密　　级：
分 类 号：
学校代码：10075
学　　号：20207014042

硕士学位论文

大数据平台图分区策略及社区发现研究

学 位 申 请 人：翟晓梦
指　导　教　师：张虹　副教授
专业学位类别：电子信息硕士
专业学位领域：计算机技术
院　系　名　称：网络空间安全与计算机学院
答　辩　日　期：二〇二三年六月

Classified Index: **CODE: 10075**

U.D.C.: **No.20207014042**

Thesis for the Degree of Master

Study on graph partitioning strategies and community detection on big data platforms

Candidate: Zhai Xiaomeng

Supervisor: A.P. Zhang Hong

Professional degree category: Master of Electronic and Information Engineering

Professional degree field: Computer Technology

College: School of Cyber Security and Computer

Date of Oral Defense: June, 2023

摘 要

复杂网络作为图结构的数据，在多个领域中被广泛研究，研究的网络涉及社交网络、交通网络、生物网络等。社区发现作为一种揭示潜在结构的有效技术，能挖掘出网络中的隐藏信息。但随着复杂网络数据量的快速增长，不得不使用分布式的图计算平台来处理庞大的网络。大规模的网络分配到多个机器进行计算，需要分成若干个部分。如何划分大规模网络来提高图计算平台的执行效率是一个重要问题。本文通过对图分区的研究对 Graphx 分布式图计算平台的图存储进行了优化，在此基础上进一步对社区发现进行了研究：

（1）针对图存储的优化，本文提出了基于程序分析的图分区框架以及基于分区策略的优化算法。现有的大多数分布式图分区方法几乎没有考虑到划分策略与图算法特性之间的关系。本文结合图算法的特点，提出了一种分布式图分区框架，该框架可以在程序分析的基础上决定哪种划分策略更好。其次，设计了一种基于三角形的划分策略，该策略受益于三角计数等多种图算法。此外，为了减少 PageRank 算法中的 shuffle 操作的次数，本文重新设计了一个基于哈希分区的 PageRank 算法。实验结果表明，无论在真实图还是合成图上，本文提出的自适应图分区框架，在三角计数和 PageRank 方面都获得了很好的表现。

（2）本文提出了一种基于核心和边缘节点的社区发现算法。该算法主要分为三部分：核心节点的选择、基于核心和边缘节点的标签传播以及标签重分配。首先，使用改进的密度峰值聚类计算节点的重要性，再利用切比雪夫不等式选择核心节点。然后通过改进的最短路径算法和基于信任的距离度量为边缘节点分配标签，结合核心节点和边缘节点进行标签传播，为未分配标签的节点分配标签，可以根据实际情况设置标签的数量。最后，为节点标签与其邻域标签不匹配的节点重新分配标签，以提高标签分配的准确性。本文使用真实网络和合成网络分别与检测非重叠社区的算法和检测重叠社区的算法进行比较。实验表明，与非重叠社区发现算法和重叠社区发现算法相比，本文提出的算法具有更好的准确性和更高的执行效率。

关键词 图计算 图分区 社区发现 复杂网络 Graphx

Abstract

Complex networks as graph-structured data are widely studied in several fields, and the studied networks involve social networks, transportation networks, biological networks, etc. Community discovery serves as an effective technique to reveal the underlying structure and unearth the hidden information in the network. However, with the rapid growth of data volume in complex networks, distributed graph computing platforms have to be used to handle the huge networks. Large-scale networks are distributed to multiple machines for computation and need to be divided into several parts. How partitioning large-scale networks can improve the execution efficiency of graph computing platforms is an important issue. In this paper, the graph storage of the Graphx distributed graph computing platform is optimized through the study of graph partitioning, based on which the community further finds that.

(1) For the optimization of graph storage, a graph partitioning framework based on program analysis and two optimization algorithms based on partitioning strategies are proposed. Most existing distributed graph partitioning methods barely consider the relationship between partitioning strategies and graph algorithm properties. In this paper, we combine the characteristics of graph algorithms and propose a distributed graph partitioning framework which can decide which partitioning strategy is better on the basis of program analysis. Next, a triangle-based partitioning strategy is designed which benefits from a variety of graph algorithms such as triangle counting. In addition, a hash partitioning-based PageRank algorithm was redesigned in order to reduce the number of shuffle operations in the PageRank algorithm. Experimental results show that the adaptive graph partitioning framework proposed in this paper achieves good performance in both triangle counting and PageRank, both on real and synthetic graphs.

(2) A community discovery algorithm based on core and edge nodes is proposed. The algorithm is divided into three parts: core node selection, label propagation based on core and edge nodes, and label redistribution. First, the importance of nodes is calculated using improved density-peak clustering, and then the core nodes are selected using Chebyshev's inequality.

Then labels are assigned to edge nodes by an improved shortest path algorithm and a trust-based metric, and labels are propagated by combining core and edge nodes to assign labels to nodes that are not assigned labels, and the number of labels can be set according to the actual situation. Finally, nodes whose node labels do not match their neighborhood labels are reassigned to improve the accuracy of label assignment. The algorithms for detecting non-overlapping communities and those for detecting overlapping communities are compared using real and synthetic networks, respectively. The experiments show that the algorithm has better accuracy and higher efficiency compared to the non-overlapping community discovery algorithm and the overlapping community discovery algorithm.

Keywords Graph computation Graph partitioning Community detection Complex networks Graphx

目 录

第一章　绪论

1.1　研究背景与意义

在现实生活中，任何一个复杂的问题都可以生成由节点和边组成的复杂网络。在不同的复杂网络中节点和边的含义也不同。比如在社交网络[1]中的节点表示某个人，而边表示相连的两个人之间的关系。在网页链接图中的节点表示网页，边表示网页之间的链接关系。还有铁路运输网络[2]、疾病传播网络、生物网络[3]、科学家合作网络等。在现今的大数据时代背景下，各种数据急剧增长，其中由节点和边组成的复杂网络也不例外。单个机器已经无法处理庞大的网络，因此对于复杂网络的研究需要分布式图计算平台的支撑。Graphx 作为 Spark 上的分布式图计算框架，具有很强的可扩展性，稳定性以及高容错性。有了分布式图计算平台的支持，就可以在大规模复杂网络上更高效的计算和处理。

在复杂网络中存在一个重要特征，那就是社区结构，社区结构是指一组内部联系相对紧密、外部联系相对稀疏的节点[4]。作为网络中的隐藏结构，社区结构对于理解网络功能、研究网络拓扑结构、揭示网络中存在的规律、预测网络演化具有重要意义。对社区的研究有助于在社交媒体中寻找潜在朋友，为不同类型的用户针对性的推荐产品。在万维网中根据不同类型的数据进行社区发现，可极大的提高搜索效率。挖掘出这些有价值的信息具有现实意义。因此对复杂网络中社区结构的发现和分析引起了众多学者的关注，同时也出现了许多社区发现算法。

社区发现算法包括非重叠的社区发现算法和重叠的社区发现算法。然而，一个节点可能存在于多个社区。例如，在一个社交网络中，一个人可以是足球俱乐部的成员，同时也可以是篮球俱乐部的成员。因此，对重叠社区的研究更具有现实意义。

但在分布式图计算框架 Graphx 中，大规模网络会被划分成多个小图，分别分配到分布式集群的各个机器中进行存储。在分配的同时要尽可能保证分配均衡，否则会出现某些机器执行时间过长，其余机器处于等待的状态，造成不必要的资源浪费，同时影响执行效率。除此之外，还要尽可能减少机器间的通信代价。机器间的数据传输也大大影响算法的执行效率。因而如何对复杂网络进行分区对应用程序的执行起到关键作用。因此，

本文首先通过对图分区的研究对 Graphx 的图存储进行优化，在此基础上，完成对社区发现算法的研究。

1.2 国内外研究现状

1. 图分区

图分区作为分布式图计算中必不可少的数据预处理步骤，图分区的质量直接影响图计算的性能。均匀地将图分布到各个分区中，并使各分区之间的通信量尽可能少，这一直是图分区所追求的目标[5]。图分区经过广泛的研究，划分方法主要分为三类：基于边切割的划分、基于顶点切割的划分以及混合划分。

基于边切割的划分方法是将顶点分配到各个分区，连接两个不同分区中的顶点的边被切割。Stanton 等人[6]提出了一种线性确定性贪婪算法，该算法使用启发式切边法对图进行分割。该方法将一个顶点分配给现有分区中邻居数量最多的分区，并引入惩罚机制以保持负载平衡。Fennel[7]是一种基于边缘切割的单通道流式图分区算法，可以在线处理大规模的图数据。Liu 等人[8]基于目标优化问题，提出了两种贪婪的边缘切割算法，用于顶点复制和负载平衡。由于在图数据中各个顶点的度数是不同的，而这些方法只考虑了各分区中顶点数是否均衡，忽略了对边的考虑。因而这些算法具有负载不平衡，通信成本高的问题。

为了解决上述问题，引入了基于顶点切割的方法。基于顶点切割的划分方法是以边为单位，将边分配到各个分区中，各分区中的边互不相同，而顶点可以重复出现在多个分区中。在 PowerGraph 系统中提出了基于顶点切割的方法，包含随机边分配、协同贪婪边分配以及遗忘的贪婪边分配[9]。在 PowerLore 中提出了一个基于度的哈希算法，两个基于度的贪婪算法[10]。还有一种基于边优先级的划分方法[11]被提出，通过参考图中的度数信息，对边进行优先级排序，并用启发式获得最优分区。但这些顶点切割的策略必须存储顶点的副本，这导致了低度顶点之间的通信开销的增大[12]。

因此，将顶点切割和边切割相结合提出了混合划分的方法。混合划分则是对高度节点使用顶点切割的方法，对低度节点使用边切割的方法。使用这种分区方法的 PowerLyra 系统，通过对高度顶点仍采用顶点切割的方法，而对低度顶点使用边切割的方法，减少低度顶点之间的通信开销[13]。但对于区分高度节点和低度节点的阈值的选取比较困难。

在图计算系统中通常具有内置的图分区策略。但一个好的图分区策略是由算法、图

数据和平台决定的。这就导致内置的图分区策略不太灵活，因此出现了很多用于图分区的分区框架。D-Galois 框架旨在通过优化通信来提高图应用的执行效率[14]。XtraPuLP 提出了一种边缘切割方法来提高分区质量[15]。为了减少图划分的时间，CuSP 提出了一种快速流式图划分框架[16]。Gill 等人提出了一种基于决策树算法的分区策略选择框架[17]。该框架是通过估计应用程序的执行时间，是否离线划分，以及集群数量三个标准来选择分区策略的。但这个分区策略选择框架是通过实验得出的，并且可选择的分区策略有限，选择性不够灵活。以上这些划分框架都是基于 MPI 的，在稳定性和扩展性上不如 Graphx。因此，本文将自动选择分区策略的思想运用到 Graphx 中，通过程序分析的方式为应用程序选择适合的分区策略，这种方法不仅更加灵活，而且开销小。

2. 社区发现

2002 年，Girvan 和 Newman 运用图分割的思想首次对网络中存在的社区结构进行挖掘，并提出了 GN 算法[18]。此后的十几年间，社区发现受到了物理学、网络科学和社会科学等多个学科领域学者的广泛关注和深入研究，计算机科学领域也掀起了社区发现的研究热潮。社区发现作为社会计算的技术基础被应用于一系列数据分析任务，比如突发事件检测、垃圾邮件识别和社交链接预测等。

社区发现对复杂网络的结构研究起着至关重要的作用，研究者们将社区结构分为非重叠社区和重叠社区。非重叠社区和重叠社区的区别在于是否存在一个节点同时属于多个社区，若存在这样的节点，则为重叠社区，反之就是非重叠社区。

在对非重叠社区的研究中，已经提出了许多方法，例如基于模块化的社区发现算法。Qiao 等人提出了一种基于近似优化的快速并行社区发现算法[19]。该算法集成了两种新技术：山体模型和滑坡算法。Ling 等人[20]提出了一种基于模块优化的快速并行社区发现算法。该算法实现了基于模块化算法在 Graphx 上的并行化。Shi 等人提出了基于拟拉普拉斯中心性峰值聚类的社区发现算法[21]。使用一个新的采用基于模块化的合并策略来确定社区的最佳数量。然而，基于模块化的方法有分辨率限制的问题，不能发现小型社区，同时该方法效率较低，不适合处理大型网络。随着互联网数据规模越来越大，考虑全局信息的传统的社区发现算法已经不适用于在大规模图数据上检测社区，因而出现了大量基于局部信息的社区发现算法用于处理大规模复杂网络。

其中，标签传播算法是最受欢迎的一种社区发现算法。Yang 等人提出了一种基于图

的标签传播算法[22]。该算法将每个节点的标签用其最相似的邻居节点来更新，以此构造标签传播图。Tang 等人设计了一种基于权重和随机游走的并行标签传播算法[23]。在标签传播的过程中使用权重和相似度来更新标签，提高了社区检测的准确性和稳定性。

利用网络中的局部结构信息，在大规模图中可以快速应用社区发现算法。但是在现实世界中，重叠社区更符合现实，比如在社交网络中，一个人既可以是篮球俱乐部的成员，也可以是足球俱乐部的成员。因此，出现了许多重叠社区发现算法。Tang 等人提出了基于重叠社区的标签传播算法[24]。使用局部最大团来初始化标签以减少标签冗余。利用节点间的传播特性进行标签传播和标签更新。如果存在高度重叠的社区，就将它们合并。Kouni 等人介绍了基于节点重要性的标签传播算法来检测重叠社区[25]。为一个节点设置多个标签。使用基于节点属性的信息进行特殊的传播和筛选过程。既保持了标签传播的简单性也提高了准确性。为了解决社区发现中标签分配不稳定的问题，一种基于节点影响力的 NI-COPRA 算法被提出，用于获取更稳定的社区[26]。LPIO 算法是一种基于局部路径信息提出的重叠社区发现算法，通过选择初始种子节点，并进行扩展的方式获得社区[27]。LSDPC 算法是一种基于节点局部相似性的两阶段密度峰值聚类算法，用于挖掘重叠社区，并获得了良好的结果[28]。Ma 等人基于 Spark 提出了一种基于概率和相似性的并行标签传播算法[29]。在标签初始化过程中使用 k-shell、邻域指数和位置指数计算节点的权值。然后计算节点间的传播概率和相似性。通过自动标签选择和相似性选择标签。

1.3 主要研究内容

本文针对复杂网络上的社区发现问题展开讨论。为了能使提出的社区发现算法获得更好的效率，在图分区方面对 Graphx 的性能做出了提升，并在此基础上提出了一个用于寻找社区的社区发现算法。本文的相关工作如下：

（1）基于图分区的图存储优化。为了提高 Graphx 图处理的效率，本文在 Graphx 上提出了一种全新的图分区框架，该框架可以根据应用程序的特性选择最适合的分区策略。一个合适的分区策略，能大大提高应用程序的执行效率。为了使分区框架中可选择的分区策略更加丰富，本文提出了一种新的基于三角形的分区策略，该策略适合于需要度量图的结构特征的应用程序。另外，为了使分区策略和应用程序相结合，设计了基于哈希分区的 PageRank 算法，该算法使每次迭代都减少一次 shuffle 操作，从而提高了 PageRank

的执行效率。

（2）基于图存储优化的社区发现研究。基于优化后的 Graphx，本文提出了一种在大型复杂网络中寻找社区的社区发现算法。无论是在寻找社区的准确性上还是执行时间上都取得了不错的成果。首先，本文提出了一个新的局部密度公式以及最小距离公式来计算节点的重要性，结合密度峰值聚类算法的思想来计算节点的重要性。其次，使用基于核心和边缘节点的标签传播完成各节点的标签分配。主要是使用改进的最短路径算法和改进的基于信任的距离度量为边缘节点寻找最近的核心节点，并用核心节点的标签更新边缘节点的标签。再用已有标签的节点向未有标签的节点传播标签。最后，为了提高标签分配的准确性，对节点的标签和其邻居节点标签不一致的节点重新分配标签。

1.4　本文结构

本文主要结构安排如下：

第一章：绪论。首先介绍图分区和社区发现的研究背景及意义，然后对图分区和社区发现的国内外研究现状加以描述，最后对本文的主要研究内容进行阐述，并给出了文章的组织架构。

第二章：相关知识与技术介绍。主要介绍大数据平台的相关技术和社区发现的知识，主要包含对 Hadoop 平台的介绍、Spark 平台的介绍、Graphx 组件的描述、密度峰值聚类以及标签传播算法的介绍。

第三章：论述基于图分区的图存储优化。首先对图分区框架进行描述，然后介绍基于三角形的图分区策略，最后设计了基于哈希分区的 PageRank 算法。

第四章：论述基于图存储优化的社区发现算法。首先利用密度峰值聚类算法的思想计算节点的重要性。然后利用改进的最短路径算法和改进的基于信任的距离度量为边缘节点分配标签。接着使用基于核心和边缘节点的标签传播算法传播标签。最后为了标签分配的准确性，进行标签重分配。

第五章：总结与展望。对本文的研究工作和成果进行总结，并对今后的研究方向进行展望。

第二章　相关知识与技术介绍

在第一章中介绍了本文的研究背景与意义、国内外的研究现状、对图分区和社区发现进行了充分了解。在本章中，就图分区和社区发现所需要的技术支持和相关知识展开介绍。

2.1　Hadoop 和 Spark 介绍

2.1.1　Hadoop 介绍

Hadoop 是 Apache Lucene 创始人道格`卡丁(Doug Cutting)创建的一个用于数据存储和分析的分布式系统。

Hadoop 的核心组件有 HDFS、YRAN 和 MapReduce。本文主要使用了 HDFS 和 YARN 两个组件，并使用 Spark 框架代替 MapReduce 计算框架。下面对 HDFS 和 YARN 进行介绍。

HDFS 是 Hadoop 中自带的分布式文件系统，全称为 Hadoop Distributed File System。它可以将庞大的数据存储到若干台独立的计算机上，很好的解决了单台机器无法提供足够的存储空间的问题。HDFS 主要由 NameNode、Secondary NameNode 和 DataNode 组成。NameNode 在 HDFS 系统中只有一个，可以看成是集群的管理者，管理整个文件系统的命名空间，用于存储元数据及处理客户端发送的请求。Secondary NameNode 是辅助名称节点，是对 NameNode 的一个补充，它会周期的执行对 HDFS 元数据的检查点任务。当 NameNode 元数据损坏的情况下，Secondary NameNode 也可用作数据恢复。DataNode 是真正存储数据的地方，有很多个。文件会以数据块的形式存储在 DataNode 中。HDFS 会将文件按照数据块的大小进行分割，以数据块为单位存储到不同或相同的 DataNode 节点上，并备份三份副本，这样其中一个 DataNode 挂掉了，也可以从其他的 DataNode 中获取数据，不会影响到程序的执行。

实验中所使用的所有数据都被存储在 HDFS 中。数据被分散存储在多台机器中。并可以结合 Spark 平台使用，方便 Spark 程序从 HDFS 中读取数据并处理。

YARN 是 Hadoop 的集群资源管理系统，也是 Hadoop 的重要组成部分之一。它的全名为 Yet Another Resource Negotiator，主要工作是资源管理和作业调度。

YARN 主要由 ResourceManager、NodeManager、ApplicationMaster 和 Container 几个组件组成。其中，ResourceManager 在集群中主要负责资源的调度、管理和分配等等。而 NodeManager 则主要报告节点的状态情况。ApplicationMaster 相当于这个 Application 的监护人和管理者，负责监控、管理这个 Application 的所有 Attempt 在 cluster 中各个节点上的具体运行，同时负责向 ResourceManager 申请资源、返还资源等。最后一个组件是 Container，它是 YARN 中负责资源分配工作的，这里的资源包含了内存、CPU 等。在 YARN 中以 Container 为单位进行资源的分配。

