软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(6):2900−2926 [doi: 10.13328/j.cnki.jos.007243] [CSTR: 32375.14.jos.007243]

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

动态信息网中持续扩展 *k*-truss 社区序列查找算法*

王芯蕊[1], 姚 越[1], 于东晓[1], 高 宏[2], 成秀珍[1]

[1](山东大学 计算机科学与技术学院, 山东 青岛 266237)

[2](浙江师范大学 计算机科学与技术学院, 浙江 金华 321004)

通信作者: 于东晓, E-mail: dxyu@sdu.edu.cn

摘 要: 动态信息网 (DIN) 包含了真实世界中随时间推移不断发生变化的对象以及对象间的联系, 常常被刻画为一系列静态无向图快照. 社区, 由信息网中一些内部联系紧密的对象组成. 动态信息网中常常存在这样的社区: 在一段时间内, 随着时间的推移, 社区成员规模不断扩大, 并且社区内部成员间始终保持紧密的联系. 这样的社区在相应时间段内的演化轨迹在动态信息网的多张图快照上形成了一个社区序列, 称为持续扩展社区序列. 在动态信息网中查找持续扩展社区序列有重要的实用价值, 但是以前的工作并未对此进行研究. 结合集合的包含关系和三角连通 k-truss 模型, 提出动态信息网中基于查询点 q 的持续扩展社区序列 (qLEC) 模型, 设计了一个正向计算社区候选顶点集-反向回溯查找社区序列的持续扩展社区序列两阶段查找算法, 并给出基于提早终止策略的时间优化和基于 TCP 索引压缩技术的空间优化方法. 通过充分的实验证明: 相比于现有动态社区模型, qLEC 模型具有特定的实际意义; 两阶段查找算法能够有效找到 qLEC 模型所刻画的持续扩展社区序列; 优化策略显著降低了两阶段查找算法的时间和空间开销.

关键词: 动态图; 三角连通 *k*-truss; 持续扩展社区序列; 基于 DFS 的回溯算法; 剪枝

中图法分类号: TP393

中文引用格式: 王芯蕊, 姚越, 于东晓, 高宏, 成秀珍. 动态信息网中持续扩展*k*-truss社区序列查找算法. 软件学报, 2025, 36(6): 2900–2926. http://www.jos.org.cn/1000-9825/7243.htm

英文引用格式: Wang XR, Yao Y, Yu DX, Gao H, Cheng XZ. Lasting Enlarging *k*-truss Community Sequence Search in Dynamic Information Networks. Ruan Jian Xue Bao/Journal of Software, 2025, 36(6): 2900–2926 (in Chinese). http://www.jos.org.cn/1000-9825/7243.htm

Lasting Enlarging *k*-truss Community Sequence Search in Dynamic Information Networks

WANG Xin-Rui[1], YAO Yue[1], YU Dong-Xiao[1], GAO Hong[2], CHENG Xiu-Zhen[1]

[1](School of Computer Science and Technology, Shandong University, Qingdao 266237, China)

[2](School of Computer Science and Technology, Zhejiang Normal University, Jinhua 321004, China)

Abstract: Dynamic information networks (DIN), which contain evolving objects in the real world and the links among them, are often modeled as a series of static undirected graph snapshots. A community consists of a group of well-connected objects in an information network. In a DIN, there is often a community whose size increases over time but its members always keep well-connected during that period of time. The evolving trajectory of such a community over time forms a sequence of the community on several snapshots of the DIN, which is termed a lasting enlarging community sequence in this study. It is meaningful to search for lasting enlarging community sequences in a DIN. However, no previous research has paid attention to such community sequences. This study formally defines the q-based lasting enlarging community sequence (qLEC) in a DIN by combining set inclusion with the triangle-connected k-truss model. A two-phase search algorithm is developed, which includes computing candidate vertex sets of communities from the beginning to the end of the time window and performing community sequence search from the end to the beginning of the time window. This study also provides

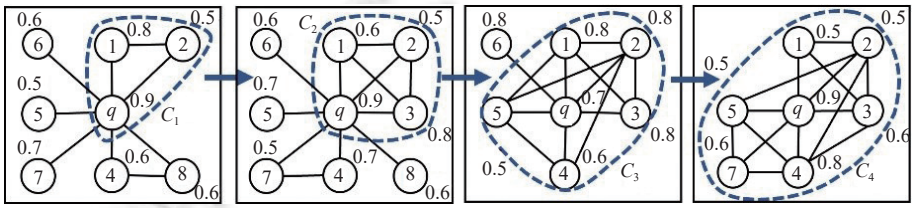
* 基金项目: 国家自然科学基金 (62202277, U22A2025)
收稿时间: 2023-09-11; 修改时间: 2024-01-12, 2024-03-13; 采用时间: 2024-06-22; jos 在线出版时间: 2024-10-12
CNKI 网络首发时间: 2024-10-14

optimization strategies based on early termination and TCP index compression to reduce time and space costs. Sufficient experiments demonstrate that the qLEC model has specific practical significance compared to existing dynamic community models. The two-phase search algorithm effectively finds qLEC-based lasting enlarging community sequences. The proposed optimization strategies significantly reduce the spatiotemporal cost of the two-phase algorithm.

Key words: dynamic graph; triangle-connected k-truss; lasting enlarging community sequence; DFS-based backtracking algorithm; pruning

信息网, 由真实世界中众多的对象以及对象之间的联系组成, 作为一种常见的数据组织形式, 在社交网络、移动应用、文献数据库等众多领域广泛存在. 例如, 在中国知网、DBLP 等文献信息网中, 作者、会议、论文都是对象, 作者发表论文、会议收录论文都是对象之间产生的联系. 信息网中的对象本身常常还具有多种属性信息, 如 DBLP 信息网中, 作者的属性信息包括学术影响力、发表的论文总数、论文被引量等.

信息网中的对象和联系常常随着时间的推移不断发生变化, 称这样的信息网为动态信息网. 如在文献信息网中, 作者不断发表新的论文, 会议和期刊不断收录新的论文, 学生作者毕业后不再继续写作论文等. 动态信息网能够表达对象丰富复杂的结构、语义、时序等信息, 近年来受到广泛关注[1–5]. 研究者常常将动态信息网刻画成一系列静态信息网的快照, 每个快照建模成一张加权无向图, 记录了在一个时刻或一段时间内出现的对象及对象间的联系, 以及对象的属性信息等. 例如, 图 1 给出了一个 DBLP 动态信息网的示例, 包含 2001–2004 年的 4 张快照 $G_{2001},G_{2002},G_{2003},G_{2004}$, 每张快照的每个顶点表示一个作者, 每条边表示作者之间在该快照对应的年份合作了至少一篇论文, 顶点旁边的权值表示该点在信息网中的重要性或影响力得分 (如度数、PageRank 值[6]等).



(a) G_{2001} (b) G_{2002} (c) G_{2003} (d) G_{2004}

图 1 动态信息网窗口 $G_{2001}-G_{2004}$ 上包含 q 的持续扩展社区序列示例

社区由信息网中一些内部紧密联系的对象组成, 如文献信息网中的科研团队、社交网中的兴趣小组等. 社区搜索是给定查询条件, 找到信息网中符合相应条件的社区, 具有重要的实际意义, 近年来被广泛研究. 根据查询条件的不同, 研究者们在各自的工作中提出了各种静态社区模型, 如基于 k-核 (该子图内的每个顶点至少有 k 个邻居) 的社区[7–10]、基于 k-truss (该子图中的每条边至少被包含在这个子图的 (k–2) 个三角形中) 的社区[11]、基于加权最稠密子图的社区[12]. 目前关于动态信息网社区搜索的工作, 一部分是考虑如何设计在线算法或增量式算法快速计算出每个当前时刻最新的社区搜索结果[13], 但是并不关注社区在一段时间内的动态演化特征; 另一部分则是结合社区的动态演化特征给出不同的查询条件, 然后搜索相应的社区, 如发现时间上周期性出现的团社区[14], 发现某时刻突然出现并持续存在一段时间的稠密子图社区 Bursting Core 等.

动态信息网中常常存在这样的社区: 在一段时间内, 随着时间的推移, 社区成员规模不断扩大, 并且社区内部成员间始终保持紧密的联系. 这样的社区在相应时间段内的演化轨迹在动态信息网的多张时间快照上形成了一个社区序列, 本文称这样的社区序列为持续扩展社区序列. 如图 1 所示, 从 2001–2004 年, 社区 $C_1=(q,1,2)$ 不断发展壮大到 $C_2=(q,1,2,3)$, 再到 $C_3=(q,1,2,3,4,5)$, 最后到 $C_4=(q,1,2,3,4,5,7)$, 并且这 4 年间 C_1、C_2、C_3、C_4 的内部成员始终紧密联系. 称 (C_1,C_2,C_3,C_4) 是一个持续扩展社区序列.

在动态信息网中查找持续扩展社区序列具有广泛的应用价值. 比如在文献信息网中, 由一些高影响力学术带头人领衔、优秀青年学者不断加入的科研团队, 多年来持续大量产出科研成果. 查找并招募这样的科研团队常常能够很好地攻坚克难、完成重大科研项目. 在社交网中，一些兴趣小组在数周或者数月内, 不断扩张小组规模, 并且一直保持着频繁的互动, 如评论、转发、点赞彼此的帖子等. 发现这样的兴趣小组，可以帮助宣传推广有益信息, 及时监控和预防非法聚集等活动.

尽管查找动态信息网中的持续扩展社区序列非常重要, 但是目前还没有相关的研究工作. 本文关注动态信息网中持续扩展社区序列的查找问题. 在实际中, 动态信息网中常常包含非常多的持续扩展社区序列, 每个持续扩展社区序列都表示一个联系紧密的顶点集在一段时间内的发展趋势. 关注一些不重要的顶点集的发展趋势是对资源的浪费, 因此可以只聚焦于特定查询点所在社区的发展趋势. 例如, 对于一个具有重要学术影响力的作者, 找到一段时间内包含该作者的持续扩展社区序列, 可以知道这段时间内该作者的核心团队成员有哪些, 新合作的成员有哪些. 此外, 由于对象在动态信息网的每张快照上都有表示其在网络中重要性或影响力的权值, 进而每个社区、每个社区序列都会有相应的权值, 用于衡量该社区或社区序列包含的所有对象的综合重要性. 例如, 在 DBLP 动态信息网中, 如果每个作者的重要性得分表示了其学术影响力, 那么一个持续扩展社区序列权值越大, 就说明该序列中包含的作者的综合学术影响力越高, 具有更重要的实际意义. 于是, 本文提出并研究动态信息网基于查询点 q 的持续扩展社区序列查找问题, 要求找到的持续扩展社区序列中的每个社区都要包含给定的查询点 q, 并且优先输出权值高的社区序列. 研究该问题的难点主要在于以下两个方面.

(1) 如何建立合理的模型形式化地定义动态信息网中基于查询点 q 的持续扩展社区序列?

(2) 如何设计高效的算法查找动态信息网中基于查询点 q 的持续扩展社区序列?

为了解决难点 (1), 经过分析可知, 动态信息网中基于查询点 q 的持续扩展社区序列主要有两个特征: 一是在一段时间内社区规模不断扩大; 二是这段时间内社区成员内部保持紧密联系. 为此, 本文首先采用窗口模型存储动态信息网一段时间的所有快照信息. 然后, 对于特征 1, 引入支持度阈值 s, 定义持续扩展社区序列包含 s 个社区, 这些社区分别存在于窗口中不同的 s 张快照上 (不要求 s 张快照对应的时间连续, 亦即 s 张快照在窗口中不必相邻), 并且社区间满足包含关系. 例如在图 1 中, 窗口包含 4 张快照, 若 (C_1,C_2,C_3,C_4) 是一个满足 s=4 的持续扩展社区序列, 则 C_1、C_2、C_3、C_4 分别存在于 4 张不同的快照上, 且 $C_1 \subseteq C_2 \subseteq C_3 \subseteq C_4$. 接下来对于特征 2, 引入参数 k, 本文采用三角连通 k-truss 模型定义社区[2], 这是因为 k-truss 社区中的成员间不仅联系紧密 (每个成员都与至少 (k–1) 个其他成员之间有直接边联系), 而且成员间的边联系很稳固 (每条边都包含在至少 (k–2) 个三角形中, 三角形具有稳定性). 例如在图 1 中, 若 k=3, 则持续扩展社区序列 (C_1,C_2,C_3,C_4) 中社区 C_1,C_2,C_3和C_4 都是三角连通 3-truss. 显然, 本文的研究工作很容易迁移运用到采用团、准团、k-core 等模型定义持续扩展社区序列中的社区的情形, 本文只是以同时考虑了社区成员间边联系的紧密型和稳固性两个特点的三角连通 k-truss 模型为代表率先进行探索研究.

为了解决难点 (2), 对于本文提出的动态信息网中基于查询点 q 的持续扩展社区序列模型, 本文设计了“正向计算社区候选顶点集-反向回溯查找社区序列”的两阶段算法: 首先从窗口起始到末尾, 正向计算每张快照上可能成为最终持续扩展社区序列中社区的所有候选顶点集, 同时记录所有可能成为持续扩展社区序列终点 (最后一个社区) 的侯选终点社区; 然后从窗口末尾到起始, 从每一个候选终点社区 C 出发, 反向回溯查找所有以 C 为终点的持续扩展社区序列. 显然, 正向计算阶段所得的每张快照上的社区候选顶点集越多, 反向回溯查找阶段的搜索空间就越大, 所需查找时间也就越长. 因此, 本文提出两个提早终止策略来减少反向回溯查找过程的时间开销. 同时, 还提出有效的 TCP 索引压缩存储技术来减少算法的内存开销.

本文的主要贡献总结如下.

(1) 提出了动态信息网中基于查询点 q 的持续扩展社区序列 (q-based lasting enlarging community sequence, qLEC) 模型, 该模型利用集合的包含关系和三角连通 k-truss 模型, 有效刻画了持续扩展社区序列在一段时间内社区规模不断扩大、社区成员内部保持紧密联系的 2 个关键特征.

(2) 为了找到动态信息网一个窗口内所有的 qLEC, 提出正向计算社区候选顶点集-反向回溯查找社区序列的持续扩展社区序列两阶段查找算法.

(3) 为了进一步减少持续扩展社区序列两阶段查找算法的时间和空间开销, 分别设计了基于提早终止策略的时间优化和基于 TCP 索引压缩技术的空间优化方法.

(4) 在多个真实世界的动态信息网上进行了大量实验, 评测了本文所提算法的性能, 验证了时空优化方法的有

效性, 并通过实际案例分析说明了本文所提 qLEC 模型的有效性以及查找持续扩展社区序列的重要实用价值.

本文第 1 节总结现有社区查询的相关研究工作. 第 2 节介绍动态信信息网和三角连通 k-truss 模型的相关概念, 并给出持续扩展社区序列模型及其查找问题的具体定义. 第 3 节设计正向计算社区候选顶点集-反向回溯查找社区序列的持续扩展社区序列两阶段查找算法, 并分析算法复杂性. 第 4 节提出基于提早终止策略的时间优化和基于 TCP 索引压缩技术的空间优化方法. 第 5 节通过在多个真实世界的动态信息网上的实验, 评估算法性能和模型的有效性. 第 6 节总结全文, 并探讨未来研究工作.

1 相关工作

与本文研究的动态信息网持续扩展社区序列查找问题最相关的工作是社区搜索[15,16]: 给定查询条件, 找到网络中符合相应查询条件的社区. 下面先对静态图网络社区搜索和动态图网络社区搜索的相关工作进行总结, 然后介绍一些其他的相关工作.

● 静态图网络社区搜索

在静态图网络中, 根据不同的用户需求, 研究者们设定不同的查询条件, 提出各种社区模型并设计相应算法在网络中对其进行搜索. 如基于 k-核 (该子图内的每个顶点至少有 k 个邻居) 的社区[17]、基于 k-truss (该子图中的每条边至少被包含在这个子图的 (k–2) 个三角形中) 的社区[7,18]、基于准团的社区[19]、基于加权最稠密子图的社区[20]、具有影响力的社区[2,15]、核心社区[21], 等等. 由于 k-核模型中每个顶点的核数可以用来度量用户在社区中的参与度[22] (用户参与度表示一个用户参与或被鼓励参与到一个社区时的热衷程度, 近年来被大量研究[23–25]), 所以基于 k-核的社区模型在实际中应用广泛, 并且被进一步扩展定义出许多新的社区模型. 如, 文献 [15] 采用候选剪枝、早期终止和极大化检查等技术设计了有效的算法来枚举极大 (k, r)-核, 一个 (k, r)-核中的每个顶点至少有 k 个邻居, 同时相邻顶点间基于属性 (如位置、兴趣爱好关键词) 的相似度不小于相似度阈值 r, 也就是说 (k, r)-核是同时考虑了用户参与度和相似度两个方面的具有内聚力的子图社区, 是同时结合结构和语义信息进行社区建模和搜索的典型研究实例. 类似的, 因为一个三角形反映了 3 个对象间稳定的联系, 所以 k-truss 的 k 值越大, 说明该子图内成员间的关系越稳定, 于是, 可以反映社区内部成员间联系的稳定程度的 k-truss 社区模型也被广泛使用和扩展. 例如, 给定一个无向图和一些查询点, 文献 [26] 提出一个 CTC 社区模型, CTC 是所有包含了查询点且具有最大 k 值的连通 k-truss 子图中, 子图直径最小的那个子图.

