软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007576] [CSTR: 32375.14.jos.007576]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

时序图数据中近似环路的检测方法*

姜 涛[1], 李战怀[2]

[1](河南财经政法大学 计算机与信息工程学院 (软件学院), 河南 郑州 450046)
[2](西北工业大学 计算机学院, 陕西 西安 710129)
通信作者: 姜涛, E-mail: jiangtao@mail.nwpu.edu.cn

摘 要: 时序图是一类节点之间交互时带有时间戳的图结构, 其比静态图具有更多的建模优势, 比如可以发现在一定时间区间内的洗钱、刷单、股权关系、金融欺诈、循环担保等行为. 环路是对时序图中的路径组成回路的行为建模. 现有的时序环路检测或挖掘方法大多数关注时间非递减的完全环路检测, 忽略了时间处于一定区间内的近似环路分析与发现, 发现此类近似环路可以检测出一些作弊手段更强的欺诈行为. 针对处于一定时间区间内, 事实上已经出现环路, 但在单一源数据上未完全展示出环路的近似环路发现问题, 提出一种基于深度优先搜索的近似环路检测方法, 简称基线方法 (Baseline). 首先在每个窗口内挖掘时间维度上满足非递减顺序的边组成的完全环路, 接着将其中符合一定特征的节点分别作为近似环路的起止点, 并在后续窗口中挖掘处于一定时间区间内的边组成的路径, 即时间区间近似环路. 针对基线方法存在的问题, 提出一种优化的近似环路检测方法, 简称优化方法 (Improved). 首先利用节点的活跃度来提升起止点的可能性, 接着使用活跃路径和热点来优化索引的特征, 最后运用起止点到热点的双向搜索与连接来加快检测速度. 在真实数据和人工数据上进行的大量实验证明了所提方法的高效性与有效性.

关键词: 时序图; 近似环路; 时间区间; 活跃度; 热点
中图法分类号: TP311

中文引用格式: 姜涛, 李战怀. 时序图数据中近似环路的检测方法. 软件学报. http://www.jos.org.cn/1000-9825/7576.htm
英文引用格式: Jiang T, Li ZH. Approximate Cycles Detection Method in Temporal Graph Data. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7576.htm

Approximate Cycles Detection Method in Temporal Graph Data

JIANG Tao[1], LI Zhan-Huai[2]

[1](School of Computer and Information Engineering (School of Software), Henan University of Economics and Law, Zhengzhou 450046, China)
[2](School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: As a kind of graph structure with timestamps when nodes interact with each other, temporal graphs have more modeling advantages than static graphs. For example, they can detect money laundering, order brushing, equity relationships, financial fraud, and circular guarantees within a certain time interval. The cycle is the modeling of the behavior that forms a cycle in a temporal graph. Existing temporal cycle detection or mining methods mostly focus on the detection of non-decreasing complete cycles in time, but overlook the analysis and discovery of approximate cycles within a certain time interval. The discovery of such approximate cycles can detect fraudulent behavior with stronger cheating techniques. To address the problem of discovering approximate cycles that have already appeared within a certain time interval but are not fully displayed in a single source of data, this study first proposes an approximate cycle detection method based on the depth-first search, which is referred to as the baseline method (Baseline). It first mines complete cycles

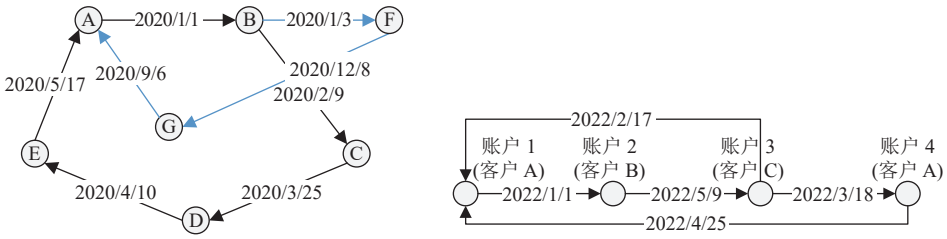
* 基金项目: 国家自然科学基金 (61702161); 河南省重点研发与推广专项 (科技攻关项目) (252102210128, 212102210386); 河南省高等学校重点科研项目 (24A520001); 河南财经政法大学校级研究课题 (国家一般项目培育项目) (24HNCDXJ54)
收稿时间: 2024-12-15; 修改时间: 2025-03-19, 2025-11-05; 采用时间: 2025-11-27; jos 在线出版时间: 2026-02-11

