

|       |            |
|-------|------------|
| 学校代码  | 10699      |
| 分 类 号 | TP311      |
| 密 级   | 公开         |
| 学 号   | 2012201676 |

题目 大规模数据并行图处理算法与计  
算平台研究

作者 尤立

---

学科、专业 计算机软件与理论

---

指 导 教 师 陈群

---

申请学位日期 2015 年 3 月

---



西北工业大学

# 硕士学位论文

(学位研究生)

题目：大规模数据并行图处理算法  
与计算平台研究

作者：尤立

学科专业：计算机软件与理论

指导教师：陈群

二零一五年三月



Northwestern Polytechnical University

A Dissertation Submitted for the  
Master Degree

Title: Research of parallel graph algorithms over  
large scale data and processing platform

Author            YouLi  
Specialty        Computer Software and Theory  
Supervisor      ChenQun

February 2015



## 摘 要

近年来,大数据已经成为各大公司追逐的热点领域。随着大数据时代的到来,大量、多样、快速的数据给业界带来诸多挑战,同时也带来了巨大的价值和机遇。很多热门研究领域,如社交网络分析、Web 文档聚类、实体识别、基因表达分析等,都必须对海量数据进行分析 and 挖掘。这些应用中通常都包含较复杂的网络结构,实际生产过程中,人们通常使用图来抽象系统中的各个实体之间的关系结构,并通过对抽象的图挖掘从数据中提取有价值的知识。在这些应用中关于图的完全图和近似完全图挖掘已经成为不可或缺的组成部分。

完全图和近似完全图是经典的 NP Complete 问题,算法时空复杂度都非常高。人们一直以来致力于找出此类问题的高效解决方法。过去的研究已经提出了一些在较小数据集上表现良好的算法。但是在大数据环境下,单机算法受制于内存和磁盘以及 CPU 的限制往往不能够满足实际生产需求。并行化成为一种必然的可行趋势,因此研究集群环境中大规模数据的并行图处理算法尤为重要。本文在充分了解现有的算法基础上,提出了一种基于图分割的完全图和近似完全图枚举算法(Binary),并通过在多种实际数据和生成数据上的单机及并行实验与经典算法(BK, Bron-Kerbosch)作对比,验证了 Binary 算法的可行性及高效性。考虑到两个算法各自的优劣的基础之上,结合两者的优势提出了一种 BK 辅助 Binary 的 Hybrid 算法,在各种数据尤其是蛋白质交互数据上有更优的表现。

在并行环境中,本文选择利用当今流行的大数据计算框架 MapReduce 的开源实现 Hadoop 作为并行算法的运行平台。针对并行环境中完全图和近似完全图枚举过程中出现的负载不均问题,本文提出了一种负载均衡的算法。实验表明此负载均衡算法得到了较好的均衡性和扩展性。

**关键词:** 完全图, 近似完全图, 并行算法, 负载均衡, Hadoop





## Abstract

Big data has become a hot area for various companies in recent years. Though the three characters volume, variety, velocity raise great challenges, value and opportunities are also derived. Many popular research areas, such as social network analysis, web document clustering, entity recognition and gene expression analysis need to mining and analysis massive data. Typically, these applications deal with complex network structure. Graph is the data structure people used to abstracting relations between entities and also mining value from. Clique and K-Plex mining have become an indispensable part of these applications.

Clique and K-Plex mining are typical NP Complete problem known with high time and space complexity. Great effort has been committed to find efficient solution and several researches have shown good performance on small datasets. But under the circumstance of big data, these solutions are subjected to limit memory, disk and CPU in single machine. Parallelization become a feasible trend, so it's particularly important to study parallel algorithms in large-scale cluster. In this paper, we fully study the existing algorithms and propose a maximal clique enumeration and maximal k-plex enumeration based on graph partitioning (Binary). Then we conduct both single and parallel experiments on variety of real data and synthetic graphs, our algorithm shows better performance and smaller search tree space than classical algorithm (BK, Bron-Kerbosch). Furthermore, we realize both algorithms have pros and cons. Combining benefits together, we present a BK-aid Binary hybrid algorithm (Hybrid). Hybrid performs better than both BK and Binary on a variety of data especially protein interaction data.

In parallel environment, we choose Hadoop which is an open source implementation of popular big data computing framework MapReduce. We propose a load balancing solution for both problem's skew in parallel environment. Experiments shown that the load balancing solutions has achieved good balancing and scalability.

**Keywords:** Clique, K-Plex, Parallel algorithm, Load balance, Hadoop



# 目录

|                                    |     |
|------------------------------------|-----|
| 摘 要 .....                          | I   |
| Abstract .....                     | III |
| 目录 .....                           | III |
| 1 绪论 .....                         | 1   |
| 1.1 研究背景 .....                     | 1   |
| 1.2 选题意义 .....                     | 1   |
| 1.3 研究现状 .....                     | 3   |
| 1.3.1 完全图单机算法研究现状 .....            | 3   |
| 1.3.2 完全图并行算法研究现状 .....            | 4   |
| 1.3.3 近似完全图单机算法研究现状 .....          | 4   |
| 1.3.4 近似完全图并行算法研究现状 .....          | 5   |
| 1.4 论文的内容及组织 .....                 | 6   |
| 1.4.1 主要研究内容及成果 .....              | 6   |
| 1.4.2 论文组织结构 .....                 | 6   |
| 2 相关理论及技术 .....                    | 7   |
| 2.1 图论相关概念 .....                   | 7   |
| 2.2 完全图和近似完全图 .....                | 8   |
| 2.2.2 完全图相关概念 .....                | 8   |
| 2.2.3 近似完全图相关概念 .....              | 8   |
| 2.3 并行计算模型 MapReduce .....         | 9   |
| 2.4 开源分布式计算平台 Hadoop .....         | 11  |
| 2.5 Hadoop 中的负载均衡 .....            | 12  |
| 2.6 本章总结 .....                     | 13  |
| 3 完全图和近似完全图枚举算法 .....              | 15  |
| 3.1 完全图枚举算法 .....                  | 15  |
| 3.1.1 极大完全图 BK 算法 .....            | 15  |
| 3.1.2 极大完全图 Binary 算法 .....        | 18  |
| 3.1.3 极大完全图 Hybrid 算法 .....        | 21  |
| 3.2 近似完全图枚举算法 .....                | 23  |
| 3.2.1 基于 BK 的 K-Plex 算法 Pump ..... | 23  |
| 3.2.2 基于 Binary 的 K-Plex 算法 .....  | 25  |
| 3.3 本章总结 .....                     | 27  |
| 4 Hadoop 平台下算法的并行化 .....           | 29  |
| 4.1 输入图数据 .....                    | 29  |
| 4.2 初始任务分配 .....                   | 30  |
| 4.3 负载倾斜问题 .....                   | 32  |
| 4.4 完全图和近似完全图枚举的负载均衡 .....         | 32  |
| 4.5 本章总结 .....                     | 36  |
| 5 实验部分 .....                       | 37  |
| 5.1 实验条件及数据 .....                  | 37  |

|                          |    |
|--------------------------|----|
| 5.1.1 实验条件.....          | 37 |
| 5.1.2 实验数据.....          | 37 |
| 5.2 单机算法实验及分析 .....      | 37 |
| 5.2.1 极大完全图枚举.....       | 38 |
| 5.2.2 极大 K-Plex 枚举 ..... | 40 |
| 5.2.3 分裂点选择.....         | 41 |
| 5.3 并行算法实验及分析 .....      | 42 |
| 5.3.1 并行极大完全图枚举.....     | 42 |
| 5.3.2 并行极大近似完全图枚举.....   | 44 |
| 5.3.3 并行算法加速比.....       | 45 |
| 5.4 本章总结 .....           | 45 |
| 6 总结与展望 .....            | 47 |
| 6.1 本文研究总结 .....         | 47 |
| 6.2 课题研究展望 .....         | 47 |
| 参考文献 .....               | 49 |
| 致谢 .....                 | 53 |

## 1 绪论

### 1.1 研究背景

随着互联网尤其是移动互联网的快速发展, 社会化交际网络的快速普及, 物联网、商业销售、Web 语义分析、生物网络信息等相关应用的丰富, 云计算相关技术的快速发展, 更多的设备被添加到网络中, 数据资源呈指数增长。正如人们所说: 大数据时代到来了。在大数据时代, 对于企业来讲最重要的是获得和使用数据, 能否从现有的海量数据中快速、高效、准确地分析提取有用信息已经成为企业能否在新环境下取得成功的关键因素。早在 2010CNNIC 统计时, 仅中国的网页规模就已经达到 600 亿, 且还在以每年 78.6% 的比率增长; Web 搜索领域 Google 索引了全球超过 500 亿的网页, 每天有 20PB 数据要处理; 同时社交网络的后来居上, Facebook 已经超过 7 亿用户[39]; 在环境和气象领域, 需要从数以万计的传感器中接受大量数据以进行监控和预测。正是有如此庞大和迫切的需求, 快速高效的海量数据处理日益受到学术界和科技界的关注。

在各种各样的海量数据应用中普遍存在网络连接结构, 如 Web 中网页的链接关系, 社交网络中人与人之间的好友关系, 基因表达数据中各个基因之间的协同表达关系等。而图正是数学上描述各个实体之间的网络关系的一种经典结构, 如此总总应的用都可以抽象成关于图的计算。

图论是数学的一个重要分支, 早在 1736 年数学家欧拉关于柯尼斯堡七桥问题的论著中就有关于图论的研究记录, 这个问题后来被推广为著名的欧拉回路问题。经过二百多年的发展, 图论已经成为一个独立的数学研究分支。图论中的问题主要可以概括为子图问题、染色问题、路径问题、网络流与匹配问题以及覆盖问题。这些问题里各自出现了许多经典的代表, 如哈密顿回路、最大团、四色问题、斯坦纳树、最短路径、中国邮递员问题、最小覆盖集等, 也出现了许多如戴克斯特拉算法、克鲁斯卡尔算法、拓扑排序等等经典算法。然而经过研究证明, 很多图相关的问题都是 NP 或者 NP Complete 问题, 算法有很高的时间和空间复杂度。在过去的现实需求中数据量较小, 传统算法可以具有较好的实用性。但在大数据时代, 数据的采集、获取变得非常方便, 数据中所蕴含的科学和商业价值推动着对于海量数据的图处理需求迅速提高。传统的单机算法受制于内存和磁盘的限制, 尽管不断优化, 还是不能够在有意义的时间内完成处理。同时现有的多机并行算法通常利用冗余来保证计算的完整和正确, 很难达到较高的并行度, 或者在计算过程中带来严重的数据膨胀问题。

### 1.2 选题意义

随着图在社交网络、生物信息、实体识别等领域广泛应用, 业界对高效的图处理算法有着迫切的现实需求。完全图和近似完全图常常用于进行分类和聚类分析, 例如在社交网络中, 通过检测完全图和近似完全图, 从而找到关系十分紧密的群体, 群体之间往

往具有相同的兴趣爱好或者消费习惯，因此在个性化服务和投放广告方面能做的更加地准确有效，从而增加用户的粘性，提升网站的流量和商业价值都；在生物信息领域，不同基因之间的协同表达是一个重要的研究问题，常常作为新型药物研制的突破口，是生物工程中常用的技术，完全图和近似完全图的检测则是分析基因协同表达中的重要技术，这些技术的应用能够有效缩短新药的研制时间和成本。

在大规模数据的背景下，TB 级甚至 PB 级的大规模数据是比较常见的情况，这些问题难以在单机上进行计算，并行算法成为必然选择。并行算法的过程中经常会出现数据不均衡或者计算不均衡的情况，严重制约平台的吞吐量和计算效率，负载均衡已经成为并行和分布式计算平台研究中最重要的一面。

尽管对于完全图与近似完全图的枚举、并行环境的负载均衡等问题现在已经有许多优秀的算法和方案[3][12][14][16][18][20][33]，然而都存在性能瓶颈和难以适用情况，又或者是针对某些具有特定特征的数据定制，没有较好的通用性。文献[1]是 Bron 和 Kerbosch 完全图枚举的经典算法（BK），在 BK 的基础上发展、衍生了一系列算法，文献[3][5]通过实际应用显示 BK 算法的效率要普遍好于现有的其他算法。然而在本文的研究过程中发现 BK 算法在许多数据上依然存在搜索空间大，速度慢等等问题。文献[2][4]等并行完全图算法虽然一定程度上缓解了单机计算能力的不足，但存在中间数据过度膨胀，冗余状态量大，需要通过大量的机器数目来存储和计算，资源利用效率低。另外现有的分布算法并行程度低，且当出现负载不均衡的情况，如社交网络数据中少数名人的好友关系非常大，而大部分普通用户关系简单，这种差异会导致系统少数机器有性能瓶颈从而影响整体执行效率，降低系统吞吐量。现有的近似完全图算法一般是借鉴完全图的解决方案稍作调整，但是目前还没有很好的并行算法。已有的近似完全图并行算法[6][7]是在特定的限制环境下的特殊解决方案。同时在近似完全图中同样也存在和完全图枚举过程中的负载不均衡问题。并行完全图和近似完全图枚举问题中所共有的负载不均衡的问题通常有两种解决思路：根据特定问题（本文中即完全图和近似完全图枚举）设计方案，在各个机器节点间互相分担高负载节点的任务量；对计算平台做相应修改，使计算平台本身可以达到自动均衡负载的功能。第一种方案具有简单、易实现的优势，但是需要设计者对输入数据特征和特定算法有较好的了解，而且这样的算法并没有通用性，需要为每个算法重新设计均衡策略；第二种方案，从平台上解决了负载均衡的问题，可以有较好的通用性，只要任务是可以切分的，系统可以自动去均衡各个机器节点的负载，而无需用户介入，缺点是实现比较复杂[10][37][38]。

完全图枚举和近似完全图枚举都是经典的 NP Complete 问题一个问题的有效解决可以对其他 NP Complete 问题有借鉴意义，同时本文提出的并行完全图和近似完全图的负载均衡方案可以使得完全图和近似完全图枚举能够取得较好的并行性从而能够使得算法获得较好的实用性。综上所述，本文的选题具有较大的研究意义和现实意义。

### 1.3 研究现状

本文主要涉及图论中的两个典型问题极大完全图枚举 (Maximal Clique Enumeration, MCE) 和极大近似完全图枚举 (Maximal Qusi-Clique Enumeration)。这两个问题自提出以来就已被广泛研究, 各有单机和并行算法两类算法。下面将从这几方面介绍相关问题的研究现状。

#### 1.3.1 完全图单机算法研究现状

完全图中的所有节点都与除自身外的其他节点相邻。在实际应用过程中尤其是计算科学相关领域中最有价值的是从杂乱无章的大量数据图中找出所有的极大完全图。这个问题也因 Luce 和 Perry 在社交网络分析中使用 Clique 表示团体中每一个人都认识其他所有人也成为极大完全图枚举 (MCE)。

抽象的图论算法中有多种极大完全图枚举算法, 以及一些与之相关的数学性质。这些算法主要分为三类:

第一类是以 Bron-Kerbosch 为典型代表的基于回溯搜索并应用剪枝算法降低搜索空间。这类算法[1][15]利用完全图必然是邻接点子集的特征, 算法每次从候选点集中选取一个节点与现有的节点构成完全图, 直到候选节点中不再有点可以与现有结果组成更大的完全图, 最后再递归回溯搜索。

第二类算法[8][9][11][13]使用反向搜索策略, 这类算法的主要特征是可以较容易地得到算法关于结果集中完全图个数的复杂度上界。

第三类算法是基于“组装”的算法[21][22], 将简单的子图自动组装拼接成更大的单元, 最终形成极大完全图。该算法首先将图分解, 分解过程中如若产生完全图则直接输出, 其他的子图按照特定的方式组装若能组装成完全图则输出。

另外还有如 Thang Nguyen Bui 等提出的基于遗传编程算法的完全图检测[17]。算法将网络连接关系图用树的形式表示, 完全图即从根到叶子节点的路径上所涉及到的点集, 找出一个点的邻接节点非常方便。根据完全图的性质结合遗传编程的特点可以较快的剪去许多不符合条件的节点, 减少候选节点数量提高搜索速度。同时由于图以树的形式表示可以借鉴许多在树上研究得较为成熟的技术。James Cheng 等根据大规模图数据中完全图检测的前提下提出一种优化的存储结构  $H^*--graph$ [19], 该结构定义图的核心部分及核心部分所邻接的节点。将其他节点数据存在外存磁盘中, 可以有效的限制大规模数据的情况下单机完全图检测过程中对内存的消耗。同时  $H^*--graph$  只保存了数据图中的核心部分, 数据的更新代价大幅降低, 避免了传统算法中数据更新时需要重新计算的缺点。

在图论领域有不少关于完全图的性质可以应用到实际的完全图检测算法中。Turon 指出, 如果一个图有足够多的边, 那么它一定包含一个最大的完全图; 若一个图中包含  $\lfloor n/2 \rfloor * \lfloor n/2 \rfloor$  条边, 则必然包含一个三个节点的完全图。Moon 和 Moser 指出, 在一个包含  $3n$  个节点的图中, 至多包含  $3^n$  个最大完全图。拉姆齐理论指出每个图或者其补图

包含一个至少有对数个节点的完全图。

### 1.3.2 完全图并行算法研究现状

随着近十几年互联网、生物信息学、社交网络、移动互联网等众多新兴技术的迅速发展,数据的规模变得越来越大。为了适应变化的需求,原本在单机环境中的完全图检测算法也在近些年被人们扩展到并行环境中。这些并行完全图检测算法[23][24][25][26][27]的基本思想都是将图或者搜索空间切分为独立的、较小的子图或子空间,然后对于这些切分出来的子图或子空间将它们分散到多个机器节点上各自进行计算,以提高整体的计算效率。

Y. Zhang 等人首先提出了并行 MCE 算法 pCLique [44], 这个算法是对 Kose 等工作[45]的一个扩展,也成为 KOSE。实际上 KOSE 在思想上和 BK 算法相同,不过与 BK 深度优先搜索的方式不同, KOSE 使用的是广度优先遍历。这种方法可以和经典的关联挖掘算法 Apriori 一样可以通过大小为  $K-1$  的完全图生成大小为  $K$  的完全图,生成的 Clique 可以按非递减的次序出现。然而 KOSE 的广度优先使得它变得内存敏感,尽管 pCLique 使用了位图向量来操作公共邻接点,但还是需要大量的内存消耗。

S Szabo 提出一种基于分解邻接矩阵的并行完全图检测算法[29],该算法充分利用完全图的性质将邻接矩阵不断切分的数据单元,并将其分散到不同的节点上计算。

另外一种并行 MCE 算法 Peamc 是 Du 等人在[46]中提出的。由于 Peamc 是一种基于串行的深度优先搜索, Peamc 中没有 pCLique 中出现的内存消耗的负担。Peamc 使用了一种简单的并行策略,将节点分为多个不想交的独立集合发散到各个计算单元上并行计算,但并没有很好的均衡策略。

Matthew C. Schmidt 等人提出一种基于多核并行的完全图检测算法[28]。该算法使用共享内存来存储全局信息,每个核负责处理部分图数据,各个核将处理出来的结果汇总合并成为最终结果。此算法可以处理较大数据量的图且可以通过核数量的扩展有接近线性的并行性,但是对系统的硬件性能有很高的要求。

之前的算法大多运行在多核机器或者 MPI 平台上,自从 MapReduce 计算模型的开源平台 Hadoop 兴起后,其良好的扩展性、稳定性,适应于异构集群、普通计算节点的特性带来了一波新的浪潮。Shengqi Yang 等首先提出了使用 MapReduce 模型在 Hadoop 平台上进行完全图检测的并行算法[4]。该算法首先提取数据图中每个节点的两跳邻接点(相邻节点以及相邻节点的相邻节点),并将其发往不同的计算节点进行计算,各个计算节点上使用传统单机算法进行检测。

### 1.3.3 近似完全图单机算法研究现状

近似完全图相对完全图而言其对图的特征要求相对较低,因此在现实生产环境中有着更加广泛的应用。对于近似完全图的研究一般是完全图的一种变形,一遍的完全图算



法都可以稍加修改，减弱限制条件使之成为近似完全图检测的算法[34]。近年来，由于社交网络如 Facebook、Twitter 等推动，近似完全图在这种场景下对于公司的广告投放、社区发现有着重要的指导依据，文献[30][31]对此有详细阐述。近似完全图在实际应用中一般都有个条件限制（结果集大小超过一定限值或者最多与  $k$  个顶点不相邻即  $K$ -Plex[34]），由于这些条件有确确实实的意义，如低于 4 个人以下的近似完全图基本没有实用价值，因此本文也采用这样的限制条件。现在已经有不少研究关注近似完全图的挖掘算法，除去对完全图的 BK 算法做相应改进的方法外，有下面这些典型的算法和优化方案：

Guimei Liu 等提出了使用度数过滤数据图中不可能满足近似完全图条件的节点，同时提出了近似完全图挖掘中的多种优化方案，包括关于图直径、设定最小节点数目阈值、子图中近似完全图能够添加的节点上下界的优化等等[32]。

James Abello 等提出了一种先期在外存中使用广度优先搜索，限制输入子图的大小，使得原本由于数据量巨大而不能完全放到内存中图数据可以在有限的内存条件下完成近似完全图检测的工作。算法在对内存中的节点的边进行搜索，同时应用一些剪枝条件和优化策略，使得处理大规模数据成为可能。算法还利用随机适应性贪心搜索的方法通过抽样每次选取可以放入内存的最小子图进行计算[33]。