Hadoop 集群中运行的各种应用程序所需要的系统资源都需要 YARN 进行分配。YARN 将系统资源调度到不同集群节点上，供执行的任务使用。YARN 的基本思想是将资源管理和作业调度/监视的功能分解为单独的守护进程 (daemon)，其拥有一个全局 ResourceManager 和每个应用程序的 ApplicationMaster。应用程序可以是单个作业，也可以是作业的 DAG。

ResourceManager 和 NodeManager 构成了数据计算框架。ResourceManager 是在系统中的所有应用程序之间仲裁资源的最终权限。NodeManager 是每台机器框架代理，负责 Containers，监视其资源使用情况（CPU，内存，磁盘，网络）并将其报告给 ResourceManager。

每个应用程序 ApplicationMaster 实际上是一个框架特定的库，其任务是协调来自 ResourceManager 的资源，并与 NodeManager 一起执行和监视任务。

2.1.2 Spark 介绍

Spark 是一个基于内存计算的可扩展集群计算引擎，诞生于加州大学伯克利分校的 AMP 实验室，采用 Scala 编程语言编写。与 Hadoop 的 MapReduce 相比，Spark 基于内存的运算速度是 MapReduce 的 100 倍。因此本文使用 Spark 作为实验平台。Spark 直接从内存中访问数据，减少了从磁盘读写数据的时间，大大提高了计算速度。Spark 能基于内存运算得益于 Spark 自己的数据抽象，弹性分布式数据集。Spark 集批处理、实施流处理、交互式查询和图计算于一体，避免了多种运算场景下需要部署不同集群带来的资源浪费。将从以下两个方面介绍 Spark。

Spark 的整体架构主要由 Driver、Cluster Manager 和 Worker 组成。Driver 负责和 Cluster Manager 交互，向它申请计算资源等。Cluster Manager 负责集群的资源管理和调

度，支持 Standalone、Mesos 和 YARN 模式。在实验中所使用的是 YARN 模式。Worker 则是集群中执行计算任务的节点。Worker 可以有很多个，每个 Worker 又由 Executor 组成。Executor 是 Worker 上的一个进程，负责运行任务和将数据存到内存或磁盘上。Task 是被送到某个 Executor 上的计算单元。

弹性分布式数据集(RDD)是 Spark 中最基本的数据抽象，具备像 MapReduce 等数据流模型的容错性，允许开发人员在大型集群上执行基于内存的计算。将数据保存在内存中能够极大地提高性能。为了有效地实现容错，RDD 提供了一种高度受限的共享内存。RDD 具有只读性，可以分区，并且只能通过其他 RDD 上的批量操作来创建。

在 RDD 上的操作可以分为两类：Transformation 算子和 Action 算子。Transformation 算子表示从现有的数据集创建一个新的数据集。Action 算子表示在数据集上进行计算后，返回一个值给 Driver 程序。Transformation 的所有算子都是惰性的，它们不会直接计算结果。需要遇到 Action 算子时 Transformation 算子才会开始执行。

2.2　Graphx 介绍

Graphx 是 Spark 的一个用于图计算的组件，具有很好的灵活性、易用性和容错性。Graphx 将图计算和数据计算集成到一个系统中，数据不仅可以被当作图操作，也可以被当作表进行操作。下面主要介绍 Graphx 的分布式存储、Pregel 和图分区策略。

2.2.1　图的分布式存储

由于 Graphx 是基于 Spark 平台运行的，因而 Graphx 也利用了 Spark 中的 RDD 来存储图数据。图数据被保存在边 RDD 和顶点 RDD 中，分区保存在集群的各个机器中。顶点 RDD 和边 RDD 分区的方式不同，顶点 RDD 用顶点 Id 进行哈希分区。而边 RDD 则可使用后面介绍的图分区策略来分区。顶点 RDD 和边 RDD 的联系通过路由表来实现。通过路由表能很快寻找每条边所连接的顶点的信息。

2.2.2　Pregel 介绍

目前的图计算框架基本上都遵循批量同步并行计算模式。计算过程包括一系列全局超步，每个超步就是一次迭代，一次迭代主要包括三个步骤：局部计算、通信以及栅栏同步。Pregel 就是基于批量同步并行计算模型实现的。Graphx 对 Pregel 进行了实现。Pregel 的计算过程同样是由一系列超步组成。在每个超步内部，每个顶点的计算都是并

行的。在超步结束之前，每个顶点都为其他顶点生成消息，这些消息会传递到下一个超步中。而且每个超步全部执行完毕后，才会进入下一个超步。这是因为同步障碍机制造成的。

每个超步都经历三个阶段：消息合并、顶点处理程序和发送消息。消息合并是指每个顶点将接收到的上一个超步发送来的消息进行合并，但不会直接对顶点进行更新。顶点处理程序是指用合并的消息和自身已有的消息进行计算，并更新自己的属性。发送消息则是顶点将自己的信息发送给其邻居节点。

2.2.3 图分区策略

在 Graphx 中包含了四种使用顶点切割的图分区策略，分别为 Edgepartition1D(E1D)、Edgepartition2D(E2D)、RandomVertexCut(RVC)以及 CanonicalRandomVertexCut(CRVC)。Graphx 在第一次读取图数据并构建图时，图是随机分区的。对图进行合适的分区可以通过减少通信和存储开销提高运行在图数据上的算法的执行效率。

E1D 分区策略根据源点的 Id 来分配边。取源点的 Id 值，对分区数取哈希值，就是该边所分配的分区 Id。这样可以将一个节点的所有出边都分配到同一个分区中。

E2D 分区策略同时使用了源点 Id 和目的顶点 Id 进行分区。并考虑了分区数是否能完全开方的问题。就分区数是否能完全开方，分别采取了两种措施。

RVC 分区策略是使用源点和目的顶点的 Id 一起对分区数做哈希，这样两个顶点之间相同方向的所有边会被分配到同一个分区中。

CRVC 分区策略和 RVC 很相似。它们的区别在于 CRVC 会考虑源点和目的顶点的 Id 大小。将 Id 值小的顶点放在前面，Id 值大的顶点放在后面。这样两个顶点间的所有边（不考虑方向）都会被分配到同一个分区中。

介绍的前面两个分区策略为了提高均衡性，在处理顶点时，都将顶点 Id 乘以一个很大的素数后再进行处理。基于 E1D 的启发，一个基于目的顶点进行分区的分区策略应运而生，它被称为 PartitioinWithDst(DST)分区策略和 E1D 采用了同样的原理。它被应用到了后面的对比实验中。

2.3 社区发现知识

2.3.1　密度峰值聚类

密度峰值聚类的聚类中心基于两个假设：聚类中心的局部密度大于其邻居节点的局部密度；聚类中心与其他局部密度较大的节点之间的距离相对较大。它使用两个变量来确定集群中心，即局部密度ρ和最小距离δ。有两种方法来计算局部密度。第一种方法是使用公式(2.1)所示的截断核函数。

$$\rho_i = \sum_j \chi(dist_{ij} - d_c) \tag{2.1}$$

$$\chi(x) = \begin{cases} 1, & x \le 0 \\ 0, & otherwise \end{cases} \tag{2.2}$$

其中，ρ_i是节点i的局部密度，$dist_{ij}$是i和j之间的距离，d_c是截断距离。公式(2.2)中的x表示$dist_{ij} - d_c$。

第二种方法是使用高斯核函数，如公式(2.3)所示。

$$\rho_i = \sum_j exp\left(-\left(\frac{dist_{ij}}{d_c}\right)^2\right) \tag{2.3}$$

其中$dist_{ij}$和d_c表示的含义与截止核函数中的含义一致。

密度峰值聚类中最小距离δ_i的计算公式见公式(2.4)。

$$\delta_i = \begin{cases} \min\limits_{j:\rho_i<\rho_j} (dist_{ij}), & if \exists j\ s.t.\ \rho_i < \rho_j \\ \max\limits_{j} (dist_{ij}), & otherwise \end{cases} \tag{2.4}$$

δ_i是节点i与其他更高密度节点之间的最小距离。

根据密度峰值聚类的两个假设，聚类中心应该选择局部密度大且相距较远的节点。该算法将所有节点的局部密度和最小距离绘制成散点图，在右上方选择与其他节点距离较远的节点作为聚类中心。然后，每个剩余的数据节点都被分配到与密度较大、距离最近的节点相同的聚类中。

2.3.2　标签传播算法

作为检测社区的最流行的算法，标签传播算法已经被许多研究人员研究。由于它的简单性和接近线性的时间复杂度，它可以在大规模网络中快速进行社区检测。标签传播算法根据网络拓扑信息传播节点信息，将网络中的每个节点分配到与它的大多数邻居相同的社区。标签传播算法的主要步骤如下：

步骤 1：用节点 ID 初始化每个节点的标签；

步骤 2：轮流刷新所有节点的标签，直到所有节点的标签不再变化，否则继续迭代。

对于每一轮的刷新，节点标签的刷新都具有一定的规则。对于某一个节点，检查其所有邻居节点的标签，进行统计，并将数量最多的标签分配给当前节点。当最多数量的标签不唯一时，就随机选择一个。

2.4 本章小结

本章主要介绍本文所用到的相关技术和知识，主要阐述了 Hadoop 平台的 HDFS 分布式文件系统和 YARN 分布式资源调度系统、Spark 框架、Graphx 分布式图计算框架、密度峰值聚类和标签传播算法。为后面的研究提供了技术支撑和知识保障。

第三章　基于图分区的图存储优化

在本章中主要介绍利用图分区策略优化 Graphx 的图存储。主要提出了基于程序分析的图分区框架、一个基于三角形的图分区策略(Triangle-based graph partitioning strategy, TriPS)以及基于哈希分区的 PageRank 算法(PageRank algorithm based on hash partitioning, PRHP)。

随着社交网络、传感器网络、交通网络和推荐系统等各种领域的图结构数据的大规模增长[30-33]，如何处理和分析如此大量的数据对研究和应用产生了巨大的挑战。为了能够并行处理图，不可避免地要将输入的图分割成更小的部分。因此，一个适当的图分区策略对于大型应用程序来说起到了至关重要的作用，可以大大提高算法的执行效率。然而，现有的大多数分布式图分区方法几乎没有考虑到划分策略与图算法特性之间的关系。目前的图分区策略都是基于图的结构，运行不同的图算法，其性能也各不相同。本文结合图算法的特点，提出了一种分布式图分区框架，该框架可以在程序分析的基础上决定哪种划分策略更好。Graphx 按照这种分区策略存储图数据，使得应用程序的执行效率得到提高。接着，设计了一种 TriPS，该策略受益于三角计数等算法。此外，为了减少 PageRank 算法中的 shuffle 操作的次数，设计出了 PRHP 算法。实验结果表明，无论在真实图还是合成图上，本文提出的自适应图分区框架，在三角计数和 PageRank 方面都获得了很好的表现。

3.1　基于程序分析的图分区框架

本章介绍了一个基于程序分析的图分区框架。针对三角计数等算法，设计了 TriPS 这种全新的分区策略。为了减少 PageRank 算法每次迭代中的 shuffle 次数，本文基于哈希分区重写了 PageRank 算法。

下面是在 Graphx 上提出的一个全新的图分区框架，可以根据应用的属性选择最合适的分区策略。这个图分区自适应框架如图 3-1 所示。源代码由解析器模块标记和解释，并由分析器使用正则表达式技术进行语义分析。然后，分析器的结果将发送到决策者，以使用分区器引擎选择适当的分区策略。图 3-1 中每个模块的详细描述如下：

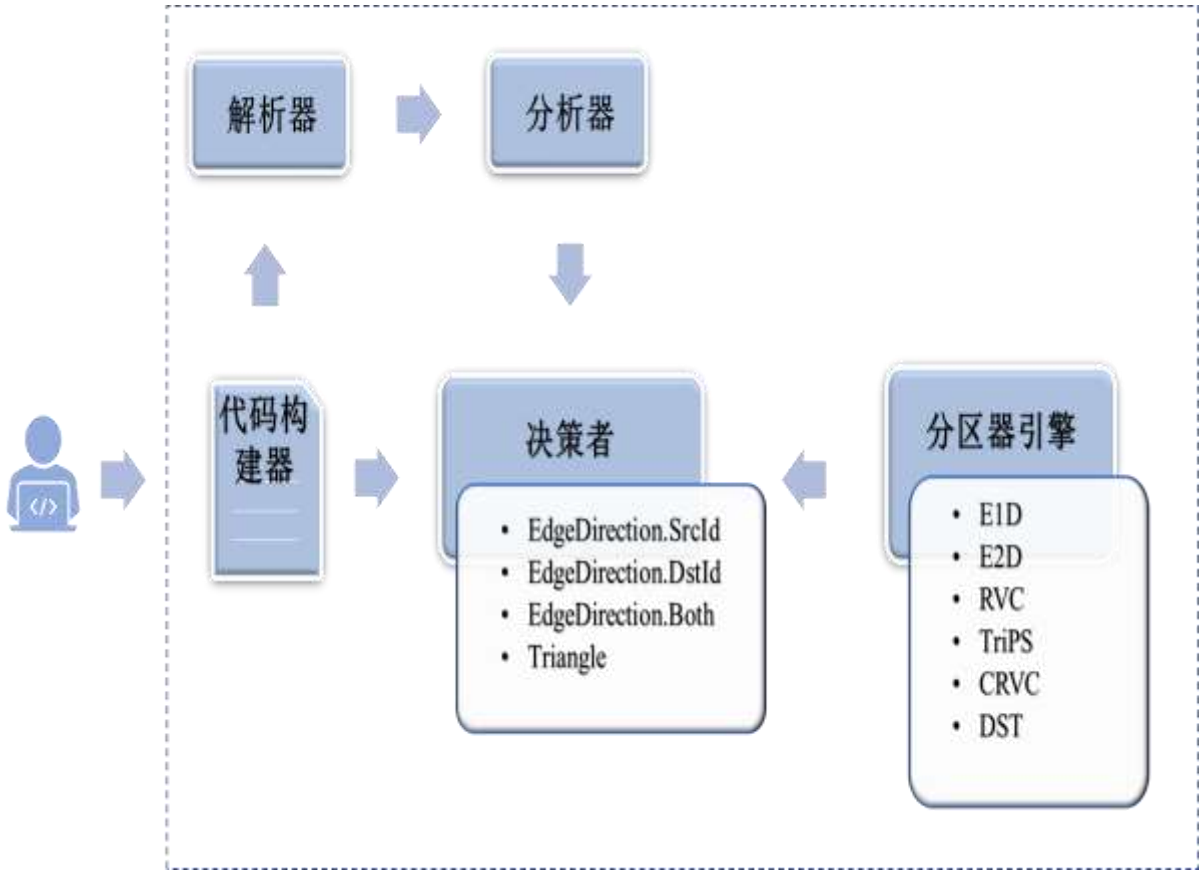

图 3-1　图分区框架

代码构建器：该部分作为系统的入口，需要输入一个算法的形式化源代码和图数据。

解析器：此模块分析来自代码构建器的代码并利用 Scala 解析工具“Scala.util.parsing.ast”提取 Scala 字节码中的抽象语法树。

分析器：解析器的输出被用作分析器的输入。在该模块中，对抽象语法树进行先序遍历，从诸如 EdgeDirection 这样的变量中确定数据流方向和控制信息。比如：EdgeDirection.SrcId、EdgeDirection.DstId、EdgeDirection.Both，还有 Triangle。

决策者：根据分析器的分析结果，选择与数据流方向和控制特征相匹配的分区策略。定义公式(3.1)，以选择最合适的分区器。

$$match(partitioner, appFlow)，\ partitioner \in PartitionerSet \tag{3.1}$$

其中$appFlow$作为应用程序的流模式。如果代码中有用户自定义的分区策略，该框架直接选择用户自定义的分区策略。否则，该框架根据以下规则动态选择分区策略：当数据流方向是从源顶点到目标顶点时，该框架将根据目标顶点选择分区策略。当数据流方向从目标顶点到源顶点时，选择基于源顶点的分区策略。当双向数据流发生在不同的

边缘上时，通过比较源顶点和目标顶点的度数，选择那个度数较大的顶点进行划分的分区策略。当在同一边缘上发生双向数据流时，从输入数据中随机抽取一小部分数据，以选择发生数据传输最小的分区策略。当数据流发生在三个顶点之间时，TriPS 是更好的选择。

分区器引擎：分区器引擎是一个分区器的集合，其中包含各种各样的基于顶点切割的分区策略，如 E1D、E2D、RVC、TriPS、CRVC 和 DST。用户还可以在分区器引擎中设计自定义分区策略。

本文提出的分区框架的开销非常小。决策过程非常快，因为只有特定的数据流标识符和应用程序特征被考虑其中，比如 EdgeDirection.SrcId、EdgeDirection.DstId、EdgeDirection.Both 和 Triangle。

在这里，使用 PageRank 算法作为一个例子来演示本文提出的分区框架的过程。首先将 PageRank 源代码作为代码构建器的输入。然后，解析器从 PageRank 源代码中提取抽象语法树。遍历抽象语法树之后，分析器找到表示数据流方向的变量 EdgeDirection.DstId。在决策者中，选择 DST 策略作为最适合 PageRank 的分区策略。通过程序分析，观察到如果考虑基于哈希的分区策略，重新设计 PageRank 算法，可以通过减少 shuffle 次数来改善 PageRank 应用程序的执行效率。

3.2　基于三角形的分区策略

由于用于分布式计算的图数据量很大，很难通过图的总体视图对图进行划分，因此一般的划分策略是通过边的方向等部分信息对边进行分配。为了提高每个分区中边的连通性，通过分析三角计数算法，设计了三个顶点连通的三角形分区策略。三角计数算法是计算聚类系数的基础，适用于衡量图的结构特征。例如，三角计数算法经常被用来寻找图的局部聚类系数。因此，用一个有三条边的三角形作为单元来划分图，这样可以有效地将几个相连的边放在同一个分区中。例如，如图 3-2 所示，三角形(1,2,4)包含边<2,1>、<2,4>和<4,1>。它们位于同一个分区中。三角形(3,5,6)由边<3,6>、<6,5>和<5,3>组成，按照三角形分区规则也被放在一个分区中。如果一条边包含在多个三角形中，就从这些三角形集合中选择哈希值最小的一个三角形进行分配。然而，并不是图中的所有边都能组成三角形。对于那些不在三角形中的边，如图 3-2 中的<6,7>，将它们分配到当前边数最少的分区中，以达到分区平衡的目的。

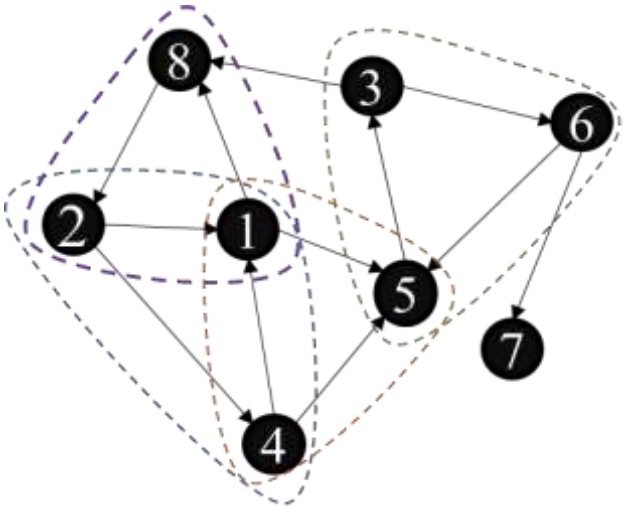

图 3-2　样例图

尽管如此，由于参差的图拓扑结构，它仍然可能会遇到不平衡的问题。在这里提出了一个促使分区平衡的算法，具体实现如算法 3.1。使用一个名为$table$的本地表来记录在每个分区中分配的边的数量。本地表包含两个列，键列PID表示分区号，值列$edgeNum$表示每个分区中的边数。注意，$minPID$是表中边数最少的分区，$maxPID$是表中边数最多的分区。$minPID/maxPID$代表当前所有分区之间的平衡性。这个数字越大，平衡就越好。从实验中发现，当这个阈值为 0.95 时，性能是最佳的。

