软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(3):1051−1073 [doi: 10.13328/j.cnki.jos.007071]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于邻域 *k*-核的社区模型与查询算法*

张 琦, 程苗苗, 李荣华, 王国仁

(北京理工大学 计算机学院, 北京 100081)

通信作者: 李荣华, E-mail: lironghuabit@126.com

摘 要: 现实生活中的网络通常存在社区结构, 社区查询是图数据挖掘的基本任务. 现有研究工作提出了多种模型来识别网络中的社区, 如基于 *k*-核的模型和基于 *k*-truss 的模型. 然而, 这些模型通常只限制社区内节点或边的邻居数量, 忽略了邻居之间的关系, 即节点的邻域结构, 从而导致社区内节点的局部稠密性较低. 针对这一问题, 将节点的邻域结构信息融入 *k*-核稠密子图中, 提出一种基于邻域连通 *k*-核的社区模型, 并定义了社区的稠密度. 基于这一新模型, 研究了最稠密单社区查询问题, 即返回包含查询节点集且具有最高稠密度的社区. 在现实生活图数据中, 一组查询节点可能会分布在多个不相交的社区中. 为此, 进一步研究了基于稠密度阈值的多社区查询问题, 即返回包含查询节点集的多个社区, 且每个社区的稠密度不低于用户指定的阈值. 针对最稠密单社区查询和基于稠密度阈值的多社区查询问题, 首先定义了边稠密度的概念, 并提出了基于边稠密度的基线算法. 为了提高查询效率, 设计了索引树和改进索引树结构, 能够支持在多项式时间内输出结果. 通过与基线算法在多组数据集上的对比, 验证了基于邻域连通 *k*-核的社区模型的有效性和所提出查询算法的效率.

关键词: 社区搜索; 邻域结构; *k*-核子图
中图法分类号: TP311

中文引用格式: 张琦, 程苗苗, 李荣华, 王国仁. 基于邻域 *k*-核的社区模型与查询算法. 软件学报, 2024, 35(3): 1051−1073. http://www.jos.org.cn/1000-9825/7071.htm

英文引用格式: Zhang Q, Cheng MM, Li RH, Wang GR. Community Model and Query Algorithm Based on Neighborhood *k*-core. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1051−1073 (in Chinese). http://www.jos.org.cn/1000-9825/7071.htm

Community Model and Query Algorithm Based on Neighborhood *k*-core

ZHANG Qi, CHENG Miao-Miao, LI Rong-Hua, WANG Guo-Ren

(School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: Real-world networks often exhibit community structures, and community query is a fundamental task in graph data mining. Existing studies introduced various models to identify communities within networks, such as *k*-core based models and *k*-truss based models. Nevertheless, these models typically confine themselves to constraining the number of neighbors of nodes or edges within a community, disregarding the relationships between these neighbors, namely, the neighborhood structure of the nodes. Consequently, the localized density of nodes within communities tends to be low. To address this limitation, this study integrates the information regarding the neighborhood structure of nodes into the *k*-core dense subgraph model, thereby introducing a community model based on neighborhood *k*-core and defining the density of a community. Based on the novel model, this study investigates the densest single community query problem which outputs the community containing the query node set with the highest community density. In real-life networks, the query nodes may be distributed across multiple disjoint communities. To this end, this study further works on the problem of multi-community query based on a density threshold. This entails returning multiple communities that encompass the query node set, with

* 基金项目: 国家重点研发计划(2021VEB3301301); 国家自然科学基金(62072034, U2241211); 中国博士后科学基金(2023M730251, 2023TQ0026)
本文由“面向多模态数据的新型数据库技术”专题特约编辑彭智勇教授、高云君教授、李国良教授、许建秋教授推荐.
收稿时间: 2023-07-16; 修改时间: 2023-09-05; 采用时间: 2023-10-24; jos 在线出版时间: 2023-11-08
CNKI 网络首发时间: 2023-12-22

each community demonstrating a density no lower than the user-specified threshold. For the problem of the densest single community query and the multi-community query based on a density threshold, this study introduces the concept of edge density with which the basic algorithms are proposed. To improve the efficiency, the index tree and the enhanced index tree structures are devised to support outputting results in polynomial time. The effectiveness of the community model based on neighborhood k-core and the efficiency of query algorithms are demonstrated through comparative analyses against basic algorithms using several different datasets.

Key words: community search; neighborhood structure; k-core subgraph

近年来, 随着信息技术的迅猛发展, 各行各业都经历了数据的爆发式增长, 积累了海量的数据. 这些数据不仅类型多样, 涵盖了文本、音频、图像、视频等多种形式, 而且数据之间的关联也变得越发复杂. 图作为一种结构化知识表示的方式, 能够将不同类型的数据映射为节点和边, 从而清晰地表达数据及其之间的关系. 这有助于更好地管理和分析多模态数据, 为各种领域带来了新的应用机会[1−3]. 例如: 在社交媒体上, 一条文本帖子可能包含图像和音频, 并且被很多人评论和转发. 对应到多模态社交网络中, 用户、文本、图像、音频可以表示为图中的节点. 这些节点之间通过不同类型的边相互关联, 如主题相关关系、评论关系、转发关系等. 通过对多模态社交网络进行分析, 可以更精准地为用户提供个性化的内容和推荐服务, 从而提高用户满意度. 在医疗领域, 病患的病历可能包括文字描述、医学图像、实验数据、治疗药物、主治医生等多种信息. 在对应的多模态医疗诊断网络中, 节点表示为病患、医生、文本、医学影像、治疗药物等, 边表示两个节点之间的关系. 比如: 病例关系边连接患者和其医学图像以及实验数据; 就诊边连接患者和医生; 治疗边连接患者和治疗方案、药物等节点. 分析多模态医疗诊断图数据, 可用于帮助医生进行疾病诊断, 预测疾病风险, 以及进行生物医学研究.

现实世界中的多模态图数据往往呈现出局部性, 即图中存在一些局部节点之间连接紧密而与其他外部节点连接稀疏的子图结构, 这种结构通常被称为社区[1−3]. 社区查询是多模态大图数据管理与分析中的一个基本问题, 旨在根据某一稠密子图模型找到包含查询节点集的社区[4−6]. 社区查询已在众多工业领域得到了广泛应用, 例如: 在社交多模态图网络中, 社区通常代表现实生活中的小团体, 查询包含一个用户的社区, 并为其推荐该社区中的非好友人员, 可以增加用户黏性, 从而维护社交网络的稳定性[7]; 在科学协作多模态网络中, 社区中的节点之间通常具有较强的合作关系, 通过查询是否存在一个包含所有成员的社区, 可以确定这些成员是否能够成为可靠的合作团队, 从而为科研项目的审批管理提供帮助[8]; 在蛋白质相互作用多模态网络中, 彼此高度相互作用的蛋白质通常具有相似的功能, 通过查询某一未知功能蛋白质所在的社区, 可以推断其生物学功能[9].

现有的研究工作已经提出了多种模型来识别图数据中的社区, 如基于 k-核的社区[10−12]和基于 k-truss 的社区[13−15]. 然而, 这些模型通常只考虑限制社区内节点或边的邻居数量, 忽略了邻居之间的关系, 即节点的整个邻域结构, 从而导致社区内节点的局部稠密性较差. 此外, 基于这些模型的社区查询通常默认返回包含查询节点集的一个社区, 可能导致社区的整体稠密性不高. 在实际应用中, 将查询节点划分在多个不同的稠密社区, 比将其划分在一个松散的社区更为合理. 以科学协作网络为例, 当查询的学者涉及不同的研究领域时, 将这些学者纳入同一个社区往往需要跨领域构建, 导致社区的稠密性降低. 相比之下, 将查询学者划分到不同的高稠密度社区, 并且每个社区中的学者具有相似的研究领域, 更符合实际应用的需求.

针对现有社区模型及社区查询的局限性, 本文将节点的邻域结构信息融入 k-核稠密子图中, 提出一种新的社区模型——邻域连通 k-核社区, 并定义了社区的稠密度. 基于这一新模型, 研究了两种社区查询问题. (1) 最稠密单社区查询: 给定图 G 和查询节点集 S, 查询包含 S 中节点且具有最大稠密度的基于邻域连通 k-核社区. (2) 基于稠密度阈值的多社区查询: 给定图 G, 查询节点集 S 和稠密度阈值 θ, 查询包含 S 中节点的一个或多个基于邻域连通 k-核社区, 其中, 每个社区的稠密度不小于 θ.

针对这两种社区查询问题, 本文定义了边稠密度的概念, 并提出了基于边稠密度的基线算法. 为了提高查询效率, 设计了索引树和改进的索引树结构, 并提出了相应的社区查询算法.

本文的主要贡献有:

(1) 新的社区模型: 提出了融合节点邻域信息的邻域连通 k-核模型, 用来表征社区结构, 并定义了两种基于邻域连通 k-核社区的查询问题——最稠密单社区查询和基于稠密度阈值的多社区查询.

(2) 高效的查询算法: 引入了边稠密度的概念, 并基于此提出了解决最稠密单社区查询和基于稠密度阈值的多社区查询的基线算法. 为了提高搜索效率, 设计了索引树结构, 能够快速地返回包含查询节点集的最稠密单社区和满足稠密度约束的多社区. 此外, 对索引树进行了优化, 提出了改进的索引树, 能够更高效地返回社区查询结果.

(3) 丰富的实验验证: 在 6 个真实数据集上设计了实验来评估所提出算法的效率. 实验结果表明: 索引树和改进索引树都能高效构建, 并且两种索引结构的内存占用不超过 1 600 MB, 能够存储在现代计算机上; 对于最稠密单社区查询和基于稠密度阈值的多社区查询问题, 基于索引树的算法和基于改进索引树的算法相较于基线算法都能高效地输出查询结果, 而基于改进索引树的算法效率更高. 此外, 通过在 WordNet 和 DBLP 数据集上的案例分析, 证明了基于邻域连通 k-核的社区模型能够找到更完整的社区结构.

1 相关工作

1.1 社区查询

社区查询是指在大规模图网络中找到包含给定查询节点的稠密子图. 常用的稠密子图模型有团[16−18]、k-核[10−12]、k-truss[13−15]等. Yuan 等人基于 k-团渗透社区模型研究了包含给定查询节点的具有最大 k 值的 k-团渗透社区查询问题, 并设计了一个索引 DCPC-Index 以支持高效的社区查询[19]. Sozio 等人提出了一个基于 k-核的社区查询问题, 旨在找到一个包含所有查询节点的连通子图, 并最大化该子图中节点的最小度[20]. 他们证明该问题可以在线性时间内求解, 并提出了基于剥离策略的算法. 随后, Sozio 等人研究了受限的 k-核社区查询问题, 增加了对输出社区大小和社区成员之间直径的限制[20]. Cui 等人研究了针对单个查询节点的 k-核社区查询问题[21]. Barbieri 等人研究了具有最少节点数的 k-核社区查询问题, 构建了一个索引来存储所有 k-核的结构信息, 通过将查询节点连接成一个候选社区, 并对其细化来找到结果[22]. 对于 k-truss 模型, Huang 等人提出了两种不同的约束条件来建立连通的社区模型[4,8]. 第 1 个 k-truss 社区模型基于三角形的连通性, 要求该社区的任意两条边都可以通过一组相互连接的三角形来连通[8]. Huang 等人设计了一个包含三角形连通性和 k-truss 结构的索引, 并提出了基于索引的社区查询算法, 以支持在最佳时间内返回结果. 第 2 个 k-truss 社区模型是基于直径最小的 k-truss[4]. Huang 等人提出了一种查询算法, 通过找到连接所有查询节点的 k-truss 并对其进行缩减, 来尽可能地减少社区的直径. 上述基于 k-核和基于 k-truss 的研究工作通常只限制社区内节点或边的邻居数量, 而忽略了邻居之间的关系, 即节点的整个邻域结构, 导致社区内节点的局部稠密性较差. 此外, 基于这些模型的社区查询通常默认返回包含查询节点集的一个社区, 可能导致社区的整体稠密性较差.

