展，更改μ会进一步影响聚类结果中聚类的大小。定理 4.1 给出了密度阈值参数μ对聚类结果的影响。

定理 4.1 在 pSCAN 聚类算法中，保持ε不变，当μ减小时，原属于同一聚类的对象还会处于同一聚类，而聚类结果中单个聚类的大小可能会增加。



(a) $\varepsilon = 0.0001, \mu = 4$　　(b) $\varepsilon = 0.0001, \mu = 3$

图 4.1 密度阈值μ对聚类结果的影响

Fig. 4.1 Impact of density threshold μ on clustering results

给出该定理的证明。在图数据集 *G* 上进行参数为ε、μ的 pSCAN 聚类，聚类 *C* 为聚类结果中的任意一聚类，对于任意 v_1，$v_2 \in C$，依据定义 2.9 聚类的连通性，属于同一聚类的任意两个对象是相连的，v_1，v_2 相连，即存在一点 *o*，v_1 和 v_2 是从 *o* 关于ε和μ可达的。对于 v_1，存在 *o*，v_3，v_4，…，v_{n-1}，v_n，其中 $v_n = v_1$，满足对于所有的 $i = 3$，4，……，*n*-1，v_{i-1} 是从 v_i 关于ε和μ直接可达的，*o*，v_3，v_4，…v_{n-1} 是关于ε和μ的核心点；同理对于 v_2，存在 *o*，v'_3，v'_4，… v'_{n-1}，v'_n，其中$v'_n = v_2$，满足对于所有的 $i = 3$，4，……，*n*-1，v'_{i-1}是从v'_i关于ε和μ直接可达的，*o*，v'_3，v'_4，… v'_{n-1}是关于ε和μ的核心点。当$\mu' < \mu$时，*o*，v_3，v_4，… v_{n-1}，v'_3，v'_4，… v'_{n-1}关于ε和μ'仍为核心点，即对于 v_1，存在 *o*，v_3，v_4，… v_{n-1}，v_n，其中 $v_n = v_1$，满足对于所有的 $i = 3$，4，……，*n*-1，v_{i-1} 是从 v_i 关于ε和μ直接可达的，即 v_1 是从 *o* 关于ε和μ'密度可达的，同理 v_2 是从 *o* 关于ε和μ'密度可达的。由此可知点 *o* 使得 v_1，p_v 是相连的，由定义 2.9

聚类的最大性，若 $v_1 \in C$，且 v_1，v_2 相连，则 $p_2 \in C$。综上所述，对于参数为 ε、μ的 pSCAN 聚类结果中属于同一聚类的任意多个聚类对象，当 pSCAN 算法参数为 ε、μ' 时，其中$\mu' < \mu$时，其仍属于同一聚类。定理 4.1 得证。

图 4.1 中给出一个阈值参数μ对聚类结果影响的例子。图 4.1(a)、图 4.1(b)为相同的数据集合，两个线圈表示聚类结果。在图 4.1(a)中，聚类参数μ等于 4，P_4 的$\varepsilon-$ neighbors值为 *count*=4，$count \geq \mu$，即 P_4 为核心点，P_5 的$\varepsilon-$ neighbors值为 *count*=3，$count < \mu$，P_5 不是核心点。当执行聚类算法时，P_5 由核心点 P_4 可达，P_5 被标记为同 P_4 同属一个聚类；由于 P_5 不是核心点，不能进行扩展，虽然 P_6 是 P_5 的邻居，但 P_6 不能被扩展本身也并不是核心点，所以不能被聚类，根据最终的聚类结果它将被标记为桥节点。在图 4.1(b)中，聚类参数μ为 3，此时 P_4 的$\varepsilon-$ neighbors值为 *count*=4，$count > \mu$，即 P_4 为核心点；P_5 的$\varepsilon-$ neighbors值为 $count = 3$，$count \geq \mu$，即 P_5 也为核心点。当 pSCAN 算法执行时，在扩展 P_4 的$\varepsilon-$ neighbors点时，P_5 被标记为同 P_4 同属一个聚类，接着在对 P_5 的$\varepsilon-$ neighbors点进行扩展时，P_6 被标记为同 P_5 同属于一个聚类。此时，实线圆圈内的所有点被聚为一类，P_4、P_5、P_6 这三个点同属于一个聚类。在μ值减小的情况下，聚类结果中的簇增大了（将点 P_6 包含进去）。

根据 pSCAN 聚类算法中密度阈值参数μ对聚类结果的影响，可以通过修改该参数使得聚类对象 *w* 属于期望聚类 C_{target}。

4.2.2 修改密度阈值阈值参数μ的基本算法

对于图 4.1 中所示例子，通过对μ参数值的修改，将μ= 4 改为μ= 3，可扩大聚类 *C*，将使得 P_6 包括到聚类 *C* 中。

随着阈值参数μ取值的减小，聚类结果中的簇变大，聚类对象 *w* 可能被聚到聚类 *C* 中。一方面并不是只要μ减小到足够小，聚类对象 *w* 就一定会被聚类到聚类 *C* 中。另一方面，μ值的改变，也会影响到其他点的聚类结果。鉴于此，本文提出修改聚类参数μ，使得聚类对象 *w* 属于某聚类 *C* 的解决方案，

由定理 4.1 可知，要根据修改μ的方法使得聚类对象 *w* 属于某聚类 C_{target}，应将μ值改小，即满足该条件的μ'的有效区间为$[1, \boldsymbol{\mu})$。

而通过第 2 章中 2.2.3 小节对 pSCAN 聚类算法的思想描述，可知其实质上还是通过判断一个节点是否为核心点，是核心点则进行聚类的扩张，而我们要做的就是如何通过修改密度阈值μ使得所期待的聚类对象 *w* 属于某个聚类 C。那我们就可以思考，如果在相似度阈值不变的条件下，若要使得期待对象 *w* 出现在某个聚类 C 中，则应该满足聚类 C 中的某一节点 V_i 到 *w* 是连通的，且路径 V_i，V_{i+1}，…，V_{m-1}，*w* 上，V_i 到 V_{m-1}

节点都应是核心点。则需要选择这条路径上的节点 V_i 的ε −邻居值最小的那个为μ'，并且满足μ'在有效区间[1, $\boldsymbol{\mu}$)内($\boldsymbol{\mu}$表示的是原始密度阈值)，因为这样满足了有最小ε −邻居的节点其$N_\varepsilon[V_i] \geq \mu'$，使其能成为核心点，那该路径上的其他节点则都可以成为核心点，这样就可以使得聚类对象 V_m 被拉进聚类 C 中；找到所有满足这种情况的路径 P_n，为了减少μ的修改代价，在路径中产生的所有路径上的 minμ_n中选出最大μ_{max}，则此时的密度阈值μ'即为μ_{max}，若无返回值则表示没有合适的μ'。

算法 4.1 修改密度阈值μ使得聚类对象 w 属于期望聚类 C

输入：结构图 G，结构相似性阈值ε， 密度阈值$\boldsymbol{\mu}$，聚类 C，聚类对象 w

输出：修正后的μ'

1. Refineμ(G, ε, $\boldsymbol{\mu}$, C, w)
2. for $V_i \in C$
3. $Path$ = PathFind(w, V_i)
4. Compute structure smiliarity of pinots in Path get $\varepsilon-$ neighbors
5. μ = minimum $\left(\varepsilon\text{-neighbors of } V_i \text{ and } \varepsilon\text{-neighbors} \in [1, \boldsymbol{\mu})\right)$
6. end for
7. if $\mu' \leq \mu$
8. $\mu' = \mu$
9. **return** μ'

算法 4.1 给出了 pSCAN 算法中求解使得聚类对象 w 属于期望聚类 C 的密度阈值μ的方法。在算法执行过程中，首先计算 why-not 对象 w 到期望聚类中点的路径（第 3 行）。接下来依次计算路径上每个点与其邻居节点的结构相似性（第 4 行）。然后得出每个节点的$\varepsilon-$neighbors 值，比较得出最小值且要满足其在有限区间 $[1, \boldsymbol{\mu})$范围内（第 5 行）。在所有路径中比较得出最大的μ，使得$\mu' = \mu$，此时μ' 即为修改值，最后返回该值（6-8 行）。算法 4.1 给出修改密度阈值μ使得聚类对象 w 属于期望聚类 C 的伪代码。

分析算法 4.1，有很多不足之处，首先寻找路径时，会利用队列数据结构记录已访问过的路径，这样会用到大量的内存空间，处理大数据时会使内存崩溃使得该算法对于大规模海量数据的可扩展性不强；其次寻找路径时，所有路径都要找，这样即使有不符合条件的路径也会去遍历，造成了资源浪费。所以此算法有待改进，接下来将根据上面两个问题，进行算法优化。

4.2.3 修改密度阈值阈值参数μ的优化算法

在本章 4.2.2 节中针对提出的算法进行了分析，发现有两个问题

（1）寻找路径时，会利用队列数据结构记录已访问过的路径，这样会用到大量的内存空间，处理大数据时会使内存崩溃使得该算法对于大规模海量数据的可扩展性不强。

（2）有不符合条件的路径也会去遍历，造成资源浪费。

所以本小节将针对已上两个方面对算法进行优化：

第一方面，利用回溯方法来避免路径存储：首先找到一条符合条件的路径，例如，V_{target}，V_i，…，V_m，w，其中 V_{target} 是期望聚类 C_{target} 内的任一点，经过路径上的若干点 V_i，…，V_m，到达 why-not 点 w。接下来以此路径为基准，从点 V_m 开始向外扩展，寻找到 why-not 点 w 的路径，接下来回溯到点 $V_{m\text{-}1}$ 再向外扩展，寻找到 why-not 点 w 的路径，而从点向外扩展寻找到 why-not 点 w 的路径时也相应地回溯，直到回溯到 C_{target}，同理还是每次找到路径里节点里最小的$\varepsilon-neighbors$作为候选μ，最后选择所有路径里的最大且在有效区间[1,$\boldsymbol{\mu}$)内的μ为μ'。

