

東北大學



NORTHEASTERN
UNIVERSITY

硕士学位论文

MASTER THESIS

学位论文题目：基于抽样的大规模图数据的可视化与系统实现

所 学 专 业：计算机技术

研 究 方 向：可视化

论 文 编 号：1670838

论文完成时间：2019年7月

東北大学



NORTHEASTERN
UNIVERSITY

硕士学位论文

MASTER THESIS

论文题目 基于抽样的大规模图数据的可视化与系统实现

作者 田宗霖

学号 1670838

学院 计算机科学与工程学院

专业 计算机技术

指导教师 鲍玉斌教授

2019年7月

分类号_____密级_____

UDC_____

学 位 论 文

基于抽样的大规模图数据的可视化与系统实现

作 者 姓 名: 田宗霖

作 者 学 号: 1670838

指 导 教 师: 鲍玉斌教授

东北大学计算机科学与工程学院

申请学位级别: 硕士 学 科 类 别: 工学

学科专业名称 计算机技术

论文提交日期: 2019 年 7 月 论文答辩日期: 2019 年 7 月

学位授予日期: 2019 年 7 月 答辩委员会主席:

东 北 大 学

2019 年 7 月

A THESIS in Computer Thechnology

Visualization and System Implementation of Large Scale Graph Data Based on Sampling

by Tian Zonglin

Supervisor: Professor Bao yubin

Northeastern University

June 2019

独创性声明

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

学位论文作者签名：

签字日期：

学位论文版权使用授权书

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

作者和导师同意网上交流的时间为作者获得学位后：

半年 一年 一年半 两年

学位论文作者签名：

导师签名：

签字日期：

签字日期：

摘要

信息可视化主要研究非数值型信息资源的视觉展示，以达到帮助人们理解并分析数据的目的。随着科技的不断发展，图亦或是网络在我们的生活中出现的越来越多，图数据的产生速度和存储规模也符合大数据时代的发展趋势。与此同时，这些复杂结构中蕴含的隐藏信息也越来越多，对这些网络的进行高效的分析、挖掘及视觉呈现一直是数据科学及可视化领域的研究热点。在这一背景下，诸如社交网络等传统的图数据达到了前所未有的规模，经典的单机可视化解决方案已经无法高效的对海量图数据进行展现，传统的基于数据处理的数据挖掘策略又很难直观的描述或解释其结果。因此迫切需要一种可以有效展现海量图数据的可视化解决方案，以达到在可容忍的时间内对大规模图数据进行直观、可解释的可视分析。如何缩短处理时间以及如何直观、有意义的对数据进行布局展现是一个十分具有挑战性的课题，这也是本文的主要研究内容。

在保证布局质量方面，本文首先分析了单机布局方案的局限性和各种已有分布式处理模型的特点。随后在分层抽样算法及 K 核分解思想的基础上，提出了面向大图的抽样算法，在尽可能保证抽样结构相似性的前提下对大规模图数据进行分层抽样，最终的实验证明，该方法在抽样结构的相似性上有着比 SBS、SS 等算法更好的表现，能够更好的保留原结构的拓扑信息。

在改善布局算法方面，本文通过将抽样方法应用到布局过程中，分层逐级对图数据进行显示。为此，本文实现了基于 GraphX 的力导向布局算法，并将其与上述分层抽样算法结合，由简入繁逐渐细化地对原结构进行布局及展示。在大规模数据处理的过程中，该方案能够有效减少计算需求，并且能够更加灵活的适应可视分析的需要。

最后，本文设计了大规模图数据可视化原型系统，基于 GraphX 实现分层布局算法，借助浏览器（B/S 结构）实现整个可视化界面。并借助此系统对上述提出的抽样及布局算法进行了实验。与现有的单机系统相比，可以处理前者无法有效处理的大图数据；同时与已有的分布式可视化系统相比，可以赋予使用者更细粒度的视图控制能力并得到更清晰的结果展示。

关键词：图数据抽样；分层图布局；分布式计算；K 核分解

Abstract

Information visualization mainly studies the visual display of non-numerical information resources in order to help people understand and analyze data. With the continuous development of science and technology, more and more graphs or networks appear in our lives. The speed and storage scale of graph data also conform to the development trend of the era of big data. At the same time, more and more hidden information is contained in these complex structures. Efficient analysis, mining and visual presentation of these networks has always been a research hotspot in the field of data science and visualization. In this context, traditional graph data such as social networks have reached an unprecedented scale. The classical single-machine visualization solution has been unable to effectively display large amounts of graph data. The traditional data mining strategy based on data processing is difficult to intuitively describe or explain the results. Therefore, there is an urgent need for a visualization solution that can effectively display massive graph data in order to achieve intuitive and interpretable visual analysis of large-scale graph data in tolerable time. How to shorten the processing time and how to visually and meaningfully display the layout of data is a very challenging topic, which is also the main research content of this thesis.

In order to ensure the layout quality, this thesis first analyses the limitations of the single-machine layout scheme and the characteristics of various existing distributed processing models. Subsequently, on the basis of hierarchical sampling algorithm and K-kernel decomposition idea, a large-scale graph-oriented sampling algorithm is proposed. On the premise of ensuring the similarity of sampling structure as much as possible, the large-scale graph data are stratified sampled. The final experiment proves that this method has better performance than other algorithms in the similarity of sampling structure, and can better retain the topological information of the original structure.

In order to improve the layout algorithm, the sampling method is applied to the layout process, and the graph data is displayed hierarchically and step by step. To this end, this thesis implements a force-oriented layout algorithm based on GraphX, and combines it with the above-mentioned hierarchical sampling algorithm, which gradually refines the layout and display of the original structure from simplicity to complexity. In the pro-

cess of large-scale data processing, this scheme can effectively reduce the computational requirements and be more flexible to meet the needs of visual analysis.

Finally, this thesis designs a large-scale graphics data visualization prototype system, using GraphX to compute the coordinates of graph nodes, using B/S mode to achieve the entire visualization interface with the help of browser. With the help of this system, the sampling and layout algorithms proposed above are experimented. Compared with the existing single-machine system, it can deal with the large-scale map data which the former can not effectively handle; at the same time, compared with the existing distributed visualization system, it can give users more fine-grained view control ability and get clearer results display.

Keywords: Graph sampling; Hierarchical layout; Distributed computing; K-core decomposition

目 录

独创性声明	I
摘要	III
Abstract	V
第 1 章 绪论	1
1.1 研究背景	1
1.1.1 图数据的产生和发展	1
1.1.2 图数据的可视化技术	2
1.2 研究意义	2
1.2.1 大规模图数据可视化的重要性	2
1.2.2 分布式大图布局计算的特点和挑战	3
1.3 本文主要贡献及组织结构	4
1.3.1 本文的主要贡献	4
1.3.2 本文的组织结构	5
第 2 章 大图可视化的相关工作	7
2.1 图布局算法的发展	7
2.1.1 单核串行阶段	7
2.1.2 单机并行阶段	11
2.2 大规模图布局技术的基础	12
2.2.1 分布式存储计算平台	13
2.2.2 大规模图数据处理框架	14
2.2.3 大规模图布局算法	17
2.3 图数据抽样算法	18
2.4 本章小结	19
第 3 章 面向大图的分层抽样算法	21
3.1 分层抽样算法	21
3.1.1 相关定义和符号说明	21
3.1.2 SS 抽样算法	22
3.2 基于 K 核分解的分层抽样算法	24
3.2.1 K-core 分解	24

3.2.2 改进后的分层抽样算法	24
3.3 抽样结构对比指标	27
3.4 实验结果与分析	28
3.4.1 实验设计	28
3.4.2 实验结果及结论	29
3.5 本章小结	33
第 4 章 分布式图布局算法	35
4.1 基于 KSS 抽样的分层布局方法	35
4.2 两种不同的分层策略	35
4.3 分布式图布局算法模型	38
4.4 基于 GraphX 的算法实现	38
4.4.1 关键编程接口	38
4.4.2 布局算法的实现	40
4.4.3 作业优化	41
4.5 实验结果与分析	42
4.5.1 实验设置	42
4.5.2 布局结果与结论	43
4.6 本章小结	45
第 5 章 大图可视化原型系统	47
5.1 技术选型及依据	47
5.1.1 系统结构	47
5.1.2 图绘制技术	48
5.1.3 数据传输及后端框架	49
5.1.4 总体流程	50
5.2 系统设计	51
5.2.1 基本任务	51
5.2.2 模块分析	51
5.3 部署及使用	54
5.4 本章小结	55
第 6 章 总结与展望	57
6.1 本文的主要贡献与结论	57
6.2 未来展望	57
参考文献	59

致谢	63
攻读硕士学位期间取得的学术成果	65

第 1 章 绪论

1.1 研究背景

1.1.1 图数据的产生和发展

图是信息科学中最常用的一类抽象数据结构，能够直观的表达现实世界中对象之间的真实关系。许多重要应用都需要用图结构表示，传统应用如最优运输路线的确定、疾病爆发路径的预测、科技文献的引用关系等；新兴应用如社交网络分析、语义 Web 分析、生物信息网络分析等，与图相关的处理和应用几乎无所不在^[1]。

随着技术的不断发展，网络在我们的生活中比以往任何时候都更加突出，上述一系列网络或图结构中都蕴含了越来越多的隐含信息，对这些网络的进行高效的分析和挖掘是亟待解决的一个问题，可视化技术在这一问题上具有无可比拟的优势，其视觉呈现可以给人们带来直观的对数据的理解和感知，Palmer 等人在其文章中证明，图比其他可视化展现形式更适合探索数据的内部关系^[2]。

面对飞速发展的信息社会，各式各样的数据集均迅速增长，部分原因是它们越来越多地被廉价且众多的信息传感物联网设备收集，如移动设备，航空（遥感），软件日志，照相机，麦克风，射频识别（RFID）阅读器和无线传感器网络。身处大数据时代，其对数据的分析和挖掘显得尤为重要，2016 年的大数据的定义在指出大数据代表的数据信息具有 4V 特性之外，还强调了其需要特定的技术和分析方法才能转化为价值。随着技术的不断进步，各大分布式计算框架相继产生，其为大数据处理提供了有力的支撑，作为分布式计算、并行计算和网格计算的发展和延续，其对于单机串行处理性能不足的问题，给出了新的解决方案。

在万物互连的今天，我们已经拥有经典的大规模数据的处理方案，可以借助诸如 Spark 或 Hadoop 等分布式计算框架来实现大规模数据集的各种数据挖掘算法，在此趋势下越来越多的由于数据规模过大而导致的分析难题都被解决，但与此同时，在数据集规模和数据维度的极度膨胀下，我们对分析结果的准确性和可解释性的把握正在逐渐降低，追求可视化的宏观展现与细粒度的数据分析结果的一致性从来都是一个美好的愿景，前者可以对后者提供视觉上的强有力的支持和解释，在信息可视化领域，有越来越多的研究者在为此而努力。

1.1.2 图数据的可视化技术

从可视化技术诞生伊始，其目标就是为了帮助人们理解抽象、混乱的数据，至今这一目标依然不曾改变。现在，数据可视化已经发展成为一个广泛的研究领域领域，处于数学、计算机科学、认知和感知科学以及工程学的交叉领域。从信号理论到成像，从计算机图形学到统计学，涵盖所有与可视化原理相关的学科^[3]。可视化的目的是通过具象的、可交互的图形，让人们深入了解我们感兴趣的过程（算法流程、科学模拟或一些真实的过程）的各个方面。可视化本身有许多定义，按照 Williams 等人的观点，可视化是人类在一个空间内构建图像时所进行的认知过程。在计算机和信息科学中，它更具体地说，是使用图形、图像、动画和声音来更详细的表示目标对象的数据、结构和动态行为的空间具象表示^[4]，这里的目标对象可以指系统、事件、过程、对象和概念的大型复杂数据集。

目前，由于互联网及物联网技术的发展，网络这一名词正在被赋予越来越多的应用场景，随之而来的是我们自身正在越来越多的网络中扮演着重要的角色：大到 Facebook 的全球社交关系网络，小到家庭智能电器组网，都成为我们日常生活中不可或缺的一部分。这些复杂的网络中往往隐含着非常有价值的信息，例如，通过分析特定的社交网络来挖掘犯罪嫌疑人的相关信息^[5]。经典的社交网络等复杂网络，其本质可用图这一数据结构相对应，自然地，图论中的各种理论也可以被应用到网络分析中，复杂网络分析方法的发展也验证了这一结论，在本文中，将统一称研究对象为“图数据”，将对其可视化呈现称为“图数据的可视化”，这里的图数据除了包含经典的社交网络之外，还包括各种可以抽象为网络的数据，总体上可以将其称为“社会网络”。

根据 IEEE VIS^[6] 的分类，可视化研究领域主要分为信息可视化（InfoVIS）、科学可视化（SCIVIS）和可视分析（VAST）三部分，其中以信息可视化最为基础，其核心目的即：将给定的数据集 D 按特定的转换（Transform）规则 T ，转换为对应的带空间信息的数据集 $V(D)$ 并显示，以此来帮助人们理解大量、复杂、抽象的原始数据。对于图数据的布局过程，这一转换 T ，即数据集 D 在给定二维（或三维）空间内的一组坐标映射 $V(D)$ 。

1.2 研究意义

1.2.1 大规模图数据可视化的重要性

在对传统的计算技术带来了挑战的同时，大数据技术的发展也促进了数据可视化的研究。作为数据最上层的展示方法，数据可视化使用图形化的手段，可

以传达清晰有效的信息，促进人们对信息的理解。目前数据可视化技术可分为这几类：基于几何投影技术的方法、基于图表的方法、基于像素的方法、基于图符的方法、基于层次的方法和组合方法。作为可视化展现形式之一的网络图，是一种简单直观的图数据可视化展现形式；由于高密度数据区域的聚集特征，散点图可以更为直观的发现群体的存在；在探索大规模数据集时，网络拓扑结构的高度重叠是最严重的缺点之一，这常常会导致数据相互的关系被隐藏或很难被发现。

分布式计算框架的飞速发展极大地提高了人们对数据的处理能力，使得人们有机会可以直接研究大规模的数据集，在这方面，图数据占据了重要的位置。随着移动互联网的飞速发展，以经典的“网络”为主的图数据越来越多的出现在我们的日常生活中。在针对图数据的可视化中，提供有效的洞察力非常重要，这具体体现在以下两个方面：

- 通过布局来展现图数据的具体特征

在这方面可以回答用户关于图数据本身的定量问题，例如在数据中拥有邻居最多的节点的信息、数据可以被划分为几个聚簇等等。这类问题的特点，是有明确的“问题驱动”，往往在没有可视化结果的帮助下用户也能得到准确的结果，但对于图数据而言，诸如想聚簇等特征，只有通过具体的布局呈现，才能更直观的得到一些运行结果的评价，像划分的聚簇间有无重叠这一问题，既需要数值上的（重叠度）准确度量，也同样需要布局上的清晰的边界划分。

- 为产生布局结果的过程提供信息

这部分内容并不针对具体问题，但对布局过程的尽可能详细的展现，使用户能够更容易的针对布局结果呈现的状态来反推出一些非常规问题的答案，例如在两方对立的意见网络中，中间（中立）节点对整个布局质量的影响（更紧密或更清晰），对于这个例子，研究者可能在一开始只发现了布局混乱这一现象，而并不知道中间节点对布局的影响，在经过对布局过程的可视化过程研究以后才得到最终结论。这类问题即 Telea 所说的“研究者对一种现象感兴趣，是为了发现数据新的特性并建立意想不到的关联”^[3]，这对于进一步理解数据中的信息起到了至关重要的作用。

1.2.2 分布式大图布局计算的特点和挑战

面对大规模图数据的处理需求，分布式图数据布局计算往往有以下特点：

(1) 需要进行计算的数据量巨大。节点规模在百万以上，单机甚至无法将全部顶图数据加载到内存中，虽然使用 Spark 等基于内存的分布式框架可以完成对此规模数据的处理，但如此规模的数据量仍然对集群的 IO 性能、网络传输性能

提出了很高的要求；

(2) 需要完成逻辑上的多次迭代。应用广播等方式可以避免显式的循环使用，但布局计算的完成仍然离不开动辄几百次的迭代步骤，这对于传统 MapReduce 计算框架来说是一个灾难，Spark 的 RDD 有效的避免了中间结果的 IO 操作，但上百次的迭代仍需要研究者十分小心的处理 RDD 的各种转换；

(3) 必须重写已有布局算法。显而易见，在布局算法的运行机制和环境发生改变后，布局算法也需要重新进行设计以适应不同的计算框架，虽然这和原有布局算法有着相同的算法思想，但在特定分布式计算框架上设计出完全适合且能够高效运行的算法，往往考验着研究人员对特定分布式计算框架的理解程度；

(4) 布局结果难以查看。单机布局算法往往能够轻松的对每次迭代的结果进行展示，像可视化布局工具 Gephi^[7]可以通过显示每次布局的结果来呈现动态的布局过程，但在分布式计算框架中这一操作往往很难实现，因为在此环境下数据会分片发往不同的工作节点进行计算，如果每次迭代完成都要汇总一次来展现当前布局结果，这将直接导致使用基于内存的计算框架的优点的丧失。

1.3 本文主要贡献及组织结构

1.3.1 本文的主要贡献

本文在现有研究工作的基础上，针对大规模图数据的特性，结合抽样技术以及布局技术的研究成果，提出了基于采样的大规模图数据的布局方法。本文的研究结果和创新总结如下：

(1) 改进了基于拓扑分层的大图抽样算法，保证在抽样过程中对图结构拓扑特征的保留；同时，为了确保抽样结构的外扩趋势，引入了基于 K-core 的节点分层划分技术。通过以上两者的结合，可以在保留拓扑结构的基础上保证对原始图结构抽样的离散程度。并且通过实验验证了该方案的有效性。

(2) 实现了基于 GraphX 的图布局算法，在所提出的分层抽样算法的基础上提出了基于抽样的分层布局算法，可以充分利用集群的计算资源来完成布局任务，并且赋予用户细粒度的探索能力，在一定程度上加速对图数据布局的计算过程，同时为可视化的清晰展现提供了基础条件。

(3) 设计并实现了 GVVIS 大图可视化原型系统，GVVIS 系统实现了将上述流程封装成一个集上传数据、呈现布局、集群监控、布局交互于一体的大图可视化工具（系统）。该工具能够完成对用户给定的图数据进行快速地分级显示的任务，并提供一个能够进行复杂拖拽、缩放、选取、上色的工具组件来完成对布局结果

的复杂交互动作，整个系统具备完整的图布局输入输出流程，使用了中间层解耦的思想进行设计实现，能够高效的利用远端计算集群为用户提供有效的大图可视化交互服务。

1.3.2 本文的组织结构

本文一共分为六章，具体章节安排如下：

第1章为绪论部分。首先介绍了图在我们生活中的重要作用，然后给出了本文有关图数据及图数据可视化的明确限定，在此基础上，介绍了由于数据规模增大而产生的大规模图数据的可视化重要性，并列举了大规模图布局计算的特点和难点，最后提出了本文解决的问题并总结了本文的贡献点。

第2章是相关工作。首先介绍了数据规模对可视化方案的影响，按不同数据规模对不同阶段的图可视化技术做了回顾，重点介绍了目前在大规模图布局方面的研究进展，最后对目前主流的抽样算法做了详细介绍。

第3章主要介绍本文提出的基于 K 核分解和度分布过滤的抽样算法（KSS），详细论述了 K 核压缩和拓扑分层抽样（SS）算法存在的优点及问题，整合了两者的优势提出了一种新的图抽样算法，并通过多组实验对抽样后的结构相似度做了验证。

第4章基于 KSS 抽样算法提出了分层的布局算法（KSS Layout）。详细论述了基于 GraphX 实现布局算法的编程接口，在此基础上实现了直接布局方法和基于抽样的分层布局算法，并通过实验验证了后者的有效性。

第5章对大图可视化原型系统进行了简单介绍，主要包括系统的技术选型、主要架构、功能模块等，并给出了系统的部署和使用方法。

第6章是全文总结和对未来工作的展望。

第2章 大图可视化的相关工作

本章内容首先叙述了数据规模对可视化方法的影响，主要集中在布局和交互两个方面。第2.1、2.2小节从布局计算的角度回顾了图布局算法的发展；第2.3小节补充了目前针对大图数据在交互上的方法和优化策略。

2.1 图布局算法的发展

由第2章中所述，图数据的可视化，核心在布局，而布局算法通常是按照一些特定的模型，将抽象数据进行具象展示，这一过程伴随大量的迭代计算，例如朴素的 FR 力导向算法其在计算斥力时的算法时间复杂度达到了 $O(n^3)$ ，这在小规模数据量下可能并不会出现问题，但随着规模的不断增大，采用如此“高昂”计算复杂度的算法变得不能接受，所以，出现了许多针对算法时间复杂度进行改进的方法，需要说明的是，在这一阶段，数据集的规模仍未达到单机处理上限，例如 OpenOrd 算法采用多线程并行来加速计算过程。随着数据规模的进一步扩大，图数据节点达到百万级别时，单机并行策略也变得无能为力，这时，分布式并行计算的方式为这种“大规模图数据”的处理提供了可能性。