与完全图检测相比而言，近似完全图由于其限制条件较低，其搜索复杂度、搜索空间以及结果数量都较完全图更大。在大数据环境下，单机算法面临着更加严重的性能问题。

#### 1.3.4 近似完全图并行算法研究现状

近似完全图的并行算法目前还不是很多，主要有两种。

第一种是 Bin Wu 等多核并行环境下并行算法优化。整体上该算法同时使用经典算法的主体结构，每个搜索树节点包含三个集合：Res, Cand, Not，分别表示已经形成  $K$ -Plex 的节点集，可扩展的候选节点集，之前已经扩展过不再需要扩展的节点集。文章中提出两种优化方案：若  $G$  是一个  $K$ -Plex，如果  $G$  中节点数目大于  $(2k-d)$ ，则图  $G$  的直径小于等于  $d$ ；若极大  $K$ -Plex 节点数目大于  $m$ ，如果包含节点  $v$ ，则  $v$  必然在  $(2k-m)$  跳之内[35]。

第二种是 Arash Khosraviani 等提出的在 MapReduce 并行框架中做近似完全图检测，这也是首次提出在 MapReduce 平台中进行相关工作。算法将数据图分解为各自独立的子图，使用 MapReduce 框架将数据发散到各个机器节点上各自做检测，这样可使得算法可以获得较好的扩展性。算法在分解数据图的过程中使用了数学特性限定  $\gamma$ -quasi clique 的  $\gamma \geq 0.5$ ，这样对于所有的搜索工作只需要在两跳数据集里面进行搜索即可。这样的限制使得算法不能够有效检测到  $\gamma$  较小或者节点数目较少的近似完全图，但是由于这些不能检测到的图基本没有使用价值，因此该优化假设是可以利用的[36]。

现有的并行近似完全图枚举算法首先没有从单机算法上将计算时间减下来，同时还没有能够较全面的考虑并行环境中的负载不均问题。本文从这两方面都做了优化工作。

## 1.4 论文的内容及组织

### 1.4.1 主要研究内容及成果

本文主要关注大规模数据环境下并行图处理算法的研究以及并行计算平台中的负载均衡问题。主要包括完全图枚举、近似完全图枚举和在 Hadoop 上算法的负载均衡问题，期望能够为并行环境中高效解决图处理算法和良好的系统均衡提供思路和经验。

本文分析了现有的经典完全图和近似完全图枚举算法的优劣，提出了一种新的完全图枚举和近似完全图枚举切分算法 Binary。通过实验证明新算法在大多数数据情况下其搜索空间以及搜索效率都优于现有算法。

在分析 Binary 和经典算法 BK 各自优势的基础上结合两者的特征将两种算法结合，提出一种 BK 辅助 Binary 的 Hybrid 算法，该算法兼具 Binary 和 BK 的优势。经验证 Hybrid 算法在所有实验数据集上都有更优的表现。契合当前大规模数据的处理需求，结合之前提出的单机算法，本文提出了基于并行计算平台 Hadoop 的 Binary 的并行算法。通过实验证明了并行 Binary 算法的并行性和高效性。

最后由于在并行计算平台 Hadoop 上实现 BK 和 Binary 的并行图算法都会涉及到负载不均问题，本文从图算法本身出发提出了负载均衡方案。

### 1.4.2 论文组织结构

本文的组织结构如下：

第一章是绪论部分，从整体上介绍本文的研究背景，主要介绍了完全图枚举、近似完全图枚举和负载均衡三个问题的研究现状，最后简要介绍了本文的研究内容和研究成果。

第二章是相关理论及技术介绍，包含完全图、近似完全图、负载均衡相关概念，MapReduce 并行计算框架和其开源实现 Hadoop。

第三章对完全图和近似完全图枚举算法进行了深入分析，选取最具代表性的经典算法做为对比实验。分析了现有算法的优缺点。提出了一种新的完全图和近似完全图算法。并结合两种算法各自的优劣提出一种可以充分发挥两者优势的 Hybrid 算法。

第四章在第三章的基础之上论述了这几种算法的并行实现。分析了算法并行化实现过程中发现的负载均衡问题，提出了针对完全图和近似完全图算法的负载均衡方案。

第五章是论文的实验部分。本章选取了一系列真实数据和生成数据通过多种维度的实验验证了本文提出的算法的高效性。同时通过实验验证了负载均衡方案的有效性。

第六章是对本文工作的总结以及未来工作的展望，总结全文的主要内容以及今后可以改进的地方及提升的空间。

## 2 相关理论及技术

本文的主要研究对象是图论中经典的两个问题——完全图和近似完全图以及 Hadoop 系统的负载均衡。两个图算法中研究者们主要关注的方面是如何从原始数据图中找出所有的且最大的完全图或近似完全图，也就是通常所说的极大完全图枚举 (Maximal Clique Enumeration, MCE) 和最大近似完全图枚举 (Maximal Qusi-Clique Enumeration, 也成为 K-Plex)。本章将本文涉及到的如图论相关概念、完全图、近似完全图、并行计算模型 MapReduce、开源平台 Hadoop 以及负载均衡等相关技术作简要介绍。

### 2.1 图论相关概念

作为数学的一个重要研究领域，图论自 1736 年欧拉在著作中首次研究柯尼斯堡七桥问题开始经过上百年发展现在已经形成了超图理论、拟阵理论、拓扑图论和代数图论等分支，并广泛应用于计算机科学、心理学、商业挖掘、运筹学等方面。

图论的主要研究对象是图。图是由顶点的有穷非空集合和顶点之间的边的集合组成，通常表示这样一个二元组： $G = \langle V, E \rangle$ ，其中， $G$  表示一个图， $V$  是图  $G$  中顶点的非空有限集合， $E$  是图  $G$  中边的可空有限集合。图中的边  $e$  是关于节点的一个二元组  $\langle a, b \rangle$  ( $a, b \in V$ )，边的两个元素  $a$  和  $b$  之间可以是有序的也可以是无序的。当边的元组之间无序时将边称之为无向边，相似地边的元组间有序时称边为有向边。由有无向边组成的图为无向图，由有向边组成的图称之为有向图。本文根据实际的应用需求以及普适实用的研究意义将研究对象限定为无向图，如无特别说明在正文中所提到的相关图都是无向图。下图 2-1 中(a)表示一个有向图，(b)表示一个无向图。

图论中两点之间距离并不是指从一个点到达另一个点的路径中边的权值和最小值，而是指的图的直径是指图中任意两个顶点之间可以到达的路径中所经过的最少节点个数。图的直径值图中任意两点之间的距离的最大值，如图 2-1 (b)中图的直径是 1。

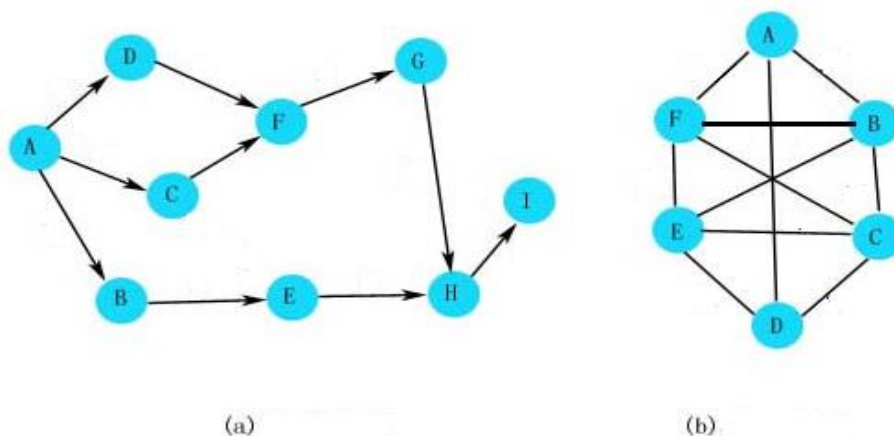


图 2-1 有向图和无向图

在有向图中，以节点  $V$  为始点的边个数称为  $V$  的出度，以节点  $V$  为终点的边个数称

为 $V$ 的入度，他们的和称为 $V$ 的度。无向图中，与节点 $V$ 相邻的边个数称为 $V$ 的度，记为 $\deg(V)$ 。给定两个图 $G=\langle V, E \rangle$ ， $G'=\langle V', E' \rangle$ ，若 $V' \subseteq V$ 且 $E' \subseteq E$ ，则称图 $G'$ 是图 $G$ 的子图。

## 2.2 完全图和近似完全图

本文的主要研究对象完全图和近似完全图都是图的一种稠密子图，下面将介绍完全图和近似完全图的相关定义和性质。

### 2.2.2 完全图相关概念

完全图指图中任意两个节点之间都有边相连，在有 $n$ 个顶点的完全图 $K_n$ 中应有 $n(n-1)/2$ 条边，也称这个图是完全的。完全图是它本身的团（Clique）。在上图 2-1 中 B、C、E、F 四个点组成的子图就是一个完全图。图 2-2 直观上展示了 $K_1-K_8$ 的完全图。

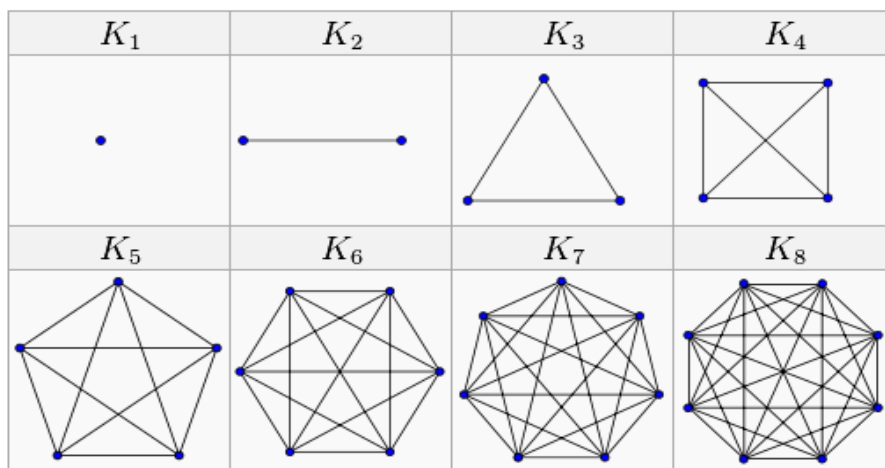


图 2-2 完全图

关于完全图有两个相似的容易混淆的概念，极大团（Maximal Clique）和最大团（Maximum Clique）。一个完全图能称为极大团指不能够通过从这个图的邻接点中扩展出一个新的更大的完全图，也就是说这个团不是其他任意一个完全图的子图。最大团指一个图中大小最大的一个完全图，也就是说团 $G$ 的顶点数目大小 $\omega(G)$ 在这个图的所有团中是最大的。指定一个顶点 $v$ ，包含顶点 $v$ 的极大完全图可以有很多个，但是包含顶点 $v$ 的最大完全图只要找到最大一个即可。在图 2-1 中对于点 C 来说，C、D、E 是三个点的一个极大完全图，但是最大完全图是 C、B、E、F。

### 2.2.3 近似完全图相关概念

近似完全图也是一种稠密图，但是可以容许图中可以有部分点之间不相邻。由于完全图对图的连接结构要求非常严格，现实场景中数据很难达到完全图的标准。近似完全图由于具有相似的特性但是要求不那么严格，在现实环境中的应用会更加普遍。一个近似完全图 $G_\gamma$ 是图 $G$ 的一个子集， $G(G_\gamma)$ 包含至少 $\lfloor \gamma * q(q-1)/2 \rfloor$ 条边，其中

$q = G_\gamma$ ,  $\gamma \in (0,1]$ 。我们称 quasi-clique 为  $\gamma$ -quasi-clique，其中参数  $\gamma$  用于表示近似完全图的疏密程度， $\gamma$  越小，近似完全图中平均每个节点的邻接点个数越少，也就是说，近似完全图更加稀疏；反之，平均每个节点邻接的点越多，近似完全图更加稠密。特别地，当  $\gamma=1$  时，即上一节中提到的完全图。

由于近似完全图要求较松，粗糙的定义给实际挖掘工作带来了较大的麻烦。在实际生产应用中我们通常对  $\gamma$  和  $q$  的值进行限制。若节点数目  $q$  过少，挖掘出来的近似完全图应用价值不高，通常会限制  $\gamma \geq 0.5$ 。鉴于多数现实应用意义本文中的算法同样使用了这个限制条件。

图论中还有另外一个度量图的密集程度的方式，即图的直径  $diam(G)$ 。文献[40]中指出  $diam(G)$  与  $\gamma$  之间的关系，具体如下面公式 2-1 所示，其中  $n$  为图  $G$  中节点的个数。由公式 2-1 可知，为了满足  $\gamma \geq 0.5$  的条件，只需保证  $diam(G)=2$  即可。因此，在近似完全图的搜索过程中，可以只使用两跳数据集，而无需全部图数据，这从一定程度上减少了搜索过程的时空代价。

$$diam(G) = \begin{cases} =1 & \text{若 } 1 \geq \gamma > \frac{n-2}{n-1} \\ =2 & \text{若 } \frac{n-2}{n-1} \geq \gamma \geq \frac{1}{2} \\ \leq 3 \left\lfloor \frac{n}{\gamma(n-1)+1} \right\rfloor - 3 & \text{若 } \frac{1}{2} > \gamma \geq \frac{2}{n-1} \text{ 且 } n \bmod(\gamma(n-1)+1) = 0 \\ \leq 3 \left\lfloor \frac{n}{\gamma(n-1)+1} \right\rfloor - 2 & \text{若 } \frac{1}{2} > \gamma \geq \frac{2}{n-1} \text{ 且 } n \bmod(\gamma(n-1)+1) = 1 \\ \leq 3 \left\lfloor \frac{n}{\gamma(n-1)+1} \right\rfloor - 1 & \text{若 } \frac{1}{2} > \gamma \geq \frac{2}{n-1} \text{ 且 } n \bmod(\gamma(n-1)+1) \geq 2 \\ \leq n-1 & \text{若 } \gamma = \frac{1}{n-1} \end{cases} \quad (2-1)$$

通常对于近似完全图，还有一种较为简洁的表示方法，K-Plex。一个近似完全图 K-Plex 定义为包含  $N$  个节点的图  $G$  中，每个节点最多与  $(N-K)$  个节点不相邻（节点认为与自身不相邻）。当  $K=1$  时，所有节点只与自身不相邻，因此就是完全图。实际上，完全图是近似完全图的一种特殊情况。

### 2.3 并行计算模型 MapReduce

随着计算规模的不断扩大涌现了许多优秀的并行计算模型，MapReduce、BSP(Bulk Synchronous Parallel Computing)、LogP 等。其中 MapReduce 模型以其简洁性和通用性获得了长足的发展。MapReduce[41]是 Google 的 J Dean 等提出的一种函数式编程模型，可以使得程序员方便地使用简单的函数实现完成复杂的分布式并行计算。其设计主要包括两个模块，分布式存储 DFS 和 MapReduce 计算框架。

MapReduce 框架将计算过程分为 Map 和 Reduce 两部分。Map 过程称为映射，映射

过程就是对逻辑上一致的列表中每个独立元素进行指定的操作，每个元素的操作是独立的，输出使用新的列表保存操作结果。由于操作并没有改变输入列表，Map 是可以高度并行的，这对并行计算领域中高性能和高并行度的需求非常有用。Reduce 过程称为化简，化简过程是对输入列表中的元素进行适当地合并，归约出最终结果[42]。

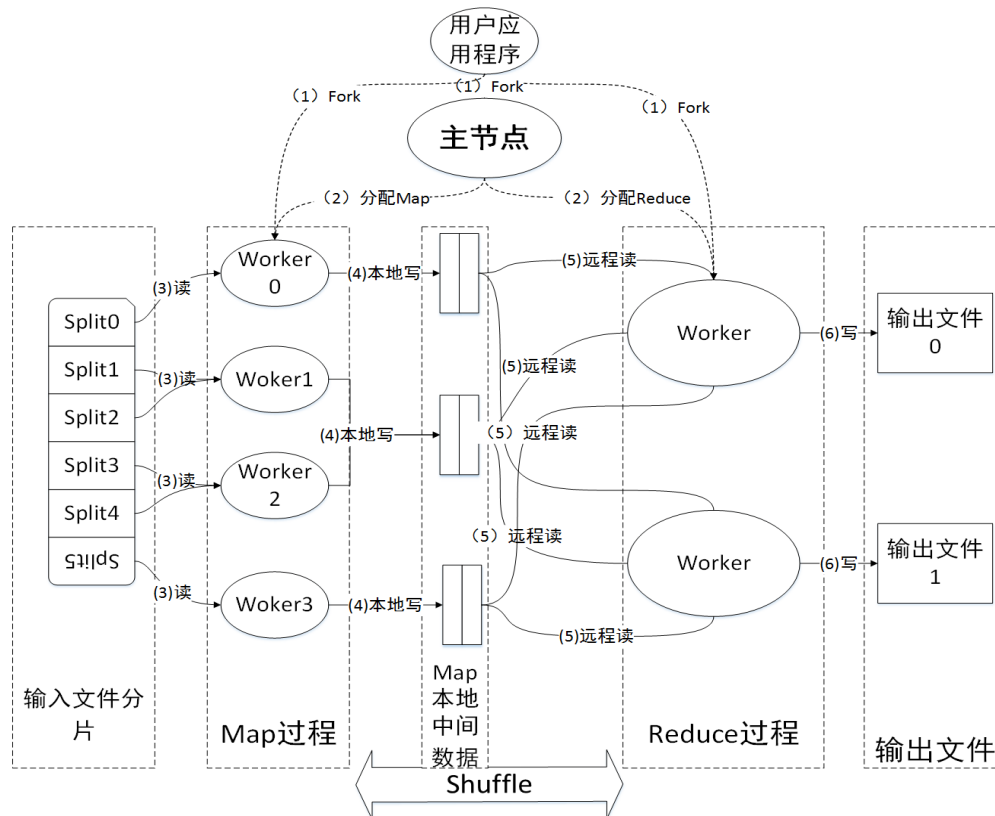


图 2-3 MapReduce 架构图

在 MapReduce 模型中输入文件一般是存储在分布式文件系统，文件在分布式文件系统中被切分为同样大小的一块块的分片 Split，每个 Split 都有多个备份。MapReduce 计算框架设计为 Master/Slave 架构，图 2-3 显示了 MapReduce 的架构设计。MapReduce 中涉及到的有这样几个元素：用户 Client，Master，Worker。MapReduce 计算模型的框架如下：

- 1) 用户提交作业任务给主节点。
- 2) 主节点检查作业配置，分配 Map 和 Reduce 任务。
- 3) Map 任务从输入文件中读入指定的分片数据，一般每个 Map 处理一个分片数据。
- 4) Map 任务执行用户指定的 map() 操作，并将数据写入到本地磁盘中。
- 5) Reduce 通过 Shuffle 过程从完成任务的 Map 端主动获取属于自己要处理的 Partition。
- 6) Reduce 取到所有 Map 的对应 Partition 输出后执行用户指定的 reduce() 工作并



将结果写到分布式文件系统中。

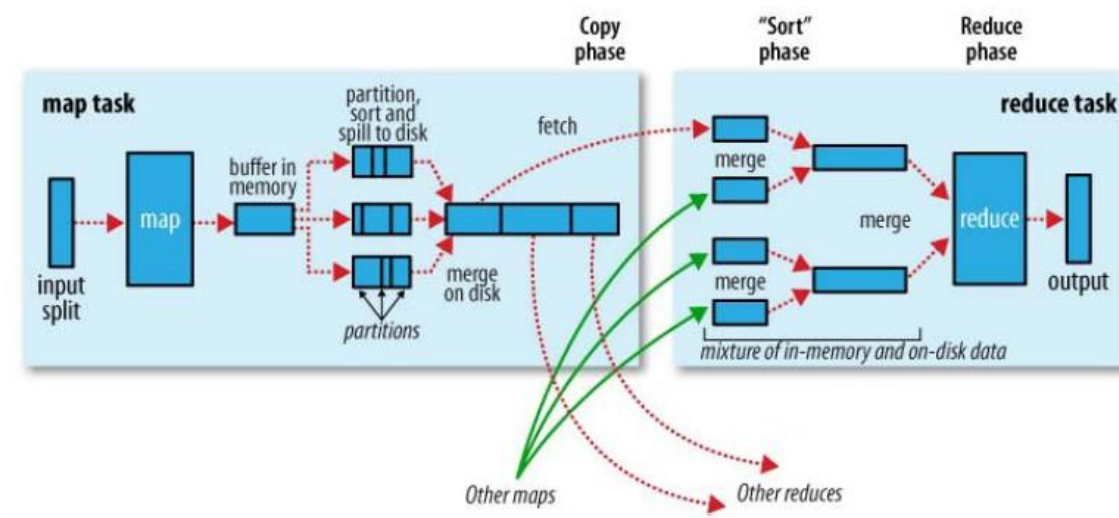


图 2-4 Shuffle 过程

MapReduce 中以 Slot 作为计算资源的分配单位，map 任务和 reduce 任务都是运行在 Slot 上。Slot 可以理解为单位计算资源，与 cpu 对应。Map 和 Reduce 过程中输入数据逻辑上都是<Key, Value>格式。Map 在处理完一条输入数据后会将输出数据先写入到本地内存环形缓冲区中，当缓冲区中的数据达到一定程度（默认为 80%）时，Map 会由 Spill 线程将内存中输出数据写到本地磁盘中。在 Map 运行过程中可能会出现多个输出文件，这些中间输出文件最终将 Merge 成一个输出文件。Merge 过程总是按 Partition 聚集，按 Value 值排序。Partition 是针对 Key 的聚集标准，通常情况下  $Partition = hash(Key) \% ReducerNum$ ，这样可以保证 Key 值相等的记录会发送到同一个 Reduce 上处理，这是保证 MapReduce 计算模型正确性的基本要求。用户可以定制划分方法，但是必须保证达到这个要求。

每一个 Reduce 都有一个指定的需要处理的 Partition 数据。Reduce 要能正确完成计算任务就需要从所有的 Map 输出文件中取得它所处理的 Partition 相关输出。Reduce 通过 Http 协议从 Map 端拷贝文件到本地，拷贝过程中同时做 Merge 工作，Merge 同样保证按 Value 有序。可见在 MapReduce 的 Shuffle 过程中一直保证了 Value 的有序性。