回溯方法实际上就是暴力穷举，此种方法还是将所有的路径都进行了计算，因此接下来对第二个问题进行优化。

第二方面，提出剪枝策略减少多余路径的计算：

(1) 回溯位置选择：因为首先要找到一条路径，例如路径，V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，选择出此条路径下节点最小的$\varepsilon-\text{neighbors}$作为候选的$\mu$，假设点 u 的$\varepsilon-\text{neighbors}$是最小的，即$\mu=\varepsilon-\text{neighbors}$。这样最小的$\varepsilon-\text{neighbors}$节点 u 都可以满足$N_\varepsilon[u]\geq\mu$，即 u 点是核心点，那么路径上的其他节点都会为核心点。此时回溯时并不是在上部分提到的 V_m 处，而是在点 u 的前一节点 $V_{u\text{-}1}$ 处开始回溯，因为我们要找所有路径的最小的μ里的最大值，这样我们回溯时要找的μ'应该是大于当前的μ且在有效区间[1, $\boldsymbol{\mu}$)内，所以若我们回溯时还是在 why-not 节点 w 前的 V_m 节点进行，那这样即使找到到 w 的路径，其最小的μ值还是$N_\varepsilon[u]$，所以对于路径 V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w 上的 u，…，V_m，处进行回溯是无意义的。

(2)回溯剪枝条件：对于路径 V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，当从 $V_{u\text{-}1}$ 处进行回溯，回溯时判断该点的邻居节点是否被访问过，选择一个未被访问过的邻居节点假设为 ui。判定条件一：判断$\sigma(u,ui)\geq\varepsilon$是否成立，即 u 与 ui 是否是结构相似的，若成立，则进行判定条件二：判断$N_\varepsilon[ui]\geq\mu$是否成立（注意这里的μ是已经选出来的 u 的$\varepsilon-\text{neighbors}$值），若成立则，要继续访问$ui$的未被标记的邻居重复进行判定条件一二的判断。判定条件二不成立的话，则回溯到 ui 的上一层节点 u 处，继续找其他未

被访问的邻居节点重复进行判定条件一二的判断。判定条件一若不成立则直接访问节点 u 未被访问标记的邻居节点重复进行判定条件一二的判断。

算法 4.2 优化算法

输入: 图数据G，结构相似性阈值ε， 密度阈值μ，聚类 C，聚类对象 w

输出: 修正后的μ'

```
1.  PathStack= PathFind( w, Vi)
2.  u = the points' value of ε − neighbors are minimum
3.  do
4.      p = pop(PathStack)
5.  until p==u
6.  μ = ε − neighbors of u
7.  for u in pop(PathStack)
8.      for ui belongs to the unmarked neighbors of u
9.       do
10.          if σ(u, ui) ≥ ε
11.              If N_ε[ui] ≥ μ
12.                  Push(PathStack,ui)
13.                  u = ui
14.              else
15.                  u = pop(PathStack)
16.      until u == w
17.        If (u==w)
18.          u=the points' value of ε − neighbors are minimum
19.            p = pop(PathStack)
20.          If(μ ≤ ε − neighbors &&μ ∈ [1, μ))
21.                μ = ε − neighbors of u
22.      end for
23.  end for
24.  μ' = μ
25.  return μ'
```

以上是对 4.2.2 小节中算法 4.1 的优化思想，此优化算法的伪代码如算法 4.2 所示。

该算法首先通过 PathFind 找到一条路径（第 1 行），注意在 w 到 V_i 的路径需满足路径上的每条边都得要是结构相似的。例如路径，V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，选择出此条路径下节点最小的$\varepsilon-$neighbors作为候选的μ，假设点 u 的$\varepsilon-$neighbors是最小的（第 2 行）。即$\mu=\varepsilon-$neighbors。这样最小的$\varepsilon-$neighbors节点 u 都可以满足$N_\varepsilon[u]\geq\mu$，即 u 点是核心点，那么路径上的其他节点都会为核心点。此时在点 u 的前一节点 $V_{u\text{-}1}$ 处开始回溯，u 点后的点都出栈（3-5 行）。对于路径 V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，当从 $V_{u\text{-}1}$ 处进行回溯，回溯时判断该点的邻居节点是否被访问过，选择一个未被访问过的邻居节点假设为 ui（7,8 行）。判定条件一：判断$\sigma(u,ui)\geq\varepsilon$是否成立，即 u 与 ui 是否是结构相似的（第 9 行），若成立，则进行判定条件二：判断$N_\varepsilon[ui]\geq\mu$是否成立（注意这里的μ是已经选出来的 u 的$\varepsilon-$neighbors 值）（第 10 行），若成立则，要继续访问ui的未被标记的邻居重复进行判定条件一二的判断。判定条件二不成立的话，则回溯到 ui 的上一层节点 u 处，继续找其他未被访问的邻居节点重复进行判定条件一二的判断。判定条件一若不成立则直接访问节点 u 未被访问标记的邻居节点重复进行判定条件一二的判断（11-19 行），且每次遇到 u==w 的情况，即已经选出一条路径，那么需要更新μ为最新的大于原μ，且属于有效区间$[1,\boldsymbol{\mu})$中（20-26 行）。最后将μ'修改成最终的μ并返回μ'（27,28 行）。

4.3 修改相似度阈值ε

4.3.1 相似度阈值ε对聚类结果的影响

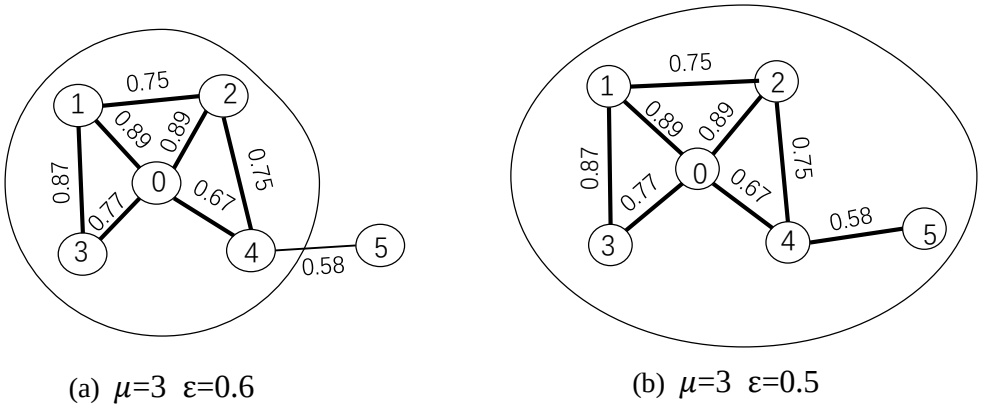
pSCAN 聚类参数必须由用户来确定，而该算法中的ε参数是两个节点实体之间的结构相似性的一个标准，若是降低这一标准则会使得原本不满足的能够满足结构相似性，这样在密度阈值一定的情况下，原本不是核心点的点会变成核心点，进而有可能会使得两个原本不在同一聚类里的实体被聚到一个类里。

pSCAN 算法过程中只对核心点的邻居节点进行扩展，更改ε会进一步影响聚类结果中聚类的大小。对于 pSCAN 算法，如果ε值选择的过大，会有大量的点不能被聚类；而ε值选择的过小，聚类会合并，大多数的点会在同一个聚类中。定理 4.2 给出了相似性阈值参数ε对聚类结果的影响。

定理 4.2 在 pSCAN 聚类算法中，保持μ不变，当ε减小时，原属于同一聚类的对象还会处于同一聚类，而聚类结果中单个聚类的大小可能会增加。

给出该定理的证明。存在 o，v_3，v_4，… ，$v_{n\text{-}1}$，v_n，其中 v_n= v_1，满足对于所有的 i = 3，4，……，n-1，$v_{i\text{-}1}$ 是从 v_i 关于ε和μ直接可达的，o，v_3，v_4，…$v_{n\text{-}1}$ 是关于ε和μ的核

心点；同理对于 v_2，存在 o，v'_3，v'_4，$\cdots v'_{n-1}$，v'_n，其中v'_n= v_2，满足对于所有的 i = 3,4,……,n-1，v'_{i-1}是从v'_i关于ε和μ直接可达的，o，v'_3，v'_4，$\cdots v'_{n-1}$是关于ε'和μ的核心点。当$\varepsilon' < \varepsilon$时，$o$，$v_3$，$v_4$，$\cdots$ v_{n-1}，其中v'_3，v'_4，$\cdots v'_{n-1}$关于ε'和μ仍为核心点，即对于 v_1，存在 o，v_3，v_4，$\cdots$ v_{n-1}，v_n，其中 v_n= v_1，满足对于所有的 i = 3，4，……，n-1，v_{i-1} 是从 v_i 关于 ε'和μ直接可达的，即 v_1 是从 o 关于 ε'和μ密度可达的，同理 v_2 是从 o 关于 ε'和μ密度 可达的。由此可知点 o 使得 v_1，v_2 是相连的，由定义 2.13 聚类的最大性，若 $v_1 \in C$，且 v_1，v_2 相连，则 $v_2 \in C$。综上所述，对于参数为 ε、μ的 pSCAN 聚类结果中属于同一聚类的任意多个聚类对象，当 pSCAN 算法参数为 ε'、μ时，其中 $\varepsilon' < \varepsilon$时，其仍属于同一聚类。定理 4.2 得证。



(a) μ=3 ε=0.6　　(b) μ=3 ε=0.5

图 4.2 相似性阈值ε对聚类结果的影响

Fig. 4.2 Impact of similarity threshold ε on clustering results

图 4.2 给出了ε值对聚类结果影响的例子。在图 4.2(a)中，pSCAN 参数分别为μ=3 和ε=0.6。点 p_4 与其邻居的结构相似度满足大于等于 ε的邻居点个数大于等于 3 个，为核心点。而点 p_5 与邻居节点的结构相似性满足大于等于ε的邻居点个数是 2，其值并不大于等于 3，所以他是非核心点，但因为其与核心点 p_4 的结构相似性大于等于 ε，因此他会被拉进 p_4 所属聚类 C，但是 p_5 不具有可扩展性并且 p_5 与 p_6 的结构相似性也并不满足大于等于ε，所以 p_6 不会被扩展到在聚类 C 中。在图 4.1(b)中，pSCAN 参数分别为μ=3 和ε=0.5。此时 p_5 与其邻居节点之间的结构相似性满足大于等于ε的邻居点个数为 3，所以 p_5 是核心点且其被聚到和 p_4 同一聚类 C 中，具有扩展性，又因为 p_5 与 p_6 的结构相似性满足大于等于ε，因此 p_6 被扩展到与 p_4 和 p_5 属于一类，即 p_6 点属于聚类 C。此例中，原本和 p_4 不属于同一个聚类的点 p_6，在ε值从 0.6 减小到 0.5 之后，变成了同 p_4 同属于一个聚类。