composed of edges satisfying non-decreasing order in each window, and then employs nodes that meet certain criteria as the start and end points of approximate cycles. In the subsequent windows, paths composed of edges within a certain time interval are mined, namely time-interval approximate cycles. To address the problems of Baseline, this study subsequently proposes an improved method for approximate cycle detection, referred to as the improved method (Improved). It first utilizes the node activity to enhance the possibility of start and end points, then improves the index features by adopting active paths and hotspots, and finally accelerates detection by employing the bidirectional search and connection from start and end points to hotspots. Extensive experiments on real and synthetic data demonstrate the efficiency and effectiveness of the proposed method.

Key words: temporal graph; approximate cycle (AC); time interval; activation; hot point

随着计算机、互联网、物联网等技术的快速发展, 图已经广泛应用于社交网络、金融交易网络、生物信息网络等网络的建模与分析中. 在诸多图中, 点表示实体, 边表示实体间的关系. 然而, 为了简化分析, 这些图中经常缺少一种类型的信息: 事实上, 一种关系发生在特定的时间并持续一定的时间. 从形式上, 该图可以建模为时序图[1], 即 (起点 u, 终点 v, 时间戳 t, 持续时间 λ). 由于时间信息的存在, 起点和终点间可以出现多条边, 且边之间也有先后顺序关系, 这是时序图的新特点[2].

时序图可以用来建模和分析许多与时间有关的活动[1–3], 比如表示贷款担保的先后关系[3], 见图 1(a). 图 1(a) 中有多个担保关系, 经过检测发现客户 A、B、C、D、E 之间有一个担保环路 A→B→C→D→E→A (黑色箭头). 在贷款发放前, 通过实时担保圈链查询, 可以帮助银行了解每笔贷款申请的合法性, 保护贷款, 避免大型连锁效应. 还可以表示银行账户间的洗钱行为[3], 见图 1(b). 非法洗钱分子从一个账户出发, 经过层层转账, 最终将钱转回自己账户. 路径上的转账数据显然会包含一些伪装数据, 比如扣除手续费. 经过分析发现, 上述两种应用场景的本质基本上是一样的, 都是从时序图中检测环路, 所以该研究在当前具有重要的意义.



(a) 银行客户贷款担保关系中的环路 (b) 具有一定时间跨度的洗钱环路

图 1 结合实际应用的时序图中的环路举例

尽管研究者在环路检测方面进行了广泛研究, 但是大多数局限于静态图[4–6], 对时序图数据中环路检测的研究相对较少[7–10], 且现有的环路检测方法存在一些不足: (1) 现有时序图执行环路枚举算法时, 往往要求时序是非递减的. 然而, 现实中节点间的交互并不一定是非递减的 (例如图 1(a) 中担保环路 A→B→ F→G→A, 图 1(b) 中两个转账环路), 甚至是随机的, 但是总体又处于一定时间区间内. 因而, 对时序环路的要求显然应该从严格非递减扩展到处于一定时间区间内, 这样有利于发现更隐蔽的时序环路. (2) 无论是静态图还是时序图, 现有环路枚举方法都假设环路的起点和终点是同一点, 这样无形中减少了环路的数量, 也就会错失更重要的信息. 现实中环路的起点和终点并不一定是同一点, 可能是同一族群 (比如图 1(b) 中账户 1 和账户 4 属于同一个客户 A), 这样通过挖掘近似环路有利于发现先前缺失的时序环路. 因此, 目前迫切需要支持时间区间近似环路检测的方法. (3) 现有研究大多检测的是完全环路, 一般是新增一条连边时, 检测是否构成环路, 而本研究是检测所有环路, 并不局限于一个起止点, 需要找到高概率的起止点, 所以对近似环路中起止点聚类是一个需要做的工作.