1.2 社区检测

社区检测的目标是找到图中的所有社区, 通常采用全局标准来寻找合格的社区[23]. 传统的社区检测算法通常基于图的拓扑结构, 如基于图分裂的方法、基于模块度的方法和基于谱聚类的方法等. 基于图分裂的方法中最为著名的是 Girvan 和 Newman 提出的 GN 算法[3], 该算法将边的中介性作为删边的条件, 在迭代过程中逐步删除边以得到社区. 为了降低 GN 算法的复杂度, 一些改进的方法被提出, 包括降低边中介性计算复杂度的方法和使用其他指标代替边中介性的方法[24−27]. 在基于模块度的方法中, Newman 等人提出了模块度来衡量社区结构划分的优劣, 从而将社区检测问题转化为了模块度最优化问题[28]. 相关技术如模拟退火、极值优化和特殊矩阵特征值等也被引入到模块度的优化中[29−32]. 基于谱分析的方法利用图的矩阵表示计算相应的特征向量和特征值来实现节点聚类[33−35]. 随着图学习技术的发展, 基于概率图模型和基于深度学习的社区检测方法被提出. 基于概率图模型的方法采用启发式或元启发式来发现网络社区, 通过网络结构模型中的边来

描述实体之间的依赖关系. 根据所使用的概率图模型, 可进一步分为有向图模型[36−48]、无向图模型[48−56]和混合图模型[57−59]. 基于深度学习的方法旨在利用面向社区的网络表示来识别社区结构, 它通过一些学习策略将网络数据从原始输入空间映射到低维特征空间来推导新的网络表示. 根据使用的学习策略, 这类方法又可分为基于自动编码器的方法[60−65]、基于生成对抗网络的方法[66−72]、基于图卷积网络的方法[73−76]以及集成图卷积网络和无向图模型的方法[77]. 社区检测利用全局信息挖掘图中的社区结构, 而本文聚焦社区查询问题, 旨在为查询节点提供个性化的社区发现.

2 基础概念

在这一节, 我们介绍与本文相关的几个重要概念, 定义邻域连通 k-核社区模型, 并基于该模型研究两种社区查询问题: 最稠密单社区查询和基于稠密度阈值的多社区查询. 表 1 罗列了本文用到的主要符号.

表 1 符号表

符号	描述
$G=(V,E)$	无向图
$n=\|V\|$, $m=\|E\|$	节点数,边数
$u,v\in V$	节点 u,v
$(u,v)\in E$	边(u,v)
$N_G(u)=\{v\in V\|(u,v)\in E\}$	节点 u 的邻居集合
$d_G(u)=\|N_G(u)\|$	节点 u 的度
$G_S=(V_S,E_S)$, $V_S=S$, $E_S=\{(u,v)\|u,v\in S,(u,v)\in E\}$	节点集 S 诱导的子图

定义 1(k-核). 给定图 $G=(V,E)$和整数 $k>0$, 如果存在子图 $G'\subseteq G$满足以下条件, 则称 G'是 G 的一个 k-核, 记为 $G_k=(V_k,E_k)$: (1) 对于任意节点 $v\in V'$, $d_{G'}(v)\geqslant k$; (2) G'是满足条件(1)的极大子图.

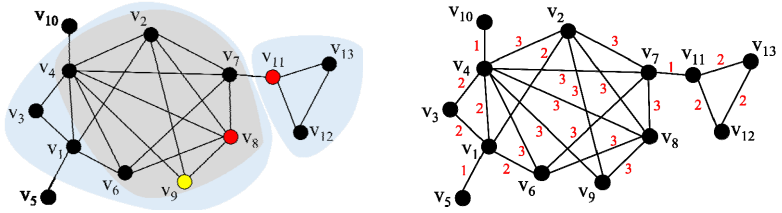
定义 2(核数). 给定图 $G=(V,E)$, 如果存在一个 k 核 $G_k=(V_k,E_k)$使得节点 $u\in V_k$, 但不存在$(k+1)$-核使得 $u\in V_{k+1}$, 那么节点 u 的核数为 k, 记为 $core_G(u)$.

定义 3(邻域网络). 给定图 $G=(V,E)$和节点 $u\in V$, u 的邻域网络, 又称自我中心网络, 是由 u 及其邻居 $N_G(u)$诱导的 G 的子图, 记为 $G_{N[u]}$, 其中, $N[u]=u\cup N_G(u)$.

定义 4(邻域 k-核). 给定图 $G=(V,E)$、节点 $u\in V$ 和整数 $k>0$, u 的邻域网络 $G_{N[u]}$的 k-核为 u 的邻域 k-核, 记为 $G_u^k=(V_u^k,E_u^k)$.

定义 5(邻域连通 k-核). 给定图 $G=(V,E)$、两个不同的节点 u, v 和整数 $k>0$, 如果 u, v 的邻域 k-核 G_u^k 和 G_v^k 至少共享 k 个节点, 即 $|V_u^k\cap V_v^k|\geqslant k$, 则 $G_u^k\cup G_v^k$ 是邻域连通 k-核, 记为 $G_u^k\sim G_v^k$.

以图 1(a)所示的图 G 为例. 假设 $k=2$ 并考虑节点 v_{11} 和 v_{13}. 根据定义, $G_{N[v_{11}]}$ 为由节点集 $N[v_{11}]=\{v_{11},v_{12},v_{13}\}$诱导的子图, $G_{N[v_{13}]}$ 为由节点集 $N[v_{13}]=\{v_{13},v_{11},v_{12}\}$诱导的子图. 可以看到: v_{11} 的邻域 2-核为 $G_{N[v_{11}]}$, v_{13} 的邻域 2-核为 $G_{N[v_{13}]}$. 由于 $N[v_{11}]\cap N[v_{13}]=\{v_{11},v_{12},v_{13}\}\geqslant k=2$ 成立, 因此, $G_{N[v_{11}]}$ 和 $G_{N[v_{13}]}$ 是邻域连通 2-核, 即 $G_{N[v_{11}]}\sim G_{N[v_{13}]}$.



(a) 图 G (b) 图 G 基于边稠密度构建的加权图

图 1 图 G 及其基于边稠密度构建的加权图

定义 6(邻域连通 k-核社区). 给定图 $G=(V,E)$、子图 $C\subseteq G$ 和整数 $k>0$, 如果满足以下条件, 则 C 是基于邻域连通 k-核的社区: (1) 对于 C 中任意一个节点 u, 都存在一个不同的节点 v 满足 $G_u^k\sim G_v^k$; (2) C 是满足条件(1)

的极大子图.

定义 7(社区稠密度). 给定图 $G=(V,E)$和子图 $C\subseteq G$, 如果 C 是邻域连通 k-核社区而不是邻域连通$(k+1)$-核社区, 则社区 C 的稠密度为$\delta(C)=k$.

- 问题定义

基于邻域连通 k-核的社区模型, 本文主要解决两种社区查询问题.

(1) 最稠密单社区查询: 给定图 G 和查询节点集 S, 返回包含 S 的稠密度最大的社区 R.

(2) 基于稠密度阈值的多社区查询: 给定图 G、查询节点集 S 和稠密度阈值$\theta>0$, 返回包含 S 的一个或多个社区集合$\mathcal{R}$, 其中, 每个社区的稠密度都不小于θ, 即$\forall R\in\mathcal{R}$, $\delta(R)\geqslant\theta$.

考虑图 1(a)所示的图 G. 假设查询节点为 v_9, 则图 1(a)中灰色阴影区域为包含 v_9 的最稠密单社区, 即$R=\{v_2,v_4,v_6,v_7,v_8,v_9\}$. 这是因为该社区中每一对节点的邻域 3-核都是连通的, 且该社区是极大的. 此外, 该社区的稠密度为$\delta(R)=3$, 因为其是邻域连通 3-核而不是邻域连通 4-核. 假设查询节点为 $S=\{v_8,v_{11}\}$且稠密度阈值$\theta=2$, 则基于稠密度阈值的多社区查询结果为图 1(a)中蓝色阴影区域覆盖的 2 个社区: $\mathcal{R}=\{C_1,C_2\}$, 其中, 社区$C_1=\{v_1,v_2,v_3,v_4,v_6,v_7,v_8,v_9\}$, 社区 $C_2=\{v_{11},v_{12},v_{13}\}$. 可以验证: C_1 中每一对节点的邻域 2-核都是连通的, C_2 中每一对节点的邻域 2-核也都是连通的. 然而, $C_1\cup C_2$ 中存在节点对的邻域 2-核不连通, 如节点 v_{11} 和 v_7. 根据定义 7, C_1 和 C_2 的稠密度都为$\delta(C_1)=\delta(C_2)=2$. 可以看到, 多社区查询可以根据阈值$\theta$将节点划分至不同的稠密社区.

3 社区查询算法

针对基于邻域连通 k-核的社区查询问题, 本节首先引入边稠密度的概念, 并提出解决最稠密单社区查询和基于稠密度阈值的多社区查询问题的基线算法. 为了提高查询效率, 设计了索引树结构, 以支持快速地返回包含查询节点集的最稠密单社区和满足稠密度约束的多社区. 最后, 对索引树进行了优化, 开发了改进索引树以更高效地返回社区查询结果.

3.1 基线查询算法

3.1.1 边稠密度的定义与计算

根据问题的定义, 最稠密单社区查询需要找到包含查询节点S的邻域连通k-核社区, 其中, k是最大的. 而基于稠密度阈值的多社区查询需要找到包含查询节点S的邻域连通k-核社区, 其中, $k>0$. 因此, 这两种查询问题都可以等价为查询包含节点集 S 的邻域连通 k-核社区. 对于查询节点集 S, 我们假设从其中的某个节点 u 开始进行邻域连通 k-核社区的查询. 由定义 5 和定义 6 可知, 查询过程中需要满足以下条件: 对于 u 邻域 k-核中的每个节点 v, 节点 u 和节点 v 的邻域 k-核至少共享 k 个节点, 即$|V_u^k\cap V_v^k|\geqslant k$. 由于 k-核是一个嵌套结构, 即$(k+1)$-核包含在 k-核中, 因此我们可以为每条边(u,v)计算满足$|V_u^k\cap V_v^k|\geqslant k$ 的最大 k 值(即边稠密度), 以有效地支持查询. 下面正式给出边稠密度的定义.

定义 8(边稠密度). 给定图 $G=(V,E)$和边$(u,v)\in E$, (u,v)的稠密度记为$\delta(u,v)$, 表示节点 u 和节点 v 属于同一邻域连通$\delta(u,v)$-核, 但不属于同一邻域连通$\delta(u,v)+1$-核.

考虑图 1(a)所示的图 G. 对于边(v_2,v_9), 节点 v_2 的邻域 3-核为节点集 $V_1=\{v_2,v_4,v_7,v_8,v_9\}$诱导的子图, 而其邻域 4-核为空图. 节点 v_9 的邻域 3-核为节点集 $V_2=\{v_2,v_4,v_8,v_9\}$诱导的子图, 同样也不存在邻域 4-核. 由于$|V_1\cap V_2|=|\{v_2,v_4,v_8,v_9\}|\geqslant 3$, 因此, v_2 和 v_9 的邻域 3-核是连通的. 然而, v_2 和 v_9 的邻域 4-核不连通, 根据定义 8, 边(v_2,v_9)的稠密度为$\delta(v_2,v_9)=3$.

下面介绍边稠密度的计算方法. 给定节点 u 和节点 v 的共同邻居 $w_1,w_2,\ldots,w_p$. 对于每个共同邻居 w_i, 设它在 u/v 邻域网络中的核数分别为 $core_{G_{N(u)}}(w_i)/core_{G_{N(v)}}(w_i)$. 定义函数 $f_{(u,v)}(w_i)=\min\{core_{G_{N(u)}}(w_i),core_{G_{N(v)}}(w_i)\}$. 令序列 L 为由共同邻居 $w_1,w_2,\ldots,w_p$ 的$f_{(u,v)}(\cdot)$组成的序列, 即 $L=f_{(u,v)}(w_1),f_{(u,v)}(w_2),\ldots,f_{(u,v)}(w_p)$. 根据定义 8, 边$(u,v)$的稠密度$\delta(u,v)$为序列 L 的 *H-index* 值[78−80],即$\delta(u,v)=H\text{-}index(f_{(u,v)}(w_1),f_{(u,v)}(w_2),\ldots,f_{(u,v)}(w_p))$. 需要注意的是: 这里, $f_{(u,v)}(w_i)$取共同邻居 w_i 的核数的最小值是因为 w_i 需要在 u 和 v 的邻域网络中都构成 $f_{(u,v)}(w_i)$-核.

算法 1 展示了边稠密度计算方法 EdgeDensityCMP 的伪代码. 该算法首先为每个节点构建邻域网络并计算其中节点的核数(第 1–4 行), 然后计算每条边的稠密度(第 5–23 行). 所有节点的邻域网络可以利用三角形枚举算法来构造[81−84]. 计算核数采用 peeling 方法, 依次删除度数最小的节点, 并将其度数作为节点的核数. 当删除一个节点后, 同时删除它的所有邻接边并更新邻居节点的度数. 给定一条边(u,v), 当计算完 u 和 v 邻域网络中节点的核数后, 算法 1 为共同邻居 $w_1,w_2,\ldots,w_p$ 计算 $f_{(u,v)}(w_i)$ 并构造序列 L(第 8 行). 在计算序列 L 的 *H-index* 时, 使用变量 m 记录不小于 $\delta(u,v)$ 的数值个数, 并使用数组 *array* 存储每个数值在序列 L 中的个数,即 *array*[i] 代表数字 i 在序列 L 中的个数. 对于序列 L 中每一个数字 x, 算法会增加数组 *array* 对应位置 *array*[x]的值. 如果 $x\geqslant\delta(u,v)$, 则 m 增加 1, 并检查是否需要更新 $\delta(u,v)$. 如果需要更新, 算法会同时修改 m 和 $\delta(u,v)$.