鉴于 pSCAN 聚类算法中相似性参数ε对聚类结果的影响，可以通过修改该参数使得聚类对象 w 属于某期望聚类 C_{target}。

4.2.2 修改密度阈值阈值参数ε的基本算法

对于图 4.2 中所示例子，通过对 ε参数值的修改，将 ε= 0.6 改为 ε= 0.5，可扩大聚类 C，将使得 P_6 包括到聚类 C 中。

随着阈值参数ϵ取值的减小，聚类结果中的簇变大，why-not 对象 w 可能被聚到聚类 C 中。一方面并不是只要ε减小到足够小，聚类对象 w 就一定会被聚到聚类 C 中，因为还有另一个参数—密度阈值μ。另一方面，ε值的改变，也会影响到其他点的聚类结果。鉴于此，本文提出修改聚类参数ε，使得 why-not 对象 w 属于某期望聚类 C_{target} 中的解决方案，由定理 4.2 可知，要根据修改 ε的方法使得聚类对象 w 属于某聚类 C，应将ε值改小。

算法 4.3 修改相似性阈值μ使得聚类对象 w 属于期望聚类 C

输入：结构图 G，结构相似性阈值$\boldsymbol{\varepsilon}$， 密度阈值μ，聚类 C，why-not 对象 w

输出：修该后的ε'

算法描述：

1. Refineϵ(G, $\boldsymbol{\varepsilon}$, μ, C, w)
2. For $V_i \in C$
3. $Path$ = PathFind(w, V_i)
4. Compute structure smiliarity of pinots in Path get $\varepsilon-$neighbors
5. For each points' $\varepsilon-$neighbors
6. If $\varepsilon-$neighbors $<$ μ
7. $\varepsilon =$ min($\sigma(V_i,V_{i+1}),\sigma_i(V_i,neighbors\ of\ V_i)$)&& $\varepsilon < \boldsymbol{\varepsilon}$
8. get minimum ε on one path
9. end for
10. if $\varepsilon' \le \varepsilon$
11. $\varepsilon' = \varepsilon$
12. **return** ε

通过第 2 章中 2.2.3 小节对 pSCAN 聚类算法的思想描述，可知其实质上还是通过判断一个节点是否为核心点，是核心点则进行聚类的扩张，而我们要做的就是如何通过修改相似性阈值ε使得让所期待的聚类对象 w 属于某个聚类 C。那我们就可以思考，如果在密度阈值不变的条件下，若要使得期待对象 w 出现在某个聚类 C 中，则应该修改ε使得满足聚类 C 中的某一节点 V_i 到 w 是连通的，且路径 V_i，V_{i+1}，…，V_{m-1}，w 上，V_i 到 V_{m-1} 节点都应是核心点。则对ε进行修改时，第一需要保证该路径上的边都能够满足结构

相似性，第二需要注意的是新确定的相似性阈值ε'能够保证每个节点的$\varepsilon-\text{neighbors}\geq\mu$，即是路径上的 V_i 到 $V_{m\text{-}1}$ 都成为核心点，每条路径都应该确定一个最小的ε并且满足 $\varepsilon<\boldsymbol{\varepsilon}$。然后找到所有满足这种情况的路径 Pn，为了减少$\varepsilon$的修改代价，在路径中产生的所有路径上的 min$\varepsilon_n$中选出最大 ε_{max}，则此时的密度阈值 ε'即为 ε_{max}，若无返回值则表示没有合适的 ε'。

在算法 4.3 中给出了 pSCAN 算法中求解使得聚类对象 w 属于期望聚类 C 的相似性阈值 ε的方法。在算法执行过程中，首先计算 why-not 对象 w 到期望聚类中点的路径（第 3 行），注意该路径要满足路径上的节点除 why-not 对象节点 w 外，其余节点的邻居个数都应该大于等于μ。接下来依次计算路径上每个点与其邻居节点的结构相似性，然后得出每个节点的$\varepsilon-\text{neighbors}$ 值（第 4 行），确定得出一个最小的 ε并且满足 $\varepsilon<\boldsymbol{\varepsilon}$（5-8 行）。在所有路径中比较得出最大的 ε，使得$\varepsilon'=\varepsilon$，此时ε' 即为修改值，最后返回该值（9-11 行）。

分析算法 4.3，拥有的很多不足之处，其一是寻找路径时，会利用队列数据结构记录已访问过的路径，这样会用到大量的内存空间，处理大数据时会使内存崩溃使得该算法对于大规模海量数据的可扩展性不强；其次寻找路径时，所有路径都要找，这样即使有不符合条件的路径也会去遍历，造成了资源浪费。所以此算法有待改进，接下来将根据上面两个问题，进行算法优化。

4.2.3 修改密度阈值阈值参数ε的优化算法

在本章的 4.2.2 节中针对提出的算法进行了分析，发现有两个问题，这两个问题与上一章面临的问题是类似的。

（1）寻找路径时，会利用队列数据结构记录已访问过的路径，这样会用到大量的内存空间，处理大数据时会使内存崩溃使得该算法对于大规模海量数据的可扩展性不强。

（2）有不符合条件的路径也会去遍历，造成资源浪费。

所以本小节将针对这两个方面对算法进行优化：

第一方面，利用回溯方法来避免路径存储：首先找到一条符合条件的路径，例如，V_{target}，V_i，…，V_m，w，其中 V_{target} 是期望聚类 C_{target} 内的任一点，经过路径上的若干点 V_i，…，V_m，到达 why-not 点 w。然后以此路径为基准，从点 V_m 开始向外扩展，寻找到 why-not 点 w 的路径，接下来回溯到点 $V_{m\text{-}1}$ 再向外扩展，寻找到 why-not 点 w 的路径，而从点向外扩展寻找到 why-not 点 w 的路径时也相应地回溯，直到回溯到 C_{target}，同理还是每次找到路径里节点里最小的ε作为候选ε，最后选择所有路径里的最大且满足的 $\varepsilon<\boldsymbol{\varepsilon}$的$\varepsilon$为$\varepsilon'$。

算法 4.4 优化算法

输入: 图数据集G，结构相似性阈值$\boldsymbol{\varepsilon}$，密度阈值μ，聚类 C，聚类对象 w

输出: 修正后的 ε'

```
PathStack= PathFind( w, Vi)
u = the points' value of ε are minimum
do
    p = pop(PathStack)
Until p==u
ε = ε_min
for u in pop(PathStack)
    for ui belongs to the unmarked neighbors of u
        If N[ui] ≥ μ
            If σ(u,ui) ≥ ε
                If N_ε[ui] ≥ μ
                    Push(PathStack,ui)
                    u = ui
                Else
                    u = pop(PathStack)
                    Break
      if (u==w)
        u=the points' value of ε are minimum
            p = pop(PathStack)
            If( ε ≤ ε_min &&ε < ε)
                ε = ε_min
    end for
end for
ε' = ε
Return ε'
```

回溯方法实际上就是暴力穷举，此种方法还是将所有的路径都进行了计算，因此要对第二个问题进行优化。

第二方面，提出剪枝策略减少多余路径的计算：

（1）回溯位置选择：因为首先要找到一条路径，例如路径，V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，选择出此条路径下节点最小的ε作为候选的ε，假设点 u 的ε是最小的。

这样可以确保路径上的其他节点都会为核心点。此时回溯时并不是在上部分提到的 V_m 处，而是在点 u 的前一节点 $V_{u\text{-}1}$ 处开始回溯，因为我们要找所有路径的最小的ε里的最大值，这样我们回溯时要找的ε'应该是大于当前的ε且满足$\varepsilon < \boldsymbol{\varepsilon}$，所以若我们回溯时还是在 why-not 节点 w 前的 V_m 节点进行，那这样即使找到到 w 的路径，其最小的ε值还是 ε，所以对于路径 V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w 上的 u，…，V_m，处进行回溯是无意义的。

（2）回溯剪枝条件：对于路径 V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，当从 $V_{u\text{-}1}$ 处进行回溯，回溯时判断该点的邻居节点是否被访问过，选择一个未被访问过的邻居节点假设为 ui。判定条件一：判断 $\mathrm{N}[ui] \geq \mu$，若成立，则判定条件二：判断$\sigma(u, ui) \geq \varepsilon$是否

成立，即 u 与 ui 是否是结构相似的，若成立，则进行判定条件三：判断$N_{\varepsilon}[ui] \geq \mu$是否成立（此时的$\varepsilon$是根据最初始的路径基准确定的），若成立则，要继续访问ui的未被标记的邻居重复进行判定条件一二三的判断。判定条件二不成立的话，则回溯到 ui 的上一层节点 $V_{u\text{-}1}$ 处，继续找其他未被访问的邻居节点重复进行判定条件一二三的判断。判定条件二，如果不成立则直接访问节点 u 未被访问标记的邻居节点重复进行判定条件一二三的判断。判定条件一若不成立则直接访问节点 u 未被访问标记的邻居节点重复进行判定条件一二三的判断以上是对 4.2.2 节提出的基本算法的优化，此优化算法的伪代码如算法 4.4 所示。

该算法首先通过 PathFind 找到一条路径（第 1 行），注意在 w 到 Vi 的路径需满足路径上的每条边都得要是结构相似的。例如路径，V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，选择出此条路径下该节点所在的边结构相似性和该节点与邻居节点之间结构相似性相比最小的ε作为候选的ε，假设点 u 的ε是最小的（第 2 行）。所以$\varepsilon = \varepsilon_{min}$。这样确定路径上的其他节点都会为核心点。此时在点 u 的前一节点 $V_{u\text{-}1}$ 处开始回溯，u 点后的点都出栈（3-5 行）。对于路径 V_{target}，V_i，…，$V_{u\text{-}1}$，u，…，V_m，w，当从 $V_{u\text{-}1}$ 处进行回溯，在回溯时，判断该点的邻居节点是否被访问过，选择一个未被访问过的邻居节点假设为 ui（7-9 行）。判定条件一：判断 $\mathrm{N}[ui] \geq \mu$（第 10 行），若成立，则进行判定条件二：判断$\sigma(u, ui) \geq \varepsilon$是否成立，即 u 与 ui 是否是结构相似的（第 11 行），若成立，则进行判定条件三：判断$N_{\epsilon}[ui] \geq \mu$是否成立（第 12 行），若成立则，要继续访问$ui$的未被标记的邻居重复进行判定条件一二三的判断。判定条件三不成立的话，则回溯到 ui 的上一层节点 u 处，继续找其他未被访问的邻居节点重复进行判定条件一二的判断。判定条件二，若不成立则直接访问节点 u 未被访问标记的邻居节点重复进行判定条件一二三的判断（13-21 行），同理判定条件三不成立则直接访问节点 u 未被访问标记的邻居节点重复进行判定条件一二三的判断，且每次遇到 u==w 的情况，即已经选出一条路径，那么