## 2.4 开源分布式计算平台 Hadoop

Hadoop[43]是 MapReduce 的一个开源实现，最初是 Apache 开源搜索引擎 Nutch 的一个子项目。在大数据时代，Hadoop 以其良好的扩展性、稳定性、容错性以及简洁的编程接口获得了巨大的成功。现在 Hadoop 已经成为大数据时代的标准。利用 Hadoop 可以方便有效地对大型数据集进行复杂的操作。

Hadoop 系统主要有两个组成部分：分布式文件系统 HDFS(Hadoop Distributed File System)和并行计算框架 MapReduce。HDFS 的设计目标是在通用的异构的廉价硬件上，具有高度容错性、可扩展性，针对一次存多次读，提供高吞吐量并发读的分布式文件系

统。适合存储少量的大文件。HDFS 放宽了 (relax) POSIX 的要求以此实现流的形式访问文件系统中的数据

HDFS 设计为使用普通的廉价硬件，且大规模的数据必然会使用到大量的磁盘，因此系统中磁盘发生 Failure 的情况被认为是通常情况，因此 HDFS 在设计之初就针对经常的磁盘 Failure 做容错。HDFS 有着高容错性的特点是通过数据块的多备份机制实现的。HDFS 中的数据都默认地被切分 64M 大小的分片，每个分片默认有 3 个备份，分别存储在一台机器上，与此机器同机架的一台机器上以及其他机架的一台机器上。当发现任意一台机器上发生数据丢失后，系统会自动从其他备份中去读取、复制，保证系统的三个备份。如此每个数据会有多个分片及其多个备份，这就使得在分配 Map 任务的时候可以保证较大的并发性。

Hadoop 现在依然不仅仅是一个计算平台，随着 Hadoop 生态圈的不断成长，有越来越多的优秀项目加入到其中。如 Hive、HBase、Pig 等，它们或是对 Hadoop 的改进或者是对 Hadoop 的高层次包装抽象，使得其满足一些简单的应用接口定义。Pig 提供了更加丰富的数据结构，一般为多值和嵌套的结构，还提供了更加强大的数据变换操作，包括 MapReduce 中忽略的连接(join)操作，其优点在于可以使用简单的控制台命令轻松处理 TB 级数据，对于研究人员和工程师而言操作数据更加方便简单；Hive 则是构建于 Hadoop 之上的数据仓库框架，其设计目的是让 SQL 使用者能更加容易的操作 HDFS 上的海量数据，由于建立在 Hadoop 之上，很多组织将它作为一个通用且可伸缩的数据处理平台；HBase 是对 Google 提出的 BigTable 的一种开源实现，是一个在 HDFS 上开发的面向列的分布式数据库，它不支持 SQL，也不是关系型数据库，它自底向上地进行构建，在廉价的硬件集群上管理超大规模的稀疏表，却能够简单地通过增加节点达到线性扩展，是在 HDFS 上实时地随机读/写超大规模数据集的有效方案。

由于 Hadoop 的广泛认同性，且大量基于 Hadoop 之上的开源实用系统，对 Hadoop 本身的优化也可以使得这些系统获得进一步提升。因此本文选择 Hadoop 作为并行计算平台。

## 2.5 Hadoop 中的负载均衡

负载均衡问题是一个广泛而普遍存在的问题。在所有的分布式平台中几乎都会提及到“长尾问题(Long Tail Problem)”，其实也就是大家常说的“短板理论”，系统的整体表现取决于表现最差的一部分。常见的分布式系统如分布式缓存，分布式存储，分布式计算，分布式数据库等等，都存在这个问题。分布式缓存中可能会遇到短时间内集中访问同一个缓存的情况；分布式存储可能单机磁盘使用过度；分布式计算可能会有单点的计算负担过重；分布式数据库可能会有单机访问量过大。有些问题早有了较成熟的解决方案，像分布式缓存系统中常见的一致性哈希算法等。



笔者总结对负载均衡问题的定义：在多点协作的系统中由于不合理的任务分配导致某个或者少量的某些节点处理负担过重，最终拖延整个系统对外的响应效率。负载均衡的主要解决方案有两种。一种是被动解决，当系统中发现倾斜后将负载迁移到空闲节点。另一种是主动预防，包括用户先验知识的介入预防以及从系统层面上分配任务的策略上预防倾斜发生。

对于 MapReduce 模型的 Hadoop 系统负载均衡问题的解决在大多数情况下是存在一个极限的，这取决于具体作业的可划分性。逻辑上，我们可以将不同 Map 输出的同一个 Key 的数据合起来看做一个小 Partition (Finer-Partition)。对于 MapReduce 模型本身，如果要保证计算的正确性，至少要保证的条件是：单个键的 Finer-Partition 必须要保证完整的拷贝到同一个 Slot 上。而不是看起来的，同一个 Hash 值对应到的 Partition 的多个键的 Finer-Partition 数据要保证到同一个 Slot 上。单个 Key 的 Finer-Partition 是 Reduce 输入数据的最小不可分单元。

在 Hadoop 平台是实现的相关算法其负载均衡主要通过算法运行前通过对输入数据的采样，分析输入数据的特征和分布。在具体的算法作业中根据这些先验知识对数据的发散过程做相应控制，使得各个节点之间的聚合工作能够具有一定的均衡性。另一种的思路是将算法分为多次任务的迭代，在各次迭代之间将各个节点的负载再次分散从而获得整体的均衡性，这也是本文算法的负载均衡方案所采用的思路。

## 2.6 本章总结

本章主要介绍了相关的背景技术，主要有三方面的内容，首先阐述了完全图、近似完全图的相关定义以及数学概念。然后对本文并行算法使用到的编程模型 MapReduce、计算平台 Hadoop 的基本原理做了简要介绍。最后介绍了负载均衡的相关概念和问题。



### 3 完全图和近似完全图枚举算法

单机算法是并行算法的基础，并行算法的有效性本质上取决于单机算法。本章重点介绍了极大完全图和极大近似完全图的对比算法 Binary 和基于图分割的 Binary 单机算法，最后结合两种算法的各自优势介绍了一种 BK 辅助 Binary 的混合算法 Hybrid。

#### 3.1 完全图枚举算法

本文选取现有完全图枚举算法中最具代表性的算法 BK 作为对比算法，提出了一种基于图分割的 Binary 算法。两者在基本定义上存在共性之处一并在下文中 BK 算法中介绍。

##### 3.1.1 极大完全图 BK 算法

BK 通过回溯搜索的方式来检查所有的点以枚举所有的极大完全图。BK 搜索过程中的搜索路径组成了一个树状搜索结构。BK 算法每次访问一个与搜索路径中已经访问的所有节点都相邻的点来扩展搜索，直到搜索路径不能再扩展，这样一条搜索路径上的所有访问点就形成一个极大完全图。

算法 BKCliqueEnumerate 显示了 BK 搜索树的访问过程。搜索树的每一个节点由下面三个点集组成，也就是搜索树的节点状态表示结构。

1. Result 集，现有的搜索路径中已经访问过的节点集合
2. Candidate 集，不在 Result 集中且与 Result 中的每一个点都相邻的点集
3. Not 集，与 Result 中点都相邻且如果与 Result 集中的点组合将导致产生冗余或者被包含的结果的点集

算法 BKCliqueEnumerate 在访问一个节点时将其加入到现有结果集中并重新构建新的 Candidate 集合 Not 集。新的 Candidate 集通过筛选现有的候选集要与包括新加入到结果集中的点在内的所有 Result 集中的点都相邻。同样 Not 集也是通过筛选现有的 Not 集要求与包括新加入的点在内的所有 Result 点都相邻。BKCliqueEnumerate 算法通过访问候选集中的点。对于每一个搜索树节点，首先访问的点是当前候选点中连接了最多候选点的点（也就是在候选点组成的子图中度数最大的点）。在搜索完第一个点之后只用那些与第一个点不相邻的候选点来扩展搜索路径。这样保证了在每一节点的搜索路径中只搜索最少的那些有可能生成新的且不冗余的点。一旦候选点 Candidate 中的任意一个点被访问后都会将其加入到 Not 集中，以表示包含这个点和当前结果集中的极大完全图已经在新生成的子图状态中考虑了，搜索树中其他兄弟子图节点状态不应当在搜索相关节点。当 BKCliqueEnumerate 已经搜索完所有的子节点状态后将回溯到之前的一个未搜索子图状态。

表 3-1 BK 算法主过程

|  |
|--|
| 算法 1: BK 算法  |
| 输入: 数据图 $G$ , 图 $G$ 的节点集 $V$ 以及边集 $E$                |
| 输出: 所有的且不重复、不冗余的极大完全图                                |
| 1. $Result \leftarrow \phi$                          |
| 2. $Candidate \leftarrow V$                          |
| 3. $Not \leftarrow \phi$                             |
| 4. 调用方法 $BKCliqueEnumerate (Result, Candidate, Not)$ |

表 3-2 极大完全图 BK 枚举递归方法

|  |
|--|
| 方法 $BKCliqueEnumerate (Result, Candidate, Not)$          |
| 1. If $Candidate = \phi$ then                            |
| 2.     If $Not = \phi$ then                              |
| 3.         输出结果 $Result$                                 |
| 4. Else  |
| 5. $fixp \leftarrow Candidate$ 中与其他 $Candidate$ 中相邻点最多的点 |
| 6. $cur\_v \leftarrow fixp$ ;                            |
| 7.     While $cur\_v \neq NULL$ do                       |
| 8. $new\_not \leftarrow Not$ 集中所有与 $cur\_v$ 相邻的点         |
| 9. $new\_cand \leftarrow Candidate$ 集中与 $cur\_v$ 相邻的点    |
| 10. $new\_res \leftarrow Result + cur\_v$                |
| 11. $ClqueEnumerate(new\_res, new\_cand, new\_not)$      |
| 12. $Not \leftarrow Not + cur\_v$                        |
| 13. $Candidate \leftarrow Candidate - cur\_v$            |
| 14.        If $Candidate$ 中存在点 $v$ 与 $fixp$ 不相邻 then     |
| 15. $Cur\_v \leftarrow v$                                |
| 16.        Else  |
| 17. $Cur\_v \leftarrow NULL$                             |
| 18. return   |

考虑图 3-1 中的图  $G(V,E)$  作为输入图, 初始状态中所有点都作为候选点包含在  $Candidate$  集合中。在所有候选节点中度数最大的点为标号为 3 的点, 与 3 不相邻的点有  $\{5,6,7\}$ , 因此根据 BK 算法将根节点子图扩展为包含 3,5,6,7 的四个子图。

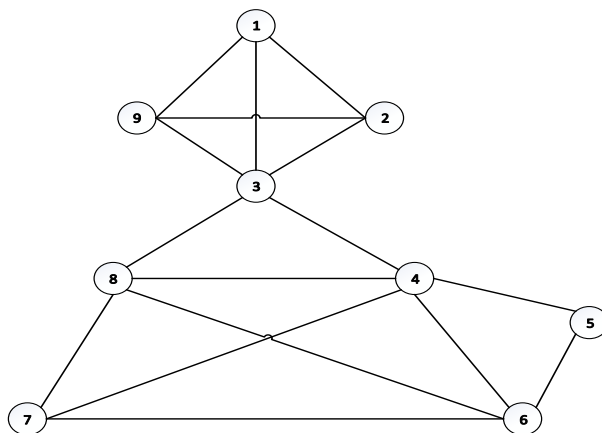


图 3-1 输入数据图

其搜索状态结构如图 3-2 所示。搜索状态中每个状态节点由三个结合组成，最上面的表示结果集 Result，中间表示候选集 Candidate，下面表示 Not 集。

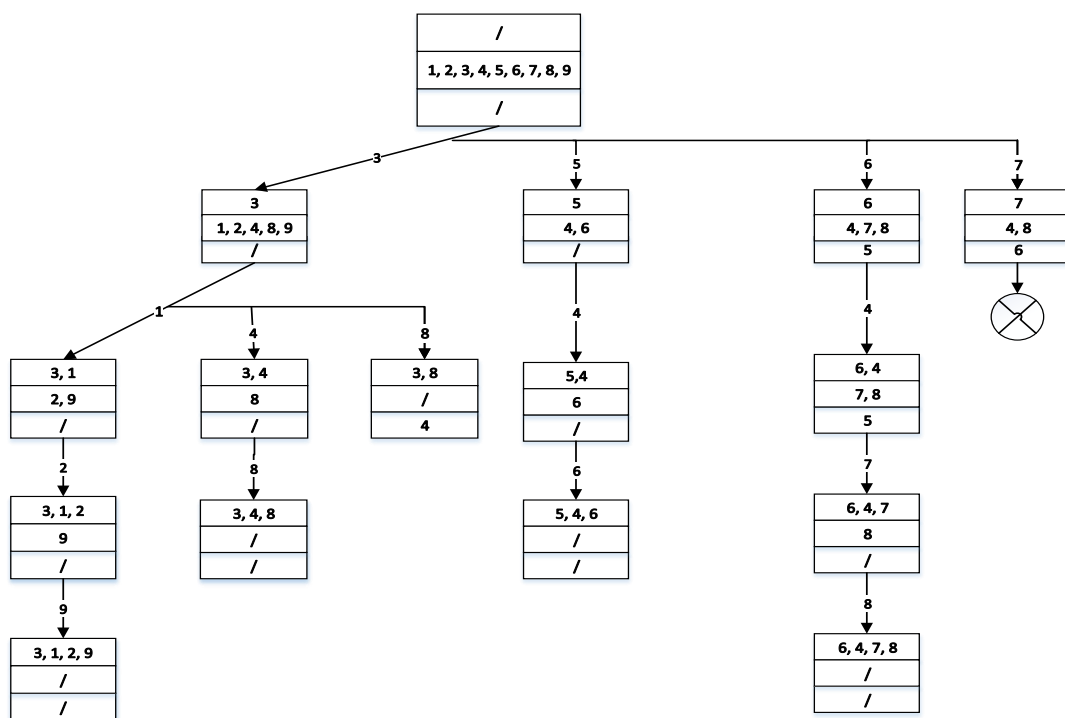


图 3-2 BK 搜索状态树

搜索状态树中连接线上点表示用来扩展的节点。搜索树中叶子状态都是需要找的极大完全图，除了一些被排除的状态。最右侧用节点 7 扩展的状态中由于 Not 集中一个点 6 与当前所有候选点{4,8}都相邻，因此这个状态节点可以被排除，因为这个状态所能扩展出来的所有极大完全图都已经在之前出现过或者被之前出现过的极大完全图包含了。事实上，可以观察到点 7 扩展的子图状态所能找到的最终结果{7,4,8}被之前 6 扩展的结果{6,4,7,8}包含了。一个完整的抽象的 BK 搜索树扩展结构如图 3-3 所示。

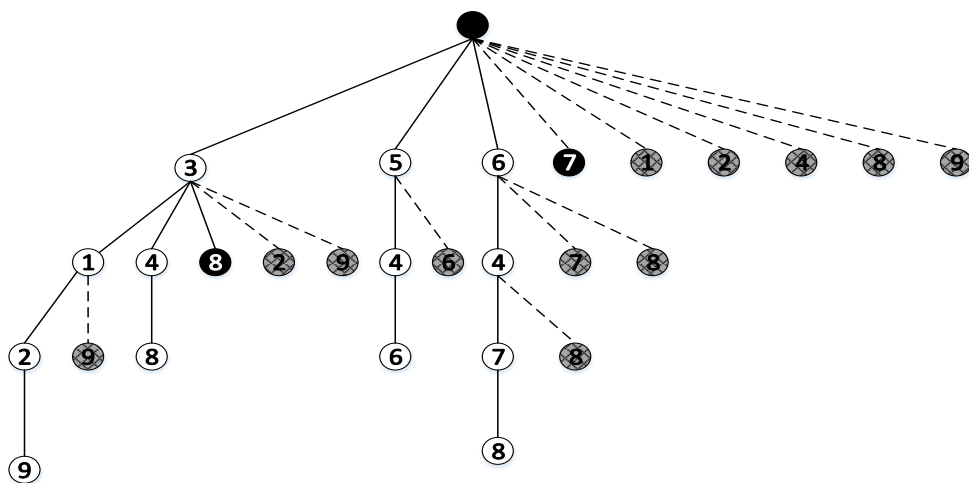


图 3-3 完整 BK 扩展树结构

尽管递归的 BK 算法描述非常简洁，但在实际应用环境中递归由于其大量的栈空间和内存、cpu 消耗并不实用。BKCliqueEnumerate 的搜索过程可以通过栈来模拟实现，如图 3-4 所示。下文 Binary 算法也具有相似的栈式实现，在实际实验过程中都是使用的基于栈的算法实现，后文不再赘述。

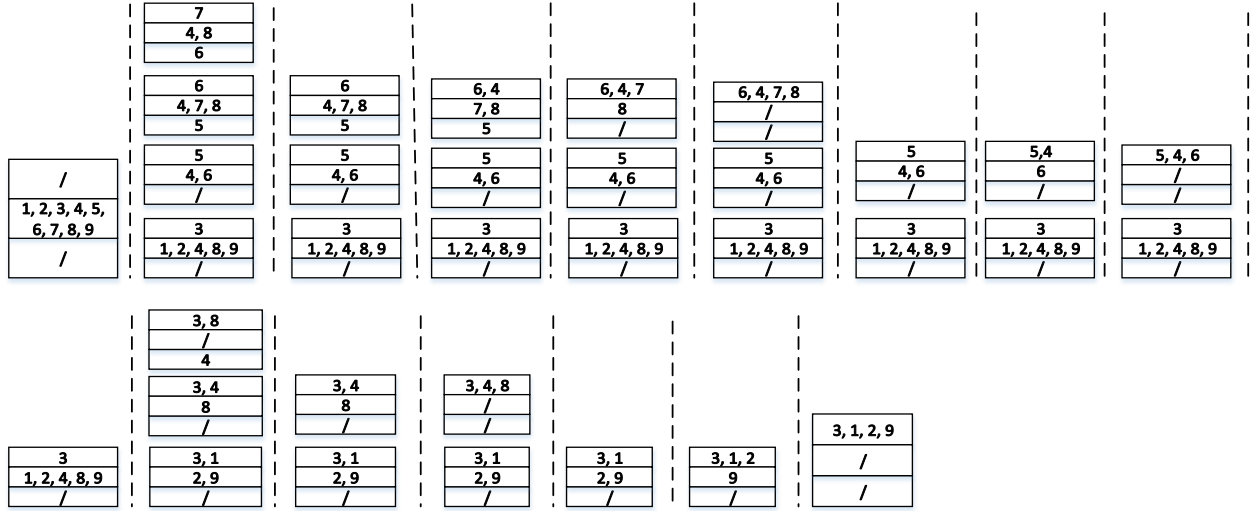


图 3-4 基于栈的 BK 算法过程

### 3.1.2 极大完全图 Binary 算法

BK 算法在执行过程中需要计算每个子图中各个候选节点的度数，但只是用来寻找度数最大的某个点，关于其他点的度数计算结果并没有充分使用。同时，观察到许多算法没有使用到完全图一个明显的特征没有使用，图  $G(V, E)$  是完全图  $\Leftrightarrow \forall \mu \in V, |N(\mu)| = |V| - 1$ 。另外不同于 BK 算法取最大度数点然后取所有与最大度数点不相邻的点来扩展搜索路径的粗粒度方式，本文提出一种高效地衡量当前状态动态地选择扩展点的搜索算法：Binary 算法。

在 BK 的数据结构设计的基础上当前子图状态表示为 (Result, Candidate, Not)，Binary 算法的基本过程是在当前候选节点集中选择一个扩展点（分裂点） $v$  并将当前搜索图  $G$  划分为包含当前  $G$  的结果集且包含点  $v$  的所有 Clique，生成新的子图  $G_v^+$ ，子图状态通过如下方式生成  $Result^+ = Result \cup \{v\}$ ， $Candidate^+ = Candidate \cap N(v)$ ， $Not^+ = Not \cap N(v)$ ；包含当前  $G$  的结果集且不包含点  $v$  的所有 Clique，生成新的子图  $G_v^-$ ，子图状态通过如下方式生成  $Result^- = Result$ ， $Candidate^- = Candidate - \{v\}$ ， $Not^- = Not \cup \{v\}$ 。

下面证明这种划分方式的正确性。对于当前子图  $G$  的状态的子分支只有两种情况，包含点  $v$  和不包含点  $v$ ，不存在第三种情况。对于包含点  $v$  的状态  $G_v^+$ ，结果 Clique 中如果包含点  $v$ ，那么可以将点  $v$  加到结果集中即： $Result^+ = Result \cup \{v\}$ ；状态  $G_v^+$  的候选点由于需要满足包含节点  $v$  的条件，因此需要保证候选点也必须都是  $v$  的邻接点即

$Candidate^+ = Candidate \cap N(v)$ ；状态  $G_v^+$  的 Not 集需要满足 Not 集的定义，Not 集中的点可以与当前结果集节点组成完全图，因此 Not 集中点需要能够与  $v$  相邻即  $Not^+ = Not \cap N(v)$ 。对于不包含节点  $v$  的状态  $G_v^-$ ，只将点  $v$  从候选点中删除，同时由于 Not 的定义且包含点  $v$  的子图必然都会在  $G_v^+$  中处理，因此将点  $v$  加入到 Not 集中，这样  $G_v^-$  保证了所有不包含节点  $v$  的完全图都是  $G_v^-$  的子图。有以上可得证 Binary 划分方式的正确性。