图在数学上通常用 $G = (V, E)$ 来表示，其中 V 是顶点的集合， E 是边的集合，且每条边 $e \in E$ 都连接两个顶点 $\langle v_i, v_j \rangle$ ，且边 e 通常由 $\langle v_i, v_j \rangle$ 来存储表示，在本文中，我们将图数据的表示限定在二维平面内，因此对于每个顶点 v_i 来说，布局的最终目的是确定 v_i 在画布上的位置 (x_i, y_i) 。对于整个图数据可视化的发展趋势，大致经过了单核串行、单机多核并行和多机分布式并行三个阶段，各阶段存在时间的先后顺序，且由于各阶段的程序运行机制的差异，导致了对可视化处理流程（布局）实现和优化方面的差异。下面就图数据可视化的各个发展阶段进行回顾。

2.1.1 单核串行阶段

在这一阶段产生了非常多经典的可视化工具，例如 Gephi^[7]，一个能够展现和操作布局的软件，其允许用户进行二次开发加入其自定义算法或实现更复杂的功能，由于它的通用性和开放性，现在已成为最为普遍的网络可视化工具软件。除此之外还有许多类似的可视化分析工具软件，像 Ucinet 是早期针对对网络分析工具^[8]，不过由于软件本身设计较早，导致整体对数据的处理能力严重不足，在 2011 年被停止维护；Pajek 也是一款诞生很早的网络分析挖掘工具^[9]，可以对

布局进行复杂的分层、置换等操作，其一直发展至今衍生出了多个版本，社区也非常活跃。在 2001 年出现的 NetMiner^[10]，将易用性和扩展性提高到了新的高度，并由此创造了成功的商业模式。在 2008 年，微软研究团队发布了 NodeXL^[11]，通过作为 Microsoft Excel 插件的形式，NodeXL 可以获得良好的易用性，不过由于其采用的 .Net 平台开发，也使得其没有跨平台的能力，且 NodeXL Pro 属于付费模块，开源部分有限，总体开发扩展性不高。

表 2.1 可视化分析工具比较。
Table 2.1 Comparison of visualization tools

名称	出现	最新版本	免费	开源	语言	动态布局	主要功能
Ucinet	1992	2011.11	×	×	Delphi	×	分析
Pajek	1996	2019.3	√	×	Delphi	√	分析、可视
NetMiner	2001	2019.2	×	×	Java	×	分析、可视
NodeXL	2008	2019.4	×	√	C#	√	可视、分析、交互
Gephi	2009	2017.9	√	√	Java	√	可视、分析、交互
Visone	1996	2017.2	√	×	Java	×	可视、分析
Graphviz	2000	2019.2	√	√	dot	×	可视、分析、交互
Network Workbench	2005	2011.3	√	×	C++	×	可视、分析
SocNetV	2014	2019.3	√	√	C++/Qt	×	分析、可视、交互
Tulip	2000	2019.4	√	√	C++	√	分析、可视、交互

表 2.1 针对部分指标对以上软件做了对比，其中，Gephi 是目前适用范围比较广泛的网络可视化工具，对于 Gephi 来说，基于 NetBeans 的模块项目构建、以可视和交互为主的分析模式以及开源的社区支持，都使得 Gephi 成为一款更为纯粹的针对图数据可视化的工具。因此，在本文的实验部分，Gephi 被选做用来完成相关比较。还有一点值得关注的是，上述工具均不支持分布式运行环境，即处理数据的规模严重依赖主机的内存和处理器，虽然在 16G 甚至更大 RAM 的条件下，个别工具依靠多核并行的布局算法可以完成数十万节点规模的粗略布局，但对于更大规模的图布局来说，其高迭代性和高时间复杂度的特性使得上述软件均不能很好的完成大规模图数据的布局操作。

不过在早期的研究阶段中，针对的图数据规模一般较小，并未达到单机处理极限，可视化研究的重点大都集中在布局模型的探索，这一时期出现的力导向模型为图布局的发展起到了重要作用，众多图布局算法均由其改进而来。除此之外，这一阶段也产生了许多基于其他模型的图布局算法。

力导向布局算法也称 FDP (Force-Directed Placement) 算法是目前在图布局算法上应用最为广泛的算法，其在自然规则模型（弹簧或电荷力）的指导下，能以人类易理解的形式充分展现图的整体结构，通用性强，在图的布局算法中占据主导地位。根据力导向算法得到的布局结果，具有节点间相关的特性，即布局过

程取决于节点间的连接而非节点具有的属性，这种方法的缺点是其对初始状态十分敏感，且布局过程可能会陷入局部最优解，同时整个过程具有不确定性，不能确保每次得到相同的结果。尽管该方法存在这些问题，但其仍然是对数据在结构上进行视觉解释的通用方案，并在随后的几十年时间里，有不少的研究都基于这一理论，且同样对图布局的研究产生了很深远的影响。对于力导向布局的改进，主要体现在能量模型和计算方式上。

以图布局为核心的图数据可视化的起点，可以追溯到 1984 年，在这一年，Eades 提出将图数据的布局模拟为弹簧和铁环的物理系统 Eades^[12]，但在实现上其并没有反应真实的胡克定律，且时间复杂度为 $O(|V|^2)$ 。之后，Kamada 和 Kawai^[13] 对其做了改进，引入了非邻居节点间最佳距离的概念（最佳距离 l 同非邻两节点间的最短路径成正比），并首次将整个布局过程抽象为能量降低（最优化）的问题，通过最小化节点间斥力和引力的和，来将节点放置在合理的位置，整个系统中的能量模型可用公式2.1表示，

$$Energy = \sum_{0 \leq i < j \leq |V|} k_{ij} (|x_i - x_j| - l_{ij})^2 \quad (2.1)$$

其中 x_i 为顶点 v_i 对应的坐标位置， k_{ij} 为 x_i 与 x_j 之间连接弹簧的弹力系数， l_{ij} 为顶点 v_i 与 v_j 之间的最佳距离。该方法每次只求取一个点的最佳位置，即其内循环（单次算法执行）时间复杂度为 $O(|V|)$ 。

之后，Davidson 和 Harel^[14] 首次在布局过程中引入模拟退火算法，Fruchterman 和 Reingold^[15] 在此基础上使用了简化的“弹簧-电荷”模型（SEM, Spring-Electrical Model）来指导布局，在其算法中两节点间的斥力 f_r 和引力 f_a 可由公式2.2表示，并首次在布局时采用画布面积来计算得到最佳的节点间的距离。

$$\begin{aligned} f_r(i, j) &= -CK^2 / \|x_i - x_j\|, i \neq j, i, j \in V \\ f_a(i, j) &= \|x_i - x_j\|^2 / K, i \leftrightarrow j \end{aligned} \quad (2.2)$$

其中， K 即为最佳节点距离， C 是调节斥力引力相对强度的参数， $\|x_i - x_j\|$ 即为点 x_i, x_j 之间的距离。

在力导向布局算法出现后的近十年中，研究者们主要的研究内容集中在模型的改进和减少绘制时的边交叉等问题，主要偏重于理论研究，在数据集的选取上均为规模较小（几十至几百顶点数）的、具有特定结构的图数据。由于这部分算法包含的内容一部分已经不具有时效性，所以这里仅列举了上述对布局算法

有很大影响的基础性的工作。

在图布局算法发展的第二个十年里，研究者们不在满足于算法仅适用于小规模数据集的现状，开始向更大规模的数据发起了挑战，这一时期，布局过程中使用多尺度布局算法来加速布局过程逐渐成为研究热点，多尺度算法的最初思路是：对于图 $G = (V, E)$ ， G_0 是整体图结构， G_0 在开始生成一系列小尺寸的图 G_1, G_2, \dots, G_k 。然后用经典的力导向算法绘制级别最小的图 G_k ，在 G_k 的基础上来绘制 G_{k-1} 的布局，递归地执行此过程直至完成 G_0 的绘制。在 1999 年，Hadany 和 Harel 首次将多尺度算法^[16] 应用到图布局的过程中，使用简单的梯度下降来最小化计算模型的能量。其算法时间复杂度为 $O(|VE|)$ ，但需要的空间复杂度为 $O(|V|^2)$ 。不久后，Walshaw 将该思想做了更为细致的阐述，提出了多尺度 FDP 算法^[17]，并对多尺度算法的时间复杂度做出了详细的分析。2003 年，Chan 等人针对符合幂律分布的网络提出了 ODL 算法^[18]，首次提出通过图的拓扑结构来分层布局以加速布局过程，这一简单思想对后来的大规模图布局研究产生了很大的影响。第二年，Hachul 和 Jünger 提出了 FM^3 算法^[19]，其通过一种“势能”系统来完成多尺度的布局思路，且首次将四叉树进入到布局过程来计算节点的近似势能信息。2006 年，HuYifan 从另一个角度出发，采用 Barnes-Hut 技术，通过构造八叉树来计算节点间的近似斥力^[20]，得到了和 Walshaw 的多尺度布局算法相近的绘制效率。

同期，国内黄竞伟等人开始对“画图”问题（图布局）展开研究^[21]，他们将布局问题抽象为函数优化问题，然后利用遗传算法来求解目标函数的最优解的近似值。随后，孙炜等人又在其基础上引入模拟退火算法来克服遗传算法局部搜索能力较差的缺点^[22]。在多尺度布局方面，最近几年国内研究者们针对分层布局提出了许多新的解决方案，包括基于图匹配^[23] 及基于改进力导向^[24] 的分层布局方法。

由于数据规模的增大，不可避免的出现了表示混乱的问题，在 2009 年，Holten 等人首次提出边捆绑的概念（Edge bundling）并将其应用到图布局的展现上^[25]，其提出在边绑定时要遵循一种“位置关联”的规则，即像长短差异很大或是距离间隔很远的边不应该捆绑在一起，这种方法的使用大大提高了图布局结果的展现能力，直到最近边捆绑技术仍然是图可视化研究的热点^[26]。在这之后的布局算法中，美学设计被提到了和算法本身同等的位置，边捆绑等技术得到了快速的发展。同年，斯坦福大学推出了 SNAP 数据集服务^[27]，其旨在搜集整理不同种类的网络数据并以公开的方式将这些数据提供给研究者，还是在这一年，Gephi

的问世使得图数据的可视化研究变得更为方便，虽然在这之前有包括 Pajek 在内的许多工具软件，但 Gephi 仍然以其通用的开放编程模型和易用的界面使对图数据的可视化研究成为了一款通用可视化工具，同时随着边捆绑等视觉呈现方式的改进，图数据的可视化逐渐进入了一个新阶段：对真实社会网络的描述。

社会网络是复杂网络的一个重要分支，其描述真实的数据关系，同时也具有真实数据集的一些特点：符合幂律分布、易出现小世界网络等等，指虽然之前的 FM^3 算法已经能够完成 20 万节点规模的绘制（AT&T 当时规模最大的数据集），但其侧重点仍然放在对算法的原理分析上。视觉展现技术和可视化软件的成熟，使得像社交网络分析等学科通过可视化来辅助验证或直接分析相关问题成为了可能，这也促进了图数据可视化同社交网络等学科的进一步融合。在这一阶段，图数据的可视化进入了第三个十年，将目标转移到描述真实的社会网络之后，如何更好的展现数据的结构似乎成为了首要目的，这一阶段有相当一部分的研究都侧重在美学设计方面，即如何减少混乱、如何呈现小世界网络^[28] 等；同时，在处理数据的规模进一步增大和算法的复杂度不断提高的前提下，传统单机单线程的算法遇到了性能瓶颈，此外，一些边捆绑技术要求更高的计算代价，这也进一步促进了并行算法的出现。

2.1.2 单机并行阶段

在这一阶段，传统的图布局算法引入了很多新的技术，例如采用并行计算来加速迭代过程，或是使用 GPU 编程来加速计算渲染过程。2009 年，在 FM^3 算法出现几年后，Godiyal 等人同 NVIDIA 公司的 Garland 对其提出了基于单指令多数据流处理器（SIMD，例如 GPU）的并行计算版本，他们的算法采用了一种基于 CPU 和 GPU 结合构建的 kd 树，在布局计算上的时间复杂度保持在 $O(N \log N)$ 内。此后，GPU 并行技术正式被引入到图可视化领域，直到目前为止，其仍然是加速 3D 图形骨骼绘制（Skeletons）技术的有效手段^[29,30]。

最初，研究者们尝试使用多显示器的方式对图布局进行显示^[31,32]，这样顶点均匀的分布在不同显示器上，每个显示器关联自己的处理器，各处理器分别计算顶点的位置。该类算法并不具备良好的伸缩性，实验仅限于处理包含数千个顶点的图。之后，Tikhonova 和 Ma 提出了一种并行的力导向算法^[33]，可以在具有几十万个边的图上运行。其算法运行在 Cray XT3 的 32 个处理器上（类似超级计算机），布局包含 260385 条边的图大约需要 40 分钟，效率仍有提升空间。

在 2011 年，美国橡树岭国家实验室提出了 OpenOrd 算法，能够完成百万节点规模的数据布局，这是一个典型的分段布局算法，即对图数据的布局不再是多

次布局算法的迭代，而是整个布局算法分为几个阶段，布局时分别执行这几个阶段，执行结束时布局结束，整个布局算法只执行一次，不存在外循环。OpenOrd 算法主要采用了四种思想，即多尺度布局、节点聚集、并行计算和启发式的边切割，主要贡献即提出了可行的并行图布局思路和具体实现（在 Gephi 0.9.0 版本被纳入到布局算法中^[34]），在模型上其沿用了 VxInsight^[35]（其前身算法）使用的网络密度模型，其整个系统的能量可由公式2.3表示。

$$E_{system} = \sum_i \left(\sum_j (w_{ij} \times d(x_i, x_j)^2) + D_{x_i} \right) \quad (2.3)$$

其中， D_{x_i} 会向画布中靠近点 x_i 的点 x_1, \dots, x_n 贡献密度值，上式的求和包括斥力和引力两部分，在计算引力时通过 $\sum_j (w_{ij} \times d(x_i, x_j)^2)$ 中的 w_{ij} （边的权值）来作用使之将关联性强的节点放置在相近的位置，斥力部分通过密度项 D_{x_i} 表示，其目的是将节点低密度的填充到画布中，对这两部分的和求取最小值来达到最终的布局状态，即关联性强的节点彼此靠近，但又必须满足低密度放置的要求。这样，在进行布局时，OpenOrd 算法采用多线程模式将带密度信息的画布复制多个副本，每个线程处理不重叠的部分数据，且都对自己的画布密度进行更新，然后在合并步骤将各处理线程的画布密度信息进行统一合并更新并调整节点的位置，循环迭代过程完成布局。该算法依靠多核处理器的并行能力，可以将单机（四核八线程）处理能力提高到百万节点级别。

2.2 大规模图布局技术的基础

要想完成大规模图的布局计算，根据上述分析的特点，在单机处理达到极限之后，分布式并行处理是一个非常有潜力的研究方向，其可以借助集群众多节点的计算能力，在有效的任务调度和资源分配的前提下，并行的完成计算任务；由于集群节点的数量并未有理论上限，所以理论上能够实现单核任务计算的任意倍计算能力，但实际上由于集群间的通信代价、任务调度的执行代价以及各工作节点的运行等待代价，集群的节点数量并不能无限增长。现今，Hadoop 作为通用的数据存储计算平台，可以为图数据处理（布局）提供稳定的底层存储支持。Spark 作为内存计算的集大成者，为图数据处理提供了高效的计算模式。本章将从基础的存储计算框架开始，简要回顾大数据处理技术的发展，然后进一步针对基于现有图计算框架产生的图布局算法进行了梳理。

2.2.1 分布式存储计算平台

(1) Apache Hadoop, Hadoop^[39] 最初是谷歌 MapReduce^[40] 等“三驾马车”的开源实现，现在已成为通用的数据存储计算平台，结合其生态圈的其他组件，可以完成许多复杂场景的任务，已经被越来越多的企业应用到实际生产中。Hadoop 隐藏了并行处理的细节，包括向处理节点分发数据、任务调度以及在计算后结果的合并等。该框架暴露特定的接口来让开发人员进行程序编写，在开发过程中，这些程序可以更好的专注于计算逻辑而不用考虑复杂的并行化问题，这是 Hadoop 赋予其应用程序的优秀特性。随着 Hadoop 的不断发展，其逐渐演化为一个拥有众多组件的生态系统，其中最核心的部分就是分布式文件系统（Hadoop distributed file system, HDFS）和 MapReduce 编程框架^[41]。

(2) Apache Spark, Spark^[42] 是一个围绕速度、易用性和复杂分析构建的大数据处理框架。最初在 2009 年由加州大学伯克利分校的 AMPLab 开发。相较于 Hadoop Mapreduce 2.X 版本的计算框架，其避免了编写大量 MapReduce 作业的负担，允许开发者使用有向无环图（DAG）开发复杂的多步数据管道。而且还支持跨有向无环图的内存数据共享，以便不同的作业可以共同处理同一个数据。同时，Spark 通过在数据处理过程中成本更低的洗牌（Shuffle）方式，将 MapReduce 提升到一个更高的层次。此外，Spark 具有一种新的数据抽象模式称为 RDD（弹性分布式数据集）^[43]，RDD 是容错的并行的数据结构，可以让用户显式的将数据保存在内存中，并且可以控制他们的分区来进行优化，此外 RDD 还提供了一系列高级操作接口来帮助重新安排计算并优化数据处理过程。在需要大量迭代计算的场景下，Spark 借助 RDD 主要拥有以下优点：

- 主要基于内存（自动进行内存和磁盘数据存储的切换）的中间结果存储策略使得 Spark 能够直接使用内存中的数据进行计算，减少迭代过程中中间结果读取的开销。
- RDD 本身是待处理数据的一种抽象，在创建 RDD 时可以指定分片个数（或采用默认值），自行决定并行计算的粒度，由于 RDD 的计算是以分片为单位的，每个 RDD 都会实现 compute 函数以达对迭代器结果进行汇总的目的，不需要显式保存每次计算的结果。
- RDD 通过构造一个有向无环图来具有 Lineage（血统）的特性，RDD 每次的 Transform 操作都会重新生成一个新的 RDD，如此可以构造出明确的依赖关系，在某些需要重新计算某个分区数据的场景下，Spark 通过这个依赖关系即可完成重算，不必对 RDD 的全部分区进行重新计算。

- 通过 Persist 方式可以指定将数据存放在内存中，迭代过程中需要重复访问的一些资源可以通过这种方式来减少每次的 IO 代价。
- 同时，RDD 也允许通过 Checkpoint 的方式将数据指定存储在文件系统中，在数据丢失时，就可以从 Checkpoint 快照中的 rdd 直接进行数据恢复。
- Spark 会根据数据在 HDFS 的 Block 所在的位置确定 RDD 分片的优先计算位置，即 Spark 进行任务调度时会尽可能少的移动数据，而是将计算“移动”到数据所在位置。

2.2.2 大规模图数据处理框架

图数据的迭代计算框架本质仍然是分布式计算过程，如何分发、调度、整合计算是一个十分复杂的过程，在上世纪 80 年代，英国计算机科学家 Viliant 提出了整体同步并行计算模型，即 BSP(Bulk Synchronous Parallel) 模型^[44]，这对后世的并行计算的发展产生了深远的影响。BSP 模型是一种异步多指令流多数据流(MIMD) 模型，它将计算分成一系列的超步(Superstep) 的迭代(Iteration)。从纵向上看，它是一个串行模式，而从横向上看，它是一个并行的模式，每两个 Superstep 之间设置一个栅栏(Barrier)，即整体同步点，确定所有并行的计算都完成后启动下一轮超步。

目前针对图的并行计算框架已经非常多，下面就列举部分主流图计算框架并进行简要介绍，这些图计算框架的发展过程如图2.1所示，这里重点指出其发展过程中遇到并解决的问题，并在最后给出本文最终使用 GraphX 的原因。

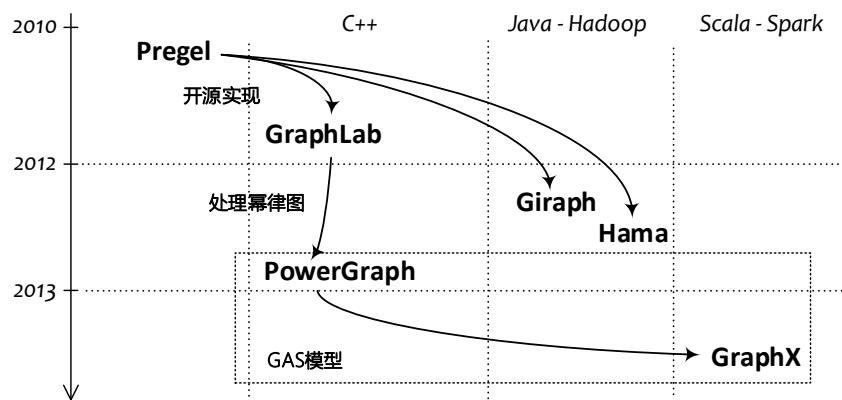


图 2.1 (分布式) 图计算框架的发展历程。
Fig. 2.1 Development of (distributed) graph computing framework.