需要更新ε为最新的大于原ε，且满足$\varepsilon < \boldsymbol{\varepsilon}$（22-29 行）。最后将$\varepsilon'$修改成最终的$\varepsilon$并返回$\varepsilon'$（30,31 行）。

4.4 本章小结

本章对支持 why-not 问题解释的社团发现算法(pSCAN 算法)具体了实现进行详细的介绍，分别在两个参数方面，即密度阈值参数μ和相似度阈值参数ε进行 why-not 问题解释算法进行研究与实现。在本章中首先探究了密度阈值参数μ对聚类结果的影响，得出相似度阈值在减小的时候能够保持属于原聚类的节点依然属于该聚类，但聚类集合会增大的结论。

基于探究出来的影响结论，能够确定密度阈值参数μ需要取值的范围。其次给出了通过修改相似度阈值参数去解释 why-not 问题的基本解释算法。最后因为提出的基本解释算法存在不足，所以为解决基本解释算法中的问题与不足，对基本解释算法进行了改进优化。同理，按照同样思想介绍相似度阈值参数的修改算法。

第 5 章 系统实现与分析

由于支持 wny-not 问题解释的社团发现系统中数据采集、数据处理以及可视化展示三个功能模块基于现有的已成熟的工具实现的，而支持 why-not 问题解释的社团发现算法是自己提出的，是支持 why-not 问题解释的社团发现系统的实现核心。所以本章将首先对提出的支持 why-not 问题解释的社团发现算法进行实验分析，然后对系统的部分功能进行测试。

5.1 支持 why-not 问题解释的自适应社团发现算法测试

本节将对支持 why-not 问题解释的社团发现算法进行实验分析，由于社团发现 Pscan 算法有两个参数，即密度阈值参数μ和相似度阈值参数 ε。所以对社团发现的 why-not 问题解释是从这两方面进行解释的。本节将从不同规模数据集下算法修改不同参数的算法性能、不同参数设定下的基本算法与改进算法共四个算法的性能对比、以及不同的聚类参数对支持 why-not 问题解释的社团发现算法的影响这三个方面对算法进行分析。

5.1.1 实验设置

本节实验所用到的硬件环境和软件环境如表 5.1 所示，并使用 GNU C++实现了 pSCAN 算法及社团发现算法 pSCAN 的 why-not 问题解释算法。

表 5.1 实验环境

Table 5.1 Environment of experiment

类别	描述	
硬件环境	CPU	Intel(R) 3.40GHz Core(TM) CPU i7
	内存	8GB
	硬盘	1TB
操作系统	Ubuntu(Linux) 14.04	
编程环境	GNU C++	

本章将在两个现成的数据集和一个 weibo 数据集上进行实验，两个现成的数据集属于 SNAP（Stanford Large Network Dataset Collection）实验数据集，分别是 ca-GrQc 数据集、DBLP 数据集，而 WeiBo 数据集是利用 Scrapy 框架爬取来的。

(1) ca-GrQc 数据集(General Relativity and Quantum Cosmology)：该数据集数据涵盖了 1993 年 1 月至 2003 年 4 月（124 个月）期间的论文。如果作者 i 与作者 j 合著了一篇论文，图中包含一个从 i 到 j 的无向边。如果这篇论文是 K 个作者共同创作的，那么

在 k 个节点上生成一个完全连通（子）图。该数据集共有 4158 个节点，即有 4000 多个作者，他们之间的联系由边表示共有 13422 条边

(2) DBLP 数据集：该数据集 DBLP 提供了一个关于计算机科学研究论文的全面清单。若有两个作者至少发表了一篇论文，那么他们之间有一条边以此表示他们是有联系的，发表在某一期刊或会议上的作者则可以组成一个社区。该数据集一共包含 4039 个作者在图中用节点表示，88, 234 条边表示节点之间的关系。

(3) WeiBo 数据集：该数据集是利用 Scrapy 框架爬取而来，其中包括有 82754 条边，4,000 个对象点。

本章对在不同参数设定下提出的解释算法的性能进行评估，包括不同数据集的大小，各算法性能比较以及还对不同参数设定下的解释算法在同样数据集下进行对比，比较两个解释算法的优劣。评估的解释算法包括修改 ε 和μ的基本解释算法 BaseRefine-X(X 表示 ε 或者μ)和改进解释算法 ImRefine- X(X 表示 ε 或者μ)。

5.1.2 解释算法在不同数据集规模下的实验分析

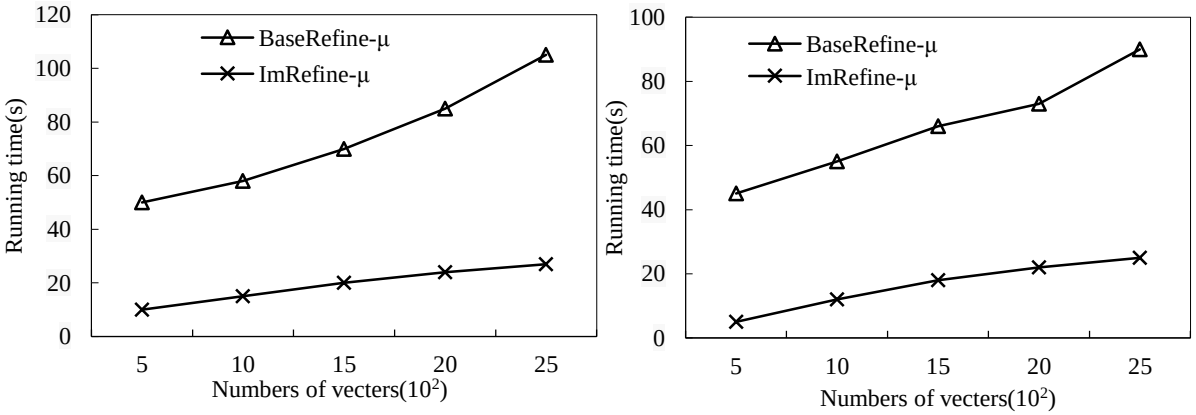
在本文针对不同的参数设定下修改 ε 和μ的解释算法，每个设定下包含基本的解释算法 BaseRefine 以及改进解释算法 ImRefine。所以本小节对于不同设定下的基本算法与改进算法进行测试实验，探究不同规模数据集对两个算法的影响。

(1) 密度阈值μ的解释算法在不同规模数据集的实验分析

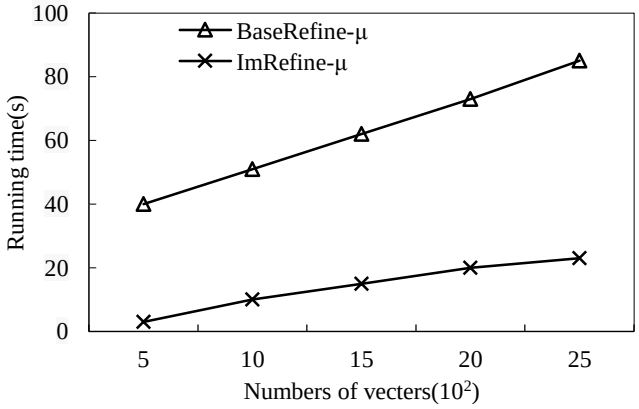
本部分实验主要进行关于密度阈值μ的解释算法在不同规模数据集下的实验，分析对比基本解释算法与改进解释算法的性能。对于实验中的三个数据集，每个数据集都被分为五个不同规模大小的分别为 500,1000,1500,2000,2500 个对象点。对于实验中的不同数据集，虽然较小的数据集是属于较大数据集的，但是对相同的 why-not 问题来说，不同规模的数据集依据密度阈值解释算法求解出来的密度阈值μ不一定是相同的，因为在增大数据集的时候会引入新的回溯路径，在新的路径上有可能会计算出能够解决这个 why-not 问题的新的或更合适的密度阈值解释值。而对于不同规模的数据集，两个解释算法的执行时间在趋势上应该是逐渐增加的，因为数据的规模越大执行算法时需要寻找的路径就会越多，消耗的时间就越大。

图 5.1 给出了修改密度阈值算法的基本算法以及改进算法在不同规模数据集上的执行时间。从图 5.1 中能够看到该 why-not 问题来说，正如我们所想的，两个算法随着数据规模的增加执行的时间也在增加，原因就如上面所提到的数据集规模越大，其需要寻找的路径就越多需要的执行时间就越久。在图 5.1 中我们还可以观察出在执行时间上密度阈值解释算法的改进算法比基本算法所用的执行时间更短。根据算法的具体执行

思想可以知道，基本算法是遍历了所有的路径从而得出其最佳的应该修改的密度阈值参数，而改进算法实在遍历路径的基础上提出了搞笑的剪枝策略，能够提前将不需要遍历的路径剪枝掉，这样也大大的降低了解释算法的执行效率。图 5.1 中对基本解释算法与改进解释算法的执行时间变化趋势进行对比可以发现，基本解释算法的执行时间增长速度比改进解释算法的执行时间增长速度大。这是因为对于改进解释算法数据集规模越大其剪枝策略的效果越明显，不需要计算的路径就越多，这样即使是数据规模变大了，其要寻找的路径也不会增加太多。