● 动态图网络社区搜索

关于动态图网络社区搜索的研究, 一部分工作是设计各种在线算法或增量式算法快速计算每个当前时刻最新的社区搜索结果[3,4,27,28], 但是并不关注社区在一段时间内的演化行为特征. 另一部分工作则是结合社区的动态演化特征给出不同的查询条件, 然后搜索相应的社区[29,30]. 例如, 文献 [14] 提出高效的搜索算法查找动态图网络中周期性出现的团模型社区. 文献 [25] 关注动态图网络中突然出现并持续存在一段时间的稠密子图社区 Bursting Core. 但是, 以上关于动态图网络社区搜索的工作, 都没有关注过一个社区在一段时间内同时具有以下两个动态特征: 一是社区规模不断扩大, 二是社区成员内部持续保持紧密联系. 本文提出的动态信息网持续扩展社区序列模型有效刻画了具有上述两个动态特征的社区, 并给出了有效的持续扩展社区序列查找算法.

如果把动态信息网的多张图快照转化为多层图模型, 那么目前关于多层图中挖掘稠密子图的方法[31,32], 可以用于发现动态信息网中成员间持续保持紧密联系的社区, 例如跨层准团[33–35], d-连贯核[1]等. 还有一些工作研究了动态图或多层图上的聚类算法[36–38]. 但是, 这些工作都没有考虑社区在一段时间内同时具有规模不断扩大、内部成员持续保持紧密联系两个特征的情形, 因此这些方法不能用于查找持续扩展社区序列.

此外, 一些工作研究了动态图网络中动态演化社区的追踪方法[39–42], 但是这些社区在未来可能扩大 (合并)、可能不变、也可能缩小 (分裂) 甚至消失, 社区内部成员间也不是持续保持紧密联系的, 本文则是聚焦于查找一段时间内社区规模不断扩大且内部成员间持续保持紧密联系的社区.

2 基本概念及问题定义

本节将介绍动态信息网和三角连通 k-truss 模型的相关基本概念, 并给出动态信息网持续扩展社区序列模型及其查找问题的具体定义.

2.1 基本概念

本节给出动态信息网络的一些基本概念及其符号表示, 介绍三角连通 k-truss 模型以及其相关定义, 并阐述本文提出问题的具体定义, 表 1 简要总结了本文常用的符号及其含义.

表 1 文中符号及其描述

符号	描述	符号	描述
$G=(V(G),E(G))$	静态信息网	s	支持度阈值
$G[H]$	点集 H 的导出子图	q	查询点
$\triangle_{uvw}$	图的三角形结构	$\mathcal{T}_x$	点 x 的TCP索引
$Sup_G(e)$	边 e 的支持度	$\delta(u,v)$	TCP索引中边的权值
$Truss(e)$	边 e 的truss值	$V_k(x,y)$	点 y 在 $\mathcal{T}_x$ 中的连通点集
$Truss(v)$	点 v 的truss值	$\mathcal{S}_{\text{cand}}$	候选社区的集合
$\mathcal{G}=(G_1,G_2,\ldots,G_i,\ldots)$	动态信息网	$\mathcal{S}_{\text{term}}$	候选终点社区的集合
$W_z=(G_z,G_{z+1},\ldots,G_{z+\tau-1})$	动态信息网窗口	$\mathcal{S}_{\text{pres}}$	前置社区的集合
$CS=(C_{t_1},C_{t_2},\ldots,C_{t_x})$	社区序列	WS_{cur}	当前序列权值
$w(v,G_i)$	G_i 中点 v 的权值	$WS_{\max}$	当前最大序列权值
$w(\mathrm{C},G_i)$	G_i 中社区 C 的权值	p	单次点
$w(CS,W_z)$	W_z 中序列 CS 的权值	$\mathcal{T}_{ix}(p)$	TCP索引生成树

一个静态信息网常被建模成简单无向图 $G=(V(G),E(G))$, 其中 $V(G)$ 代表图 G 的顶点集, $E(G)\subseteq V(G)\times V(G)$ 代表图 G 的边集. $n=|V(G)|$ 是图 G 中顶点的总数, $m=|E(G)|$ 是图 G 中所有边的总数.

对于图 G 中存在的任意顶点 v, 使用 $Nei(v,G)$ 来表示 v 在图 G 中所有邻居的集合, 用 $Deg(v,G)=|Nei(v,G)|$ 表示 v 在图 G 中所有邻居的总数, 即 v 在图 G 中的度数. 给定图 G 的一个顶点子集 $H\subseteq V(G)$, 可以用 $G[H]=(H,E(G[H]))$ 来代表在原图上由点集 H 导出的子图, 其中 $E(G[H])$ 满足条件 $E(G[H])\subseteq E(G)$.

对于图 G, 用 $\triangle_{uvw}$ 表示 G 的一个三角形, 如果 $u,v,w\in V(G)$ 且 $(u,v),(u,w),(v,w)\in E(G)$. 对于任意一条边 $e=(u,v)\in E(G)$, 定义 e 在 G 中的支持度 $Sup_G(e)$ 为 G 中所有包含了边 e 的三角形的数目, 即 $Sup(e)=|\{\triangle_{uvw}|w\in(G)\}|$.

给定图 G 和一个参数 k, 对于一个顶点子集 $C\subseteq V(G)$, 如果其导出子图 $G[C]$ 的每一条边都被包含在至少 $k-2$ 个三角形中, 即 $Sup_{G[C]}(e)\geqslant k-2(e\in E(G[C]))$, 且 $G[C]$ 是极大的, 则 $G[C]$ 被称为 k-truss. 对于任意一条边 $e=(u,v)\in E(G)$, 用 $Truss(e)$ 表示 e 在 G 中的 truss 值, 等于 G 中包含 e 的所有 k-truss 中最大的 k 值. 对于任意一个顶点 $v\in V(G)$, 用 $Truss(v)$ 表示 v 在 G 中的 truss 值, 等于 G 中以 v 为端点的所有边的 truss 值的最大值, 即 $Truss(v)=\max\limits_{e=(v,u)\in E(G)}\{Truss(e)\}$.

给定图 G 中的两个三角形 $\triangle_1$和$\triangle_2$, 如果 $\triangle_1$和$\triangle_2$ 有共同的边, 则称 $\triangle_1$和$\triangle_2$ 是三角邻接的, 记为 $\triangle_1\cap\triangle_2\neq\varnothing$.

给定两个三角形 $\triangle_s$ 和 $\triangle_t$, 如果存在一系列的三角形 $\triangle_1,\triangle_2,\ldots,\triangle_n(n\geqslant 2)$, 其中 $\triangle_1=\triangle_s$ 且 $\triangle_n=\triangle_t$, 且对于 $1\leqslant j<n$, 都满足 $\triangle_j\cap\triangle_{j+1}\neq\varnothing$, 也就是说, $\triangle_s$ 和 $\triangle_t$ 能够通过一系列相邻的三角形而连通, 则称 $\triangle_s$ 和 $\triangle_t$ 之间存在三角连通关系, 记为 $\triangle_s\Leftrightarrow\triangle_t$.

给定图 G 中的任两条边 e_1 和 e_2, 如果存在三角形 $\triangle_s$ 和 $\triangle_t$, 满足 $e_1\in\triangle_s$ 且 $e_2\in\triangle_t$, 且 $\triangle_s\Leftrightarrow\triangle_t$, 则称边 e_1 和 e_2 之间存在三角连通关系, 记为 $e_1\Leftrightarrow e_2$. 特别地, 如果 $\triangle_s=\triangle_t$, 仍有 $e_1\Leftrightarrow e_2$.

定义 1 (三角连通 k-truss 社区). 给定图 G 和一个参数 $k\geqslant 2$, 以及图 G 的一个顶点子集 $\mathrm{C}\subseteq V(G)$, C 被称为 G 的一个三角连通 k-truss 社区[2], 当且仅当 C 满足以下 3 个条件.

(1) $G[C]$ 是一个 k-truss.

(2) $\forall\ e_1,e_2\in E(G[C])$, 都有 $e_1\Leftrightarrow e_2$.

(3) C 是 G 中满足上述 2 个条件的极大顶点子集, 即不存在 $C'\subseteq V(G)$ 且 $C\subset C'$, 且 C' 同时满足条件 (1) 和条件 (2).

需要注意的是, 当一个顶点子集 $C\subseteq V(G)$ 只满足条件 (1) 和条件 (2) 时, 称 C 是 G 的一个三角连通 k-truss.

定义 2 (动态信息网). 动态信息网 $\mathcal{G}=(G_1,G_2,\ldots,G_i,\ldots)$ 是一组静态信息网快照的序列, 其中, $G_i=(V(G_i),E(G_i))$ 为动态信息网 $\mathcal{G}$ 在第 i 时刻的快照, $V(G_i)$ 和 $E(G_i)$ 分别为 G_i 的顶点集和边集.

给定一个起始时刻 z, 窗口长度 τ, 则 $\mathcal{G}$ 的一个时间窗口 W_z 是以下 τ 个快照的序列: $G_z,G_{z+1},\ldots,G_{z+\tau-1}$.

2.2 问题定义

定义 3 (动态信息网社区序列). 对动态信息网 $\mathcal{G}$ 上窗口 W_z 的若干张快照 $(G_{t_1},G_{t_2},\ldots,G_{t_x})$ ($t_1\leqslant t_2\leqslant\ldots\leqslant t_x$), 存在一组顶点集 $CS=(C_{t_1},C_{t_2},\ldots,C_{t_x})$, 使得 $C_{t_1}\subseteq V(G_{t_1}),C_{t_2}\subseteq V(G_{t_2}),\ldots,C_{t_x}\subseteq V(G_{t_x})$, 则称 CS 为 W_z 上的一个社区序列. 每个 $C_{t_x}\in CS$ 为一个社区, 特别地, C_{t_1} 为社区序列 CS 的起点社区 (简称起点), C_{t_x} 为社区序列 CS 的终点社区 (简称终点). 社区序列的长度为该序列包含的社区总数.

需要注意的是, 在本文中, 对动态信息网 $\mathcal{G}$ 的每一张快照 G_i, 每个顶点 v 都有一个权值 $w(v,G_i)$. $w(v,G_i)$ 是基于图计算方法获得的 v 在图 G_i 中的重要性或影响力得分 (度数、PageRank 值[6]等).

对快照 G_i 上的任意一个三角连通 k-truss C, 定义 C 的权值为 $w(\mathrm{C},G_i)=\min\limits_{v\in\mathrm{C}}\{w(v,G_i)\}$, 即 C 的权值为其包含的所有顶点权值的最小值.

定义 4 (动态信息网社区序列权值). 对动态信息网 $\mathcal{G}$ 的窗口 W_z 的一个社区序列 $CS=(C_{t_1},C_{t_2},\ldots,C_{t_x})$, 定义 CS 的权值为 $w(CS,W_z)=\min\limits_{C\in CS,G_i\in W_z}\{w(\mathrm{C},G_i)\}$, 即社区序列 CS 的权值为其包含的所有社区权值的最小值.

本文定义社区序列权值的方法是合理的, 该定义直观地保证了社区序列中每个社区的每个顶点都应该有更大的权值 (重要性或影响力)[43]. 之所以不使用平均权值的定义方法, 是因为一个社区序列即使有很大的平均权值, 其中仍然可能包含一些权值 (重要性或影响力) 很小的顶点.

定义 5 (动态信息网持续扩展社区序列). 给定动态信息网 $\mathcal{G}$ 的一个窗口 $W_z=(G_z,G_{z+1},\ldots,G_{z+\tau-1})$, 参数 k, 支持度阈值 s ($2\leqslant s\leqslant\tau$), 称一个社区序列 CS 在 W_z 中是持续扩展的, 如果 CS 满足以下 3 个条件.

(1) $CS=\left(C_{t_j},C_{t_{j+1}},\ldots,C_{t_{j+s-1}}\right)$ 的长度为 s, 其中 $C=C_{t_j}\subseteq V\left(G_{t_j}\right)$, $C_{t_{j+1}}\subseteq V\left(G_{t_{j+1}}\right),\ldots,C_{t_{j+s-1}}\subseteq V(G_{t_{j+s-1}})$, 且 G_{t_j}, $G_{t_{j+1}},\ldots,G_{t_{j+s-1}}\in W_z$;

(2) 在 CS 中, $C=C_{t_j}\subseteq C_{t_{j+1}}\subseteq\ldots\subseteq C_{t_{j+s-1}}$, 且每个社区 $C_{t_x}\in CS$ 的导出子图 $G_{t_x}[C_{t_x}]$ 都是三角连通 k-truss;

(3) 在 CS 中, $C_{t_j},C_{t_{j+1}},\ldots,C_{t_{j+s-1}}$ 分别是 $G_{t_j},G_{t_{j+1}},\ldots,G_{t_{j+s-1}}$ 上满足条件 (1) 和 (2) 的极大顶点子集, 即不存在 $C'_{t_j}\subseteq V\left(G_{t_j}\right)$, $C'_{t_{j+1}}\subseteq V\left(G_{t_{j+1}}\right)$ 或 $C'_{t_j}\subseteq V\left(G_{t_j}\right)$ 满足条件 (1) 和 (2), 同时有 $C_{t_j}\subset C'_{t_j}$, $C_{t_{j+1}}\subset C'_{t_{j+1}}$ 或 $C_{t_{j+s-1}}\subset C'_{t_{j+s-1}}$.

需要注意的是, 在一个窗口 W_z 中, 多个持续扩展社区序列可能具有相同的起点社区, 也可能具有相同的终点社区. 其中, 相同的起点社区经过一段时间, 持续扩展为具有不同终点社区的多个社区序列, 代表了起点社区在这段时间具有不同的发展方向. 而不同的起点社区经过一段时间, 持续扩展为具有相同终点社区的多个社区序列, 则代表这些起点社区在这段时间的发展方向逐渐趋同. 因此, 当多个持续扩展社区序列具有相同的终点社区时, 可以选择一个代表性社区序列 (如权值最大的一个社区序列), 来代表这些社区序列这段时间最终共同的发展方向.

定义 6 (动态信息网基于查询点 q 的持续扩展社区序列). 给定动态信息网 $\mathcal{G}$ 的一个窗口 W_z, 查询点 q, 参数 k 以及支持度阈值 s, 称一个长度为 s 的社区序列 $CS=\left(C_{t_j},C_{t_{j+1}},\ldots,C_{t_{j+s-1}}\right)$ 是 W_z 中基于查询点 q 的持续扩展社区序列, 如果 CS 在窗口 W_z 上是持续扩展的, 且每个社区 $C_{t_x}\in CS$ 都包含了查询顶点 q, 即 $\forall C_{t_x}\in CS,q\in C_{t_x}$.

问题定义 (动态信息网基于查询点 q 的持续扩展社区序列查找问题). 给定动态信息网 $\mathcal{G}$ 的一个窗口 W_z, 查询点 q, 参数 k 以及支持度阈值 s, 查找 W_z 中所有基于查询点 q 的持续扩展社区序列. 特别地, 当多个基于查询点 q 的持续扩展社区序列具有相同的终点社区时, 只输出这些序列中权值最大的一个序列作为代表.

3 动态信息网持续扩展社区序列查找算法

为查找动态信息网窗口 W_z 中基于查询点 q 的持续扩展社区序列 qLEC, 基本的查找算法包括以下两个阶段:

(1) 正向计算 qLEC 社区序列在各快照中的社区候选顶点集: 对于给定的查询点 q 和参数 k, 从窗口 W_z 起始位置开始, 依次对 W_z 的每张快照, 计算所有可以作为 qLEC 中的社区的候选顶点集.

(2) 反向回溯查找 qLEC: 对于给定的查询点 q, 参数 k 和支持度阈值 s, 从窗口 W_z 末尾开始, 以阶段 (1) 获得的每个社区候选顶点集 C 为终点, 往前进行回溯, 查找所有以 C 为终点的 qLEC, 计算每个 qLEC 的权值. 当多个 qLEC 具有相同的终点时, 只输出其中具有最大序列权值的那一个 qLEC.