(1) Google Pregel，2010 年，谷歌发表了名为 Pregel 的分布式图处理系统的详细设计论文，其被认作是谷歌在后 Hadoop 时代的“新三驾马车”之一，在分布式图计算框架的发展上有重要地位，Pregel 将 BSP 模型成功的应用到图计算

上，其将 BSP 模型首次应用到图计算上，并实现了基于 GFS 的 Pregel 框架，自此，根据 Pregel 思想，研究者们开始提出各种改进和优化的开源图计算框架，直到今天，Pregel 依然对该领域有这非常大的影响。

(2) Giraph 和 Hama^[45]，Giraph 对用户来讲是一个基于 Pregel 模型的图计算模块，但是对于 Hadoop 来讲，它其实是一个普通的 MapReduce 任务。在运行时可以把 Giraph 的图计算任务看成一个 MapReduce 任务，但巧妙的是，Giraph 并不重写 Map 和 Reduce 函数，而只重写了 run 函数，而且有关于 Pregel 模型的实现均在此 run 函数内。Hama 由于和 Giraph 同属 ASF 社区的同一时期的孵化项目，所以常常被一起比较，不同于 Giraph 的是它是一个纯粹的 BSP 模型的 Java 实现，其目的不局限于图计算，意在提供一个通用的 BSP 模型的应用框架。但正因为它的纯粹，使得在处理迭代的图计算任务时同 Giraph 相比有较大差距，一般很少用于实际生产。

(3) GraphLab^[46] 和 PowerGraph^[47]，在 Pregel 被提出的同一年，来自卡耐基梅隆大学的 Select 实验室就使用 C++ 实现了开源的 GraphLab 计算框架，其特别之处在于 GraphLab 并没有直接使用 Pregel 遵循的 BSP 模型的思想，而是对其进行了一系列修改，最显著的特征就是取消了 BSP 模型中每个超步运行完成后的全局同步点，这就使得计算可以完全异步进行，从而加快任务的完成时间。两年后，原团队由针对自然图幂律分布（Power-law）的特点，指出了 GraphLab 的不足，并进而提出了新的图计算框架 PowerGraph（后期被转化为 GraphLab2.0 的核心代码），这里的自然图（Natural Graph）即实际生活中接触到的真实图数据，这种数据一般有两个特点：包含超级节点，且节点的度符合幂律分布。

Pregel 处理超级节点时，会向邻接点发送消息，这就使得超级节点会发送大量消息给邻接点，造成单个 Worker 的重度负载。GraphLab 是基于异步锁机制的，和 Pregel 类似，处理超级节点时会产生大量锁，同样严重影响处理效率。导致上述问题的核心是其对图的切分策略为边划分。边划分方式的随机 Hash 分区策略只能保证节点均匀分布在整个集群中，边则是被分为两份储存在集群中。对于自然图来说，边的数量远大于节点数量，因此边划分会使各个工作节点的存储和计算不均衡，且这种不均衡在集群规模变大时更为突出。

由此，超级节点的存在是自然图难处理的根本原因，而低效的边划分策略是造成计算瓶颈的直接原因。PowerGraph 对于这两个问题给出了解决方案：对于超级节点，其提出了 GAS 计算模型；对于划分策略，其提出了点划分来避免边划分在存储和计算上的不均衡，这非常有利于符合幂律分布的图分区。

GAS 计算模型：

- Gather 阶段的主要工作主要发生在各个计算节点，搜集这个计算节点图数据中某个顶点的相邻边和顶点的数据进行计算（例如在 PageRank 算法中计算某个顶点相邻的顶点的数量）。
- Apply 阶段的主要工作是将各个节点计算得到的数据（例如在 PageRank 算法中各计算节点计算出来的同一节点的相邻节点数）统一发送到某一个计算节点，由这个计算节点对图节点的数据进行汇总求和计算，这样就得到这个图顶点的所有相邻节点总数。
- Scatter 阶段的主要工作是将中心计算节点计算的图顶点的所有相邻节点总数发送更新给各个计算节点中，这些收到更新信息的节点将会更新本计算节点中与这个图顶点相邻的顶点以及边的相关数据。

(4) Bagel/GraphX，在Spark 0.5 版本中，作为 Pregel 开源实现的 Bagel 诞生了，其可以提供类似 Pregel 的功能，但由于这个版本非常原始，其图计算的功能和性能都比较弱。到后来，业界对分布式图计算的需求日益增长，在此背景下，Spark 借鉴 PowerGraph 的 GAS 模型开发出了独立分支模块 GraphX，并在 Spark 1.0 版本中被正式投入使用，至此，GraphX 代替 Bagel 开始作为 Spark 的主要部分之一来专门处理图计算任务。

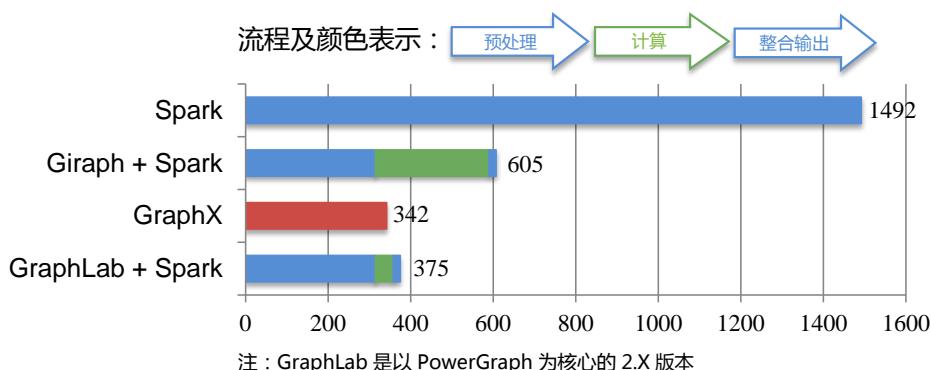


图 2.2 Pipeline 模式下的性能分析-10 次 PageRank 的运行时间。

Fig. 2.2 Performance test based on pipeline mode - Times of 10 PageRank iterations.

GraphX 在 13 年提出，实际上其和 PowerGraph 及 GraphLab 是一脉相承的，都有相似的研究团队背景并均继承了前者优良的特性，由于 GraphX 基于 Spark 来实现，这使得其兼具以下优点：

- 使用 GAS 计算模型而非传统的 Pregel 模型（提供低层基于 GAS 模型的类 Pregel 接口），能够高效处理数据中存在的超级节点。
- 使用点划分的策略，尽可能减少（边）副本的跨集群数量，从而使存储

和计算更加高效。

(c) 基于 Spark 实现，无缝衔接 Spark Streaming 等应用模块，能够以最快速度完成流水线作业。

(d) 继承 RDD 的全部特性，为用户提供 RDD 粒度的封装接口，允许将复杂的图处理逻辑用若干算子操作的集合表示。

在最近的版本，GraphX 针对其核心函数 *mapreduceTriplet* 进行了多种优化，这些优化对于 Pregel 以及所有上层算法工具包的性能，都有着重大的影响。在此之后 GraphX 的性能逐渐逼近 GraphLab。同时 GraphX 依托于 Spark 生态，可以完成一整套完整的图处理流水线（Pipeline），所以，虽然在单纯的迭代计算阶段 GraphX 或许没有性能优势，但以流水线角度（包括图的构建、合并、结果查询等）来看，GraphX 可以无缝衔接各个阶段，相较于其他方案可以提供更为优秀的表现，在如图2.2所示的测试中，测试的是从输入到输出的整个流程的全部执行时间，这时的 GraphX 就体现出其和 Spark 结合的优势了，能够在比 GraphLab 更短的时间里完整整个流程。

2.2.3 大规模图布局算法

在2.2节中提到的算法大多仍运行在单机环境，它们的基本思想是在处理器之间划分顶点集，并在整个计算过程中尽可能地保持数据计算的局部性。这部分算法大多是以 TLAV (Think-Like-A-Vertex) 模式来进行算法设计。TLAV 是一种以顶点为中心的方法，其从顶点而非图形的角度设计分布式算法，可用于重新解释许多集中式迭代图算法。

随着图处理框架的成熟，基于不同框架的图布局算法也不断涌现。2015 年，Hinge 和 Auber 基于 GraphX 在 Spark 框架中实现了分布式力导向算法^[48]，但方法基于 MapReduce 而非 TLAV 范式进行设计，并未充分利用 GraphX 的处理能力。其方法效率并不高效，对 8000 顶点和 35000 条边的图进行布局需要花费 5 小时（16 台机器 ×48GB RAM），针对这一点，在第4章本文借助 GraphX 来实现遵循 TLAV 范式的图布局算法。

与此同时，基于 Fruchterman-Reingold 模型的分布式版本 GiLA 在 Giraph 框架中被实现^[49]，其在流程上基本遵循 BSP 模型进行，且在初始对数据针对 worker 进行了划分（以边数最少来划分），虽然最终成功实现了布局计算并完成了详细的测试与分析，但作为最初的遵循 TLAV 范式的布局算法在分布式计算框架上的探索，该算法仍然有很多不足，例如 Giraph 由于基于 BSP 模型实现从而导致不适用于幂律图（自然图）的缺陷。

目前，有关于大规模图布局算法已经不仅仅停留在将已有算法实现为分布式版本，而是越来越多的从其他角度入手对该问题进行解决，诸如和 OLAP 方向结合或者利用降维等映射等思路从其他角度完成图的布局。

2.3 图数据抽样算法

图数据规模的增大，使得在对图数据进行可视化以及其他一些数据挖掘操作时必须对其进行抽样操作来降低数据的规模。用户通常希望获得的抽样子结构可以很好的代表原结构，使抽样结构与原结构具有相似的各类特征，从而利用抽样关系图代替原始关系图来进行仿真、观测以及挖掘分析。根据 Leskovec 等人发表在 SIGKDD 上的文章^[50]，现有的图抽样算法可以大致分为三类，包括基于点选择策略的随机抽样算法、基于边选择策略的随机抽样算法、基于勘探 (exploration) 的随机抽样算法^[51]。

随机选点 (RN, Random Node) 算法，即从点集中进行抽样，抽样完毕后附加对应（两端点均被抽取）的边即可得到抽样子结构，抽样对象是整体点集，因此对于无标度网络（自然图或幂律图），出入度低的节点容易被抽出，这导致抽样结果不具备很好的代表性，抽样误差较大。

随机选边 (RE, Random Edge) 算法，即从边集中抽取，最终追加对应的节点（被抽取边的两个端点）即得到抽样结构。对于幂律图来说，由于是边选抽取，因此最终关联边较多即出入度大的节点容易被选中，由于抽样时需要多次扫描边集，且对于幂律图来说边的数量巨大，因此，该类算法不具备良好的效率。

基于探索的抽样算法，经典的有随机游走算法 (RW, Random Walker)。两者一般都有随机的起始抽样点 (seed)，递归地执按固定的抽样逻辑对整个结构进行抽样，这类算法的优势在于抽样结构必然连续且为原结构的真子集；缺点在于虽然该类算法较 RN 和 RE 考虑了更多的结构因素，但其效果仍然有限，对其实值敏感、终止条件判断困难以及拓扑特性无法保证等问题仍然存在。

滚雪球算法 (SBS, Snow Ball Sampling) 是传播类型的算法之一，其核心思路是开局随机选择几个节点，再由该节点向外选择几个邻居向外传播，一直循环此过程。该类算法的优点是可以控制起始点以观测具体的结构（该类结构可能在整体中并不明显），其缺点是对起始位置十分敏感，会造成由部分代表整体的局限性。

针对上述抽样算法在各方面存在的问题，研究者们开始将抽样算法与分层策略相结合，用不同层级来约束算法的抽样行为，从而使抽样结果在出入度上更

加平均，进而缩小抽样的误差。这类算法的代表有 SS 算法^[52] 和最近的 SNS 算法^[51]，SS 算法有单一的抽样起点，基于最短路径分层，是本文抽样算法思路的来源之一，在第4章会做进一步介绍；SNS 算法则基于 K-Means 聚类进行分层，分层后逐层对节点进行抽取，在其论文中通过实验证明了该算法的高效性和有效性。

2.4 本章小结

本章主要介绍了图布局计算的相关工作：以时间顺序总结了当前单机串行和单机并行图布局计算的特点和不足；围绕数据规模的变化，分析了各阶段遇到的问题并给出了解决问题的相关研究内容；概述了主流分布式图处理框架的发展历程并分析了选择 GraphX 的原因，同时针对基于分布式框架开发的布局算法做了简要介绍。最后，在图数据抽样方面详细介绍了现阶段的各种抽样方法，并指出了这些方法存在的问题。

第3章 面向大图的分层抽样算法

图数据相较于一般数据而言，除了携带属性信息的节点之外，还有节点之间表示关联性的边，一般抽样算法往往只能孤立的针对点集或者边集进行抽样，其结果通常会对原始结构产生很大的破坏，而能够保留拓扑结构的随机游走式的抽样方法又很难按照理论的既定设想完成抽样。本章将对这两种思路分别进行详细分析，结合二者的优缺点给出本文提出的抽样算法，并对其抽样效果给出了实验验证。

3.1 分层抽样算法

图数据规模的增大，使得在对图数据进行可视化以及其他一些数据挖掘操作时必须对其进行抽样操作来降低数据的规模。用户通常希望获得的抽样子结构可以很好的代表原结构，使抽样结构与原结构具有相似的各类特征，从而利用抽样关系图代替原始关系图来进行仿真、观测以及挖掘分析。

3.1.1 相关定义和符号说明

图 (Graph)。本文使用 $G = (V, E)$ 来表示图（无向图），其中 V 是顶点集， E 是边集。如果两个点 v_i 和 v_j 组成一条边，那么将其表示为 $i \leftrightarrow j$ 或 $\langle i, j \rangle$ 。

抽样 (Sampling)。是实验中常用的一种处理数据的方式，其最终要的就是要确保抽取的样本集合对数据集整体有充分的代表性，体现在图数据上，即要求抽样出的子图能够有效的代表原图的拓扑结构特征。

图数据抽样。沿用 $G = (V, E)$ 表示原始图结构， $G_s = (V_s, E_s)$ 表示抽样出的结构，则应有 $V_s \in V, E_s \in E$ 。

节点的度 (degree)：代表图中该节点所拥有的全部连接（边）的数量，对于出度和入度来说，就是分别计算出边和入边的数量。

度分布 (degree distribution)：是针对图结构整体的特征，代表整个图中节点度数的总体描述，其数学定义为网络中全部节点的度的概率分布或频率分布，如公式3.1所示，其中 C_V 表示图中全部节点的数量， C_k 表示度为 k 的节点数量。对于随机图来说，其度分布就是指的图中顶点度数的概率分布。

$$P(X = k) = C_k / C_V, k = \{d_{\min}, \dots, d_{\max}\} \quad (3.1)$$

最短路径 (shortest path): 对于加权图, 两互相关联的节点之间的最短拓扑距离即为边的权值, 对于非加权图, 则相连的两点之间的距离为 1, 最短路径目的是寻找图中两结点之间通过边能够联通的距离最短的路径 d_{sp} 。算法涉及的情况非常多样, 计算特定节点 v_i 到其余 $N - 1$ 节点的最短路径的算法被称为单源最短路径算法 (SSSP)。

图的直径 (diameter): 即整个图中最长的节点间最短路径的距离。通过一次单源最短路径算法可以确定从某个节点 v_i 出发的最大 d_{sp} , 通过对全部 N 个节点进行求取最大 d_{sp} 的过程后, 即可得到该图结构的全部最短路径的最大值, 即该图的直径。

关键节点: 对于本文的研究对象图来说, 用户能够理解的信息 (这里主要指节点的稀疏程度、聚集程度以及轮廓特征) 大部分直接来自布局结果本身, 而在图布局算法执行的过程中, 有几类节点对布局结果有这很大的影响, 在本文提出的分层抽样算法中, 这部分节点包括以出入度来衡量的超级节点和基于 K-core 的最内层节点两部分。

3.1.2 SS 抽样算法

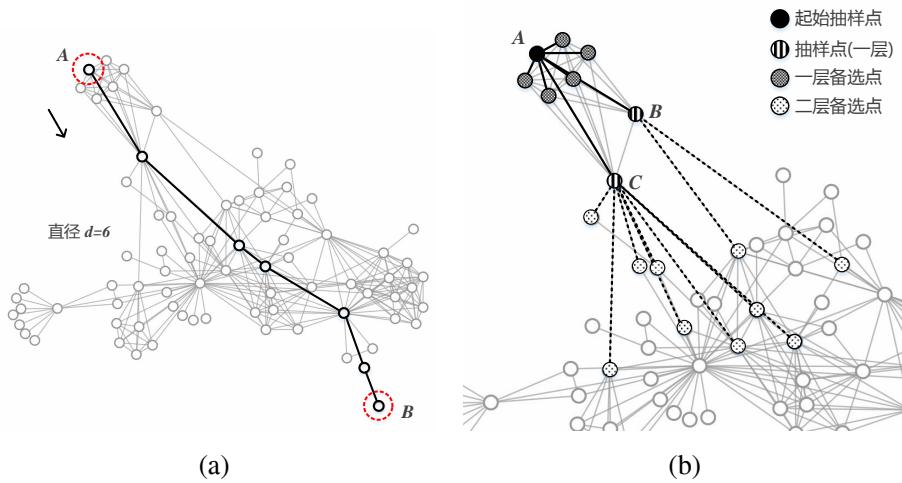


图 3.1 SS 分层抽样算法。(a)示例图结构及直径, (b)理想的抽样情况。

Fig. 3.1 Diagram of SS algorithm. (a)Sample diagram structure with diameter, (b)Ideal sampling structure.

目前已经有许多应用在图数据的抽样算法, 大致基于三种方式来实现: 随机选点、随机选边和启发式抽样。

基于随机选点的抽样思想是计算图中每个节点的重要性 (权值), 然后根据特定规则进行抽样; 随机选边同样采用类似的方式, 只是计算边的权值; 当按一定比例对较大规模的数据进行抽样时, 这两种抽样算法都有可能导致抽样结果

出现孤立点的情况，即会严重破坏原数据的拓扑结构。基于启发式的图抽样算法通常具有较好的连贯性，其一般通过模拟病毒扩散或随机游走的方式来实现，但这一过程一般是递归进行的，很难控制游走结果（即抽样结果）的规模。

图的分层抽样抽样算法 (SS, Stratified Sampling)^[52] 由 Yueping Li 等人首次提出，其目的是针对大规模的图数据进行抽样，具体流程如算法3.1所示，首先需要获取整个图的直径 d ，通过直径来确定抽样的起点；然后其将全部节点按照距起始点拓扑距离的长度分为不同的集合，距离为一的点（邻节点）是第一层，为二的节点是第二层（见图3.1b），以此类推得到 d 个集合，这些集合即体现算法“分层”的特性，依次从不同集合中按相应比例对不同类型（是否和上一层有关联）的节点进行抽取，集合全部抽取完毕后算法结束。

Algorithm 3.1 SS algorithm^[52]

```

1: 输入: 节点数为  $N$  的原始图  $G$ , 抽样百分比  $P$ , 随机数  $k$ 
2: 输出: 抽样子图  $G_s = (V_s, E_s)$ 
3: 计算  $G$  的直径  $diameter$ , 并得到对应路径  $l$ 
4: 任意选取  $l$  的一个端点  $v_{start}$  作为参考点
5: 计算其余  $N - 1$  各节点到  $v_{start}$  的拓扑距离, 并分别划分到  $diameter$  个集合
6:  $V_s \leftarrow v_{start}$ 
7:  $i \leftarrow 1$ 
8: while  $diameter > i$  do
9:    $V_i$  是离  $v_l$  拓扑距离为  $i$  的点的集合
10:   $V_{connected} \leftarrow V_i$  集合中与  $V_{i-1}$  集合内有相连的节点
11:   $V_{disconn} \leftarrow V_i$  集合中与上一层无关的节点
12:  在  $V_{connected}$  中抽取  $k * P$  的节点加入到  $V_s$  中
13:  在  $V_{disconn}$  中抽取  $(1-k) * P$  百分比的节点加入到  $V_s$  中
14:   $i \leftarrow i + 1$ 
15: end while
16: 得到  $V_s$  对应的边  $E_s$ 
17: return  $G_s = (V_s, E_s)$ 

```

该算法有以下几个优点：首先，直径的使用确保了构造集合的完整性和独立性，对于每一个联通分量来说，以此规则划分集合可以将全部节点均不重复的分开到不同集合中；其次，对每个集合进行抽样可以确保抽样的离散程度，对比类似随机游走的算法来说，其不存在异常的终止路线；最后，在对每个集合进行抽样时将集合内的点分为与上一层相连和不相连的部分，分别对这两部分进行抽样，这样也确保了直接连通以外的节点也能被抽取，对保留结构特征更为有利。

分析该算法可知，该算法仍可能出现的异常情况是抽样结构不连贯，不能保证抽取结构的完整性。下面，主要介绍了通过引入 K-core 分解等技术来完成对其的改进。在介绍该内容之前，在3.2小节简要回顾一下 K-core 分解。

3.2 基于 K 核分解的分层抽样算法

3.2.1 K-core 分解

K-core 思想是一种分解思想，K-core 结构就是不断去掉图中出入度小于等于 K 的节点后的剩余结构，在此，不同的 Core 结构即对应不同的抽样层级，该结构内的所有节点的出入度均大于 K，对于 K-shell 而言，其表示的是 K-core 层在 $(K - 1)_{core}$ 层的补集，对于固定的图结构来说，core 和 shell 符合如下关系：

$$(K-1)_{core} = K_{core} + K_{shell}, \quad K \geq 1 \quad (3.2)$$

对于 K-core 这一算法在图数据上已有不少应用，在 2016 年国内学者提出的 CABK 算法^[53] 中，将 1-shell 从原结构中去除，完成对图的“压缩”，处理前后的布局如图3.2所示，可以看出 CABK 算法还停留在比较初级的应用阶段，只是 K-core 思想和图数据的简单融合，且此算法对图结构的压缩性也十分有限，仅去掉 1-shell 内的部分并不能解决根本问题，在下节中将给出本文的改进方案。

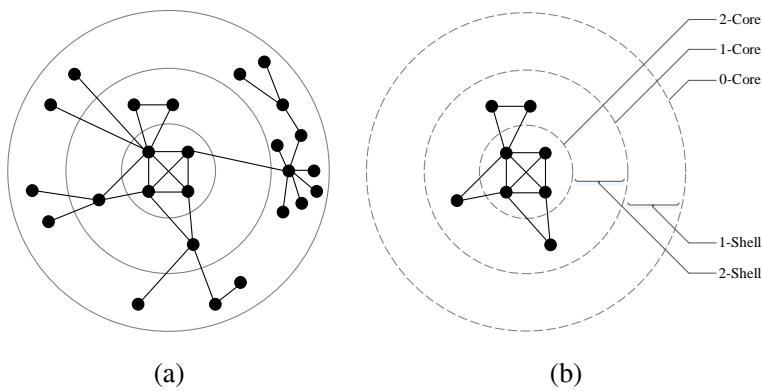


图 3.2 CABK 压缩算法处理前后的节点分布。(a) 原结构, (b) 压缩结构。

Fig. 3.2 Diagrammatic sketch of SS algorithm.(a) Demo graph with diameter, (b) An ideal sample case.