(a) ca-GrQc 数据集 (b) DBLP 数据集



(c) WeiBo 数据集

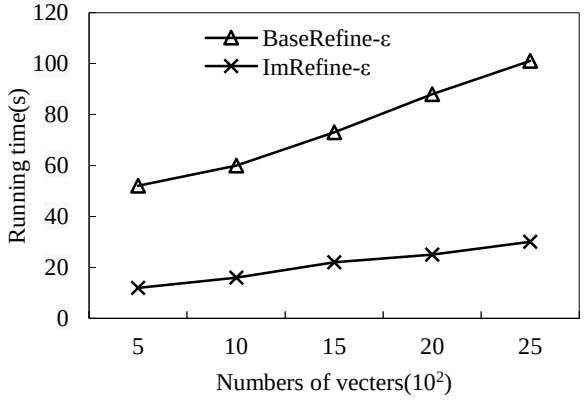
图 5.1 修改密度阈值μ的两种算法的执行性能

Fig. 5.1 The performance of algorithms BaseRefine-μ and ImRefine-μ

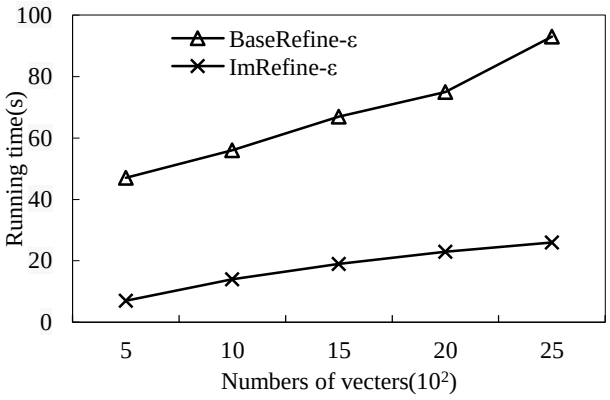
修改密度阈值μ解释算法的测试实验表明，基本解释算法与改进解释算法都可以对密度阈值的 why-not 问题进行解释，两个算法的执行时间都随着数据集的规模增加而增加，但改进解释算法的增幅更小，时间效率更高。相比较而言修改密度阈值μ的改进算法能够更有效地解释 pSCAN 聚类算法中的 why-not 问题。

(2) **相似度阈值 ε 的解释算法在不同规模数据集的实验分析**

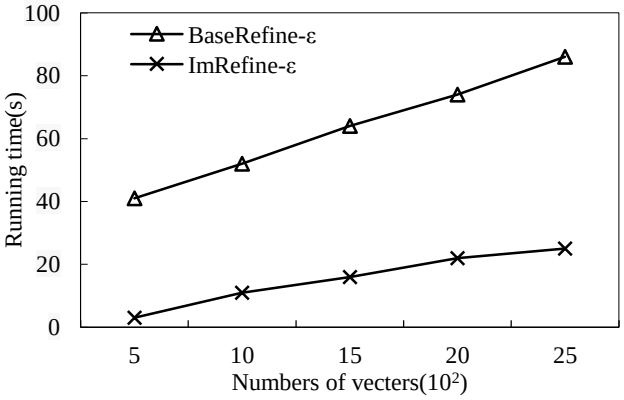
本部分实验主要进行关于相似度阈值 ε的解释算法在不同规模数据集下的实验，如密度阈值一样，本部分主要分析对比相似度阈值的基本解释算法 BaseRefine- ε与改进解释算法 ImRefine- ε的性能。与密度阈值解释算法分析一样，对于实验中的三个数据集，每个数据集也都被分为五个不同规模大小，分别代表 500,1000,1500,2000,250 个对象点。当然同样的，对于实验中的不同数据集，虽然较小的数据集是属于较大的数据集的，但是对相同的 why-not 问题来说，不同规模的数据集在依据密度阈值解释算法求解出来的相似度阈值ε不一定是相同的，分析其原因同上。而对于不同规模的数据集，两个解释算法的执行时间在趋势上应该是逐渐增加的，因为数据的规模越大执行算法时需要寻找的路径就会越多，消耗的时间就越大。



(a) ca-GrQc 数据集



(b) DBLP 数据集



(c) WeiBo 数据集

图 5.2 修改相似度阈值 ε 的两种算法的执行性能

Fig. 5.2 The performance of algorithms BaseRefine- ε and ImRefine- ε

图 5.2 给出了修改相似度阈值算法的基本算法以及改进算法在不同规模数据集上的执行时间。从图 5.2 中能够看到该 why-not 问题来说，正如我们所想的，两个算法随着数据规模的增加执行的时间也在增加，分析其原因是数据集规模越大，其需要寻找的路径就越多需要的执行时间就越久。在图 5.2 中我们也能观察出在执行时间上相似度阈

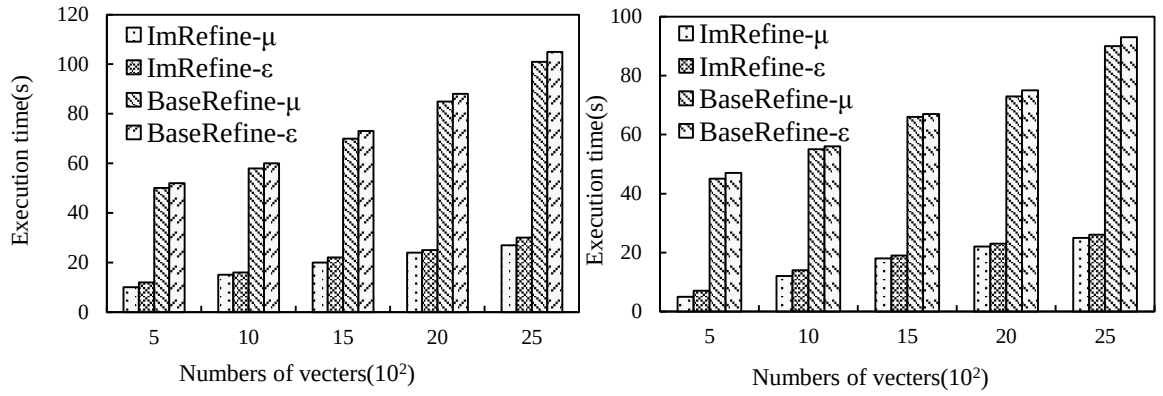
值解释算法的改进算法比基本算法所用的执行时间更短。同样的根据算法思想分析，基本算法是遍历了所有的路径从而得出其最佳应该修改的相似度阈值参数，而改进算法是在遍历路径的基础上提出了高效的剪枝策略，能够将不需要遍历的路径提前剪枝掉，这样大大的降低了解释算法的执行效率。图 5.2 中对基本解释算法与改进解释算法的执行时间变化趋势进行对比也能够发现与密度阈值解释算法相同的特点，基本解释算法的执行时间增长速度比改进解释算法的执行时间增长速度大，即相似度阈值解释算法比基本解释算法更快速且高效。

修改相似度阈值 ε解释算法的测试实验表明，基本解释算法与改进解释算法都能够对相似度阈值的 why-not 问题进行解释，两个算法的执行时间都是随着数据集的规模增大而增大，但改进解释算法的增幅更小，时间效率也更高。相比较而言修改相似度阈值 ε的改进算法能够更有效地解释 pSCAN 聚类算法中的 why-not 问题。

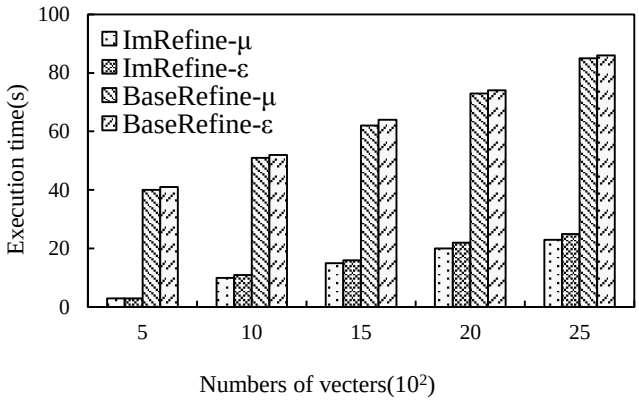
5.1.3 在ε与μ参数设定下的基本与改进算法性能

本部分的实验主要是对不同参数设定下的基本解释算法与改进解释算法之间进行分析和对比探究修改密度解释算法的基本算法和改进解释算法以及修改相似度解释算法的基本算法和改进算法这四个算法的时间效率。本小节对比实验中，与前两个实验分析一样，对于实验中的三个数据集，每个数据集也都被分为五个不同规模大小，分别代表 500，1000，1500，2000，2500 个对象点是由横轴表示，而纵轴表示算法的执行时间。测试结果如图 5.3 所示。

同样的，不同参数设定下的基本解释算法，因为需要遍历所有的路径去计算与比较进而确定要修改成的参数值，因为路径量比较大，导致算法的运行时间很大。从图 5.3 中能够看出改进算法有着显著优势其执行时间与基本解释算法相比缩短很多，原因在实验一二中进行过分析，主要是因为改进解释算法通过剪枝策略减少了大量的无用路径的计算，并且减少内存的使用，这使得计算代价大大降低。另外，还可以从图 5.3 看出四个算法中，密度阈值中的改进算法执行时间最短，相似度阈值的改进解释算法执行时间稍长，而其基本解释算法的执行时间最长。基于相似度阈值的解释算法比密度阈值参数的解释算法多进行了一步计算，即需要计算一条路径上每个点与其邻居点的节点相似度，进而找出节点相似度的最小值。这个步骤增加了相似度阈值参数解释算法的计算代价，使得执行时间增大。



(a) ca-GrQc 数据集 (b) DBLP 数据集



(c) WeiBo 数据集

图 5.3 四种算法的运行时间比较

Fig. 5.3 The running time of four algorithms

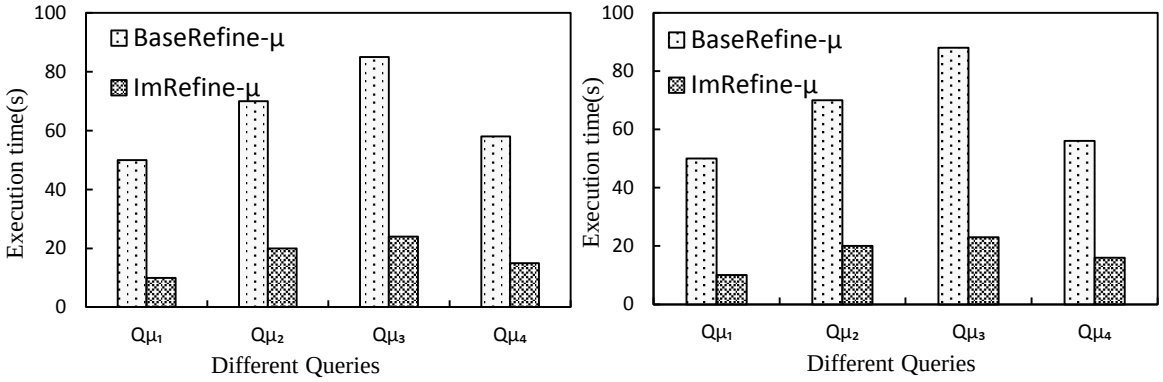
本小节的实验结果表明，基于密度阈值参数下的改进解释算法执行时间最短，基于相似度阈值参数下的基本解释算法执行时间最长。改进解释算法相对于基本解释算法执行效率明显更好。基于密度阈值参数的改进解释算法执行效率比基于相似度的效果更好一些。