以上两个阶段都需要调用基于查询点 q 的三角连通 k-truss 社区查找算法[2], 因此, 在第 3.1 节, 先简要介绍 TCP 索引[2]及三角连通 k-truss 社区查找算法. 然后, 在第 3.2 节和第 3.3 节, 针对本文提出的动态信息网持续扩展社区序列查找算法, 分别详细介绍如何正向计算各快照中的社区候选顶点集, 以及如何反向回溯查找 qLEC. 最后在第 3.4 节, 对我们提出的算法进行复杂性分析.

3.1 TCP 索引与三角连通 *k*-truss 社区查找算法

给定一张快照 G_i, 以及查询点 q 和参数 k, 为查找 G_i 中所有基于查询点 q 的三角连通 k-truss 社区, 首先要对 G_i 进行 truss 分解[20], 获得每条边的 truss 值; 然后从点 q 的每条邻边出发进行广度优先遍历 (BFS), 根据边的 truss 值以及邻边之间的三角连通性质, 查找包含查询点 q 并且符合定义 1 的三角连通 k-truss 社区. 为节省遍历边的时间开销, 文献 [2] 提出了 TCP 索引结构来加速计算.

3.1.1 Truss 分解

给定一张快照 G_i, 为计算每条边 e 的 truss 值 $Truss(e)$, 采用如下步骤[2,20].

(1) 初始 k=2, 计算每条边 e 所在的三角形数目, 即 e 的支持度 $Sup_G(e)$.

(2) 按支持度大小的升序排列所有边.

(3) 对每条当前支持度最小的边 e, 如果 $Sup_G(e) \leqslant k-2$, 在图中删除边 e, 记 $Truss(e)=k$, 同时所有与 e 在同一个三角形中的边的支持度都要减 1. 之后根据新的支持度大小重新升序排列所有边.

(4) k 加 1. 重复第 (3) 步直到 G_i 上所有支持度小于等于 $(k-2)$ 的边都被删除为止. 此时获得每条边 e 的 truss 值 $Truss(e)$.

图 2 给出了快照 G_i 的 truss 分解示例. 初始 k=2, 图中不存在 $Sup \leqslant 0$ 的边, 因此令 k 加 1 (k=3), 找到所有 $Sup \leqslant 1$ 的边 $(8,9),(9,10),(5,7),(6,7)$, 将这些边的 truss 值标记为 3 并删除, 并且更新这些边的邻边的 Sup 值. 在所有 truss 值为 3 的边被删除后, 继续令 k 加 1, 并重复上述步骤, 直至图中所有的边被标记 truss 值. 可以看到, 随着 truss 值从小到大, 图中的边形成了层次结构, truss 值越大, 社区成员间的联系越紧密、越稳定 (包含了越多的三角形); 反之, truss 值越小, 社区成员间的联系越松散、越不稳定.

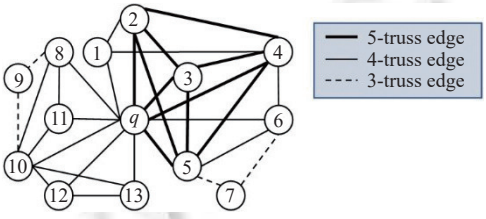


图 2 快照 G_i 的 truss 分解示例

令 k=4 时, 图 2 中的顶点集 $\{q,1,\ldots,6,8,10,\ldots,13\}$ 形成了一个 4-truss, 其中, 顶点集 $\{q,1,\ldots,6\}$ 和顶点集 $\{q,8,10,\ldots,13\}$ 分别形成了两个包含查询顶点 q 的三角连通 4-truss 社区.

3.1.2 TCP 索引

对快照 G_i 进行 truss 分解后, 为减少查找基于查询点 q 的三角连通 k-truss 社区所需的图遍历时间, 为每个顶

点 $x \in V(G_i)$, 建立一个森林结构的索引, 称为 TCP 索引, 用 $\mathcal{T}_x$ 表示 (初始化为 $Nei(x,G_i)$). 建立 $\mathcal{T}_x$ 的具体步骤如下[2].

(1) 对每个 x, 建立一个子图 $G_i[x]=(V(G_i[x]),E(G_i[x]))$, 其中 $V(G_i[x])=Nei(x,G_i)$, $E(G_i[x])=\{(y,z)|(y,z)\in E(G_i),y,z\in Nei(x,G_i)\}$, 即 x 在 G_i 上的所有邻居构成了 $G_i[x]$ 的顶点集, 另外, 如果 x 在 G_i 上的 2 个邻居在 G_i 上有边, 则这 2 个邻居在 $G_i[x]$ 上也有边.

(2) 为每条边 $(y,z)\in E(G_i[x])$, 赋一个权值 $\delta(y,z)=\min\{Truss((x,y)),Truss((x,z)),Truss((y,z))\}$.

(3) 令 $k_{\max}$ 为 $G_i[x]$ 中的最大边权值, 即 $k_{\max}=\max\{\delta(y,z)|(y,z)\in E(G_i[x])\}$. 对每个 k 值 (从 $k_{\max}$ 依次减小到 2), 用 S_k 记录 $G_i[x]$ 中所有权值等于 k 的边, 即 $S_k=\{(y,z)|(y,z)\in E(G_i[x]),\delta(y,z)=k\}$. 对 S_k 中的每条边 (y,z), 如果此时顶点 y,z 分别在 $\mathcal{T}_x$ 的两个不同连通分量中, 则在 $\mathcal{T}_x$ 中为顶点 y,z 连一条边 (y,z), 并在 $\mathcal{T}_x$ 中为边 (y,z) 赋予和在 $G_i[x]$ 中相同的权值 $\delta(y,z)$.

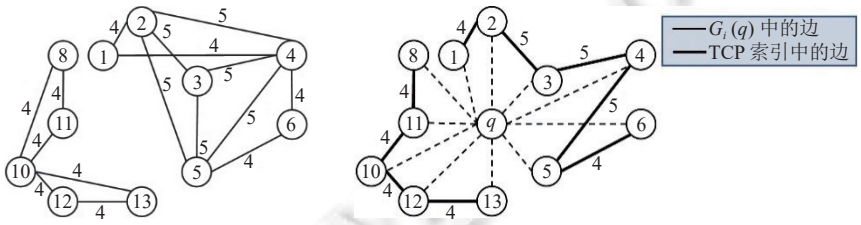
TCP 索引构建算法的伪代码如算法 1 所示. 图 3 是对图 2 快照 G_i 建立的顶点 q 的 TCP 索引示例.

算法 1. TCP 索引的构建算法.

输入: 无向图 $G=(V,E)$;

输出: 图中每个点 x 的 TCP 索引结构 $\mathcal{T}_x$.

```
1. 在 G 上进行 truss 分解;
2. for x∈V do
3.     G_x ← {(y,z)|y,z∈N(x),(y,z)∈E}
4.     for (y,z)∈E(G_x) do
5.         δ(y,z) ← min{Truss((x,y)),Truss((y,z)),Truss((x,z))};
6.     T_x ← N(x)
7.     k_max ← max{δ(y,z)|(y,z)∈E(G_x)};
8.     for k ← k_max to 2 do
9.         S_k ← {δ(y,z)|(y,z)∈E(G_x),δ(y,z)=k}
10.        for (y,z)∈S_k do
11.            If y 与 z 在 T_x 的 2 个不同连通分量中
12.                将边 (y,z) 加入 T_x, 并为 (y,z) 赋权值 δ(y,z);
13. return {T_x|x∈V};
```



(a) 顶点 q 导出的子图 $G_i[q]$ (b) 顶点 q 的 TCP 索引 $\mathcal{T}_q$

图 3 快照 G_i 中顶点 q 的 TCP 索引示例

3.1.3 基于查询点 q 的三角连通 k-truss 社区查找算法

建立 TCP 索引 $\mathcal{T}_x$ 后, 对任一查询点 q, 在快照 G_i 上采用以下步骤查找包含查询点 q 的所有三角连通 k-truss 社区[2].

(1) l 初始化为 0, 队列 Q 为空.

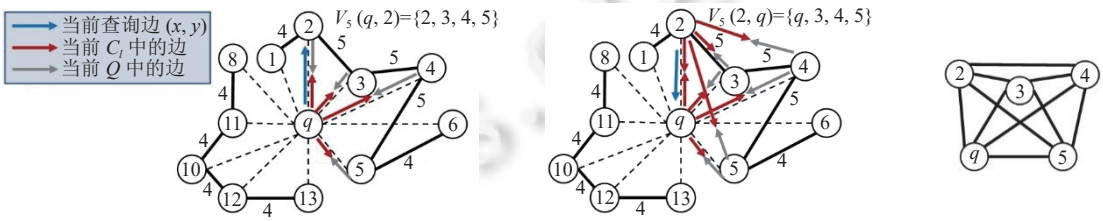
(2) 对 q 当前未被访问过的邻居 $u \in Nei(v_q, G_i)$, 如果 $Truss((x,y)) \geqslant k$, 将有向边 (v_q,u) 压入队列 Q 中, l 加 1.

(3) 若队列 Q 不为空, 从 Q 中取出一条目前未被访问过的有向边 (x,y), 计算顶点集 $V_k(x,y)=\{z|$ 顶点 z 与顶点 y 通过 TCP 索引 $\mathcal{T}_x$ 中权值不小于 k 的边所连接$\}\cup\{y\}$.

对于 $V_k(x,y)$ 中每个顶点 z, 将有向边 (x,z) 加入第 l 个社区 C_l, 并将该边标记为已访问; 如果逆有向边 (z,x) 未被访问过, 则将 (z,x) 压入队列 Q 中.

(4) 重复第 (3) 步直到 Q 为空, 获得第 l 个社区 C_l. 返回第 (2) 步, 计算下一个社区 C_{l+1}, 直到 v_q 在 G_i 上的邻居都被访问过.

以上步骤找到的 l 个社区 $C_1,\ldots,C_l$ 就是所有包含查询点 q 的三角连通 k-truss 社区. 也就是说, 一个查询点 q 可能同时被多个不同的三角连通 k-truss 社区所包含. 如图 4 所示, 给出了 k=5 时, 查找基于 q 的三角连通 5-truss 社区的前两次访问过程示例. 其中, (a) $k=5$, 访问 $(q,2)$, $Q=\varnothing$, $V_5(q,2)=\{2,3,4,5\}$, $C_1=\{(q,2),(q,3),(q,4),(q,5)\}$, $Q=\{(2,q),(3,q),(4,q),(5,q)\}$. (b) $k=5$, 访问 $(2,q)$, $Q=\{(3,q),(4,q),(5,q)\}$, $V_5(2,q)=\{q,3,4,5\}$, $C_1=\{(q,2),(q,3),(q,4),(q,5),(2,q),(2,3),(2,4),(2,5)\}$, $Q=\{(3,q),(4,q),(5,q),(3,2),(4,2),(5,2)\}$.



(a) 第1次访问　　(b) 第2次访问　　(c) 结果社区

图 4　使用 TCP 索引查找包含 q 的三角连通 5-truss 社区的前两次访问过程示例

3.2　正向计算 qLEC 社区序列在各快照中的社区候选顶点集

根据定义 5 可知, 一个 qLEC 社区序列 CS 中的每个社区 $C_{t_x} \in CS$ 的导出子图 $G_{t_x}[C_{t_x}]$ 都是快照 G_{t_x} 上的一个三角连通 k-truss. 再结合定义 1, 一个三角连通 k-truss 社区是其所在快照上满足三角连通 k-truss 性质的极大顶点集, 因此, 如果窗口 W_z 中存在一个 qLEC 社区序列 CS, 那么对快照 G_i 上的社区 $C_i \in CS$, 一定在 G_i 上存在一个三角连通 k-truss 社区包含 C_i.

于是, 给定查询点 q 和参数 k, 在快照 G_i 上找 qLEC 社区序列的社区候选顶点集之前, 先判断以下两个条件.

(1) 查询点 $q \in V(G_i)$ (因为 qLEC 社区序列的每个社区必须包含点 q);

(2) 在 G_i 上, 点 q 的 truss 值 $Truss(q)=\max\{Truss(e_j)|e_j=(q,u)\in E(G_i)\}\geqslant k$ (因为如果 $Truss(q)<k$, 说明不存在边 $e_j=(q,u)\in E(G_i)$, 满足 $Truss(e_j)\geqslant k$, 则显然 G_i 上不存在包含点 q 的三角连通 k-truss 社区).

如果以上两个条件都满足, 则调用第 3.1.3 节基于查询点 q 的三角连通 k-truss 社区查找算法, 找到 G_i 上所有基于查询点 q 的三角连通 k-truss 社区, 就是快照 G_i 上所有可能成为 qLEC 社区序列中社区的候选顶点集, 将它们加入到集合 $\mathcal{S}_{\text{cand}}$ 中; 如果以上任意一个条件不满足, 直接跳过快照 G_i, 对下一张快照 G_{i+1} 重新开始以上两个条件的判断和计算. 正向计算社区序列的社区候选顶点集的伪代码如算法 2 所示.

算法 2. 正向计算社区候选顶点集算法.

输入: 动态信息网络 $\mathcal{G}=(G_1,G_2,\ldots)$ 的一个窗口 $W_z=(G_z,G_{z+1},\ldots,G_{z+\tau-1})$; 参数 k、支持度阈值 s; 查询顶点 q;
输出: 社区序列 qLEC 中社区候选顶点集的集合 $\mathcal{S}_{\text{cand}}$, 社区序列 qLEC 的候选终点社区的集合 $\mathcal{S}_{\text{term}}$.

1. $\mathcal{S}_{\text{cand}} \leftarrow \varnothing; \mathcal{S}_{\text{term}} \leftarrow \varnothing$;
2. **for** $i \leftarrow z$ **to** $z+\tau-1$ **do**

3. **if** $q \in G_i$ **then**
4. $Truss(q) \leftarrow \max\{Truss(e_j) \mid e_j=(q,u) \in E(G_i)\}$;
5. **if** $Truss(q) \geqslant k$ **then**
6. 计算 G_i 上包含 q 的每个三角连通 k-truss 社区 C, 将 $\langle i, C\rangle$ 加入 $\mathcal{S}_{\text{cand}}$;
7. **if** $i \geqslant z+s-1$ **then**
8. 将 $\langle i, C\rangle$ 加入 $\mathcal{S}_{\text{term}}$;
9. **return** $\mathcal{S}_{\text{cand}}$, $\mathcal{S}_{\text{term}}$

需要注意的是, 本文的 qLEC 社区序列查找算法, 只在最开始的预处理阶段为窗口中每张快照进行 truss 分解、建立 TCP 索引, 之后每给定一组查询条件 (包括查询点 q、参数 k 和支持度阈值 s), 在正向计算社区候选顶点集的过程中, 无需对每张快照重新进行 truss 分解、建立 TCP 索引, 而是直接使用预处理阶段建立的 TCP 索引, 即可获得所有满足当前查询条件的社区候选顶点集.

另外, 在正向计算 qLEC 社区序列在各快照中的社区候选顶点集的同时, 可以记录 qLEC 社区序列所有的候选终点社区.

根据定义 5, qLEC 社区序列的终点社区 C_i 是其所在图快照 G_i 上的一个三角连通 k-truss 社区, 因为如果 C_i 不是一个三角连通 k-truss 社区, 根据定义 1, G_i 上就会存在一个极大的三角连通 k-truss 子图包含 C_i, 这与定义 5 条件 (3) 矛盾, C_i 就不是 qLEC 社区序列的终点社区; 同时, 根据支持度阈值 s 的约束, 对于当前窗口 $W_z=(G_z, G_{z+1}, \ldots, G_{z+\tau-1})$, 终点社区 C_i 只能存在于 G_{z+s-1} 快照及其之后的快照上 (即 $i \geqslant z+s-1$), 才能保证发现的 qLEC 社区序列的长度不小于 s.

基于以上分析, 当找到快照 G_i 上所有的三角连通 k-truss 社区后 (记录在 HS_{ik} 中), 判断是否满足 $i \geqslant z+s-1$, 如果满足, 则将 $\mathcal{S}_{\text{cand}}$ 中所有三角连通 k-truss 社区作为 qLEC 社区序列在快照 G_i 上所有可能的候选终点社区, 存入集合 $\mathcal{S}_{\text{term}}$ 中 (算法 2 第 7,8 行).