算法 1. EdgeDensityCMP.

输入: 图 $G=(V,E)$.

输出: 每条边(u,v)的稠密度 $\delta(u,v)$.

1. **for** V 中的每个节点 u **do**
2. 构建 u 的邻域网络 $G_{N[u]}$;
3. 计算 $G_{N[u]}$中每个节点 v 的核数 $core_{G_{N[u]}}(v)$;
4. **end for**
5. **for** E 中的每条边(u,v) **do**
6. $L\leftarrow[\cdot]$; //初始化为一个空序列
7. **for** 每一个 u,v 的共同邻居 w_i **do**
8. $f_{(u,v)}(w_i)=\min\{core_{G_{N(u)}}(w_i),core_{G_{N(v)}}(w_i)\}$; $L.append(f_{(u,v)}(w_i))$; //构造序列 L
9. **end for**
10. $\delta(u,v)\leftarrow 0$;
11. $m\leftarrow 0$; //m 代表序列 L 中比$\delta(u,v)$大的数值的个数
12. *array*$\leftarrow[\cdot]$; //*array* 存储每个 L 中的数值在 L 中对应的个数
13. **for** $x\in L$ **do**
14. *array*[x]$\leftarrow$*array*[x]+1;
15. **if** $x\geqslant\delta(u,v)$ **then**
16. $m\leftarrow m+1$;
17. **end if**
18. **while** $m-$*array*[$\delta(u,v)$]$\geqslant\delta(u,v)+1$ **do**
19. $m\leftarrow m-$*array*[$\delta(u,v)$];
20. $\delta(u,v)\leftarrow\delta(u,v)+1$;
21. **end while**
22. **end for**
23. **end for**
24. **return** 每条边(u,v)的稠密度 $\delta(u,v)$

3.1.2 基于边稠密度的算法

根据定义 8, 可以为图 G 中的每条边计算其稠密度, 并将边稠密度视为边的权重, 从而构造出一个加权图. 当加权图中的两个节点通过权值为$\delta(\cdot,\cdot)$的边连接时, 可以确保加入这两个节点的特定$(\delta(\cdot,\cdot)-2)$个共同邻居能构成一个新的邻域连通$\delta(\cdot,\cdot)$-核. 并且, 如果这个邻域连通$\delta(\cdot,\cdot)$-核中存在节点 p 和其他节点 q, 仍然通过权值不小于$\delta(\cdot,\cdot)$的边连接, 则将 q 及特定$(\delta(\cdot,\cdot)-2)$个共同邻居加入其中, 可以形成一个更大的邻域连通$\delta(\cdot,\cdot)$-核. 通过不断地扩展邻域连通$\delta(\cdot,\cdot)$-核, 必定可以获得邻域连通$\delta(\cdot,\cdot)$-核社区. 因此, 针对最稠密单社区查询和基于稠密度阈值的多社区查询问题, 一种最直观方法是, 在基于边稠密度构建的加权图上对查询节点执行广

度优先搜索. 其中, 前者在搜索过程中维护一个变量γ来表示能连通所有查询节点且最大的 k 值, 后者则在搜索过程中判断查询节点是否可以通过权值不小于θ的边连接. 我们将用于解决最稠密单社区查询问题的这种直观方法称为 BasicSCS 算法, 用于解决基于稠密度阈值的多社区查询问题的直观方法称为 BasicMCS 算法. 由于空间限制, 我们省略了 BasicSCS 算法和 BasicMCS 算法的伪代码.

图 1(b)展示了将图 G 中边的稠密度视为边的权重而构造出的加权图. 假设查询节点集 $S=\{v_4,v_9\}$. 对于最稠密单社区查询问题, BasicSCS 算法执行广度优先搜索. 它首先将 v_4 和 v_9 的连边加入优先队列, 其中, 优先队列的优先级为边的稠密度. 接着, 从优先队列中取出稠密度最大的边(v_9,v_8), 更新变量γ的值为$\delta(v_9,v_8)=3$, 并将该边加入集合 $\ddot{R}$. 然后, 算法判断查询节点是否可以通过 $\ddot{R}$ 中的边连通: 如果可以, 则 BasicSCS 算法重新执行广度优先搜索, 将边稠密度不小于γ的边加入结果集 R 并终止; 否则, 算法将 v_8 的连边加入优先队列, 继续下一轮迭代. 在本例中, R 为图 1(a)灰色阴影区域所示. 对于基于稠密度阈值的多社区查询问题, 假设查询节点集 $S=\{v_8,v_{11}\}$且$\theta=2$. BasicMCS 算法同样执行广度优先搜索. 由于 v_7 和 v_{11} 的边稠密度$\delta(v_7,v_{11})=1<\theta=2$, 因此, 广度优先搜索断开, 基于稠密度阈值的多社区查询结果为图 1(a)中蓝色阴影区域覆盖的 2 个社区.

3.2 基于索引树的查询算法

基线算法 BasicSCS 和 BasicMCS 的时间复杂度较高, 因为它们无法快速判定查询节点是否处于同一个社区. 考虑到邻域连通 k-核的传递性, 一种更高效的方法是, 根据不同边稠密度的连通性构造索引. 并查集(disjoint-set)是一种经典的数据结构, 通过维护若干个不重叠集合, 可以高效地判断两个元素是否在同一个集合. 因此, 可以采用类似于并查集的方式将加权图转化成索引树结构, 从而支持高效的社区查询. 本节首先介绍类并查集的索引树结构及其构造方法, 接着介绍针对两种社区查询问题的基于索引的处理算法.

3.2.1 索引树的结构与构建

索引树 Index, 记为 T, 是由图 G 中所有节点构成的树. 对于树中的每一个节点 u, 都附有变量$f(u)$表示其在树中的父亲节点, 变量 $deep(u)$表示其在树中的深度, 变量 $w(u)$表示该节点与其父亲节点之间连边的稠密度, 即 $w(u)=\delta(u,f(u))$. 索引树 T 的构建方法 IndexConstruct 的伪代码如算法 2 所示. 具体而言, 该算法首先执行 EdgeDensityCMP 算法(算法 1)计算出边的稠密度, 按照边稠密度非递增的顺序对 G 中的所有边排序, 并初始化一棵空树 T(第 1–3 行). 然后, 算法按序将边加入索引树 T 中(第 7–13 行). 设 $r(u)$表示 u 在当前 T 中对应的根节点, 即所属的集合.对于新加入的边(u,v), 算法通过查询 $r(u)$和 $r(v)$来判断节点 u 和 v 是否已经在 T 中连通. 如果已经连通, 即 $r(u)=r(v)$, 则表示 u 和 v 可以被邻域连通 k-核所包含, 其中, $k\geqslant\delta(u,v)$, 因此不需要将这条边加入索引树 T 中; 否则, 将边(u,v)加入当前的索引树 T 中并更新相关变量值(第 9 行). 最后,算法计算每个节点在 T 中的深度 $T.deep(u)$和其父亲节点 $T.f(u)$并返回索引树 T(第 12、13 行). 可见: 在索引树 T 中任意由根节点到叶子节点的路径上, 离根节点越远的边权值越大. 也就是说, 对于选定的节点 u,假设 u 与父亲节点 $f(u)$之间的边的权值为 k, 则 u 和其子孙在 T 中的连边权值都不小于 k, 因此, $f(u)$、u 及 u 的子孙可以组成一个邻域连通 k-核.

算法 2. IndexConstruct.

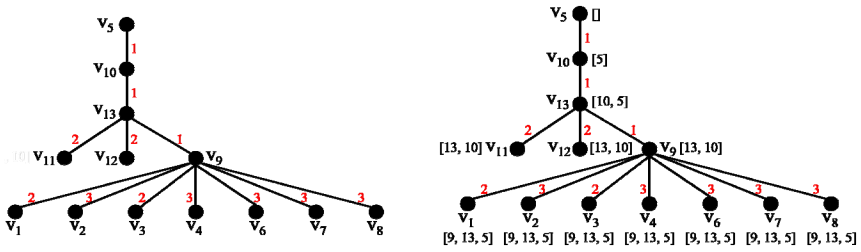
输入: 图 $G=(V,E)$, 每条边(u,v)的稠密度$\delta(u,v)$.

输出: 索引树 Index T.

1. 执行 EdgeDensityCMP 算法;
2. 对 E 中所有的边按稠密度非递增的顺序排序;
3. $T\leftarrow$一个包含所有节点, 但没有边的空树;
4. **for** V 中的每个节点 u **do**
5. $\quad T.w(u)\leftarrow 0$; $T.f(u)\leftarrow u$; $T.deep(u)\leftarrow 0$; $r(u)\leftarrow u$;
6. **end for**
7. **for** E 中的每条边(u,v) **do**
8. $\quad$ **if** $r(u)\neq r(v)$ **then**

9. $r(u)\leftarrow r(v)$; $T.w(u)\leftarrow\delta(u,v)$; $T\leftarrow T\cup(u,v)$;
10. **end if**
11. **end for**
12. 在 T 上执行 DFS 搜索得到每个节点的深度 $T.deep(u)$和其父亲节点 $T.f(u)$;
13. **return** 索引树 Index T

考虑图 G 如图 1(a)所示, 图 2(a)展示了由 G 构建的索引树 T, 其中, 节点之间连边上的值为边的权值, 即边稠密度. 对于节点 v_2, 我们有 $T.f(v_2)=v_9$, $T.w(v_2)=\delta(v_2,v_9)=3$, $T.deep(v_2)=4$. 可以观察到: 在索引树 T 中, 沿着从根节点到叶子节点的任意路径上, 边的权值都是非递减的.



(a) 图 G 构建的索引树 T (b) 图 G 构建的改进索引树 $\hat{T}$

图 2 图 G 构建的索引树 T 和改进索引树 $\hat{T}$

3.2.2 基于索引树的算法

首先介绍基于索引树的最稠密单社区查询算法 IndexBasedSCS. 假设查询节点集 $S=\{q_1,q_2,\ldots,q_m\}$. 根据索引树的性质, 查询节点集合 S 的最稠密社区, 即 k 最大的邻域连通 k-核, 需要首先获得这些节点在索引树上的最近公共祖先, 即 $a_S=lca(q_1,q_2,\ldots,q_m)$. IndexBasedSCS 算法通过迭代计算当前节点与之前所有查询节点的最近公共祖先来得到 a_S, 即 $lca(q_1,q_2,\ldots,q_m)=lca(q_1,lca(q_2\ldots lca(q_{m-1},q_m)))$. 在计算两个节点 u 和 v 的最近公共祖先时, IndexBasedSCS 算法首先找到 u 和 v 具有相同深度的祖先节点 a_u 和 a_v, 即 $deep(a_u)=deep(a_v)$. 当 a_u 与 a_v 深度相同后, 令它们同时向父亲节点移动直到第 1 次遇见相同的祖先节点, 这个相同的祖先节点即为最近公共祖先. 在搜索 a_S 的过程中, IndexBasedSCS 算法使用变量γ表示最大稠密度值, 即各查询节点到最近公共祖先路径上的边权最小值. 如果 a_S 与其父亲节点 $f(a_S)$连边的权值不小于γ, 则算法继续迭代, 直到找到满足条件 $\delta(\hat{a}_S,f(\hat{a}_S))<\gamma$ 的祖先节点 $\hat{a}_S$; 否则, a_S即为满足条件的 $\hat{a}_S$. 设集合 M 表示与公共祖先 $\hat{a}_S$ 连边不小于γ的孩子节点. IndexBasedSCS 算法返回以 M 中节点为根的子树中节点以及公共祖先节点 $\hat{a}_S$ 作为最稠密单社区查询的结果. 在最坏情况下, 单个最近公共祖先的时间复杂度可能达到 O(n), 因此, IndexBasedSCS 算法的总时间复杂度达到 O($|S|n$), 其中, n 为节点的个数.

下面介绍基于索引树的多社区查询算法 IndexBasedMCS. 在索引树上查询包含多个节点且稠密度不小于阈值θ的社区等价于查询这些节点的祖先, 并要求这些祖先与其父亲节点的连边权值小于θ, 但与子孙节点的连边权值都不小于θ. 因此, 与 IndexBasedSCS 算法类似, IndexBasedMCS 算法也需要通过每次向父亲节点移动来搜索祖先节点. 不同的是, IndexBasedMCS 算法不需要对所有查询节点的祖先进行合并, 从而得到一个公共祖先, 而是需要找到查询集合中每个节点的合法祖先, 即与其父亲节点的连边权值小于θ, 但与子孙节点的连边权值都不小于θ的节点. 假设 $\hat{a}_S$ 为任意一个合法祖先,集合 M 表示与 $\hat{a}_S$ 连边不小于θ的孩子节点, 则 IndexBasedMCS 算法返回以 M 中节点为根的子树中节点以及合法祖先 $\hat{a}_S$ 作为一个满足稠密度约束的社区. IndexBasedMCS 算法的最坏时间复杂度仍为 O($|S|n$).