5.1.4 不同聚类参数对解释算法的影响

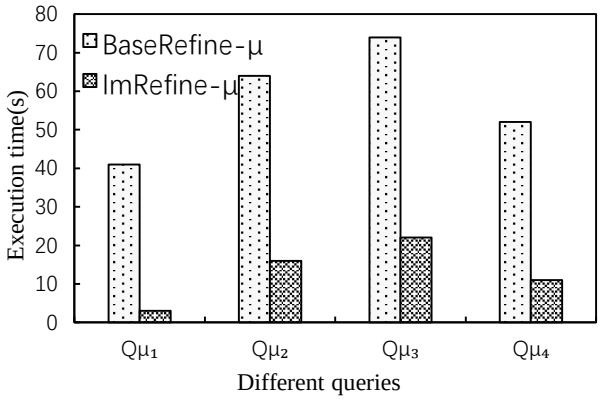
本小节主要针对不同参数设定下解释算法的不同聚类参数进行测试实验，探究不同的聚类参数对不同参数设定即基于密度阈值参数和相似度阈值参数下的基本解释算法 BaseRefine- X(X 表示 ε 或者μ)以及改进解释算法 ImRefine- X(X 表示 ε 或者μ)的影响。

(1) 不同查询参数对密度阈值μ的解释算法的影响

在本部分实验中，横轴表示的不同查询参数即$Q_{\mu_1}, Q_{\mu_2}, Q_{\mu_3}, Q_{\mu_4}$，纵轴表示的则是基本解释算法 BaseRefine-μ与改进算法 ImRefine-μ的执行时间。实验结果如图 5.4 所示，对于不同的数据集上的进行不同查询的 why-not 问题，基于密度阈值参数的解释算法的解释时间都是非常不同的。



(a) ca-GrQc 数据集 (b) DBLP 数据集



(c) WeiBo 数据集

图 5.4 密度阈值解释算法下不同查询的解释时间

Fig. 5.4 Explanation time of different queries about BaseRefine-μ and ImRefine-μ

从图 5.4 中可以看出，对于三个数据集来说，不同的数据集对于其数据集上不同的查询参数所用到的执行时间也是大不相同，并且这种情况对基本解释算法 BaseRefine-μ以及改进解释算法 ImRefine-μ都是同时成立的，还可以观察到两个解释算法的变化幅度没有具体的规律可言。造成这种情况的原因是对于不同的数据集来说每个数据集中的对象分布是不同的，这样的到的聚类结果也是不一样的。这样的话会导致为 why-not 问题计算候选参数时间大不相同，如图 5.4(a)所示。

对于同一数据集上不同查询的 why-not 问题，解释算法的解释时间也是有很大的不同的。因为对于不同的查询参数，同一数据集上的数据分布式是一样的，但是因为参数

不同，那么聚类的结果是不一样的，因此期望簇以及 why-not 对象的是不同的，因此这样使得能够得到的密度阈值是不一样的，进而计算新的密度阈值时计算的路径页不尽相同。因此，对于相同数据集上的不同查询，基于密度阈值的基本解释算法 BaseRefine-μ与改进解释算法 ImRefine-μ的解释时间是不同的。从图 5.4 中还能够看到，基本解释算法 BaseRefine-μ的执行时间与改进解释算法 ImRefine-μ的执行时间相比，改进的解释算法在不同的数据集上以及在不同的查询参数上执行时间都是比基本改进算法 BaseRefine-μ短的且差距很大。

不同查询参数对解释算法的影响的测试实验表明，对于不同数据集上的不同查询的 why-not 问题，基于密度阈值参数的解释算法的解释时间是不尽相同的，对于相同数据集上的不同 why-not 问题，两个解释算法的解释时间也是不相同的，并且改进算法 ImRefine-μ的解释时间很短，而基本解释算法 BaseRefine-μ的解释时间很长。

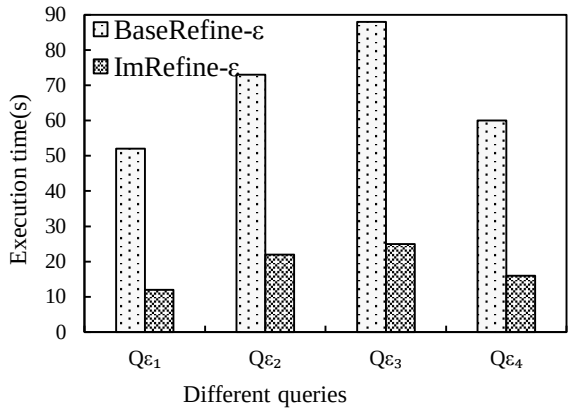
(2) 不同查询参数对相似度阈值ε的解释算法的影响

本部分实验中，还是采用不同的查询参数即$Q_{\varepsilon_1}, Q_{\varepsilon_2}, Q_{\varepsilon_3}, Q_{\varepsilon_4}$，探究基本解释算法 BaseRefine-$\varepsilon$与改进算法 ImRefine-$\varepsilon$的执行时间。同第一部分一样，实验结果如图 5.5 所示，对于不同的数据集上的进行不同查询的 why-not 问题，基于相似度阈值参数的解释算法的解释时间都是非常不同的。

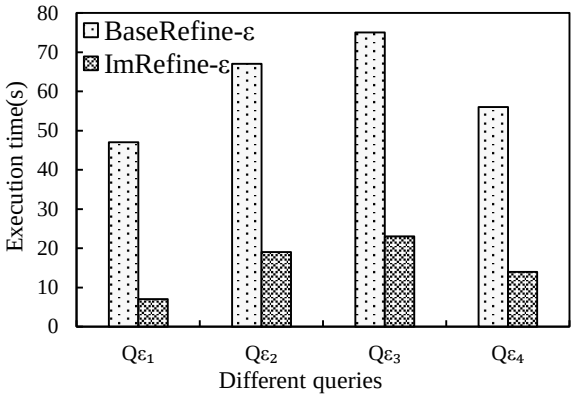
从图 5.5 中可以看出，对于三个数据集来说，不同的数据集对于其数据集上不同的查询参数所用到的执行时间也是大不相同，并且这种情况对基本解释算法 BaseRefine-ε以及改进解释算法 BaseRefine-ε都是适用的。其主要原因是对于不同的数据集来说每个数据集中的对象分布是不同的，这样得到的聚类结果是不一样的。因此会导致给 why-not 问题计算候选参数的时间大不相同，如图 5.5(a), 5.5(b), 5.5(c)所示。

同样，对于同一数据集上不同查询的 why-not 问题，解释算法的解释时间也有很大不同。因为对于不同的查询参数，同一数据集上的数据分布是一样的，但是因为参数不同，聚类的结果是不一样的，因此期望簇以及 why-not 对象是不同的，因此这样使得能够得到的相似度阈值是不一样的，计算新的密度阈值时需要计算的路径不尽相同。因此对于相同数据集上的不同查询，基于相似度阈值的基本解释算法 BaseRefine-ε与改进解释算法 ImRefine- ε的解释时间是不同的。从图 5.5 中还可看出，基本解释算法 BaseRefine-ε的执行时间与改进解释算法 ImRefine-ε的执行时间相比，改进的解释算法在不同的数据集上以及在不同的查询参数上执行时间都是比基本改进算法 BaseRefine-ε短且优化很多。不同查询参数对解释算法的影响的测试实验表明，对于不同数据集上的不同查询的 why-not 问题，基于相似度阈值参数的解释算法的解释时间是不同的，对于同一数据集

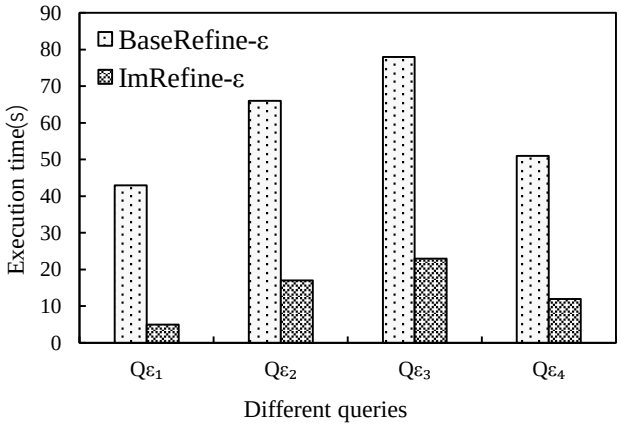
上的不同查询的 why-not 问题，两个解释算法的解释时间也是不同的，并且改进算法 ImRefine-ε的解释时间很短。



(a) ca-GrQc 数据集



(b) DBLP 数据集



(c) WeiBo 数据集

图 5.5 相似度阈值解释算法下不同查询的解释时间

Fig. 5.5 Explanation time of different queries about BaseRefine- ε and ImRefine- ε

5.2 系统测试

5.2.1 测试方法

本系统的测试方法依据现在的软件工程测试方法白盒测试和黑盒测试，按照系统的功能和模块依次进行。具体内容如下：

白盒测试是系统开发人员根据程序的控制流程，按照程序内部的执行逻辑对软件的过程性细节进行详细的检查，大到每一个模块，小到每一个方法甚至一条语句，通过不断的调试跟踪代码的输入和输出，对测试过程中出现的错误或者可能出现的错误做出记录并进行保存，从而方便后续的修改和处理。

黑盒测试则不像白盒测试要求的那么严格，它不要求测试人员必须是开发人员，相反测试人员是非开发人员是最好的，因为他们并不知道所测试部分的具体代码实现，而是当前测试结果是否达到他们期望的结果，它测试的是每个模块的功能是否可以正常使用并能达到想要的效果。黑盒测试是站在客户的角度，按照本文的操作手册而逐步进行展开，它主要是以界面和功能作为对象。

5.2.2 测试内容

本系统根据测试的方法不同，其完成的阶段也有所不同。其中，本系统的单元测试是在当前模块完成之后进行的，它是针对模块进行测试；集成测试是在各个模块集成之后进行的，它主要是测试模块相互之间是否能正常工作；系统测试是在最后完成的，它是在实际应用场景下进行的测试。