3.3 反向回溯查找 qLEC

在介绍 qLEC 反向回溯查找算法前, 先引入一些基本概念: 给定任一社区序列 $CS=(C_{t_1}, C_{t_2}, \ldots, C_{t_x})$, 对序列中任意 2 个相邻的社区 C_{t_j} 和 $C_{t_{j+1}}$, 称 C_{t_j} 为 $C_{t_{j+1}}$ 的前置社区, $C_{t_{j+1}}$ 为 C_{t_j} 的后置社区, 注意 t_j 和 t_{j+1} 不一定是相邻的两个时刻. 另外, 用 $C.pos$ 来索引社区 C 在序列 CS 的第 $C.pos$ 个位置, 如 $C_{t_1}.pos=1$ 表示 C_{t_1} 在 CS 的第 1 个位置 (即 C_{t_1} 是 CS 的第 1 个社区).

需要注意的是, 在第 3.2 节获得的社区候选顶点集和候选终点社区的集合 $\mathcal{S}_{\text{cand}}$ 和 $\mathcal{S}_{\text{term}}$ 中, 每个元素都是二元组 $\langle i, C\rangle$, 表示出现在快照 G_i 上的社区候选顶点集或候选终点社区 C, 为便于说明, 用 $C.sid$ 来索引 C 在窗口 W_z 的第 $C.sid=i$ 张快照.

qLEC 反向回溯查找算法的主要思想是, 从窗口 W_z 末尾开始, 对每个候选终点社区 C_w, 反向回溯找到 C_w 的前置社区 C_u, 然后再反向回溯找到 C_u 的前置社区 C_v, 回溯过程持续迭代进行, 直到找到所有以 C_w 为终点社区的、长度为 s 的 qLEC 为止. 接下来从另一个候选终点社区重新进行反向回溯过程找新的 qLEC. 整个回溯过程相当于在窗口中从后往前对每个候选终点社区 C_w 进行一次深度优先搜索.

例如, 图 5 给出了 s=3 时, 在窗口 $W_1=(G_1, G_2, G_3, G_4)$ 中反向回溯查找所有 qLEC 的示例. 其中, $C_{(i,j)}$ 表示快照 G_i 中的第 j 个社区候选顶点集. 首先, 对候选终点社区 $C_{(4,1)}$, 反向回溯到其前置社区 $C_{(3,1)}$, 再反向回溯到其前置社区 $C_{(2,1)}$, 此时序列 $CS=(C_{(2,1)}, C_{(3,1)}, C_{(4,1)})$ 长度为 3, 是一个 qLEC. 接下来, 分别从候选终点社区 $C_{(3,1)}$ 以及 $C_{(3,2)}$ 进行反向回溯, 找到另一个长度为 3 的序列 $CS'=(C_{(1,2)}, C_{(2,2)}, C_{(3,2)})$, 作为一个新的 qLEC. 反向回溯过程结束.

接下来, 先详细介绍如何找一个候选终点社区或社区候选顶点集的前置社区, 然后给出 qLEC 反向回溯查找算法.

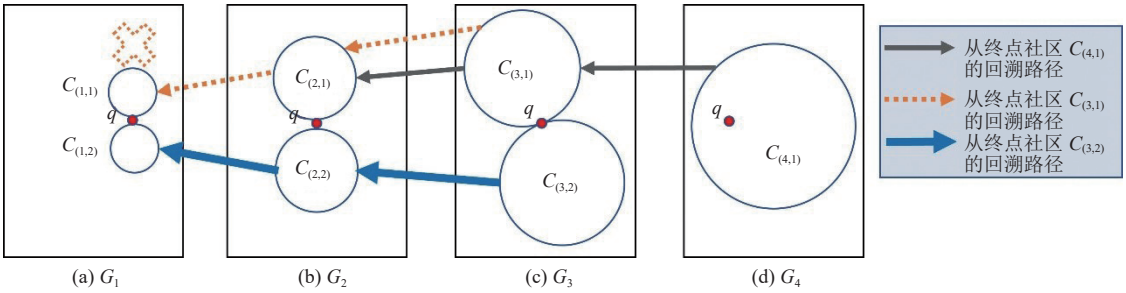


图 5 动态信息网窗口 G_1-G_4 上的反向回溯过程示例

3.3.1 查找前置社区

给定一个候选终点社区或社区候选顶点集 C, 其前置社区一定包含在第 3.2 节获得的社区候选顶点集中. 因此, 在窗口中从后往前依次考虑第 3.2 节获得的 $\mathcal{S}_{\text{cand}}$ 中的每个社区候选顶点集, 从中找出 C 的前置社区. 查找候选终点社区或社区候选顶点集 C 的前置社区的伪代码如算法 3 所示.

算法 3. 前置社区查找算法.

输入: 动态信息网窗口 $W_z=(G_z,G_{z+1},\ldots,G_{z+\tau-1})$; 所有社区的候选顶点集的集合 $\mathcal{S}_{\text{cand}}$; 参数 k; 查询顶点 q; 候选终点社区或社区候选顶点集 C;

输出: C 的所有前置社区的集合 $\mathcal{S}_{\text{pres}}$.

1. $\mathcal{S}_{\text{pres}} \leftarrow \varnothing$;
2. **for** each $C_{\text{cand}} \in \mathcal{S}_{\text{cand}}$ **do** // C_{cand} 按 $C_{\text{cand}}.sid$ 的非升序依次输出
3. **if** $C_{\text{cand}}.sid < C.sid$ **and** $C_{\text{cand}}.sid \geqslant C.pos-1$ **then**
4. $InterS \leftarrow C_{\text{cand}} \cap C$;
5. **if** $InterS.size \geqslant k$ **then**
6. i $\leftarrow C_{\text{cand}}.sid$, 对 $G_i InterS$ 进行 truss 分解;
7. $Truss_{G_i InterS}(q) \leftarrow \max\{Truss_{G_i InterS}(e_j) \mid e_j=(q,u) \in E(G_i InterS)\}$;
8. **if** $Truss_{G_i InterS}\, q \geqslant k$ **then**
9. 在 $G_i InterS$ 上计算所有包含 q 的三角连通 k-truss 社区 C';
10. $\mathcal{S}_{\text{pres}} \leftarrow \mathcal{S}_{\text{pres}} \cup \{C'\}$;
11. **return** $\mathcal{S}_{\text{pres}}$;

具体地, 若 C 是 qLEC 的社区 ($C.pos$ 表示 C 在 qLEC 的位置), 且此时 C 的前置社区是 $preC$, 则 $preC$ 所在快照 ($preC.sid$ 表示 $preC$ 所在快照的序号) 应满足以下 2 个条件.

(1) $preC.sid < C.sid$, 即在窗口 W_z 中, C 的前置社区 $preC$ 所在快照一定比 C 所在快照更靠前;

(2) $preC.sid \geqslant C.pos-1$, 因为 C 是 qLEC 的第 $C.pos$ 个社区, 如果 $preC$ 是前置社区, 那么只有当 $preC$ 所在快照及其之前的快照数量足够多时 (大于等于 $C.pos-1$), C、$preC$ 及 $preC$ 所在快照之前的快照上的社区候选顶点集才能形成满足支持度阈值 s 约束的 qLEC. 例如, 在图 5 中, s=3, 若 $C=C_{(4,1)}$, $preC=C_{(3,1)}$, 此时 $C_{(3,1)}$ 是 qLEC 的第 2 个社区, 则为了形成一个长度为 3 的 qLEC, $C_{(3,1)}$ 作为前置社区, $C_{(3,1)}$ 所在快照及其之前的快照总数一定不小于 3–1=2 ($C_{(4,1)}.pos=3$), 亦即 $C_{(3,1)}$ 的快照序号 $C_{(3,1)}.sid \geqslant 2$.

如果 $preC$ 所在快照满足上述两个条件, 接下来计算 C 在快照 $G_{preC.sid}$ 上最终所有可能的前置社区.

引理 1. 设一个 k-truss 包含 m 个顶点, 直径为 d, 则有[44]:

$$m \geqslant \begin{cases} \frac{d+1}{2}k, & d\text{为奇数} \\ \frac{d}{2}k+1, & d\text{为偶数} \end{cases}.$$

定理 1. 一个连通的 k-truss 包含至少 k 个顶点.

证明: 显然, 对任何一个联通的 k-truss, 其直径 d 至少为 1, 于是由引理 1 可知, 其至少包含 k 个顶点.

由定义 5, C 包含其前置社区, 所以先对 C 和 $preC$ 作交集运算, 得顶点集 $InterS$, 显然 C 的前置社区包含于 $InterS$. 由定理 1, 若 $Inter$ 包含的顶点数少于 k, 则 $InterS$ 在快照 $G_{preC.sid}$ 上无法形成三角连通 k-truss, 由定义 5, $Inter$ 无法成为 qLEC 中的一个社区, 放弃 $InterS$. 否则, 对 $InterS$ 的导出子图 $G_{preC.sid}InterS$, 计算其中所有包含 q 的三角连通 k-truss 社区, 即是 C 在快照 $G_{preC.sid}$ 上最终所有可能的前置社区, 存入集合 $\mathcal{S}_{\text{pres}}$.

3.3.2 qLEC 反向回溯查找算法

在使用算法 2 获得社区序列 qLEC 中社区的候选顶点集和候选终点社区的集合 $\mathcal{S}_{\text{cand}}$, $\mathcal{S}_{\text{term}}$ 后, 本文提出 qLEC 反向回溯查找算法, 从窗口 W_z 末尾开始, 以 $\mathcal{S}_{\text{term}}$ 中的每个社区候选顶点集 C 为终点, 往前进行回溯, 查找所有以 C 为终点的 qLEC, 计算每个 qLEC 的权值. 当多个 qLEC 具有相同的终点时, 只输出其中具有最大序列权值的那一个 qLEC. qLEC 反向回溯查找算法的伪代码如算法 4 所示.

算法 4. qLEC 反向回溯查找算法.

输入: 动态信息网络 $\mathcal{G}=(G_1,G_2,\ldots)$ 的一个窗口 $W_z=(G_z,G_{z+1},\ldots,G_{z+\tau-1})$; 参数 k、支持度阈值 s; 查询顶点 q;
输出: 所有基于查询点 q 的持续扩展社区序列的结果集 $\mathcal{R}$.

```
1. R ← ∅;
2. 使用算法 2 计算社区序列 qLEC 中社区候选顶点集和候选终点社区的集合 S_cand, S_term;
3. for each C ∈ S_term do // C 按 C.sid 的非升序依次输出
4.    WS_max ← 0;
5.    WS_cur ← w(C);
6.    C.pos = s;
7.    Sequence(C, WS_max, WS_cur);
8.    if WS_max ≠ 0 then
9.       输出 WS_max 对应的以 C 为终点社区的 qLEC;
Procedure: Sequence (C, WS_max, WS_cur)
10. 使用 Algorithm 3 查找 C 的所有前置社区, 存入集合 S_pres
11. if S_pres ≠ ∅ then
12.    for each preC ∈ S_pres do
13.       WS_cur ← min{w(preC), WS_cur};
14.       if C.pos = 2 then //找到一个长度为 s 的社区序列
15.          WS_max ← max{WS_cur, WS_max}; //取多个序列权值的最大值
16.       else
17.          preC.pos = C.pos − 1;
18.          Sequence(preC, WS_max, WS_cur);
19. return
```

qLEC 反向回溯查找算法的关键在于如何从一个候选终点社区 C 迭代地进行反向回溯, 找到以 C 为终点社区的所有 qLEC, 并找到其中具有最大权值 $WS_{\max}$ 的 qLEC. 为此, 本文提出递归函数 Sequence, 在搜索空间 $\mathcal{S}_{\text{cand}}$ 中采

用深度优先搜索策略, 找到以 C 为终点社区的所有 qLEC.

具体地, 对每个候选终点社区 C 的反向回溯过程如下:

(1) $WS_{\max}$ 初始化为 0. 维护一个以 C 为终点的临时社区序列的当前权值 WS_{cur}, 初始化为 C 的权值. 令 $C.pos = s$, 因为 C 是 qLEC 的终点社区, 即第 s 个社区.

(2) 对 C 调用前置社区查找算法, 找到 C 所有的前置社区.

(3) 对 C 的每个前置社区 $preC$, 更新 $WS_{\text{cur}} = \min\{w(preC),\ WS_{\text{cur}}\}$. 如果在当前迭代步中获得长度为 s 的临时社区序列 ($C.pos$ =2, 即 $preC$ 是当前社区序列的起点社区), 则该序列就是一个 qLEC, 此时更新 $WS_{\max} = \max\{WS_{\text{cur}},\ WS_{\max}\}$;如果在当前迭代步中临时社区序列的长度仍小于 s, 则更新 $preC$ 在社区序列中的相对位置 $preC.pos = C.pos - 1$, 然后对 $preC$ 调用 Sequence 算法, 重复迭代以上过程, 直到从 C 出发反向回溯完 $\mathcal{S}_{\text{cand}}$ 所有可能成为 qLEC 中社区的候选顶点集.

当对 C 的反向回溯过程结束, 如果 $WS_{\max} \neq 0$, 说明至少存在一个以 C 为终点的 qLEC, 输出 $WS_{\max}$ 对应的以 C 为终点社区的 qLEC 即可.

3.4 动态信息网持续扩展社区序列查找算法复杂性分析

下面对动态信息网持续扩展社区序列查找算法的两个主要阶段分别进行复杂性分析.

对大小为 τ 的窗口 $W_z = (G_z, G_{z+1}, \ldots, G_{z+\tau-1})$, 令 $n = \max_{t\in[z,z+\tau-1]}\{|V(G_t)|\}$, $m = \max_{t\in[z,z+\tau-1]}\{|E(G_t)|\}$, $|\mathcal{S}_{\text{cand}}|$ 表示 W_z 中所有社区候选顶点集的数目, $h = \max_{t\in[z,z+\tau-1]}\{|\mathcal{S}_{\text{cand}}[t]|\}$ 表示 W_z 各快照所包含的社区候选顶点集数目的最大值, $|C_{\max}|$ 表示窗口 W_z 中所有社区候选顶点集的尺寸的最大值, $|E(C_{\max})|$ 表示 $C_{\max}$ 在其所在快照的导出子图中包含的总边数, $|\mathcal{S}_{\text{term}}|$ 表示 W_z 中所有候选终点社区的数目, $|\mathcal{S}_{\text{pres}}|$ 表示任意社区 C 的前置社区的数目. 设 $Ans = C_1 \cup \ldots \cup C_l$ 为某张静态图 G_t 上对查询点 q 进行三角连通 k-truss 社区查找所得到的 k-truss 社区的并集, 记 $|E(Ans)|$ 是集合 Ans 中边的总数.

在正向计算社区候选顶点集阶段, 首先, 建立 TCP 索引的总时间和总空间分别是 $O(\tau\times m^{1.5})$ 和 $O(\tau\times m)$ [2]; 其次, 在每张快照 G_t 上, 计算包含查询点 q 的社区候选顶点集的时间和空间都是 $O(|E(Ans)|)$, 此外, 还需要存储社区候选顶点集的集合 $\mathcal{S}_{\text{cand}}$ 以及候选终点社区的集合 $\mathcal{S}_{\text{term}}$, 因此对窗口 W_z, 正向计算社区候选顶点集的总时间是 $O(\tau\times|E(Ans)|)$, 总空间是 $O(\tau\times|E(Ans)|+|\mathcal{S}_{\text{cand}}|+|\mathcal{S}_{\text{term}}|)$.

在反向回溯查找阶段, 对 $\mathcal{S}_{\text{term}}$ 中的每个终点社区, 最坏的情况下, 进行一次深度优先搜索查找 qLEC 的时间为 $O(h^{\tau-1}\times(|C_{\max}|+|E(C_{\max})|^{1.5}))$, 所以反向回溯查找 qLEC 的总时间为 $O(|\mathcal{S}_{\text{term}}|\times h^{\tau-1}\times(|C_{\max}|+|E(C_{\max})|^{1.5}))$, 总空间为 $O\left(|\mathcal{S}_{\text{cand}}|+|\mathcal{S}_{\text{term}}|+\tau\times|\mathcal{S}_{\text{pres}}|\right)$.

4 动态信息网持续扩展社区序列查找算法的时间和空间优化策略

对于第 3 节提出的动态信息网持续扩展社区序列查找算法, 本节分别设计基于提早终止策略的时间优化和基于 TCP 索引压缩技术的空间优化方法.

4.1 基于提早终止策略的时间优化

由第 3.3 节可知, 动态信息网持续扩展社区序列查找算法的反向回溯查找 qLEC 阶段需要对每个候选终点社区, 在规模为 $|\mathcal{S}_{\text{cand}}|$ 的搜索空间进行一次深度优先搜索. 当 $|\mathcal{S}_{\text{cand}}|$ 较大时, 反向回溯查找 qLEC 阶段需要很大的时间开销. 因此, 本节给出两种加速策略, 其核心思想是对任意候选终点社区 C, 利用当前以 C 为终点的所有序列的最大权值 $WS_{\max}$ 和当前以 C 为终点的临时序列权值 WS_{cur}, 提早终止从 C 出发在搜索空间 $\mathcal{S}_{\text{cand}}$ 中进行的深度优先搜索.