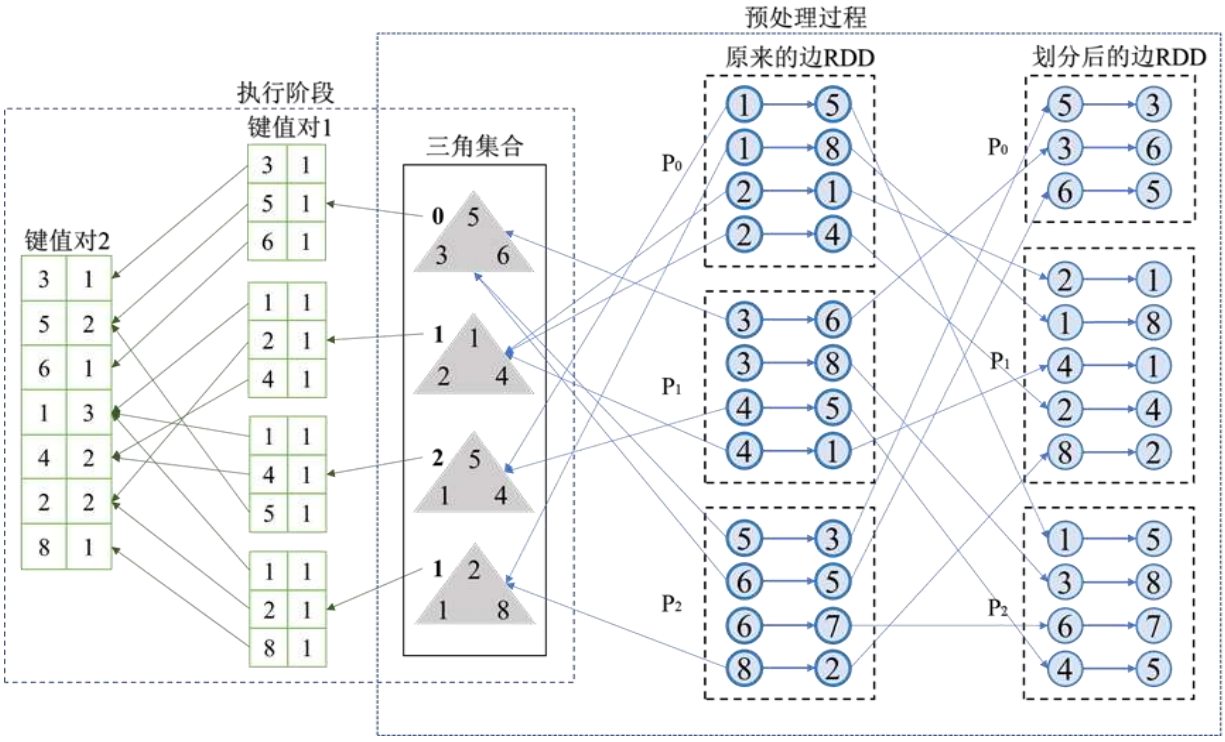

图 3-3　三角计数算法的预处理和执行过程

当遍历一个分区中的边时，如果这条边至少包含在一个三角形中并且$minPID/maxPID >= 0.95$，那么选择具有最小哈希值的分区来分配这条边。否则，如果$minPID/maxPID < 0.95$，将该边分配到边数最少的分区中。注意，$triangles = \{t_1, t_2, \ldots\ldots, t_k\}$表示边$e$对应的三角形集合，其中$t_i$表示边$e$存在在三角形$t_i$中。$t_i.hashcode$表示三角形$t_i$的$hashcode$值。如果分区不均衡，则将该边分配给边数最少的分区。否则，将边e分配给分区$minHashcode\%numParts$，其中$minHashcode$表示最小的$hashcode$。其中，在算法中使用的哈希函数是 Graphx 内置的哈希函数，它也在 Graphx 中的 HashPartitioner 中使用。如果两个三角形有相同的最小哈希码，选择顶点Id和最小的三角形。

算法 3.1　分配边的算法$assignEdge$

输入：三角形集合$triangles$；记录每个分区中边数的表$table$
输出：分区号$parID$

```
1)  if !triangles.isEmpty then
2)    if table.minPID/table.maxPID >= 0.95 then
3)      for each t in triangles do
4)        if t.hashcode < minHashcode then
5)          minHashcode = t.hashcode
6)          parID = t.getPID
7)        end if
8)      end for
9)    else
10)     parID = table.getMinPid
11)   end if
12) else
13)   parID = table.getMinPid
14) end if
15) table(parID) + +
```

在使用三角形分区策略之前，第一个任务是计算每条边所在的三角形集合。该三角形是由源顶点和目标顶点的相邻顶点的交集得到的。边e被包含的三角形的个数正好是该边的两端点的邻居节点的交集的大小。例如，图 3-2 中边<4,1>，源顶点 4 的邻居顶点集为$N_4 = \{1,2,5\}$，目的顶点 1 的邻居顶点集为$N_1 = \{2,4,5,8\}$。由于$N_4 \cap N_1 = \{2,5\}$，因此边<4,1>被包含在(1,2,4)和(5,1,4)两个三角形中。整个三角形分区策略的具体细节见算法 3.2。此外，当一个顶点被随机或意外删除时，所有缓存的带有该顶点的三角形也会被删除，这表明本文提出的划分算法是稳健的。图 3-3 演示了对图 3-2 中的图进行划分并进行三角计数的过程。划分后的边 RDD 包含三个分区。这里，以原来的边 RDD 的 P_0 分区为例。当边<1,5>被加载时，确定这条边位于三角形(1,4,5)中，该三角形由哈希函数分配给划分后的边 RDD 中的分区 P_2。用同样的规则将<1,8>和<2,4>分配给分区 P_1。对于边<2,1>，将它分配到 P_1，因为三角形(2,1,8)的$hashcode$小于三角形(1,2,4)。由于<6,7>不包含在任何三角形中，因此它被分配给划分后的边 RDD 中边数最小的分区 P_2 中。

算法 3.2　三角形分区策略

输入：图$G(V,E)$；记录每个分区中边数的表$table$；分区数$numParts$

输出：分区后的图$newGraph$

```
1)  neighborIDs = collectNeighborIDs()
2)  newGraph = G.outerJoinVertices(neighborIDs).mapTriplets(
3)    e => {set ← e.srcAttr.intersect(e.dstAttr)
4)      set.foreach(v => {
5)        if v! = e.srcId and v! = e.dstId then
6)          triangles.add(triangle(v, e.srcId, e.dstId))
7)        end if
8)      })
9)      pid = assignEdge(triangles, table)
10)     (pid, triangles)
11)   }
12) )
13) return newGraph
```

分区过程对三角计数算法做了很大的改进。使用三角形分区策略后，这些三角形集合被存储为边的属性。在实现三角计数算法时，每个三角形都被转换为一个键值对，然后被聚合到一起。然后通过函数$reduceByKey(count)$将其与相同的vID聚合起来，这与wordcount 算法类似。

3.3　基于哈希分区的 PageRank 算法

PageRank 算法被广泛应用于各个方面。它最初是用来在谷歌搜索引擎中对网页进行排名，现在它可以应用在很多方面，比如社会影响分析，文本摘要等。在本小节中，提出了 PRHP 算法，该算法将每次迭代的 shuffle 次数从两次减少到一次，从而提高了 PageRank 的执行效率。

算法 3.3 带有两次 shuffle 的 PageRank 算法

输入：图$G(V,E)$；最大迭代次数$maxIterations$

输出：PageRank 的分数$ranks$

```
// 初始化 PageRank 分数
ranks = graph.vertices.mapValues(v => 1.0)
// 生成带出度的links
links = graph.outerJoinVertices(graph.outDegrees){
  (vid, attr, outDegree) => outDegree.getOrElse(0)
  }.mapTriplets(edge => edge.srcAttr)
  .edges.setKey(edge => edge.srcId).cache()
// 通过迭代计算 PageRank 分数
for each i in 0 until maxIterations do
  ranks = links.join(ranks).mapValues{
    ((edge, outDegree), rank) => (edge.dstId, rank/outDegree)
    }.reduceByKey(_ + _)
    .mapValues(rank => 0.15 + 0.85 * rank)
end for
return ranks
```

在原来的 PageRank 算法中，如算法 3.3 所示，每次迭代都需要两次 shuffle。迭代过程由图 3-4 和图 3-2 来展示。在图 3-4 中,“等级 RDD”表示由哈希分区器得到的所有顶点的 PageRank 分数的 RDD，在迭代之前将每个顶点初始化为 1。“链接 RDD”是使用出度作为属性的边集，具有与“边 RDD”相同的分区器。第一次 shuffle 发生在“链接 RDD”和“等级 RDD”之间的连接操作。由于同一源顶点的边不存储在同一分区，所以“等级 RDD”必须分配给位于不同分区的相应边。第二次 shuffle 用于聚合具有相同目标顶点的等级，通过 reduceByKey 操作计算新的 PageRank 分数。

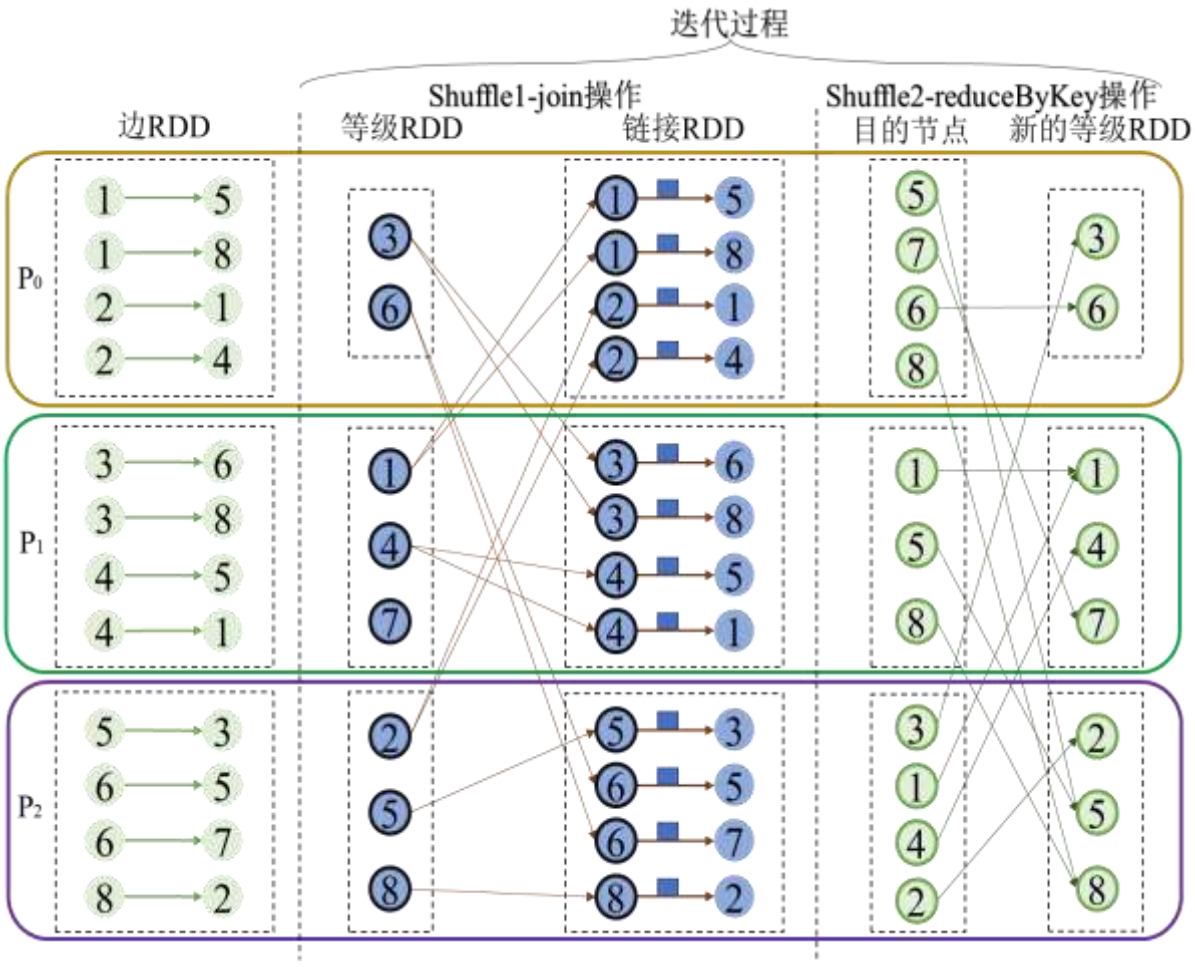

图 3-4　原来的 PageRank 算法的迭代过程

在这里，以分区 P_0 为例描述具体过程。在第一次迭代时，有两个出边的顶点 1 将现有分数的一半转移到顶点 5，另一半转移到顶点 8。同样，顶点 2 在第一个 shuffle 阶段将其现有分数的一半转移到顶点 1，将另一半转移到顶点 4。在 P_2 分区中，虽然顶点 7 没有出边不传输数据，但有必要注意的是，顶点 7 仍然需要从源顶点 6 接收消息并更新

其值。在图分区阶段，将顶点 6 和顶点 7 所形成的边根据指定的分区算法分配给相应的分区。在 reduceByKey 阶段，将具有相同目标顶点的所有分数相加，以计算每个顶点的新分数。例如，从分区 P_0 收集顶点 2 的 0.5 分，从分区 P_1 收集顶点 4 的 0.5 分来计算节点 1 的新的分数，所以顶点 1 收集 1.0 分。顶点 7 将只从顶点 6 接收 0.5，因此顶点 7 的值从 1 更新为 0.5。聚合后，“新的等级 RDD”被用于下一个迭代。

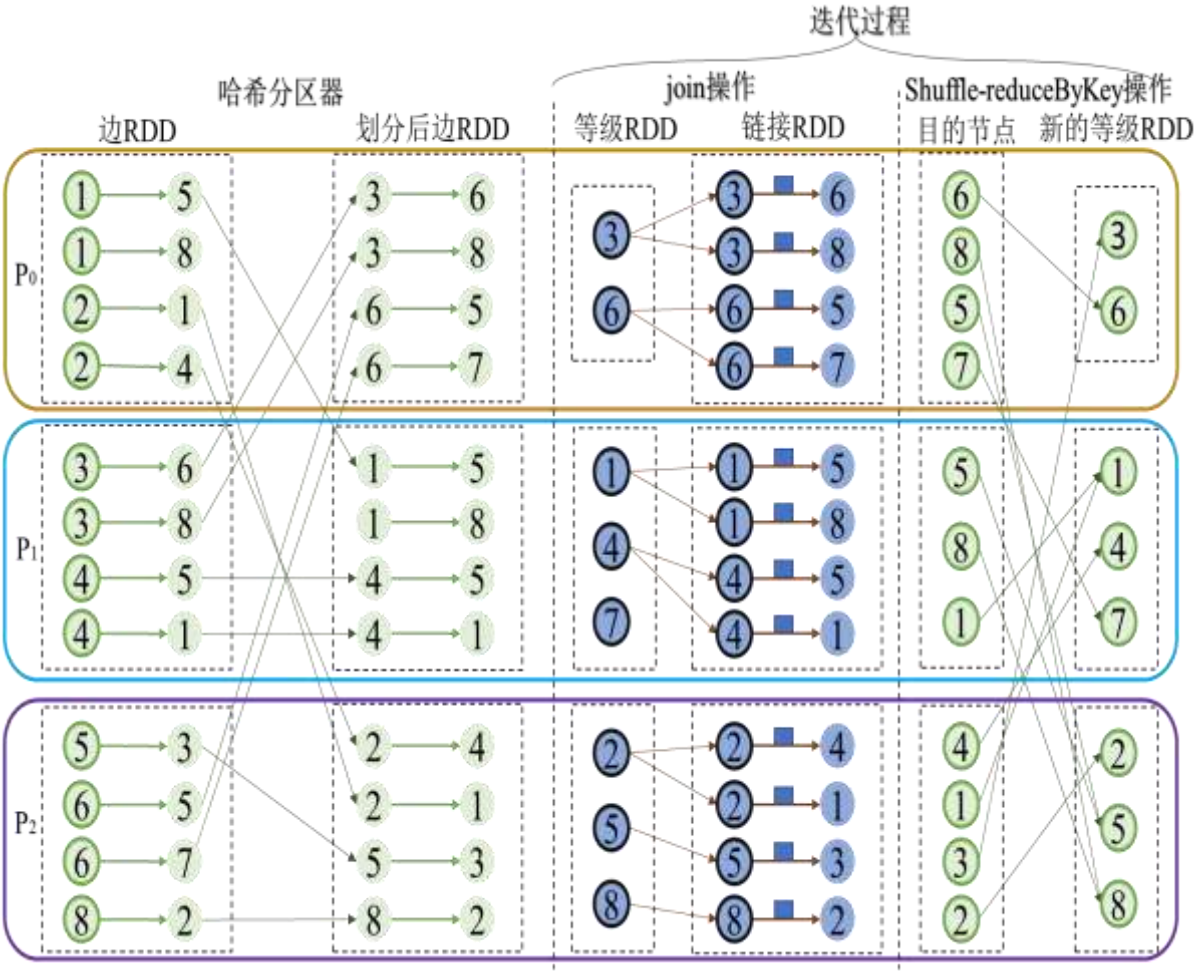

图 3-5　PRHP 算法的迭代过程

在改进的 PageRank 算法 3.4 中，由于$edgesRdd$是由源顶点的哈希值划分的，所以具有相同源顶点的边存储在相同的分区中。因此，如果$ranks$的分区器与$edgesRdd$的分区器相同，它允许 Spark 实现没有 shuffle 的连接。这样，在每次迭代中就只剩下 reduceByKey 这一个不可或缺的 shuffle 操作了。与图 3-4 中的过程相比，图 3-5 清楚地显示了本文设计的 PageRank 只有一个 shuffle 的迭代过程。从图 3-5 中注意到“等级 RDD”和“链接 RDD”在同一个分区中执行，不进行数据通信。因此，PRHP 算法减少

了“迭代”阶段机器之间的通信。尽管哈希分区器的使用增加了分区阶段机器之间的数据传输，但 PageRank 算法需要多次迭代，每次迭代减少一次 shuffle 都可以大大提高执行效率。

算法 3.4 带有一次 shuffle 的 PageRank 算法

输入：图$G(V,E)$；最大迭代次数$maxIterations$

输出：PageRank 的分数$ranks$

```
// 初始化 PageRank 分数
ranks = graph.vertices.mapValues(v => 1.0)
// 按源点划分的哈希分区器
edgesRdd = graph.edges.setKey(e => e.srcId)
  .partitionBy(new HashPartitioner(numParts))
// 计算每个节点的出度
outDegrees = edgesRDD.mapValues(e => 1)
  .reduceByKey(_ + _)
links = edgesRDD.join(outDegrees).cache()
// 通过迭代计算 PageRank 分数
for each i in 0 until maxIterations do
  ranks = links.joinWithTheSamePartition(ranks).mapValues{
    ((edge,outDegree),rank) => (edge.dstId,rank/outDegree)
    }.reduceByKey(_ + _)
    .mapValues(rank => 0.15 + 0.85 * rank)
end for
return ranks
```

3.4 实验分析

本文所作的实验是在 Graphx 上实现的，运行在由六台机器组成的 Spark 集群上。所有数据集都存储在 HDFS 中。每台机器的配置为：Intel(R) Core i9-10900K、3.70GHz、20 核 CPU、64GB 内存和 1000Mb/s 带宽。Spark 集群是基于 CentOS Linux Server 8 系

统、JDK 1.8 版本、Apache Hadoop 3.2 和 Apache Spark 3.0 搭建的。

由节点和边组成的图数据普遍存在于社交网络[34,35]、网络图[36]、交通系统[37]、生物网络[38,39]、书目引文网络[40]等网络中。本章使用的真实网络分为六种，分别为引文网络、道路网络、社交网络、超链接网络、社区网络和时态网络，一共九个数据集。除了从这些复杂网络中选择的几个数据集[41-44]外，还生成合成图来测试性能。使用 Graphx 框架中的 GraphGenerators 接口生成 logNormalGraph、rmatGraph 和 RandomGraph 三种合成图。在本章的实验中作为基准的数据集的详细信息如表 3-1 所示。