(1) 单元测试

单元测试的测试对象是系统中的各个模块，本系统的单元测试是由我个人来完成的，在测试过程中我会列出出现的问题，并针对出现的这些问题一个个的解决。单元测试是软件测试环节中的重要一步，是软件质量的保障，本系统通过这种测试可以尽可能的减少系统中出现的问题，而且可以避免集成测试过程中遇到的棘手问题。

(2) 功能测试

系统的功能测试是在集成测试的基础上，根据需求说明书来对系统的整体功能进行测试，这种测试可以是开发人员或其他人员，是一种黑盒测试，工作人员只需要测试系统的功能是否达到客户的要求即可。下面是从功能的角度出发，介绍系统的测试结果，包括采集功能、处理功能、why-not 问题解释功能、数据可视化，虽然等。由于本系统内的采集功能、处理功能以及数据可视化功能是由现有的工具实现的，所以仅着重侧测试 why-not 问题解释功能模块下的 pSCAN 社团发现功能以及解释算法的解释功能，同时因为数据可视化就是对 why-not 问题解释功能中的聚类结果展示，所以在测试 why-not 问题解释功能时直接通过查看可视化聚类结果对比即可。

(a) pSCAN 聚类算法

此功能下会输出两个文本文件，分别为点文件和边文件。其文件存储格式分别如表 5.2 和 5.3 所示。点文件中需要提供节点的 Id，节点的标签也就是说节点代表的含义，还有当前节点所处聚类。而对于边文件格式包含源和目的节点，边的类型(有向或无向)，边所属聚类以及边所代表的权重。

表 5.2 点文件格式

Table 5.2 Format of nodes file

Id	Label	occurrences
节点的 id	节点的标签	节点现在所处聚类

表 5.3 边文件格式

Table 5.3 Format of edges file

source	target	type	Id	Label	Weight
源节点	目的节点	边的类型	边所属聚类	边的标签	边的权重

对于 5.1.1 小节中提供的 ca-GrQc 数据集进行数据处理之后再进行 pSCAN 聚类其可视化的结果如图 5.6 所示

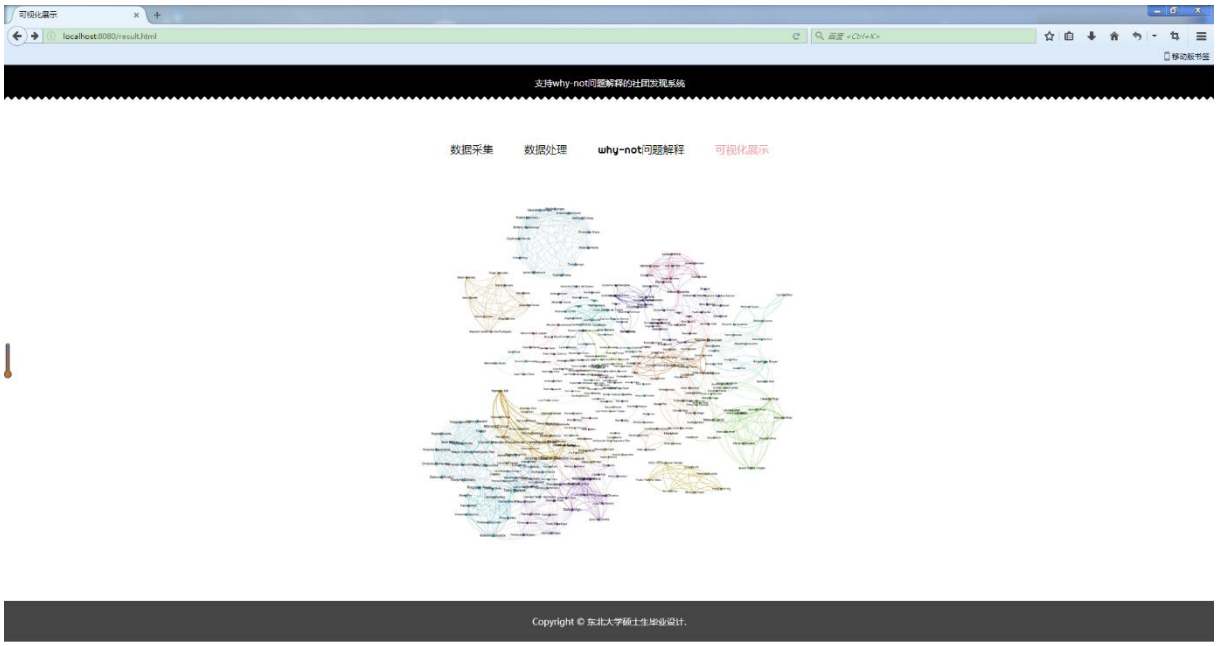


图 5.6 聚类结果可视化展示

Fig. 5.6 Visualization of clustering results

(b) why-not 问题解释算法

而为直观的看 why-not 问题解释实现的对比，则直接通过可视化结果进行对比。如图 5.7 中的 5.7(a)和 5.7(b)所示。

(a) 修改前密度阈值参数 μ

(b)修改后密度阈值参数μ

图 5.7 why-not 问题解释功能实现效果

Fig. 5.7 The realization effect of why-not problem interpretation function

通过图 5.7(a)和 5.7(b)的对比可以看出经过解释算法对参数的修改，原始聚类的参数中的密度阈值参数是 4，红框中的节点是灰色，表示不在一个聚类中，想要让矿中的节点被聚到紫色类中，经过解释功能模块的运算得出密度阈值参数应改为 3，达到图 5.7(b)的效果。

(3) 集成测试

集成测试是按照需求规格说明书的要求把多个已经经过单元测试的模块组装起来，也就是把系统中所有模块集中在一起进行测试，确保各个模块之间可以正常的交互。集成测试是系统测试的最后一个环节，这一步直接绝对系统是否可以进行部署，完成最后的开发任务。系统主界面如图 5.8 所示。

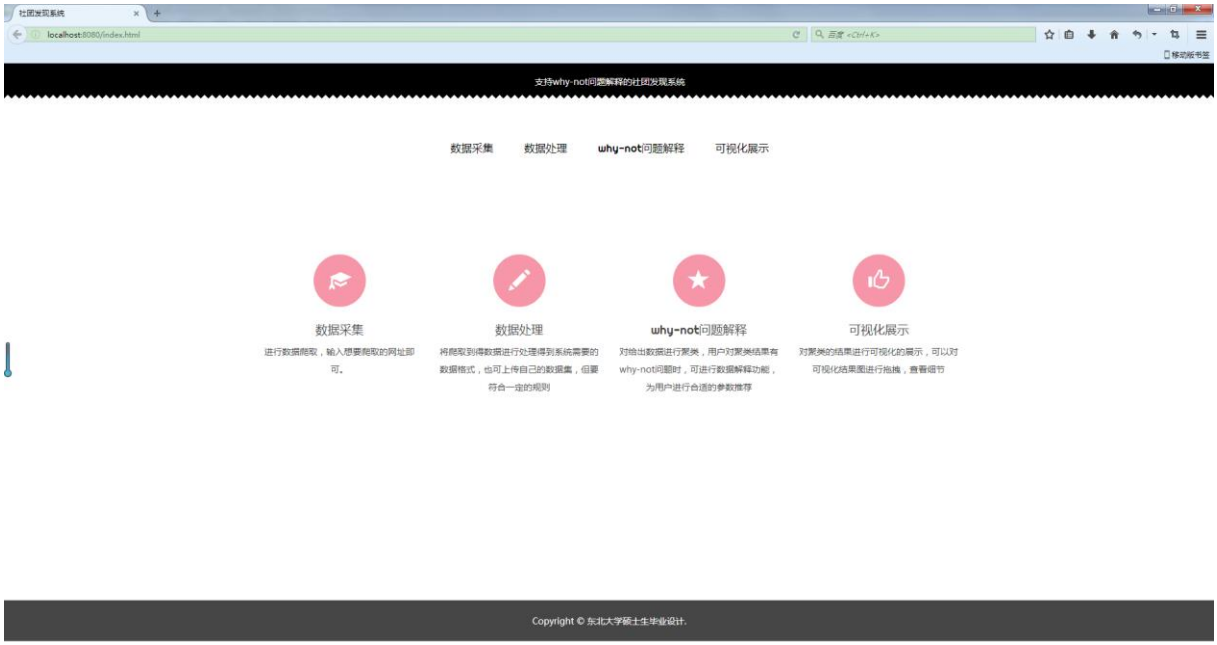


图 5.8 系统界面

Fig. 5.8 System interface

5.2.3 测试结论

本小节主要介绍系统中所用到的测试方法和测试内容。分别从白盒测试和黑河测试进行详细测试。然后介绍了本系统所测试的内容，分别从单元测试、集成测试和功能测试三个方面对测试内容进行描述。最后是对系统进行功能测试，分别从数据采集、数据清洗与加载、数据检索和数据操作四个方面进行测试。通过以上方法的测试，本系统在测试结果基本达到了预期的设计目标，符合预期结果。

5.3 本章小结

本章首先对本文提出的支持 why-not 问题解释的社团发现算法（pSCAN 算法）中的不同参数设定下的基本解释算法与改进解释算法进行了详细的测试和分析。测试数据是真实的 ca-GrQc 数据集、DBLP 数据集和爬取的 WeiBo 数据集，确保了本文测试结果的有效性。本文首先测试了不同参数设定下的基本解释算法与改进解释算法在不同规模数

据集下的执行效率，然后给出了不同参数设定下的基本解释算法与改进解释算法这四个算法执行效率的对比试验，最后也实验分析了不同聚类查询参数对解释算法的影响。最后进行了系统的测试，主要对 why-not 解释功能进行测试。本节的实验结果说明了本文提出的解释 pSCAN 算法中的 why-not 问题的解释算法的有效性和较好的可扩展性。进而也说明本文提出的聚类系统有着较好的有效性以及可扩展性。

第 6 章 总结与展望

6.1 本文总结

研究社团结构的发现简称社团发现可以在网络的局部结构上对节点与边的结构和功能有更好的理解。到目前为止社团发现问题的相关研究日渐成熟，但还是存在着问题，目前并没有能够支持 why-not 问题解释的社团发现系统，去解决聚类过程中出现的某个对象点未出现在期望聚类中。所以本文提出去设计并实现一个支持 why-not 问题解释的社团发现系统。本文的主要贡献为：

(1) 根据 pSCAN 聚类算法的思想以及 why-not 问题的定义，本文提出了 pSCAN 算法的 why-not 问题的定义。根据所了解的情况，目前不存在该问题的任何相关研究。