定义 1. 起止点聚类. 在时序图 $G=(V, E, T)$ 中, 其中 V 是节点集合, $E\subseteq V\times V\times T$ 是带时间戳的有向边集合, T 是时间戳集合. 给定阈值 θ, 起止点聚类是从构成环路的节点中筛选出的节点集合. 阈值 θ 的设定基于以下理论依据.

统计特性: 阈值 θ 可根据数据的统计特性 (如均值和标准差) 动态计算. 例如, 计算历史数据的均值 μ 和标准差 σ, 将阈值设为 $\mu+k\sigma$ (如 k=2 对应 95% 置信区间), 以适应数据分布变化. 具体定义如下.

(1) 起点集合 C_s: 从所有节点 V 中筛选出出度超过阈值 θ 的节点, 形成起点集合 C_s:

$$C_s = \{v \in V \mid degr_{\text{out}}(v) > \theta\} \tag{1}$$

其中, $degr_{\text{out}}(v)$ 表示节点 v 的出度.

(2) 终点集合 C_e: 从所有节点 V 中筛选出入度超过阈值 θ 的节点, 形成终点集合 C_e:

$$C_e = \{v \in V \mid degr_{\text{in}}(v) > \theta\} \tag{2}$$

其中, $degr_{\text{in}}(v)$ 表示节点 v 的入度.

(3) 起止点对 $\langle c_s, c_e\rangle$: 从起点集合 C_s 和终点集合 C_e 中分别选择节点, 形成起止点对 $\langle c_s, c_e\rangle$:

$$\langle c_s, c_e\rangle \in C_s \times C_e \tag{3}$$

其中, $c_s \in C_s$ 且 $c_e \in C_e$.

这些从环路中筛选出的节点后续形成欺诈行为的可能性更大, 因此也可以称为高概率起止点聚类. 同时, 与其他节点相比, 这些起止点对在后续检测更隐蔽的非环路欺诈行为时能发挥更重要的作用.

在 IBM 发布的金融交易数据集 (详见 https://ibm.ent.box.com/v/AML-Anti-Money-Laundering-Data) 上的假设性检验结果表明, 通过出入度计算的活跃度能够有效区分正常交易节点和潜在洗钱节点, 节点的活跃度与洗钱活动有着正相关关系. 以账户交易次数为根本, 构建节点的活跃度, 能显著区分洗钱账户与正常账户: 洗钱账户的活跃度平均分值为 4.11, 正常账户只有 1.53, 差距较大. 统计检验显示, 这种差别几乎不可能是运气造成的 (概率不到 1/1000), 而且分值拉开得非常远, 就像两支球队相差 3 个级别. 用活跃度检测洗钱, 命中率比平时高 3 倍, 说明分值越高, 洗钱可能性越大.

定义 2. 近似环路. 在时序图 $G=(V, E, T)$ 中, 其中 V 是节点集合, $E \subseteq V \times V \times T$ 是带时间戳的有向边集合, T 是时间戳集合. 给定时间区间 Δt, 以及从起止点聚类中发现的高概率起止点对 $\langle c_s, c_e\rangle$, 当且仅当满足以下条件时, 路径 $P=(v_1, v_2, \ldots, v_n)$ 被称为时间区间近似环路 (本文检测近似环路).

(1) $v_1=c_s$ 且 $v_n=c_e$, 即路径始于起点 c_s 并终止于终点 c_e.

(2) 对于任意 $i \in [1, n-1]$, 边 $(v_i, v_{i+1}, t_i) \in E$ 满足 $|t_i - t_{i+1}| \leqslant \Delta t$, 即路径上的边在时间上不一定满足严格的非递减顺序, 但任意两个连续边的时间差的绝对值不超过 Δt.

(3) 路径中的每个节点只经过 1 次, 即 $v_i \neq v_j$ 对于所有 $i \neq j$.

条件 (2) 中的 $|t_i - t_{i+1}| \leqslant \Delta t$ 表示路径上的边在时间上不一定满足严格的非递减顺序, 但任意两个连续边的时间差的绝对值不超过 Δt. 具体来说, 对于路径上的任意两个连续的边 (v_i, v_{i+1}, t_i) 和 $(v_{i+1}, v_{i+2}, t_{i+1})$, 有 $|t_i - t_{i+1}| \leqslant \Delta t$, 即 t_i 和 t_{i+1} 的时间差的绝对值可以是正的或负的, 但不超过 Δt, 这允许了在时间上的“跳跃”.