表 3-3 Binary 算法主过程

|   |
|---|
| 算法 1: Binary 算法   |
| 输入: 数据图 $G$ , 图 $G$ 的节点集 $V$ 以及边集 $E$                     |
| 输出: 所有的且不重复、不冗余的极大完全图                                     |
| 5. $Result \leftarrow \phi$                               |
| 6. $Candidate \leftarrow V$                               |
| 7. $Not \leftarrow \phi$                                  |
| 8. 调用方法 BinaryCliqueEnumerate( $Result, Candidate, Not$ ) |

Binary 的划分方式需要每次从当前子图状态中找到一个分裂点，如何选择分裂点对算法的效率有着至关重要的影响。考虑到图数据的各种特征且充分利用现有信息，本文使用候选点的度数作为选择切分点的衡量指标。显然，用度数作为选择指标有三种选择方式：选择度数最小的点、选择度数最大的点、随机选择一个点。根据 Binary 的划分方式，包含点  $v$  的子图  $G_v^+$  候选点的规模  $|Candidate^+| = |Candidate \cap N(v)| = |N(v)|$ ，子图  $G_v^-$  候选点的规模  $|Candidate^-| = |Candidate - \{v\}| = |Candidate| - 1$ 。在给定子图  $G$  的情况下  $|Candidate|$  是不变的，由此可见选择最小度数点作为切分点则会生成一个较大的子图和一个较小的子图可以减少搜索空间。下图 3-5 是最大和最小度数切分的一个对比实例，后续的实验也验证了这一论断。

表 3-4 递归极大完全图 Binary 枚举方法

|  |
|--|
| 方法 BinaryCliqueEnumerate( $Result, Candidate, Not$ )           |
| 1. If Not 中存在一个点与 Candidate 中所有点都相邻 then                       |
| 2.     此子图往下都是冗余结果，返回  |
| 3. If SubGraph(Candidate)是 Clique then                         |
| 4.     输出 $Result \cup Candidate$                              |
| 5. Else  |
| 6.     在 Candidate 中选择一个分裂点 $v$                                |
| 7. $Result^+ \leftarrow Result \cup \{v\}$                     |
| 8. $Candidate^+ \leftarrow Candidate \cap N(v)$                |
| 9. $Not^+ \leftarrow Not \cap N(v)$                            |
| 10.    BinaryCliqueEnumerate( $Result^+, Candidate^+, Not^+$ ) |
| 11. $Result^- \leftarrow Result$                               |
| 12. $Candidate^- \leftarrow Candidate - \{v\}$                 |
| 13. $Not^- \leftarrow Not \cup \{v\}$                          |
| 14.    BinaryCliqueEnumerate( $Result^-, Candidate^-, Not^-$ ) |

使用 Binary 算法处理同样的输入图，其搜索树的结构如下图 3-6 所示。图 (a) 是算法描述的 Binary 搜索树结构左子节点表示  $G^-$  右子节点表示  $G^+$ 。由于递归算法的栈消耗以及潜在的内存使用问题，本文采用栈记录子图状态的迭代方式实现 Binary 的递归算法。实际上在算法迭代过程中会将栈顶子图一直选最小度数点切分直到栈顶子图成为一个完全图或者栈顶子图已经不可能形成新的有意义的完全图。

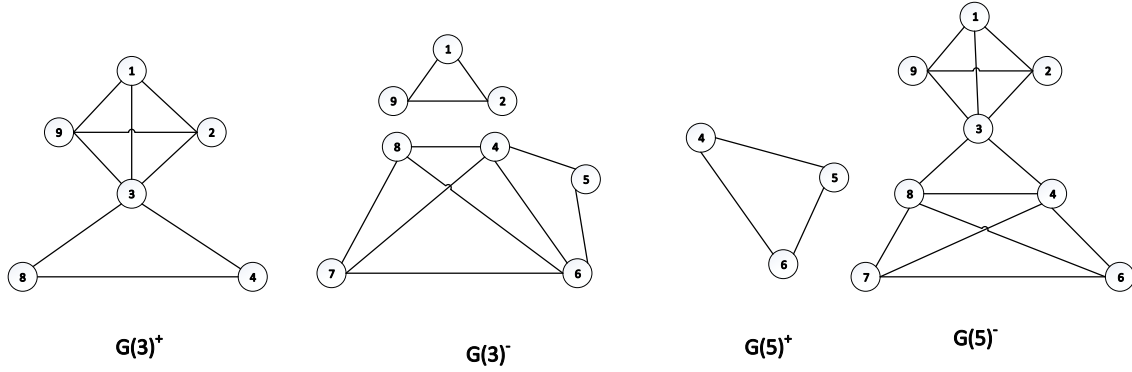


图 3-5 最大度数切分和最小度数切分

迭代算法切分过程中  $G^-$  是一个可变化的子图状态，将左子树节点  $G^-$  都映射到同一个子图状态，如图 (b) 所示。可以看出 Binary 算法的划分方式可以在很大程度上减少搜索子图的个数，这一点在后续的试验中得到了有效验证。

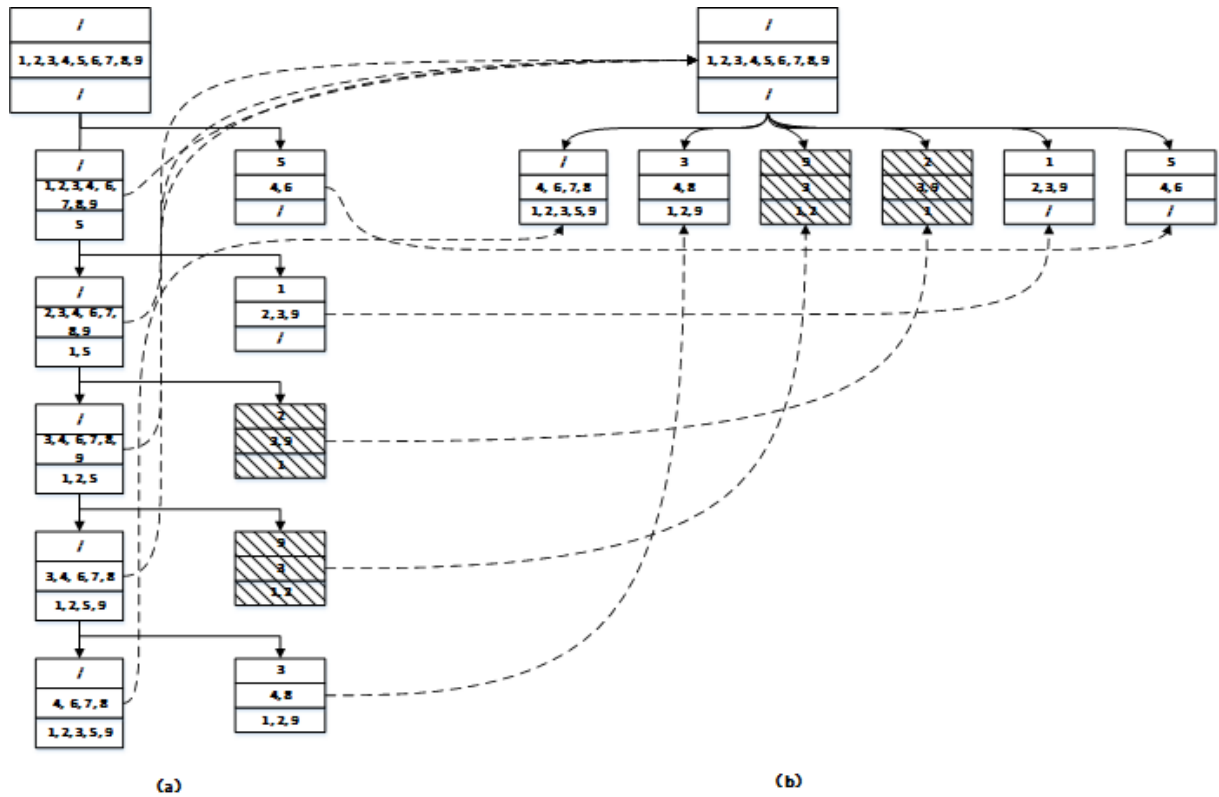


图 3-6 Binary 搜索树状态结构



Binary 算法的实现中使用与 BK 相似的数据集合定义 Result, Candidate, Not。不同的是, Binary 算法需要始终跟踪 Candidate 节点中度数最小的点, 而且在算法执行过程中由于需将与分裂点相邻的点度数减一, Candidate 中节点的度数不断变化的。对此, 最直接的方法是每次需要选度数最小的点时遍历当前所有候选点的度数, 从中选出度数最小的点。假设子图候选点个数为  $n$ , Binary 算法需要切分子图  $O(n)$  次, 第  $i$  次切分需要筛选  $n-i$  个节点个数, 因此在筛选最小度数点上需要  $O(\sum_{i=1}^{n-1} i) = O(n(n-1)/2) = O(n^2)$  的时间复杂度, 在先期的实验中此部分成为算法效率的一个主要瓶颈。本文设计了一个最小度数结构用来更新和维护各个候选节点的度数, 提供了  $O(1)$  的最小度数点选取操作, 以及  $O(1)$  每个节点度数更新的操作。

如下图 3-7 (a) 所示圆形表示度数, 方形表示节点标号。最小度数结构将度数相同的节点放在一个哈希桶内并用它们的度数作为桶的标志, 圆形度数标志之间形成一个有序的链表结构, 同时每个节点标号有指向本身所在桶的引用。

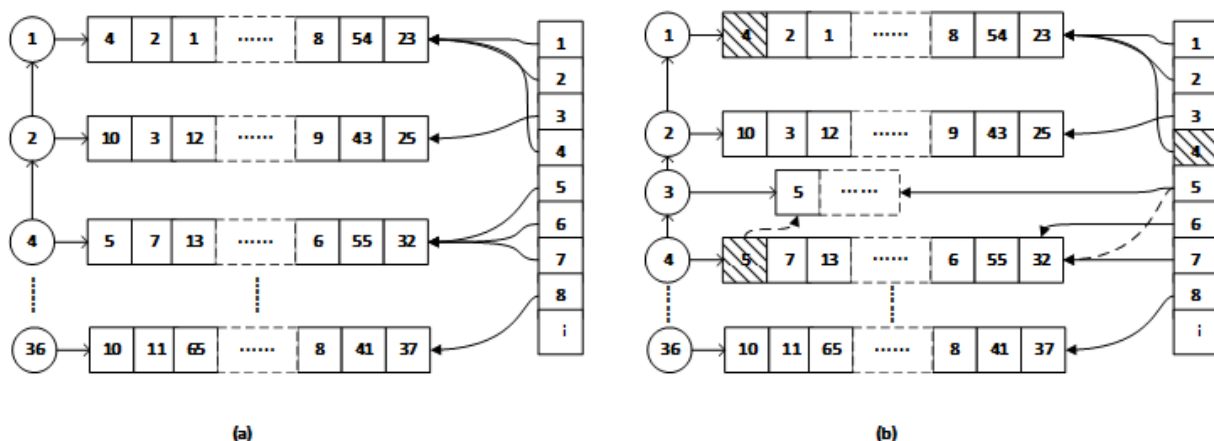


图 3-7 最小度数集合结构

在图 3-7 (a) 表示的结构中进行图分割时, 从度数链表选取头结点所指向的桶中选取第一个节点 4 (标号为 4 的节点度数为 1, 假设候选点中与 4 号节点相邻的是标号为 5 的点) 作为分裂点, 在不包含节点 4 的子图中需要更新度数集合, 将与 4 相邻的节点 5 的度数减一。度数减一操作如图 3-7 (b) 所示, 将节点 5 从原来的桶中移除并将其加到其之前度数小一的桶中 (如果不存在的话需要新建一个桶, 并将度数节点插入到原链表中) 同时更新节点 5 所在的桶引用。其中所有操作都是  $O(1)$  复杂度, 建立这个最小度数集合的额外复杂度是  $O(k \log k)$  ( $k$  表示所有候选点的不同度数个数)。

### 3.1.3 极大完全图 Hybrid 算法

BK 算法和 Binary 算法一个重要的区别在于关于图分割的定义, BK 算法将图分割为包含最大度数节点的子图其余与该店不相邻的点依次罗列, 相当于一当选点即可将当前搜索图全部切分完; Binary 算法每次选用最小度数点将子图划分为多个小的子图, 多次迭代分步将搜索图切分掉。在不同的情况下两者存在着各自的优势, 当图中存在一个

节点的与大部分节点都相邻时 BK 算法可以将图有效的切分为较少的几个子图；当图中节点度数偏小时 Binary 算法可以有效地将图切分为冗余较少的且易计算的多个小子图。

在大多数实际数据中 Binary 算法都能够有较好的适应性，但 BK 算法在蛋白质交互网络数据中具有良好的切分效果。鉴于实际应用中数据图的多样性以及算法的适用性，本文提出了一种 Binary 和 BK 综合的算法 Hybrid。表 3-5 描述了 Hybrid 算法的枚举过程。由前面算法的描述过程可以看出 BK 算法和 Binary 算法在搜索树节点状态的整体结构上相似，因此两者可以较方便地实现融合。

表 3-5 Hybrid 算法枚举过程

| 方法 HybridCliqueEnumerate(Result, Candidate, Not) |   |
|--|---|
| 1.   | If Not 中存在一个点与 Candidate 中所有点都相邻 then                   |
| 2.   | 此子图往下都是冗余结果，返回  |
| 3.   | If SubGraph(Candidate)是 Clique then                     |
| 4.   | 输出 $Result \cup Candidate$                              |
| 5.   | Else  |
| 6.   | 从最小度数结构中选择度数最小的点 $v$                                    |
| 7.   | If $Deg(v) > \alpha *  Candidate $ then                 |
| 8.   | 从最小度数结构中取度数最大节点 $\mu$                                   |
| 9.   | $cur\_v \leftarrow \mu$ ;                               |
| 10.  | While $cur\_v \neq NULL$ do                             |
| 11.  | $new\_not \leftarrow Not$ 集中所有与 $cur\_v$ 相邻的点           |
| 12.  | $new\_cand \leftarrow Candidate$ 集中与 $cur\_v$ 相邻的点      |
| 13.  | $new\_res \leftarrow Result \cup \{cur\_v\}$            |
| 14.  | CliqueEnumerate( $new\_res, new\_cand, new\_not$ )      |
| 15.  | $Not \leftarrow Not \cup \{cur\_v\}$                    |
| 16.  | $Candidate \leftarrow Candidate - \{cur\_v\}$           |
| 17.  | If Candidate 中存在点 $v$ 与 fixp 不相邻 then                   |
| 18.  | $Cur\_v \leftarrow v$                                   |
| 19.  | Else  |
| 20.  | $Cur\_v \leftarrow NULL$                                |
| 21.  | Return  |
| 22.  | BinaryCliqueEnumerate( $Result^+, Candidate^+, Not^+$ ) |
| 23.  | Else  |
| 24.  | $Result^+ \leftarrow Result \cup \{v\}$                 |
| 25.  | $Candidate^+ \leftarrow Candidate \cap N(v)$            |
| 26.  | $Not^+ \leftarrow Not \cap N(v)$                        |
| 27.  | $Result^- \leftarrow Result$                            |
| 28.  | $Candidate^- \leftarrow Candidate - \{v\}$              |
| 29.  | $Not^- \leftarrow Not \cup \{v\}$                       |
| 30.  | BinaryCliqueEnumerate( $Result^-, Candidate^-, Not^-$ ) |

Hybrid 算法以 Binary 算法为基础，对候选点集合中的所有点建立最小度数集合，在选择分裂点时考虑最小度数点邻接的候选节点比例，当被切割的图中最小度数点都邻接

了大部分节点时,不再取最小度数点使用 Binary 方式切分,而是重新从最下度数集中选取最大度数点使用 BK 的切分方式将当前剩余的搜索图一次切分完成,否则还是使用最小度数点继续依次分割搜索图。由上一节中最小度数集合的设计可以看出,选最大度数点也只需从链表中取尾节点的桶中取任意一个点即可,其时间复杂度也只有  $O(1)$ 。Hybrid 算法的实现要求,BK 和 Binary 都可以较好地支持,不需做数据结构上的大改动。在 Hybrid 算法中使用到了一个变量  $\alpha, \alpha \in (0,1)$ ,表示最小度数点与候选点中邻接个数比例,实验部分中验证该值一般取 0.6 即最小度数点与 60% 以上的候选点都相邻的时候使用 BK 切分能够获得较好的效果。

### 3.2 近似完全图枚举算法

K-Plex 算法是 BK 算法的一种变形,其本质上也是图的分割。为简化下文算法的描述有如下定义:

定义 1: 顶点  $v$  与顶点集合  $S$  相邻,要求满足  $\exists \mu \in S, (v, \mu) \in E$  用  $v \odot S$  表示; 顶点  $v$  与  $S$  不相邻,要求满足  $\forall \mu \in S, (v, \mu) \notin E$  用  $v \oslash S$  表示。

K-Plex 算法 Pemp 与完全图算法相似,也有关于搜索节点相似的定义 Result, Candidate, Not。在 K-Plex 中用来扩展现有搜索路径的 Candidate 候选节点  $v$  需要满足以下两个条件

条件 1:  $v$  与 Result 中至少  $|\text{Result}| - k + 1$  个点相邻

条件 2:  $\forall \mu \in \text{Result}, \mu$  与  $\text{Result} \cup \{v\}$  中至少  $|\text{Result}| - k + 1$  个点相邻

显然任意一个满足条件 1 和条件 2 的点都可以与现有的结果集组成一个新的更大的 K-Plex。根据 Candidate 和 Not 集合的定义, Candidate 和 Not 集中的点都需要满足条件 1 和条件 2。

定义 2: 搜索路径上一个子图节点的结果集 Result 中的点  $v$  与  $\{\text{Result} - v\}$  中  $k-1$  点不相邻,则称  $v$  是临界点; 由临界点组成的集合即临界点集合 Critical。

为了满足条件 2, 当搜索路径子图中临界点集合 Critical 不为空时,要求可用来扩展的 Candidate 中的任意一点  $v$  必须与临界点集合中的所有点都相邻。否则,那么当不满足此条件的点  $v$  用来扩展现有路径时  $\text{Result} \cup \{v\}$  中会至少存在一个临界点与  $K$  个点不相邻,也就是违背了 K-Plex 的必要条件。

#### 3.2.1 基于 BK 的 K-Plex 算法 Pump

在 Pemp 算法中为 Result、Candidate 和 Not 集中点都关联了一个计数: counter1。counter1 用来记录其关联的顶点与 Result 中的多少个点不相邻,其中 Result 的 counter1 表示 Result 中的点  $v$  与  $\text{Result} - \{v\}$  中多少个点不相邻。为了满足条件 1,只需要保证点的  $\text{counter1} \leq k-1$ 。表 3-6 描述了 Kplex 单机算法的主过程,表 3-7 描述了单机 Kplex 中的 FindAllMaximalKplex 方法, FindAllMaximalKplex 是实际枚举过程。

表 3-6 KPLEX 单机算法主过程

| KPLEX 单机算法 Pump                     |  |
|-------------------------------------|--|
| 输入：数据图 $G$ ，图 $G$ 的节点集 $V$ 以及边集 $E$ |  |
| 输出：所有的且不重复、不冗余的 Kplex               |  |
| 1.                                  | $Result \leftarrow \phi$                           |
| 2.                                  | $Candidate \leftarrow V$                           |
| 3.                                  | $Not \leftarrow \phi$                              |
| 4.                                  | While $Candidate \neq \phi$                        |
| 5.                                  | 依次取 $cur\_v \in Candidate$                         |
| 6.                                  | $Result \leftarrow Result + cur\_v$                |
| 7.                                  | 调用方法 $FindAllMaximalKplex(Result, Candidate, Not)$ |
| 8.                                  | $Not \leftarrow Not + cur\_v$                      |
| 9.                                  | $Candidate \leftarrow Candidate - cur\_v$          |
| 10.                                 | $Result \leftarrow Result - cur\_v$                |

表 3-7 FindAllMaximalKplex 方法

| 方法 $FindAllMaximalKplex(Result, Candidate, Not)$ |   |
|--|---|
| 1.   | $Connected\_Candidate \leftarrow Candidate$ 中与 $Result$ 中任意一个点相邻的点                                |
| 2.   | $Connected\_Not \leftarrow Not$ 中与 $Result$ 中任意一个点相邻的点  |
| 3.   | While $Connected\_Candidate \neq \phi$  |
| 4.   | If $Connected\_Candidate \neq \phi$   |
| 5.   | If $Connected\_Not \neq \phi$   |
| 6.   | 输出 $Result$   |
| 7.   | Else  |
| 8.   | $Result$ 是冗余结果  |
| 9.   | $cur\_v \leftarrow SelectExpandNode(Result, Critical\_Res, Connected\_Candidate, Connected\_Not)$ |
| 10.  | If $cur\_v = NULL$  |
| 11.  | Return  |
| 12.  | $Candidate \leftarrow Candidate - cur\_v$   |
| 13.  | $Cur\_cand \leftarrow Candidate; Cur\_not \leftarrow Not; Cur\_res \leftarrow Result$             |
| 14.  | $Cur\_res \leftarrow Cur\_res + cur\_v$   |
| 15.  | 更新 $Cur\_cand$ 和 $Cur\_not$ 中各个点的计数，同时移除计数 $counter1$ 大于 $k-1$ 的点                                 |
| 16.  | $Critical\_Res \leftarrow Result$ 中与其他节点不相邻个数等于 $k-1$ 的点  |
| 17.  | 移除 $Cur\_cand$ 和 $Cur\_not$ 中与 $Critical\_Res$ 不相邻的点  |
| 18.  | $FindAllMaximalKplex(Result, Candidate, Not)$   |
| 19.  | $Not \leftarrow Not + cur\_v$   |
| 20.  | $Connected\_Not \leftarrow Connected\_Not + cur\_v$   |