表 3-1　数据集统计分析

类型	名称	顶点数	边数	最大度数	平均度数
引文网络	cit-Patents[41]	3774768	16518948	793	8
道路网络	USA-road-d.USA.gr[42]	23947347	58333344	18	4
道路网络	CTR.gr[42]	14081816	34292496	18	4
社交网络	Soc-Pokec[43]	1632803	30622564	20518	37
社交网络	soc-LiveJournal[41]	4469328	68993773	22889	28
超链接网络	wikipedia[44]	8098346	437217424	1052390	64
超链接网络	trackers[44]	2652175	140613762	11571953	10
社区网络	ULiveJournal[41]	3997962	34681189	14815	17
时态网络	sx-stackoverflow[41]	2601977	63497050	194806	48
合成网络	logNormalGraph	100000	12544733	12426	250
合成网络	RandomGraph	100000	10000000	246	200
合成网络	rmatGraph	8192	1000000	537	30

本文在 3.4.1 部分评估了使用 TriPS、CRVC、RVC、E1D、E2D 和 DST 分区策略的三角计数算法的预处理时间和执行时间。还评估了图中的三角形数量对执行效率的影响。在 3.4.2 部分，对比了 PRHP 算法和需要两次 shuffle 的传统 PageRank 算法的执行时间以及 PRHP 算法的可扩展性。在 3.4.3 部分还进行了深入评估，将使用除 Graphx 内置分区策略以外的分区策略的三角计数算法和 PageRank 算法和本文提出的算法进行比较。除此之外，还进行了框架间的比较。

表 3-1　数据集统计分析（续）

类型	名称	三角形数	密度	聚类系数
引文网络	cit-Patents[41]	7515026	2.32E−6	0.2023
道路网络	USA-road-d.USA.gr[42]	438804	2.03E−7	0.025
道路网络	CTR.gr[42]	228918	5.83E−8	0.0222
社交网络	Soc-Pokec[43]	32557458	2.30E−5	0.0876
社交网络	soc-LiveJournal[41]	-	6.91E−6	0.1703
超链接网络	wikipedia[44]	-	1.33E−5	0.0143
超链接网络	trackers[44]	-	4.0E−5	0.0013
社区网络	ULiveJournal[41]	177820130	4.34E−6	0.8384
时态网络	sx-stackoverflow[41]	-	1.86E−5	0.01
合成网络	logNormalGraph	10412502	0.0025	0.0397
合成网络	RandomGraph	1319141	0.002	0.0079
合成网络	rmatGraph	107190	0.0298	0.0227

3.4.1　三角计数评估结果

1.　预处理时间和执行时间

在这里，使用本章提出的 TriPS 与其他分区策略（CRVC、RVC、E1D、E2D 和 DST）进行比较。这些分区策略的介绍在 2.2.3 部分，前四种分区策略是 Graphx 中内置的分区策略，最后一种 DST 分区策略和 E1D 是相似的，只是依据分区的节点不同。图 3-6 和图 3-7 分别显示了 TriPS 和其他分区策略对真实图和合成图的预处理时间。从图 3-6 和图 3-7 中可以发现，TriPS 的预处理时间比其他五种分区策略的预处理时间都要长。这是因为 TriPS 相对比较复杂，首先要计算图中的所有三角形，而不是像其他五种分区策略那样仅通过哈希源顶点或目标顶点来进行分区。

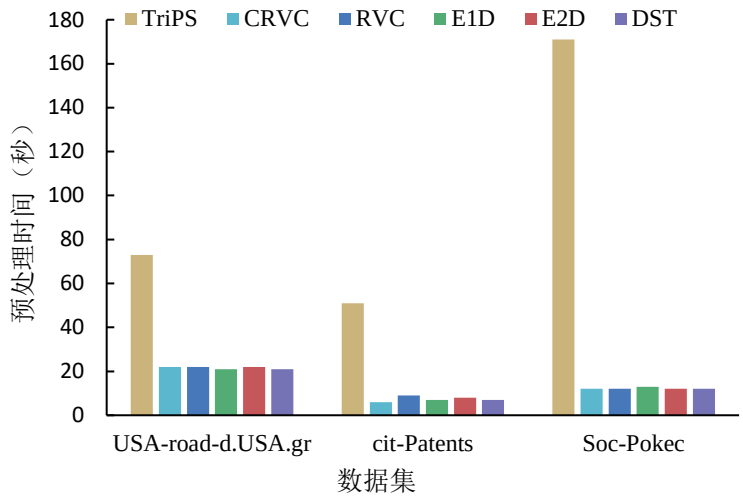

图 3-6　不同分区策略在真实图上的预处理时间

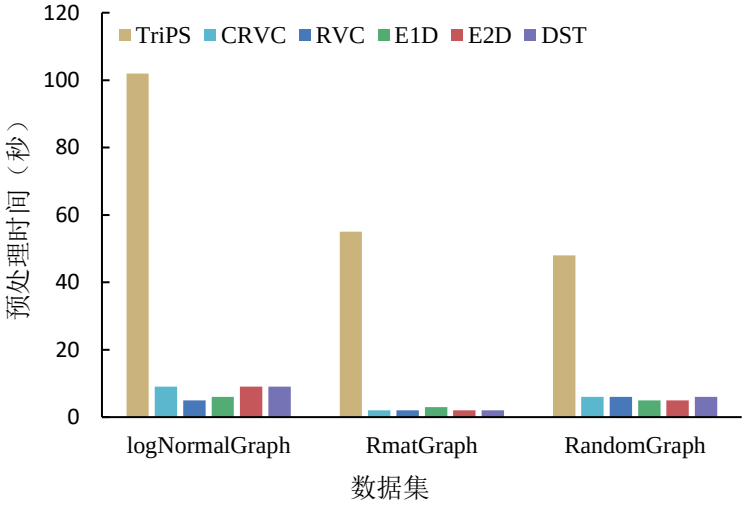

图 3-7　不同分区策略在合成图上的预处理时间

如图 3-8 所示，使用 TriPS 分区的三角计数算法在真实数据集上的性能比使用其他 Graphx 内置分区算法（CRVC、RVC、E1D、E2D 和 DST）快 15 到 273.6 倍。当数据集为 USA-road-d.USA.gr 时，使用 TriPS 的性能是最好的。图 3-9 展示了使用 TriPS 分区的三角计数算法在合成图上的性能，其平均速度是使用其他分区策略算法的 42.7 倍。虽然使用 TriPS 分区的预处理阶段比其他分区策略花费的时间长，但它对每个图数据只执行一次，而且结果可以保留并重复使用。因此，与采用内置分区策略的三角计数算法相比，TriPS 仍能大大提高性能。

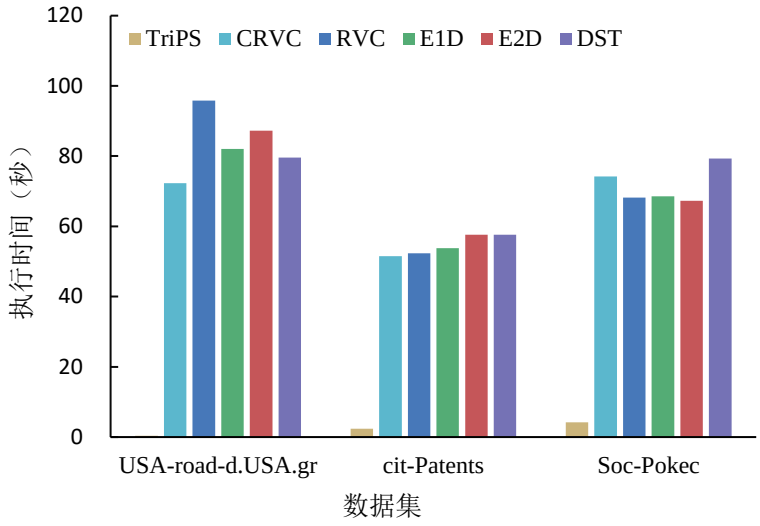

图 3-8　使用不同分区策略的三角计数算法在真实图上的执行时间

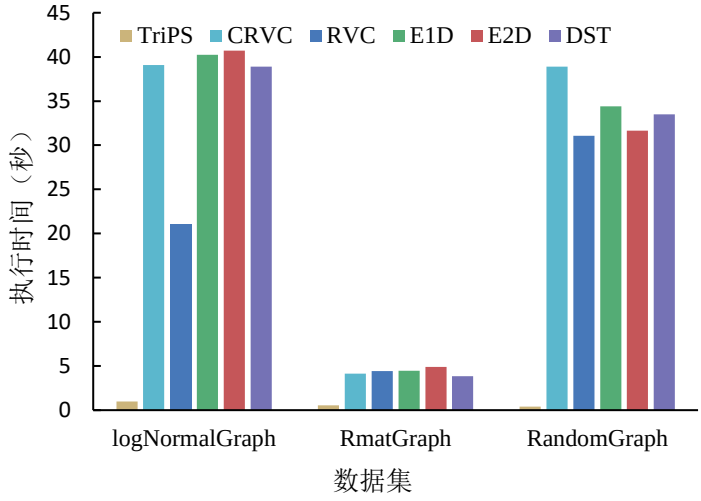

图 3-9　使用不同分区策略的三角计数算法在合成图上的执行时间

2.　可扩展性

为了衡量三角形数量对三角计数算法的预处理时间和执行时间的影响，应用 Graphx 中的 generateRandomEdges 函数来生成顶点和边数量相同但三角形数量不同的图。每个图中的顶点数为 300 万个，边数为 3000 万个。将三角形的数量从 900 万改变到 3600 万。图 3-10 给出了在大小相同而三角形数量不同的图上的三角计数算法的预处理时间和执行时间。结果发现预处理时间和执行时间都几乎与三角形的数量呈线性正比。

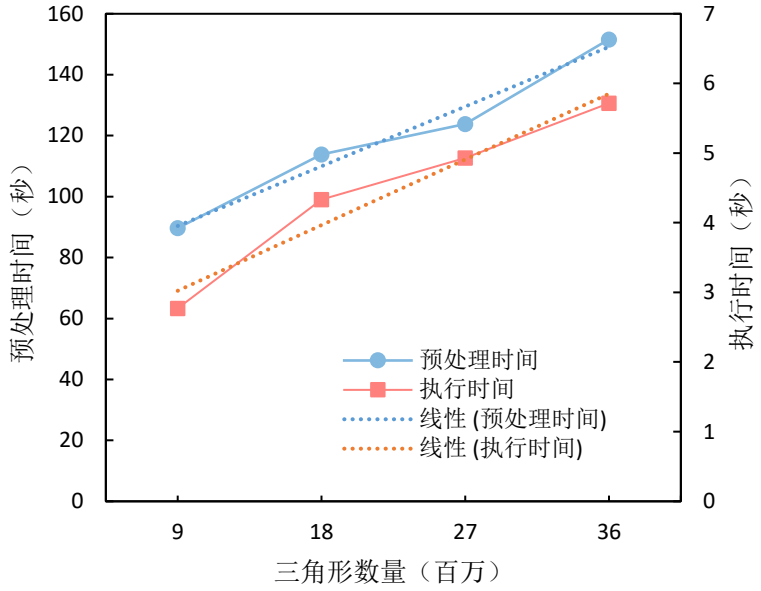

图 3-10　三角计数算法在包含不同三角形数量的图上的预处理时间和执行时间

3.4.2　PageRank 评估结果

1.　执行时间

将改进后的只需要一次 shuffle 的 PageRank 算法和原来需要两次 shuffle 的 PageRank 算法进行比较。需要一次 shuffle 的 PageRank 算法使用基于哈希分区策略，而原来的 PageRank 算法使用其他五种分区策略，分别是 CRVC、RVC、E1D、 E2D 和 DST。

图 3-11 展示了 PRHP 算法与使用内置分区策略的原来的 PageRank 算法在真实图上的执行时间的比较。从图 3-11 中发现，经过一次 shuffle 的改进的 PageRank 算法的性能明显优于原来需要两次 shuffle 的 PageRank 算法。PRHP 算法在数据集 cit-Patents、USA-road-d.USA.gr 和 Soc-Pokec 上分别比原来的 PageRank 算法快 85%、80%和 47%。对于合成图，也发现了相似的结果，需要一次 shuffle 的 PageRank 比使用 CRVC、RVC、E1D、E2D 和 DST 的原来的 PageRank 算法的性能更好，见图 3-12。例如，本文提出的 PageRank 算法要比使用 E1D 分区策略的原 PageRank 算法在 logNormalGraph 数据集上快 87%，比使用 DST 分区的原 PageRank 算法在 RandomGraph 数据集上快 64%。

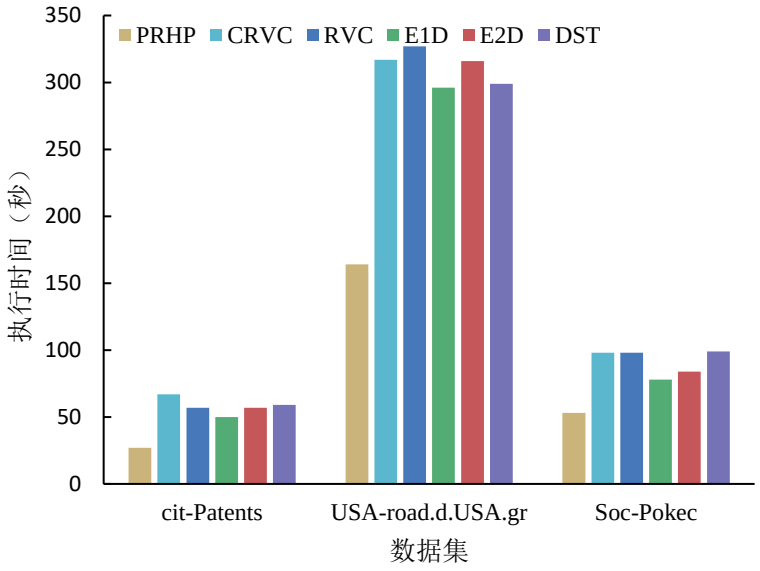

图 3-11　PRHP 和使用不同分区策略的传统的 PageRank 算法在真实图上的执行时间

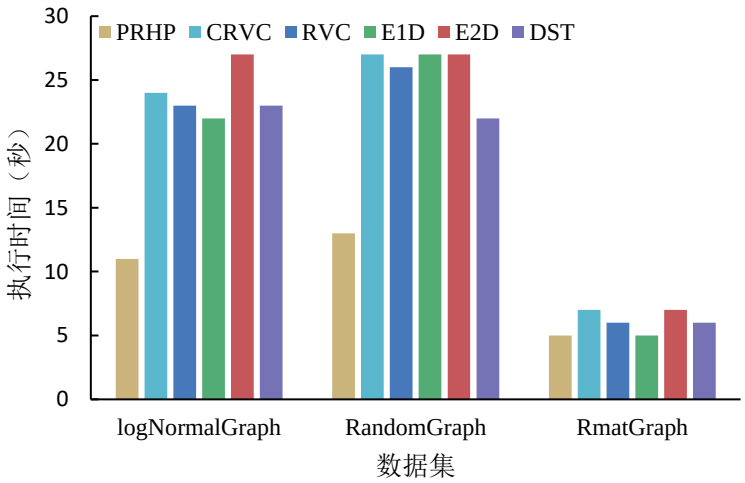

图 3-12　PRHP 和使用不同分区策略的传统的 PageRank 算法在合成图上的执行时间

2.　迭代次数的影响

为了研究迭代次数的影响，本文研究了 PRHP 和原来的 PageRank 的执行时间。在数据集 Soc-Pokec 上使用目前的五种分区策略，把分区数设置为 24，迭代次数依次为 5、10、15 和 20。从图 3-13 中可以看出，采用一次 shuffle 的 PageRank 性能最好，在迭代次数为 5、10、15 和 20 时，分别比采用 E2D 分区策略的传统 PageRank 快 36%、58%、67%和 59%。另外，随着迭代次数的增加，需要一个 shuffle 的 PageRank 的性能获得了更快的提高。

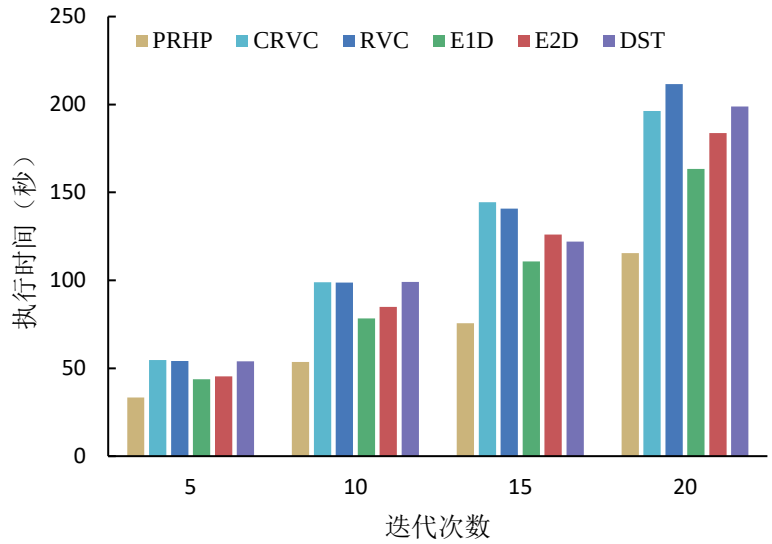

图 3-13　PRHP 和使用不同分区策略的传统 PageRank 在不同迭代次数上的性能

3.　可扩展性

在这个实验中，测量了本文提出的算法在具有不同顶点数的图上的性能。通过改变图的大小，生成了顶点数分别为 1000、10000、100000 和 1000000 的图。在图 3-14 中，当顶点数为 1000 或 10000 时，PRHP 算法与传统 PageRank 的执行时间相似。但当顶点数量增加时，例如，增加到 100000 或更多，一个 shuffle 的 PageRank 算法的性能明显更好。这是因为与传统的 PageRank 相比，只需要一次 shuffle 的 PageRank 的预处理有点复杂，因此当图很小时，PRHP 算法获得的优势会被预处理过程的成本所抵消。

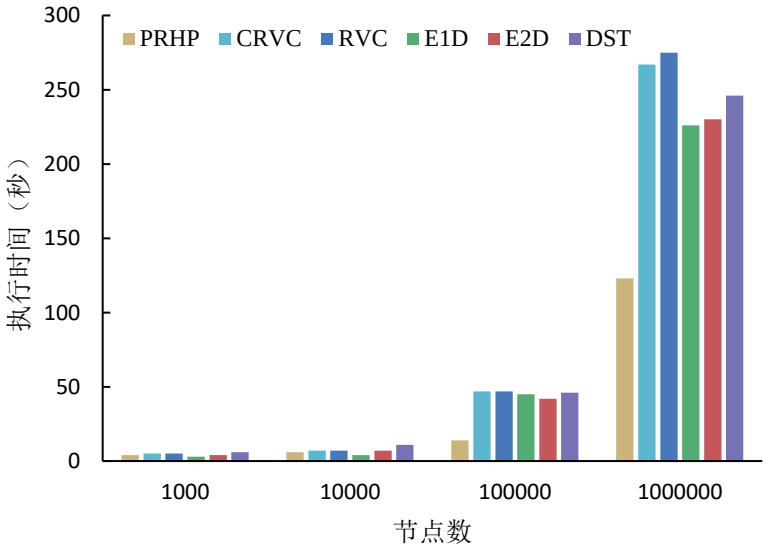

图 3-14　PRHP 和使用不同分区策略的传统 PageRank 算法在不同大小的图上的性能

另一方面，通过将分区数量设置为 12 到 48 来比较这两种算法。如图 3-15 所示，无论分区数量如何，需要一次 shuffle 的 PageRank 都优于传统的 PageRank。另外还发现分区的数量不能太大也不能太小，以避免过度分布或分布不足。从实验中可以看出，数据集 Soc-Pokec 的最好分区数为 24。