提早终止策略 1: 对于 C 的任意前置社区 $preC$, 如果 $w(preC) \leqslant WS_{\max}$, 则停止从 $preC$ 继续往前回溯.

提早终止策略 2: 从 C 出发反向回溯过程中获得的当前临时序列 CS, CS 的权值为 WS_{cur}, 如果 $WS_{\text{cur}} = WS_{\max}$, 则停止从 CS 继续往前回溯.

定理 2. 提早终止策略 1 能够保证正确的输出结果.

证明: 假设 $w(preC) \leqslant WS_{\max}$, 仍然从 $preC$ 继续往前回溯, 得到新的临时序列 CS'. 由定义 4, $w(CS', W_z) \leqslant w(preC)$, 则 $w(CS', W_z) \leqslant WS_{\max}$, 即 CS' 的序列权值至多为 $WS_{\max}$, 所以是仍然输出 $WS_{\max}$ 对应的以 C 为终点的社区序列作为代表. 因此 $w(preC) \leqslant WS_{\max}$ 时, 可以停止从 $preC$ 继续往前回溯.

定理 3. 提早终止策略 2 能够保证正确的输出结果.

证明: 假设对于当前临时序列 CS, $w(CS, W_z) = WS_{\text{cur}} = WS_{\max}$, 仍然从 CS 继续往前回溯, 得到新的临时序列 CS''. 由定义 4, $w(CS'', W_z) \leqslant w(CS, W_z)$, 则 $w(CS'', W_z) \leqslant WS_{\max}$, 即 CS'' 的序列权值至多为 $WS_{\max}$, 仍然会输出 $WS_{\max}$ 对应的以 C 为终点的社区序列作为代表. 因此可以停止从 CS 继续往前回溯.

举例: 如图 6 所示, 以 $C_{(4,1)}$ 作为终点社区反向回溯时, 先得到序列 $CS1 = (C_{(2,1)}, C_{(3,1)}, C_{(4,1)})$, 此时 $WS_{\max} = w(CS2, W_z) = 0.8$. 之后继续以 $C_{(4,1)}$ 为终点社区反向回溯时, 对 $C_{(4,1)}$ 的前置社区 $C_{(3,2)}$, 因为 $w(C_{(3,2)}) = 0.7 < 0.8$, 所以根据提早终止策略 1, 停止从 $C_{(3,2)}$ 继续往前回溯.

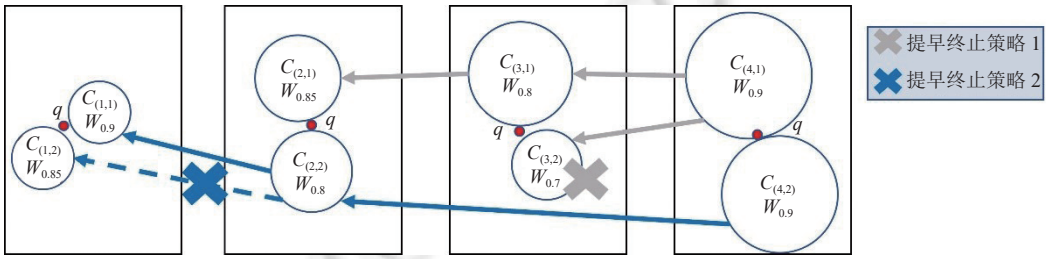


图 6 动态信息网窗口 $G_1 - G_4$ 上提早终止的反向回溯过程示例

接下来, 以 $C_{(4,2)}$ 作为终点社区反向回溯时, 先得到序列 $CS2 = (C_{(1,1)}, C_{(2,2)}, C_{(4,2)})$, 此时 $WS_{\max} = w(CS2, W_z) = 0.8$. 之后继续以 $C_{(4,2)}$ 为终点社区反向回溯时, 对当前临时序列 $CS3 = (C_{(2,2)}, C_{(4,2)})$, 因为 $WS_{\text{cur}} = w(CS3, W_z) = 0.8$, 有 $WS_{\text{cur}} = WS_{\max}$, 所以根据提早终止策略 2, 停止从 $CS3$ 继续往前回溯 (即停止考虑新的序列 $CS4 = (C_{(1,2)}, C_{(2,2)}, C_{(4,2)})$).

包含了两个提早终止策略的递归函数 Sequence_earlyTerm 如算法 5 所示, 在算法 4 中用 Sequence_earlyTerm 替换 Sequence, 即是具有提早终止加速效果的 qLEC 反向回溯查找算法.

算法 5. Procedure: Sequence_earlyTerm.

输入: 后置社区 C;社区的候选顶点集的集合 HS_k;长度约束参数 s;当前序列权值 WS_{cur};当前最大序列权值 $WS_{\max}$.

```
1. 使用算法 3 查找 C 的所有前置社区 S_pres
2. if S_pres ≠ ∅ then
3.    for each preC ∈ S_pres do
4.       if w(preC) > WS_max then //提早终止策略 1
5.          WS_cur ← min{w(preC), WS_cur};
6.          if C.pos = 2 then //找到一个长度为 s 的社区序列
7.             WS_max ← WS_cur;
8.          else
9.             PS.pos ← C.pos − 1;
10.            Sequence_earlyTerm(preC, WS_max, WS_cur);
11.            if WS_cur = WS_max then //提早终止策略 2
12.               return;
```

4.2 基于 TCP 索引压缩技术的空间优化

在第 3.2 节中提到, 本文的动态信息网持续扩展社区序列查找算法, 只在预处理阶段为窗口中每张快照进行 truss 分解、建立 TCP 索引, 之后每给定一组查询条件 (包括查询点 q、参数 k 和支持度阈值 s), 在正向计算社区候选顶点集的过程中, 无需对每张快照重新进行 truss 分解、建立 TCP 索引, 而是直接使用预处理阶段建立的 TCP 索引, 即可获得所有满足当前查询条件的社区候选顶点集.

当窗口中快照规模较大, 预处理阶段建立的 TCP 索引所占的内存空间也就越大. 为此, 本节提出有效的 TCP 索引压缩存储技术来减少 TCP 索引所占的内存空间.

首先, 给出单次点的概念: 在窗口 W_z 中, 任意顶点 p 如果仅在一张快照上出现, 则称该顶点 p 为 W_z 中的单次点. 由定义 5, 支持度阈值 $s \geqslant 2$, 即所有的 qLEC 至少包含 2 个社区 (各社区分别存在于不同的快照上). 因此, 单次点不会出现在 W_z 的任何一个 qLEC 中, 所以删除 W_z 中所有单次点不会影响最终输出的 qLEC.

于是, 在预处理阶段, 对 W_z 中各快照进行 truss 分解和建立 TCP 索引的同时, 记录 W_z 中所有的单次点, 并以二元组 $\langle i, p\rangle$ 的形式 (表示单次点 p 存在于快照 G_i) 存储在集合 $\mathcal{S}_{\text{single}}$ 中. 接下来, 进行 TCP 索引压缩: 从 W_z 初始到末尾 (i 从 z 依次递增到 $z+\tau-1$), 对每张快照 G_i 的每个单次点 p, 先删除为点 p 建立的 TCP 索引 $\mathcal{T}_{ip}$, 再在为任意非单次点 x 建立的、并且包含了点 p 的 TCP 索引 $\mathcal{T}_{ix}$ 中删除点 p, 并对 $\mathcal{T}_{ix}$ 进行调整更新. 最后, 使用压缩后的 TCP 索引, 根据每一组查询条件, 先正向计算社区候选顶点集, 再反向回溯查找 qLEC.

基于单次点的 TCP 索引压缩算法的伪代码如算法 6 所示, 其关键在于删除某个单次点 p 后, 如何调整更新包含了点 p 的 TCP 索引 $\mathcal{T}_{ix}$.

调整更新 $\mathcal{T}_{ix}$ 过程的伪代码如算法 6 中的函数 TCP_update 所示, 其主要步骤如下:

(1) 找到 $\mathcal{T}_{ix}$ 中包含点 p 的 TCP 索引生成树 $\mathcal{T}_{ix}(p)$ (因为 $\mathcal{T}_{ix}$ 可能是森林), 初始化 $\mathcal{N}_p$ 为空集 (用于存储 $\mathcal{T}_{ix}(p)$ 中具有相同权值的边)

(2) 找到 $\mathcal{T}_{ix}(p)$ 中点 p 的所有邻边 (p,y), 根据 (p,y) 的权值 k, 将其添加 $\mathcal{N}_p$ 中对应的集合 $\mathcal{N}_p[k]$, 接下来根据以下 2 个调整更新策略在 $\mathcal{T}_{ix}(p)$ 中添加新的边.

边添加策略 1: 按 k 从大到小的顺序, 对 $\mathcal{N}_p$ 中每个不为空的集合 $\mathcal{N}_p[k]$, 对 $\mathcal{N}_p[k]$ 中所有的点, 按顺序首尾相连起来, 并将这些连起来的边, 均以权值 k 添加到 $\mathcal{T}_{ix}(p)$ 中.

边添加策略 2: 按 k 从大到小的顺序, 对 $\mathcal{N}_p$ 中每个不为空的集合 $\mathcal{N}_p[k]$, 找到 $\mathcal{N}_p$ 中键值最接近 k 而小于 k 的数值 k' 所对应的集合 $\mathcal{N}_p[k']$, 将 $\mathcal{N}_p[k]$ 中的第 1 个点和将 $\mathcal{N}_p[k']$ 中的第 1 个点相连, 并以权值 k' 添加到 $\mathcal{T}_{ix}(p)$ 中.

(3) 从 $\mathcal{T}_{ix}(p)$ 中删除点 p 以及其所有邻边.

算法 6. 基于单次点的 TCP 索引压缩算法.

输入: W_z 中每个点 v 的 TCP 索引 $\mathcal{T}_{iv}$ ($z \leqslant i \leqslant z+\tau-1$);
输出: 压缩后的 TCP 索引 $\mathcal{T}'_{iv}$ ($z \leqslant i \leqslant z+\tau-1$).

1. **for** $i \leftarrow z$ **to** $z+\tau-1$ **do**
2. **for** each $\langle i,p\rangle \in \mathcal{S}_{\text{single}}$ **do**
3. 删除点 p 的 TCP 索引 $\mathcal{T}_{ip}$;
4. **for** 每个包含点 p 的非单次点 x 的 TCP 索引 $\mathcal{T}_{ix}$ **do**
5. **TCP_update**($\mathcal{T}_{ix}$, p);

Procedure: TCP_update($\mathcal{T}_{ix}$, p)

6. $\mathcal{T}_{ix}(p) \leftarrow \mathcal{T}_{ix}$ 中包含 p 的生成树;
7. $\mathcal{N}_p \leftarrow \varnothing$;
8. **for** each $(p,y) \in E(\mathcal{T}_{ix}(p))$ with $\delta(p,y)$ **do**
9. $k \leftarrow \delta(p,y)$;

10. **if** k 是 $\mathcal{N}_p$ 中没有的关键字 **do**
11. 为 $\mathcal{N}_p$ 构建关键字为 k 的元素集合 $\mathcal{N}_p[k]$;
12. $\mathcal{N}_p[k]$.push (y);
13. 从 $\mathcal{T}_{ix}(p)$ 中删除点 p 及其所有邻边;
14. **for** $k \in kt|\mathcal{N}_p[kt]$ 非空} 以 k 的降序 **do** //边添加策略 1
15. **if** $|\mathcal{N}_p[k]| \geqslant 2$ **do**
16. **for** $j \leftarrow 1$ **to** $|\mathcal{N}_p[k]|-1$ **do**
17. 将 $\mathcal{N}_p[k]$ 中的第 j 个和第 $j+1$ 个顶点连接起来, 将连接边以权值 k 加入 $\mathcal{T}_{ix}(p)$ 中;
18. **for** $k \in kt|\mathcal{N}_p[kt]$ 非空} 以 k 的降序 **do** //边添加策略 2
19. $k' = \max k_d|\mathcal{N}_p[k_d]$ 非空且 $k_d < k$;
20. 将 $\mathcal{N}_p[k]$ 中的第 1 个和 $\mathcal{N}_p[k']$ 中的第 1 个顶点连接起来, 将连接边以权值 k' 加入 $\mathcal{T}_{ix}(p)$ 中;
21. **return** $\mathcal{T}_x$;

定理 4. 使用边添加策略后的 $\mathcal{T}_{ix}(p)$ 仍是一棵 TCP 索引生成树, 且其中所有的点保持与之前相同的连通关系.

证明: (1) $\mathcal{T}_{ix}(p)$ 中边的数量仍然是点的数量–1.

假设调整修改前索引生成树 $\mathcal{T}_{ix}(p)$ 中的点的数量是 N, 根据生成树的性质, 此时索引中边的数量为 $N-1$. $N = N_{\text{else}} + N_p$, 其中 N_p 是跟 p 相邻的点, 而跟 p 相邻的边的数量为 N_p, N_{else} 是图中其他的点.

根据算法对 $\mathcal{N}_p$ 的分类, 假设与 p 相邻的 N_p 个点分别被分到了 M 个集合$\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_M\}$中, 我们有 $|\mathcal{N}_1|+|\mathcal{N}_2|+\ldots+|\mathcal{N}_M| = N_p$, 其中每个集合 $\mathcal{N}_i$ 的点数量至少为 1. 将一个集合中的 n 个点首尾相连需要 $n-1$ 条边, 所以在根据策略 1 调整过后, $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_M\}$中分别存在 $|\mathcal{N}_1|-1, |\mathcal{N}_2|-1, \ldots, |\mathcal{N}_M|-1$ 条边. 而在策略 2 调整过程中, 从 $\mathcal{N}_M$ 开始, 找到一个最接近 $\mathcal{N}_M$ 的权值 k 且比 k 更小的权值 k' 对应的集合 $\mathcal{N}_{M-1}$, 从两个集合中各取第 1 个点, 形成一条边加入生成树中. 对于除了 $\mathcal{N}_1$ 之外的每个集合, 都能形成这样的一条边, 所以形成边的数量是 $M-1$. 则最后生成树中, 因 p 的删除而被影响的顶点之间的边的数量, 为根据两个策略所添加边的数量之和, 也即 $|\mathcal{N}_1|-1+|\mathcal{N}_2|-1+\ldots+|\mathcal{N}_M|-1+M-1 = |\mathcal{N}_1|+|\mathcal{N}_2|+\ldots+|\mathcal{N}_M|-1 = N_p-1$, 因为 p 被删除, 所以 $\mathcal{T}_{ix}(p)$ 中边与点的数量关系仍然保持与删除前一致.

(2) 原本在 $\mathcal{T}_{ix}(p)$ 中, 两个能够通过边的权值均至少为 k 的路径连通的点 v_1v_2, 在调整后仍然能够通过边的权值均至少为 k 的路径连通.

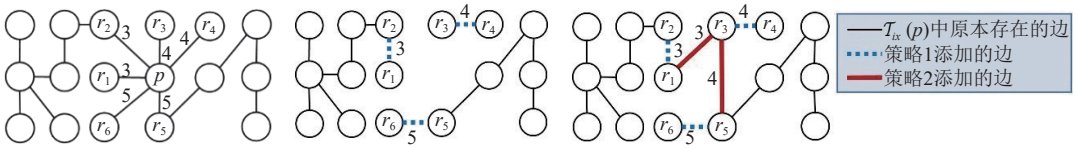
在原本的 $\mathcal{T}_{ix}(p)$ 中, v_1v_2 只能通过 p 相连通, 否则就在生成树中形成了环, 所以两个点连通的路径是唯一的, 且此路径的权值 k 是两条边 $(v_1, p), (v_2, p)$ 的权值中的较小值. 那么 v_1v_2 中至少有一个点的权值跟 k 相同, 另外一个点的权值则只有 k 或者大于 k 两种可能. 接下来通过讨论这两种情况, 来证明在调整之后, v_1v_2 仍然能够通过边的权值均至少为 k 的路径连通.

情况 1: 两个点的权值相同. 根据策略 1, 权值相同的点被分在同一个集合 $\mathcal{N}_p[k]$ 中, 并通过添加边而首尾相连. 这种情况下, 两个点仍然能够通过边的权值均至少为 k 的路径连通.