算法去冗余的方法是通过最后  $Connected\_Candidate$  为空且  $Connected\_Not$  不为空表示  $Connected\_Not$  中的任意一个点  $v$  都可以和现有的结果集组成一个  $K$ -plex，但是包含点  $v$  的所有  $K$ -plex 已经在搜索路径的其他分支上已经生成了，故而无需输出。如果可以在搜索路径的过程中提前判断  $Connected\_Not$  中存在一个点不可能被过滤掉，那么就可以保证这个节点下的整棵子树都是冗余的，可以提前剪枝。如果  $Connected\_Not$  中存在一个点与  $Result$  和  $Candidate$  中的所有点都相邻，那么在此之后无论如何扩展这个搜索路径，都无法通过条件 1 和条件 2 来将这个点过滤掉。也就是说在此之后所有的结果中

Connected\_Not 都不为空，此分支可以剪去。

剪枝条件： $\exists v \in \text{Not}, \forall \mu \in \text{Candidate} \cup \text{Result}, (v, \mu) \in E$

在算法中为 Not 关联一个新的计数器 counter2, counter2 表示 Not 中的点与 Result 和 Candidate 中有多少个点不相邻。当 Not 中存在点的 counter2 = 0 时也就是剪枝条件满足了。

为了加快剪枝速度，算法尽可能的使得 counter2 降到 0。为此提出一个新的集合 Prunable\_Not, Prunable\_Not 是 Not 的一个子集，包含 Not 中与 Result 集中所有点都相邻的点，也就是 counter1=0 的那些点。注意到，每当选择一个新的点 v 加到 Result 中来扩展搜索时，Not 中与 v 不相邻的点的 counter2 计数就会每次降 1。如果一直从 Candidate 中选与 Prunable\_Not 集中某个点 v 不相邻的那些点来扩展，那么点 v 的 counter2 很快降到 0 且将不可能存在 Not 集中的其他点会比 v 更快满足剪枝条件。显然，为了能够最快地达到剪枝条件应该选择 Connected\_Candidate 中与 Prunable\_Not 中 counter2 最小的点 v 相邻的那些点。但是在算法刚开始时还不存在临界点集合时 Connected\_Candidate 中不存在与 Prunable\_Not 不相邻的点，因此还是需要按顺序取扩展点。详细过程在表 3-8 Pump 寻找扩展点方法 SelectExpandNode 中描述。

表 3-8 Pump 寻找扩展点方法

| 方法 SelectExpandNode (Result, Critical_Res, Connected_Candidate, Connected_Not) |  |
|--|--|
| 1.   | If Result $\neq \emptyset$ && Critical_Res $\neq \emptyset$  |
| 2.   | Return Connected_Candidate 中的第一个点  |
| 3.   | Else   |
| 4.   | Prunable_Not $\leftarrow$ Connected_Not 中与 Result 中所有点都相邻的点  |
| 5.   | If Prunable_Not $\neq \emptyset$   |
| 6.   | If $\text{Min}(v.\text{counter2} : v \in \text{Prunable\_Not}) = 0$  |
| 7.   | Return NULL  |
| 8.   | Else   |
| 9.   | Return $v$ , $v.\text{counter2} = \text{Min}(\mu.\text{counter2} : \mu \in \text{Connected\_Candidate} \& \mu \in \text{Prunable\_Not})$ |
| 10.  | Else   |
| 11.  | Return Connected_Candidate 中的第一个点  |

### 3.2.2 基于 Binary 的 K-Plex 算法

K-Plex 的 Binary 算法使用与完全图相似的切分方式。K-Plex 的 Binary 算法中对一个子图状态中的节点都关联两个度数 cDeg 和 rDeg。cDeg 表示关联的节点与 Candidate 中有多少个点不相邻，rDeg 表示关联的节点与 Result 中有多少个点不相邻。参考文献指出有意义的 K-Plex 的所有节点必然包含在两跳以内，因此本文在使用候选节点时使用两跳数据集。

表 3-9 K-Plex Binary 算法主过程

| KPLEX 单机算法 Binary                   |   |
|-------------------------------------|---|
| 输入：数据图 $G$ ，图 $G$ 的节点集 $V$ 以及边集 $E$ |   |
| 输出：所有的且不重复、不冗余的 Kplex               |   |
| 1.                                  | $Result \leftarrow \phi$  |
| 2.                                  | $Candidate \leftarrow V$  |
| 3.                                  | $Not \leftarrow \phi$   |
| 4.                                  | While $Candidate \neq \phi$   |
| 5.                                  | 依次取 $cur\_v \in Candidate$  |
| 6.                                  | $Result \leftarrow Result + cur\_v$                                       |
| 7.                                  | $TwoHop \leftarrow N(cur\_v) \cup N(N(cur\_v))$                           |
| 8.                                  | 调用方法 $BinaryMaximalKplex(Result, Candidate \cap TwoHop, Not \cap TwoHop)$ |
| 9.                                  | $Not \leftarrow Not + cur\_v$   |
| 10.                                 | $Candidate \leftarrow Candidate - cur\_v$                                 |
| 11.                                 | $Result \leftarrow Result - cur\_v$                                       |

表 3-10 Binary 的 K-Plex 枚举算法

| 方法 $BinaryMaximalKplex(Result, Candidate, Not)$ |   |
|---|---|
| 1.  | 计算最小度数结构、临界点 $Critical$ 、                                       |
| 2.  | $Prunable\_Not \leftarrow Not$ 中 $rDeg=0$ 的点                    |
| 3.  | While $Candidate$ 还有切分的意义                                       |
| 4.  | If $Prunable\_Not$ 中存在 $cDeg=0$ 的点                              |
| 5.  | 此分支可以减掉；Return  |
| 6.  | If $\forall v \in Candidate \cup Result, v.rDeg + v.cDeg < k-1$ |
| 7.  | 输出 $Candidate \cup Result$ 为 K-Plex；Return；                     |
| 8.  | 从最小度数结构中选出 $rDeg+cDeg$ 最大的点 $cur\_v$                            |
| 9.  | $Result^+ \leftarrow Result \cup \{cur\_v\}$                    |
| 10.   | $Critical^+ \leftarrow Result^+$ 中 $rDeg=k-1$ 的点                |
| 11.   | $Candidate^+ \leftarrow Candidate - \{cur\_v\}$                 |
| 12.   | 更新 $Result$ 中各点的 $rDeg$ 和 $cDeg$                                |
| 13.   | $Not^+ \leftarrow Not \cap N(v)$                                |
| 14.   | $BinaryCliqueEnumerate(Result^+, Candidate^+, Not^+)$           |
| 15.   | $Candidate \leftarrow Candidate - \{v\}$                        |
| 16.   | 更新 $Cur\_cand$ 和 $Cur\_not$ 中各个点的计数，同时移除计数 $cDeg$ 大于 $k-1$ 的点   |
| 17.   | $Critical\_Res \leftarrow Result$ 中与其他节点不相邻个数等于 $k-1$ 的点        |
| 18.   | 移除 $Cur\_cand$ 和 $Cur\_not$ 中与 $Critical\_Res$ 不相邻的点            |
| 19.   | $Not \leftarrow Not + cur\_v$                                   |
| 20.   | If $cur\_v.rDeg =  Result $                                     |
| 21.   | $Prunable\_Not \leftarrow Prunable\_Not + cur\_v$               |

表 3-9 K-Plex Binary 算法主过程和表 3-10 Binary 的 K-Plex 枚举算法简要介绍了 Binary 的 K-Plex 算法。Binary 的 K-Plex 算法中依然使用到了 3.1.2 节中使用到的最小度数集合结构。不同的是，在 K-Plex 算法中最小度数集合结构中的度数指的是  $rDeg$  与

cDeg 的和。每次取分割点，从最小度数集合结构中去度数最大的点( $rDeg+cDeg$  度数最大的点，亦即与结果集和候选集相邻最少的点，符合 Binary 算法中关于最小度数的点分割的预期)。判断一个 Candidate 是否能够形成 K-Plex 可以通过判断其中点的  $cDeg+rDeg < k-1$ 。对于以上两个需求最小度数集合都可以高效地适用，寻找分割点只需要从最小度数集合结构中选取最后一个度数关联的点集中任意一个点即可；判断是否是 K-Plex 可以通过最大的  $cDeg+rDeg$ （即最小度数结构中最大值）是否满足条件即可。Binary 的 K-Plex 算法中使用的剪枝条件也是 Not 集中存在一个点与 Candidate 和 Result 中所有节点都相邻。

在 K-Plex 问题中注意到 K-Plex 问题比完全图问题更加复杂，搜索空间是 K-Plex 最主要的性能瓶颈。事实上 Pemp 算法的剪枝策略并没有很好的提前剪枝，其搜索空间依然十分庞大，本文 Binary 的 K-Plex 算法继承了 Binary 算法中减少搜索空间以提高搜索效率的优势，在搜索空间上远小于 Pemp 算法，并在实验中得到了验证。

### 3.3 本章总结

本章主要详细介绍了极大完全图和极大近似完全图枚举的对比算法 BK 及其变种和本文提出的两个问题的 Binary 算法。在综合两者各自的优点之上提出了极大完全图的 Hybrid 算法。本章主要介绍本文所提出的算法的单机方案，单机算法的高效性是并行算法的基础。本章中算法都保证枚举出数据图中所有的有意义的且不重复的极大完全图或极大近似完全图，应用剪枝方法避免了冗余搜索路径的消耗和重复结果的出现。





## 4 Hadoop 平台下算法的并行化

本文旨在研究图算法以及其高效并行化。在并行化方面，本文采用分布式计算平台 Hadoop 使用 MapReduce 编程模型实现。并行算法的可用性和高效性主要通过算法的可分割性以及可均衡性来实现。算法可分割便可以并行化，算法可均衡负载则并行算法可以解决长尾问题，使得算法能够有很好的扩展性从而能够通过机器数目的简单叠加获得计算时间的降低，提高吞吐量。本文提出的三个算法都可以较好地满足可分割性和可均衡性。

对于可分割性：综合以上完全图和近似完全图的几个算法可以发现它们本质上都是对于图的一个树状搜索，在搜索树路径上任意一个子树的搜索过程都与其不相交的另一子树不存在任何依赖关系，因而可以较方便地将不同的子树分配到多个机器节点上并行地进行运算。另外，由于不相交子树间没有信息依赖，因而可以适用于 Hadoop 这类 Share-Nothing 的计算平台。

对于可均衡性：在算法真正遍历完搜索之前，并不能够从子图数据中准确计算单个子图的搜索代价，同时由于实际数据存在大量的计算倾斜，以上算法在实际的分布环境中都存在负载不均衡的问题。得益于分割后的子图的独立性，在 Hadoop 平台上，算法可以暂停当前任务，将剩余子图及新分割出来的子图通过 Hadoop 的 Shuffle 过程随机分发到各个计算单元上进行新一轮的计算，通过多轮作业迭代算法可以较好地均衡各节点间的计算负载。

此外，要提升并行算法的加速比，还需要尽可能降低算法并行化所带来的额外消耗。Hadoop 平台中各个 Slave 节点之间是异构、等价的，Slave 节点之间不直接交互作业信息。同时 Hadoop 系统中所有中间结果或者数据都通过磁盘进行持久化，并行算法需要考虑尽可能地降低磁盘读写数据，减少数据传输量。极大完全图枚举和极大 K-Plex 枚举在 Hadoop 并行化算法上具有相似的结构，包括读入输入图数据、初始任务分配、负载均衡和终止条件。

在讨论了算法并行化的初步实现后，由于子树的一次划分以及各个子树的搜索代价不可预估带来了负载倾斜问题。为了提升算法的扩展性以及实用性本文提出了基于 Hadoop 平台的完全图和近似完全图负载均衡方案。

### 4.1 输入图数据

在并行 Hadoop 程序中，输入图数据是在 Map 中完成的，Map 函数将输入数据按需求发送到各个 Reduce 端，在 Reduce 端进行完全图和近似完全图搜索工作。

对于极大完全图检测：传统检测算法中一般使用两跳数据集作为输入数据集，一个点的两跳数据可以保证找到包含这个点的所有极大完全图的正确性和完整性。引入两跳数据集的初衷是用来减少不相关点的干扰，提前筛去无意义点，但实际数据中两跳数据

集的筛选能力较差，依然导致大量无效的中间输出。注意到所有有实际意义的极大完全图（组成极大完全图的点的个数不小于 4 的完全图）都是由多个三角形构成的。本文采用文献[3]中的方法挖掘出输入数据图中所有的三角形，极大完全图挖掘的输入数据是一行行的三角形数据，每行三个点 $\langle A, B, C \rangle$ 。对于每个三角形 $\langle A, B, C \rangle$ ，map 函数按 Key 值将其分别映射到三个点上输出， $\langle A, \langle B, C \rangle \rangle$ 、 $\langle B, \langle A, C \rangle \rangle$ 、 $\langle C, \langle A, B \rangle \rangle$ 。这样在每个 Reduce 端都可以接收到包含一个点的所有三角形(比如点 A，将接收到 $\langle A, \langle B, C \rangle \rangle$ 、 $\langle B, D \rangle$ 、 $\langle C, D \rangle$ 等)。三角形数据同时包含了检测一个点的完全图所需的边连接信息，Reduce 端将除当前点外的一条边，构建成邻接表结构（即将边 $\langle B, C \rangle$ 、 $\langle B, D \rangle$ 、 $\langle C, D \rangle$ 组成邻接结构）。表 4-1 并行极大完全图算法 Map 过程描述了在 Hadoop 平台上算法的 Map 过程。

表 4-1 并行极大完全图算法 Map 过程

| 输入数据 Map                                 |  |
|--|--|
| 输入：三角形数据 $\langle A, B, C \rangle$ ，每行一个 |  |
| 1.                                       | For Each $\langle A, B, C \rangle$                         |
| 2.                                       | Context.Write( $\langle A, \langle B, C \rangle \rangle$ ) |
| 3.                                       | Context.Write( $\langle B, \langle A, C \rangle \rangle$ ) |
| 4.                                       | Context.Write( $\langle C, \langle A, B \rangle \rangle$ ) |

对于极大近似完全图检测，需要指出的是在近似完全图的定义中并没有对邻接性作要求，比如两个独立的三角形整个的来看也可以形成一个 K-Plex( $k>3$ )，事实上这样的近似完全图并没有应用价值。K-Plex 应当视作为现实环境中完全图要求过于严格而进行的一种条件弱化，因此本文规定所找出的 K-Plex 都是一个连通图，这样的结果具有实际意义。根据在第一章中介绍的文献[35][36]对于近似完全图的理论研究，一个点的两跳数据集可以满足检测出包含这个点的所有有意义的 K-Plex。在其 Map 中只需要根据输入数据的所有节点将其分发到各个 Reduce 上，在 Reduce 端各自从本地文件中读入并构建两跳数据集。

## 4.2 初始任务分配

在单机算法中尽管搜索树中处于同一层的节点间相互独立，满足可分布的特性，但初始搜索状态中所有点都是候选点，直接将这样的结构发散到各个计算节点上必然会带来大量的重复计算。另外，单机算法的搜索树中根节点到第一层子节点的扩展是集中计算的，不适合 MapReduce 编程模型 Share-Nothing 的特征，也是并行算法切分任务的基本部分。BK 算法搜索树节点由根节点扩展到第一层子节点时，通过最大度数点和与其不相邻的点来分割，这有可能导致分割出来的第一层子图个数过少从而导致单个计算节点的负载过高。Binary 算法搜索树由根节点扩展到第一层子节点时，可以较好的均匀各个子图负载，但一次分割耗时过多且需集中分割。

考虑到并行系统的分布特性及 MapReduce 的模型要求，本文初始任务分配选择每

一个顶点作为初始任务的分割点。也就是说对于每一个顶点都在第一层生成该节点的搜索子图。Hadoop 中每一个计算单元 Slot 都有一个唯一的 ID (MapTask\_ID 或者 ReduceTask\_ID)，Map 任务可以通过节点 ID 和 Slot\_ID 判断这个节点的计算任务需要发送到个 Slot 上进行计算，这样任务就可以无需集中切分以实现第一搜索树节点的发散。

按照节点 ID 和计算单元 Slot\_ID 发散任务过程中需要考虑搜索树状态节点的数据集的完整和正确性。包含节点  $v$  的所有 Clique 中所有点都应该在  $v$  的邻接表中，三角形数据可以完整地复原一个顶点的相邻节点的邻接表状态。节点  $v$  的第一层子树状态中只有一个结果集 Result 的元素即  $v$  本身。Not 集的含义是之前已经搜索过的节点，但是在分布的算法中无法获知哪些节点已经搜过了。因此分布算法中各个节点的搜索次序需要保持逻辑上的全序  $v_1 \prec v_2 \prec \dots \prec v_n$ ，即逻辑上认为先搜索  $v_1$  点的子树再搜索  $v_2$  的子树...最后搜索  $v_n$  的子树。这样按照 Candidate 和 Not 的定义，对于一个点的邻接点集合，比该点 ID 值大的点认为没有搜索过加入到 Candidate 集中；比该点 ID 值小的点认为已经搜索过了，加到 Not 集中。至此，各个计算单元可以独立地并行搜索各个子图。

类似于前文中关于 Binary 切分点选择的问题，对于第一层子树各个点的全序可以多种设计方式，本文选取最简便直接的点 ID 大小。对应于 BK 和 Binary 的分裂点设计思想可以有各自的按点度数的全序。按 BK 的设计思路按照先搜索度数大的点在搜索度数小的点，这样可以使得度数大的点更可能出现在度数小的点的 Not 集中。同时，度数大的点期望邻接的候选点也会更多，这样就可以使得小度数点的子图更容易达到剪枝条件，从而加快搜索速度。反之，按照 Binary 的设计思想应先搜索度数小的点从而减少各个 Size-1 子图的搜索树大小。这两种方式都要求预先处理输入数据，计算出各个点的度数大小，并且将结果在全局发布。这限制了算法的分布性和适用性，同时带来了额外数据处理代价。综合考量，本文最终选择简便的 ID 作为全序设计标准。表 4-2 并行算法的 Reduce 过程描述在 Hadoop 平台上并行算法的 Reduce 工作。

表 4-2 并行算法 Reduce 过程

| 初始任务分配 Reduce   |  |
|---|--|
| 输入数据：三角形数据 $\langle A_1, \langle \langle B_1, C_1 \rangle, \langle D_1, E_1 \rangle \dots \langle F_1, G_1 \rangle \rangle \rangle, \langle A_2, \langle \langle B_2, C_2 \rangle, \langle D_2, E_2 \rangle \dots \langle F_2, G_2 \rangle \rangle \rangle, \dots \langle A_n, \langle \langle B_n, C_n \rangle, \langle D_n, E_n \rangle \dots \langle F_n, G_n \rangle \rangle \rangle$ |  |
| 1.  | For Each $\langle A_i, \langle \langle B_i, C_i \rangle, \langle D_i, E_i \rangle \dots \langle F_i, G_i \rangle \rangle \rangle$      |
| 2.  | 构建三角形除当前顶点 $A_i$ 之外第三条边 $\langle B_i, C_i \rangle, \langle D_i, E_i \rangle \dots \langle F_i, G_i \rangle$ 构成的邻接表 $Graph_i(V_i, E_i)$ |
| 3.  | $Result \leftarrow Result \cup \{A_i\}$  |
| 4.  | For $v \in V_i$  |
| 5.  | If $v > A_i$   |
| 6.  | $Candidate \leftarrow Candidate \cup \{v\}$  |
| 7.  | Else   |
| 8.  | $Not \leftarrow Not \cup \{v\}$  |
| 9.  | EnumAllMaximalClique(Candidate, Result, Not)   |

初始任务分配完成以后，各个 Reduce 的 Slot 各自运行单机算法执行搜索过程，直到所有 Slot 都完成各自的搜索任务后整个作业完成。K-Plex 的初始任务分配方法与 Clique 的思路基本一致，不再赘述。

### 4.3 负载倾斜问题

在上一节中分布算法只对初始任务进行分散划分，每一个 Reduce Slot 处理一棵 Size-1 的搜索子树。但是由于在处理这棵子树之前无法通过底层子节点的状态集合以及图的邻接表计算出这棵子树的搜索代价，因此无法通过预先制定分散策略使得各个 Slot 的负载相对均衡。在实际数据的处理过程中观察到，大多数现实数据存在着严重的负载倾斜问题。社交网络数据是一个明显的实例，一些名人的节点虽然数目少，但是由于名人效应他们的关注者非常多。这些点都连接着数十万乃至上百万的点，其搜索代价也远大于其他普通节点。

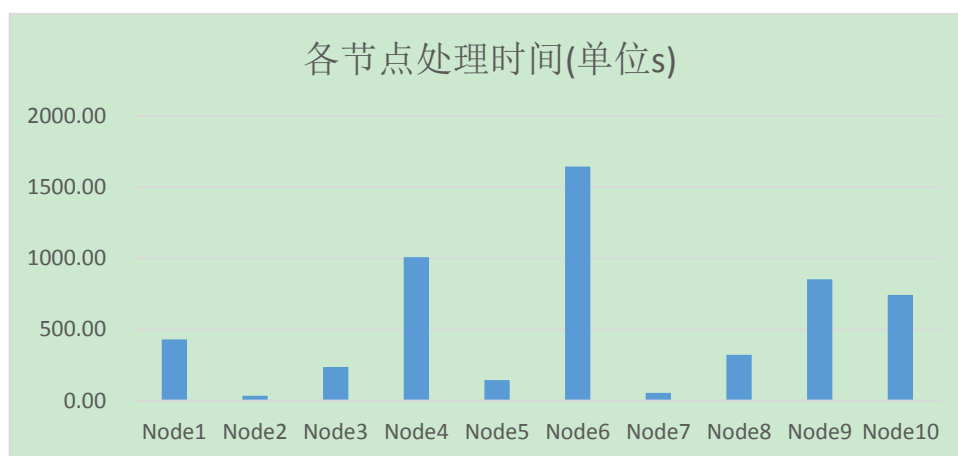


图 4-1 各机器节点处理时间对比

上图 4-1 是 Twitter 数据集的一个样本各个 Size-1 子树的搜索代价在各个计算节点上的分布情况，可以看出大部分 Reduce Slot 可以在很少的时间内搜索完成，但是少量的却需要很长时间才能计算结束。显然，将负载直接分散到各个计算单元而不做负载均衡处理未能完全发挥集群的计算能力，出现了长尾现象。同时由于单棵搜索树的瓶颈效应，即使向集群中添加更多的机器，也不能够带来处理效率的有效提升，阻碍了算法的可扩展性。因此进一步的负载均衡势在必行，下一节将会详细讨论完全图和近似完全图算法的负载均衡问题。