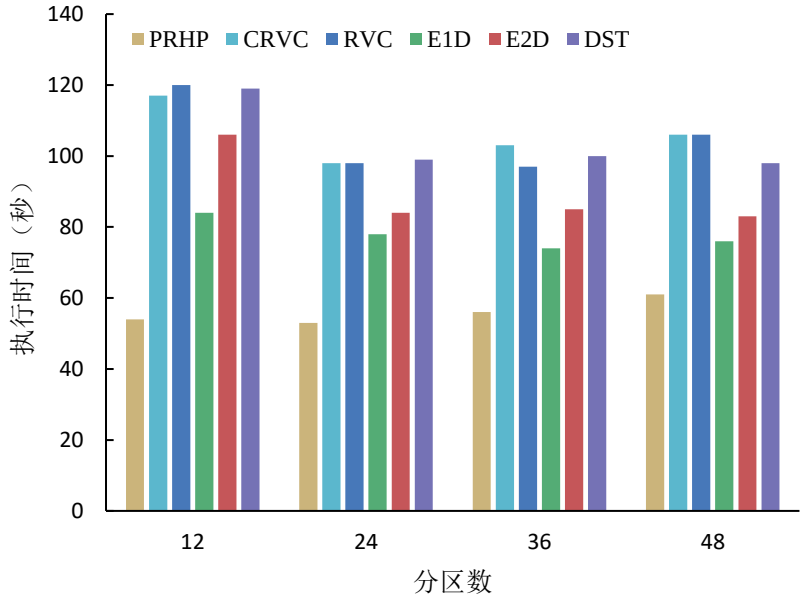

图 3-15　PRHP 和使用不同分区策略的传统 PageRank 算法使用不同分区数的性能

3.4.3　深入评估

为了评估的完整性，将本文提出的分区策略与其他非 Graphx 内置的分区策略[10,13,45]进行比较。此外，还分别比较了三角计数算法和 PageRank 算法在 Graphx 和另一个分布式图计算平台 GraphFrames[46]上使用相同分区策略的执行时间。

将使用基于度的哈希算法(DBH)[10]、混合切割策略(Hybrid)[13]和基于网格的分区方法(Grid)[45]的 PageRank 与需要一次 shuffle 的 PageRank 算法进行了比较。实验在三个数据集上进行：soc-LiveJournal、wikipedia 和 trackers。如图 3-16 所示，需要一次 shuffle 的 PageRank 算法的执行时间在这三个数据集上分别比使用 DBH 分区的 PageRank 算法快 73%、108%和 77%。

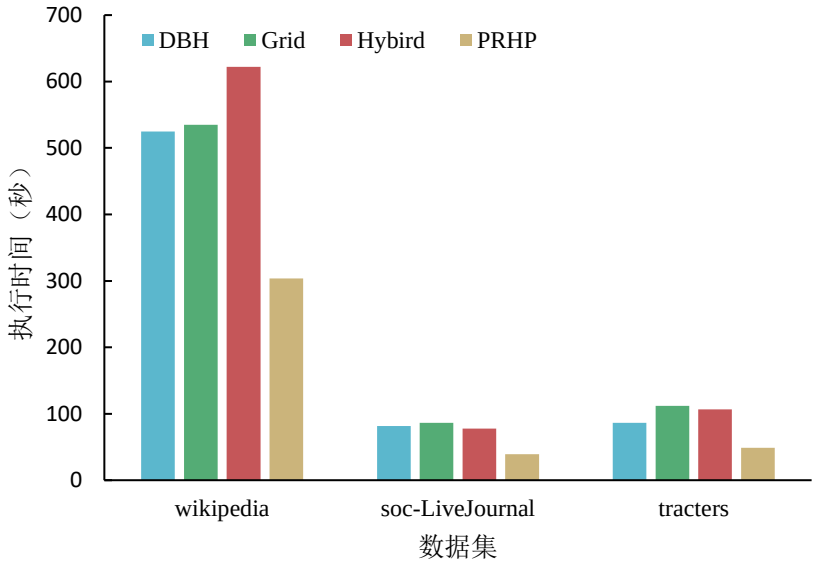

图 3-16　PRHP 和使用非内置分区策略的传统 PageRank 算法的性能

本文使用分区中最少的边数与最多的边数之比来衡量分区的平衡性，平衡比率越高表示分区越平衡。表 3-2 显示了 TriPS 在数据集 CTR.gr、logNormalGraph 和 RandomGraph 上获得了最好的平衡。加粗的数字表示对应行中最平衡的值。

表 3-2　在真实图和合成图上不同分区策略的平衡性

	TriPS	DBH	Grid	Hybrid
ULiveJournal	0.97707	**0.99008**	0.98439	0.96758
CTR.gr	**0.99997**	0.99982	0.9946	0.99738
logNormalGraph	**0.99967**	0.99176	0.9684	0.93325
RandomGraph	**0.99985**	0.9611	0.98919	0.95813

在图 3-17 中显示了在 GraphFrames 和 Graphx 平台上分别执行使用 TriPS 分区的三角计数算法的执行时间。通过对比可以观察到在 Graphx 上执行的三角计数算法在 ULiveJournal、CTR.gr、rmatGraph 和 RandomGrpah 数据集上的执行时间比在 GraphFrames 框架上执行要快得多。在 ULiveJournal 数据集上，Graphx 上的三角计数算法的执行效率比 GraphFrames 上的三角计数算法的执行效率快 6 倍。在 PageRank 上也发现了类似的结果。如图 3-18 所示，在数据集 ULiveJournal 和 CTR.gr 上，Graphx 上执行的改进的 PageRank 算法的性能比 GraphFrames 框架上执行的改进的 PageRank 算法的性能要好一倍以上。

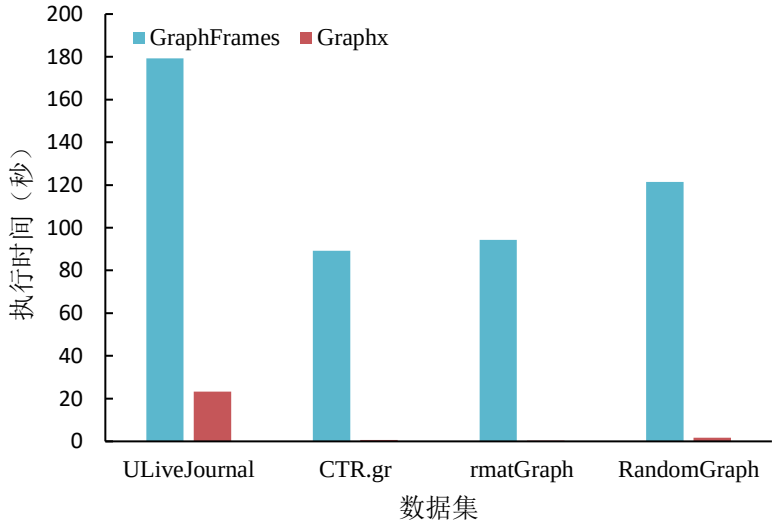

图 3-17　使用 TriPS 的三角计数算法在 Graphx 和 GraphFrames 上的执行时间

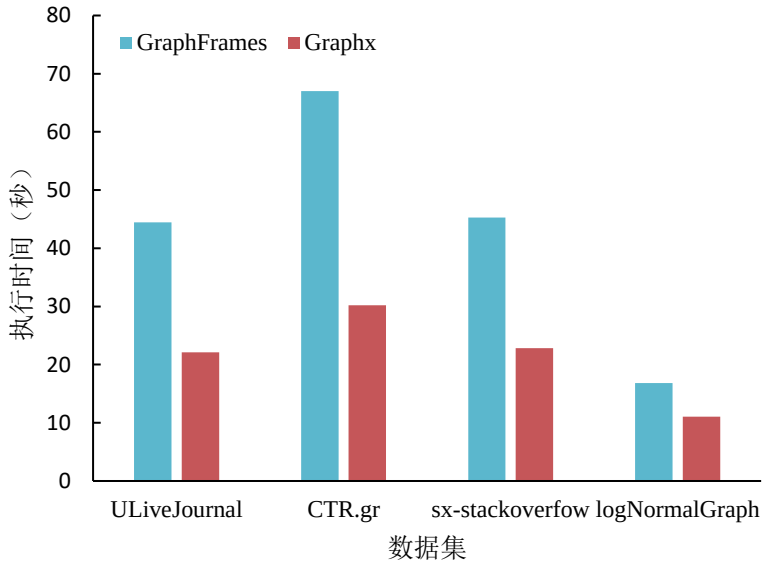

图 3-18　PRHP 在 Graphx 和 GraphFrames 上的执行时间

3.5　本章小结

本章提出了一种基于程序分析的图分区框架，该框架能够根据算法的特点和数据的流向选择最合适的划分策略。本文还设计了一种基于平衡的分区算法，并将其应用于三角计数等算法中。对于 PageRank 算法，一个基于哈希分区的改进的 PageRank 算法被提出，以减少每次迭代中的 shuffle 次数。本文提出的算法都是在 Graphx 框架上实现的。

第四章　基于图存储优化的社区发现研究

基于 Graphx 的图存储优化，在本章节通过结合密度峰值和标签传播算法提出了一种基于核心和边缘节点的社区发现算法(Community detection algorithm based on core and edge nodes, CD-CE)。这种方法不仅可以检测重叠社区，也可以检测非重叠社区。该方法由三部分组成：核心节点的选择、基于核心和边缘节点的标签传播以及标签重新分配。本章使用改进的密度峰值聚类来计算节点的重要性，使用切比雪夫不等式选择核心节点。然后在给边缘节点分配标签后，结合边缘节点和核心节点进行标签传播，为没有标签的节点分配标签。节点标签及其邻居标签不匹配的节点在标签重新分配阶段会被重新分配标签，以保证标签的准确性。本章在真实网络和合成网络上评估了准确性和性能。实验主要分为两部分来进行，分别对具有非重叠社区和重叠社区的网络进行评估。本章提出的算法分别与检测非重叠社区的算法和检测重叠社区的算法进行了比较。实验表明，与非重叠社区发现算法和重叠社区发现算法相比，本章提出的算法具有更好的准确性和更高的效率。

4.1　整体概述

该算法的整体流程如图 4-1 所示。首先，图$G(V,E)$被用作输入，其中$V = \{v_1,\dots,v_n\}$是数量为n的节点集合。在核心节点的选择部分，节点i的局部密度和最小距离用ρ_i和δ_i计算，将归一化的ρ_i^*和δ_i^*结合起来，得到节点i的重要性分数γ_i。使用切比雪夫不等式选择核心节点。在给边缘节点分配标签的部分，首先计算边缘节点j到核心节点的最短路径长度。在最小路径长度一致的情况下，使用基于信任的度量得到最近的核心节点c，那么边缘节点j的标签就是c。此时，图中的节点被分为三种类型：核心节点、边缘节点和未更新标签的节点。初始化阶段就是对这三种类型的节点进行初始化。然后，使用基于 Pregel 的标签传播算法为尚未获得标签的节点分配标签。在所有节点都获得了标签后迭代停止。最后，使用后面的公式(4.16)和(4.17)对节点标签与邻居的标签不匹配的节点重新分配标签，从而获得带有社区的网络。

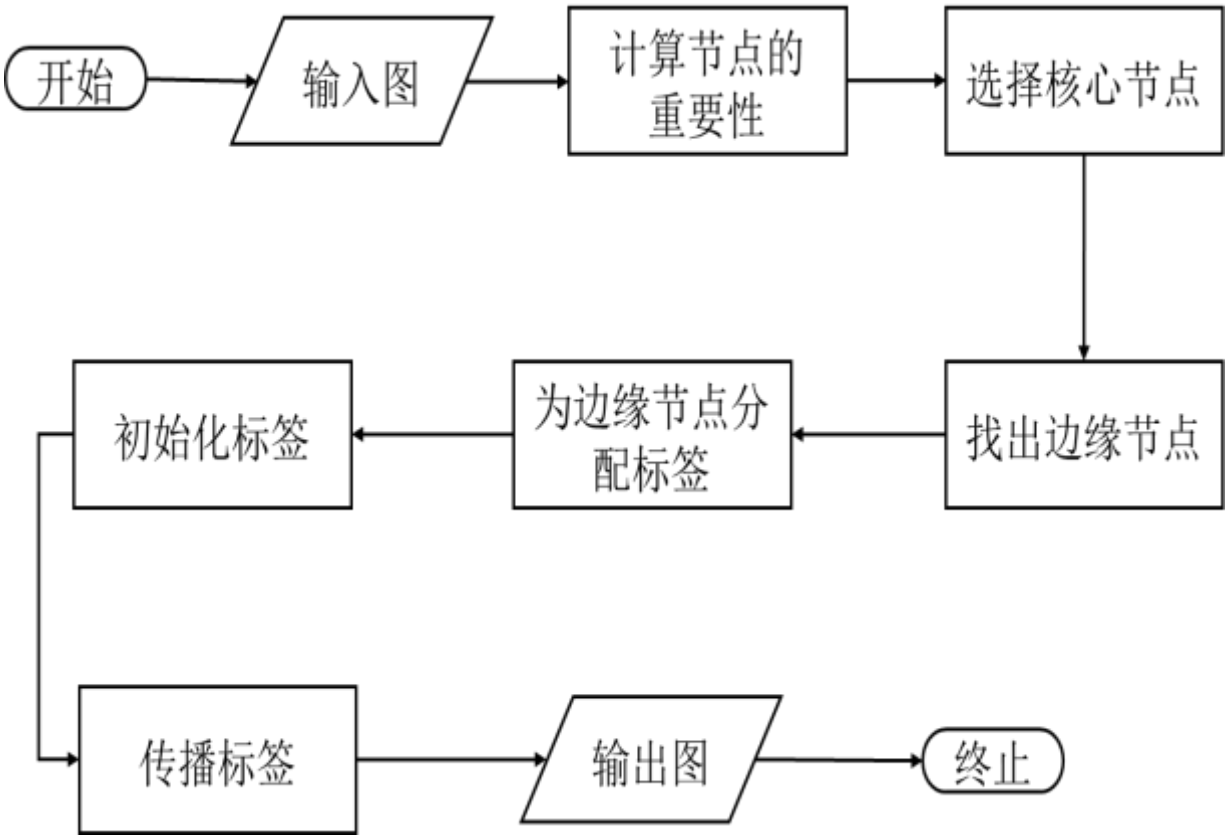

图 4-1　整体流程图

4.2　核心节点的选择

传统的标签传播算法忽略了节点之间的关系，并将所有的标签传播到整个网络。这种方法没有利用网络的拓扑信息，也没有注意到节点之间的差异。事实上，网络中的每个节点都有像度数和邻域等这些特征。因此，本文将网络中的拓扑信息融入到密度峰值聚类中，应用于选择核心节点的部分。选择核心节点的算法显示在算法 4.1 中。

4.2.1　局部密度

在密度峰值聚类中，集群中心通常有很高的密度。通常使用一个核函数来计算密度。而在网络中，可以利用节点的特征和它们之间的关系来计算节点的密度。因此，本章提出了一个新的计算局部密度的公式，如 4.1 所示。

$$\rho_i = \frac{2*(EN(i)+d(i))}{|N(i)|*(|N(i)|+1)} * d(i)^2 \tag{4.1}$$

这个公式使用节点的度和相邻节点之间的关联来计算节点的局部密度。其中$EN(i)$表示节点i的邻居节点之间的边数。$d(i)$表示节点i的度数。$N(i)$指节点i的邻居节点的数量。

算法 4.1　核心节点的选择

输入：网络$G(V,E)$；一个大于 0 的数e

输出：选择的核心节点集合$cores$

1) 计算网络G中每个节点的度数和包含的三角形数
2) 遍历每个节点，根据局部密度公式(4.1)计算节点的局部密度
3) 找出最大局部密度
4) **for each** v in V do
5) **if** v的局部密度! =最大局部密度 then
6) 使用公式(4.5)计算节点v的相对距离
7) **end if**
8) **end for**
9) 找到最大相对距离，将其分配给具有最大局部密度的节点
10) **for each** v in V **do**
11) 使用公式(4.7)对节点v的局部密度进行最小-最大归一化
12) 使用公式(4.8)对节点v的距离进行最小-最大归一化
13) 使用公式(4.6)计算节点v的重要性
14) **end for**
15) 节点重要性的平均值用E表示
16) 用σ表示节点重要性的标准差
17) **for each** i in V do
18) **if** $\gamma_i > (E + e * \sigma)$ then
19) 将节点i放入$cores$中
20) **end if**
21) **end for**
22) **return** $cores$

4.2.2　最小距离

在密度峰值聚类中，集群中心之间通常有相对较大的距离。这里提出了一个新的距

离度量标准来衡量节点之间的距离。ZieglerC N[47]认为社交网络中人与人之间的信任和偏好相似性之间存在着正相关。因此，如果两个节点有更多的共享邻居，他们之间就有更多的信任和更高的相似性。互相信任的节点就越有可能被分配到同一个社区。由于成员倾向于与受信任的成员交往，因此相对距离较短。本文便提出了一个基于“信任度”的距离度量，如(4.2)-(4.4)所示。

$$\alpha_{ij} = \frac{|CN(i,j)|+1}{max\ (|N(i)|,|N(j)|)} \tag{4.2}$$

在公式(4.2)中，α_{ij} 描述节点之间的关系。$CN(i,j)$表示节点i和节点j的共同邻居数量。

$$\beta_{ij} = \begin{cases} \frac{2*(E(CN(i,j)))}{|CN(i,j)|*(|CN(i,j)|-1)}, & |CN(i,j)| > 2 \\ 1, & |CN(i,j)| \leq 2 \end{cases} \tag{4.3}$$

在公式(4.3)中，$E(CN(i,j))$表示节点i和j的共同邻居之间的边的数量，β_{ij}描述了节点i和节点j之间边关系的紧密程度。

$$dist_{ij} = \frac{1}{log(1+\alpha_{ij})*log\ (1+\beta_{ij})} \tag{4.4}$$

在公式(4.4)中，$dist_{ij}$是基于“信任度”的距离度量。将公式(4.4)融入密度峰值聚类的距离公式中，得到节点间相对距离的计算公式为：

$$\delta_i = \begin{cases} \min\limits_{j:\rho_i<\rho_j} dist_{ij}, & if \exists j\ s.t.\ \rho_i < \rho_j \\ \max\limits_{j} dist_{ij}, & otherwise \end{cases} \tag{4.5}$$

4.2.3 核心节点的选择策略

密度峰值聚类的目标是确定具有相对高密度ρ_i和长距离δ_i的中心节点。为了确定哪些节点更重要，应该被选为核心节点，将最小-最大归一化密度ρ_i^*和距离δ_i^*相乘，得出γ_i。γ_i代表节点i的重要性。该公式如下所示：

$$\gamma_i = \rho_i^* * \delta_i^* \tag{4.6}$$

$$\rho_i^* = \frac{\rho_i - min\ (\rho)}{max(\rho) - min\ (\rho)} \tag{4.7}$$

$$\delta_i^* = \frac{\delta_i - min\ (\delta)}{max(\delta) - min\ (\delta)} \tag{4.8}$$

接下来使用切比雪夫不等式来选择核心节点。通过计算随机变量X的期望$E(X)$和非

零标准差$\sigma(X)$，对于任何实数$\varepsilon > 0$，它都满足如下公式：

$$P(|X - E(X)| \geq \varepsilon * \sigma(X)) \leq \frac{1}{\epsilon^2} \tag{4.9}$$