3.2.2 改进后的分层抽样算法

对于上述的 SS 算法和 CABK 算法，其本身在设计时均未考虑在大规模图数据上的应用（非单机），因此，下面给出其所对应的问题，并给出本文的改进方案：

(1) 对于算法3.1, Step.9 需要借助最短路径来求取距 v_{start} 距离为 i 的点，这对于大图来说计算代价巨大，需要计算每个点同 v_{start} 的最短路径，然后根据拓扑距离划分为不同的集合。所以，此处的核心任务是对原数据进行数据分层（集合），为了加速此步骤的执行，本文使用了 K-core 思想来完成数据的分层工作。

(2) 分析 SS 算法可知该算法必须依赖单一起始点来对节点进行分层划分，而该节点的选取需要寻找到图结构的直径，通常图数据集会带有直径信息，但该算法流程仍需要找到一条确切的直径才能开始抽样，如果选取错误，则会直接导致整个划分的出错，对起始点和划分层数非常敏感，而且获取直径的过程计算代价也比较高。相比较而言，本文提出的抽样算法由于已经引入 K-core 思想来分层，具体是以 K-shell 来划分的，所以这里自然地需要定义核心 k 值，但这里的划分结果对 k 值并不敏感，因为依照此思想进行的分层是由外而内、逐渐深入的，具有全局的一致性，即执行多次会有相同的分层结果。所以，改进后的分层部分不需要单独指定最终的 k 值， k 值可以由算法来灵活确定。

(3) 对于 CABK 算法，在处理大图数据时仅靠去除 1-shell 节点的最终效果十分有限。对于该部分的改进，主要是将“删除”改外“抽样”，结合 SS 算法的分层思想，将 K-shell 作为分层的依据，然后在采用拓扑分层抽样的算法思想对完成分层的不同集合进行抽样，继而完成整个抽样（压缩）流程。

综合上述改进内容，文本提出了基于 K 核分解的分层抽样算法（K-SS，K-core based Stratified Sampling），整个流程主要包括三个部分：数据的分层、超级节点的补充、分层抽样的执行。详细过程如下：

(1) 数据的分层

以 K-core 思想对原图结构 G 进行分层，这里在计算 K-shell 时，间隔即分层的层数 M 取决于数据的原始规模，在处理给定直径 d 的数据集时（例如 SNAP 数据集^[27]），直接以 d 来定义分层数 M （层数 M 也可以由用户自定义），取两倍的 D_{av} (D_{av} 是该结构中节点的平均度) 作为 k_{max} ，则对应的间隔 T 应该是 $2D_{av}/d$ ，这一指标的选择仍需更细致的实验验证。例如，处理数据的直径为 5，节点的平均度 D_{av} 为 30，则 $M = d = 5$, $T = 60/5 = 12$ ，即 k 的取值应为 {12,24,36,48,60}，对应到 K-core 分层时，对应为 5 层节点集合 X_1, \dots, X_5 ，即 $X_1 = S_{12}, X_2 = S_{24}, X_3 = S_{36}, X_4 = S_{48}, X_5 = C_{60}$ ，其中 C_k 和 S_k 分别代表 K-core 和 K-shell 中所包含的节点，核心层使用 Core 包含的节点，外层均采用 shell 包含的节点，0-shell 没有实际意义故没有对应集合。

(2) 超级节点的补充

通过对第 2 章中的图布局算法的介绍可知，斥力和引力的计算中，引力直接通过边来作用两端的节点，换句话说，对于出入边数目非常大的节点，其所受其他节点的引力，和本身作用于其他节点的引力都是直接影响布局结果的。如果希望通过采样得到的子图能够和原图有相似的布局结果，那么此类关键节点必

Algorithm 3.2 KSS 获取原始图数据分层 $\{X_1, \dots, X_M\}$

```

1: 输入:  $G = (V, E)$ ,  $d$ 
2: 输出:  $\{X_1, \dots, X_M\}$ 
3:  $M \leftarrow d$ 
4:  $D_{av} \leftarrow \text{getAverageDegree}()$ 
5:  $K_{max} \leftarrow 2D_{av}$ 
6:  $T \leftarrow \lfloor K_{max}/M \rfloor$ 
7:  $X_M \leftarrow \text{getCoreSet}() // C_{kmax}$ 
8:  $i \leftarrow K_{max} - T$ 
9: while  $i > 0$  do
10:    $M \leftarrow M - 1$ 
11:    $X_M \leftarrow \text{getShellSet}(T * M) // (S_0, \dots, S_k)$ 
12:    $i \leftarrow i - T$ 
13: end while
14: return  $X = \{X_1, \dots, X_M\}$ 

```

须要得到保留。对于步骤（1）集合中的核心集合，即 C_{kmax} （算法3.2第7行），不仅仅作为抽样流程中的一层，其还需要作为能代替整个图结构的概要图结构，因此单纯的依靠 K-core 流程得到的核心结构并不会包含全部的关键节点，例如图3.2a中 1-shell 内包含了整个结构中出入度第二大的节点，但其并不在该结构的核心集合 C_2 (2-core) 中，因此为了降低抽样对布局的影响，出入度大的节点必须被考虑在内，在这个过程中，这部分节点先被独立的选择出来，之后再合并到数据分层后的核心 Core 层内。

Algorithm 3.3 KSS 获取抽样结构 G_s

```

1: 输入:  $V_{sup}$ ,  $C_k$ ,  $S_0, \dots, S_k(X_1, \dots, X_M)$ ,  $T$ ,  $p$ 
2: 输出:  $G_s = (V_s, E_s)$ 
3: foreach  $v \in S_k$  do
4:   if  $\text{Corevalues}(\text{Neighbors}(v)) == k$  then  $V_{conn} \leftarrow v$ 
5:   else  $V_{disconn} \leftarrow v$ 
6: end
7:  $i \leftarrow k - T$ 
8: while  $i > 0$  do
9:   foreach  $v \in S_i$  do
10:    if  $\text{Shellvalues}(\text{Neighbors}(v)) \in [i - T, i]$  then  $V_{conn} \leftarrow v$ 
11:    else  $V_{disconn} \leftarrow v$ 
12:   end
13:    $V_s += \text{Sampling}(V_{conn}, k * p)$ 
14:    $V_s += \text{Sampling}(V_{disconn}, (1 - k) * p)$ 
15:    $i \leftarrow i - T$ 
16: end while
17:  $E_s \leftarrow \text{getEdges}(V_s)$ 
18: return  $G_s = (V_s, E_s)$ 

```

这里采用经验值 20% 来区分节点的关键与否（关键节点集合用 V_{sup} 表示），这一数值来源于经典的巴莱多定律^[54]（二八定律），已在各种社会网络结构的处理中被证明有效，在这里作为划分的默认值。通过对部分社会网络数据集的超级

节点的检测，可以发现这部分节点占比均非常小（ $< 3\%$ ），在合并入分层的核心集合后，不会过多的增加其布局计算量。

(3) 分层抽样的执行

如果用 V_{sup} 表示步骤(2)获得的超级节点集合，那么在分层工作结束后，核心层 C_{kmax} 已经变为 $C_{kmax} \cup V_{sup}$ ，此时，两部分节点一起连同对应的边一起，构成整个图结构的概要图 G' ，需要注意的是可能出现 V_{sup} 中的点并非与 C_{kmax} 相连，对于不相连的点这里直接从 V_{sup} 剔除。

综上，在经由三个阶段的处理后，由输入图 $G = (V, E)$ 可以得到输出的抽样结构 $G_s = (V_s, E_s)$ ，此时的 G_s 即为原结构 G 的抽样概括。

在介绍完具体的算法流程后，下面通过图3.3来具体说明 KSS 算法的执行过程。在图示实例中，抽样对象为如图3.3a所示的由 25 个节点组成的结构，整体可分为颜色不同的三层（每层的 K 值不同）；在如图3.3b所示的首层结构中，节点 v_1, v_2, v_3, v_4 为 1-Core 集合的全部节点，节点 v_9 为前 20% 度分类中包括的节点（此处只有一个），这五个顶点构成的结构共同组成初始的首层结构，这也是用户能看到的最顶层的结构；之后，在图3.3c中，在图3.3b的基础上对最外层节点进行抽样，图示结果即为最终的 G_s ，在最终的原型系统中 G_s 将代替 G 展现给用户。需要注意的是这里并未涉及布局算法，给出的形状结构只是为了方便描述。

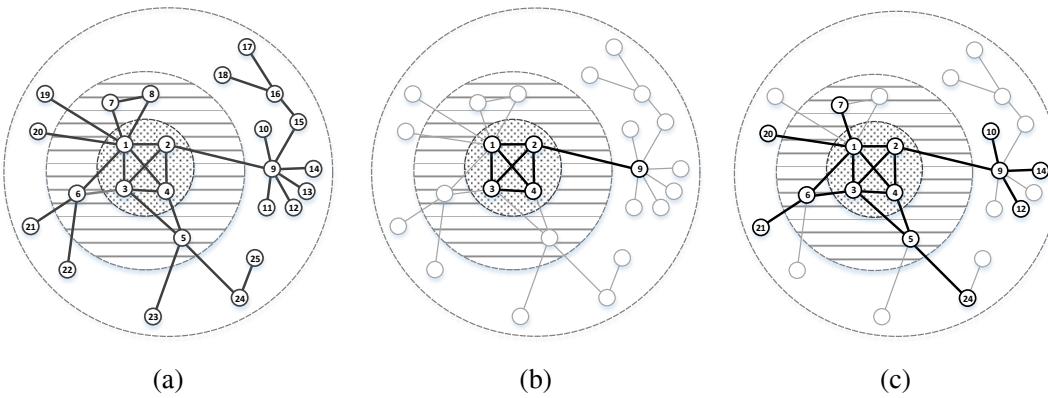


图 3.3 KSS 抽样示例。(a) 示例结构, (b) 首层结构, (c) 最终结构。

Fig. 3.3 Demo case of KSS. (a) Demo graph, (b) First layer structure, (c) Final layer structure.

3.3 抽样结构对比指标

由度分布的定义可以知道其可以代表整个图的组成特征，例如对于幂律图来说，其度分布是典型的长尾分布，自然地，对于抽样后的图，其也应当符合形状相同的长尾分布，如果度分布形状有较大差异，自然会对布局结果产生影响。由于幂律图中超级节点的数量占比极少，这也会导致在抽样中有“重要作用”的

超级节点很容易被漏掉，从而使度分布形状发生改变，所以本文采用了关键节点的保留策略来避免这种情况。

在比较两者分布的相似性时，使用 KL 散度来将此问题转化为数值问题，假设原始图结构的节点度分布为 $p(x)$ ，抽样结构节点的度分布为 $q(x)$ ，按公式3.3来计算两者的 KL 散度值（使用最终形式进行编码计算），注意此时抽样分布 $q(x)$ 的分量必然对应小于等于 $p(x)$ 的分量，即对于抽样前后的分布比较，有 $p(x)/q(x) \geq 1$ 。对于此公式，若要使 $D_{KL}(p \parallel q)$ 的值很小，则 $q(x)$ 要和 $p(x)$ 高度接近，否则 $p(x)/q(x)$ 比值变大会使 $D_{KL}(p \parallel q)$ 也变大。即可以用 $D_{KL}(p \parallel q)$ 的值来衡量抽样结构同原结构度分布的相似性。

$$D_{KL}(p \parallel q) = \int_x p(x) \ln \frac{p(x)}{q(x)} dx \rightarrow \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \quad (3.3)$$

此外，抽样前后结构的数值特征也需要进行比较，包括度分布、聚集系数以及平均度；度分布的差异性可以衡量抽样算法在节点选择上的特征，类似随机选点（RN）的算法会使整个结果的度分布出现较大差异，因为出入度大的超级节点同叶子节点在其算法中是一致对待的，这不利于抽样结构同原结构的相似性，通过该指标可以进行衡量；聚集系数考察的是结构的稀疏程度，相似的结构必然有相似的稀疏程度；平均度能够衡量整个结构中节点与边的数值关系，即理想的抽样结构应该和原结构有相似的平均度，例如随机选点算法在抽样率为 20% 的前提下，抽出的边也应该是原结构边数的 20%，但一般算法难以完成该任务，但其仍然可以作为一项衡量指标。

3.4 实验结果与分析

3.4.1 实验设计

由于具体评估指标的计算方法时间复杂度太大，无法直接对集群的大规模数据集进行计算，所以为了便于计算和评估算法的有效性，本章仅在单机环境下进行实验验证，实验环境为 PC 一台，配置为 Intel Core i7 6700mq，16GB 内存和 256GB 固态硬盘。

实验分为两组对照，分别是本文的 KSS 算法和其他算法，其他算法包括：随机选边算法、随机选点算法、滚雪球算法和分层拓扑抽样算法（RE，RN，SBS，SS）。

实验：验证抽样结构的相似性

实验数据集如表3.1所示,这里我们选取 ego-Facebook、email-Enron、loc-Gowalla 三个数据集进行实验,该部分数据集均为无向图结构,适合于本章的实验。实验中分别考察不同算法在 20%、40%、60% 抽样率的情况下的表现。

根据上一小节的叙述,在此实验部分的评估指标具体有: 度分布的相似性 (SM_d), 外加以下数值特征,包括全局聚集系数 (V_{gc}), 平均聚集系数 (V_{cc}), 节点平均度 (V_{av})。(在3.4.2小结的结果,均为抽样后的各值与表3.1中原始数值的比值,用 V' 表示)

表 3.1 实验数据集。
Table 3.1 Information of data sets.

数据集	节点数	边数	V_{gc}	V_{ac}	V_{av}	V_d
ego-Facebook	4039	88234	0.0281	0.6055	43.7	8
email-Enron	36692	183831	0.0107	0.4970	20.0	11
loc-Gowalla	196591	950327	0.0049	0.2367	19.3	14

表 3.2 ego-Facebook 数据集的抽样效果。
Table 3.2 Sampling result of ego-Facebook data set.

$Rate(\%)$	$Algorithm$	SM_d	V'_{gc}	V'_{cc}	V'_{av}
20	RN	105.65	25.784	3.463	0.187
	RE	45.61	4.855	3.144	0.199
	SBS	8.267	3.963	2.015	0.306
	SS	13.196	0.838	0.225	2.204
	KSS	13.343	0.823	0.167	2.373
40	RN	55.204	6.114	3.141	0.402
	RE	20.5	2.445	2.038	0.401
	SBS	3.169	3.728	1.65	0.369
	SS	10.065	0.937	0.256	1.743
	KSS	9.507	0.941	0.238	1.735
60	RN	22.268	2.767	2.243	0.601
	RE	10.364	1.662	1.707	0.595
	SBS	4.64	2.814	1.509	0.464
	SS	6.382	0.993	0.352	1.42
	KSS	5.474	0.997	0.329	1.406

3.4.2 实验结果及结论

在本章实验中,使用的数据集规模在节点数的规模上分别为 1k、10k、100k,在边数量上的规模分别为 10k、100k、1000k, 分别对这些数据集在 20%、40%、60% 的抽样率下进行抽样,然后比较抽样后的结构同原结构在度分布、全局聚集系数、局部聚集系数及节点平均出入度上的差异,以此来说明 KSS 相较于其他四类算法的更优秀的表现。最终数据结果详见表3.2、表3.3及表3.4, 图3.4是由实验数据所得,表示了各抽样结果同原结构度分布上的差异性。分析数据结果可

表 3.3 email-Enron 数据集的抽样效果。
Table 3.3 Sampling result of email-Enron data set.

<i>Rate(%)</i>	<i>Algorithm</i>	<i>SM_d</i>	<i>V'_{gc}</i>	<i>V'_{cc}</i>	<i>V'_{av}</i>
20	RN	120.97	64.102	0.512	0.21
	RE	63.8	10.775	0.731	0.208
	SBS	54.605	3.637	0.596	2.062
	SS	58.965	1.174	0.125	3.247
	KSS	56.862	1.168	0.101	3.175
40	RN	69.891	13.741	0.713	0.421
	RE	25.891	5.321	0.782	0.415
	SBS	43.336	3.342	0.629	1.912
	SS	38.199	1.172	0.193	2.177
	KSS	30.456	1.164	0.192	1.951
60	RN	31.38	5.992	0.79	0.622
	RE	10.167	3.551	0.765	0.622
	SBS	45.905	3.153	0.633	1.835
	SS	11.098	1.154	0.277	1.539
	KSS	11.907	1.157	0.263	1.556

表 3.4 loc-Gowalla 数据集的抽样效果。
Table 3.4 Sampling result of loc-Gowalla data set.

<i>Rate(%)</i>	<i>Algorithm</i>	<i>SM_d</i>	<i>V'_{gc}</i>	<i>V'_{cc}</i>	<i>V'_{av}</i>
20	RN	452.898	12.408	0.501	0.192
	RE	206.66	2.244	0.671	0.193
	SBS	239.564	1.035	0.59	1.398
	SS	237.636	0.712	0.113	2.756
	KSS	232.134	1.542	0.108	2.851
40	RN	281.304	2.88	0.66	0.386
	RE	84.086	1.114	0.684	0.387
	SBS	207.437	0.869	0.567	1.446
	SS	167.298	0.612	0.175	2.017
	KSS	133.185	1.26	0.203	1.872
60	RN	105.168	1.227	0.683	0.582
	RE	40.49	0.738	0.66	0.58
	SBS	180.316	0.807	0.574	1.423
	SS	166.767	0.516	0.295	1.451
	KSS	111.889	1.043	0.407	1.299

表 3.5 KSS 与 SS 参数均值的比较。
Table 3.5 Comparison of valuesáverage between KSS with SS.

Algorithm	F20	E20	G20	F40	E40	G40	F60	E60	G60
SS	0.982	1.289	1.087	0.887	1.03	1.038	0.886	0.804	1.047
KSS	1.064	1.26	1.259	0.859	0.906	0.839	0.65	0.834	0.465

得以下结论。首先，RN 和 RE 两种抽样算法的度分布差异性均随数据规模的扩大而增大，这符合随机抽样的特点，但对于 KSS、SS 和 SBS 来说其差异性减小的趋势不明显，仅在规模较大的数据集上有所表现；其次，在同样的抽样参数的条件下，KSS 算法的表现仅在个别情况（如 G40，loc-Gowalla 应用 40% 抽样率）下比 RE 差（细节比较参见图3.5），该现象是由于 RE 直接对边进行随机选择，那么出入度大的节点（存在的边也多）被选中的概率就大，同样的抽样率下，RE 算法的抽样结果能够在度分布上贴近原图，KSS 算法仅次于 RE 也能够说明其在出入度方面同原结构保持着很好的相似性。

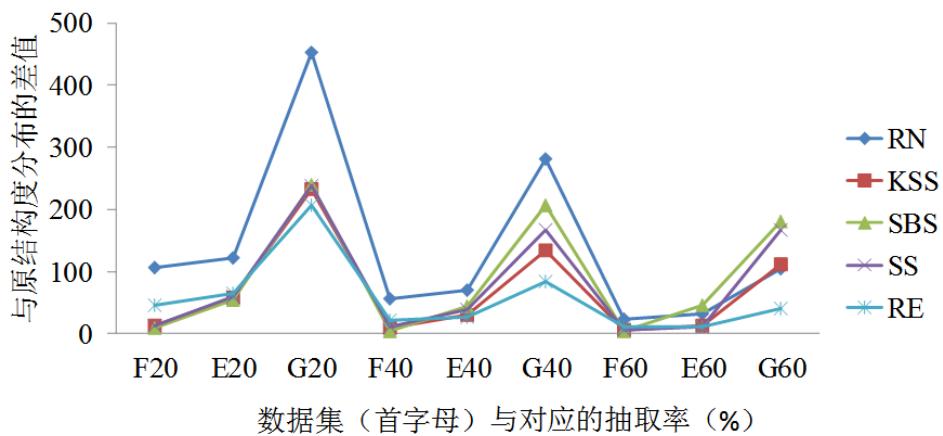
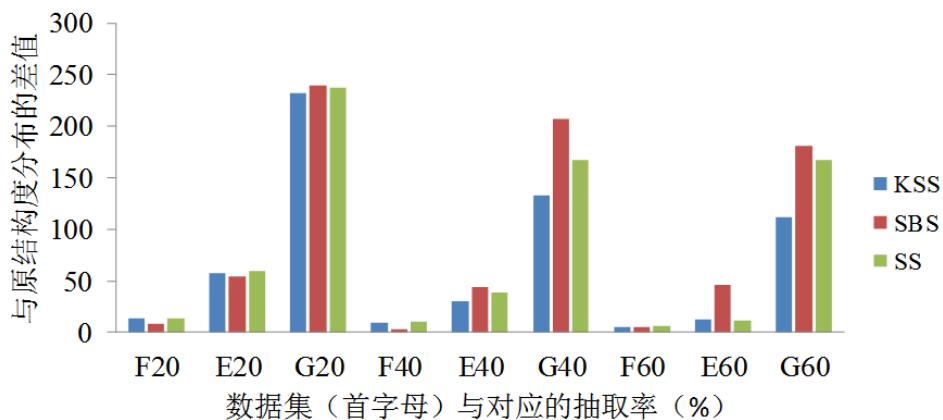
图 3.4 不同数据集对于 SM_d 的比较。Fig. 3.4 Comparison of SM_d of all data sets.图 3.5 去除 RE 和 RN 后的 SM_d 的比较。Fig. 3.5 Comparison of SM_d of all data sets without RN and RE.