情况 2: 两个点的权值不同. 假设 v_1v_2 的权值分别为 k 和 k''(k''为$\mathcal{T}_{ix}(p)$ 中最接近 k 且又大于 k 的权值数值), 根据策略 2, k 和 k'' 对应的两个集合 $\mathcal{N}_p[k]$ 和 $\mathcal{N}_p[k'']$ 通过各自的第 1 个点而连通了起来, 那么两个集合中的 v_1v_2 也是联通的. 如果 v_2 的权值要大于 k'', 根据算法策略 2 中对集合 $\mathcal{N}_p$ 的遍历顺序, v_1v_2 所在的集合能够通过集合间的路径$\{\mathcal{N}_p[k], \mathcal{N}_p[k+1], \ldots, \mathcal{N}_p[k'']\}$而间接连通起来. 所以在这种情况下, 两个点仍然能够通过边的权值均至少为 k 的路径连通.

如图 7 所示, 在 TCP 索引 $\mathcal{T}_{ix}$ 中, 包含被删除点 p 的 TCP 索引树 $\mathcal{T}_{ix}(p)$ 如图 7(a) 所示, 其中 r_1-r_6 是删除点 p 的所有邻居, 相应邻边分别具有权值 3, 4, 5. 首先删除点 p 及其邻边. 随后在图 7(b) 中, 通过边添加策略 1, 我们将同属于 $N_p[3]$ 的 r_1, r_2, 同属于 $N_p[4]$ 的 r_3, r_4, 同属于 $N_p[5]$ 的 r_5, r_6 分别以权值 3, 4, 5 连接起来并加入到 $\mathcal{T}_{ix}(p)$ 中;

在图 7(c) 中, 通过边添加策略 2, 我们将分别属于 $N_p[3]$ 的 r_1 , $N_p[4]$ 的 r_3 两个点, 以及 $N_p[4]$ 的 r_3,$N_p[5]$ 的 r_5 两个点分别以权值 3, 4 连接起来并加入到 $\mathcal{T}_{ix}(p)$ 中.



(a) TCP 索引树 $\mathcal{T}_{ix}(p)$ (b) 策略1在 $\mathcal{T}_{ix}(p)$ 中添加边 (c) 策略2在 $\mathcal{T}_{ix}(p)$ 中添加边

图 7 在 $\mathcal{T}_x$ 上将单次点 p 删除的过程示例

5 实 验

5.1 数据集和实验设置

(1) 数据集

本文的实验使用了 4 个真实的动态信息网数据集, 包括 YouTube[45], Stack[46], Flickr[47], DBLP[48]. 具体地, YouTube 数据集记录了 YouTuBe 网站上用户间的社交数据, 顶点表示网站用户, 边表示用户间的朋友关系, 以每周的用户交互数据作为一张快照. Stack 数据集记录了 Stack Overflow 上的用户数据, 其中顶点表示用户, 顶点间的边表示不同用户在同一帖子上的互动发言关系, 以每月的用户交互数据作为一张快照. Flickr 数据集记录了图片分享网站 Flickr 的数据, 顶点表示网站用户, 边表示用户间的朋友关系, 以每周的用户交互数据作为一张快照. DBLP 数据集记录了论文作者之间的合作关系, 其中顶点表示论文作者, 边表示两个作者合作了相同的论文, 以每年作者间的合作关系数据作为一张快照. 各数据集的信息如表 2 所示. 每个数据集中顶点的权值是度数或 PageRank[6].

表 2 数据集信息

数据集	点的总数量	边的总数量	最大快照的边数
YouTube	3 223 589	9 375 374	95 523
Stack	2 601 977	63 497 050	212 759
Flickr	2 302 925	33 140 017	295 895
DBLP	1 824 701	29 487 744	3 161 088

(2) 对比算法

首先, 为测试本文提出的持续扩展社区序列两阶段查找算法的性能, 以及基于提早终止策略的时间优化和基于 TCP 索引压缩技术的空间优化方法的有效性, 我们分别实现了下列算法.

Base 算法: 持续扩展社区序列两阶段查找算法, 不提前对 TCP 索引进行压缩, 反向回溯查找持续扩展序列阶段不采用提早终止策略.

Base+Early 算法: 不提前对 TCP 索引进行压缩, 反向回溯查找持续扩展序列阶段采用提早终止策略.

Base+Comp 算法: 提前对 TCP 索引进行压缩, 反向回溯查找持续扩展序列阶段不采用提早终止策略.

其中, 通过对比 Base 算法和 Base+Early 算法测试反向回溯阶段使用提早终止策略的 CPU 运行时间优化效果, 通过对比 Base 算法和 Base+Comp 算法测试 TCP 索引压缩方法的内存空间优化效果.

然后, 为了验证本文提出的 qLEC 持续扩展社区序列模型在真实世界中所特有的重要应用价值, 我们运行了分别用于查找 MBC 社区[25](一段时间内爆发性增长的社区) 和 MPC 社区[14](一段时间内周期性出现的社区) 的两种动态图社区查找算法, 并与我们提出的 Base 算法找到的 qLEC 持续扩展社区结果进行对比.

(3) 实验参数和评价指标

影响本文所提出算法性能的实验参数主要是参数 k 、支持度阈值 s 和窗口大小参数 τ , 因此, 分别变化 k 、 s

和 τ 来测量本文所提 Base 算法的 CPU 运行时间. 各参数取值范围和默认值如表 3 所示. 一般不作特殊说明的话, 当变化一个参数的值时, 其他参数都设为其默认值. 在测量参数 τ 对算法运行时间的影响时, 在不同的数据集上, 参数 τ 采用不同的取值范围和默认值.

表 3 数据集实验参数配置

参数	数据集	取值范围	默认值
k	—	{3, 4, 5, 6, 7}	3
s	—	{3, 4, 5, 6, 7}	3
τ	YouTube	{21, 22, 23, 24, 25}	25
	Stack	{10, 15, 20, 25, 30}	30
	Flickr	{3, 6, 9, 12, 15}	15
	DBLP	{6, 7, 8, 9, 10}	10

此外, 查询点度数对算法性能也有影响. 一个查询点的度数越大, 越可能被包含在多张快照中多个较为稠密的社区中, 因此可能会需要更多的查询时间. 为此, 我们将每个数据集中所有顶点按度数从大到小排列, 分为 10 个区间, 依次包含度数排在前 10% 的顶点, 度数排在前 10%–20% 的顶点, 依次类推. 在验证参数 k 、 s 和 τ 对 Base 算法的影响时, 在度数排在前 30% 的顶点和度数排在后 70% 的顶点中各自随机采样 100 个查询点, 作为一组, 共采样 10 组, 取算法对 10 组查询点查找时间的平均运行时间作为结果, 进行对比实验. 在验证基于提早终止策略时间优化方法的有效性时, 在度数排在 10 个不同区间的顶点中各自随机采样 100 个查询点, 作为一组, 共采样 10 组, 取算法对 10 组查询点查找时间的平均运行时间作为结果. 在验证基于 TCP 索引压缩技术空间优化方法的有效性时, 分别计算运行 Base 和 Base+Comp 算法时所构建 TCP 索引的边数.

(4) 实验环境

所有的实验是在一台安装了 64 位 Windows 10 操作系统、配备了 2.9 GHz CPU 和 64 GB 内存的 PC 机上运行的, 算法采用 C++语言编程实现, 在 Microsoft Visual Studio 2019 上进行编译.

5.2 算法性能分析

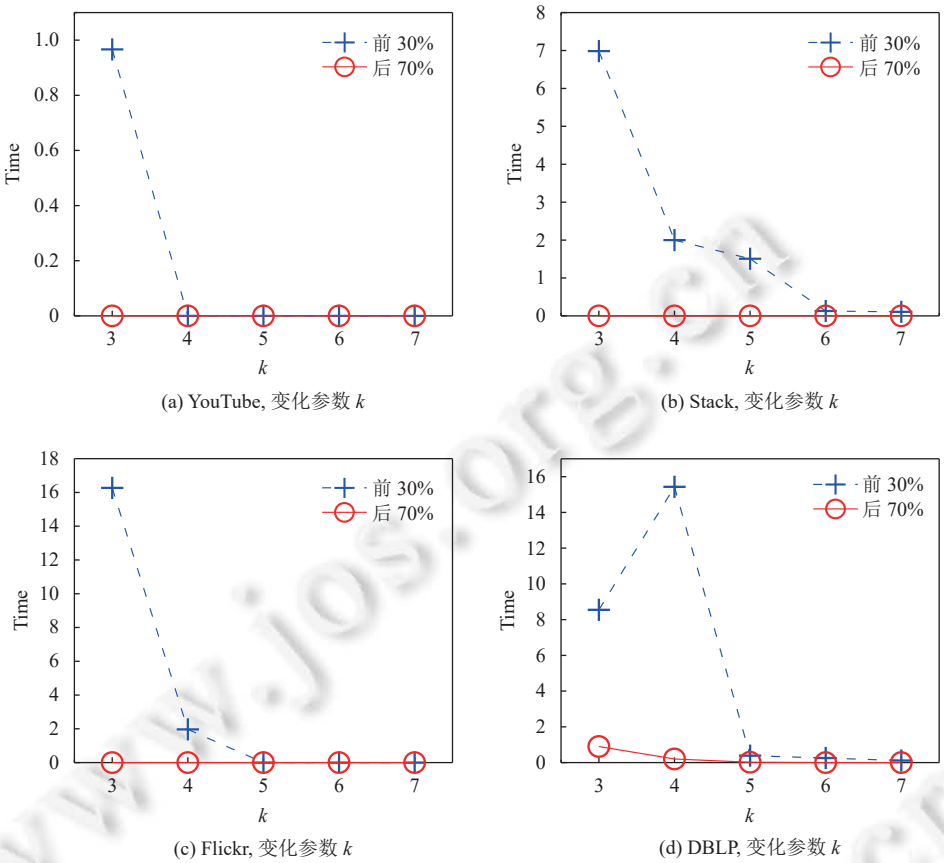
5.2.1 参数 k 对算法运行时间的影响

图 8 展示了变化参数 k 时, 4 个真实数据集上 Base 算法的 CPU 运行时间. 每个图中的两组曲线表示分别选取度数处在前 30% 的点和度数处在后 70% 的点的实验结果.

在图 8(a)–(c) 上, 对于度数处在前 30% 的点, Base 算法的运行时间呈现出随 k 增大而下降的趋势, 这是因为随着 k 的上升, 包含查询点的三角连通 k -truss 社区的规模和数量下降, 查找社区的时间和计算次数减少, 因此正向计算社区候选顶点集的时间减少; 同时随着 k 的增加, 社区的前置社区的数量和规模减少, 算法的迭代轮次和计算时间减少, 因此反向回溯查找的时间减少. 所以算法的 CPU 运行时间随着 k 的增大而下降. 其中 YouTube 数据集上, 曲线下降得更快, 这是因为 YouTube 数据集较为稀疏, 随着 k 增大, 快照中符合查询条件的候选社区数量迅速变少. 相对而言, Stack 数据集较为稠密, 随着 k 增大, 运行时间曲线下降得更为平缓, 在 k 较大时仍能返回查找结果. 在图 8(d) 上, 对于度数处在前 30% 的点, Base 算法的运行时间呈现出先上升后下降的趋势, 这是由 DBLP 数据集本身的特点所决定的, 在 DBLP 中, 论文作者们较少以 3 个人或以下规模的团队来合作发表论文结果, 所以在 k =3 时, 所查找到的 qLEC 序列数量较少, 查找时间短; 而到了 k =4 时, 由于科研团队的规模和数量上升, 持续扩展社区序列的查找时间也随之上升, 而在 k =4 之后, DBLP 数据集上的运行时间随着 k 的增长, 同样呈现出下降的趋势.

在图 8(a)–(c) 上, 对于度数处在后 70% 的查询点, 由于快照上不存在包含该类查询点的社区, 也就查找不到相应的持续扩展社区序列, 因此运行时间为 0. 在图 8(d) 上, 对于度数处在后 70% 的点, 由于 DBLP 数据集中顶点的联系普遍较为紧密, 当 k 取较小的数值时, 仍能查找到一定数量的社区序列. 但由于查询点的度数较低, 查询点所

对应的社区规模和数量较小, 因此整体的运行时间远小于前 30% 的查询点的运行时间. 同样地, 后 70% 的点的运行时间也呈现出随着 k 的增大而下降的趋势, 并在 k=5 之后查找不到持续扩展序列.



(a) YouTube, 变化参数 k　(b) Stack, 变化参数 k

(c) Flickr, 变化参数 k　(d) DBLP, 变化参数 k

图 8　真实数据集上 Base 算法的 CPU 运行时间, 变化参数 k

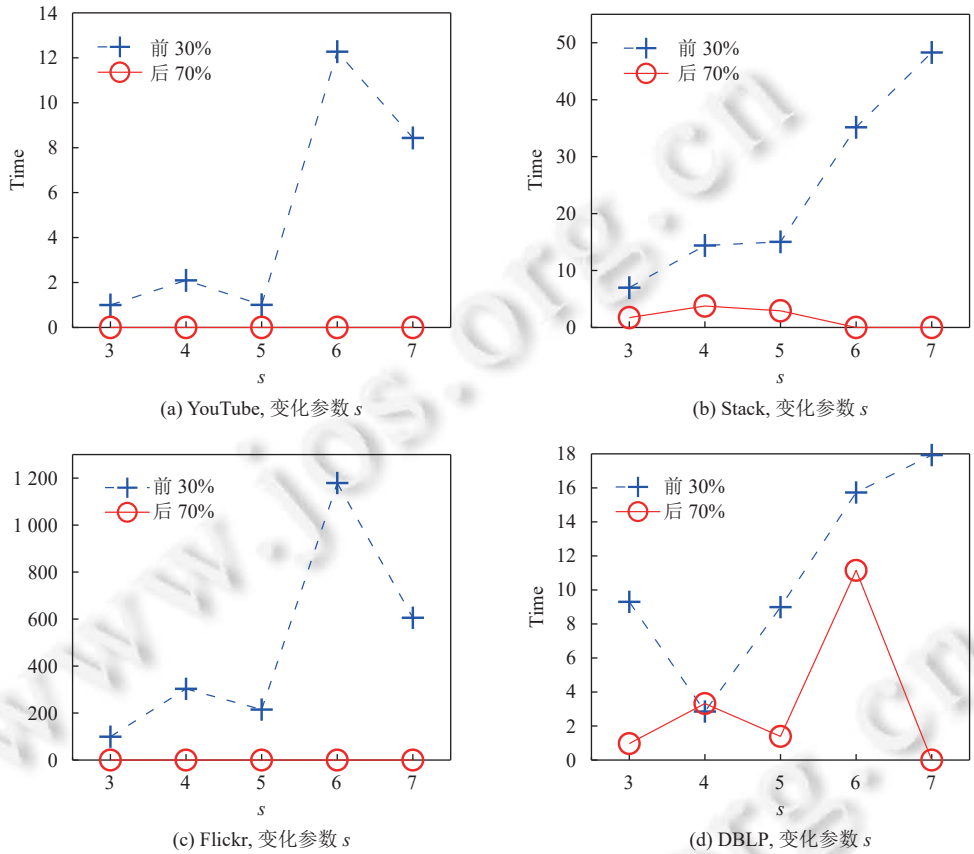
5.2.2　参数 s 对算法运行时间的影响

图 9 展示了变化参数 s 时, 4 个真实数据集上 Base 算法的 CPU 运行时间. 每个图中的两组曲线表示分别选取度数处在前 30% 的点和度数处在后 70% 的点的平均实验结果.

根据实验结果可以看出, 在图 9(a)、(c), 对于度数处在前 30% 的点, Base 算法的运行时间整体上都呈现出先上升到一个峰值而后回落的趋势. 这是因为在一个窗口中, 算法运行时间的变化曲线存在着拐点, 在拐点的前后, 运行时间随参数 s 的变化趋势不同: 在参数 s 没有抵达拐点时, 随着 s 增大, 对社区进行迭代查找和访问的轮次增加, 运行时间变长; 在 s 超过拐点之后, 随着 s 增大, 候选顶点集和候选终点的数量和规模变少, 查找算法的运行时间变短. 所以运行时间总体上会出现先增后降的趋势. 在图 9(a)、(c) 上, 当 s=5 时, 运行时间稍微下降, 这是由随机采样的偶然性所造成的. 在图 9(b), 对于度数处在前 30% 的点, 运行时间始终保持着上升趋势, 这是因为 stack 数据集中拐点对应的参数 s 较大, 而实验选取的 s 范围较小, 没有到达拐点, 窗口中仍存在着较多的社区候选顶点集和候选终点社区, 所以随着 s 的上升, 曲线没有出现回落现象. 在图 9(d), 对于度数处在前 30% 的点, Base 算法的运行时间呈现出先下降后上升的特点, 这是由 DBLP 数据集本身的特点所决定的, 在 DBLP 网络中, 论文作者们的合作关系有短期合作和长期合作的区别, 导致在 s=4 前算法的运行时间下降, 在 s=4 之后持续上升.