使用该方法可以找到核心节点和非核心节点的边界，它与ε相关。因此，满足以下公式的节点被认为是核心节点。

$$\gamma_i > (E(\gamma) + \varepsilon * \sigma(\gamma)) \tag{4.10}$$

4.3　基于核心和边缘节点的标签传播

4.3.1　边缘节点的标签

利用密度峰值聚类中给节点分配标签的想法。接下来找到所有的边缘节点，即度数为 1 的节点，并计算它们与核心节点的距离，这样每个边缘节点就可以选择最近的核心节点的标签。主要是使用改进的最短路径算法和基于信任的距离度量来获得边缘节点的标签。由于计算多个节点之间的最短路径很耗时，本节提出了一种改进的最短路径算法，为每个边缘节点找出具有最短路径长度的核心节点。当一个边缘节点有多个具有最短路径长度的核心节点时，使用基于信任的距离度量来选择最近的核心节点。

改进的最短路径算法的基本思想如下：核心节点将它们的标签和路径长度发送给相邻的节点。相邻节点收到邻居节点的标签和路径长度后，选择路径长度最短的标签进行更新，并将路径长度增加 1。已经更新的节点将其标签和路径长度发送给尚未更新的邻居节点，直到所有节点都被更新。通过这种方式，每个边缘节点可以快速获得路径长度最短的核心节点的标签。本文提出的方法只需要对每个节点进行一次更新，就可以迅速使边缘节点获得路径长度最短的核心节点的标签。

边缘节点到核心节点的最短路径长度存在于两种情况：如果边缘节点只收到一个核心节点的信息，那么该边缘节点就只离一个核心节点最近；如果边缘节点从多个节点接收信息，那么从该边缘节点到多个核心节点的最短路径长度是相等的。

用图 4-2 来说明，假设节点 1 和节点 2 是核心节点，核心节点向它们的邻居节点发送（ID，路径长度）键-值对。那么节点 3 收到(1,0)，(2,0)。因为这两个键值对中包含的路径长度相等，所以它们都被保留，并且路径长度都加 1。因此节点 3 保留的信息是(1,1)和(2,1)。节点 5 作为节点 1 的另一个邻居节点也收到(1,0)，路径长度加 1 后被(1,1)所更新。节点 6 和节点 7 收到(2,0)，路径长度都加 1，都变成(2,1)。根据收到信息的节点向

没有收到信息的邻居节点发送信息的原则，节点 6 和节点 7 没有可以发送信息的邻居节点。节点 3 和节点 5 分别向没有收到消息的相邻节点（节点 4 和节点 8）发送消息。节点 8 收到(1,1)，而节点 4 收到(1,1)和(2,1)。在路径长度加 1 之后，节点 8 到节点 1 的最短路径长度是 2，而节点 4 到节点 1 和 2 的最短路径长度都是 2。此时，在没有可以接收消息的节点之后，算法就自动终止了。因此，边缘节点 6 离核心节点 2 最近，边缘节点 8 离核心节点 1 最近，而节点 4 离核心节点 1 和核心节点 2 同样近。

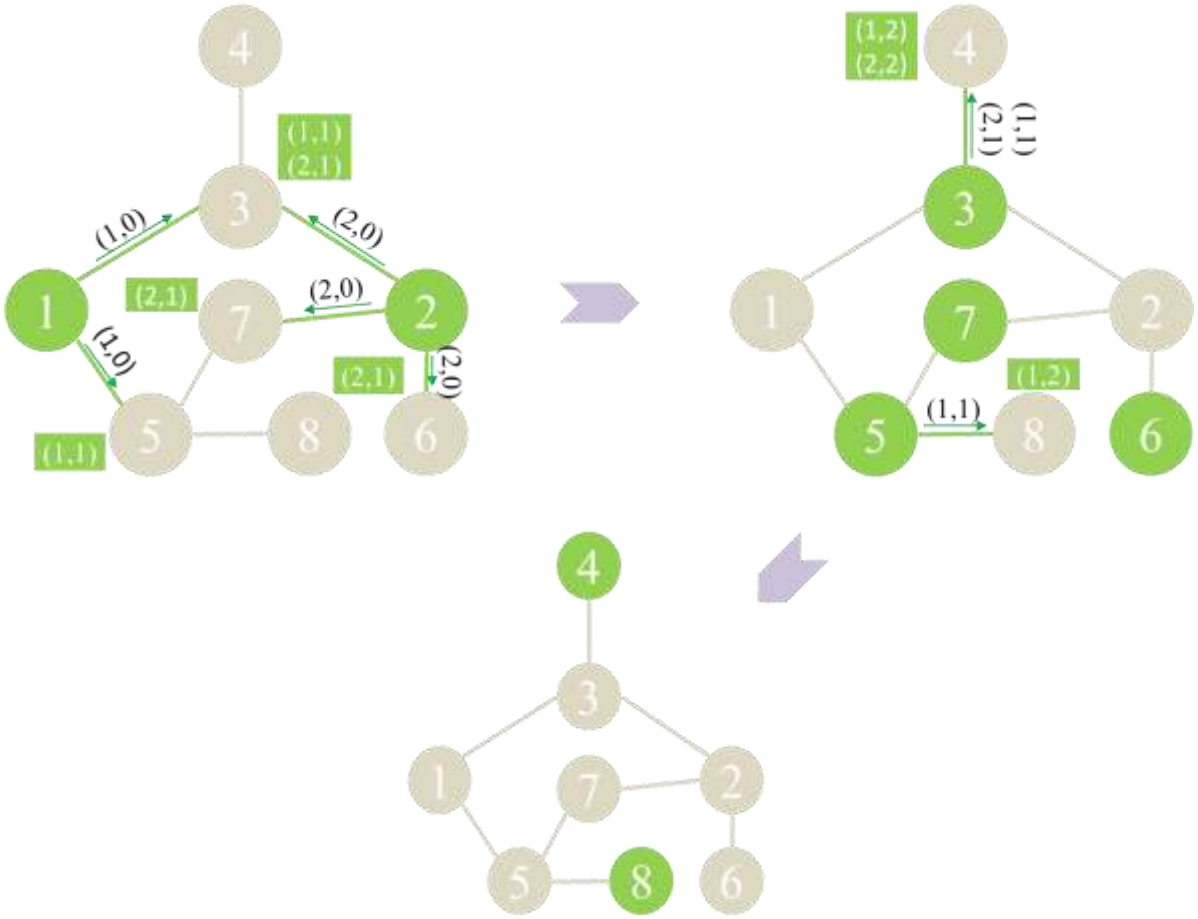

图 4-2　改进的最短路径算法的执行过程

仅仅使用改进的最短路径算法来确定一个边缘节点到一个核心节点的距离显然是不够的。对于一个边缘节点到多个核心节点的最短路径长度相等的情况，则使用基于信任的距离指标来进一步计算具有最短路径长度相等的边缘节点和核心节点之间的距离。为边缘节点获取标签的算法如算法 4.2 所示。

算法 4.2　为边缘节点分配标签的算法

输入：带有核心节点的网络$GWithCN(V,E)$

输出：为边缘节点分配核心节点的网络$GWithSP$

```
//改进的最短路径算法
用isUpdated和Map集合更新节点。
遍历每个节点，用isUpdated = true，Map((v,0))更新核心节点；用isUpdated =
false，Map()更新其余节点
用updatedNodes表示isUpdated = true的节点
while Count(updatedNodes) > 0 do
    将updatedNodes中的节点放入GWithSP中
    for each v in updatedNodes do
        将节点v的Map集合发送给尚未更新的邻居节点
    end for
    用unupdatedNodes表示isUpdated = false的节点
    for each v in 收到消息的 unupdatedNodes do
        更新节点v的isUpdated为true，并从Map中选择距离最小的核心节点，在它
们的距离上加 1
    end for
    从GWithCN中移除updatedNodes中的节点，剩下的节点重新组成GWithCN
    用updatedNodes表示isUpdated = true的节点
end while
//处理具有多个最短路径长度的边缘节点
for each v in 边缘节点 do
    if v距离多个核心节点都有最短路径长度 then
        使用公式(4.4)来选择距离最小的核心节点的标签
    end if
end for
return GWithSP
```

4.3.2 标签初始化

在获得核心节点和边缘节点的标签后，用标签列表$L(i)$和$isUpdated$初始化节点。对于核心节点，它们使用自己的Id和$true$来初始化。对于边缘节点，使用上一部分获得的标签和$true$来初始化，而其他节点则用0和$false$初始化。

4.3.3 标签的传播

在这节的标签传播方法中，每个节点向各个邻居节点传递标签的概率是可变的，因为每个节点的重要性和它们之间的相似度都是不同的。标签传播的概率随着节点重要性和相似性的增加而增加。使用下列公式来表示这种可能性：

$$p_{ij} = \gamma_i \frac{Sim(i,j)}{\sum_{k \in N(i)} Sim(i,j)} + Sim(i,j) \tag{4.11}$$

$$p_{ji} = \gamma_j \frac{Sim(i,j)}{\sum_{k \in N(i)} Sim(i,j)} + Sim(i,j) \tag{4.12}$$

在公式(4.11)中p_{ij}表示节点i向节点j传播标签的概率。在公式(4.12)中p_{ji}表示节点i收到节点j的标签的概率。

利用两个节点之间的距离越大，它们的相似度越小的特点，可以得到相似度公式为：

$$Sim(i,j) = \frac{1}{dist_{ij}} \tag{4.13}$$

当一个节点从邻居节点收到多个标签时，用以下公式计算每个标签的权重：

$$P(l_i) = \sum_{j \in N(i)} p_{ji} * TF(l_i, l_j) \tag{4.14}$$

$$TF(l_i, l_j) = \begin{cases} 1, & l_i = l_j \\ 0, & l_i \neq l_j \end{cases} \tag{4.15}$$

在公式(4.15)中$TF(l_i, l_j)$表示节点i和节点j的标签是否相同，如果相同为 1，不同为 0。

根据权重将标签从大到小排序，然后定义一个自定义的阈值φ，该阈值表示一个节点可以拥有的最大标签数。如果标签的数量小于或等于φ，所有的标签都被选为该节点的标签；如果标签的数量大于φ，则选择前φ个标签。传播标签阶段的算法描述见算法 4.3。

算法 4.3　基于核心和边缘节点的标签传播算法

输入：初始化后的图$initGraph$；一个节点可以拥有的最大标签数φ

输出：带标签的图$LabelGraph(V,E)$

1) 用$updatedNodes$表示$isUpdated = true$的节点
2) **while** $updatedNodes$的数量大于 0 do
3)　将$updatedNodes$放入$LabelGraph$中
4)　**for each** i in $updatedNodes$ do
5)　　**for each** j in i的尚未更新的邻居 do
6)　　使用公式(4.11)和(4.12)来计算节点传播标签的概率。
7)　　**end for**
8)　**end for**
9) **for** $initGraph$中$isUpdated = false$的节点i do
10)　**if** i 收到了消息 then
11)　　使用公式(4.14)来计算收到的每个标签的重要性。
12)　　按重要性降序排列标签。
13)　　**if** 非重叠社区 then
14)　　选择第 1 个标签
15)　　**else**
16)　　选择前φ个标签
17)　　**end if**
18)　**end if**
19) **end for**
20) 从$initGraph$中移除$updeatedNodes$
21) 重新用$updatedNodes$表示$isUpdated = true$的节点
22) **end while**
23) **return** $LabelGraph$

4.4　标签重分配

为了提高标签分配的准确性，找出那些标签与邻居的标签不匹配的节点重新分配标

签。这些节点将接收其邻居节点发送的标签，然后计算每个标签的频次和到每个标签所在节点的最高相似度。使用公式(4.16-4.17)来记录节点i收到的标签的频次以及与每个标签所在节点的最大相似度。

$$F(l_i) = \sum_{j \in N(i)} TF(l_i, l_j) \tag{4.16}$$

$$maxSim(l_i) = \max_{j \in N(i)} Sim(i,j) * TF(l_i, l_j) \tag{4.17}$$

在上述公式中l_i表示节点i的标签。在计算了每个标签的频次和最大相似度后，将标签按降序排列，频次为第一顺序，最大相似度为第二顺序。然后根据要保留的标签数量，选择属于节点的标签。该算法在算法 4.4 中描述。

算法 4.4 标签重分配

输入：带有标签的图$LabelGraph(V,E)$；每个节点的最大标签数φ；最大迭代次数$MaxIter$

输出：带标签的最终图$FinalGraph$

1) $num = 0$
2) **while** $num < MaxIter$ do
3) **for each** i in V do
4) 在获得i的邻域节点后，使用公式(4.16)和(4.17)来计算i邻域的标签频次和最
5) 大相似度
6) 相邻节点的标签按照先频次后相似度的顺序降序排序
7) 选择前φ个标签
8) **end for**
9) **end while**
10) **return** $FinalGraph$

4.5 实验分析

在本节验证和评估了 CD-CE 算法在真实网络和合成网络上的表现。该实验是在一个由六台相同配置的机器组成的集群上进行的。每台机器都配置了英特尔®酷睿 i9-10900K、3.70GHz、20 核 CPU、64GB 内存和 1000MB/s 的带宽。实验主要分为两个部分：对含有非重叠社区的网络的实验评估和对含有重叠社区的网络的实验评估。

4.5.1　数据集

本节使用了两种类型的数据集：真实世界的网络和合成网络。真实世界的网络如表4-1 所示。此外，合成网络使用 LFR[48]基准来生成。主要生成了两种类型的网络，一种包含非重叠社区，另一种包含重叠社区。合成网络的相关参数显示在表 4-2 和表 4-3 中。

表 4-1　真实数据集

名称	节点数	边数	社区数
Karate[44]	34	78	2
Dolphins[44]	62	159	2
Football[49]	115	613	12
Polbooks[49]	105	441	3
Polblogs[49]	1490	19090	2
Email-eu-core[41]	1005	25571	42
Amazon[41]	334863	925872	75149
Orkut[41]	3072441	117185083	6288363
Youtube[41]	1134890	2987624	8385
DBLP[41]	317080	1049866	13477

表 4-2　非重叠社区 LFR1 的参数表

节点数	平均度	最大度	最小规模	最大规模	序列负指数	分布负指数	混合参数
10000	20	100	50	100	2	1	0.1-0.8

表 4-3　重叠社区 LFR2 的参数表

节点数	平均度	最大度	最小规模	最大规模	重叠节点比例	最大标签数	混合参数
10000	20	100	50	100	0.1	3	0.1-0.8

4.5.2　评估指标

因为本实验既与检测非重叠社区的算法进行比较，也与检测重叠社区的算法进行比较。因此，本节选择了一些评估标准来分别评估检测非重叠社的准确性和检测重叠社区

的准确性。主要包括标准化互信息(NMI)[50]、模块度(Q_{ov})[51]、 F1 分数(F1-Score)[52]、 调节的兰德系数(ARI)[53]和准确度(ACC)[54]。

1. NMI

NMI 是一个被广泛使用的衡量标准，用于衡量真实社区和发现社区的相似度。可以有效评估网络中寻找社区的算法的准确性。NMI 的取值范围在[0,1]之间。该值越接近 1，说明发现的社区越接近真实社区，用于发现社区的算法的准确性越高。该公式描述如下：

$$NMI(X,Y)=\frac{-2\sum_{i=1}^{C_X}\sum_{j=1}^{C_Y}N_{ij}log(\frac{N_{ij}N}{N_iN_j})}{\sum_{i=1}^{C_X}N_ilog\,(\frac{N_i}{N})+\sum_{j=1}^{C_Y}N_jlog\,(\frac{N_j}{N})} \tag{4.18}$$

在公式(4.18)中C_X指的是真实社区的数量。C_Y指的是发现社区的数量。N_{ij}是混淆矩阵中的一个元素，代表真实社区i和发现社区j中共同节点的数量。N_i表示混淆矩阵中第i行的元素之和，N_j表示混淆矩阵中第j列的元素之和。

2. Q_{ov}

Q_{ov} 是一个可以在不知道真实社区的情况下衡量社区划分质量的方法。它衡量社区内的密度以及社区之间的连接程度。公式如下：

$$Q_{ov}=\frac{1}{2m}\sum_{c=1}^{K}\sum_{i,j\in C_c}\frac{1}{O_iO_j}(A_{ij}-\frac{k_ik_j}{2m}) \tag{4.19}$$

在公式(4.19)中，m代表网络中的边的数量。K指的是社区的数量。C_c表示第c个社区集合。A_{ij}是节点i和节点j之间的权重。k_i和k_j分别表示i和j的度。O_i和O_j分别代表节点i和节点j所在社区的数量。当O_i和O_j都是 1 时，这个指标也可以用来评估非重叠社区。

3. F1-Score

F1-Score 也是衡量发现社区和真实社区准确性的一个指标。它是真实社区到每个发现社区最匹配的 F1-Score 和发现社区到每个真实社区最匹配的 F1-Score 的平均值。F1-Score 的计算公式如下所示：

$$F1-Score=\frac{1}{2}(\frac{1}{|C|}\sum_{C_i\in C}F1(C_i,C'_{p(i)})+\frac{1}{|C'|}\sum_{C'_i\in C'}F1(C'_i,C_{p'(i)})) \tag{4.20}$$

$$p(i)=argmax_kF1(C_i,C'_k) \tag{4.21}$$

$$p'(i)=argmax_kF1(C_k,C'_i) \tag{4.22}$$

在上述公式中，$p(i)$表示发现社区中的社区C'_k和真实社区C_i取得的最大 F1 值。同样，

$p'(i)$指的是真实社区中的社区C_k在真实社区和发现社区C_i'达到的最大 F1 值。$F1(C_i, C_k')$指精确度和召回率的调和平均值。

4. ARI

ARI 反映了两个分区之间的重叠程度，数值越高越好，最高为 1。使用这个指标需要数据本身具有类别标记，并适用于非重叠的社区。

$$ARI = \frac{\Sigma_{ij}\binom{n_{ij}}{2} - (\Sigma_i\binom{b_i}{2}\Sigma_j\binom{t_j}{2})/\binom{n}{2}}{\frac{1}{2}(\Sigma_i\binom{b_i}{2}+\Sigma_j\binom{t_j}{2}) - (\Sigma_i\binom{b_i}{2}+\Sigma_j\binom{t_j}{2})/\binom{n}{2}} \tag{4.23}$$

在公式(4.23)中n_{ij}表示真实社区i和发现社区j中的共同节点数。b_i表示真实社区i中的节点数。t_j表示在发现社区j中的节点数。

5. ACC

ACC 是另一种评估检测到的社区准确性的方法。它被用来评估非重叠的社区。ACC 是由 Huang 和 Ng[54]提出的。

$$ACC = \frac{\sum_{i=1}^{l} num_i}{n} \tag{4.24}$$

在公式(4.24)中，n表示节点数，num_i表示从社区i中正确识别的节点数，l是社区的数量。

4.5.3　对非重叠社区的实验评估

在本节中，将 CD-CE 算法与这四种算法 PCOPRA[55]、PSCAN[56]、LPA[57]、PLDLS[58]进行比较，以检测真实网络和合成网络上的非重叠社区。PCOPRA 算法是 COPRA 算法的并行版本。PSCAN 是 SCAN 算法的并行版本。LPA 是 Graphx 中的内置算法。PLDLS 是一种基于标签扩散和标签选择的算法。这里选择 NMI、Qov、F1-Score、ARI 和 ACC 来评估这五种算法在非重叠社区检测方面的准确性。

首先使用 NMI 标准来评估检测真实网络上的非重叠社区的算法。对 CD-CE 和其他四种算法在五个真实世界网络上的 NMI 评估结果见表 4-4。PCOPRA 和 LPA 在六个数据集上都没有良好的表现。PSCAN 算法可以很好地检测出 football 中的社区，获得了五种算法中的最高值。另外，PLDLS 算法在 karate 和 polblogs 上检测社区的效果最好。然而，CD-CE 算法在 dolphins、polbooks 和 email-Eu-core 上都获得了最高的 NMI 值，而且在 dolphins 上达到了最高值 1，这意味着 dolphins 中的所有社区都可以被准确检测到。