由于空间限制, 我们省略了 IndexBasedSCS 和 IndexBasedMCS 的伪代码.

考虑图 G 如图 1(a)所示, 其索引树如图 2(a)所示. 假设查询节点为 v_9. 由于只有 1 个查询节点, IndexBasedSCS 算法将 v_9 作为根节点. 以 v_9 为根节点的子树中边的最大权值为 3, 因此包含 v_9 的最稠密单社区为$\{v_2,v_4,v_6,v_7,v_8,v_9\}$, 即图 1(a)中灰色阴影区域, 并且该社区的稠密度为 3. 假设查询节点为 $S=\{v_8,v_{11}\}$且稠密

度阈值θ=2. IndexBasedMCS 算法首先在索引树中找到节点 v_8 的父亲节点 v_9, 由于 v_9 与其父亲节点 v_{13} 的边稠密度为 $1<\theta=2$, 因此包含 v_8 且稠密度不小于θ的社区为 $C_1=\{v_1,v_2,v_3,v_4,v_6,v_7,v_8,v_9\}$.对于节点 v_{11}, 同样找到其父亲节点 v_{13}, 由于 v_{13} 与其父亲节点 v_{10} 的边稠密度为 $1<\theta=2$, 因此包含 v_{11} 且稠密度为 2 的社区为 $C_2=\{v_{11},v_{12},v_{13}\}$. IndexBasedMCS 算法返回$\mathcal{R}=\{C_1,C_2\}$作为基于稠密度阈值的多社区查询结果.

3.3 基于改进索引树的查询算法

3.3.1 改进索引树的结构与构建

通过分析基于索引树的查询算法可知: 当查询多个节点的社区时, 需要计算这些节点在索引树 T 上的最近公共祖先. 在索引树 T 中, 每个节点仅存储了其父亲节点的信息. 搜索两个节点的最近公共祖先需要依次向父亲节点搜索, 这种方法在最坏情况下的时间复杂度可以达到 O(n). 如果对每一个节点 u 存储多阶祖先节点, 即存储 2^d(d=0,1,2,…,$\lfloor\log_2 deep(u)\rfloor$)阶祖先节点, 则每个节点最多存储 log($n$)个多级祖先. 通过跳跃式的查询方法, 可以在 O(log$n$)的时间复杂度下查询节点的最近公共祖先. 基于以上思路, 我们提出了改进的索引树结构, 通过存储多级祖先节点来提高最近公共祖先的查询效率.

改进索引树 EnIndex 的结构与索引树结构相似, 区别在于索引树只为每个节点存储其父亲节点, 而改进索引树则存储其 2^d(d=0,1,2,…,$\lfloor\log_2 deep(u)\rfloor$)阶祖先节点. 改进索引树的构建方法称为 EnIndexConstruct, 如算法 3 所示. 该算法首先利用算法 2 构建索引树 T(第 1 行), 然后从 T 的根节点开始执行先序遍历的深度优先搜索(第 2–16 行). 令 *root* 为索引树 T 的根节点, $\mathcal{F}(u)$表示存储节点 u 多阶祖先的数组. 当搜索到一个节点时,可以确定该节点的父亲节点为它的 2^0 阶祖先. 根据先序遍历的性质, 搜索当前节点时已经计算完祖先节点的多阶祖先, 因此可以通过递推计算当前节点的多阶祖先. 假设当前节点为 u, 其 2^i 阶祖先节点为$\mathcal{F}(u)[i]$,可以推导出 u 的 2^{i-1} 阶祖先节点的 2^{i-1} 阶祖先节点是 u 的 2^i 阶祖先节点, 即$\mathcal{F}(u)[i]=\mathcal{F}(\mathcal{F}(u)[i-1])[i-1]$(第 11 行).通过此递推公式, 可以快速得到每个节点的多阶祖先节点. 最后, EnIndexConstruct 算法返回改进索引树 $\hat{T}$ (第 18 行).

算法 3. EnIndexConstruct.

输入: 图 $G=(V,E)$, 每条边(u,v)的稠密度$\delta(u,v)$.

输出: 改进索引树 EnIndex $\hat{T}$.

1. 执行 IndexConstruct 构建查询树索引 T;
2. *root*←T 的根节点;
3. $\mathcal{Q}$←一个空的队列;
4. $\mathcal{Q}$.*push*(*root*);
5. **while** $\mathcal{Q}\neq\varnothing$ **do**
6. u←$\mathcal{Q}$.*pop*(·);
7. d=$\lfloor\log_2 T.deep(u)\rfloor$;
8. $\mathcal{F}(u)$←[·]; //大小为 d 的数组,用于存储节点 u 的多阶祖先节点;
9. $\mathcal{F}(u)[0]$←$T.f(u)$;
10. **for** i 从 1 到 d **do**
11. $\mathcal{F}(u)[i]$←$\mathcal{F}(\mathcal{F}(u)[i-1])[i-1]$;
12. **end for**
13. **for** $v\in N_T(u)$且 $v\neq T.f(u)$ **do**
14. $\mathcal{Q}$.*push*(v);
15. **end for**
16. **end while**
17. $\hat{T}\leftarrow T\cup(\mathcal{F}=\{\mathcal{F}(u)\,|\,u\in V\})$;
18. **return** 改进索引树 EnIndex $\hat{T}$

考虑图 1(a)所示图 G. 图 2(b)展示了 G 构建的改进索引树 $\hat{T}$, 其中, 每个节点保存了 2^d(d=0,1,2,…,$\lfloor \log_2 deep(u) \rfloor$)阶祖先节点. 对于节点 v_2, 其深度为 $T.deep(v_2)$=4, 因此我们保存 2^d(d=0,1,2,…,$\lfloor \log_2 4 \rfloor$)阶祖先. 第 0 阶祖先为 v_2 的父亲节点, 即$\mathcal{F}(v_2)[0]\leftarrow v_9$. 根据递推公式, 可以得到第 1 阶祖先为 v_{13}, 第 2 阶祖先为 v_5. 对于节点 v_{11}, 其深度为 $T.deep(v_{11})$=3, 因此改进索引树 $\hat{T}$ 保存其第 0 阶祖先为父亲节点 v_{13}, 根据递推公式, 可以得到第 1 阶祖先为 v_{10}.

3.3.2 基于改进索引树的算法

基于改进索引树的最稠密单社区查询算法称为EnIndexBasedSCS, 其伪代码如算法4所示. 该算法利用改进索引树存储的多阶祖先信息, 通过类似于二分搜索的方法实现高效的最近公共祖先查找. 具体而言, 当按照二进制从大到小的顺序查找两个节点 u 和 v 的祖先节点时, 如果当前得到的两个祖先节点 a_u 和 a_v 是不同的, 即 $a_u \neq a_v$, 说明最近公共祖先的深度比 a_u 和 a_v 都要浅. 而当 a_u 和 a_v 相同且所有更深的 a_u 和 a_v 不同时, a_u/a_v 就是 u 和 v 的最近公共祖先节点. 在 EnIndexBasedSCS 算法中, 节点 u 被认为是深度更深的节点. 该算法首先找到与节点 v 深度相同的 u 的祖先节点(第 8–13 行), 然后对同一深度的两个节点按照类似于二分搜索的方法查找最近公共祖先 a_S(第 17–21 行). 在找到 a_S 后, EnIndexBasedSCS 算法使用变量γ来表示最大稠密度值, 并根据 a_S 与其父亲节点连边的权值来决定是否继续向上搜索祖先节点 $\hat{a}_S$ (第 25–28 行). 最后, EnIndexBasedSCS 算法返回以 M 中节点为根的子树中节点以及公共祖先节点 $\hat{a}_S$ 作为最稠密单社区查询的结果(第 30 行). 利用改进索引树, EnIndexBasedSCS 算法可以在 O(|S|log$n$)的时间内返回包含查询节点集 S 的最高稠密度社区.

算法 4. EnIndexBasedSCS.

输入: 图 G=(V,E), 查询节点集 S,多级祖先索引树 $\hat{T}$.

输出: 包含查询节点集 S 的最稠密社区 R.

1. $r\leftarrow S[0]$; //初始化 r 为集合 S 中第 1 个节点
2. **for** $S\backslash S[0]$中的每个节点 v **do**
3. $u\leftarrow r$;
4. **if** $\hat{T}.deep(u) < \hat{T}.deep(v)$ **then**
5. $t\leftarrow u$; $u\leftarrow v$; $v\leftarrow t$; //更新节点 u 为深度更深的节点
6. **end if**
7. $d \leftarrow \left\lfloor \log_2 \hat{T}.deep(u) \right\rfloor$;
8. **while** $d\geqslant 0$ **do** //统一节点 u 和 v 的祖先节点的深度
9. **if** $\hat{T}.deep(u) - (1 \ll d) \geqslant \hat{T}.deep(v)$ **then**
10. $u\leftarrow \mathcal{F}(u)[d]$;
11. **end if**
12. $d\leftarrow d-1$;
13. **end while**
14. **if** u=v **then**
15. $r\leftarrow u$;
16. **else**
17. **for** d 从$\left\lfloor \log_2 \hat{T}.deep(u) \right\rfloor$到 0 **do** //按照二进制递减顺序查找最近公共祖先
18. **if** $\mathcal{F}(u)[d]\neq\mathcal{F}(v)[d]$ **then**
19. $u\leftarrow \mathcal{F}(u)[d]$; $v\leftarrow \mathcal{F}(u)[d]$;
20. **end if**
21. **end for**

22. $r\leftarrow\mathcal{F}(u)[0]$;
23. **end if**
24. **end for**
25. $\gamma\leftarrow$以 r 为根节点的子树中的最小边权;
26. **while** $\delta(r,\mathcal{F}(r)[0])=\gamma$ **do**
27. $r\leftarrow\mathcal{F}(r)[0]$;
28. **end while**
29. $M\leftarrow\{w|\delta(r,w)\geqslant\gamma\}$;
30. **return** $R\leftarrow$以 M 中节点为根的子树中节点以及 r 节点组成的社区

针对基于稠密度阈值的多社区查询问题, 算法 5 展示了基于改进索引树的查询算法 EnIndexBasedMCS. 对于查询节点集中的节点 u, 该算法利用改进索引树中存储的多阶祖先信息, 通过类似二分查找的方法找到 u 所在的邻域连通θ-核社区. 在查找过程中, 如果当前得到的祖先节点 r 与父亲节点的连边小于稠密度阈值θ, 说明合法祖先节点的深度比 r 深, 则舍弃得到的祖先节点 r; 如果得到的祖先节点 r 与父亲节点的连边要大于 θ, 说明合法祖先节点的深度比 r 浅, 则接收这个祖先节点 r. 也就是说, EnIndexBasedMCS 算法仍然按照多阶祖先从高到低的顺序查找, 找到最后一个与父亲连边权值小于θ的合法祖先节点 $\hat{a}_S$ (第 4–11 行). 在找到每个查询节点的合法祖先节点后, 令集合 M 表示与 $\hat{a}_S$ 连边不小于θ的孩子节点, 则以 M 中节点为根的子树中节点以及合法祖先 $\hat{a}_S$ 是一个满足稠密度约束的社区. 对查询的节点进行归类, 确定在同一邻域连通θ-核的节点和在不同邻域连通θ-核的节点, 即可计算出满足稠密度约束的多个社区(第 14–19 行).

在 EnIndexBasedMCS 算法中, 对于每一个节点 u, 查询它所在邻域连通θ-核社区的时间复杂度为 O(log$n$). 由于共有$|S|$个待查询节点, 因此算法 EnIndexBasedMCS 的时间复杂度为 O($|S|$log$n$).

算法 5. EnIndexBasedMCS.

输入: 图 $G=(V,E)$, 查询节点集 S,稠密度阈值θ, 多级祖先索引树 $\hat{T}$.

输出: 包含查询节点集 S 的多社区$\mathcal{R}$.