近似环路在真实数据中是存在的, 本研究也通过初步实验挖掘出了部分近似环路. 为验证近似环路确实存在, 在 IBM 发布的金融交易数据集上, 在每个时间窗口内首先挖掘出完整环路, 接着利用符合起止点特征的点作为种子, 在后续窗口可以挖掘出一定数量的近似环路. 具体参数与挖掘效果如图 2 所示, 参数含义见第 2 节.

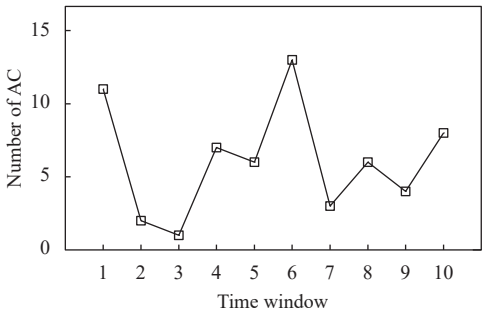


图 2 新概念近似环路 (AC) 在实际中的初步效果 (HI-Small_Trans, $|W|$=2 h, θ=6)

在每个时间窗口内寻找环路, 利用这些聚类后的起止点对, 在后续时间窗口中检测近似环路, 具有实际应用价值. 比如在金融领域, 反洗钱检测使得不法分子在从事一次洗钱之后, 形成更高的警惕, 规避洗钱特征的出现, 所以先前形成的环路在后续行为中很少出现, 但是缺少最后一跳的路径通常还会出现, 而这种行为很可能是在继续从

事洗钱活动. 以图 1(b) 为例, 通常我们获取不到账户 1 和账户 4 之间的关系 (即同属于客户 A). 正是由于存在这个天然的漏洞, 使得洗钱活动变得更加隐蔽, 即不法分子 A 可以经常使用账户 1 经过层层转账向账户 4 洗钱, 且不出现账户 4 向账户 1 的资金回流. 这样, 就可以在不形成时序环路的情况下完成洗钱.

发现近似环路相当于寻找非闭合路径, 这一过程面临以下 3 个主要难点与挑战: (1) 如何找到高概率的起点和终点: 在复杂的时序图数据中, 如何准确识别高概率的起点和终点是一个关键问题. 特别是在涉及犯罪行为 (如洗钱) 的场景中, 犯罪分子在早期可能形成环路, 但随着反侦查意识的增强, 他们可能会通过其他形式规避环路的形成. 此外, 有作案前科的节点再次作案的可能性较大, 但如何从海量数据中筛选出这些高风险节点仍然是一个挑战. (2) 现有方法在遍历过程中保持大量中间结果, 严重影响搜索效率: 现有的环路检测方法在从起点到终点的遍历过程中, 通常会保持大量的中间结果. 这种方法在大规模时序图数据中会导致存储和计算资源的显著增加, 从而严重影响搜索效率. 此外, 当遇到中间出入度同时较大的节点 (本文称为热点) 时, 如何有效管理这些中间结果以避免性能瓶颈是一个亟待解决的问题. (3) 时间区间近似环路不符合严格时间非递减的特性, 影响检测效率: 时间区间近似环路的检测放宽了对时间严格非递减的要求, 这虽然增加了检测的灵活性, 但也带来了更多的中间结果和更复杂的路径组合. 这种放宽条件的方法可能导致检测效率的显著下降, 尤其是在大规模数据集中, 如何在保持检测灵活性的同时提高效率是一个重要的研究问题.

为了发现时序图中的时间区间近似环路, 本文首先提出一种基线方法 (Baseline). 其首先在每个窗口内挖掘时间维度上满足非递减顺序的边组成的完全环路, 接着将其中符合一定特征的节点分别作为近似环路的起止点, 并在后续窗口中使用深度优先搜索 (depth-first search, DFS) 方法挖掘处于一定时间区间内的边组成的近似环路. 由于基线方法存在起止点组合多、中间结果庞大、搜索效率不高等问题, 本文接着提出一种优化方法 (Improved). 其首先利用节点的活跃度来提升起止点的可能性, 接着使用活跃路径和热点来优化索引的特征, 最后运用起止点到热点的双向搜索与连接来加快检测速度.