### 4.4 完全图和近似完全图枚举的负载均衡

上一节中根据 Size-1 子树之间的独立性，通过对 Size-1 子树的发散获得了一定的并行性。在上一节发散 Size-1 子树方案的基础上，注意到整个搜索树中，每个节点的子节点之间都可以相互独立计算。因而可以在子图切分过程中将部分子图保存暂时不予搜索，在下一步中再发散到其他计算单元上进行计算。使用多个 Hadoop 作业的多轮迭代将计

算任务在各个计算节点之间发散,从而获得更好的均衡性。在 Hadoop 中,各个 Slave 之间不能够直接交互,因而需要将临时保存的子图在下一个作业中 Shuffle 到其他节点上。通过 Hadoop 的 Shuffle 过程发散子图需要涉及到存储子图到磁盘和启动新作业发散数据两个过程。这两个过程需要磁盘 I/O 以及网络 I/O,它们相对于 CPU 的运算来说都是非常耗时的。这就出现了两个矛盾的因素:

- a. 尽快停止当前作业的 Slot 搜索过程,发散子图,避免任何 Slot 空闲
- b. 充分发挥 CPU 的计算能力,尽力搜索一个子图,避免启动新任务的开销

因素 a.的最理想状态是在任意一个 Reduce Slot 处理任务完成时立即停止当前作业,发散任务以避免系统中出现空闲的计算单元。在 MapReduce 的同步计算模型限制下,各个 Reduce Slot 负载处理已分配的任务,截止条件不依赖于其他 Slot,这种方式只是一个理想情况。因素 b.希望尽可能减少启动新任务迭代的开销,其极端情况就是第三章中提出的只有一次分散任务的并行方法。综合两个因素考量,其关键问题在于何时停止当前子图搜索以及何时开始新一轮任务发散。

表 4-3 并行负载均衡 Reduce 过程

| 第一类作业 Reduce             |  |
|--------------------------|--|
| 输入                       | : 三角形数据 $\langle A_1, \langle \langle B_1, C_1 \rangle, \langle D_1, E_1 \rangle \dots \langle F_1, G_1 \rangle \rangle \rangle, \langle A_2, \langle \langle B_2, C_2 \rangle, \langle D_2, E_2 \rangle \dots \langle F_2, G_2 \rangle \rangle \rangle, \dots \langle A_n, \langle \langle B_n, C_n \rangle, \langle D_n, E_n \rangle \dots \langle F_n, G_n \rangle \rangle \rangle$ |
| Reduce.Setup             |  |
| 从配置文件读入 N, T; 获得本地临时输出文件 |  |
| Reduce.Reduce            |  |
| 1.                       | Time $\leftarrow 0$  |
| 2.                       | For Each $\langle A_i, \langle \langle B_i, C_i \rangle, \langle D_i, E_i \rangle \dots \langle F_i, G_i \rangle \rangle \rangle$  |
| 3.                       | 构建三角形除当前顶点 $A_i$ 之外第三条边 $\langle B_i, C_i \rangle, \langle D_i, E_i \rangle \dots \langle F_i, G_i \rangle$ 构成的邻接表 $Graph_i(V_i, E_i)$   |
| 4.                       | Result $\leftarrow$ Result $\cup \{A_i\}$  |
| 5.                       | For $v \in V_i$  |
| 6.                       | If $v > A_i$   |
| 7.                       | Candidate $\leftarrow$ Candidate $\cup \{v\}$  |
| 8.                       | Else   |
| 9.                       | Not $\leftarrow$ Not $\cup \{v\}$  |
| 10.                      | If Time $< T$  |
| 11.                      | Time += ComputeOneSubgraph(Candidate, Result, Not, Time)   |
| 12.                      | Else   |
| 13.                      | SpillGraphToDisk(Candidate, Result, Not)   |

为解决以上两个问题本文引入两个参数 N 和 T。参数 N 指切分出一个子图 G 后如果 G 中的候选点个数大于 N 则不再深度遍历这个子图而直接将其存到本地磁盘,以待下一次 Shuffle 到其他 Slot 计算;反之,直接将子图 G 的所有路径搜索完成找到所有可能的结果。参数 N 的设计基于一个假设:候选节点越多的子图其计算量可能越大。参数 N 的设置可以避免花费大量时间来计算同一棵子下的节点,而导致其他节点得不到处理,以至于发散任务后计算量还是无法分散到各个节点上。参数 T 指单个 Reduce Slot 运行

时间累计到达  $T$  时不再进行子图搜索枚举结果的工作，转而做子图切分工作，以便可以在下一个作业中发散未计算子图到其他节点。参数  $T$  的设置可以避免单个节点负载重的情况下一直计算而无法进行同步发散数据的问题。

在写本地临时文件中存在三种格式的数据边邻接信息，子图信息和未切分的 Size-1 子图信息。对于这三种类型的数据本文规定如下输入格式：

边邻接信息：-2 \t 1#p1,p2...pn#rootKey#edge

子图信息：-1 \t 1 @p@visitingPoint%level%candidate%not%result

Size-1 子图信息：-1 \t 0@p@ rootKey@candidate%not%result@edge

其中第一个值取-1 或者-2 用来区分边邻接信息和子图信息；第二个值取 0 或 1 用来将数据在下一次迭代中利用 Hadoop 默认的排序功能使得未切分过的 Size-1 子图可以排到前面尽快得到切分。对于边邻接信息第三个域值表示这个邻接信息相关的子图发送给了下一步的哪些 Reduce，也就是邻接信息需要发送到哪些 Reduce 上，这样相对全局发送可以减少通信数据冗余。邻接信息第四个域值表示边邻接信息的根节点也就是三角数据中公共的点，最后是实际的三角形数据中第三条边形成的邻接表。子图信息中第三个域值表示这个子图在下一步将要被发送到哪个 Reduce 上，接着是当前切分子图正在访问的节点。然后子图信息都包括了候选集、not 集以及结果集。与一般子图不同的是，Size-1 子图由于从未切分过，其边邻接信息也只在它本身需要，因而直接将该子图的边邻接信息加在 Size-1 子图信息最后。

有负载均衡的 MapReduce 实现分为两类作业，第一类作业是从原始输入数据进行搜索工作，第二类是从第一类作业以及第二类作业本身未处理完的切分出来的子图作为输入数据进行完全图和近似完全图挖掘工作。第一类作业的输入 Map 处理与上一节中的方式相同，Reduce 的算法逻辑如表 4-3 所示。与第一类作业不同第二类作业的输入数据是第一类作业输出的临时文件，因而其输入的 Map 过程也存在不同。Map 过程如表 4-4 所示，各个 Map 从本地读入上一步的输出文件解析，分子图信息和边邻接信息两类分别做 Map。

在 Map 中将信息类型、所要发送到的 Reduce 以及当前数据的访问节点三个信息组合为 Map 的输出键 PairInt<Type,Partition,rootKey>。对于组合键，需要定制分区函数和键值比较方法以保证发送到 Reduce 端的数据的完整性和有序性。Partition 函数提取组合键 PairInt 中第二域值 Partition 并使用 Hash 函数将其映射到 ReduceNum 个 Reduce 上。这样对于发送到同一个 Reduce 中的数据可以保证完整性。算法为 PairInt 定制了比较函数，首先比较 PairInt 的第一个域值 Type，保证未切分过的 Size-1 子图可以获得较高的处理优先级，尽快得到切分。然后比较 PairInt 的第三个域值 rootKey，使得同一个 rootKey 的数据可以集中到一起，从而使得同一个 rootKey 的子图数据和边邻接信息得到聚集，这样使得输入数据同时保证的正确性和有序性。

表 4-4 并行极大完全图算法第二类作业 Map 过程

| 第二类作业 Map 过程  |   |
|---------------|---|
| 输入：各个机器本地输出文件 |   |
| 1.            | For each 输入文件的每行信息  |
| 2.            | If 是子图信息  |
| 3.            | 将第 2,3,4 域值构建成 Map 的 Key:<Type, Partition, rootKey>,剩余部分 Rst 为值 |
| 4.            | Context.write<<Type,Partition,rootKey>,Rst>                     |
| 5.            | Else 是边邻接信息   |
| 6.            | 提取其要发送到的 Reduce 号 parts[]和边邻接信息 graph                           |
| 7.            | For each part in parts  |
| 8.            | Context.write<<Type,part,rootKey>,graph>                        |

表 4-5 单个子图处理流程

| 方法 ComputeOneSubGraph        |                                     |
|------------------------------|-------------------------------------|
| 输入：Candidate,Result,Not,Time |                                     |
| 输出：找到的结果，输出子图文件，耗时           |                                     |
| 1.                           | tmpTime←Time                        |
| 2.                           | 将输入子图<Candidate,Result,Not>入栈 Stack |
| 3.                           | While Stack ≠ $\phi$                |
| 4.                           | Start←SystemTime                    |
| 5.                           | 取栈顶子图 curCandidate,curResult,curNot |
| 6.                           | 按单机算法的方式切分子图 G'                     |
| 7.                           | If tmpTime<T and $ G'  < N$         |
| 8.                           | 调用单机算法将 G' 计算完                      |
| 9.                           | Else                                |
| 10.                          | 将 G' 输出到新的临时文件                      |
| 11.                          | End←SystemTime                      |
| 12.                          | tmpTime+=End-Start                  |

第一类作业的 Reduce 首先在初始化时读入配置文件中的 T,N 等相关参数设置，获取该 Reduce 一个本地临时文件用作超时或者超大子图的临时写入区。然后读入 Map 发送过来的三角形数据，按前文中的方法构建 Size-1 子图状态，最后调用 ComputeOneSubgraph 来处理这个子图。当该 Reduce 的计算时间超过设定时间 T 时，构建出来的子图不再进行搜索计算，而是直接写到本地输出文件中。

在 ComputeOneSubGraph 中使用单机算法的切分方式切分输入的子图。对于切分出来的子图 G' 考虑是否超时和是否超大两个因素，只有当前 Reduce 的计算时间未超时且切分出的子图 G' 未超大小才调用单机算法直接将这个子图搜索完成，其他情况下都先将 G' 写入到本地临时输出文件中，待下一步处理。

还有另外一种情况，当该 Reduce 的计算负载比较小，可能未达到时间阈值 T，Reduce 的搜索工作就已经完成。如果停止工作等待其他 Reduce 完成再进行下一步的处理就浪费了本地的处理能力。为充分利用计算资源，本文负载均衡过程中利用 Reduce 的清理

工作函数 **CleanUp** 利用剩余的工作时间，将 **Reduce** 中输出的超大文件读入到内存中处理。这样在一个 **Reduce** 的执行过程中除非本地的所有任务都已经计算完成，**Reduce** 在规定的的时间阈值 **T** 内都会一直处于计算状态，从而提高了系统的计算资源利用率。

表 4-6 并行负载均衡 **Reduce** 清理过程

|   |
|---|
| <b>Reduce.CleanUp</b>   |
| 输入：本地临时输出文件   |
| 输出：搜索结果，新的本地临时输出文件  |
| <ol style="list-style-type: none"> <li>1. 获取新的输出文件</li> <li>2. 从本地输出文件中读入一个子图 <math>G(\text{Candidate}, \text{Result}, \text{Not})</math></li> <li>3. If <math>G = \text{Null}</math></li> <li>4.     本地所有任务都已计算完成，退出</li> <li>5. Else If <math>\text{Time} &gt; T</math></li> <li>6.     将原输出文件中未计算的部分直接拷贝到新输出文件后，退出</li> <li>7. Else</li> </ol> |
| $\text{Time} += \text{ComputeOneSubgraph}(\text{Candidate}, \text{Result}, \text{Not}, \text{Time})$  |

和两个因素  $a, b$  之间的矛盾一样，时间参数 **T** 和子图大小参数 **N** 之间也存在着权衡和制约。当 **T** 过小时，一次作业能够处理的任務减少，增加了任务数，带来大量读写文件及启动作业的代价；当 **T** 过大时每个 **Slot** 都可以充分计算本地任务，但是有些任务少的机器的计算资源被闲置了。当 **N** 过小时，很少有子图能够在本地计算掉，需要等到下一步的 **Shuffle** 发散，带来了网络传输量的增大；当 **N** 过大时，由于深度搜索会出现一直在计算前几个子图而其他子图得不到切分，计算量不能够有效发散的问题。

可以预期的是 **T** 和 **N** 的取值存在一个中间的较优情况，这一点也在后续的实验中得到验证。本文的分布实验过程中都是选择的 **T**、**N** 的较优情况作为参数配置。与一次任务发散的分布算法相比，负载均衡的分布算法表现出更优的性能和扩展性。

本章的负载均衡算法可以应用于第三章中所描述的 **BK** 完全图枚举、**Binary** 完全图枚举、**Hybrid** 完全图枚举、**Pump K-Plex** 枚举以及 **Binary K-Plex** 枚举。这里只给出了完全图的描述，其余 **K-Plex** 等分布算法的负载均衡类似，不再赘述。

## 4.5 本章总结

本章介绍了针对分布式极大完全图和近似极大完全图的并行化方案，保证了并行算法的正确性和有效性。同时对于一次任务分配所带来的负载倾斜问题提出了一种负载均衡方案，通过 **MapReduce** 作业的多次迭代，在各个计算单元上从搜索树的任意位置切分均衡负载。根据 **Hadoop** 作业特性在极大完全图和近似极大完全图两个算法中引入 **T** 和 **N** 两个参数，充分利用每一个计算单元的计算能力以提高整体作业的扩展性和均衡性。



## 5 实验部分

本文选取多组实际数据以及 SSCA 和 RMAT 两种特征可控数据对文中所提出的算法和做了充分的实验。单机算法验证了本文所提出的图分割算法高效性，并行实验验证了负载均衡策略的有效性。实验数据佐证了算法设计中关于切分点选择的讨论。

### 5.1 实验条件及数据

#### 5.1.1 实验条件

软件环境：Ubuntu10.04，JDK1.6，Hadoop-1.1.2。开发环境：Eclipse3.6，JDK1.6。硬件环境：10 台 1.87GHz 的 4 核 Xeon E5502 CPU，16G 内存，160G 硬盘的曙光系列服务器。

#### 5.1.2 实验数据

表 5-1 实验数据集

| 数据集    | 数据集描述                 | 顶点个数              | 边数目        |
|--------|-----------------------|-------------------|------------|
| D1     | EU 研究机构的邮件网络          | 265,009           | 364,481    |
| D2     | 谷歌 Web 图              | 875,713           | 4,322,051  |
| D3     | Berkeley&Stanford Web | 685,230           | 7,600,595  |
| D4     | Wikipedia 通讯网络        | 1,928,669         | 3,494,674  |
| D5     | Pokec 在线社交网络          | 1,632,803         | 30,622,564 |
| D6     | Twitter 社交圈           | 11,316,811        | 85,331,846 |
| D7     | 编号 4952 蛋白质交互网络数据     | 5,099             | 681,252    |
| D8     | 编号 568206 蛋白质交互网络数据   | 5,816             | 313,628    |
| D9     | 编号 329726 蛋白质交互网络数据   | 8,176             | 457,991    |
| R-MAT  | 符合幂律分布和小世界特性的合成图      | 点和边数目可设置          |            |
| SSCA#2 | 基于距离度量的分层团边分布         | 点数和最大 Clique 大小可设 |            |

表 5-1 列举了本文实验中所使用到的实验数据图的详细信息及其点边数目特征。D1-D9 是真实的网络数据，包含了社交网络、Web 网络以及生物数据等多个方面。其中数据 D6 由于 Twitter 的规模超出了单机以及本文中实验集群的计算能力，因此在实验中对其采样抽取部分子图进行实验。在实际生产应用过程中小于 4 个节点的 Clique 和 K-Plex 结果基本没有使用价值，因此本文的实验中默认枚举出的完全图和近似完全图节点数不小于 4。

### 5.2 单机算法实验及分析

单机算法实验使用较小的数据集 D1-D5，在一台服务器上从搜索树规模和搜索时间两个方面对比 BK 及其变种与 Binary 算法在极大完全图枚举和极大近似完全图枚举两

个问题上的表现。利用数据属性可控的模拟数据 R-MAT 和 SSCA#2 控制数据的点边比和最大完全图大小对比两类算法的适应性。最后对 Binary 算法中分裂点度数选择策略，最大、最小和随机三种方式的结果做了验证。

### 5.2.1 极大完全图枚举

极大完全图枚举以及极大近似完全图枚举单机算法的输入数据都是每行一条<A,B>形式的无向边信息。默认地，无特别说明 Binary 算法中切分点选择策略都是使用的最小度数切分。关于 Binary 算法对分裂点选择的敏感性实验在 5.2.3 节中有详细描述。



图 5-1 极大完全图枚举

在图 5-1 中可以看到 Binary 的搜索树大小远小于 BK 算法，Binary 算法的搜索树大小一般在 BK 的一半左右。总的来说可以从图 5-1 中看出，极大完全图枚举中 Binary 算法比 BK 算法在运行时间上有一定的优势。这是因为 Binary 算法过程中需要保证每次选取最小度数点作为切分点需要维护最小度数集，每次切分过程的代价会高于 BK 一次选择全部切分的策略，但是由于 Binary 在搜索树大小上占有绝对优势总体运行时间上还是 Binary 算法占优。

图 5-1 中的实验结果使用 D1-D5 五个较小的实际数据验证了 Binary 和 BK 搜索树大小和运行时间，另外本文还使用 RMat 和 SSCA#2 两种属性可控的数据集来测试 Binary 算法和 BK 算法在数据特性变化时的适应性。图 5-2 中显示了 RMat 数据在点边比变化时 Binary 和 BK 极大完全图枚举算法的搜索树大小和运行时间表现。RMat 数据中下表数据 RmatX-Y，X、Y 单位为万，表示 Rmat 生成数据时设定生成的数据图中点的个数为 X 万个边的条数为 Y 万条，运行时间纵坐标单位为秒。图 5-2 的实验中测试了 100:500 到 100:5000（即点设定为 100 万，边的数目设定由 500 万增加到 5000 万）六个区间点边比 RMat 数据 Binary 和 BK 算法的性能表现。从图 5-2 中可以看出在 Rmat 数据上在边点比较小时 Binary 算法和 BK 算法的性能相差不大，而随着边点比的不断增大，Binary 算法比 BK 算法的性能优势越来越明显。同时 Binary 算法比 BK 算法的搜索

树大小优势得到了一致的保持。

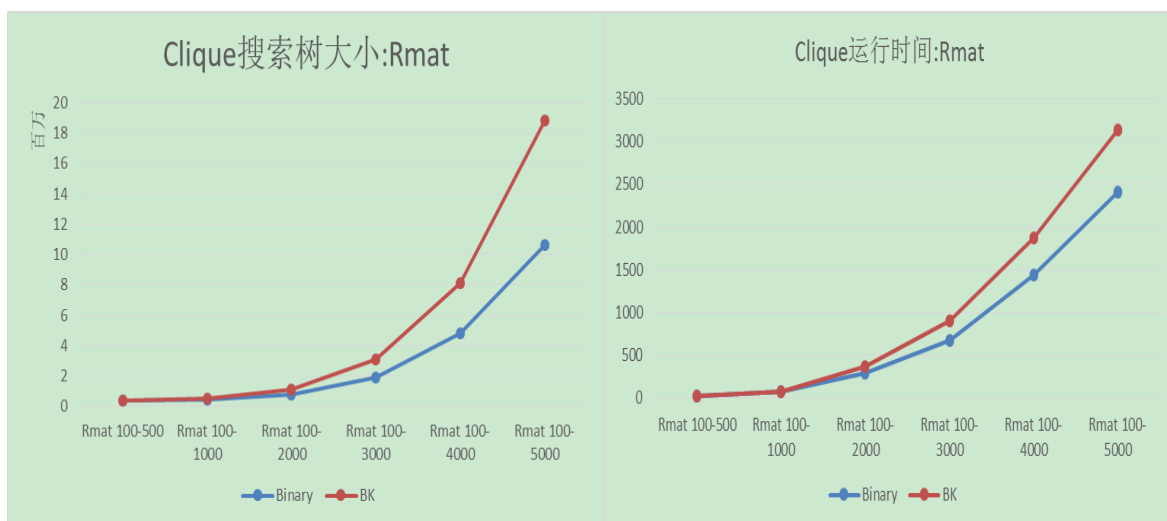


图 5-2 RMat 数据特性变化极大完全图枚举算法性能对比实验

图 5-3 中选用 SSCA#2 数据设定 SSCA#2 生成测试数据时点的个数为  $2^{20}$ ，控制数据中最大完全图大小由 20 一直增大到 100 五个区间点时 Binary 和 BK 算法的搜索树大小和运行时间表现。从图 5-3 的实验可以看出，Binary 算法的搜索树大小都要小于 BK 算法。在 SSCA#2 数据集上 Binary 算法搜索树大小一直稳定在 BK 算法搜索树大小的二分之一左右。从运行时间上看，与 RMat 一样 SSCA#2 中最大完全图大小增加，也就是在数据变稠密的情况下，Binary 极大完全图枚举算法比 BK 极大完全图枚举算法优势更加明显且稳步提升。