图3.6显示的是全局聚类系数的差异，可以看出在该指标上随机选点和随机选边有较大的波动，尤其在小规模数据集上；由于 RN 和 RE 算法差异性过大，这里将去掉此二种算法后的全局聚集系数的比较结果通过图3.7列出，由于此处

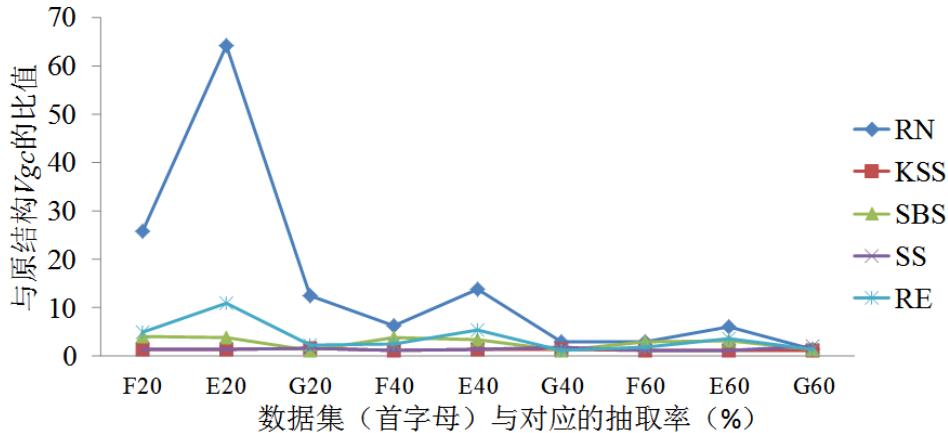


图 3.6 不同数据集对于 V_{gc} 的比较。
Fig. 3.6 Comparison of V_{gc} of all data sets.

显示的是与原结构 V_{gc} 的比值，因此这里以 1 为纵轴起始。分析图表可以发现 SBS 算法有较大的波动，而 KSS 和 SS 算法则比较稳定，且相较于 SS 算法，KSS 算法同原结构 V_{gc} 的相似程度随数据规模和抽样比率的增大而增大。

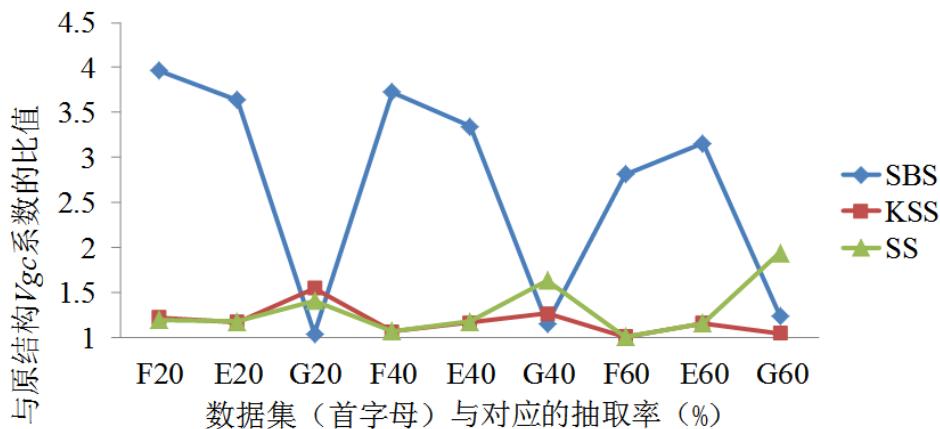


图 3.7 去除 RE 和 RN 后的 V_{gc} 的比较。
Fig. 3.7 Comparison of V_{gc} of all data sets without RN and RE.

另外，其余算法的表现由于数值过小无法准确展示，且其余指标均有类似特点，所以，这里还计算了所有指标的平均值，这里平均值的计算是通过对原实验结果进行了几步变换得到的，首先对数值型的 SM_d 进行了归一化处理，然后对比值型的结果通过倍率来完成归一化，之后将上述转换后的结果求取平均值，最终通过不同抽样率在不同数据集上的均值结果如表3.5所示，由此来重点考察 KSS 同 SS 算法的改进和提升，由图3.8可以看出，KSS 算法在抽样结构的参数比较上（转化后的均值越小越好），整体具有同原结构更相似的表现。

综上，KSS 算法在度分布上的表现比大部分比对的算法都表现更好，在有关

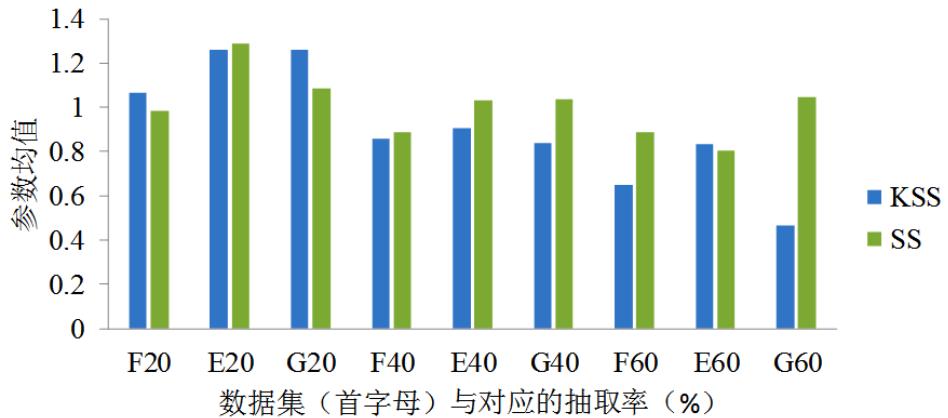


图 3.8 KSS 与 SS 所有指标均值的比较。

Fig. 3.8 Comparison of average values of all indicators of KSS and SS.

于拓扑结构的数值参数的比较上同其余算法，尤其是 SS 算法相比，同样能够打得到更好的结果，这为下一章的分层布局算法在抽样后保证结构的相似性上提供了有力的保证。

3.5 本章小结

本章主要介绍了本文提出的基于 K-core 分解的分层抽样方法，K-core 本身就是一种测量网络结构中核心结构的手段，在合适的 K 值引导下能够快速的找出整个结构的核心部分；此外，为了进一步贴合原结构的数值特征及拓扑（布局结果）特征，在本章的算法中还引入了基于度筛选的一部分节点。在提出 KSS 抽样算法后，本章对其实现做了详细介绍，并设置了两组实验证明了 KSS 算法的效果，实验表明，KSS 算法在结构相似性保证上有更优秀的表现。

第4章 分布式图布局算法

分布式图布局算法通常是借助通用的分布式存储计算框架来实现，基本遵循单机图布局算法的设计思路，但需要额外适应分布式计算框架的实现过程，因此在计算迭代和并行上与传统图布局算法并不相同。由于目前分布式图布局算法多是理论验证目的，就某一图处理框架（GraphX 或 Giraph）并不存在原生内嵌图布局算法，各大分布式计算框架均只提供细粒度的编程接口，研究者们多从顶层利用这些接口（例如 Spark 算子）来实现图布局算法，因此实现方案并不唯一。本章将首先给出基于 GraphX 的基本版图布局算法的详细设计与实现，然后给出结合第3章提出的抽样算法对原过程进行改进的布局算法。

4.1 基于 KSS 抽样的分层布局方法

分层布局或是多尺度布局，在第2章已经进行了相关介绍，本质仍然是采用Walshaw的思想^[17]，对不同尺度的 G_1, \dots, G_n 分别进行绘制，在构建不同层级视图的这一过程中，又衍生出从不同角度出发的不同的解决方案，下面本文从抽象和具体两种构建角度来对分层布局方法的原理过程进行阐述。

4.2 两种不同的分层策略

对于层间抽象的构建流程来说，不同层级的节点及边代表的意义各不相同，顶层节点是下层结构（点以及关联的边）的抽象表示，顶层边则是下层结构与结构之间的关联的重新映射。此类流程往往伴随显式的聚类算法，需要通过聚类来区分不同的结构（结构内部通常是致密连接）。采用该类算法，可以使待显示结构以抽象视图的形式快速反馈给用户，而用户则可以通过层层抽象的引导，最终找到最底层的感兴趣的（真实）结构。

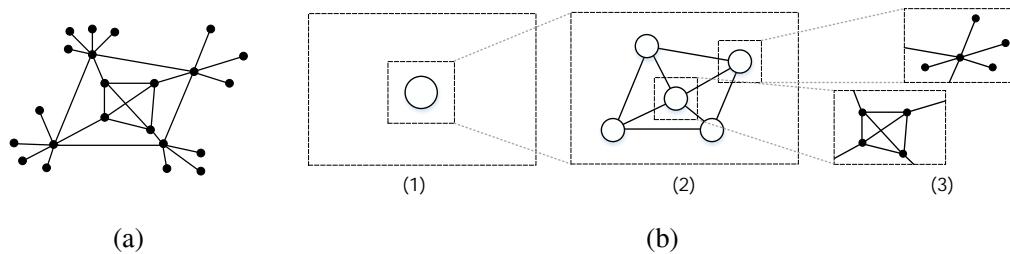


图 4.1 抽象分层示意图。(a) 示例结构, (b) 对应的用户访问顺序, 构建顺序相反。
Fig. 4.1 Diagrammatic sketch of SS algorithm.(a) Demo graph with diameter, (b) An ideal sample case.

对于每层结构均为原图结构的构建流程来说，上层结构一般都是下层结构的子集，各层视图的节点和边均源自真实结构。类似于图4.2a的北京市路网的缩放，在图a-1中只显示主干道路，在更聚焦的图a-2中则显示更为详细的道路信息；这里无论是a-1还是a-2中的道路均为原路网中的道路而非抽象表示，这是和基于抽象的分层方法最大的区别。对于这种分层策略来说，其关键点即如何选择出具有代表性的“主干道”，对于图数据的结构选取，最朴素的方法就是抽样，分别抽取不同比例的点或边来作为不同层级的视图。采用该类算法，同样原真实结构的小部分子图率先反馈给用户，用户根据需要，来选择相应粒度的层级进行观察或研究。

对于抽象的构建流程，除了最底层的真实结构外，顶层的抽象均需要刻画对应底层拓扑结构的特征，即各抽象节点/边必须包含额外的、能够给用户提供“内视”能力的语义抽象信息，否则，用户会在选择抽象实体的过程中迷失方向。而目前绝大部分抽象的构建流程均会导致高级抽象节点的出边是内部结构整体出边的总合，这样直接破坏了原结构的关联特征，在经过层层抽象之后，用户面对的是完全丢失原始真实结构信息的抽象结构，所以基本无法有效的对感兴趣的部分进行探索。另一个十分重要的缺点，就是采用该类算法，会使得用户的查看“视野”从全局变为局部，虽然基于原始结构的构建也需要视野的缩小来聚焦更详细的细节，但其是作为整体进行显示的；相反，抽象的分层流程由于前面的层层聚类，其最终区域（用户的视野）已经失去了同周围结构的关联性，重构这些关联需要考虑在布局上的邻居，其均属于其他上层抽象类簇且当前的布局只考虑了当前抽象类簇中的节点。综上，第二个缺点即当用户下放浏览到最底层时，该部分结构的邻域关联性是缺失的、且很难进行重构。

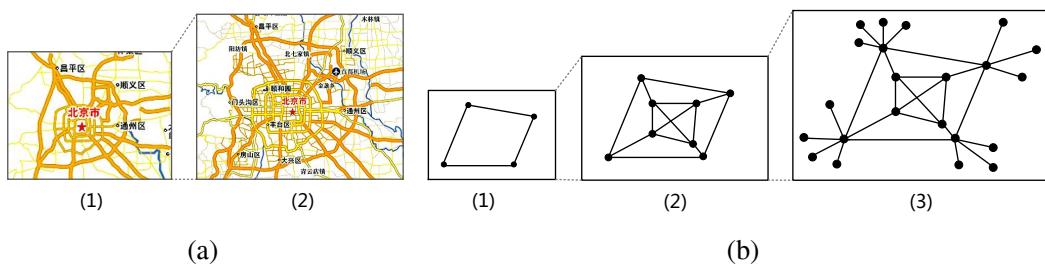


图 4.2 抽象分层示意图。(a) 示例结构, (b) 对应的用户访问顺序, 构建顺序相反。

Fig. 4.2 Diagrammatic sketch of SS algorithm.(a) Demo graph with diameter, (b) An ideal sample case.

而对于基于原始结构的构建流程来说，其初始层即为真实的原始结构生成，如图4.2b所示的用户的查看过程非常类似于路网的查看。采用这种方式，最终用户视野内的区域其实是整个结构的一部分，即用户可见的布局结构可以很容易

地向外扩展。也正是因为这一点，每次重绘整个结构会造成严重的资源浪费，但这一缺点很容易通过增量图绘制的方式来缓解；同时，也可以通过“视窗限定”来优化，即当用户查看某一区域时，超出视窗的部分不进行绘制。

由上一节可知，类似“路网探索”的分层方式能够更好的帮助用户完成浏览，而这一方式的关键就是“主干道”的选取，在第3章中，本文提出了以 K-core 分层抽样为主、以度分布筛选为辅的综合抽样算法，并已经通过实验验证了该方法在保留拓扑结构上的有效性。在本章，我们将该抽样方法应用到“主干道”的选取，以此提出基于 KSS 的分层布局算法，整合后的详细流程如图4.3所示。。

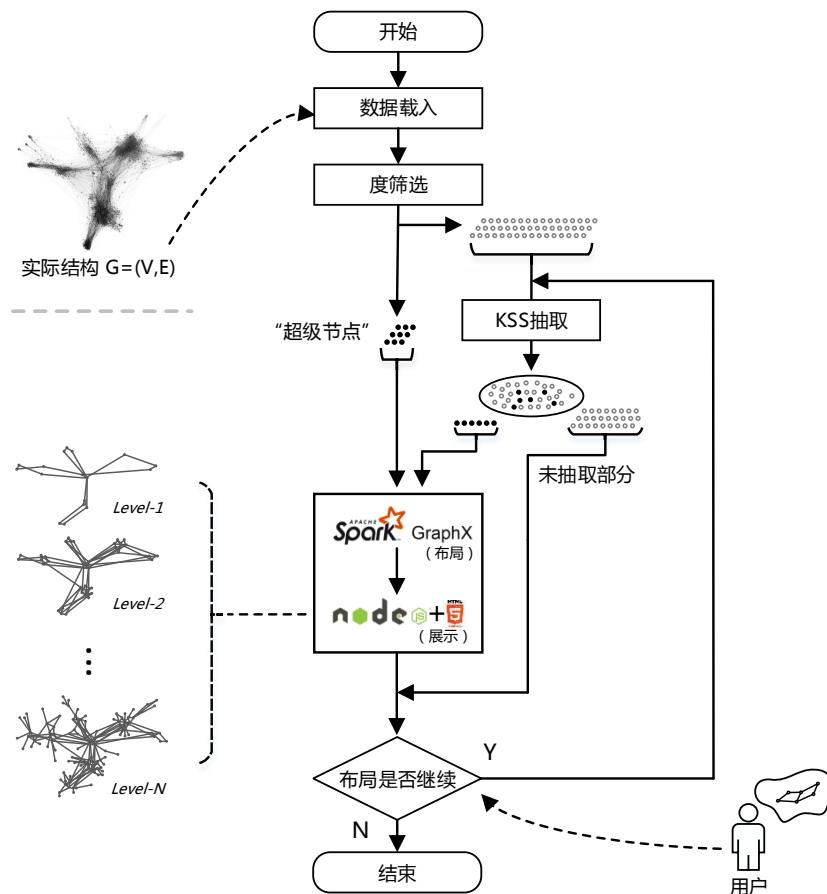


图 4.3 KSS 分层布局流程。
Fig. 4.3 Process of KSS Layout.

其基本思路是针对不同的分层结构 G_1, \dots, G_n 逐层进行布局显示，顶层结构 G_1 是最粗糙的视图，类似于图4.3中的左图部分；在第5章的原型系统中，会根据用户的操作来控制布局过程的进行，详情见图4.3。如图所示， G_1 由初次采样点和度筛选后补充的节点两部分组成，此抽样结构先后经由基于 Spark GraphX 的布局计算模块和基于 Nodejs 及 Html5 的显示模块并完成一轮执行，用户此时所见即为该层视图的布局结果，在不加干涉的情况下，该循环过程将一直运行到最

后一层 G_n , 即最细粒度的层级, 此时的结构最接近大图原始结构本身, 但由于采用了抽样策略, 在这里 G_n 仍然是原结构的近似模拟。在此过程中的层级数 n 在第3章的算法3.2中被确定, 或者由用户指定。

4.3 分布式图布局算法模型

关于图的定义, 沿用3.1.1节的描述, 在本文的图布局中, 节点 v_i 的坐标表示为 x_i , 两节点 v_i 和 v_j 之间的欧式距离可以用 $\|x_i - x_j\|$ 来表示, 拓扑距离用 $D_{sp}(i, j)$ 表示, 意为两节点在拓扑结构上的跳数。结合以上表示, 本文的图布局问题可以抽象为: 在给定的图 G 已知的情况下, 通过计算来确定顶点坐标 (集合) $X = \{x_i | i \in V\}$, $x_i \in R^2 or R^3$, 要求各节点在此位置下, 相互作用达到平衡, 进而使得整个图处于一个低能量状态。

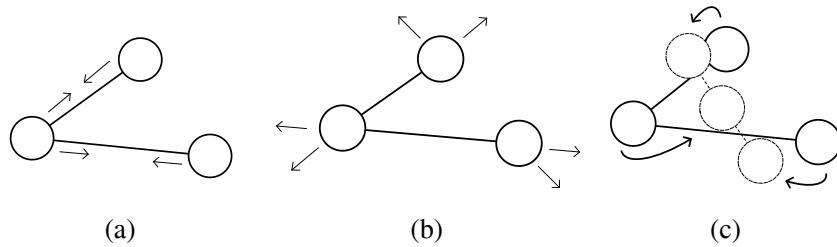


图 4.4 布局时节点的相互作用。(a) 吸引力, (b) 排斥力, (c) 最终移动趋势及可能的终止态。
Fig. 4.4 The interaction of nodes. (a) attraction, (b) repulsion, (c) final movement trend and possible termination position.