本文的主要贡献如下.

(1) 提出时间区间近似环路的概念及应用: 定义了时间区间近似环路, 并探讨了其在多个领域的应用, 如金融交易网络中的洗钱检测. 这一概念丰富了现有环路检测研究的内涵, 为相关领域的研究提供了新视角和方法.

(2) 提出一种高概率的起止点发现方法: 针对复杂时序图数据, 设计一种能够挖掘高概率起止点的方法. 通过分析每个窗口中的环路, 识别出符合起点与终点特征的节点, 为后续路径检测提供了可靠的起止点.

(3) 提出一种时间区间内路径发现方法: 设计一种路径发现方法, 能够在时间区间内检测路径, 即使这些路径不符合严格的时间非递减特性. 该方法通过两个方向的搜索来发现近似环路, 并利用轻量级索引和热点分析来有效减少计算量, 提高了检测效率.

(4) 广泛的实验验证: 在真实数据和生成数据上进行了较为全面的实验, 验证所提方法的高效性和有效性. 实验结果表明, 所提方法在不同规模和特征的数据集上基本上能取得较好的性能.

本文第 1 节对相关工作进行回顾与总结. 第 2 节介绍相关定义与问题描述. 第 3 节给出一种时间区间近似环路检测的基线方法. 第 4 节提出一种时间区间近似环路检测的优化方法. 第 5 节实验验证所提方法的高效性与有效性. 第 6 节对全文进行总结并进行展望.

1 相关工作

本节从静态图中的简单环路检测、时序图中的简单环路检测、时序图中的可达性查询这 3 方面来介绍与本研究有关的国内外研究进展.

1.1 静态图中的简单环路检测

早在 20 世纪 70 年代, 人们就开始研究图中所有简单环路的枚举问题. 简单环路指每个中间节点只发生一次节点自身到自身的路径. 该问题所有解决方法中性能较好的要数 Tiernan 算法[4]、Weinblatt 算法[5]、Johnson 算法[6].

Tiernan 算法[4]提出通过贪婪搜索方法在静态有向图中寻找所有满足一定长度的简单环路的算法. 其以深度优先遍历为基础, 从初始节点开始, 按照节点编号递增顺序, 试探将邻居节点加入路径是否成环路. 如果构成环路, 则输出该环路, 并从图中将该邻居去掉, 继续判断其他邻居是否成环, 直到试探完所有节点. 在进行理论分析时, 对时间复杂度的上限进行了理论证明. 实验中, 分别在含有 1–8 个节点的完全图上进行所有环路的检测, 结果证明该算法更适用于寻找小规模图中的环路, 对于节点超过 7 或边数超过 50 的较大规模有向图, 该计算则非常耗时.

Weinblatt 算法[5]的适用范围比 Tiernan 算法[4]要大, 其可以应用于大图. 该算法首先删除只有出边或者入边的节点, 因为环路中节点既有出边又有入边; 接着从某初始节点出发, 遇到之前检测过的节点, 就回溯到上一个分支节点, 继续检测是否有未被搜索的路径; 如果不存在, 则选择新的初始节点, 继续上述检测过程. 该算法用一个数据结构保存当前初始节点和正在搜索的路径. 当需要回溯时, 从该数据结构中删除那些点和边, 并判断剩余路径是否可以构成环路. 一般情况下, 该算法效率相对较高. 其最坏的时间复杂度为指数级, 所以对于百万以上规模的点和边构成的图并不适用. 另外, 其对存储空间的需求也相对较大.