1. $R\leftarrow[\cdot]$;
2. **for** S 中的每个节点 u **do**
3. $a_u\leftarrow u$;
4. **for** d 从$\left\lfloor \log_2 \hat{T}.deep(a_u)\right\rfloor$到 0 **do**
5. **if** $\mathcal{F}(a_u)[d]$与父亲节点连边的权值$\geqslant\theta$ **then**
6. $a_u\leftarrow\mathcal{F}(a_u)[d]$;
7. **end if**
8. **end for**
9. **if** $a_u\neq root$ 且 a_u 与$\mathcal{F}(a_u)[0]$连边的权值$\geqslant\theta$ **then**
10. $a_u\leftarrow\mathcal{F}(a_u)[0]$;
11. **end if**
12. $R\leftarrow R\cup\{a_u\}$;
13. **end for**
14. **for** R 中的每个节点 $\hat{a}_S$ **do**
15. $M\leftarrow\{w|\delta(\hat{a}_S,w)\geqslant\theta\}$;
16. $\hat{R}\leftarrow$ 以 M 中节点为根的子树中节点以及 $\hat{a}_S$ 节点组成的社区;
17. $\mathcal{R}\leftarrow\mathcal{R}\cup\hat{R}$;
18. **end for**

19. **return** 满足稠密度约束的多个社区$\mathcal{R}$

3.4 复杂度分析

表 2 总结了不同算法的时间复杂度和空间复杂度, 下面提供相关的理论证明.

表 2 算法复杂度

算法	功能	时间复杂度	空间复杂度
EdgeDensityCMP	边稠密度计算	O(αm)	O($m+n$)
IndexConstruct	索引树构建	O($\alpha m+\log m+n$)	O($m+n$)
EnIndexConstruct	改进索引树构建	O($\alpha m+\log m+n\log n$)	O($m+n\log n$)
BasicSCS	最稠密单社区查询	O($\|S\|(n+m)$)	O($m+n$)
IndexBasedSCS		O($\|S\|n$)	O(n)
EnIndexBasedSCS		O($\|S\|\log n$)	O($n\log n$)
BasicMCS	基于稠密度阈值的多社区查询	O($\|S\|(n+m)$)	O($m+n$)
IndexBasedMCS		O($\|S\|n$)	O(n)
EnIndexBasedMCS		O($\|S\|\log n$)	O($n\log n$)

定理 1. 给定图 $G=(V,E)$, EdgeDensityCMP 算法的时间复杂度和空间复杂度分别为 O(αm)和 O($m+n$), 其中, α为图 G 的树性.

证明: 在算法 1 的第 1、2 行, EdgeDensityCMP 为每个节点构建邻域网络需要的时间是 $\mathrm{O}\left(\sum_{(u,v)\in E}\min\{d(u),d(v)\}\right)=\mathrm{O}(\alpha m)$. 第 3 行为每个节点的邻域网络计算核数, 共需要 $\mathrm{O}\left(\sum_{u\in V}d(u)\right)=\mathrm{O}(m)$, 因此, 第 1–4 行的时间复杂度为 O($\alpha m$). 在第 5–12 行中, 算法为每条边计算共同邻居, 并构造序列 L, 消耗的时间为 $\mathrm{O}\left(\sum_{(u,v)\in E}\min\{d(u),d(v)\}\right)=\mathrm{O}(\alpha m)$. 第 13 –22 行计算序列 L 的 *h-index* 来得到每条边的稠密度, 时间复杂度为 $\mathrm{O}\left(\sum_{(u,v)\in E}\min\{d(u),d(v)\}\right)=\mathrm{O}(\alpha m)$. 综上, EdgeDensityCMP 算法的时间复杂度为 O(αm). 对于空间复杂度, 算法构建节点的邻域网络最多消耗 O($m+n$)的空间. 在存储核数, 序列 L 以及边稠密度等时, 只需要几个与节点数或边数成线性关系的数据结构. 因此, EdgeDensityCMP 算法的空间复杂度为 O($m+n$).

定理 2. 给定图 $G=(V,E)$和查询节点集 S, BasicSCS 算法和 BasicMCS 算法的最坏时间复杂度为 O($|S|(n+m)$), 空间复杂度为 O($m+n$).

证明: BasicSCS 算法和 BasicMCS 算法的核心思想都是在基于边稠密度构建的加权图上对查询节点执行广度优先搜索. 对于查询节点集 S 中的每个节点, 执行广度优先搜索的时间复杂度为 O($m+n$), 因此, BasicSCS 和 BasicMCS 算法的最坏时间复杂度为 O($|S|(n+m)$). 至于空间复杂度, 算法存储图需要 O($m+n$)的空间, 执行广度优先搜索需要 O(n)的空间, 因此总空间复杂度为 O($m+n$).

定理 3. 给定图 $G=(V,E)$, IndexConstruct 算法构建索引树的时间复杂度为 O($\alpha m+\log m+n$), 空间复杂度为 O($m+n$).

证明: 在算法 2 的第 1 行中, IndexConstruct 算法执行算法 1 计算每条边的稠密度, 需要 O(αm)的时间. 第 2 行对边排序需要 O($\log m$)的时间. 第 4–6 行为每个节点初始化一些变量, 需要 O(n)的时间. 在第 7–10 行, 为每条边执行并查集的查询与合并操作需要 O(m)的时间. 第 12 行在构建好索引树 T 上从根节点执行 DFS 得到每个节点的深度, 由于 T 包含 n 个节点和($n-1$)条边, 因此消耗 O($n+n-1$)=O(n)的时间. 综上, IndexConstruct 算法的时间复杂度为 O($\alpha m+\log m+n+m+n$)=O($\alpha m+\log m+n$). 对于空间复杂度, IndexConstruct 执行算法 1 需要 O($m+n$)的空间. 对 T 的每个节点存储变量 $r(\cdot)$, $T.w(\cdot)$, $T.deep(\cdot)$以及 $T.f(\cdot)$需要 O(n)的空间. 因此, IndexConstruct 算法占用的总空间为 O($m+n$).

定理 4. 给定图 $G=(V,E)$和查询节点集 S, 基于索引树 T 的 IndexBasedSCS 算法和 IndexBasedMCS 算法的时间复杂度和空间复杂度分别为 O($|S|n$)和 O(n).

证明: IndexBasedSCS 算法和 IndexBasedMCS 算法需要执行$|S|$次循环来找到所有查询节点的最近公共祖先. 计算两个节点的最近公共祖先需要 O(n)的时间, 因此, IndexBasedSCS 算法和 IndexBasedMCS 算法的时间复杂度为 O($|S|n$). 至于空间复杂度, 这两个算法都只需要存储索引树 T, 因此空间复杂度为 O(n).

定理 5. 给定图 $G=(V,E)$, EnIndexConstruct 算法构建改进索引树 $\hat{T}$ 的时间复杂度为 $O(\alpha m+\log m+n\log n)$, 空间复杂度为 $O(m+n\log n)$.

证明: 在算法 3 的第 1 行中, EnIndexConstruct 构建索引树 T 需要 $O(\alpha m+\log m+n)$的时间. 第 5–16 行从根节点执行深度优先遍历的时间复杂度为 $O(n)$, 为每个节点计算多级祖先需要 $O\left(n\sum_{u\in V}d\right)=O(n\log n)$ 的时间, 因此共消耗 $O(n\log n)$的时间. 综上, 算法 3 的时间复杂度为 $O(\alpha m+\log m+n\log n)$. 对于空间复杂度, 改进索引树 $\hat{T}$ 在索引树的基础上还需为每个节点储存多级祖先, 这需要额外的 $O(n\log n)$的空间, 因此, 算法 3 的空间复杂度为 $O(m+n\log n)$.

定理 6. 给定图 $G=(V,E)$和查询节点集 S, 基于改进索引树 $\hat{T}$ 的 EnIndexBasedSCS 和 EnIndexBasedMCS 算法的时间复杂度和空间复杂度分别为 $O(|S|\log n)$和 $O(n\log n)$.

证明: 在算法 4 和算法 5 中, EnIndexBasedSCS 和 EnIndexBasedMCS 需要执行$|S|$次循环来找到所有查询节点的最近公共祖先. 基于改进索引树 $\hat{T}$, 计算两个节点的最近公共祖先需要 $O(1\log n)$的时间, 因此, EnIndexBasedSCS 算法和 EnIndexBasedMCS 算法的时间复杂度为 $O(|S|\log n)$. 对于空间复杂度, 两个算法只需要存储索引树 $\hat{T}$, 因而空间复杂度为 $O(n\log n)$.

4 实验分析

4.1 实验数据

我们在 6 个不同规模的真实数据集上进行实验, 这些数据集包括 4 个社交网络、一个通信网络以及一个科学协作网络. 所有的数据集都可以从 https://snap.stanford.edu/data 网站下载. 数据集的详细统计信息见表 3.

表 3 实验数据集

数据集	节点数	边数	类型	空间占用(MB)
Facebook	2 888	2 981	社交网络	0.022
Youtube	1 134 890	2 987 624	社交网络	22.794
WikiTalk	2 394 385	4 659 565	通信网络	35.550
DBLP	1 843 615	8 350 259	科学协作网络	63.707
Pokec	1 632 803	22 301 964	社交网络	170.150
LiveJournal	3 997 962	34 681 189	社交网络	264.596

4.2 实验环境配置

针对最稠密单社区查询问题, 本文实现了基于边稠密度的查询算法 BasicSCS、基于索引树的查询算法 IndexBasedSCS 以及基于改进索引树的查询算法 EnIndexBasedSCS. 针对基于稠密度阈值的多社区查询, 实现了基于边稠密度的查询算法 BasicMCS、基于索引树的查询算法 IndexBasedMCS (算法 4)以及基于改进索引树的查询算法 EnIndexBasedMCS (算法 5). 我们还实现了索引树和改进索引树的构建算法: IndexConstruct (算法 2)和 EnIndexConstruct (算法 3). 所有算法都使用 C++实现, 使用 gcc7.5 编译. 实验均在一台系统为 Ubuntu18.04WSL 的服务器上进行, 服务器内存为 64 GB, CPU 为 Intel(R) Core(TM) i5-7200UCPU@2.50 GHz.

4.3 实验方法

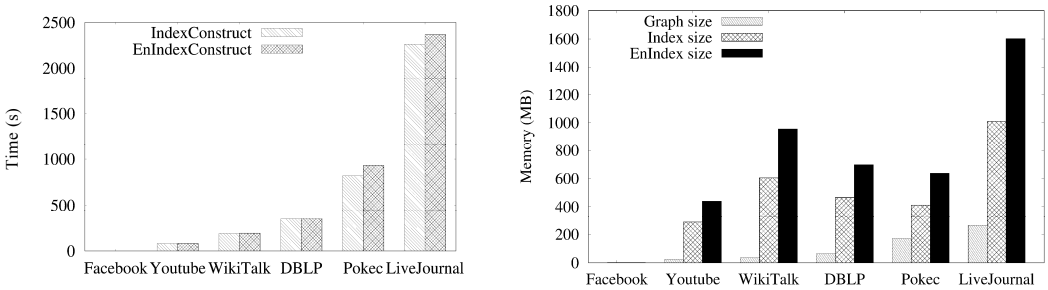
最稠密单社区查询问题中存在一个参数: 查询节点集 S. 在评估最稠密单社区查询算法时, 我们变化 S 的大小为$|S|\in\{5,10,15,20,25\}$, 且$|S|$的默认值为 10. 基于稠密度阈值的多社区查询问题中, 除查询节点集 S 外, 还有一个参数为稠密度阈值θ. 我们变化 S 的大小$|S|\in\{5,10,15,20,25\}$和$\theta\in\{3,4,5,6,7\}$进行算法评估, 其中, $|S|$的默认值为 10, θ的默认值为 3. 在实验中, 除非特殊声明, 所选取的参数均为其默认值. 对于每一个查询集大小, 我们针对每一个数据集随机生成 10 000 个查询集进行查询, 算法消耗的时间为 10 000 次查询的总和.

4.4 实验结果与分析

1) 索引树和改进索引树的评估

实验使用 IndexConstruct 和 EnIndexConstruct 算法为 6 个数据集分别构建索引树和改进索引树. 图 3(a)展

示了这两个算法在不同数据集上的运行时间. 在所有数据集上, IndexConstruct 算法构建索引树的时间略快于 EnIndexConstruct 算法构建改进索引树的时间. 这是因为 EnIndexConstruct 算法需要先执行 IndexConstruct 来构建索引树, 然后再对其进行改进. 同时, 这两种索引构建算法效率非常高, 均能在 2 500 s 内完成对具有百万级节点和千万级边的 LiveJournal 数据集的索引构建. 实验还评估了 6 种数据集上索引树和改进索引树的内存占用情况, 实验结果如图 3(b)所示. 可以看到: 在所有数据集上, 索引树的占用空间大小不超过 1 200 MB, 改进索引树的占用空间大小不超过 1 600 MB. 改进索引树的大小大于索引树的大小, 但不超过索引树的 1.6 倍. 因此, 两种索引可以被存储在现代计算机上. 从后面社区查询的实验结果可以看出: 使用改进索引树, 只需要付出略高一些的存储空间, 就可以获得搜索效率的巨大提升.