(2) 提出了基本解释算法与改进解释算法修改密度阈值μ解释 pSCAN 聚类算法中 why-not 问题的两个算法。本文中首先定义了修改密度阈值μ解释 pSCAN 聚类算法中 why-not 问题的定义，探究了密度阈值μ对 pSCAN 聚类结果的影响，并给出了如何求解修改密度阈值μ解释 pSCAN 聚类算法中 why-not 问题的算法。

(3) 提出了修改相似度ε解释 pSCAN 聚类算法中 why-not 问题的算法。本文中首先定义了修改相似度ε解释 pSCAN 聚类算法中 why-not 问题的定义，探究了相似度ε对 pSCAN 聚类结果的影响，并给出了如何求解修改相似度阈值ε解释 pSCAN 聚类算法中 why-not 问题的算法。

(4) 设计并实现一个支持 why-not 问题解释的社团发现系统，目前并没有这方面的系统的实现。

(5) 在两个现成并真实的数据集和一个网络爬取的数据集上，对支持 why-not 问题解释的社团发现算法执行效率分别进行了实验，并对实验结果进行了详尽的分析，实验结果说明了本文提出的支持 why-not 问题解释的社团发现算法的有效性和较好的可扩展性。

6.2 工作展望

本文根据 pSCAN 聚类和 why-not 问题的概念，提出了 pSCAN 聚类算法中的 why-not 问题的定义，对 pSCAN 聚类算法参数——密度阈值μ和相似度阈值ε对 pSCAN 聚类结果的影响进行了探究，并提出了支持 why-not 问题解释的社团发现算法，根据此算法完成支持 why-not 问题解释的社团发现系统的实现。虽然通过算法测试以及系统测试证

明支持 why-not 问题解释的社团发现算法能够有效的解释 pSCAN 聚类算法中的 why-not 问题，但对于该问题仍然有些问题需要解决：

(1) 本文提出利用回溯法修改密度阈值μ和修改相似度阈值ε的算法，需要遍历图上所有路径，造成了算法在执行时间上的巨大开销。如何降低算法执行过程中的路径查找次数，进而减少算法的执行时间，值得将来工作中进一步仔细进行研究。

(2) 本文提出的支持 why-not 问题解释的社团发现算法，只是单独的对两个参数进行修改，如何同时修改多个参数（如同时修改密度阈值μ和修改相似度阈值ε），更快地计算出修改后的参数值，值得在将来的工作中进行深入研究。

参考文献

[1] Newman M E J. The Structure and Function of Complex Networks[J]. Siam Review, 2003, 45(2): 167-256.

[2] Barabasi AL, Albert R. Emergence of scaling in random networks[J]. Science, 1999, 286(5439): 509-512.

[3] Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks.[J]. Nature, 1998, 393(6684): 440-442.

[4] Song C, Havlin S, Makse H A. Self-similarity of complex networks.[J]. Nature, 2005, 433(7024): 392-395.

[5] SBoccaletti, VLatora, YMoreno, et al. Complex Networks: Structure and Dynamics[J]. Complex Systems & Complexity Science, 2007, 424(4-5): 175–308.

[6] Wang X F, Chen G. Complex networks: small-world, scale-free and beyond[J]. Circuits & Systems Magazine IEEE, 2003, 3(1): 6-20.

[7] Scott J. Social network analysis: a handbook[M]. Sage Publica-tions, 2000.

[8] Milo R, Shenorr S, Itzkovitz S, et a1. Network Motifs: Simple Building Blocks of Complex Networks[J]. Science, 2002, 98: 824-827.

[9] Roger Guimer à, Lu ís A. Nunes Amaral. Functional cartography of complex metabolic networks[J]. Nature, 2005, 433(7028): 895-900.

[10] Palla G, Dereyi I, Farkas I, et al. Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society[J]. Nature, 2005, 435(7043): 814-818.

[11] Rosvall M, Bergstrom CT. An Information-theoretic Framework for Resolving Community Structure in Complex Networks[J]. Proceedings of the National Academy of Science, 2007, 104(18): 7371-7331.

[12] Rosvall M, Bergstrom CT. Maps of Random Walks on Complex Networks Reveal Community Structure[J]. Proc Natl Acad, 2008, 105(4): 1118-1123.

[13] Newman M E J, Girvan M. Finding and evaluating community structure in networks[J]. Physical review. E, Statistical, nonlinear, and soft matter physics, 2004, 69(2 Pt 2): 026113.

[14] Xu X, Yuruk N, Feng Z, et al. SCAN: a structural clustering algorithm for networks[C]. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2007: 824-833.

[15] Chang L, Li W, Lin X, et al. pSCAN: Fast and exact structural graph clustering[C].IEEE, International Conference on Data Engineering. IEEE, 2016: 253-264.

[16] Chapman A, Jagadish H V. Why not?[C]. ACM SIGMOD International Conference on Management of Data. ACM, 2009: 523-534.

[17] Ester M, Kriegel H, Sander J, et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise [C]. Proceedings of International Conference on Knowledge Discovery and Data Mining, 1996, 96(34): 226-231.

[18] 孙立伟, 何国辉, 吴礼发. 网络爬虫技术的研究[J]. 电脑知识与技术, 2010, 06(15): 4112-4115.

[19] Bo Y, Liu D Y, Liu J, et al. Complex Network Clustering Algorithms[J]. Journal of Software, 2009, 20(1): 54-66.

[20] Yang N, Lin S X, Gao Q, et al. Discovering signature of potential Web communities from clusters of MCL[J]. Chinese Journal of Computers, 2007, 30(7): 1086-1093.

[21] Arenas A, Díaz-Guilera A, Pérez-Vicente C J. Synchronization reveals topological scales in complex networks[J]. Physical Review Letters, 2006, 96(11): 102-114.

[22] He Z, Lo E. Answering Why-not Questions on Top-k Queries[C]. IEEE, International Conference on Data Engineering. IEEE Computer Society, 2012: 750-761.

[23] Zhou R, Liu C, Islam M S. On answering why-not questions in reverse skyline queries[C]. IEEE, International Conference on Data Engineering. IEEE, 2013: 973-984.

[24] Tran Q T, Chan C Y. How to ConQueR why-not questions[C]. ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, Usa, June. DBLP, 2010: 15-26.

[25] Huang J, Chen T, Doan A H, et al. On the Provenance of Non-Answers to Queries over Extracted Data [J]. PVLDB, 2008, 1(1): 736-747.

[26] João B, Rocha-Junior. Top-k spatial keyword queries on road networks[C]. International Conference on Extending Database Technology. ACM, 2012: 168-179.

[27] Chen L, Lin X, Hu H, et al. Answering why-not questions on spatial keyword top-k queries[C]. IEEE, International Conference on Data Engineering. IEEE, 2015: 279-290.

[28] Meliou A, Gatterbauer W, Moore K F, et al. Why so? or Why no? Functional Causality for Explaining Query Answers[J]. Eprint Arxiv, 2009.

[29] 胡海波. 复杂网络拓扑结构的研究[D]. 西安理工大学, 2006.

致谢

时间转瞬即逝，两年半的硕士研究生的学习生涯就要结束了，突然感到有些悲伤，这不仅仅意味着研究生生活即将结束，同样意味着学生时代将要画上句号。虽然时间短暂，但是却也是我人生经历重要蜕变的两年半。在东北大学学习生活的两年半里，我有幸结识了许多良师益友，是他们教我如何克服困难，让我懂得如何更好地生活。回首两年半的时间所走过的路，心中充满了无限感慨，更有感激。也就是在这两年半的时间里，我才开始真正意义上的接触科研，从一个一无所知的科研门外汉慢慢走上学术研究的道路。从读论文到思考，从做实验到写论文，我认真经历了科研中的每一个环节。每天生活虽过得忙碌，但却非常充实。

值此论文完成之际，首先要向我的导师王斌老师表示衷心的感谢与诚挚的敬意，王老师将我带进了这个富有创造力和活力的实验室，让我结识了这些优秀而又不乏幽默的同学，使两年半的科研生活不再枯燥，充满乐趣。王老师专业知识渊博，思维敏捷，对我的科研工作予以热心指导，我在这两年中取得的点滴进步都离不开王老师的谆谆教诲，王老师的督促和指导保障了我的研究工作的顺利进行。王老师的幽默风趣总能将实验室的气氛活跃起来，有他在的地方从来不乏欢乐与笑声。感谢王老师这两年来对我的关心和指导。

感谢软件所的杨晓春老师。这两年来的研究生学习生活中，我常常感受到杨老师严肃认真的科学态度，严谨执着的学术精神，精益求精的工作作风，为我以后的工作生活树立了很好的榜样。杨老师在学术上孜孜不倦启发我们学术研究的灵感，在生活中待人和蔼可亲，给予我们热情亲切的关心，让我们体会到了家庭般的温暖。在此谨向您表示最衷心的感谢。

此外，本文的顺利完成，还要感谢实验室各成员的帮助与支持。感谢宗传玉师兄在学习上对我的指导，感谢杨凯、李思垚、王凯、胡金林、郭策、郭润东、梁慧超、李金旭和任开毓同学，与你们一起学习的两年半收获颇多，感谢实验室的全体师兄师姐师弟师妹们，很高兴与你们共同相聚在东大，携手并进，度过两年半的快乐时光。感谢我最好的小伙伴杨茗周和刘丽然，因为有你们，在东大的日子一直温暖。还要感谢宿舍室友郭文静、谷斯琪、朱春颖和周琪琪同学两年半的陪伴，谢谢你们的关心和照顾。

最后，我还要感谢我的父母、家人和朋友们，正是因为有你们无限的关爱、理解和支持，我才能够顺利的完成课题的研究及论文的写作。在未来的日子里，我会更加努力的学习和工作，不辜负你们对我的殷切期望！

攻硕期间参与项目、发表论文、参加测评及获奖情况

参加项目：

1. 国家自然科学基金项目：“溯源驱动的弱可用性轨迹数据管理关键技术”（项目编号：61572122），2016.01-2019.12
2. 国家自然科学基金重点项目：“面向物联网搜索的隐私保护理论和关键技术”（项目编号：61532021）， 2016.01-2020.12

获奖情况：

1. 获得 2015-2016 年度东北大学硕士研究生二等奖学金
2. 获得 2016-2017 年度东北大学优秀团干部
3. 获得 2016-2017 年度东北大学硕士研究生一等奖学金
4. 获得 2017-2018 年度东北大学硕士研究生一等奖学金