在布局的过程中, 必然要通过各种条件进行约束, 换言之, 即对布局过程建模, 例如将整个图抽象为弹簧-电荷模型 (SEM) 时, 其相互作用即符合弹簧的弹力公式和电荷的作用力公式, 这非常易于理解且产生的结果具有现实的物理意义, 给定一个结构在遵循此类模型的前提下, 其结构的变化可由图4.4解释, 首先相关联的节点会产生直接的吸引力, 这在物理意义中对应的是弹簧的弹力; 同时, 不同节点之间均会排斥对方, 这对应的是电荷之间相互的斥力 (即使没有边)。如此, 整个系统会由于受力不平衡而产生移动 (顶点移动), 直到整个结构内的斥力和引力达到平衡状态, 如图4.4c所示, 整个结构会趋向于移动到虚线状态, 此时, 该结构各节点的斥力和引力均在同一直线达到平衡状态。

4.4 基于 GraphX 的算法实现

4.4.1 关键编程接口

在具体介绍基于 GraphX 的实现之前, 需要首先介绍 $mapReduceTriplets$ 这一函数, 其核心功能是实现了邻边聚合, 它是 GraphX 中最核心和强大的函

数。GraphX 提供的 Pregel 函数也基于它而来，所以对它的优化，能很大程度上影响整个系统的性能。顾名思义，该函数是一个二阶段函数，包括 *map* 方法和 *reduce* 方法，且操作对象是 *EdgeTriplet*（具有边加两个端点的全部信息），*mapReduceTriplets* 的计算过程如下：

- *map*: 应用于每一个 *triplet* 上，生成一个或者多个消息，消息以 *triplet* 关联的两个顶点中的任意一个或两个为目标顶点
- *reduce*: 应用于每一个 *Vertex* 上，把发送给每一个顶点的消息合并起来
- *mapReduceTriplets* 最后返回的是一个 *VertexRDD*，它包含了每一个顶点聚合之后的消息，没有接收到消息的顶点不会包含在返回的 *VertexRDD* 中。

在本章，我们借助 GraphX 提供的 Pregel 编程接口来实现前面提出的布局算法，值得注意的是此处的 Pregel 编程接口并不严格遵循经典的 Pregel 模型，其本质是 GraphX、基于 GAS 模型抽象出的一个具有 Pregel 编程特点的接口以方便用户快速熟悉并使用。类似经典的 Pregel 模型，在 GraphX 实现的该接口中，同样具有三段函数体结构，包括顶点处理逻辑 (*vprog*)、发送消息逻辑 (*sendMsg*) 以及合并消息逻辑 (*mergeMsg*)。这种基于 *mapReduceTriplets* 方法的 Pregel 模型，和标准的 Pregel 的最大区别是，它的第 2 段参数体，接受的是 3 个函数参数，而不接受 *messageList*。它不会在单个顶点上进行消息遍历，而是会将顶点的多个 *ghost* 副本收到的消息聚合后，发送给 *master* 副本，再使用 *vprog* 函数来更新点值。消息的接收和发送，都是被自动并行化处理的，无需担心超级节点的问题。

Algorithm 4.4 Pregel in GraphX

```

1: vprog(vertexId, vertex, message):
2: begin
3:   //更新顶点
4: end
5: sendMsg(EdgeTriplet[···]):
6: begin
7:   //发送消息，该函数会不断迭代
8: end
9: mergeMsg(msg1,msg2):
10: begin
11:   //合并消息
12: end
```

常见的代码模板如算法4.4所示，可以看出其综合了 Pregel 模型和 GAS 模型两者的特点，即接口相对简单又保证性能，可以应对点分割的图存储模式，胜任符合幂律分布的自然图的大型计算。

4.4.2 布局算法的实现

首先明确算法整体的执行过程，即遵循图4.3的流程可由算法4.5表示。其部分内容均已在第3章被详细介绍，核心布局算法的完成由 Pregel 接口实现，其内部的三段逻辑可详见算法4.6。此外，算法4.5中的第 11 行的 Merge 即表示下一层布局时的顶点坐标是基于本层的，即在本层顶点的坐标锚定后进行下一层节点的坐标计算，这对于控制不同层次间布局形状的一致性非常重要。

Algorithm 4.5 KSS layout algorithm

```

1: 输入:  $graph = (V, E)$ 
2: 输出:  $coordinates$ 
3:  $g \leftarrow null$ 
4:  $iterator \leftarrow 1$ 
5:  $layerNums \leftarrow KSS\_Process()$ 
6: while  $iterator < layerNums$  do
7:   if  $contin == true$  then
8:     if  $iterator == 1$  then
9:        $g \leftarrow KSS\_Sampling(graph) + KSS\_DegreeFilter(graph)$  // 详见第三章
10:    else
11:       $g \leftarrow GraphMerge(g, KSS\_Sampling(graph))$  // 即 Join 操作
12:       $coordinates \leftarrow g.Pregel(vprog, sendMsg, mergeMsg)$  // 使用 GraphX
13:       $View\_Process(coordinates)$ 
14:       $contin \leftarrow Users\_Action()$ 
15:       $iterator \leftarrow iterator + 1$ 
16:    else
17:       $exit()$  // 结束算法
18: end while

```

对于具体的布局逻辑，本章采用了经典的 FR 算法作为基准，该模型的斥力及引力计算遵循公式2.2。该模型的布局效果已经得到广泛的证明，同时，作为“弹簧-电荷”模型的一种，其布局结果也具有宏观的可理解性。由上一小节的论述可知，借助 GraphX 的计算能力必须得通过其提供的编程接口，在此，本章使用 Pregel 接口对 FR 算法进行重写，整体代码详见4.6。

为了方便叙述，这里先介绍算法的核心部分，即 11-20 行的 $sendMsg()$ ，该函数是主要布局计算逻辑的实现部分，在计算完斥力、引力及其合力后，在 18 行该方法通过 $Iterator$ 迭代的向每个顶点的关联顶点发送最终的合力位移。如果某节点收到了来自不同节点的不同消息，那么 3-9 行的 $mergeMsg()$ 函数会对这些消息进行合并，并返回合并后的最终偏移（在模型中即为合力的作用）。对于 22-26 行的节点处理函数来说，其完成了最终的更新操作，此处的更新操作需要满足特定的条件，第 6 行表示只有最终位移 $disp$ 大于允许的最小移动距离 MIN 时，才对该节点的布局坐标（结果）进行更新。

以上三部分在迭代过程中并不存在明确的先后顺序，各部分各司其职异步执行，通过底层 *mapReduceTriplets* 方法的判别机制，在没有（要发送消息的）活动节点后整个迭代终止（需指定最大迭代次数），即整个布局迭代结束，此时各节点属性的坐标即为最终的布局结果。

Algorithm 4.6 vrog, sendMsg & mergeMsg

```

1:  $k \rightarrow \sqrt{width * width / (vertexNums + 1)}$ 
2:
3:  $vprog(vertexId, attr[...], finalDist) :$ 
4: begin
5:    $disp \leftarrow finalDist$ 
6:   if  $disp > MIN$  then
7:      $newLocate \leftarrow attr.locate + disp$  //坐标运算
8:     updateItems(attr, newLocate)
9: end
10:
11:  $sendMsg(source \rightarrow target) :$ 
12: begin
13:   Iterator{
14:      $distance \leftarrow computeDist(Locate_{source}, Locate_{target})$ 
15:      $attractDist \leftarrow pow(distance, 2) / k$ 
16:      $repulsiveDist \leftarrow pow(k, 2) / distance$ 
17:      $forceDist \leftarrow attractDist + repulsiveDist$  // 坐标运算
18:      $send(Id, forceDist)$ 
19:   }
20: end
21:
22:  $mergeMsg(forceDist1, forceDist2) :$ 
23: begin
24:    $finalDist \leftarrow forceDist1 + forceDist2$  // 坐标运算
25:   return  $finalDist$ 
26: end
  
```

4.4.3 作业优化

在实际编写上述代码时，还需要注意图的缓存问题。`cache()` 和 `persist()` 方法修改当前 RDD 的存储方案 `StorageLevel`，在代码默认的执行状态下是 `MEMORY-ONLY` 级别，保存到内存。这两个方法都不会触发任务，只是修改了 RDD 的存储方案，当 RDD 被执行的时候按照方案存储到相应位置。而 `checkpoint` 方法则会单独执行一个 job，并把数据写入磁盘。`checkpoint` 检查 RDD 是否被物化或计算，一般在程序运行比较长或者计算量大的情况下，需要进行 `checkpoint`。这样可以避免在运行中出现异常导致 RDD 回溯代价过大的问题。此外，`checkpoint` 会把数据写在本地磁盘上，且 `checkpoint` 之后的数据可以被同一 session 的多个 job 共用。

4.5 实验结果与分析

4.5.1 实验设置

本节实验中的实验结果均使用第5章的原型系统得到，故具体实验配置同表5.3一致。关于 KSS 分层布局算法的效果的衡量，区别于第3章实验从拓扑结构相似性进行对比，这里主要针对布局结果（即显示结果）进行比较，在这方面基于美学角度的评价尤为重要。由于目前没有绝对综合的美学评价指标，且诸如边交叉数等指标计算时时间复杂度过高，因此此处着重比较 KSS 分层布局算法自身的分层效果，以及其同基准布局效果的对比。实验数据如表4.1所示，数据均源自 SNAP 公开数据集。

表 4.1 实验所使用的数据集。
Table 4.1 Data sets of experiments.

数据集	类型	边数	点数	平均聚集系数	三角形数	直径
ego-facebook	Undirected	88234	4039	0.6055	1612010	8
email-Enron	Undirected	183831	36692	0.4970	727044	11
gemsec-Deezer	Undirected	498202	54573	0.4123	1533267	9
com-youtube	Undirected	2987624	1134890	0.0808	3056386	20

实验设计与实施

本章实验采用了表4.1所示的数据集，由于上文中提出的布局算法还只适用于无向图，所以这里均采用无向图的数据集进行布局实验，顶点规模从 4k 到 100w，边集规模从 8w 到 300w，这几种数据集均为有社区特征的数据集，可以更为方便的验证布局的效果。实验程序的执行均在表5.3所示的环境下完成，布局展现均使用第5章的 GVIS 原型系统显示。

对于 ego-facebook 数据集来说，其规模较小以至于可以在单机运行，使用 Gephi 可以完成对其的布局及展示，因此，ego-facebook 被用作来验证本章的分层布局算法的布局准确性；对于 email-Enron 数据集来说，Gephi 中的大部分（基于力导向的）布局算法已经无法支持对该算法的布局计算，对该数据集的布局可以说明本章布局算法的优势；对于 com-youtube，其点集和边集规模已经达到百万，基于 GraphX 的传统力导向算法也基本不能完成布局任务，通过对该数据集的布局来进一步说明本章提出的算法的优势。

对于布局分层的选择，这里采用人工给定的方式完成，对于规模较小可以完成全部布局的数据集（例如 ego-facebook），这里直接指定具体的分层数次（本次实验是 3 或 4），首层的分层参数选择同样由人工给出；对于规模很大的数据集

(例如 com-youtube)，则不对布局层数进行限制，只在初始布局的参数基础上，层层向下布局，这里最终只选择了有限的层数进行显示。

4.5.2 布局结果与结论

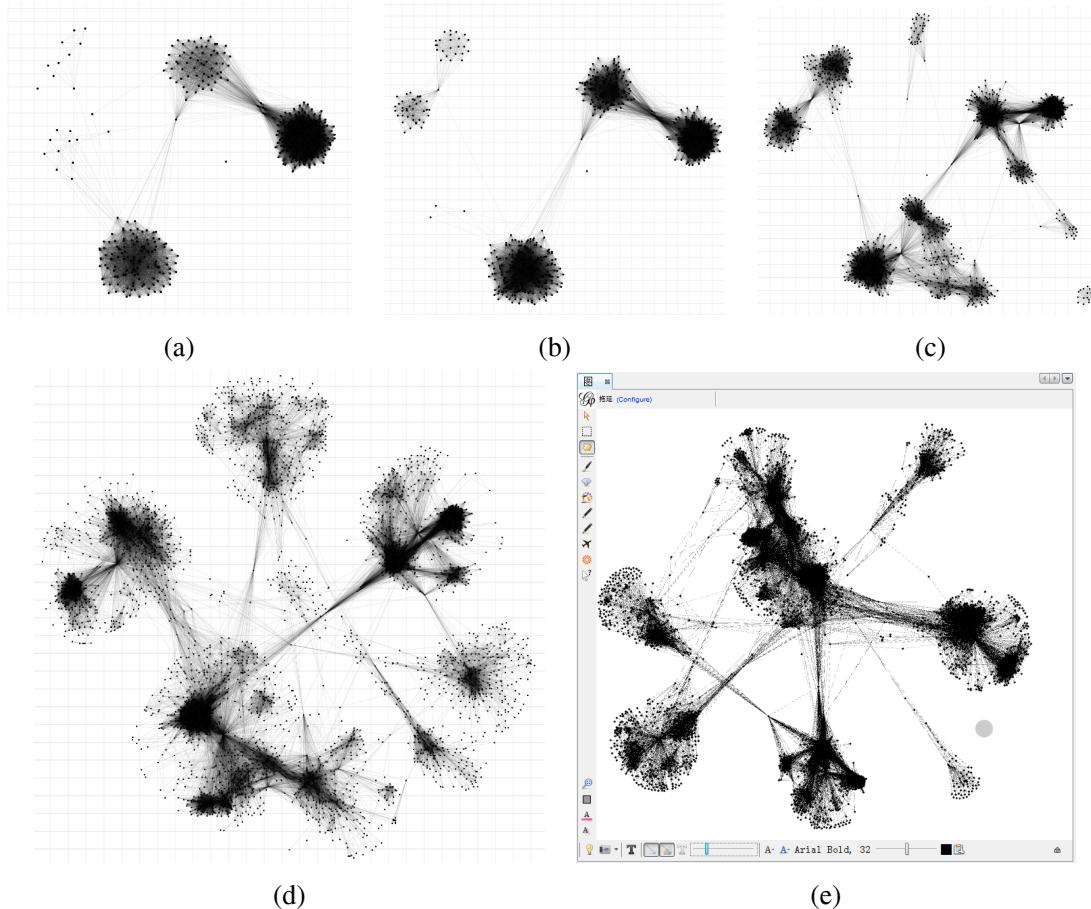


图 4.5 ego-facebook 的分层布局效果。(a) 第一层 (21%)，(b) 第二层 (38%)，(c) 第三层 (65%)，(d) 第四层 (100%)，(e) Gephi 中 FR 算法对该数据集的布局结果。

Fig. 4.5 Results of KSS Layout on ego-facebook data set.(a) 1st layer (21%), (b) 2nd layer (38%), (c) 3rd layer (65%), (d) 4th layer (100%), (e) Result of FR Layout with Gephi.

图4.5是数据集 ego-facebook 的分层布局效果，图中的 (a), (b), (c), (d) 分别为四个不同的层级；可以看出层级之间是增量布局的，即第二层布局是在第一层的基础上进行的布局；并且层与层之间并不使用抽象的节点，每层视图显示的节点都是原数据集中的节点（第4.2小节已对该种方式进行讨论），由图中结果可以看出，增量布局被很好的体现在了不同层级的视图上。同时可以发现，ego-facebook 的各分层的结构在不断细化的基础上整体基本保持不变，这体现了 KSS 布局初始布局结构的准确性；整个过程在视觉上呈现出节点不断增多、细节不断增强、同时部分结构出现“分裂”等现象，这符合 KSS 布局算法在设计时的构想。

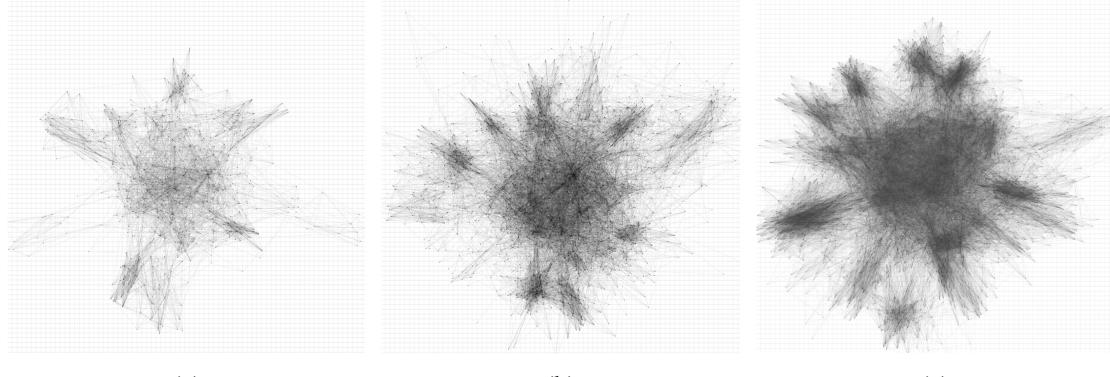


图 4.6 gemsec-Deezer 的分层布局效果。(a) 第一层 (1%), (b) 第二层 (4%), (c) 第三层 (12%)。

Fig. 4.6 Results of KSS Layout on gemsec-Deezer data set.(a) 1st layer (1%), (b) 2nd layer (4%), (c) 3rd layer (12%).

由图4.5e所示的使用 Gephi 完成的对该数据集的布局的结果可知，图4.5d所示的最终结果基本与其一致，说明该算法在布局计算阶段大致符合预期效果。

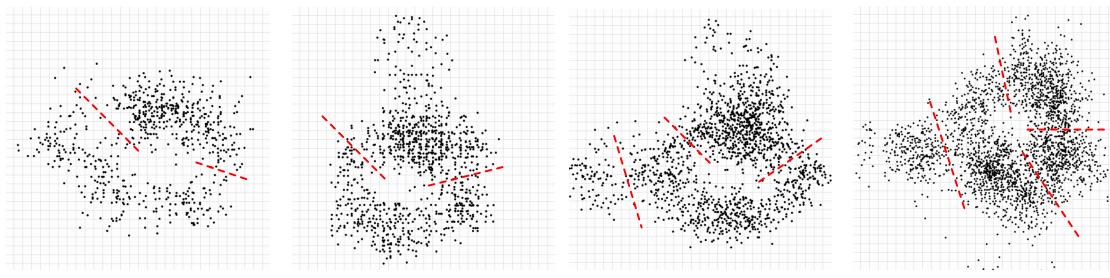


图 4.7 email-Enron 的分层布局效果。(a) 第一层 (26%), (b) 第二层 (41%), (c) 第三层 (57%), (d) 第四层 (78%)。

Fig. 4.7 Results of KSS Layout on email-Enron data set.(a) 1st layer (26%), (b) 2nd layer (41%), (c) 3rd layer (57%), (d) 4th layer (78%).

图4.6、4.7、4.8均为补充实验。分别是对 email-Enron、gemsec-Deezer 以及 com-youtube 数据集的布局结果。由于节点较多且较为密集时，边的加入会增加视觉杂乱，因此对于后两组实验结果仅对节点进行显示，由于此三组数据均为具有社区特征的数据集，因此通过观察节点的聚集程度可以表明布局结果的有效性。此外，该三组布局的结果均只显示了从初始布局开始到前几层布局的结果。

上述几组实验的布局结果表明，该布局算法对小规模数据集的分层效果更加明显，对于规模较大的数据集同样能够有效的完成布局，结果也显示出了一定程度的聚集特征，并且图4.8表明该算法可以应用到百万点边规模的超大规模数据集，并且结果具有十分良好的聚集效果。总体来看，KSS 布局算法能够给出小规模数据集清晰的布局显示，同时对于大规模数据集也有一定的布局展现能力。

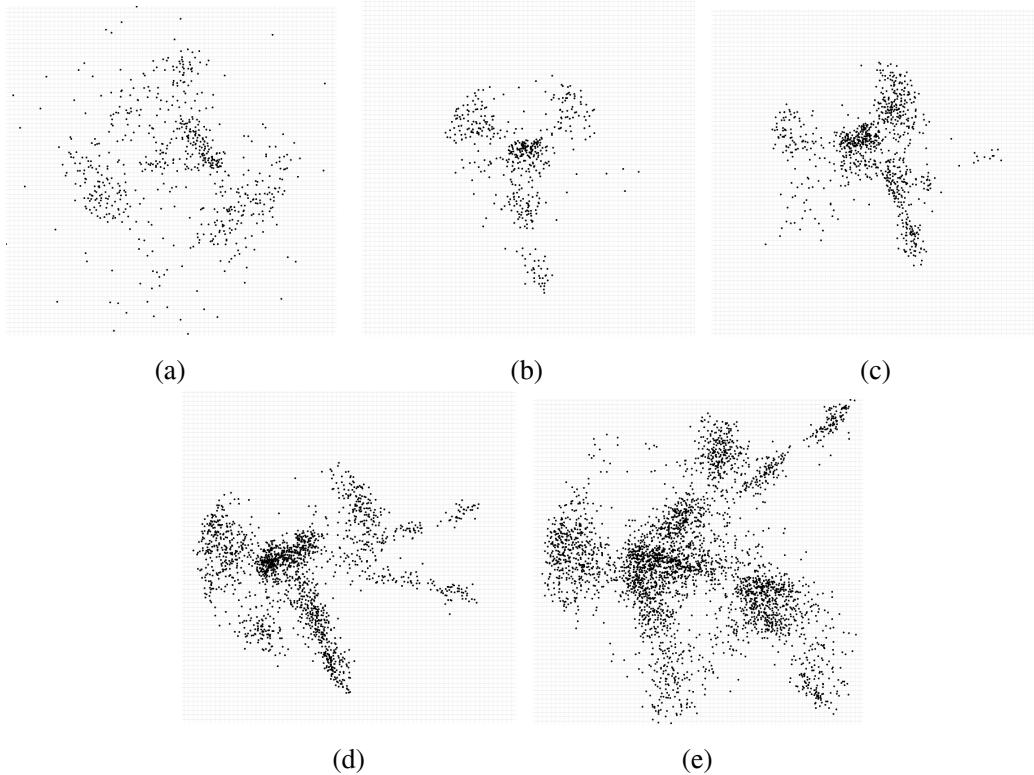


图 4.8 com-youtube 的分层布局效果。(a) 第一层 (0.5%)，(b) 第二层 (1.1%)，(c) 第三层 (2%)，(d) 第四层 (4.1%)，(e) 第五层 (7%)。

Fig. 4.8 Results of KSS Layout on com-youtube data set.(a) 1st layer (0.5%), (b) 2nd layer (1.1%), (c) 3rd layer (2%), (d) 4th layer (4.1%), (e) 5th layer (7%).