在图 9(a)、(c) 上, 对度数处在后 70% 的点, 由于 YouTube 和 Flickr 数据集较为稀疏, 故均未找到持续扩展社

区序列, 因此运行时间为 0. 在图 9(b) 上, 由于 Stack 数据集比较稠密, 对于度数处在后 70% 的查询顶点, 快照中仍存在较多候选社区, 故在 s=6 之前仍能查找到持续扩展社区. 同时, 相比于度数处在前 30% 的点, 对度数处在后 70% 的查询点, 由于快照中包含该类查询点的候选社区数量和规模较小, 因此算法运行时间减少. 在图 9(d) 上, 对于度数处在后 70% 的点, 算法仍能查找到持续增长序列, 这是因为 DBLP 数据集中顶点之间联系紧密, 因此即便对后 70% 的点随机采样, 快照中仍存在包含查询点的候选社区, 算法仍能查找到持续扩展社区序列, 算法的运行时间整体上相对于前 30% 的点有所下降, 但下降幅度比其他 3 个数据集小.



(a) YouTube, 变化参数 s　(b) Stack, 变化参数 s

(c) Flickr, 变化参数 s　(d) DBLP, 变化参数 s

图 9　真实数据集上 Base 算法的 CPU 运行时间, 变化参数 s

5.2.3　参数 τ 对算法运行时间的影响

图 10 展示了变化参数 τ 时, 4 个真实数据集上 Base 算法的 CPU 运行时间. 每个图中的两组曲线分别表示选取度数处在前 30% 的点和度数处在后 70% 的点的实验结果.

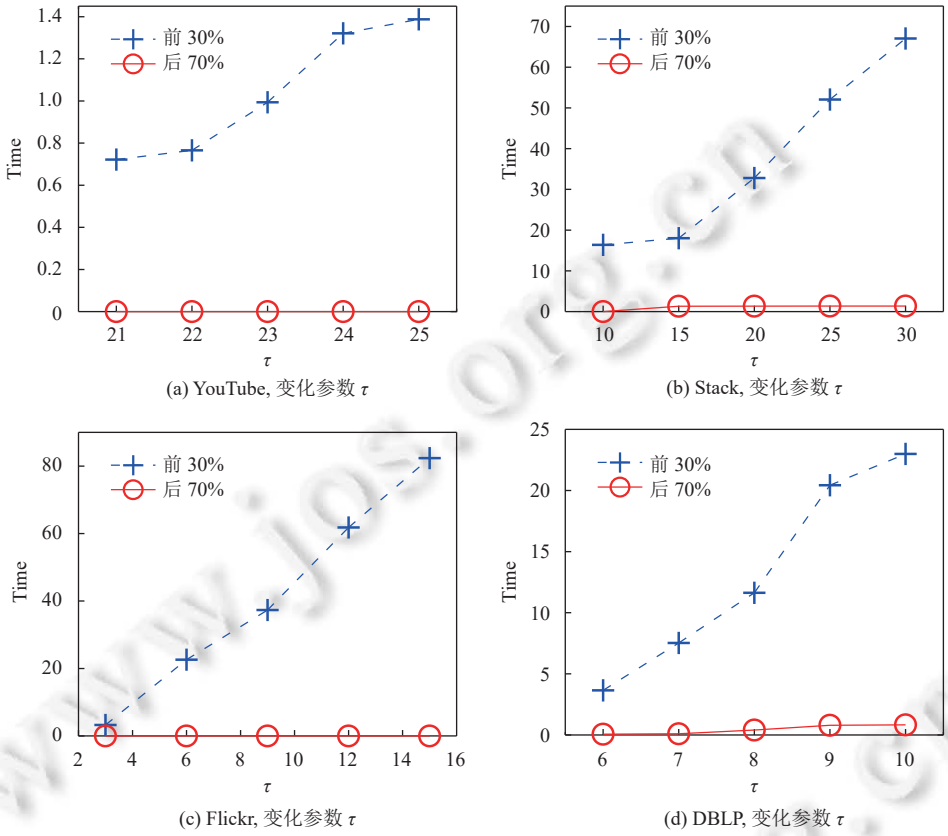
总体而言, 参数 τ 越大, 持续扩展社区序列查找算法的运行时间越长. 对于度数处在前 30% 的点, 在所有数据集上, 随着 τ 的增大, 算法运行时间都会逐渐增大, 这是因为窗口中包含的快照数目变多, 导致包含查询点的候选社区数量变多, 因此算法用于正向计算社区候选顶点集以及反向回溯查找社区序列的时间都会增多.

对于度数处在后 70% 的点, 在 YouTube 和 Flickr 数据集上, 随着 τ 增大, 算法运行时间始终为 0, 这是因为对于度数较低的查询点, 不存在包含这些查询点的候选社区, 因此正向查找候选顶点集和反向回溯查找社区序列都不耗时. 在 Stack 和 DBLP 数据集上, 当 τ 增大到一定数值时, 算法的运行时间轻微上升, 这是因为对于度数较低的查询点, 存在极少量包含这些查询点的候选社区, 但使用很短的时间就可以找到查询结果.

5.2.4　提早终止策略对算法运行时间的优化效果

图 11 展示了 4 个真实数据集上 Base 和 Base+Early 两种算法的 CPU 运行时间, 来测试提早终止策略优化运

行时间的效果, 在默认设置下, 首先对度数排在 10 个不同区间范围的顶点, 分别测量持续扩展序列的查找时间, 然后对每个区间上的结果取平均值. 对于每种算法, 我们分别采用了顶点度数 (Deg) 和 PageRank 值作为顶点权值进行实验, 以测试顶点间的局部连接关系 (Deg) 和全局连接关系 (PageRank) 是否对提早终止策略的剪枝效果有不同影响.



(a) YouTube, 变化参数 τ　(b) Stack, 变化参数 τ

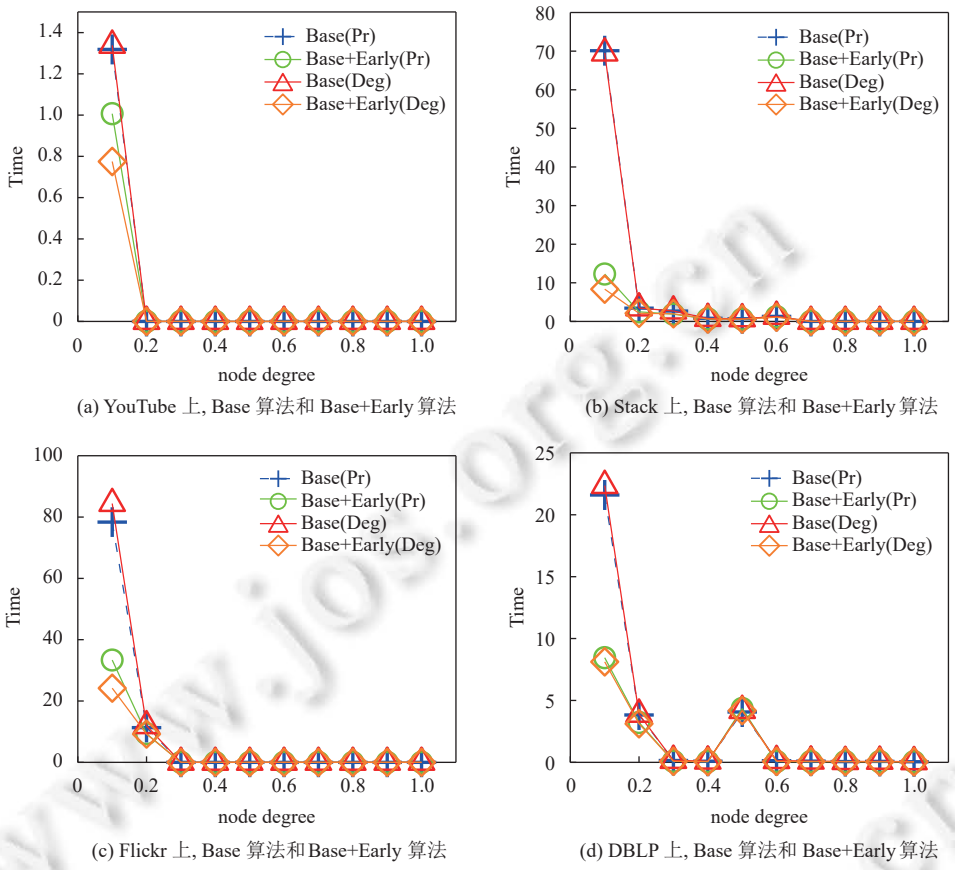
(c) Flickr, 变化参数 τ　(d) DBLP, 变化参数 τ

图 10　真实数据集上 Base 算法的 CPU 运行时间, 变化参数 τ

首先, 由图 11 可以看出, 当节点度数处在前 30% 的区间时, Base+Early 算法在各个数据集上的运行时间都比 Base 算法大幅减少, 说明我们的提早终止策略确实是有效的, 其中 YouTube 数据集由于自身的稀疏性, 只在前 20% 的区间上优化效果较为明显. 而当节点度数处在后 70% 的区间时, 效果则并不是很明显, 也就是说, 在查询点的度数较大时, 我们提出的剪枝策略效果更好. 这是因为对于度数越大的查询点, 包含该类查询点的三角连通 k-truss 社区的规模就越大, 计算这些社区需要更多的时间, 那么在采用提早终止策略后, 就可以避免对部分大规模社区的重复计算, 所以剪枝效果更明显. 此外, 对于度数较大的查询点, 包含查询点的社区有可能处在多条不同的序列当中, 在这种情况下, 提早终止策略能够提前终止对更多持续扩展社区序列的查找, 因此时间优化效果更好. 此外, 无论采用哪种算法, 算法的运行时间大体上都随着顶点度数的减小而下降, 并且对于度数排在后 70% 的查询顶点, 几乎都找不到持续扩展社区序列, 查找时间接近于 0. 特别地, 在 DBLP 数据集上, 在 40%–50% 的区间上, 仍能查找到结果, 这是由 DBLP 数据集上对查询点随机采样的偶然性所造成的.

其次, 由图 11 可以看出, 顶点度数 (Deg) 和 PageRank 分别作为顶点权值时, 对提早终止策略的剪枝效果有不同程度的影响. 总体来说, 数据集中顶点的局部连接较稀疏时 (如 YouTube 和 Flickr 数据集), 提早终止策略对 Deg

的剪枝效果比对 PageRank 的剪枝效果好一些; 而当数据集中顶点的局部连接较稠密时 (如 Stack 和 DBLP 数据集), 提早终止策略对 Deg 和 PageRank 的剪枝效果没有较大差异.



(a) YouTube 上, Base 算法和 Base+Early 算法

(b) Stack 上, Base 算法和 Base+Early 算法

(c) Flickr 上, Base 算法和 Base+Early 算法

(d) DBLP 上, Base 算法和 Base+Early 算法

图 11　真实数据集上 Base 算法和 Base+Early 算法的 CPU 运行时间 (顶点权值为 Pr 或 Deg)

5.2.5　TCP 索引压缩策略对算法占用空间的优化效果

图 12 展示了 4 个真实数据集上 Base 和 Base+Comp 两种算法运行时构建的 TCP 索引的边数, 来测试 TCP 索引优化策略优化算法占用内存的效果.

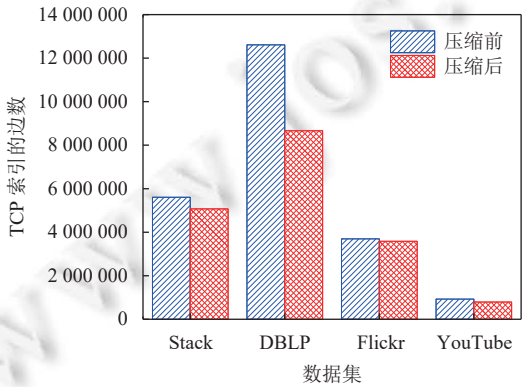


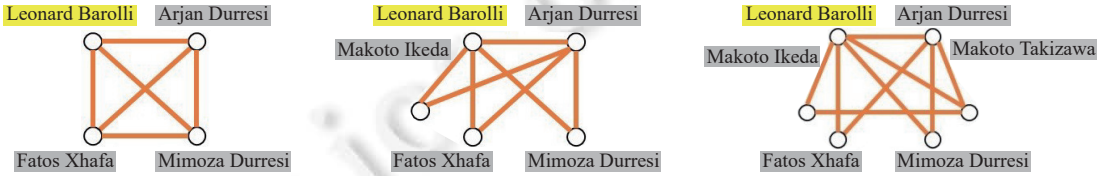
图 12　在各个数据集上 TCP 压缩方法的效果

由图 12 中可以看出, 在 4 个数据集上, 在采用了 TCP 索引压缩的策略后, TCP 索引结构所包含的边数都有下降, 验证了我们所提出的 TCP 索引压缩策略的有效性, TCP 索引压缩过程能将图上存在的单次点删除, 使得 TCP 索引结构得到有效的压缩. 其中, 在 DBLP 数据集上, TCP 索引的优化效果尤其明显, 这是由 DBLP 本身的特点所决定的, 科研团队的合作关系随着年份变化、行业波动、人员更替、技术迭代等原因, 会出现明显的变动, 从而导致窗口中单次点的大量存在, 因此在使用 TCP 索引压缩策略后, 对 TCP 索引结构的优化效果更明显. 可以考虑将 TCP 压缩策略的删除对象从单次点推广到只出现在两张快照上的点, 以及在给定具体的支持度阈值 s 的情况下出现次数小于 $s-1$ 的点, 从而进一步优化压缩效果, 这是我们未来的工作之一.

5.3 案例分析

在 DBLP 数据集上, 我们选择包含 2005–2014 年间共 10 张快照的窗口进行如下案例分析.

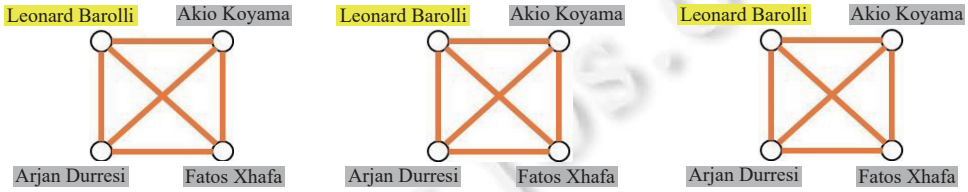
在运行 Base 算法查找 qLEC 持续扩展社区时, 设参数 $s=3$, $k=3$, $q=$“Leonard Barolli”. 算法共查询到 7 个持续扩展社区序列, 其中规模最小的社区包含 3 个顶点, 规模最大的社区包含 10 个顶点. 图 13 中展示了其中一个社区随时间变化的情况, 在 2007 年, 包含 Leonard Barolli 在内的 4 人组成了快照上的科研团队; 在 2008 年, Makoto Ikeda 加入这个团队, 并与 Leonard Barolli 和 Arjan Durresi 建立了合作关系; 在 2012 年, Makoto Takizawa 的加入使得这个团队的规模进一步增大.



(a) 2007 年 (b) 2008 年 (c) 2012 年

图 13 2007–2012 年 DBLP 上包含 Leonard Barolli 的 qLEC 社区序列示例

在运行 MPC 社区[14] (一段时间内周期性出现的社区) 查找算法时, 设参数 $\theta=3$, $k=3$, $q=$“Leonard Barolli”, 即查找包含了 Leonard Barolli, 并且周期性出现至少 3 次的社区 (社区用 k-团模型刻画), 算法共查找到了 3 个 (3, 3)-MPC 社区结果, 其中每个社区都是规模为 4 的团. 图 14 展示了其中的一个社区结果: Leonard Barolli 和其余 3 名学者组成的科研团队在 2008 年、2009 年、2010 年周期性地出现了 3 次. 相比于 MPC 社区, 我们的 qLEC 社区序列数量更多、社区规模更大. 这是因为 MPC 模型要求社区中的顶点满足 k-团的结构定义, 约束条件较为严格, 而 qLEC 模型采用了相对松弛的 k-truss 模型来定义社区结构; 此外, 时间维度上, MPC 模型要求社区周期性地出现, 而我们的 qLEC 模型只要求社区在一段时间内呈现扩大趋势, 并没有强调周期性地出现.