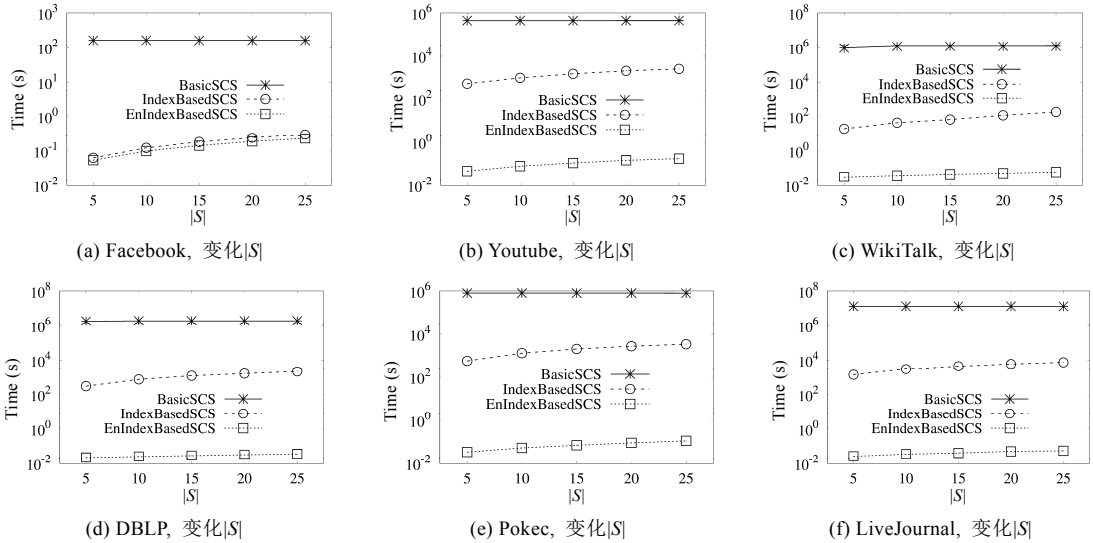


(a) 索引树和改进索引树的构建时间　　(b) 索引树和改进索引树的内存占用

图 3　索引树和改进索引树的构建时间和内存占用

2) 最稠密单社区查询算法的评估

实验评估 BasicSCS 算法、IndexBasedSCS 算法以及 EnIndexBasedSCS 算法的效率. 由于最稠密单社区查询只需要输入查询集合 S, 因此实验变化 S 的大小进行测试. 不同数据集下, 最稠密单社区查询算法的实验结果如图 4 所示.



(a) Facebook, 变化$|S|$　(b) Youtube, 变化$|S|$　(c) WikiTalk, 变化$|S|$

(d) DBLP, 变化$|S|$　(e) Pokec, 变化$|S|$　(f) LiveJournal, 变化$|S|$

图 4　最稠密单社区查询算法的效率

从图 4 可以看到: 在所有数据集上, 随着节点集合大小的增加, BasicSCS 算法、IndexBasedSCS 算法以及 EnIndexBasedSCS 算法的运行时间是递增的. 对于同一个查询节点集, BasicSCS 比 IndexBasedSCS 至少慢 2 个数量级, 比 EnIndexBasedSCS 至少慢 3 个数量级. 此外, 在小数据集 Facebook 上, EnIndexBasedSCS 比

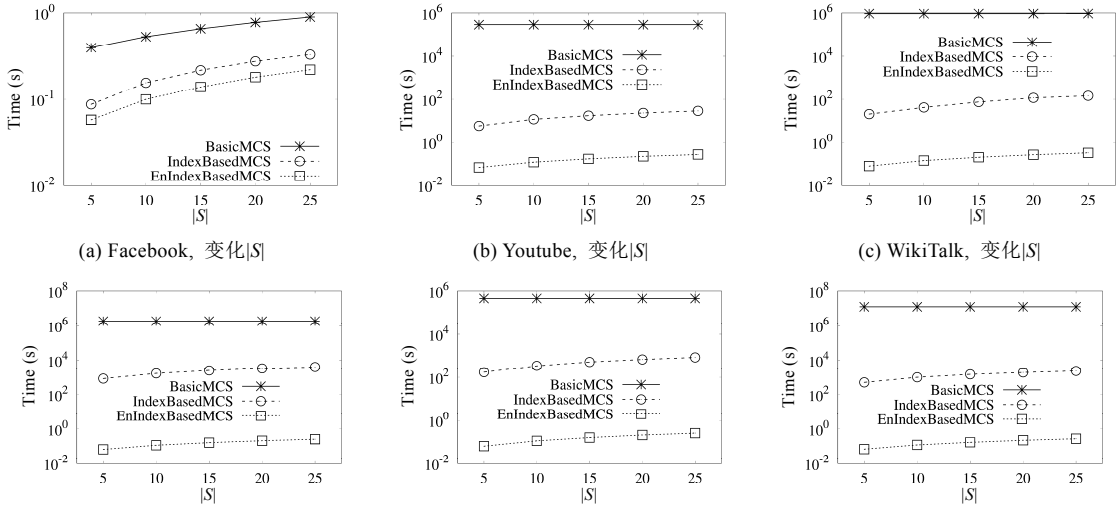
IndexBasedSCS 略快; 在其他数据集上, EnIndexBasedSCS 比 IndexBasedSCS 快至少 2 个数量级. 这是因为 EnIndexBasedSCS 算法基于改进的索引树, 在搜索最近公共祖先时比 IndexBasedSCS 算法更高效. 例如: 在 DBLP 上, 当集合大小为 10 时, BasicSCS 算法的查询时间为 1 933 810 s, 而基于索引树的 IndexBasedSCS 算法和基于改进索引树的 EnIndexBasedSCS 算法分别运行了 749.286 s 和 0.026 s. BasicSCS 算法分别比 IndexBasedSCS 算法和 EnIndexBasedSCS 算法慢了 3 个数量级和 7 个数量级, 而 IndexBasedSCS 算法比 EnIndexBasedSCS 算法慢 4 个数量级. 实验结果证实了基于改进索引树的算法比基于索引树的算法更能显著提升最稠密单社区查询的效率.

3) 基于稠密度阈值的多社区查询算法评估

实验测试 BasicMCS 算法、IndexBasedMCS 算法以及 EnIndexBasedMCS 算法的效率. 这 3 个算法需要输入查询集合 S 和稠密度阈值 θ, 我们分别变化这两个参数来进行实验. 不同数据集下, 多社区查询算法变化 S 大小和变化 θ 的实验结果分别如图 5 和图 6 所示.

从图 5 可以看到: 在所有数据集上, BasicMCS、IndexBasedMCS 以及 EnIndexBasedMCS 算法的运行时间随着节点集合大小的增加而增加. 对于同一个查询节点集, 在小数据集 Facebook 上, BasicMCS 算法的运行时间大约是 IndexBasedMCS 算法运行时间的 2–4 倍, 是 EnIndexBasedMCS 算法运行时间的 4–6 倍; 在其他数据集上, IndexBasedMCS 比 BasicMCS 至少快 3 个数量级, 而 EnIndexBasedMCS 比 IndexBasedMCS 至少快 1 个数量级. 例如: 在数据集 DBLP 上, 当查询集合大小为 10 时, EnIndexBasedMCS 算法的查询时间比 IndexBasedMCS 算法的查询时间大约低 3 个数量级, 比 BasicMCS 算法的查询时间大约低 6 个数量级. 实验结果证实了所提出的基于索引树的算法和基于改进索引树的算法的效率.

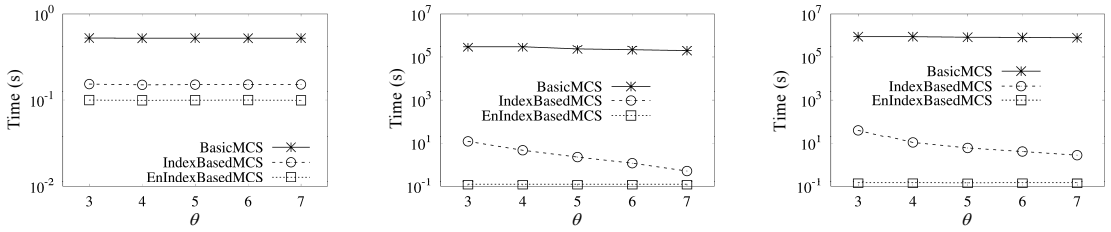
此外, 从图 6 可以看到: 在所有数据集上, BasicMCS、IndexBasedMCS 以及 EnIndexBasedMCS 算法的运行时间随着稠密度阈值 θ 的增加而降低. 这是因为越大的稠密度阈值所能找到的社区大小越小, 在查询过程中向祖先节点搜索的步数就越少. 对于同一个 θ, 在所有数据集上, BasicMCS 算法慢于 IndexBasedMCS 算法和 EnIndexBasedMCS 算法; 而 EnIndexBasedMCS 算法明显快于 IndexBasedMCS 算法. 例如: 在 DBLP 上, 当 θ=3 时, EnIndexBasedMCS 算法的查询时间比 IndexBasedMCS 算法的查询时间大约低 4 个数量级, 比 BasicMCS 算法的查询时间大约低 7 个数量级. 实验结果再次证实了基于索引树的算法和基于改进索引树的算法的效率.



(a) Facebook, 变化|S| (b) Youtube, 变化|S| (c) WikiTalk, 变化|S|

(d) DBLP, 变化|S| (e) Pokec, 变化|S| (f) LiveJournal, 变化|S|

图 5 基于稠密度阈值的多社区查询算法效率(变化|S|)



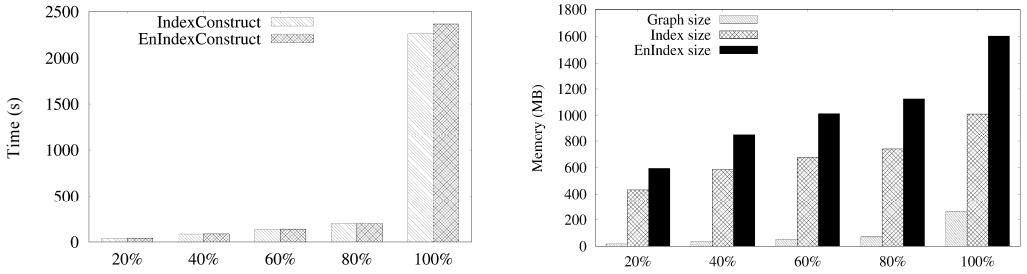
(a) Facebook, 变化θ (b) Youtube, 变化θ (c) WikiTalk, 变化θ

(d) DBLP, 变化θ (e) Pokec, 变化θ (f) LiveJournal, 变化θ

图 6 基于稠密度阈值的多社区查询算法效率(变化θ)

4) 可扩展性测试

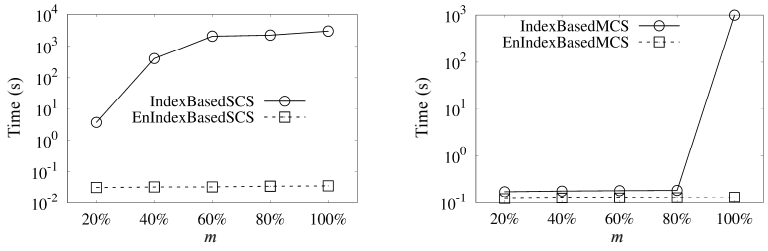
实验在每个数据集上随机选取 20%–80%的边生成 4 个子图, 并在这些子图上评估索引树和改进索引树的可扩展性. 图 7 展示了在 LiveJournal 不同子图上的索引树和改进索引树的实验结果, 其他数据集的结果类似. 可以看到: 随着 m 的增加, 两个索引的构建时间增加, 它们的占用空间大小也在增加. 对于同一个子图, IndexConstruct 算法构建索引树的时间略快于 EnIndexConstruct 算法构建改进索引树的时间; 改进索引树的空间占用高于索引树的空间占用, 两种索引都可以被存储在现代计算机上. 这些结论与之前的实验结果一致.



(a) LiveJournal, 变化 m (b) LiveJournal, 变化 m

图 7 LiveJournal 不同子图上的索引树和改进索引树的构建时间和内存占用

实验还评估了最稠密单社区查询算法和基于稠密度阈值的多社区查询算法的可扩展性. 图 8 展示了在 LiveJournal 上的实验结果, 其他数据集的结果类似.



(a) LiveJournal, 变化 m (b) LiveJournal, 变化 m

图 8 LiveJournal 不同子图上执行社区查询算法的运行时间

可以看到: 随着 m 的增加, EnIndexBasedSCS 算法和 EnIndexBasedMCS 算法的运行时间平稳增加, 而 IndexBasedSCS 算法和 IndexBasedMCS 算法的运行时间增加得更快. 同样, 在所有参数设置下, EnIndexBasedSCS 算法明显快于 IndexBasedSCS, EnIndexBasedMCS 算法明显快于 IndexBasedMCS, 这与之前实验的发现一致. 实验结果证实了基于改进索引树的 EnIndexBasedSCS 算法和 EnIndexBasedMCS 算法具有良好的可扩展性.