4.6 本章小结

本章在 KSS 抽样算法的基础上，提出了基于 KSS 的分层布局算法。最顶层（第一层）的结构由关键节点 G_{key} 构成；之后每一层的结构由上一层结构为起始布局状态递归的进行抽取及布局；整个布局过程可以在符合用户要求时被用户中断，从而最大可能的满足不同用户的探索需求。此外，采用本章提出的分层方案可以很好的解决对于大图无法有效呈现的问题，并且通过多组实验的布局结果初步验证了该方法布局上的准确性和有效性。

第 5 章 大图可视化原型系统

为了确保用户对大规模图数据的有效理解和探索，本文设计了 GVIS 原型系统，系统核心计算代码采用第三章的算法实现，并应用了第四章提到的全部优化策略。本章将介绍 GVIS 系统的技术选型、主要架构、功能模块，并给出了系统的部署和使用方法。

5.1 技术选型及依据

5.1.1 系统结构

本章介绍的大图可视化系统其核心就是对图进行布局计算然后展示，用户对该系统的全部操作均通过展示层，因此其可靠性、可移植性以及性能对用户体验十分重要。考虑到计算能力完全来自 Spark 集群，所以，此处的展示层应该是一种客户端应用，服务器端的核心即完成图布局任务的相关程序，客户端应用通过网络协议（HTTP 等）与服务端进行通信和数据交换。通常，这类应用分为 B/S 和 C/S 结构。

对于 Client-Server 结构来说，整个系统由 Client 和 Server 两部分组成，两者通过 http 协议进行网络通信，对于用户而言通常是使用专用的 Client 软件，相关的开发语言或平台有 Java/JVM 和 C#/.NET，通过此种方式开发的客户端有着操作系统原生应用的操作体验，但其缺点是可移植性较低，对于使用 C#/.NET 开发的应用很难在 Windows 平以外的操作系统上完美运行。目前的 Qt 似乎可以解决跨平台的问题，但 Qt 本质是 C++ 的类库，跨平台仍需要 C++ 等各种环境的支持。对于 Browser-Server 结构来说，其不再需要向 C/S 模式一样开发定制的客户端，而是借助浏览器来提供 GUI 界面，开发者只需设计和实现功能代码，不必花时间在（C/S 结构的）复杂的 GUI 绘制上，目前，B/S 结构的系统的 Browser 端一般采用 HTML(5)+CSS+JavaScript 来实现，HTML(5) 提供丰富的 GUI 控件支持，CSS 则是定义这些控件的外观样式，而 JavaScript 则完成控件绑定事件的任务，即赋予整个 GUI 界面以动态性。在通信方面，B/S 结构提供较 C/S 更为透明的代码实现，仍然是采用 HTTP 协议进行通信。遵循该种结构设计的系统最大的优点就是有良好的跨平台特性。

虽然使用 C/S 结构使得客户端能够获得大部分用户主机的性能，从而在图绘制等功能上获得更快的渲染速度，但针对于本文的 2D 图绘制需求，各绘制技

术的效率并没有太大差距，使用用户本机的 GPU 加速渲染确实可以极大提高显示速度，但此类方法并不通用且已超出本文的讨论范围。而 B/S 结构以其良好的跨平台性和高效的开发效率更适合本文原型系统的实现，且对于需要高性能要求的渲染操作浏览器也能提供比较可靠的支持。最后在系统维护上，C/S 模式只能通过更新远程用户的 Client 代码来实现相关硬编码升级，而 B/S 模式的运行方式（均可运行在服务器）决定了其维护与用户分离，只需刷新浏览器即可获得最新的系统功能。因此，结合上述两者的优点和缺点，本文的 GVVIS 系统最终选用了 B/S 结构进行实现。

5.1.2 图绘制技术

这里所说的图绘制技术/引擎通常是指将图形绘制在显示器屏幕的基于特定语言的方法库（或类库），本文原型系统的主要的绘图需求是 2D 点线绘制，基于这一基础，本文不考虑一系列 3D 绘制引擎的使用，诸如 Direct3D、OpenGL、WebGL 等 3D 绘制引擎虽然也可以直接绘制 2D 图形，但其自身的特性基本均是面向 3D 场景进行优化，所以对于 2D 场景的绘制，还是应考虑 2D 图绘制技术/引擎。表 5.1 列举了主要的 2D 图绘制技术/引擎，Windows 平台下的 2D 绘制引擎从 GDI 到 Direct2D 是一脉相承的，发展到目前已经有非常不错的性能，但在考虑跨平台能力选择 B/S 结构之后，可选择的就只有 Java2D、Canvas、以及 SVG，通常 Java2D 和 Swing 绑定在一起，前者专门作为后者图形绘制的工具类库，所以要使用 Java2D 就要依附相应的 Client（软件）之上，这对于本文先前的选择相违背，故不考虑对它的使用。最后，对于 Canvas 和 SVG，它们均不是传统的图形绘制引擎，实际上只是图形的容器，需要依靠诸如 JavaScript 脚本语言进行图形的填充，这一切操作均依附在 HTML 之上，绘制动作完成在浏览器内，所以，在表 5.1 中的 Canvas 是支持硬件加速的，只是这个特性是由浏览器间接提供的。

表 5.1 二维图绘制技术比较。
Table 5.1 Comparison of 2D drawing techniques.

技术/引擎	平台支持	处理结果	绘图语言	评价	加速
GDI	Windows	位图	C++	技术老旧，封装性不足	无
GDI+	Windows	位图	C++	封装 GDI，交互性不足	无
Direct2D	Windows	位图、矢量图	C++	替代 GDI+，用于游戏	硬件加速
Java2D	跨平台	位图	Java	依附于 Swing，灵活性不足	硬件加速
Canvas/2D	跨平台	位图、矢量图	脚本（JavaScript）	适于高数据量高绘制频率	硬件（浏览器）加速
SVG(dom)	跨平台	矢量图	脚本（JavaScript）	适于低数据量低绘制频率	无

对于 Canvas 和 SVG（只是一种机遇 XML 的存储格式）的选择，目前基本的结论是对于高频率的显示高数量的场景，Canvas 要比 SVG 的性能好的多，这是因为 SVG 是基于 DOM 的渲染机制，DOM 即文档对象模型，可以理解为 SVG 将

每个图形元素均以 dom 对象（例如 html 的一个标签）的方式进行渲染，这样可以轻松完成事件绑定并带来高自由度的交互能力，同时其绘制的结果也是放缩无失真的矢量图，但问题是在高数量（比如网络绘制）图形的绘制场景下，SVG 会直接将图形加载到 dom 上，而最终网页的加载和图形的渲染要首先加载 dom 结构，这使得 SVG 在高数据量的绘制场景下会非常缓慢，而 Canvas 并不基于 dom 渲染（结果一般是位图），更类似于 Java2D 等的画布的概念，渲染速度在高数据量的场景下比 SVG 快的多，且在正确的刷新重绘流程下可以得到很高的高帧率绘制性能（用作动画效果）。因此，结合上述总结，GVIS 系统最终基于 Canvas 使用 HTML(5)+CSS+JavaScript 完成整个图可视化展示功能。

5.1.3 数据传输及后端框架

作为计算资源与用户分离的系统，数据的传输直接关系用户系统的正常工作，且通常这部分对用户并不可见，因此数据传输的稳定性就非常重要；此外，对于本文的图可视化系统来说，根据第4章图4.3的布局流程可知，每次需要显示的数据就是含有坐标信息的文本数据，这部分数据需要在用户发出查看请求后传送的用户的浏览器端，然后前端系统的图绘制部分将图形渲染到用户的屏幕，对于 GVIS 系统来说，布局的目标数据一般很大，产生的坐标信息的文本也非常巨大，这对数据的稳定传输提出了更高的要求；基于 Web 的前后端通信方式从最初的 Request/Response 请求，发展到现今的 WebSocket/Socket.io，交互方式已经有很大的变化，目前几种主流的支持长连接的通讯方式如表5.2所示。在 WebSocket 出现前，长轮询方式只能通过短连接（例如 ajax 请求）的轮询来完成，这对于一些监控场景很不友好，GVIS 系统需要实时获取任务的执行进度，那么长连接的需求就必不可少；另外对于可视化数据的传输部分，使用长连接可以更轻松的处理动画一类的场景，通过在本系统中使用长连接逐帧显示请求自 Server 端的数据分片，在数据量较少的情况下可以达到肉眼不可分辨的连贯绘制效果。总的来说，WebSocket 可以使 Server 端和前端系统结合的更为紧密，使两者始终保持通信状态，这对于数据传输要求较高的本系统来说非常重要，因此，在实现上最终采用了 WebSocket 进行前后端通信。

表 5.2 可支持长连接的通信方式。

Table 5.2 Communications that support long connections.

通信方式	支持方式	通信方向	长连接时的请求	底层
Ajax	短连接	单向	轮询 + 每次 Http Request	XMLHttpRequest
WebSocket	长连接	双向	单次 Http Request	WebSocket API
Server-sent event	长连接	单项	单次 Http Request	基于 WebSocket

运行在 Spark 集群的计算任务简洁高效地被前端系统所使用，必须借助某些后端框架实现。对于本文的原型系统而言，是典型的前端主导后端模式，对于像 Struts2、SpringMVC 等传统的前后端分离的重量级框架，在此并不适用。通常，对于“请求-反馈”的动作来说，执行顺序是前端向后端发出请求（点击等），后端框架拦截该请求，并将其转发至指定的处理代码接收，处理代码完成逻辑处理后将结果返回给前端。对于这一过程，WebSocket 的使用使得前后端通信不再需要拦截转发，二者可以直接交互，这使得本文的系统可以精简前后分离的传统框架。弃用传统的拦截转发转而采用路由（Router）来统一整理和引导不同的 WebSocket 连接，目前，Express（基于 Node.js）是应用最多的、遵循上述思路的开发框架，简单高效并继承 Node.js 全部的功能特性。由此，本原型系统的后端采用 Express 框架进行开发，同时，为了配合 WebSocket 的应用，系统后端的逻辑代码均采用 Node.js 实现并以服务的方式发布。

5.1.4 总体流程

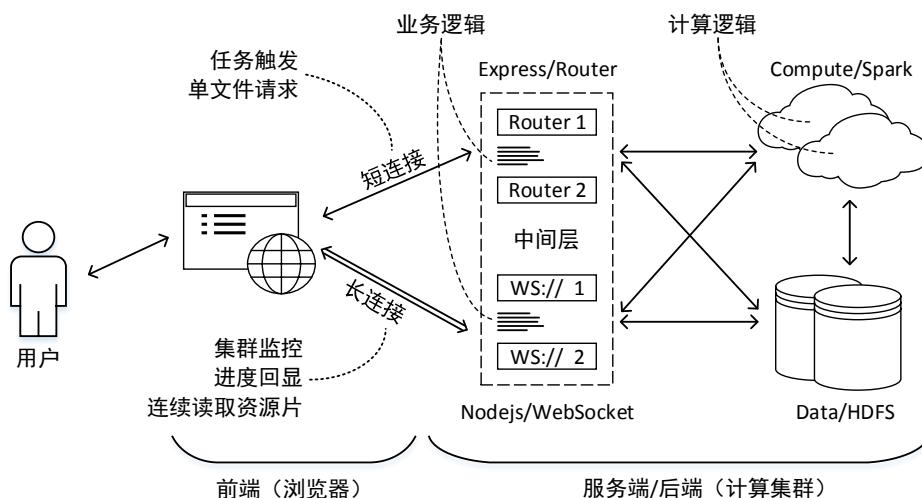


图 5.1 GVIS 系统架构图。
Fig. 5.1 System architecture of GVIS.

至此，本章讨论了原型系统所使用的相关技术或框架的优点及使用原因，最终系统的整体流程应如图5.1所示，系统的核心部分运行在远端集群上，通过浏览器来对用户提供指定的服务，对于 GVIS 系统来说，用户典型的行为就是对自己感兴趣的数据进行可视化探索，在流程5.1中涉及包括任务触发在内的全部操作，用户在前端的点击动作会提交不同的资源请求，这些请求均在由 Node 和 Express 组成的中间层所接收并处理，在处理结束后仍由中间层将结果返回到页面，即用户看到先前操作的反馈结果。整个流程没有使用复杂的 Java 开发框架，而是使用中间层来做联结，实现前后端的数据通信，同时也实现了计算部分和整

个系统的解耦，可以使两者在运行上逻辑更加清晰，资源调度和任务执行更加独立。

5.2 系统设计

5.2.1 基本任务

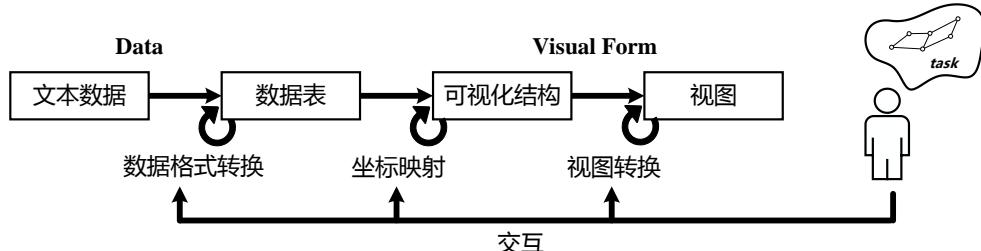


图 5.2 信息可视化参考模型^[55]。
Fig. 5.2 Information visualization model.

根据 Card 等人^[55]提出的信息可视化模型，整个可视化流程如图5.2所示，遵循这一经典模型理论，可视化系统的设计需要完成以下几个任务：

- 初始数据的格式转换：在系统实现中，需要考虑对非标准输入数据的支持，这要求系统具有能够将不同数据格式转化为标准格式的能力，在此过程中还应当完成数据的清洗工作。另外，对于有框架依赖的系统，还需要在此环节将数据载入到相关底层文件系统中。
- 数据信息的可视化映射：这一过程是可视化流程中的关键步骤，其确定了原数据抽象信息的视觉呈现，对于图数据的可视化来说，在此步骤之前数据只是建立在内存中的图结构，经过布局算法的迭代后才得到每个节点的坐标信息，这时即完成了抽象结构到可视结构的映射。
- 数据视图的变换：随着数据规模的增大和所处理数据复杂程度的增大，直接呈现可视化映射后的结果往往不能清晰的展现原数据的数据特征。例如，对于小世界网络的布局结果，如果直接展现，将产生“毛球”问题，从而使可视结果不可理解。解决这一问题的方法，一是在可视化映射（坐标计算）步骤进行处理，使布局结果直接具有可读性；二是在视图呈现上允许用户进行不同维度的视图变换和探索，从而建立对源数据结构的理解。

5.2.2 模块分析

由以上分析可得，整个原型系统须包括输入数据、布局计算、布局展示、布局调参、布局交互、作业监控、异常判别、结果输出等功能，如图5.3所示。其均

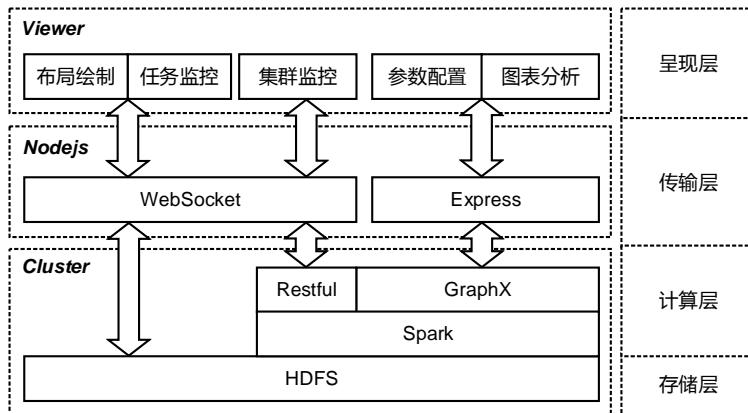


图 5.3 系统模块组成结构。

Fig. 5.3 Composition structure of system modules.

需要在呈现层也就是用户可见的界面上进行实现，结合本章5.1.4节所使用的技术及框架，下面结合系统的最终实现对各部分进行详细介绍。

- 输入数据。允许用户上传指定格式的数据集进行计算，用户需使用上传文件功能将文件上传至 HDFS（Nodejs 实现）；同时，系统提供示例图数据文件，用户可在指定位置选择需要计算的图数据文件进行计算。

- 布局计算。用户选取输入文件后，基于 GraphX 的核心计算代码即开始读取文件构建 Spark 作业，完成后将提交 Spark 集群运行，计算结果以会议文件的形式分别存储在集群的不同节点上，因此，系统还完成了将多分片结果重新拼凑完整的工作。拼凑完成后将结果以 WebSocket 的方式传回前端系统进行展示。

- 布局展示。获得计算结果数据后（JSON 格式），前端会解析 JSON 数据并利用 Canvas 开始绘制布局结果，由于采用了 Websocket 进行数据传输，因此本系统支持以类似流的形式连续对接收到的数据进行绘制，而不用等待数据全部接收完毕。初始的布局展示即以默认参数执行的结果（如图5.4所示）。另外，为了减轻 Canvas 的渲染负载，在显示模块只会绘制窗口内的部分，如果某个点的坐标不在当前窗口内，那么该节点不会被绘制，这会大大减轻浏览器的渲染压力，因为通常（在浏览结构内部时）浏览窗口只占整个布局的一小部分。

- 布局调参。对于布局结果的控制，用户可以通过布局参数的调整来实现，调整迭代次数、抽样比率之后前端系统会重新发出计算请求，Spark 计算代码接收到请求后会重新计算并将结果返回到前端。

- 布局交互。在布局结果展示完毕后，GVIS 系统还允许用户对布局结果进行交互式查看，如图5.5右侧所示，主要包括：结构的放缩；绘制的背景、网格、节点及边线的粗细和颜色。用户可以以极高的自由度来处理绘制结果。

- 作业监控。用户提交计算请求后，在系统的监控模块用户可以实时获取计

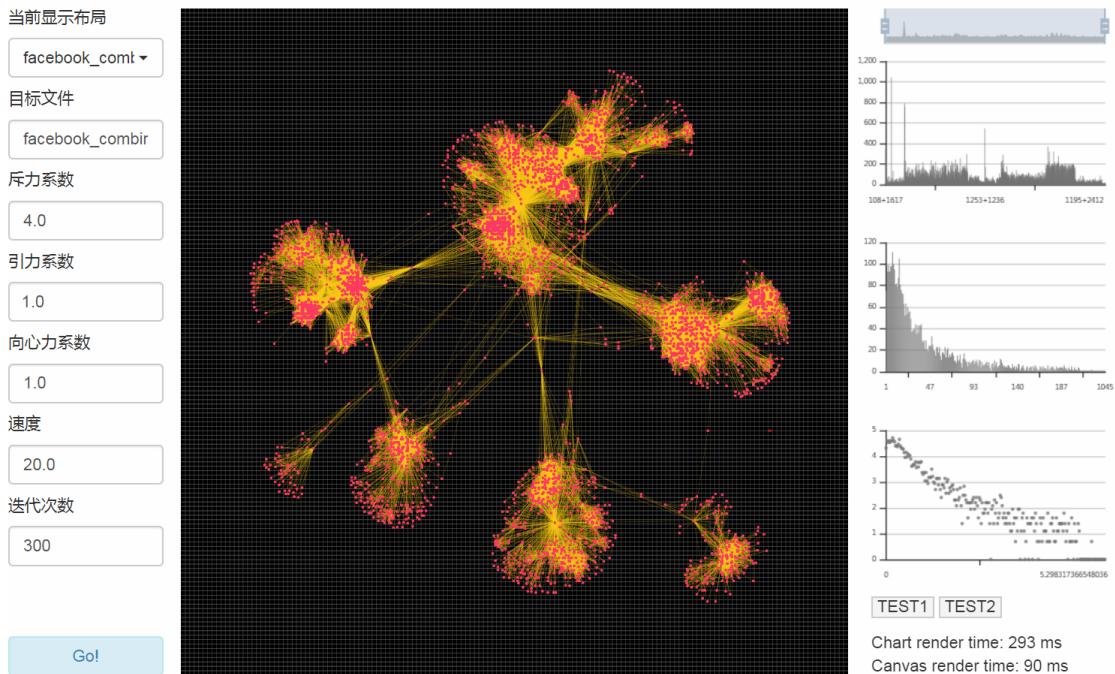


图 5.4 布局模块系统效果。分别是参数面板(左), 布局及统计显示(右)
Fig. 5.4 View of GVIS's layout module. Parameters panel (left) and layout view with statistics(right)

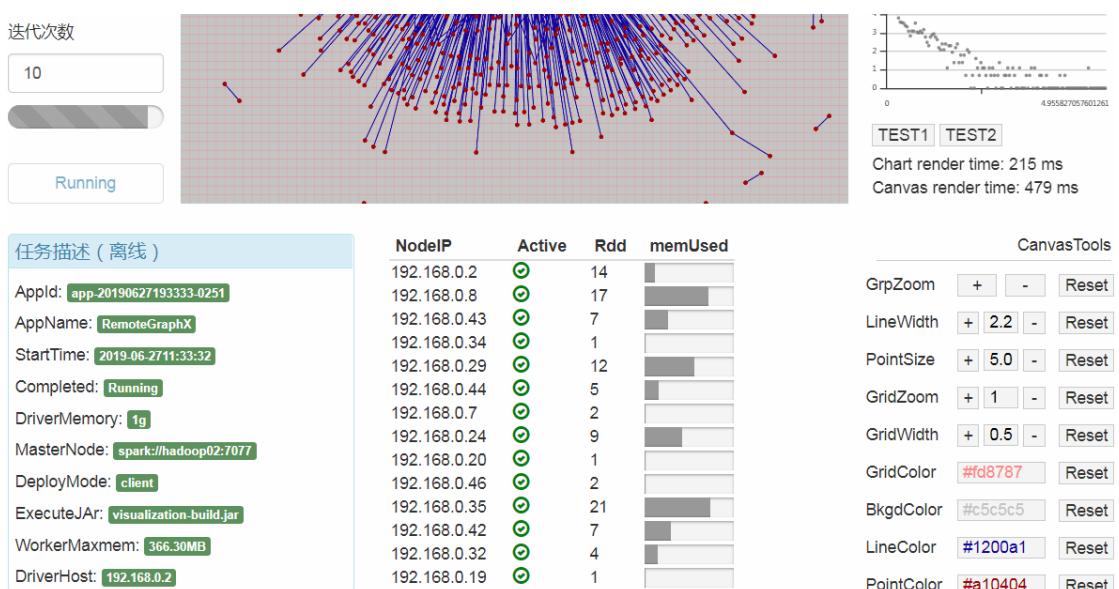


图 5.5 任务监控及布局交互。任务监控(左), 调色变换工具(右)。
Fig. 5.5 Task monitoring (left) and transform toolbox (right).