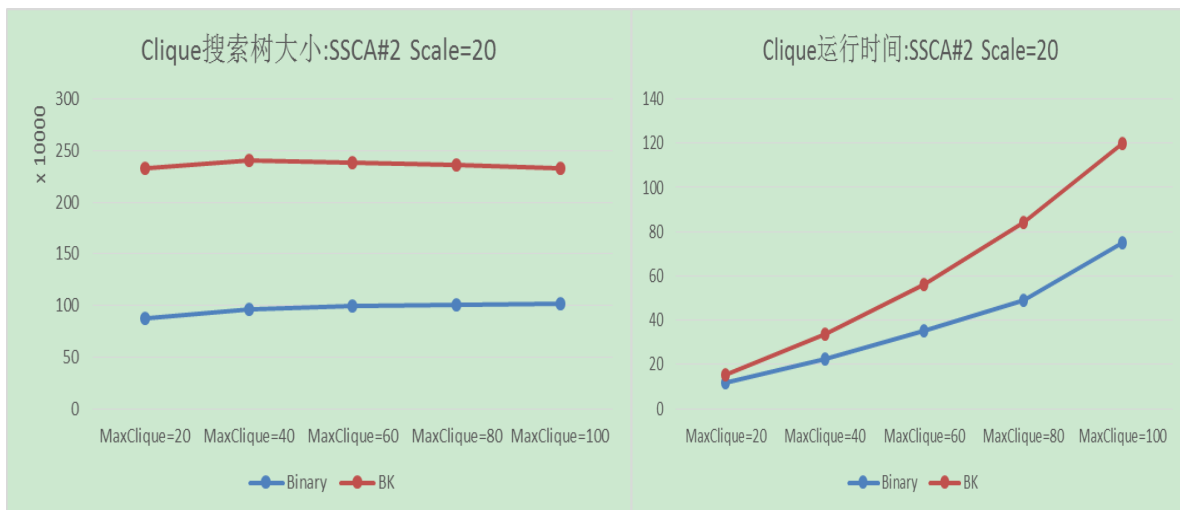


图 5-3 SSCA#2 数据特性变化极大完全图枚举算法性能对比实验

从整体上看，无论是实际数据还是生成数据，Binary 算法在运行时间以搜索树空间上都要优于 BK 算法，但是有一类数据蛋白质交互网络数据 D6-D8 时出现了反例。如下图 5-4 所示 D6-D8 中，Binary 算法的运行时间都要远大于 BK 算法。通过对数据的分析，出现这样的结果是因为蛋白质交互网络中存在大量的点与大部分点都相邻，给

Binary 的切分带来了极大的代价。通过上文介绍的 Hybrid 算法的优化，从图 5-4 中可以看出 Hybrid 算法在全部数据集上都有不错的表现，Hybrid 算法处于 Binary 和 BK 之间或者比两者都优，只有在 D3 比两者略高。

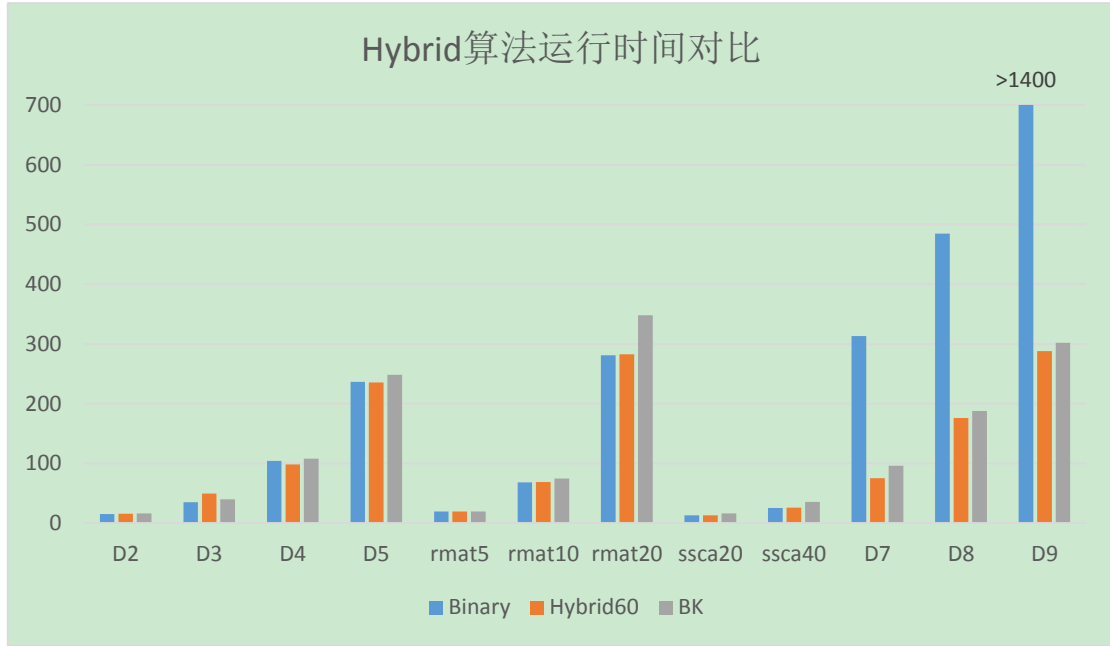


图 5-4 Hybrid 算法与 Binary 和 BK 算法对比

### 5.2.2 极大 K-Plex 枚举

极大近似完全图枚举的计算复杂度高于完全图枚举且随着 K-Plex 中 K 值的增大其运算复杂度也越来越大，本文选择 K=2 的 2-Plex 作为测试问题。枚举出数据集 D1-D5 中的所有近似极大完全图超过了单机计算能力或可容忍时间，因此本文选取 10% 的 D1，0.1% 的 D2-D5 中的点来其在全局图中的 2-Plex。基于同样的原因，2-Plex 的实验数据也减少了规模，点边比由 1 万:10 万到 1 万:30 万。

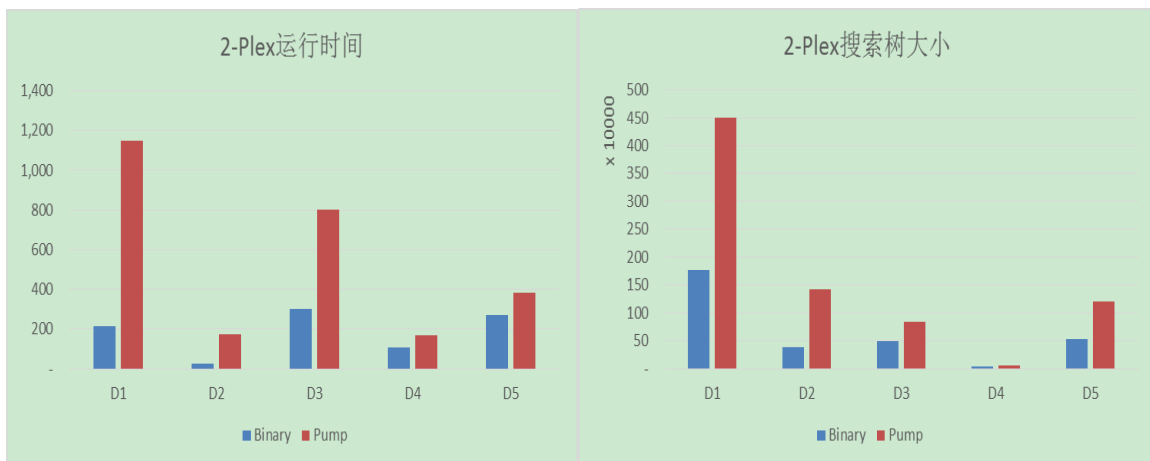


图 5-5 2-Plex 在实际数据中的性能实验

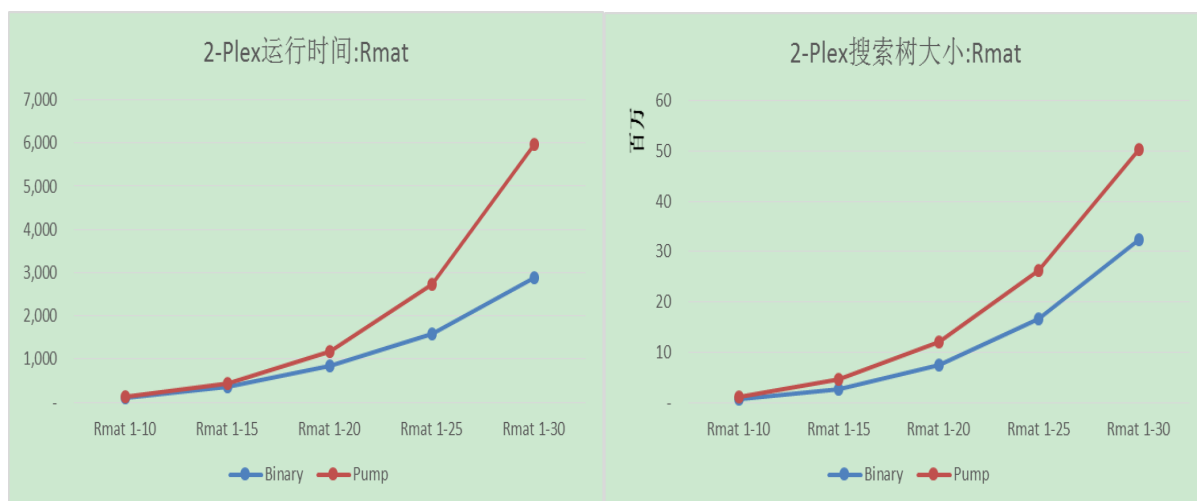


图 5-6 2-Plex 在 RMat 数据中的性能实验

从图 5-5 中可以看出在 D1-D5 上 2-Plex 的枚举过程中 Binary 算法的搜索树大小和运行时间都有明显优于对比算法 Pump。从图 5-6 中可以看出，与完全图一致，随着边点比的增加，Binary 算法的优势也是在逐步显著。

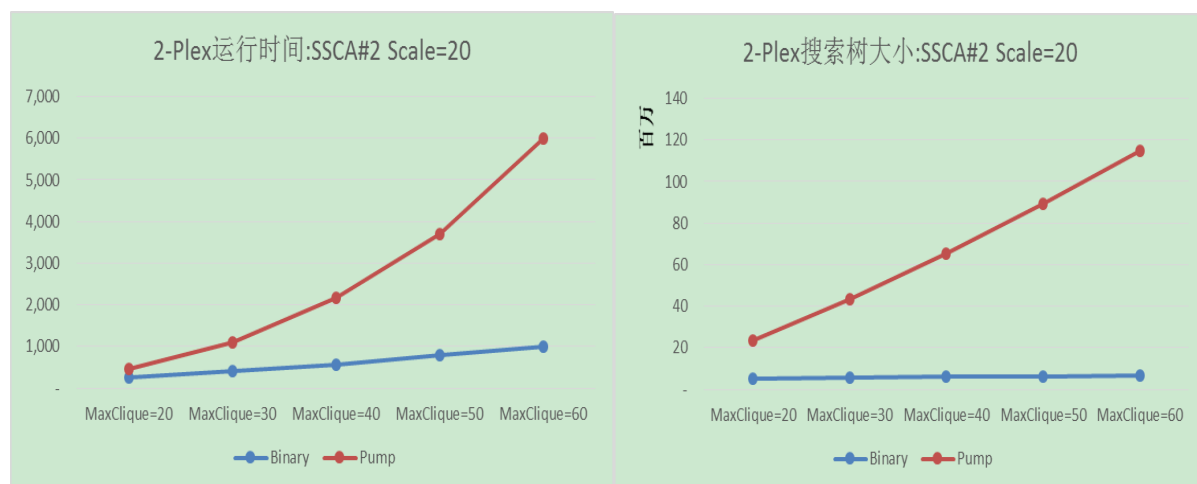


图 5-7 2-Plex 在 SSCA#2 数据中的性能实验

图 5-7 显示了在 SSCA#2 数据集上进行 2-Plex 枚举，设定图的规模为  $2^{20}$ ，逐步增加图中最大完全图的大小时 Binary 算法和 Pump 算法在搜索树和运行时间上的对比趋势。可以看出 Binary 算法的运算时间和搜索树大小同样明显优于 Pump 算法。另外可以观察到，在 SSCA#2 数据集上，Binary 算法对最大完全图的大小变化不敏感，在最大完全图大小增加时，Binary 算法的搜索树和运行时间增加很小或基本保持不变。而 Pump 算法的运行时间和搜索树大小随着最大完全图大小的变化呈线性相关。

### 5.2.3 分裂点选择

Binary 算法中关于分裂点选择的最大度数、最小度数和随机度数选取三种策略，经过实验验证了正文中相关分析。一般地，最大度数切分的效果最差，其次随机，最好的是使用最小度数切分。因此文中的其他 Binary 实验中都选取最小度数切分策略。K-Plex

中关于分裂点选择的问题有相似的结果，不再赘述。

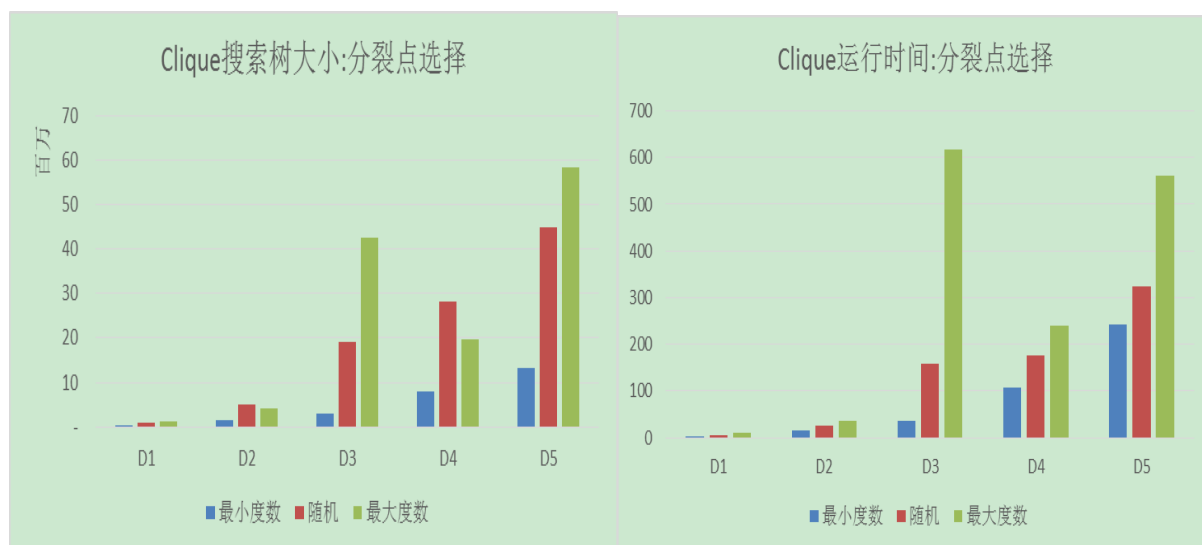


图 5-8 Binary 算法分裂选择策略

### 5.3 并行算法实验及分析

在并行算法实验中本文将 Binary 算法与 BK 的变种算法 Pump 做对比，由于所有的算法都使用同样方法获取同样的输入数据图，这样排除了由于数据图获取或输入使用的数据图带来的影响。并行极大完全图中使用三角形数据作为输入数据，并行极大近似完全图中使用两跳数据作为输入数据。并行算法中由于用户一般关注运行时间且作业启动及准备时间对 MapReduce 计算模型有较大影响，本文主要考量总运行时间和作业迭代次数两个指标衡量算法的性能。

#### 5.3.1 并行极大完全图枚举

在并行算法部分本文引入了两个参数  $T$ 、 $N$ ，两个参数的设置直接影响到算法的整体效率。本文经过多组实验选取各种不同的  $T$ 、 $N$  组合，找到算法表现最好的一组参数作为算法的输入设置。图 5-9 显示了一个典型的实验结果，可以看出在  $N$  一定时，随着  $T$  的增大整体的运行时间先降低后升高，在  $T=300$  时取得最优值；在  $T$  一定时，随着  $N$  的增大，整体运行时间也是先降低后增大波动，在  $N=50$  时取得最优。本节中极大完全图枚举算法的设置中选取最优设置  $T=300S$ ， $N=50$ 。较小的一些数据 D1-D5 可以使用单机在很快的时间内处理，因此使用并行实验并不适用。另外 SSCA#2 的数据在一次发散后就已经取得了较好的均衡不存在负载均衡问题，这里也不再展示其并行结果。

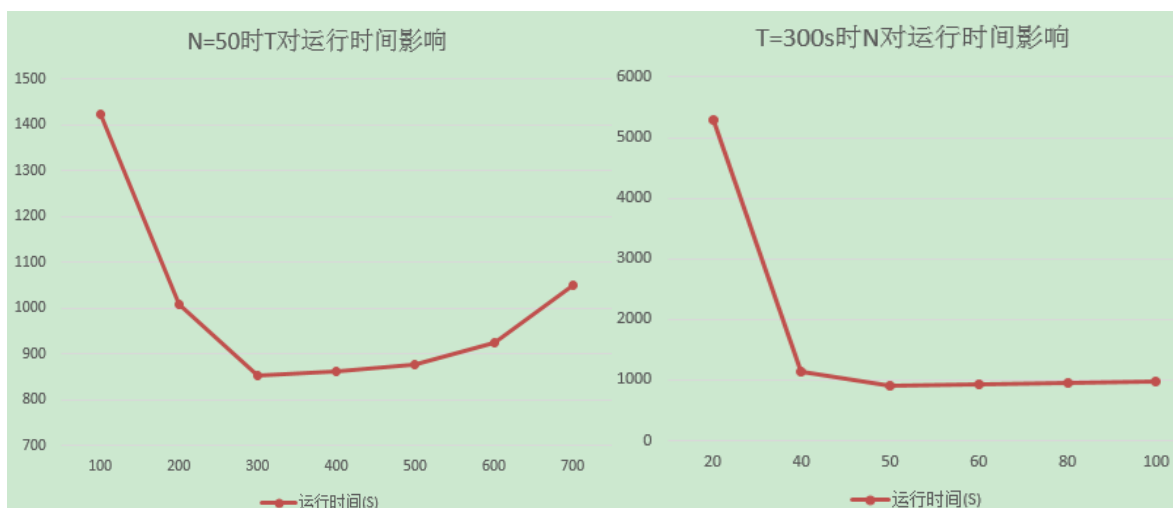


图 5-9 参数 T、N 设置对于算法性能影响

本文选取最大的实际数据集 D6 Twitter 中的部分采样点做为测试数据，在 5 个样本的实验中都显示了 Binary 并行化算法比 BK 并行化算法具有更少的处理时间以及更少的迭代次数。从图 5-10 中可以看出不采用负载均衡方案的情况下其运行时间都明显高于采用了负载均衡方案的两种方法，尤其是在样本 2 和样本 3 上其运行时间已经超过了 4 小时，从而进一步说明了负载均衡方案的重要性。

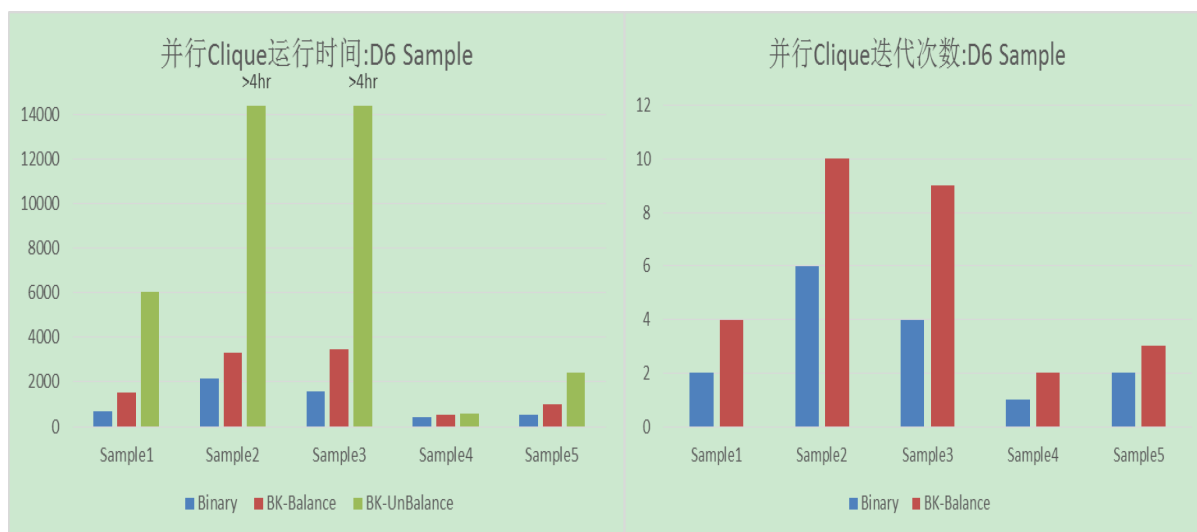


图 5-10 D6 的多个样本数据并行 Clique 实验

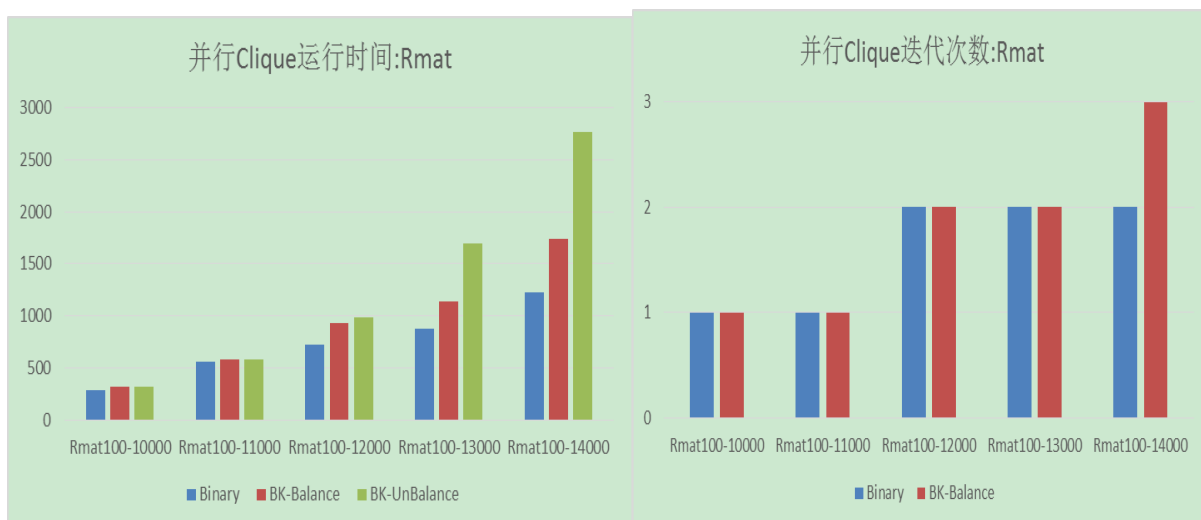


图 5-11 并行 Clique 在 RMat 变化时性能表现

为了验证并行算法在随着数据特征变化的情况下的适应性,本文再次增加了 RMat 数据图的规模,图 5-11 中点边比的单位是万,设定点数为 100 万,边数由 10000 万增大到 14000 万。可以看到随着数据规模和点、边点比的增加,并行 Binary 算法的优势逐渐体现。由于 SSCA#2 数据在 Binary 算法中可以很快速处理完成,无需并行环境下的验证,本文不作其并行讨论。