4.5 案例分析

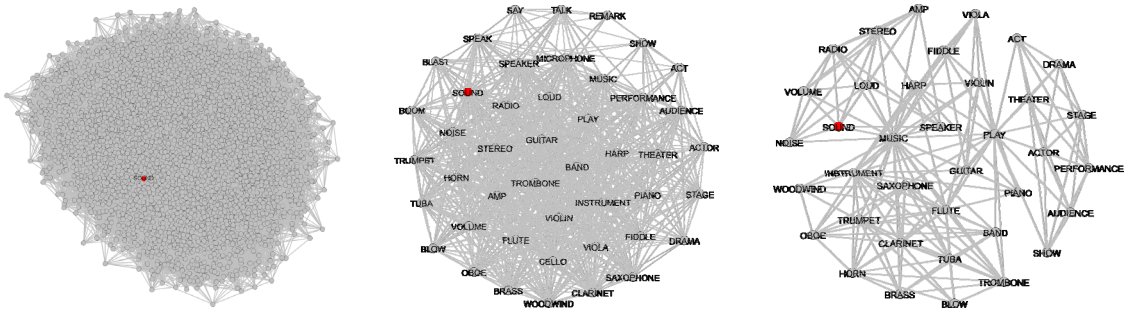
实验对单词关联网络 WordNet 进行了案例分析, 以评估基于邻域连通 k-核的社区模型的有效性. WordNet 包含 5 040 个节点和 55 258 条边, 其中, 节点表示单词; 边连接两个单词, 表示它们是相关的或强关联的. WordNet 可以从 http://w3.usf.edu/FreeAssociation/网站下载. 实验设置了 3 种不同的查询, 分别是包含某一单词的基于邻域连通 k-核的最稠密单社区、(max-core)社区以及(max-truss)社区.

图 9 展示了包含单词“DRUNK”的社区结果. 可以看到, (max-core)社区包含大量节点, 而最稠密单社区和(max-truss)社区包含的节点较少. 从单词含义的角度来看, 最稠密单社区和(max-truss)社区中的节点都与“DRUNK”关联紧密. 此外, 与(max-truss)社区相比, 最稠密单社区包含了额外的两个单词“GIN” (金酒)和“BRANDY” (白兰地). 这两个单词显然与“DRUNK”具有较强的关联. 因此, 基于邻域连通 k-核的社区能够找到包含“DRUNK”的完整社区结构. 图 10 展示了包含单词“SOUND”的社区. 同样地, (max-core)社区十分庞大, 最稠密单社区和(max-truss)社区具有较少的节点且都与“SOUND”词义接近. 最稠密单社区不仅包含了(max-truss)社区中的所有节点, 还增加了一些额外的节点, 如“MICROPHONE” (麦克风)、“CELLO” (大提琴)、“SPEAK” (说话)等, 因而更完整地表示了“SOUND”的社区结构. 图 11 为包含单词“POLITICS”的社区结果, 其中, 最稠密单社区和(max-truss)社区结构相同, 它们都具有较少的节点且词义与“POLITICS”高度相关, 如“POLITICIAN” (政治家)、“CONGRESS” (议院)、“SENATOR” (参议院)等. 图 12 为包含单词“MONEY”的社区. 可以看到: 最稠密单社区包含的节点较少, 节点词义与“MONEY”相关, 节点间连接更为稠密. 而(max-core)社区和(max-truss)社区都包含较多的节点. 包含单词“WOOD”的社区结构如图 13 所示, 其中, (max-core)社区、最稠密单社区和(max-truss)社区都包含较多的节点.

通过这些实验结果, 我们有以下发现: 当查询节点与其父亲之间的边稠密度较大时, 基于邻域连通 k-核的社区往往包含数量较少的关键节点, 同时, 这些节点间关联紧密, 如图 9–图 12; 而当节点与其父亲连边的稠密度较小时, 社区通常包含较多节点, 但仍然比(max-core)社区小, 如图 13. 当节点与其父亲的边稠密度较大且该节点的 max-truss 值也较大时, 基于邻域连通 k-核的社区模型相较于(max-truss)社区能够挖掘到更完整的结构, 从而提供更详细的社区信息(图 9、图 10). 总的来说, 这些实验结果表明: 基于邻域连通 k-核的社区模型在节点边稠密度较大的情况下具有优势, 能够更好地捕捉到社区的稠密性和完整性.

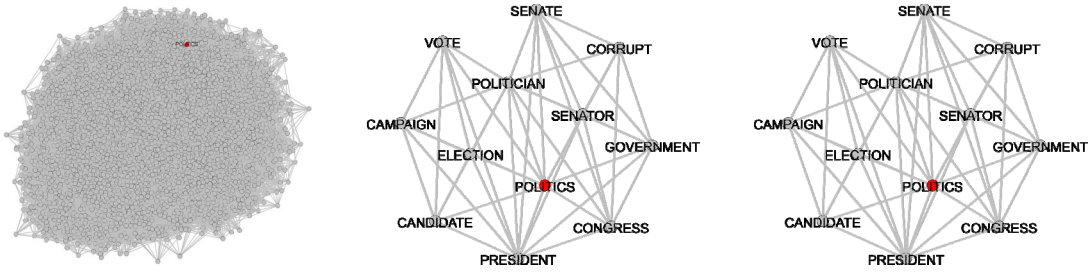
(a) (max-core)社区 (b) 最稠密单社区 (c) (max-truss)社区

图 9 单词“DRUNK”的社区结果



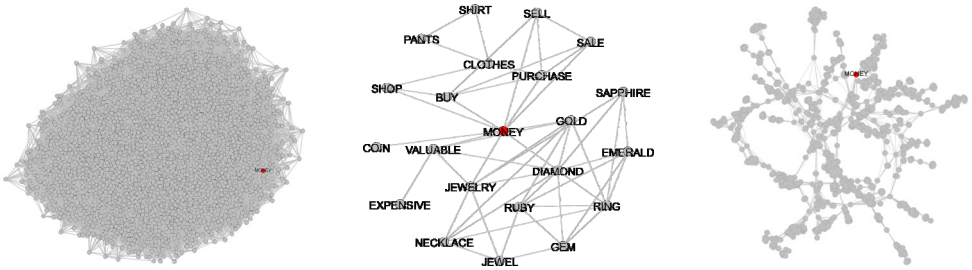
(a) (max-core)社区　(b) 最稠密单社区　(c) (max-truss)社区

图 10　单词“SOUND”的社区结果



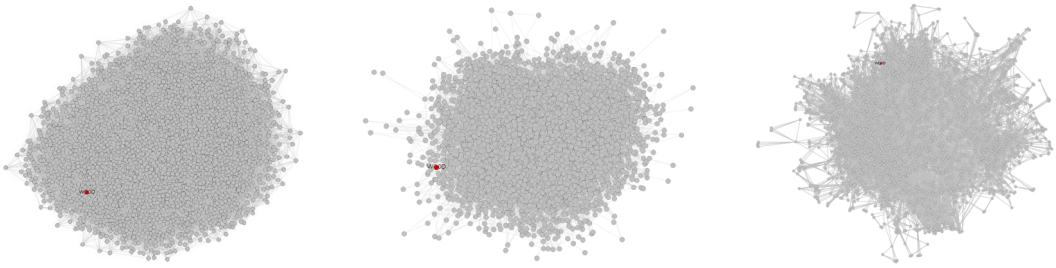
(a) (max-core)社区　(b) 最稠密单社区　(c) (max-truss)社区

图 11　单词“POLITICS”的社区结果



(a) (max-core)社区　(b) 最稠密单社区　(c) (max-truss)社区

图 12　单词“MONEY”的社区结果

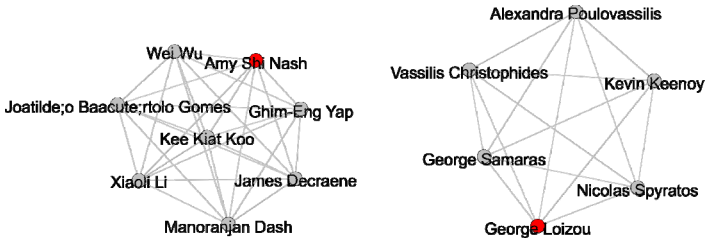


(a) (max-core)社区　(b) 最稠密单社区　(c) (max-truss)社区

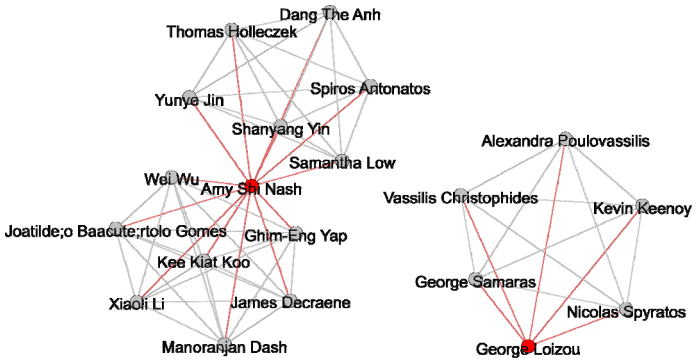
图 13　单词“WOOD”的社区结果

此外, 实验对科学协作网络 DBLP 进行了案例分析. 我们提取了 DBLP 中在数据库和数据挖掘相关会议上发表过至少 1 篇论文的作者及他们之间的合作关系, 共包含 37 177 个节点和 131 715 条边. 实验设置 k=6,

并查询包含 Amy Shi Nash 和 George Loizou 的基于邻域连通 k-核的社区和 k-truss 社区. 实验结果如图 14 所示. 可以看到, k-truss 社区和基于邻域连通 k-核的社区模型均能将两个点分在不同的社区里, 而基于邻域连通 k-核的社区较好地保留了关键节点 Amy Shi Nash 的完整社区结构. 这些结果证实了基于邻域连通 k-核的社区模型的有效性.



(a) 基于 k-truss 模型的社区查询结果



(b) 基于邻域连通 k-核的社区查询结果

图 14 DBLP 的案例分析

5 总 结

本文考虑节点的邻域结构信息, 提出了邻域连通 k-核模型来刻画社区结构. 基于该模型, 研究了两个社区查询问题: 最稠密单社区查询和基于稠密度阈值的多社区查询. 为了解决这两个查询问题, 提出了边稠密度的概念并开发了相应的基线算法. 为了提高查询效率, 设计了索引树和改进索引树来高效支撑最稠密单社区查询和基于稠密度阈值的多社区查询. 我们在 6 个真实数据集上进行了实验, 证明了基于索引树和改进索引树的查询算法在两种社区查询问题上都能提供高效的性能; 我们在 WordNet 和 DBLP 上的案例分析, 证明了基于邻域 k-核的社区模型的有效性.

我们相信, 本文的研究工作势必能为更精准的社区查询做出贡献. 本文提出的社区模型将节点的邻域信息融入经典的 k-核稠密子图中. 在未来, 我们将探索更多的稠密子图模型, 如 k-truss、k-ecc 等, 以进行更广泛的邻域社区模型研究. 我们相信, 通过这一手段, 可以获得更多有趣的社区.

References:

[1] Watts DJ, Strogatz SH. Collective dynamics of ‘small-world’ networks. Nature, 1998, 393(6684): 440–442.

[2] Broder A, Kumar R, Maghoul F, *et al*. Graph structure in the Web. Computer Networks, 2000, 33(1–6): 309–320.

[3] Girvan M, Newman MEJ. Community structure in social and biological networks. National Academy of Sciences, 2002, 99(12): 7821–7826.

[4] Huang X, Lakshmanan LVS, Yu JX, *et al*. Approximate closest community search in networks. Proc. of the VLDB Endowment, 2015, 9(4): 276−287.

[5] Cui W, Xiao Y, Wang H, *et al*. Online search of overlapping communities. In: Proc. of the SIGMOD. New York: ACM, 2013. 277−288.

[6] Huang X, Lakshmanan LVS, Xu J. Community search over big graphs: Models, algorithms, and opportunities. In: Proc. of the ICDE. Piscataway: IEEE, 2017. 1451−1454.

[7] Ahn YY, Bagrow JP, Lehmann S. Link communities reveal multiscale complexity in networks. Nature, 2010, 466(7307): 761−764.

[8] Huang X, Cheng H, Qin L, *et al*. Querying *k*-truss community in large and dynamic graphs. In: Proc. of the SIGMOD. New York: ACM, 2014. 1311−1322.

[9] Palla G, Derényi I, Farkas I, *et al*. Uncovering the overlapping community structure of complex networks in nature and society. Nature, 2005, 435(7043): 814−818.

[10] Seidman SB. Network structure and minimum degree. Social Networks, 1983, 5(3): 269−287.

[11] Batagelj V, Zaversnik M. An $O(m)$ algorithm for cores decomposition of networks. arXiv:cs/0310049v1, 2003.