表 4-4　不同算法在非重叠社区获得的 NMI 值

	karate	dolphins	football	polbooks	polblogs	email-Eu-core
CD-CE	0.837	**1**	0.900	**0.673**	0.580	**0.687**
PCOPRA	0.223	0.563	0.823	0.531	0.540	0.455
PSCAN	0.503	0.202	**0.914**	0.522	0.190	0.588
LPA	0.330	0.512	0.839	0.448	0.542	0.338
PLDLS	**1**	0.676	0.659	0.601	**0.692**	0.043

表 4-5 显示了五种算法在真实世界网络中的 F1-Score 评估结果。从表中可以看出，使用 F1-Score 评估时，PCOPRA 和 LPA 仍然表现不佳。PSCAN 在 football 上取得了最高的 F1-Score 值，与使用 NMI 评估时相似。PLDLS 只在 karate 上取得了最高的 F1-Score 值。当使用 F1-Score 作为评估标准时，除了 karate 和 football，CD-CE 算法在其他四个数据集上都取得了最高的 F1-Score 值。

表 4-5　不同算法在非重叠情况下获得的 F1-Score 值

	karate	dolphins	football	polbooks	polblogs	email-Eu-core
CD-CE	0.970	**1**	0.743	**0.770**	**0.753**	**0.376**
PCOPRA	0.614	0.596	0.410	0.562	0.488	0.138
PSCAN	0.389	0.144	**0.773**	0.400	0.101	0.150
LPA	0.572	0.717	0.527	0.504	0.459	0.107
PLDLS	**1**	0.747	0.351	0.753	0.631	0.020

在表 4-6 中显示了五种算法在六个真实网络上使用 ARI 评估标准的评价结果。PSCAN 在 football 上的 ARI 得分最高。PLDLS 在 karate 和 polblogs 数据集上有最高的 ARI 分数。PCOPRA 和 LPA 在所有六个数据集上的评价结果都比较差。CD-CE 算法在 dolphins、polbooks 和 email-Eu-core 数据集上的 ARI 得分最高。虽然 PSCAN 在 football 上的 ARI 值比 CD-CE 高出 5.6%，但 CD-CE 在 karate、polbooks、polblogs 和 email-Eu-core 上分别比 PSCAN 高出 195%、24%、3765%和 570%。虽然 PLDLS 在 karate 和 polblogs 上的 ARI 分数分别比 CD-CE 高 0.12 倍和 0.21 倍。然而，CD-CE 算法在 dolphins、football

和 polbooks 上的 ARI 得分比 PLDLS 高 0.72 倍、1 倍和 0.09 倍。

表 4-6　不同算法在非重叠社区获得的 ARI 值

	karate	dolphins	football	polbooks	polblogs	email-Eu-core
CD-CE	0.882	**1**	0.826	**0.730**	0.657	**0.516**
PCOPRA	0.069	0.608	0.654	0.630	0.575	0.084
PSCAN	0.299	-	**0.852**	0.587	0.017	0.077
LPA	0.269	0.532	0.663	0.496	0.573	0.034
PLDLS	**1**	0.580	0.412	0.668	**0.797**	-

在六个真实网络数据集上还使用 ACC 指标对五种算法进行了评估。如表 4-7 所示，CD-CE 在 dolphins、polbooks 和 email-Eu-core 上获得了最高的准确率。PLDLS 在 karate 和 polblogs 上有最高的准确率。PSCAN 在 football 上的准确率最高。得出的结论与使用 NMI 和 ARI 进行评估时相同。

表 4-7　不同算法在非重叠社区获得的 ACC 值

	karate	dolphins	football	polbooks	polblogs	email-Eu-core
CD-CE	0.971	**1**	0.765	**0.867**	0.904	**0.491**
PCOPRA	0.647	0.823	0.730	0.781	0.725	0.182
PSCAN	0.5	0.387	**0.896**	0.714	0.114	0.134
LPA	0.647	0.855	0.739	0.752	0.695	0.146
PLDLS	**1**	0.790	0.574	0.849	**0.944**	0.101

为了进一步扩展实验，使用公式(4.19)在六个真实世界的数据集上评估这五种算法。根据表 4-8，尽管 PCOPRA 在使用 NMI、F1-Score、ARI 和 ACC 进行评估时，没有在六个数据集中的任何一个上获得最高值。然而，当使用模块度进行评估时，该算法在 football 和 polbooks 上获得了最高的模块度。相比之下，PSCAN、LPA 和 PLDLS 在每个数据集上都没有获得最高的模块化值。除此以外，除了 football 和 polbooks 数据集，在其他 4 个数据集上，CD-CE 都获得了最高的模块化值。

表 4-8　不同算法在非重叠社区获得的模块化值

	karate	Dolphins	football	polbooks	polblogs	email-Eu-core
CD-CE	**0.372**	**0.518**	0.579	0.446	**0.306**	**0.189**
PCOPRA	-	0.278	**0.596**	**0.511**	0.245	0.014
PSCAN	0.311	0.214	0.579	0.470	0.071	-
LPA	0.091	0.266	0.581	0.424	0.219	-
PLDLS	0.371	0.481	0.454	0.450	0.305	-

本文还对合成网络进行了实验分析。将合成网络的混合参数 mu 设定为 0.1-0.8。mu 值决定了生成网络的复杂性。mu 值越小，社区之间的界限就越清晰，识别社区也就越容易。mu 值越大，网络中边之间的联系就越紧密，社区就越难被识别。通过改变 mu 值，本文用 NMI 和 F1-Score 评估了合成网络。

在图 4-3 中可以看出，当 mu 在 0.1 和 0.5 之间时，CD-CE、PCOPRA 和 LPA 的 NMI 值都很接近，但 CD-CE 取得的 NMI 值最高。当 mu 大于 0.5 时，CD-CE 是除 PSCAN 以外下降最慢的。在 mu 在 0.1 和 0.7 之间，CD-CE 算法的 NMI 值是所有算法中最高的。当 mu 在 0.1 和 0.3 之间时，PSCAN 算法有一个下降的趋势，当 mu 超过 0.3 时就稳定下来。PLDLS 算法随着 mu 的增加，算法得到的 NMI 值也有下降的趋势。当 mu 为 0.8 时，PCOPRA、LPA 和 PLDLS 的 NMI 值都趋近于 0。

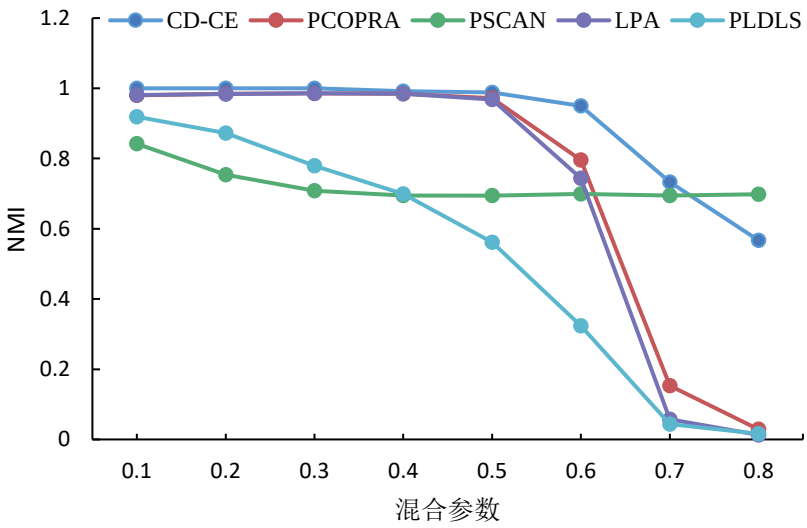

图 4-3　合成网络上的非重叠社区的 NMI 值

图 4-4 显示了五种算法在合成网络上使用 F1-Score 评估的结果。从该图可以看出，在 mu 为 0.1-0.8 时，CD-CE 算法的 F1-Score 值在所有算法中最高。PCOPRA 和 LPA 得到的 F1-Score 值最接近。CD-CE 在 mu 大于 0.6 时显示出急剧下降的趋势。而 LPA 和 PCOPRA 在 mu 为 0.4 以后也显示出急剧下滑的趋势。而 PSCAN 和 PLDLS 则表现出先快速下降，然后随着 mu 的增加缓慢下降的总体趋势。在 mu 为 0.8 时，除了 CD-CE，所有算法的 F1-Score 都接近于 0。

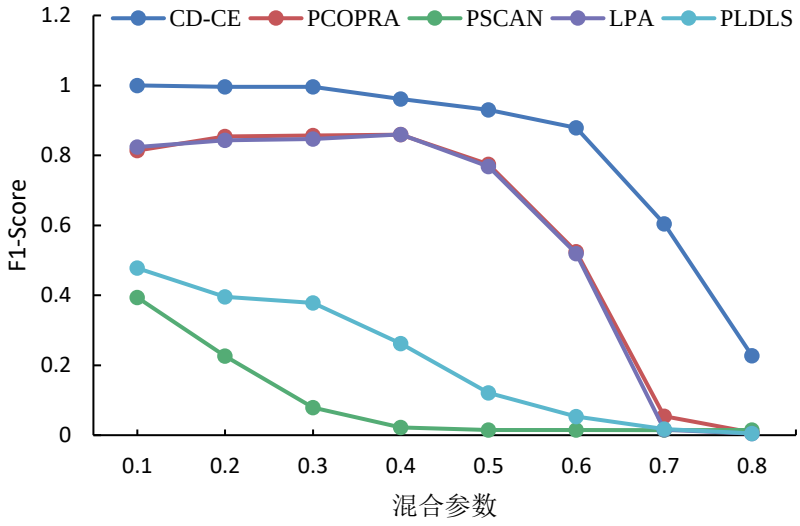

图 4-4　合成网络上非重叠社区的 F1-Score 值

4.5.4　对重叠社区的实验评估

在这一部分，选择 PASLPA[59]、PCOPRA[60]、PGLPA[61]、PMLPA_IM[62]、PMLPA[63] 这五种算法和 CD-CE 算法来比较重叠社区的检测结果。其中，PASLPA 算法是 SLPA 算法的改进版本。PCOPRA 算法是 COPRA 算法的并行版本。PGLPA 是一种并行灰色标签传播算法。PMLPA_IM 是一种基于影响模型的并行多标签传播算法。PMLPA 是一种并行多标签传播算法。这里使用三个指标来评估检测重叠社区的准确性：NMI、Q_{ov} 和 F1-Score。此外，还评估了六种算法的性能和 CD-CE 的可扩展性。

图 4-5 显示了六种算法在真实世界网络中检测重叠社区使用 NMI 进行评估的评估结果。从图中可以看出，在 amazon 和 orkut 中，PMLPA_IM 算法获得了最低的 NMI 值，也就是说，该算法的准确性很低。而在 dblp 中，PMLPA 算法的表现最差。然而，CD-

CE 算法获得了六种算法中最高的 NMI 值。这表明 CD-CE 所发现的社区是最接近真实的。

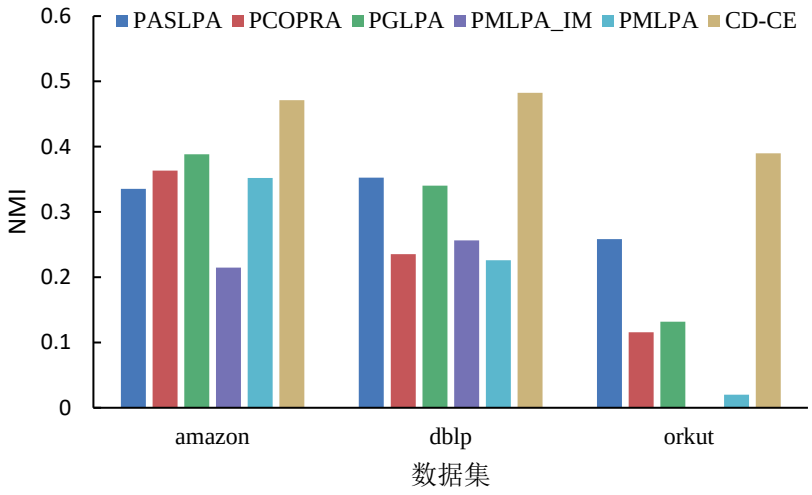

图 4-5　现实世界网络中的重叠社区的 NMI 值

用 F1-Score 来评估六种算法在三个真实网络上的表现，如图 4-6 所示。在这三个图上表现最差的算法是 PMLPA。在 dblp 中，CD-CE 算法获得的 F1-Score 值分别比 PASLPA、PCOPRA、PGLPA、PMLPA_IM 和 PMLPA 获得的 F1-Score 值高 1.6%、28%、5.3%、14.5%、119%。在 amazon 数据集上，PASLPA 获得了最高的 F1-Score 值，其次是 CD-CE，而在 dblp 和 orkut，CD-CE 算法获得了最高的 F1-Score 值。

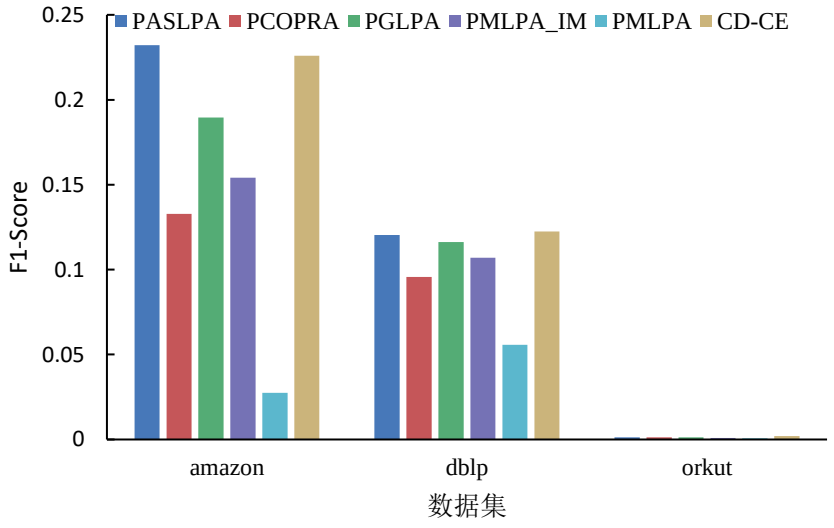

图 4-6　现实世界网络中的重叠社区的 F1-Score 值

Q_{ov} 是衡量社区质量的一个重要指标。在图 4-7 中，显示了在三种数据集上的运行的六种算法的模块度。从图中可以看出，CD-CE 算法在所有三个数据集上的模块化程度最高。在 amazon 和 dblp 中，PASLPA 算法的表现最差。在 orkut 数据集上，PMLPA_IM 获得了最低的模块度。与检测重叠社区的五种算法相比，CD-CE 在使用 NMI 和模块化指标时都获得了最高分。

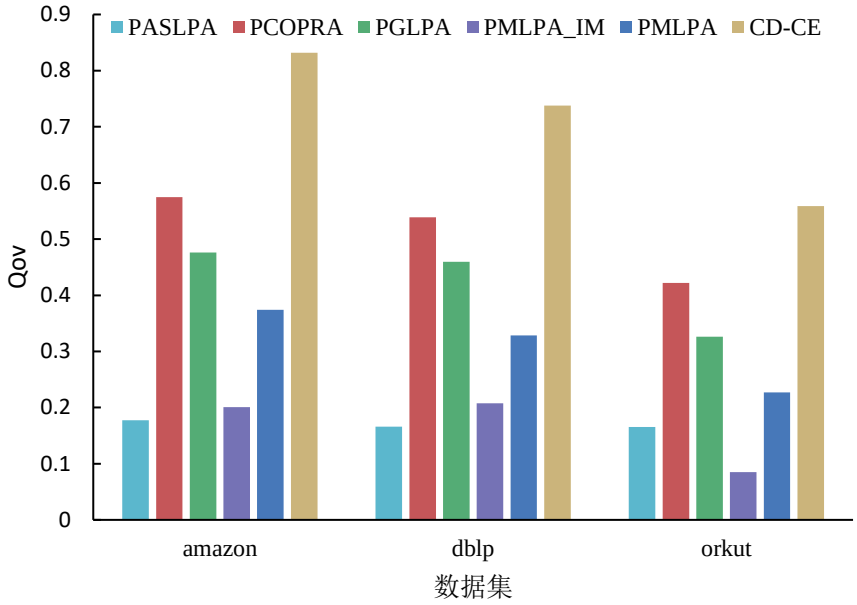

图 4-7　现实世界网络中的重叠社区的模块度

图 4-8 显示了六种算法在不同 mu 值的合成网络上的 NMI 值。当混合参数 mu 的值小于等于 0.5 时，PCOPRA 和 CD-CE 的 NMI 值相当接近。而且它们的 NMI 值都大于 PMLPA、PMLPA_IM、PGLPA 和 PASLPA。当 mu 值大于 0.5 时，PCOPRA 算法的 NMI 下降得最快。其他算法的下降速度较慢。然而，CD-CE 算法的 NMI 是所有算法中最高的。当 mu 值较大时，网络中的社区将更加模糊，更难识别。网络越模糊，鲁棒性越高的算法将显示出更高的准确性。当 mu 值大于 0.5 时，CD-CE 算法获得了最高的 NMI 值，这也表明了 CD-CE 算法的高鲁棒性。

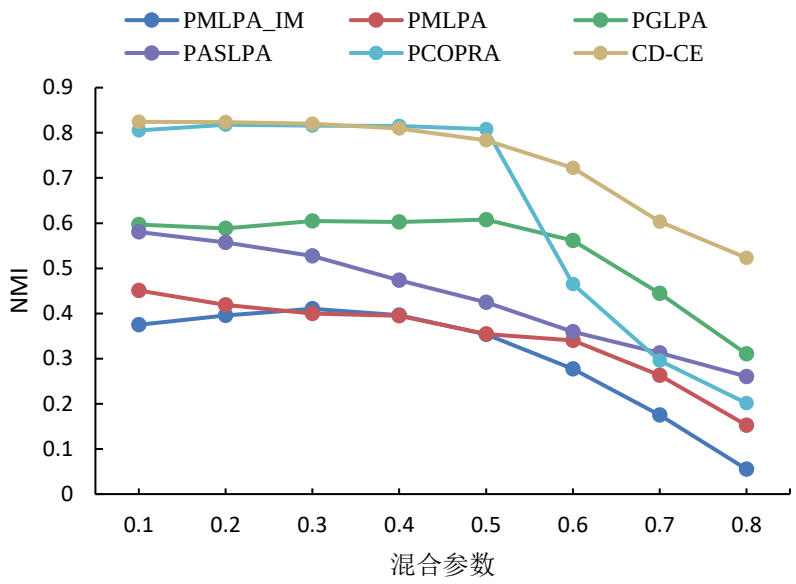

图 4-8　合成网络上的重叠社区的 NMI 值

如图 4-9 所示，当 mu 小于等于 0.5 时，所有算法的 F1-Score 都处于相对较大的值，而当 mu 大于 0.5 时，所有算法的 F1-Score 值都下降的很快。当 mu 小于等于 0.5 时，CD-CE、PCOPRA 和 PGLPA 表现出更高的 F1-Score 值。但当 mu 大于 0.5 时，CD-CE 算法在所有算法中保持最高的 F1-Score，PCOPRA 和 PMLPA 的 F1-Score 值都在向 0 无限接近。当 mu 大于 0.6 时，PGLPA 和 PMLPA_IM 的 F1-Score 值也趋近于 0。