### 5.3.2 并行极大近似完全图枚举

由于 K-Plex 计算的复杂度非常高,并行算法部分依然采用  $K=2$  的 2-Plex 作为实验内容,另外实际数据 D3-D5 无法在本文的计算集群中全部处理,因此对数据集 D3-D5,本实验中随机取 1% 的点作为采样点,计算这些点的 2-Plex。与并行极大完全图枚举相似地, K-Plex 也需要将参数  $T$  和  $N$  设置为其最优情况,经过前期实验,在 2-Plex 的实验中  $T$  设置为 500s,  $N=50$  可以取得最优效果。可以在实验结果图 5-12 中看到,并行 2-Plex 实验中 Binary 表现出对比算法有极大的优势。

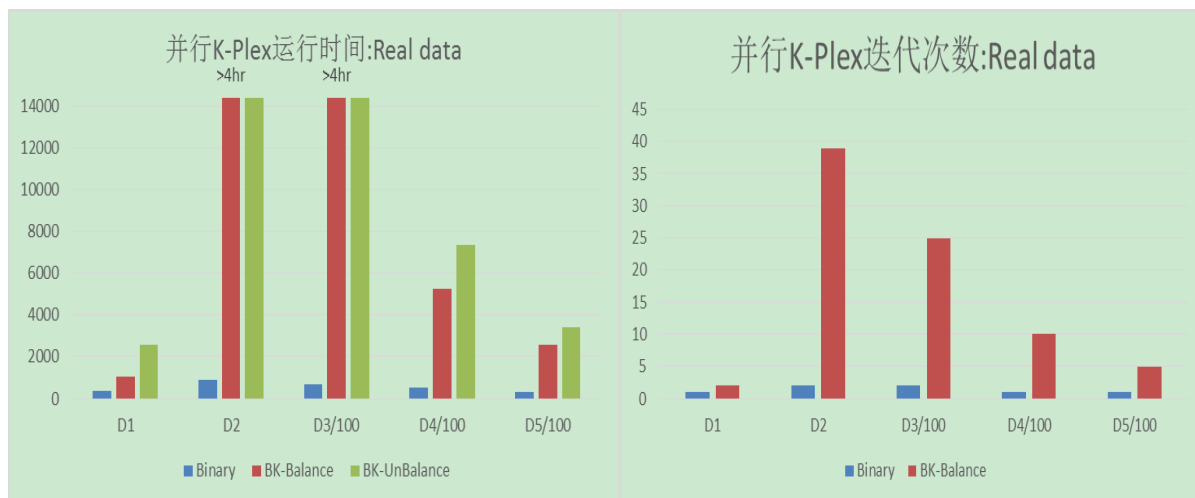


图 5-12 并行 2-Plex 在真实数据中性能表现



### 5.3.3 并行算法加速比

并行算法的加速比是并行算法可用性的一个重要衡量标准。本文实验中将集群机器数目由 2 增加到 10，观察 Binary 算法在各种数据上的集群的加速比情况。

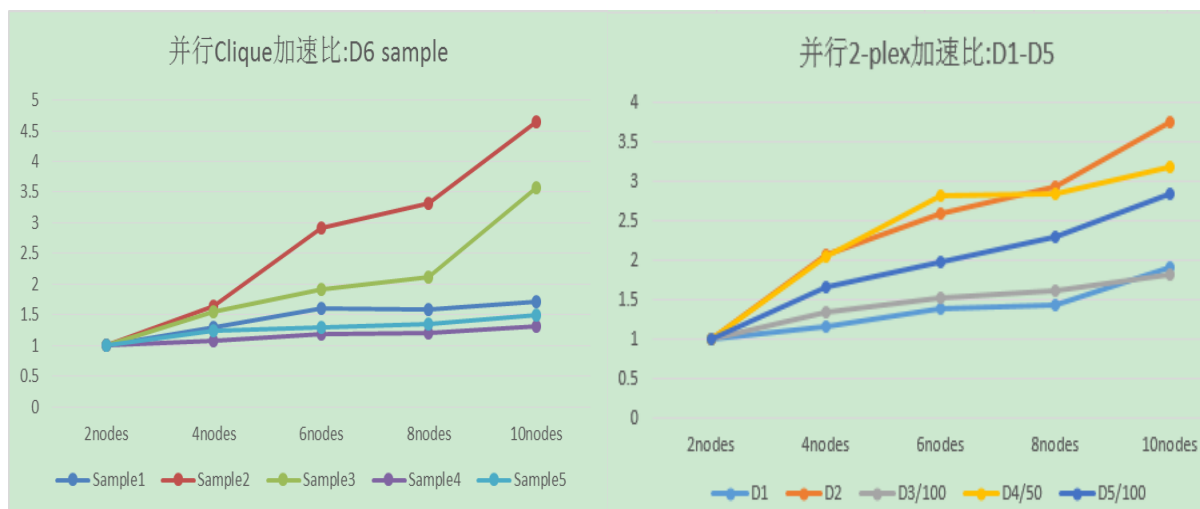


图 5-13 并行加速比实验

本文对 Binary 算法并行完全图枚举以及 2-Plex 并行枚举在实际数据集上做了相关测试，图 5-13 中可以看出算法在大多数时候取得了较好的加速比。但在样本 1、4、5 的完全图枚举中加速比不明显，这是由于最少的两个节点时作业的迭代次数和计算量就很小，增加机器节点后由于初次任务划分的不均衡，不能带来明显的加速。

## 5.4 本章总结

本章主要介绍了实验条件与实验数据，通过充分且全面的对比实验验证了本文提出的 Binary 算法和 Hybrid 算法的有效性和高效性。单机算法表明 Binary 算法比 BK 算法具有更小的搜索树空间且整体上运算时间更优，为并行算法打下良好基础。进而由并行实验验证了 Binary 算法及其负载均衡方案具有良好的并行性以及可扩展性，可以通过机器的简单叠加获得整体处理效率的提升。



## 6 总结与展望

### 6.1 本文研究总结

本文提出了一种基于图分割的极大完全图枚举和近似极大完全图枚举算法 **Binary**。在大多数数据集上与经典对比算法 **BK** 相比，新算法能够有效地减少极大完全图枚举和近似极大完全图枚举过程中搜索树大小，同时节约搜索时间，提升搜索效率。在分析两种算法各自优劣的情况下，以 **Binary** 为主，**BK** 为辅，提出一种结合两者优势的混合算法 **Hybrid**。**Hybrid** 获得了两者在性能和搜索空间上的权衡，避免了极端差的情况出现，甚至在一些连接度较高的蛋白质数据上获得比两者都好的表现。

针对本文提出的 **Binary** 算法以及完全图枚举和近似完全图枚举的特定应用场景，本文提出了该算法在 **Hadoop** 平台上的并行化方案以及其负载均衡方案。进行负载均衡后的 **Binary** 并行化算法比同样条件下的 **BK** 算法具有更少的迭代次数和运行时间，且具有较好的扩展性。

### 6.2 课题研究展望

本文提出的 **Binary** 算法需要在切分过程中持续维护最下度数集合，每次选取最小度数点作为切分点，导致平均单次切分过程中切分代价大于 **BK** 算法，如何减少 **Binary** 算法的单次切分代价是一个可以继续探讨的问题。

在近似完全图枚举过程中，其剪枝条件与完全图一样，要求存在 **Not** 集中的一个点与候选集以及结果集中的点都相邻，才可保证此分支都是冗余的。但近似完全图的条件弱于完全图，剪枝条件相对也是可以进一步放松的，找到一种较弱的剪枝条件将能够极大地提升近似完全图枚举的效率。本文使用 **Hadoop** 作为并行计算平台，业界还有许多优秀的广泛应用的并行计算平台，**Binary** 算法在这些平台上的表现值得进一步验证与优化。

另外，本文所讨论的问题是完全图和近似完全图的枚举。在实际生产环境中，可能对完全图和近似完全图的查询工作可能会有更多的需求。同时注意到在实际生产环境中，输入图数据的网络结构、点边等信息是动态变化的。显然对于这样变化的图数据，每次重新挖掘枚举将会带来极大的冗余工作。如何在并行平台上应用 **Binary** 思想高效地实现这两种问题的并行查询处理，同时兼容增量更新也是一个重要挑战。

本文所提出的负载均衡方案是针对本文所设计的并行完全图和近似完全图枚举算法而设计的。然而实际的生产环境复杂多变，一种通用的系统层面的负载均衡方案可以用户透明地获得计算的均衡。考虑到 **Hadoop** 集群中异构的机器处理能力，不仅会出现数据倾斜同时还有计算倾斜。理论上可以通过细粒度的数据分区获得更好的系统均衡。引入放大系数  $\lambda$ ，将 **Hadoop** 的 **Partition** 切分为更细粒度的 **Finer-Partition**。在获得细粒度

分区后，给启动的所有 Reduce 分配一个默认的 Finer-Partition 并按照原生的 Hadoop 系统机制运行。Reduce 从已完成的 Map 端获取已分配 Finer-Partition 的数据，该 Finer-Partition 数据 Shuffle 完成后即可立即执行 reduce 函数工作。为充分利用现代计算机的多核架构以及 DMA 等存取技术，在 reduce 函数开始处理已经 Shuffle 完成的 Finer-Partition 数据时，可以请求并开始下一个 Finer-Partition 的数据传输。当第一个默认的 Finer-Partition 处理完成后且第二个 Finer-Partition 的数据 Shuffle 到本地后开始第二个 Finer-Partition 数据的处理以及第三个 Finer-Partition 的申请。这样可以使得 Reduce 一边执行的过程中一边从 Map 端准备下一个 Finer-Partition 的数据。JobTracker 中记录各个 Finer-Partition 的分配及执行情况，以便在发生节点故障时指派其他机器重新执行该节点的任务。与 Hadoop 原有的故障恢复机制相比，动态均衡可以在发生故障时只需将分配给故障节点的 Finer-Partition 重新指派给多个申请新任务的节点，充分利用系统的处理能力提高故障恢复效率。这里只是提出了一种设计思路，但具体在什么样的时机去获取新 Finer-Partition、Finer-Partition 是否直接随机分配还是有序分配、恢复及推测执行策略如何实现等详细设计问题是下一步的工作方向。

## 参考文献

- [1] Bron C, Kerbosch J. Algorithm 457: finding all cliques of an undirected graph[J]. Communications of the ACM, 1973, 16(9): 575-577.
- [2] Lu, Li, Yunhong Gu, and Robert Grossman. dmaximalcliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution[C]. ICDMW 1320-1327.
- [3] Cazals F, Karande C. A note on the problem of reporting maximal cliques[J]. Theoretical Computer Science, 2008, 407(1): 564-568.
- [4] Yang S, Wang B, Zhao H, et al. Efficient dense structure mining using MapReduce[C]//Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on. IEEE, 2009: 332-337.
- [5] Eppstein D, Strash D. Listing all maximal cliques in large sparse real-world graphs[M]//Experimental Algorithms. Springer Berlin Heidelberg, 2011: 364-375.
- [6] Wu B, Pei X. A parallel algorithm for enumerating all the maximal k-plexes[M]//Emerging technologies in knowledge discovery and data mining. Springer Berlin Heidelberg, 2007: 476-483.
- [7] Khosraviani A, Sharifi M. A Distributed Algorithm for  $\gamma$ -Quasi-Clique Extractions in Massive Graphs[M]//Innovative Computing Technology. Springer Berlin Heidelberg, 2011: 422-431.
- [8] Tomita E, Tanaka A, Takahashi H. The worst-case time complexity for generating all maximal cliques and computational experiments[J]. Theoretical Computer Science, 2006, 363(1): 28-42.
- [9] Tsukiyama S, Ide M, Ariyoshi H, et al. A new algorithm for generating all the maximal independent sets[J]. SIAM Journal on Computing, 1977, 6(3): 505-517.
- [10] Gufler B, Augsten N, Reiser A, et al. Load balancing in mapreduce based on scalable cardinality estimates[C]//Data Engineering (ICDE), 2012 IEEE 28th International Conference on. IEEE, 2012: 522-533.
- [11] Chiba N, Nishizeki T. Arboricity and subgraph listing algorithms[J]. SIAM Journal on Computing, 1985, 14(1): 210-223.
- [12] Akkoyunlu E A. The enumeration of maximal cliques of large graphs[J]. SIAM Journal on Computing, 1973, 2(1): 1-6.
- [13] Makino K, Uno T. New algorithms for enumerating all maximal cliques[M]//Algorithm Theory-SWAT 2004. Springer Berlin Heidelberg, 2004: 260-272.

- [14]Stix V. Finding all maximal cliques in dynamic graphs[J]. Computational Optimization and applications, 2004, 27(2): 173-186.
- [15]Osteen R E, Tou J T. A clique-detection algorithm based on neighborhoods in graphs[J]. International Journal of Computer & Information Sciences, 1973, 2(4): 257-268.
- [16]Eppstein D, Löffler M, Strash D. Listing all maximal cliques in sparse graphs in near-optimal time[M]. Springer Berlin Heidelberg, 2010.
- [17]Bui T N, Eppley P H. A hybrid genetic algorithm for the maximum clique problem[C]//Proceedings of the 6th international conference on genetic algorithms. Morgan Kaufmann Publishers Inc., 1995: 478-484.
- [18]Tsukiyama S, Ide M, Ariyoshi H, et al. A new algorithm for generating all the maximal independent sets[J]. SIAM Journal on Computing, 1977, 6(3): 505-517.
- [19]Cheng J, Ke Y, Fu A W C, et al. Finding maximal cliques in massive networks by h\*-graph[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010: 447-458.
- [20]Racha S C. Load Balancing Map-Reduce Communications for Efficient Executions of Applications in a Cloud[J]. 2012.
- [21]Rama R, Badarla S, Krithivasan K. Clique-detection algorithm using clique-self-assembly[C]//Bio-Inspired Computing: Theories and Applications (BIC-TA), 2011 Sixth International Conference on. IEEE, 2011: 225-230.
- [22]Walteros J L, Pardalos P M. A decomposition approach for solving critical clique detection problems[M]//Experimental Algorithms. Springer Berlin Heidelberg, 2012: 393-404.
- [23]Wu B, Yang S, Zhao H, et al. A distributed algorithm to enumerate all maximal cliques in mapreduce[C]//Frontier of Computer Science and Technology, 2009. FCST'09. Fourth International Conference on. IEEE, 2009: 45-51.
- [24]Lu L, Gu Y, Grossman R. dmaximalcliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution[C]//Data Mining Workshops (ICDMW), 2010 IEEE International Conference on. IEEE, 2010: 1320-1327.
- [25]Yang S, Wang B, Zhao H, et al. Efficient dense structure mining using MapReduce[C]//Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on. IEEE, 2009: 332-337.
- [26]Schmidt M C, Samatova N F, Thomas K, et al. A scalable, parallel algorithm for maximal clique enumeration[J]. Journal of Parallel and Distributed Computing, 2009, 69(4): 417-

- 428.
- [27]Cheng J, Zhu L, Ke Y, et al. Fast algorithms for maximal clique enumeration with limited memory[C]//Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012: 1240-1248.
- [28]Schmidt M C, Samatova N F, Thomas K, et al. A scalable, parallel algorithm for maximal clique enumeration[J]. Journal of Parallel and Distributed Computing, 2009, 69(4): 417-428.
- [29]Szabó S. Parallel algorithms for finding cliques in a graph[C]//Journal of Physics: Conference Series. IOP Publishing, 2011, 268(1): 012030.
- [30]Balasundaram B, Butenko S, Hicks I V. Clique relaxations in social network analysis: The maximum k-plex problem[J]. Operations Research, 2011, 59(1): 133-142.
- [31]Haraguchi M, Okubo Y. A method for pinpoint clustering of web pages with pseudo-clique search[M]//Federation over the Web. Springer Berlin Heidelberg, 2006: 59-78.
- [32]Liu G, Wong L. Effective pruning techniques for mining quasi-cliques[M]//Machine Learning and Knowledge Discovery in Databases. Springer Berlin Heidelberg, 2008: 33-49.
- [33]Abello J, Resende M G C, Sudarsky S. Massive quasi-clique detection[M]//LATIN 2002: Theoretical Informatics. Springer Berlin Heidelberg, 2002: 598-612.
- [34]McClosky B, Hicks I V. Combinatorial algorithms for the maximum k-plex problem[J]. Journal of combinatorial optimization, 2012, 23(1): 29-49.
- [35]Wu B, Pei X. A parallel algorithm for enumerating all the maximal k-plexes[M]//Emerging technologies in knowledge discovery and data mining. Springer Berlin Heidelberg, 2007: 476-483.
- [36]Khosraviani A, Sharifi M. A Distributed Algorithm for  $\gamma$ -Quasi-Clique Extractions in Massive Graphs[M]//Innovative Computing Technology. Springer Berlin Heidelberg, 2011: 422-431.
- [37]Gufler B, Augsten N, Reiser A, et al. Handling Data Skew in MapReduce[C]//Proceedings of the 1st International Conference on Cloud Computing and Services Science. 2011, 146: 574-583.
- [38]Kwon Y C, Balazinska M, Howe B, et al. Skewtune: mitigating skew in mapreduce applications[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012: 25-36.
- [39]于戈, 谷峪, 鲍玉斌, 等. 云计算环境下的大规模图数据处理技术[J]. 计算机学报,

- 2011, 34(10): 1753-1767.
- [40]Pei J, Jiang D, Zhang A. On mining cross-graph quasi-cliques[C]//Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005: 228-238.
- [41]Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
- [42]MapReduce in <http://zh.wikipedia.org/wiki/MapReduce>
- [43]Hadoop in <http://hadoop.apache.org/>
- [44]Zhang Y, Abu-Khzam F N, Baldwin N E, et al. Genome-scale computational approaches to memory-intensive applications in systems biology[C]//Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference. IEEE, 2005: 12-12.
- [45]Kose F, Weckwerth W, Linke T, et al. Visualizing plant metabolomic correlation networks using clique–metabolite matrices[J]. Bioinformatics, 2001, 17(12): 1198-1208.
- [46]Du N, Wu B, Xu L, et al. A parallel algorithm for enumerating all maximal cliques in complex network[C]//Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on. IEEE, 2006: 320-324.
- [47]Ren K, Gibson G, Kwon Y C, et al. Hadoop's Adolescence; A Comparative Workloads Analysis from Three Research Clusters[C]//SC Companion. 2012: 1452.



## 致谢

在本文完成后，我即将结束研究生生活，在此衷心地感谢在硕士期间，所有对我的学习、工作及生活给予过帮助的人。

感谢我的硕士生导师陈群教授，感谢他在我研究生期间对我的关心和帮助。生活中，陈老师像家长一样，常常与我进行沟通与交流，并给予我鼓励与安慰，使我倍感温暖；工作中，陈老师一丝不苟，从严治学，他严谨的工作风格给我留下了深刻的印象，是我以后科研及工作的榜样。

另外还要感谢潘巍老师，在进行项目研究的过程中，潘老师丰富的经验以及扎实的理论功底，使我受益匪浅，收获颇丰。同时，我要感谢教研室的各位同学们，是他们在遇到难关时，及时给予帮助。在我困惑于问题时，牺牲时间与我讨论，再一次感谢他们。

最后，我要特别感谢我的父母，我的家人，他们是我学习、科研的坚强后盾，一直在为我默默地付出，无私地奉献。无论我遇到什么困难，什么挫折，他们总会站在我身边，给我最强有力的支持，他们的支持是我顺利完成学业的保证。




知识产权声明书

西北工业大学

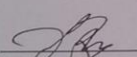
学位论文知识产权声明书

本人完全了解学校有关保护知识产权的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属于西北工业大学。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电子版。本人允许论文被查阅和借阅。学校可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律注明作者单位为西北工业大学。

保密论文待解密后适用本声明。

学位论文作者签名： 

2015年 3月19日

指导教师签名： 

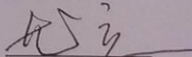
2015年 3月19日

西北工业大学

学位论文原创性声明

秉承学校严谨的学风和优良的科学道德，本人郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容和致谢的地方外，本论文不包含任何其他个人或集体已经公开发表或撰写过的研究成果，不包含本人或其他已申请学位或其他用途使用过的成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式表明。

本人学位论文与资料若有不实，愿意承担一切相关的法律责任。

学位论文作者签名： 

2015年 3月19日