算任务的处理进度，如图5.5左侧所示，包括各节点 Spark 任务的进度和任务执行总进度。计算节点的任务进度由 Spark Restful API 获取。通过作业的监控用户可以更高效的使用本系统进行任务提交。

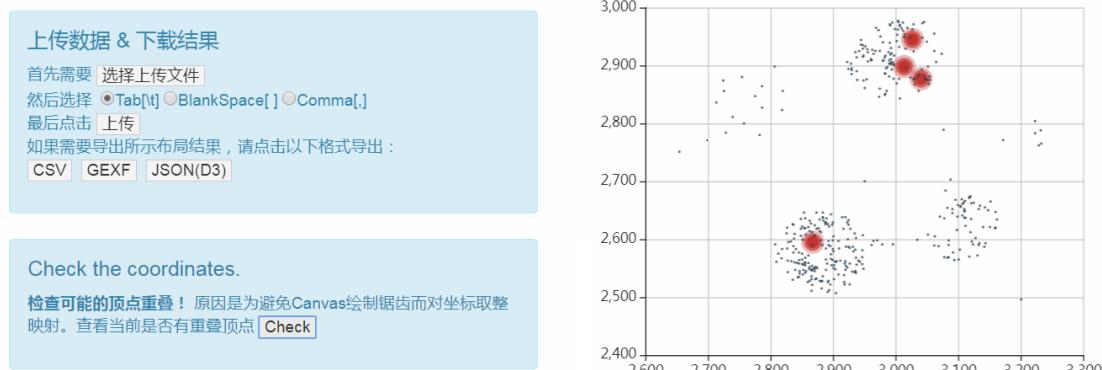


图 5.6 异常判别（右）及数据的导入导出（左）。

Fig. 5.6 Functions of Abnormal discrimination (right), and import / export of results (left).

- 异常判别。这是针对可能出现的错误而设立的检测部分。例如，对于使用 Canvas 绘制的结构，在缩放比例过小时会导致像素饱和，即多个节点共用一个像素，这会导致错误的显示从而误导用户，考虑到实时检测像素重叠的代价，在本系统中以单独提供的检测功能来检查这种情况的发生。当用户发现有趣的可视化信息时，使用此功能来确保其所看见的结构是正确显示的。如图5.6右侧所示，红色的点表示该像素（坐标）上放置了多余一个的顶点，也就是出现了顶点重叠现象，如果该现象十分严重，那么当前显示的布局结果就不准确。

- 结果输出。最终，对于用户希望保留某次布局结果的需求，本系统支持将结果保存为图片的形式。此外，对于计算结果的文件，系统也允许用户将其下载到本地以便在其他可视化工具上使用。（如图5.6左上所示）

上述各部分功能均为显示层（呈现层）所有，在系统架构层面，GVIS 系统还有传输层、计算层和存储层，结合已经论述的内容，可以抽象出如图5.3所示的系统架构图，这可以看做是图5.1的抽象概括。至此，对 GVIS 系统的介绍已基本完成，下面，将围绕系统的部署及实际使用情况来对其进行进一步介绍。

5.3 部署及使用

由于采用了 B/S 模式，即整个系统只需通过浏览器对外提供系统界面，因此全部系统代码均部署在服务器端（计算集群对外通信的主节点）用户本机无需软件和环境的安装和配置，系统在代码上整体分为三部分，包括前端系统界面、中间层路由和逻辑处理、Spark 集群和计算模块。各部分使用的软硬件版本如表5.3所示，下面由后端至前端依次介绍系统的部署过程。

表 5.3 系统相关配置信息。
Table 5.3 System configuration information.

配置项	参数值	备注
Operating System	RHEL Server release 6.1	节点操作系统
Java Version	1.8.0_65 (Oracle Corporation)	Java 版本
Hadoop/HDFS Version	2.7.3	Hadoop 版本
Hadoop Master	hadoop02:50070	Hadoop 主控端口
Configured Capacity	3.29 TB	HDFS 配置容量
Hadoop Live Nodes	14	在线节点数
Spark/GraphX Version	2.2.0	Spark 版本
Hadoop Master	hadoop02:8888	Spark 主控端口
Scala Version	2.11.8	Scala 版本
Spark Cores in use	14	在线节点数
Spark Memory in use	14GB	整体内存大小
Node Version	10.9.0	Node 版本

GVIS 系统所使用的 Spark 集群搭建在 14 台小型机之上，整个集群内部采用局域网 IP 进行通信，02 号节点作为主控节点同时具有对外 IP，负责整个集群的内外信息交换；存储方面，整个集群挂载的硬盘容量总计 3.29TB，剩余空间 59.85%，均可用作原型系统的输入、输出及持久化；在内存方面，集群中每台小型机均有 8GB 内存，但实际空闲内存 3GB 左右（多方使用），因此在部署 Spark 集群时，只为每个节点分配了 1 个 Worker 和 1GB 内存，即原型系统可用的集群内存 14GB，在多数场景（不加载超大数据集）下能够应对实验及系统演示。

对于核心的计算部分，采用提交 Spark 作业的方式运行，提交时的参数由前端系统用户配置，提交运行完毕后，结果会被 Nodejs 实现的中间层所抓取并返回前端系统。中间层的具体任务是路由转发及 WebSocket 对接，对应不同的通信场景，整体均由 Nodejs 实现，运行在 02 号主控节点，整体以服务监听的方式运行，响应来自前端的请求。对于前端呈现系统，采用的服务器容器是基于 Node 的 Http-Server，整个前端系统代码均部署在此容器内，同样运行在 02 号节点，通过对外 IP 向外部用户提供服务。

5.4 本章小结

GIVS 系统是本文针对大图可视化处理而开发的原型系统，用于实现大图数据的布局和交互式展现。本章首先详细介绍了系统的技术选型和主要系统架构，然后详细介绍了其界面功能及最终效果，最后介绍了系统整体的部署结构和具体的使用方法。

第6章 总结与展望

6.1 本文的主要贡献与结论

随着大图数据应用场景的日益增多，大图数据的可视化一直是一个很难完成的任务。一方面，这项工作对计算资源有很高的要求；另一方面，对于原始图结构的直接显示又不能有效的表达出数据包含的有效信息。为此，本文针对大规模图数据的计算难、展示难等问题，从图布局算法开始，分析了现有算法的特点和不足，并从数据计算过程和可视化过程上进行了相关研究，并实现了 GVVIS 原型系统来完成对研究内容的整合。总体上本文的主要贡献有以下几点：

(1) 改进了基于拓扑分层的大图抽样算法，保证在抽样过程中对图结构拓扑特征的保留；同时，为了确保抽样结构的外扩趋势，引入了基于 K-core 的节点分层划分技术。通过以上两者的结合，可以在尽可能保留图的结构特征的基础上保证对原始图结构抽样的离散程度。最后通过与 RN, RE, SBS 及 SS 的对比实验，表明了 KSS 在不同抽样率及不同规模的数据集上得到的结果（相比较其他算法而言）更接近于原始结构，验证了 KSS 抽样算法的效果。

(2) 实现了基于 GraphX 的分布式大图布局算法并验证了算法的有效性，可以充分利用集群的计算资源来完成布局任务。之后，将上述抽样算法应用到布局计算过程中，结合 GraphX 的运行机制提出了基于抽样的大图布局算法，能够获得不同粒度的布局结果，同时为可视化的清晰展现提供了基础条件。通过实验中的布局效果图来看，该算法能够比较清晰地、分层次地对布局结果进行显示，并且最终布局效果同现有布局算法大致一致。

(3) 设计并实现了 GVVIS 大图可视化原型系统，GVVIS 系统实现并使用了前文提出的基于抽样的大图布局算法，并针对图布局结果实现了分层显示等交互式操作功能，整个系统具备完整的图布局输入输出流程，使用了中间层解耦的思想进行设计实现，能够高效的利用远端计算集群为用户提供有效的大图可视化交互服务。

6.2 未来展望

(1) 对输入数据的优化存储

本文未针对图的结构对点集或边表进行存储，整个系统采用的还是 GraphX 的默认点划分策略，由于图算法在执行过程中不可避免的需要频繁访问邻接点，

从而产生大量的跨物理机访问请求，这在设计算法时可以考虑在内。最有效的方法是：采用聚类等方法首先将布局上可能靠近的点存储在同一节点的磁盘上，显然这可以显著减少通信成本，但如何通过图划分将结构尽量完整的（不同聚簇间的相连边最少）分开存储，又涉及计算节点的接近中心性（betweenness）等参数，而类似参数的计算代价非常大，无法直接应用至规模较大的结构，期待在将来可以通过近似接近中性计算等方式来指导完成对图的划分及存储，进而减少跨节点通信请求，进一步优化整体的运行时间。

(2) 对 KSS 分层参数选取的优化

在本文中已经实现并证实了 KSS 的可用性和布局结果的有效性，但目前结果的产生仍然比较依赖分层参数的初始选择（对应初始层抽样结构的规模），在本文第4章的实验中采用的是人工给定分层的形式，在 KSS 布局算法的设计中只暂时给出了层数计算的估计方法，还比较粗糙，下一步可以通过对不同数据的实验来发现分层参数同图的规模、聚集系数等特征的关系，从而可以自动的确定合适的分层参数。

(3) 对各层布局结果覆盖度的检测

在第3章的实验部分，虽然本文已经采用了若干参数来对抽样前后的布局结果进行了相似性检测，但仍然局限在数值指标，对于每层布局同原图的相似性并未进行衡量，目前也没有类似的衡量指标。这里可以用算法的最终布局结果来当做原图的真实结构，然后用每次的布局结果来跟最后的布局结果进行比对，对于整个布局区域来说，两者匹配上的边越多、越分散，则当前层的布局的覆盖度越高，当前层的结构的概括性越好。匹配边的数量对比很容易实现，但其分散程度的衡量还没有合适的解决方案。

(4) 对有向图及加权图的支持

虽然本文中实现的布局算法均针对的是无向图，但对于适应有向图和加权图，需要改动少量代码。对于有向图主要是修改力导向的作用模型，需要调整计算引力时的传播消息的方式，即由原来的双向传播改为单向传播，即对于节点 A 出边指向节点 B 的结构，只计算 B 对 A 的吸引力，斥力仍然保持不变。对于加权图，同样是修改斥力和引力的计算公式，但这里是将权值与力的大小相关联，例如权值大的节点表明关联重要，则需要减小斥力增大引力，从而让该边的两个端点靠的更近。

参考文献

- [1] 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.
- [2] PALMER S, ROCK I. Rethinking perceptual organization: The role of uniform connectedness [J]. Psychonomic bulletin & review, 1994, 1(1): 29-55.
- [3] C T A. Data visualization: principles and practice[M]. AK Peters/CRC Press, 2007.
- [4] WILLIAMS M. Visualization.[J]. Annual Review of Information Science and Technology (ARIST), 1995(30): 161-207.
- [5] JOHNSON J, REITZEL J D, NORWOOD B, et al. Social network analysis: A systematic approach for investigating[J]. FBI Law Enforcement Bulletin., 2013, 350.
- [6] IEEE VIS[EB/OL]. 2019. <http://ieeveis.org/year/2019/welcome>.
- [7] BASTIAN M, HEYMANN S, JACOMY M. Gephi: an open source software for exploring and manipulating networks[C]//Third international AAAI conference on weblogs and social media. 2009.
- [8] BORGATTI S P, EVERETT M G, FREEMAN L C. Ucinet[J]. Encyclopedia of social network analysis and mining, 2014: 2261-2267.
- [9] MRVAR A, BATAGELJ V. Analysis and visualization of large networks with program package pajek[Z]. 2016.
- [10] GHIM G H, CHO N, SEO J. Netminer[M]. 2014.
- [11] SMITH M A, SHNEIDERMAN B, MILIC-FRAYLING N, et al. Analyzing (social media) networks with nodeXL[C]//Proceedings of the fourth international conference on Communities and technologies. ACM, 2009: 255-264.
- [12] EADES P. A heuristic for graph drawing[J]. Congressus numerantium, 1984, 42: 149-160.
- [13] KAMADA T, KAWAI S, et al. An algorithm for drawing general undirected graphs[J]. Information processing letters, 1989, 31(1): 7-15.
- [14] DAVIDSON R, HAREL D. Drawing graphs nicely using simulated annealing[J]. ACM Transactions on Graphics (TOG), 1996, 15(4): 301-331.
- [15] FRUCHTERMAN T M, REINGOLD E M. Graph drawing by force-directed placement[J]. Software: Practice and experience, 1991, 21(11): 1129-1164.
- [16] HADANY R, HAREL D. A multi-scale algorithm for drawing graphs nicely[C]//International Workshop on Graph-Theoretic Concepts in Computer Science. Springer, 1999: 262-277.
- [17] WALSHAW C. A multilevel algorithm for force-directed graph drawing[C]//International Symposium on Graph Drawing. Springer, 2000: 171-182.
- [18] CHAN D M, CHUA K S, LECKIE C, et al. Visualisation of power-law network topologies[C]//The 11th IEEE International Conference on Networks, 2003. ICON2003. IEEE, 2003: 69-74.

- [19] HACHUL S, JÜNGER M. Drawing large graphs with a potential-field-based multilevel algorithm [C]//International Conference on Graph Drawing. 2004.
- [20] HU Y. Efficient, high-quality force-directed graph drawing[J]. Mathematica Journal, 2005.
- [21] 黄竞伟, 康立山. 基于遗传算法的无向图画图算法[J]. 数学杂志, 1998(s1): 68-72.
- [22] 孙炜, 吴伟民, 陈志峰. 基于遗传模拟退火算法的图的三维可视化[J]. 广东工业大学学报, 2002, 19(1): 37-41.
- [23] 赵玉聪, 钟志农, 吴烨, 等. 基于图匹配的分层布局算法[J]. 计算机与现代化, 2015(8): 107-111.
- [24] 汤颖, 盛风帆, 秦绪佳. 基于改进力导引图布局的层级视觉抽象方法[J]. 计算机辅助设计与图形学学报, 2017, 29(4): 641-650.
- [25] HOLTEN D, VAN WIJK J J. Force-directed edge bundling for graph visualization[C]// Computer graphics forum: volume 28. Wiley Online Library, 2009: 983-990.
- [26] BACH B, RICHE N H, HURTER C, et al. Towards unambiguous edge bundling: Investigating confluent drawings for network visualization[J]. IEEE Transactions on Visualization & Computer Graphics, 2016, 23(1): 541-550.
- [27] LESKOVEC J, KREVL A. SNAP Datasets: Stanford Large Network Dataset Collection [EB/OL]. 2014. <http://snap.stanford.edu/data>.
- [28] NOCAJ A, ORTMANN M, BRANDES U. Adaptive disentanglement based on local clustering in small-world network visualization[J]. IEEE transactions on visualization and computer graphics, 2016, 22(6): 1662-1671.
- [29] TELEA A, JALBA A C. Computing curve skeletons from medial surfaces of 3d shapes.[C]// TPCG. 2012: 99-106.
- [30] FALCÃO A, FENG C, KUSTRA J, et al. Multiscale 2d medial axes and 3d surface skeletons by the image foresting transform[M]//Skeletonization. Elsevier, 2017: 43-70.
- [31] TILBURY A, CLEMENTSON T. A scalable parallel force-directed graph layout algorithm. [C]//Eurographics Conference on Parallel Graphics and Visualization. 2008.
- [32] CHAE S, MAJUMDER A, GOPI M. Hd-graphviz: highly distributed graph visualization on tiled displays[C]//Eighth Indian Conference on Computer Vision. 2012.
- [33] TIKHONOVA A, MA K L. A scalable parallel force-directed graph layout algorithm[C]// Proceedings of the 8th Eurographics conference on Parallel Graphics and Visualization. Eurographics Association, 2008: 25-32.
- [34] Gephi Releases[EB/OL]. 2015. <https://github.com/gephi/gephi/releases>.
- [35] DAVIDSON G S, HENDRICKSON B, JOHNSON D K, et al. Knowledge mining with vxinsight: Discovery through interaction[J/OL]. Journal of Intelligent Information Systems, 1998, 11(3): 259-285. <https://doi.org/10.1023/A:1008690008856>.
- [36] 赵南雨, 陈莉君. 一种面向 Hadoop 中间数据存储的混合存储系统[J]. 信息技术, 2017 (11): 169-174.

- [37] 王娟, 冯丹, 王芳, 等. 一种元数据服务器集群的负载均衡算法[J]. 小型微型计算机系统, 2009, 30(4): 757-760.
- [38] 左大鹏, 徐薇. 基于 Hadoop 处理小文件的优化策略[J]. 软件, 2015(2): 107-111.
- [39] Hadoop Releases[EB/OL]. 2019. <https://hadoop.apache.org/>.
- [40] DEAN J, GHEMAWAT S. Mapreduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
- [41] 董新华, 李瑞轩, 周湾湾, 等. Hadoop 系统性能优化与功能增强综述[J]. 计算机研究与发展, 2013, 50(s2): 1-15.
- [42] Apache Spark[EB/OL]. 2019. <https://spark.apache.org/docs/>.
- [43] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.
- [44] GERBESSIONTIS A V, VALIANT L G. Direct bulk-synchronous parallel algorithms[J]. Journal of parallel and distributed computing, 1994, 22(2): 251-267.
- [45] SPANGENBERG N, ROTH M, FRANCZYK B. Evaluating new approaches of big data analytics frameworks[C]//International Conference on Business Information Systems. Springer, 2015: 28-37.
- [46] LOW Y, BICKSON D, GONZALEZ J, et al. Distributed graphlab: a framework for machine learning and data mining in the cloud[J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.
- [47] 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.
- [48] HINGE A, AUBER D. Distributed graph layout with spark[C]//2015 19th International Conference on Information Visualisation. IEEE, 2015: 271-276.
- [49] ARLEO A, DIDIMO W, LIOTTA G, et al. A distributed force-directed algorithm on gigraph: Design and experiments[J/OL]. CoRR, 2016, abs/1606.02162. <http://arxiv.org/abs/1606.02162>.
- [50] LESKOVEC J, FALOUTSOS C. Sampling from large graphs[C]//Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2006: 631-636.
- [51] 朱君鹏, 李晖, 陈梅, 等. SNS: 一种快速无偏的分层图抽样算法[J]. 计算机科学, 2019, 45(11): 249-255.
- [52] LI Y, DU X, YE Y, et al. Stratified sampling large relational networks using topologically divided strata[J]. Procedia Engineering, 2011, 15: 3774-3779.
- [53] 孜甜甜, 卢罡, 许南山, 等. 基于 k-core 的大规模复杂网络压缩布局算法[J]. 计算机工程, 2016, 42(5): 308-312.

- [54] CHEN Y S, CHONG P P, TONG M Y. Mathematical and computer modelling of the pareto principle[J]. Mathematical & Computer Modelling, 1994, 19(9): 61–80.
- [55] CARD S K, MACKINLAY J D, SHNEIDERMAN B. Readings in information visualization: using vision to think[M]. 1999.

致 谢

值此论文完成之际，谨以此篇向对我给予帮助的所有人表示感谢。研究生阶段即将结束，我也即将开始新的征程，很荣幸自己人生中非常重要的几年在东北大学计算机学院度过，治学严谨、求真务实的老师，积极进取、刻苦钻研的同学，无不深深感染着我，短暂的时光里充满了各种美好的回忆，在这里我要感谢所有关心过、帮助过我的老师和同学，并借此机会送上我最诚挚的祝福。

首先要感谢我的硕士导师鲍玉斌教授，感谢您在学习、生活上给我的无私帮助和谆谆教诲，研究生期间在您的带领下我参与了大大小小的各类项目，锻炼了我的实践能力；此外，无数次的组内讨论会也使我的思考能力和学术视野有了极大的提高，这对我即将开始的下阶段学习乃至今后的人生都大有裨益。同时，您对我在今后人生道路的规划上给出了十分重要的意见，也对我在迷茫时给予了我目标和坚定的支持。在此向鲍老师表示我最衷心的感谢。

此外，还要感谢冷芳玲老师对我课题的悉心指导，感谢于戈教授、张天成老师、冯时老师在项目中对我的帮助，感谢王大玲教授对我个人能力的肯定，同时还要感谢计算机学院的其余各位老师，是你们让我感受到了学术的严谨、科研的神圣。再次向各位给予我指导和帮助的老师表示感谢。

还有鲍师门的各位师兄师姐、师弟师妹，是你们让我的研究生生活变得充实完整，是你们让我拥有了一个和睦团结的大家庭。感谢世奇、刘欣、李兰、施雯、林冰、小乐，你们是我研究生期间陪伴我时间最长的人，是一起挥洒过汗水一起奋斗过的“战友”。感谢你们在研究生期间给我留下了最珍贵的回忆，在此祝愿各位已毕业的同门工作顺利，祝愿师弟师妹们能够学业有成，谢谢你们给我带来的这一段最快乐的时光。

感谢父母，是你们的鼓励让我走到了今天，你们在学业和生活上给予了我全力支持，让我拥有了最坚实的后盾，感谢你们为我所做的-切。

最后，还要向参与论文评审的各位专家、学者和老师们表示感谢，感谢您在百忙之中给予我的批评和指正，给我的论文提出最宝贵的意见和建议。

一日东大人，终身东大人，祝愿母校今后能够再创辉煌，祝愿计算机科学与工程学院越办越好。

攻读硕士学位期间取得的学术成果

攻读硕士学位期间参加的科研项目：

- (1) 国家自然科学基金青年基金项目“基于热点导航的大图数据迭代计算过程可视化关键技术研究”(项目编号 61602103), 2017.9 至今。
- (2) 辽宁省渔业渔政省级数据中心项目, 2016.12-2018.5