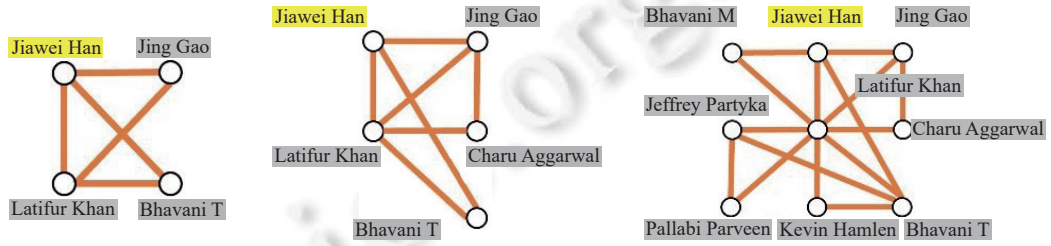
(a) 2008 年 (b) 2009 年 (c) 2010 年

图 14 2008–2010 年 DBLP 上包含 Leonard Barolli 的 MPC 周期性社区示例

在运行 MBC 社区[25] (一段时间内爆发性增长的社区) 查找算法时, 设参数 $\theta=3$, $l=3$, $q=$“Leonard Barolli”, 即查找包含了 Leonard Barolli, 并且瞬时爆发性增长的社区 (社区用 k-core 模型刻画), 算法只查找到了一个 (3, 3)-MBC 社区结果, 该社区包含 1 910 个顶点, 且其规模不再随着时间的推移继续增长. MBC 模型是基于 k-core 定义的社区, 而 qLEC 模型是基于 k-truss 定义的社区, 相比于 k-core, k-truss 不仅要求社区成员间联系紧密, 而且要求成员间的联系越稳定越好. 此外, MBC 模型只要求社区中的顶点在多张快照的并集上形成 k-core 即可, 并不要求

它们在多张快照上分别形成 k -core, 而 qLEC 模型要求社区中的顶点在多张快照上分别形成 k -truss. 同时, 从时间维度上, MBC 模型只关注瞬时爆发性增长的社区, 并不关注一段时间内持续扩大的社区; 而 qLEC 模型关注的是一段时间内规模持续扩大的社区. 当希望查找多个由核心成员组成的精简、稳定、持续发展的科研团队时, 采用我们的 qLEC 模型更好.

接下来, 为了测试 TCP 索引压缩技术是否会影响查询结果的准确度, 设参数 s =3, k =3, q = “Jiawei Han”, 分别运行了我们的 Base 算法和 Base+Comp 算法, 其中 Base+Comp 算法用 6.842 8 s 得到了 9 个包含 Jiawei Han 在内、长度为 3 的 qLEC-3-truss 持续扩展社区结果, 且两种算法输出的实验结果基本无差异, 这说明使用 TCP 索引压缩技术对最终查询结果几乎无影响. 图 15 给出了一个包含 Jiawei Han 的 qLEC 社区序列示例. 可以看出, 在 2009 年, 这个团队由 Jiawei Han, Jing Gao, Latifur Khan, Bhavani T 组成, 之后在 2010 年, 加入了 Charu Aggarwal, 接着在 2011 年, 这个维持着紧密合作关系的团队又进一步扩展到了更大的规模. 我们手动查询了图 15 中出现的所有作者都具有非常高的学术影响力, 且在此年间一直保持着密切的合作关系, 产出了大量优秀成果. 这说明通过我们提出的模型和算法, 确实可以找到 DBLP 上不断发展、持续高产出的科研团队.



(a) 2009 年 (b) 2010 年 (c) 2011 年

图 15 2009–2011 年 DBLP 上包含 Jiawei Han 的 qLEC 社区序列示例

6 总 结

动态信息网中社区搜索的问题已成为图数据挖掘领域的研究热点, 具有广泛的应用价值. 本文关注动态信息网中持续扩展社区序列的查找问题, 提出了动态信息网中基于查询点 q 的持续扩展社区序列 (qLEC) 模型, 设计了一个持续扩展社区序列两阶段查找算法, 并给出基于提早终止策略的时间优化和基于 TCP 索引压缩技术的空间优化方法. 大量真实世界动态信息网上的实验证实了相比于现有动态社区模型, qLEC 模型具有特定的应用价值, 同时验证了本文的两阶段查找算法能够有效找到 qLEC 模型所刻画的持续扩展社区序列, 还验证了本文的优化策略显著降低了两阶段查找算法的时间和空间开销. 未来的研究包括: (1) 设计其他时空优化策略进一步减少动态信息网中持续扩展社区序列查找算法的时空开销; (2) 使用 k -truss 以外的社区模型定义持续扩展社区序列中的社区, 结合图神经网络发现语义更加丰富的持续扩展社区序列; (3) 改进持续扩展社区序列查找算法, 将其部署在分布式集群环境, 从而能够快速高效地处理更大规模的动态信息网数据.

References:

[1] Zhu R, Zou ZN, Li JZ. Fast diversified coherent core search on multi-layer graphs. The VLDB Journal, 2019, 28(4): 597–622. [doi: 10.1007/s00778-019-00542-3]

[2] Huang X, Cheng H, Qin L, Tian WT, Yu JX. Querying k-truss community in large and dynamic graphs. In: Proc. of the 2014 ACM SIGMOD Int’l Conf. on Management of Data. Snowbird: Association for Computing Machinery, 2014. 1311–1322. [doi: 10.1145/2588555.2610495]

[3] Wang XR, Gao H, Wang JB, Yue TB, Li JZ. Detecting top-k active inter-community jumpers in dynamic information networks. In: Proc. of the 23rd Int’l Conf. on Database Systems for Advanced Applications. Gold Coast: Springer, 2018. 538–546. [doi: 10.1007/978-3-319-91452-7_35]

[4] Liu XM, Ge TJ, Wu YH. Finding densest lasting subgraphs in dynamic graphs: A stochastic approach. In: Proc. of the 35th Int’l Conf. on

Data Engineering (ICDE). Macao: IEEE, 2019. 782–793. [doi: 10.1109/ICDE.2019.00075]

[5] Li YH, Wang M, Li GH, Luo CY, Du XK. Multi-rule shortest path query algorithm in time-dependent networks. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 3115–3136 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6285.htm [doi: 10.13328/j.cnki.jos.006285]

[6] Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the Web. Stanford InfoLab, 1999. [doi: 10.1007/978-3-319-08789-4_10]

[7] Cui WY, Xiao YH, Wang HX, Wang W. Local search of communities in large graphs. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. Snowbird: Association for Computing Machinery, 2014. 991–1002. [doi: 10.1145/2588555.2612179]

[8] 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 with English abstract). http://www.jos.org.cn/1000-9825/7071.htm [doi: 10.13328/j.cnki.jos.007071]

[9] Zhang QF, Liu SX. Efficient exact minimum *k*-core search in real-world graphs. In: Proc. of the 32nd ACM Int'l Conf. on Information and Knowledge Management. Birmingham: Association for Computing Machinery, 2023. 3391–3401. [doi: 10.1145/3583780.3614861]

[10] Ahmad A, Yuan LH, Yan D, Guo GM, Chen JY, Zhang CC. Accelerating *k*-core decomposition by a GPU. In: Proc. of the 39th Int'l Conf. on Data Engineering (ICDE). Anaheim: IEEE, 2023. 1818–1831. [doi: 10.1109/ICDE55515.2023.00142]

[11] Yang YX, Fang YX, Lin XM, Zhang WJ. Effective and efficient truss computation over large heterogeneous information networks. In: Proc. of the 36th Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 901–912. [doi: 10.1109/ICDE48307.2020.00083]

[12] Wu YB, Jin RM, Li J, Zhang X. Robust local community detection: On free rider effect and its elimination. Proc. of the VLDB Endowment, 2015, 8(7): 798–809. [doi: 10.14778/2752939.2752948]

[13] Cui WY, Xiao YH, Wang HX, Lu YQ, Wang W. Online search of overlapping communities. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: Association for Computing Machinery, 2013. 277–288. [doi: 10.1145/2463676.2463722]

[14] Qin HC, Li RH, Wang GR, Qin L, Cheng YR, Yuan Y. Mining periodic cliques in temporal networks. In: Proc. of the 35th Int'l Conf. on Data Engineering (ICDE). Macao: IEEE, 2019. 1130–1141. [doi: 10.1109/ICDE.2019.00104]

[15] Zhang F, Zhang Y, Qin L, Zhang WJ, Lin XM. When engagement meets similarity: Efficient (*k*, *r*)-core computation on social networks. Proc. of the VLDB Endowment, 2017, 10(10): 998–1009. [doi: 10.14778/3115404.3115406]

[16] Ugander J, Backstrom L, Marlow C, Kleinberg J. Structural diversity in social contagion. Proc. of the National Academy of Sciences of the United States of America, 2012, 109(16): 5962–5966. [doi: 10.1073/pnas.1116502109]

[17] Batagelj V, Zaveršnik M. An O(*m*) algorithm for cores decomposition of networks. arXiv:cs/0310049, 2003.

[18] Cai TT, Li JX, Haldar NAH, Mian A, Yearwood J, Sellis T. Anchored vertex exploration for community engagement in social networks. In: Proc. of the 36th Int'l Conf. on Data Engineering. Dallas: IEEE, 2020. 409–420. [doi: 10.1109/ICDE48307.2020.00042]

[19] Ouvrard X. Hypergraphs: An introduction and review. arXiv:2002.05014, 2020.

[20] Wang J, Cheng J. Truss decomposition in massive networks. Proc. of the VLDB Endowment, 2012, 5(9): 812–823. [doi: 10.14778/2311906.2311909]

[21] Yuan L, Qin L, Lin XM, Chang LJ, Zhang WJ. Diversified top-*k* clique search. The VLDB Journal, 2016, 25(2): 171–196. [doi: 10.1007/s00778-015-0408-z]

[22] Fang YX, Yang YX, Zhang WJ, Lin XM, Cao X. Effective and efficient community search over large heterogeneous information networks. Proc. of the VLDB Endowment, 2020, 13(6): 854–867. [doi: 10.14778/3380750.338075]

[23] Lu Z, Zhu YY, Zhong M, Yu JX. On time-optimal (*k*, *p*)-core community search in dynamic graphs. In: Proc. of the 38th Int'l Conf. on Data Engineering (ICDE). Kuala Lumpur: IEEE, 2022. 1396–1407. [doi: 10.1109/ICDE53745.2022.00109]

[24] Jiang YQ, Fang YX, Ma CH, Cao X, Li CS. Effective community search over large star-schema heterogeneous information networks. Proc. of the VLDB Endowment, 2022, 15(11): 2307–2320. [doi: 10.14778/3551793.3551795]

[25] Qin HC, Li RH, Yuan Y, Wang GR, Qin L, Zhang ZW. Mining bursting core in large temporal graphs. Proc. of the VLDB Endowment, 2022, 15(13): 3911–3923. [doi: 10.14778/3565838.3565845]

[26] Zhang F, Zhang WJ, Zhang Y, Qin L, Lin XM. OLAK: An efficient algorithm to prevent unraveling in social networks. Proc. of the VLDB Endowment, 2017, 10(6): 649–660. [doi: 10.14778/3055330.3055332]

[27] Yu M, Wen D, Qin L, Zhang Y, Zhang WJ, Lin XM. On querying historical *k*-cores. Proc. of the VLDB Endowment, 2021, 14(11): 2033–2045. [doi: 10.14778/3476249.3476260]

[28] Lin Z, Zhang F, Lin XM, Zhang WJ, Tian ZH. Hierarchical core maintenance on large dynamic graphs. Proc. of the VLDB Endowment, 2021, 14(5): 757–770. [doi: 10.14778/3446095.3446099]

[29] Qian CW. Updating strategy for *k*-core hierarchy index tree on dynamic graph [MS. Thesis]. Shanghai: Donghua University, 2022 (in

Chinese with English abstract). [doi: 10.27012/d.cnki.gdhuu.2022.000788]
[30] Duan XY. Research on methods of dynamic social network community detection based on graph embedding [MS. Thesis]. Xuzhou: China University of Mining and Technology, 2022 (in Chinese with English abstract). [doi: 10.27623/d.cnki.gzkyu.2022.000971]
[31] Liu L, Kang Z, Ruan JJ, He XX. Multilayer graph contrastive clustering network. Information Sciences, 2022, 613: 256–267. [doi: 10.1016/J.INS.2022.09.042]
[32] Liu WY, Suzumura T, Ji HY, Hu GM. Finding overlapping communities in multilayer networks. PLoS One, 2018, 13(4): e0188747. [doi: 10.1371/journal.pone.0188747]
[33] Pei J, Jiang DX, Zhang AD. On mining cross-graph quasi-cliques. In: Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery in Data Mining. Chicago: Association for Computing Machinery, 2005. 228–238. [doi: 10.1145/1081870.1081898]
[34] Boden B, Günnemann S, Hoffmann H, Seidl T. MiMAG: Mining coherent subgraphs in multi-layer graphs with edge labels. Knowledge and Information Systems, 2017, 50(2): 417–446. [doi: 10.1007/s10115-016-0949-5]
[35] Zeng ZP, Wang JX, Zhou LZ, Karypis G. Coherent closed quasi-clique discovery from large dense graph databases. In: Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Philadelphia: Association for Computing Machinery, 2016. 797–802. [doi: 10.1145/1150402.1150506]
[36] Agarwal MK, Ramamritham K, Bhide M. Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments. Proc. of the VLDB Endowment, 2012, 5(10): 980–991. [doi: 10.14778/2336664.2336671]
[37] Wang RQ, Wang CX, Liu GX. A novel graph clustering method with a greedy heuristic search algorithm for mining protein complexes from dynamic and static PPI networks. Information Sciences, 2020, 522: 275–298. [doi: 10.1016/j.ins.2020.02.063]
[38] Jiang YN, Xia H. Adversarial attacks against dynamic graph neural networks via node injection. High-confidence Computing, 2024, 4(1): 100185. [doi: 10.1016/j.hcc.2023.100185]
[39] Linghu QY, Zhang F, Lin XM, Zhang WJ, Zhang Y. Towards user engagement dynamics in social networks. arXiv:2110.12193, 2021.
[40] Dakiche N, Tayeb FBS, Slimani Y, Benatchba K. Tracking community evolution in social networks: A survey. Information Processing & Management, 2019, 56(3): 1084–1102. [doi: 10.1016/j.ipm.2018.03.005]
[41] Wang ZX, Li ZC, Yuan G, Sun YL, Rui XB, Xiang XG. Tracking the evolution of overlapping communities in dynamic social networks. Knowledge-based Systems, 2018, 157: 81–97. [doi: 10.1016/j.knosys.2018.05.026]
[42] Song HJ, Kim T, Hwang YW, Seo D, Lee IY. A study on dynamic group signature scheme with threshold traceability for blockchain. High-confidence Computing, 2024, 4(2): 100163. [doi: 10.1016/j.hcc.2023.100163]
[43] Li JX, Wang XJ, Deng K, Yang XC, Sellis T, Yu JX. Most influential community search over large social networks. In: Proc. of the 33rd Int'l Conf. on Data Engineering (ICDE). San Diego: IEEE. 2017. 871–882. [doi: 10.1109/ICDE.2017.136]
[44] Cohen J. Trusses: Cohesive subgraphs for social network analysis. Fort Meade: National Security Agency, 2008. 1–29.
[45] YouTube dataset. 2024. http://konect.cc/networks/youtube-u-growth/
[46] Stack dataset. 2024. https://snap.stanford.edu/data/sx-stackoverflow.html
[47] Flickr dataset. 2024. http://konect.cc/networks/flickr-growth/
[48] DBLP dataset. 2024. http://konect.cc/networks/dblp_coauthor/

附中文参考文献:

[5] 李艳红, 王猛, 李国徽, 罗昌银, 杜小坤. 动态网络中多规则的最短路径查询算法. 软件学报, 2022, 33(8): 3115–3136. http://www.jos.org.cn/1000-9825/6285.htm [doi: 10.13328/j.cnki.jos.006285]
[8] 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 [doi: 10.13328/j.cnki.jos.007071]
[29] 钱成威. 面向动态图的 k-core 索引维护算法研究 [硕士学位论文]. 上海: 东华大学, 2022. [doi: 10.27012/d.cnki.gdhuu.2022.000788]
[30] 端祥宇. 基于图嵌入的动态社交网络社区发现方法研究 [硕士学位论文]. 徐州: 中国矿业大学, 2022. [doi: 10.27623/d.cnki.gzkyu.2022.000971]

王芯蕊(1993－), 女, 博士, 助理教授, CCF 专业会员, 主要研究领域为数据挖掘, 大数据.

姚越(1998－), 男, 硕士生, 主要研究领域为图数据库挖掘.

于东晓(1984－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据挖掘, 边缘智能.

高宏(1966－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据挖掘, 大数据, 物联网.

成秀珍(1970－), 女, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数链融合, 信息安全.