[12] Cheng J, Ke Y, Chu S, *et al*. Efficient core decomposition in massive networks. In: Proc. of the ICDE. Piscataway: IEEE, 2011. 51−62.

[13] Wang J, Cheng J. Truss decomposition in massive networks. Proc. of the VLDB Endowment, 2012, 5(9): 812−823.

[14] Akbas E, Zhao P. Truss-based community search: A truss-equivalence based indexing approach. Proc. of the VLDB Endowment, 2017, 10(11): 1298−1309.

[15] Cohen J. Trusses: Cohesive subgraphs for social network analysis. Technical Report, National Security Agency, 2008.

[16] Bron C, Kerbosch J. Algorithm 457: Finding all cliques of an undirected graph. Communications of the ACM, 1973, 16(9): 575−577.

[17] Cheng J, Ke Y, Fu AWC, *et al*. Finding maximal cliques in massive networks by h^*-graph. In: Proc. of the SIGMOD. New York: ACM, 2010. 447−458.

[18] Wang J, Cheng J, Fu AWC. Redundancy-aware maximal cliques. In: Proc. of the KDD. New York: ACM, 2013. 122−130.

[19] Yuan L, Qin L, Zhang W, *et al*. Index-based densest clique percolation community search in networks. IEEE Trans. on Knowledge and Data Engineering, 2017, 30(5): 922−935.

[20] Sozio M, Gionis A. The community-search problem and how to plan a successful cocktail party. In: Proc. of the KDD. New York: ACM, 2010. 939−948.

[21] Cui W, Xiao Y, Wang H, *et al*. Local search of communities in large graphs. In: Proc. of the SIGMOD. New York: ACM, 2014. 991−1002.

[22] Barbieri N, Bonchi F, Galimberti E, *et al*. Efficient and effective community search. Data Mining and Knowledge Discovery, 2015, 29: 1406−1433.

[23] Fortunato S. Community detection in graphs. Physics Reports, 2010, 486(3-5): 75−174.

[24] Tyler JR, Wilkinson DM, Huberman BA. Email as spectroscopy: Automated discovery of community structure within organizations. In: Proc. of the C&T. Berlin: Springer, 2003. 81−96.

[25] Gregory S. Local betweenness for finding communities in networks. Technical Report, University of Bristol, 2008.

[26] Radicchi F, Castellano C, Cecconi F, *et al*. Defining and identifying communities in networks. National Academy of Sciences, 2004, 101(9): 2658−2663.

[27] Jin D, Liu J, Jia ZX, *et al*. *k*-nearest-neighbor network based data clustering algorithm. Pattern Recognition and Artificial Intelligence, 2010, 23(4): 546−551 (in Chinese with English abstract).

[28] Newman MEJ, Girvan M. Finding and evaluating community structure in networks. Physical Review E, 2004, 69(2): No. 026113.

[29] Brandes U, Delling D, Gaertler M, *et al*. On modularity clustering. IEEE Trans. on Knowledge and Data Engineering, 2007, 20(2): 172−188.

[30] Clauset A, Newman MEJ, Moore C. Finding community structure in very large networks. Physical Review E, 2004, 70(6): No. 066111.

[31] Guimera R, Amaral LAN. Functional cartography of complex metabolic networks. Nature, 2005, 433(7028): 895−900.

[32] Ye Z, Hu S, Yu J. Adaptive clustering algorithm for community detection in complex networks. Physical Review E, 2008, 78(4): No. 046115.

[33] Newman MEJ. Spectral methods for community detection and graph partitioning. Physical Review E, 2013, 88(4): No. 042822.

[34] Li Y, Sha C, Huang X, *et al*. Community detection in attributed graphs: An embedding approach. In: Proc. of the AAAI. Menlo Park: AAAI, 2018. 338−345.

[35] Cai XY, Dai GZ, Yang LB. Survey on spectral clustering algorithms. Computer Science, 2008, 35(7): 14−18 (in Chinese with English abstract).

[36] Holland PW, Laskey KB, Leinhardt S. Stochastic blockmodels: First steps. Social Networks, 1983, 5(2): 109−137.

[37] Airoldi EM, Blei D, Fienberg S, *et al*. Mixed membership stochastic blockmodels. Advances in Neural Information Processing Systems, 2008, 9: 1981−2014.

[38] Pal S, Coates M. Scalable MCMC in degree corrected stochastic block model. In: Proc. of the ICASSP. Piscataway: IEEE, 2019. 5461−5465.

[39] Karrer B, Newman MEJ. Stochastic blockmodels and community structure in networks. Physical Review E, 2011, 83(1): No. 016107.

[40] Chen K, Lei J. Network cross-validation for determining the number of communities in network data. Journal of the American Statistical Association, 2018, 113(521): 241−251.

[41] Xing EP, Fu W, Song L. A state-space mixed membership blockmodel for dynamic network tomography. Applied Statistics, 2010, 4(2): 535−566.

[42] Xu K. Stochastic block transition models for dynamic networks. In: Proc. of the AISTATS. 2015. 1079−1087.

[43] Latouche P, Birmelé E, Ambroise C. Overlapping stochastic block models with application to the French political blogosphere. Applied Statistics, 2011, 5(1): 309−336.

[44] Shi X, Lu H, He Y, *et al*. Community detection in social network with pairwisely constrained symmetric non-negative matrix factorization. In: Proc. of the ASONAM. 2015. 541−546.

[45] Sun BJ, Shen H, Gao J, *et al*. A non-negative symmetric encoder-decoder approach for community detection. In: Proc. of the CIKM. New York: ACM, 2017. 597−606.

[46] Yang J, Leskovec J. Overlapping community detection at scale: A nonnegative matrix factorization approach. In: Proc. of the WSDM. New York: ACM, 2013. 587−596.

[47] He D, You X, Feng Z, *et al*. A network-specific Markov random field approach to community detection. In: Proc. of the AAAI. Menlo Park: AAAI, 2018. 306−313.

[48] He D, Song W, Jin D, *et al*. An end-to-end community detection model: Integrating LDA into Markov random field via factor graph. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2019. 5730−5736.

[49] Jin D, Ge M, Li Z, *et al*. Using deep learning for community discovery in social networks. In: Proc. of the ICTAI. Piscataway: IEEE, 2017. 160−167.

[50] Xie Y, Wang X, Jiang D, *et al*. High-performance community detection in social networks using a deep transitive autoencoder. Information Sciences, 2019, 493: 75−90.

[51] Tian F, Gao B, Cui Q, *et al*. Learning deep representations for graph clustering. In: Proc. of the AAAI. Menlo Park: AAAI, 2014. 1293−1299.

[52] Bhatia V, Rani R. DFuzzy: A deep learning-based fuzzy clustering model for large graphs. Knowledge and Information Systems, 2018, 57: 159−181.

[53] Xu R, Che Y, Wang X, *et al*. Stacked autoencoder-based community detection method via an ensemble clustering framework. Information Sciences, 2020, 526: 151−165.

[54] Choong JJ, Liu X, Murata T. Learning community structure with variational autoencoder. In: Proc. of the ICDM. Piscataway: IEEE, 2018. 69−78.

[55] Sarkar A, Mehta N, Rai P. Graph representation learning via ladder Gamma variational autoencoders. In: Proc. of the AAAI. Menlo Park: AAAI, 2020. 5604−5611.

[56] Jin D, Li B, Jiao P, *et al*. Network-specific variational auto-encoder for embedding in attribute networks. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2019. 2663−2669.

[57] Yang Z, Tang J, Li J, *et al*. Social community analysis via a factor graph model. IEEE Intelligent Systems, 2011, 26(3): 58−65.

[58] Jia Y, Gao Y, Yang W, *et al*. A novel ego-centered academic community detection approach via factor graph model. In: Proc. of the IDEAL. 2014. 223−230.

[59] Passarella A, Dunbar RIM, Conti M, *et al*. Ego network models for future Internet social networking environments. Computer Communications, 2012, 35(18): 2201−2217.

[60] Yang L, Cao X, He D, *et al*. Modularity based community detection with deep learning. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2016. 2252−2258.

[61] Cao J, Jin D, Dang J. Autoencoder based community detection with adaptive integration of network topology and node contents. In: Proc. of the KSEM. Berlin: Springer, 2018. 184−196.

[62] Bhatia V, Rani R. A distributed overlapping community detection model for large graphs using autoencoder. Future Generation Computer Systems, 2019, 94: 16−26.

[63] Sun H, He F, Huang J, *et al*. Network embedding for community detection in attributed networks. ACM Trans. on Knowledge Discovery from Data, 2020, 14(3): 1−25.

[64] Wang C, Pan S, Long G, *et al*. MGAE: Marginalized graph autoencoder for graph clustering. In: Proc. of the CIKM. New York: ACM, 2017. 889−898.

[65] Kingma DP, Welling M. Auto-encoding variational Bayes. arXiv:1312.6114, 2013.

[66] Jia Y, Zhang Q, Zhang W, *et al*. CommunityGAN: Community detection with generative adversarial nets. In: Proc. of the WWW. New York: ACM, 2019. 784−794.

[67] Zhang Y, Xiong Y, Ye Y, *et al*. SEAL: Learning heuristics for community detection with generative adversarial networks. In: Proc. of the KDD. New York: ACM, 2020. 1103−1113.

[68] Tao Z, Liu H, Li J, *et al*. Adversarial graph embedding for ensemble clustering. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2019. 3562−3568.

[69] Sun Y, Wang S, Hsieh TY, *et al*. Megan: A generative adversarial network for multi-view network embedding. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2019. 3527−3533.

[70] Gao H, Pei J, Huang H. ProGAN: Network embedding via proximity generative adversarial network. In: Proc. of the KDD. New York: ACM, 2019. 1308−1316.

[71] Hong H, Li X, Wang M. GANE: A generative adversarial network embedding. IEEE Trans. on Neural Networks and Learning Systems, 2019, 31(7): 2325−2335.

[72] He D, Zhai L, Li Z, *et al*. Adversarial mutual information learning for network embedding. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2020. 3321−3327.

[73] Wu Z, Pan S, Chen F, *et al*. A comprehensive survey on graph neural networks. IEEE Trans. on Neural Networks and Learning Systems, 2020, 32(1): 4−24.

[74] Jin D, Li B, Jiao P, *et al*. Community detection via joint graph convolutional network embedding in attribute network. In: Proc. of the ICANN. Berlin: Springer, 2019. 594−606.

[75] He D, Song Y, Jin D, *et al*. Community-centric graph convolutional network for unsupervised community detection. In: Proc. of the IJCAI. San Francisco: Morgan Kaufmann Publishers, 2021. 3515−3521.

[76] Zheng Y, Chen S, Zhang X, *et al*. Heterogeneous graph convolutional networks for temporal community detection. arXiv:1909.10248, 2019.

[77] Qu M, Bengio Y, Tang J. GMNN: Graph Markov neural networks. In: Proc. of the ICML. New York: ACM, 2019. 5241−5250.

[78] Bornmann L, Daniel HD. What do we know about the h index? Journal of the American Society for Information Science and Technology, 2007, 58(9): 1381−1385.

[79] Hirsch JE. Does the *h* index have predictive power? National Academy of Sciences, 2007, 104(49): 19193−19198.

[80] Alonso S, Cabrerizo FJ, Herrera-Viedma E, *et al*. *h*-index: A review focused in its variants, computation and standardization for different scientific fields. Journal of Informetrics, 2009, 3(4): 273−289.

[81] Epasto A, Lattanzi S, Mirrokni V, *et al*. Ego-net community mining applied to friend suggestion. Proc. of the VLDB Endowment, 2015, 9(4): 324−335.

[82] Chiba N, Nishizeki T. Arboricity and subgraph listing algorithms. SIAM Journal on Computing, 1985, 14(1): 210−223.

[83] Itai A, Rodeh M. Finding a minimum circuit in a graph. In: Proc. of the STOC. New York: ACM, 1977. 1−10.

[84] Latapy M. Main-memory triangle computations for very large (sparse (power-law)) graphs. Theoretical Computer Science, 2008, 407(1-3): 458−473.

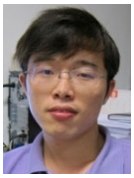
附中文参考文献:

[27] 金弟, 刘杰, 贾正雪, 等. 基于 k 最近邻网络的数据聚类算法. 模式识别与人工智能, 2010, 23(4): 546−551.

[35] 蔡晓妍, 戴冠中, 杨黎斌. 谱聚类算法综述. 计算机科学, 2008, 35(7): 14−18.

张琦(1996－), 女, 博士, CCF 专业会员, 主要研究领域为大规模图数据管理与挖掘, 图计算.

程苗苗(1998－), 女, 硕士, 主要研究领域为大规模图计算系统, 图算法.



李荣华(1985－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为大规模图数据管理与挖掘, 社交网络分析与挖掘.

王国仁(1966－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为图数据库, 图数据挖掘, 大数据系统.