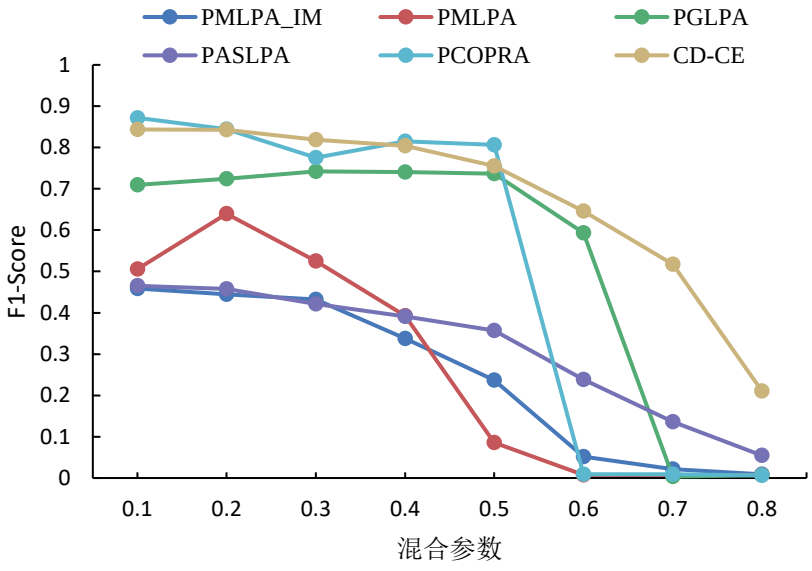

图 4-9　合成网络上的重叠社区的 F1-Score 值

为了测试 CD-CE 算法的可扩展性，通过改变每个 worker 中执行的 CPU 核数来测试核数如何改变 CD-CE 算法的执行时间。图 4-10 显示了 CD-CE 算法在四个数据集上的执行时间，每个 worker 使用 1 核、2 核、3 核和 4 核。在 youtube 数据集上，随着核数的增加，执行时间在明显减少。在 orkut、dblp 和 amazon 数据集上，执行时间也随着核数的增加而趋于减少，尽管使用不同核数的执行时间差异并不明显。因此，CD-CE 算法具有良好的可扩展性。该算法的执行效率可以通过增加核数来提高。

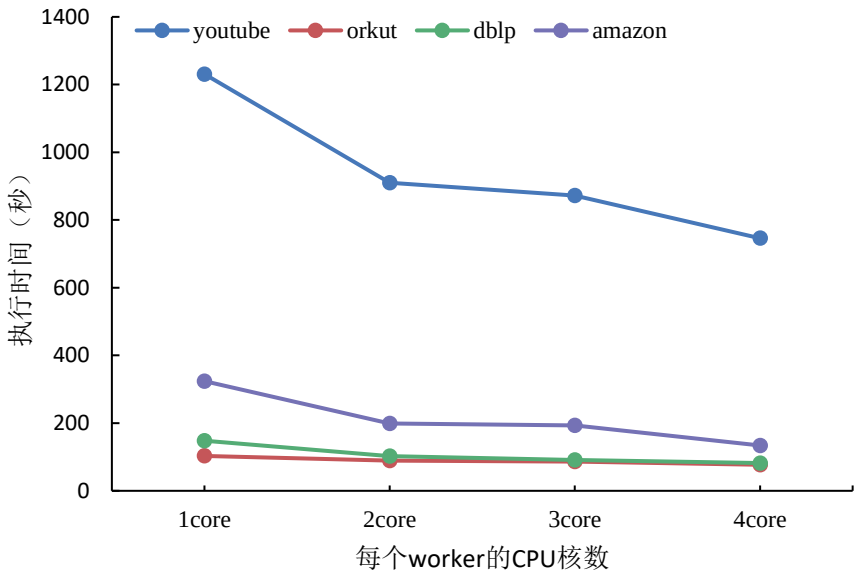

图 4-10　CD-CE 在不同数量的 CPU 核心上的执行时间

图 4-11 显示了六种算法在真实网络和合成网络上的执行时间。很明显，PMLPA 算法在 dblp 中的执行时间是最高的。在 LFR2 中，PMLPA_IM 的执行时间最高。在 orkut 中，PCOPRA 的执行时间最高，其次是 PMLPA_IM 和 PMLPA 算法，而 PASLPA、PGLPA 和 CD-CE 算法的执行时间都偏低。PASLPA 算法在现实世界的网络上显示出最好的性能，其次是 CD-CE 和 PGLPA。而在 LFR2 中，CD-CE 的性能最好，其执行时间分别比 PASLPA、PGLPA、PCOPRA、PMLPA 和 PMLPA_IM 快 386%、418%、77%、273%和 9173%。

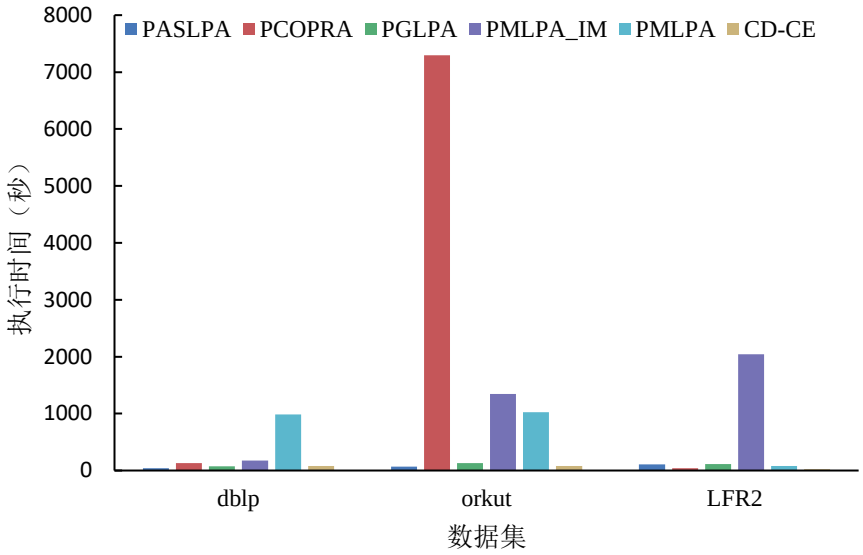

图 4-11　六种算法在真实网络和合成网络上的执行时间

4.6　本章小结

通过结合密度峰值聚类和标签传播算法，在本章提出了 CD-CE 算法。首先，提出了一种新的方法来寻找节点的局部密度，并使用改进的基于信任的距离度量结合密度峰值聚类的思想来计算节点的重要性。然后，确定网络中的边缘节点，为其选择最近的核心节点，并将最近的核心节点的标签作为该边缘节点的标签。最后，为了使节点获得的标签更加准确，为邻域标签和自身标签不一致的节点重新选择标签。

第五章　总结与展望

5.1　总结

在大数据时代背景下，随着信息的海量增长，由节点和边构成的复杂网络的规模也在不断扩大。因而大规模网络上的社区发现研究面临巨大挑战。为了进一步提高 Graphx 执行应用程序的效率，从图分区入手，对 Graphx 图存储进行了优化。因此，本文主要针对图分区和社区发现两部分进行研究，主要工作如下：

（1）针对 Graphx 的图存储优化，本文提出了一个基于程序分析的图分区框架。通过对图分区的研究，提高了 Graphx 处理图数据的效率。它可以根据算法的特点和信息流的方向来选择最合适的分区策略。还设计了 TriPS，并将其应用于三角计数等算法。对于 PageRank 算法，引入了一个改进的基于哈希分区的 PageRank 算法，以减少每次迭代中的 shuffle 次数。本文提出的算法都是在 Graphx 框架上实现的。在六台计算机上的评估表明，采用 TriPS 分区的三角计数算法的执行效率明显优于采用 CRVC、RVC、E1D、E2D 和 DST 作为分区策略的三角计数算法。本文提出的 PageRank 的性能也优于传统的 PageRank 算法，在每个迭代中都能减少一次 shuffle 操作。

（2）本文结合密度峰值聚类和标签传播算法，提出了 CD-CE 算法。该算法分为三个主要部分：核心节点的选择、基于核心和边缘节点的标签传播算法以及标签重新分配。在核心节点的选择部分，使用新提出的局部密度和最小距离公式来计算网络中节点的重要性并选择核心节点。然后确定网络中的边缘节点，为其选择最近的核心节点，并将最近的核心节点的标签作为边缘节点的标签。有标签的节点被传播到没有标签的节点，直到所有节点都获得标签。为了使节点获得的标签更加准确，对邻域标签和自身不一致的节点的标签进行重新选择。实验是在一个由六台机器组成的集群上进行的，分别与非重叠社区发现算法和重叠社区发现算法进行了比较。用于检测非重叠社区和重叠社区准确性的评价标准是 Q_{ov}、NMI 和 F1-Score，用于检测非重叠社区的指标是 ARI 和 ACC。通过使用这些评价标准，本文提出的算法体现出了更高的准确性。此外，在执行时间上也有更高的执行效率。

5.2　展望

本文在大规模复杂网络上的研究还存在着许多不足之处，在本文研究的基础之上，在未来的工作中可以从以下几个方面进行进一步研究：

（1）本文所研究的网络未曾考虑网络中节点和边上的属性信息。仅是基于抽象出的由节点和边组成的网络，对网络本身的拓扑结构进行的研究。这并不一定能应用到实际生产中。因此结合实际应用场景是有必要的。

（2）本文所使用的网络均是静态网络，而现实中的网络都是实时变化的。对于图分区来说，很容易造成分区不均衡，降低算法的执行效率的情况。对于社区发现来说，网络中的社区结构也会随时发生改变。

参考文献

[1] Pattanayak H S, Sangal A L, Verma H K. Community detection in social networks based on fire propagation[J]. Swarm and evolutionary computation, 2019, 44: 31-48.

[2] Lu F, Liu K, Duan Y, et al. Modeling the heterogeneous traffic correlations in urban road systems using traffic-enhanced community detection approach[J]. Physica A: Statistical Mechanics and its Applications, 2018, 501: 227-237.

[3] M'barek M B, Borgi A, Bedhiafi W, et al. Genetic algorithm for community detection in biological networks[J]. Procedia Computer Science, 2018, 126: 195-204.

[4] Newman M E J, Girvan M. Finding and evaluating community structure in networks[J]. Physical review E, 2004, 69(2): 026113.

[5] Karypis G, Kumar V. Multilevelk-way partitioning scheme for irregular graphs[J]. Journal of Parallel and Distributed computing, 1998, 48(1): 96-129.

[6] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs[C]//Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. 2012: 1222-1230.

[7] Tsourakakis C, Gkantsidis C, Radunovic B, et al. Fennel: Streaming graph partitioning for massive scale graphs[C]//Proceedings of the 7th ACM international conference on Web search and data mining. 2014: 333-342.

[8] Liu X, Zhou Y, Guan X, et al. A feasible graph partition framework for parallel computing of big graph[J]. Knowledge-Based Systems, 2017, 134: 228-239.

[9] Gonzalez J E, Low Y, Gu H, et al. Powergraph: Distributed graph-parallel computation on natural graphs[C]//Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12). 2012: 17-30.

[10] Xie C, Yan L, Li W J, et al. Distributed power-law graph computing: Theoretical and empirical analysis[J]. Advances in neural information processing systems, 2014, 27.

[11] 喻斗. 幂律图中基于边优先级的划分方法研究[D].西南财经大学,2021.DOI:10.27412/d.cnki.gxncu.2021.002115.

[12] Zhu X, Chen W, Zheng W, et al. Gemini: A computation-centric distributed graph processing system[C]//OSDI. 2016, 16: 301-316.

[13] Chen R, Shi J, Chen Y, et al. Powerlyra: Differentiated graph computation and partitioning on skewed graphs[J]. ACM Transactions on Parallel Computing (TOPC), 2019, 5(3): 1-39.

[14] Dathathri R, Gill G, Hoang L, et al. Gluon: A communication-optimizing substrate for distributed heterogeneous graph analytics[C]//Proceedings of the 39th ACM SIGPLAN conference on programming language design and implementation. 2018: 752-768.

[15] Slota G M, Root C, Devine K, et al. Scalable, multi-constraint, complex-objective graph partitioning[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(12): 2789-2801.

[16] Hoang L, Dathathri R, Gill G, et al. Cusp: A customizable streaming edge partitioner for distributed graph analytics[J]. ACM SIGOPS Operating Systems Review, 2021, 55(1): 47-60.

[17] Gill G, Dathathri R, Hoang L, et al. A study of partitioning policies for graph analytics on large-scale distributed platforms[J]. Proceedings of the VLDB Endowment, 2018, 12(4): 321-334.

[18] Girvan M, Newman M E J. Community structure in social and biological networks[J]. Proceedings of the national academy of sciences, 2002, 99(12): 7821-7826.

[19] Qiao S, Han N, Gao Y, et al. A fast parallel community discovery model on complex networks through approximate optimization[J]. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(9): 1638-1651.

[20] Ling X, Yang J, Wang D, et al. Fast community detection in large weighted networks using graphx in the cloud[C]//2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2016: 1-8.

[21] Shi T, Ding S, Xu X, et al. A community detection algorithm based on Quasi-Laplacian centrality peaks clustering[J]. Applied Intelligence, 2021, 51(11): 7917-7932.

[22] Yang G, Zheng W, Che C, et al. Graph-based label propagation algorithm for community detection[J]. International Journal of Machine Learning and Cybernetics, 2020, 11(6): 1319-1329.

[23] Tang M, Pan Q, Qian Y, et al. Parallel label propagation algorithm based on weight and random walk[J]. Mathematical Biosciences and Engineering, 2021, 18(2): 1609-1628.

[24] Tang Z, Li C, Tang Y. An efficient method based on label propagation for overlapping community detection[C]//2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD). IEEE, 2021: 168-173.

[25] El Kouni I B, Karoui W, Romdhane L B. Node importance based label propagation algorithm for overlapping community detection in networks[J]. Expert Systems with Applications, 2020, 162: 113020.

[26] 姚越,程晓辉.一种基于节点影响力的重叠社区发现算法[J].科技创新与应用,2023,13(01):37-42.DOI:10.19981/j.CN23-1581/G3.2023.01.009.

[27] 郑文萍,王宁,杨贵.一种基于局部路径信息的重叠社区发现算法[J].计算机科学,2022,49(12):155-162.

[28] 段小虎,曹付元.基于节点局部相似性的两阶段密度峰值重叠社区发现方法[J].计算机科学,2022,49(12):170-177.

[29] Ma T, Yue M, Qu J, et al. PSPLPA: Probability and similarity based parallel label propagation algorithm on spark[J]. Physica A: Statistical Mechanics and its Applications, 2018, 503: 366-378.

[30] Buluç A, Gilbert J R. The Combinatorial BLAS: Design, implementation, and applications[J]. The International Journal of High Performance Computing Applications, 2011, 25(4): 496-509.

[31] Cheng R, Hong J, Kyrola A, et al. Kineograph: taking the pulse of a fast-changing and connected world[C]//Proceedings of the 7th ACM european conference on Computer Systems. 2012: 85-98.

[32] Stutz P, Bernstein A, Cohen W. Signal/collect: graph algorithms for the (semantic) web[C]//The Semantic Web–ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I 9. Springer Berlin Heidelberg, 2010: 764-780.

[33] Çatalyürek Ü V, Aykanat C. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication[C]//Parallel Algorithms for Irregularly Structured Problems: Third International Workshop, IRREGULAR'96 Santa Barbara, CA, USA, August 19–21, 1996 Proceedings 3. Springer Berlin Heidelberg, 1996: 75-86.

[34] Tang J, Zhang J, Yao L, et al. Extraction and mining of an academic social network[C]//International Conference on World Wide Web. ACM, 2008.

[35] Ding Y, Yan S, Zhang Y B, et al. Predicting the attributes of social network users using a graph-based machine learning method[J]. Computer Communications, 2016, 73: 3-11.

[36] Brin S, Page L. The anatomy of a large-scale hypertextual web search engine[J]. Computer networks and ISDN systems, 1998, 30(1-7): 107-117.

[37] Alazawi Z, Abdljabar M B, Altowaijri S, et al. ICDMS: an intelligent cloud based disaster management system for vehicular networks[C]//Communication Technologies for Vehicles: 4th International Workshop, Nets4Cars/Nets4Trains 2012, Vilnius, Lithuania, April 25-27, 2012. Proceedings 4. Springer Berlin Heidelberg, 2012: 40-56.

[38] Tian Y, Mceachin R C, Santos C, et al. SAGA: a subgraph matching tool for biological graphs[J]. Bioinformatics, 2007, 23(2): 232-239.

[39] Oh S, Ha J, Lee K, et al. DegoViz: an interactive visualization tool for a differentially expressed genes heatmap and gene ontology graph[J]. Applied Sciences, 2017, 7(6): 543.

[40] Ding Y. Scientific collaboration and endorsement: Network analysis of coauthorship and citation networks[J]. Journal of informetrics, 2011, 5(1): 187-203.

[41] Leskovec J, Krevl A. SNAP Datasets: Stanford large network dataset collection[J]. 2014.

[42] Demetrescu C, Goldberg A V, Johnson D S. The shortest path problem[C]//Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA. 2006, 74: 29.

[43] Dharavath R, Arora N S. Spark's GraphX-based link prediction for social communication using triangle counting[J]. Social Network Analysis and Mining, 2019, 9: 1-12.

[44] Kunegis J. Konect: the koblenz network collection[C]//Proceedings of the 22nd international conference on world wide web. 2013: 1343-1350.

[45] Jain N, Liao G, Willke T L. Graphbuilder: scalable graph etl framework[C]//First international workshop on graph data management experiences and systems. 2013: 1-6.

[46] Dave A, Jindal A, Li L E, et al. Graphframes: an integrated api for mixing graph and relational queries[C]//Proceedings of the fourth international workshop on graph data management experiences and systems. 2016: 1-8.

[47] Ziegler C N, Lausen G. Analyzing correlation between trust and user similarity in online communities[C]//International Conference on Trust Management. Springer, Berlin, Heidelberg, 2004: 251-265.

[48] Lancichinetti A, Fortunato S, Radicchi F. Benchmark graphs for testing community detection

algorithms[J]. Physical review E, 2008, 78(4): 046110.

[49] Newman M E J. url: http://www-personal. umich. edu/~ mejn/netdata[J]. Last checked March, 2008, 8.

[50] Danon L, Diaz-Guilera A, Duch J, et al. Comparing community structure identification[J]. Journal of statistical mechanics: Theory and experiment, 2005, 2005(09): P09008.

[51] Shen H W, Cheng X Q, Guo J F. Quantifying and identifying the overlapping community structure in networks[J]. Journal of Statistical Mechanics: Theory and Experiment, 2009, 2009(07): P07042.

[52] Yang J, Leskovec J. Overlapping community detection at scale: a nonnegative matrix factorization approach[C]//Proceedings of the sixth ACM international conference on Web search and data mining. 2013: 587-596.

[53] Rand W M. Objective criteria for the evaluation of clustering methods[J]. Journal of the American Statistical association, 1971, 66(336): 846-850.

[54] Huang Z. Clustering large data sets with mixed numeric and categorical values[C]//Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD). 1997: 21-34.

[55] Gregory S. Finding overlapping communities in networks by label propagation[J]. New journal of Physics, 2010, 12(10): 103018.

[56] Zhao W, Martha V, Xu X. PSCAN: a parallel Structural clustering algorithm for big networks in MapReduce[C]//2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2013: 862-869.

[57] Malak M, East R. Spark GraphX in action[M]. Simon and Schuster, 2016.

[58] Roghani H, Bouyer A, Nourani E. PLDLS: a novel parallel label diffusion and label selection-based community detection algorithm based on spark in social networks[J]. Expert Systems with Applications, 2021, 183: 115377.

[59] Sedighpour N, Bagheri A. PASLPA-Overlapping community detection in massive real networks using apache spark[C]//2018 9th International Symposium on Telecommunications (IST). IEEE, 2018: 233-240.

[60] Gregory S. Finding overlapping communities in networks by label propagation[J]. New journal of Physics, 2010, 12(10): 103018.

[61] Zhang Q, Qiu Q, Guo W, et al. A social community detection algorithm based on parallel grey label

propagation[J]. Computer Networks, 2016, 107: 133-143.

[62] Qiu Q, Guo W, Chen Y, et al. Parallel multi-label propagation based on influence model and its application to overlapping community discovery[J]. International Journal on Artificial Intelligence Tools, 2017, 26(03): 1760013.

[63] Li R, Guo W, Guo K, et al. Parallel multi-label propagation for overlapping community detection in large-scale networks[C]//International Workshop on Multi-disciplinary Trends in Artificial Intelligence. Springer, Cham, 2015: 351-362.