Johnson 算法[6]是现有解决有向图环路检测问题比较有效的方法之一, 其是对 Tiernan 算法的改进. 该算法首先利用 Tarjan 算法将原图分解成强连通分量, 接着分别在各个强连通分量上寻找环路. 在遍历过程中, 使用 3 个数组存储搜索过的节点, 数组 *A* 记录从初始节点能到达的点, 数组 *Blocked* 记录处于阻塞状态的点, 数组 *B* 记录所有出邻居处于阻塞状态的点. 从初始节点 s 开始, 使用深度优先遍历方法搜索, 上述 3 个数组各司其职. 若当前节点 a 的所有出邻居都为阻塞状态, 那么将 a 节点放入数组 *B* 且在后续计算中不再对其搜索, 因为这几个分支不可能形成环路. 继续增加节点, 如果数组 *A* 中最后一个节点 y 的邻居中有初始节点 s, 那么输出该环路. 如果 y 的所有邻居都已访问完毕, 则将 y 与邻居的指向关系记录在数组 *B* 中, 根据深度优先遍历规则退回 y 的上一级 x, 继续深度优先遍历, 直到退回到 y 的邻居 x, 此时清空 *B* 中关于 y 和 x 的指向关系; 否则继续深度优先搜索访问 y 未被访问的邻居节点. 当出发于初始节点 s 的所有环路都已找到或者相应分支搜索完毕, 则在图中删去节点 s, 从剩下节点中选取一个节点 s′重新执行 Johnson 算法, 不断递归下去, 直到所有节点的环路算法都搜索完毕, 则终止算法. 由于该算法每次只寻找关于一个节点的所有环路, 因此其最坏情况下的时间复杂度为 $O((|V|+|E|)(|\gamma|+1))$, 空间复杂度为 $O(|V|+|E|)$, 其中$|V|$表示节点个数, $|E|$表示边数, $|\gamma|$表示简单环路的个数.

经过分析, 发现现有静态有向图中简单环路检测方法适用于规模不大的小图, 图中节点间也只允许有一条边, 不像时序图中节点间可以有多条不同时间戳的边, 该特性使得这类方法不能直接应用于时序图.

1.2 时序图中的简单环路检测

在静态图中, 边从不重复, 而在时序图中, 交互的重复非常常见. 不考虑时序图的这一特点会给出效率非常低的解决方案.

时序图与静态图的区别在于节点不断重复交互. 因此, 在研究时序图中的环路时, 必须考虑数据的时间性质. Kumar 等人[7]提出了一种基本的时序环路枚举算法, 其利用时间窗口处理数据, 使用深度优先遍历该窗口内的时序图节点, 不断将新的未被访问的点追加到当前形成的路径中, 直至形成环路时结束. 上述方法简单、容易理解、易于实现, 但是算法的复杂度会随着时序图规模的增大而变大, 不适合规模较大的时序图. 当路径增多时, 占用的内存空间也会增大.

随后, Kumar 等人[8]提出了 2SCENT 算法, 解决的问题和该作者前期提出的基本方法[7]是相同的, 即检测出时序图中所有时序非递减的简单环路. 二者相比, 2SCENT 算法运行速度显著提升, 适用于较大规模的时序图, 比如具有百万个以上节点和数亿条以上交互边的大规模网络. 该算法借鉴了 Johnson 算法[6]的思想, 将计算分为两个阶段. 在第 1 阶段, 使用一个三元组〈起始节点, 终止节点, 时间戳〉表示每条时序边, 并找出时间窗口内可能构成环路的节点放入候选节点集合. 当时间窗口或网络规模过大时, 会占用很多内存, 使用布隆过滤器进行了优化. 在第 2 阶段, 在第 1 阶段得到的候选节点集合中实施深度优先遍历, 同时为了减少或者避免重复搜索, 设计了“关闭时间”封锁节点. 由于节点间多条边会影响搜索性能, 使用“路径束”降低递归次数.

最近, 潘敏佳等人[9]利用基于“最小长度”约束的剪枝方法对 2SCENT 算法进行了改进. 该算法也是一个两阶段算法, 在第 1 阶段, 遍历原图得到可能构成环路的节点及其时间、长度信息; 在第 2 阶段, 利用约束性深度优先遍历方法, 获得满足长度要求的时序非递减的环路. 与 2SCENT 算法相比, 第 1 阶段中的候选节点数量大大减少, 在搜索符合某一长度约束的环路所耗时间很少. 然而, 遍历过程中依然存在重复搜索的问题, 如果能解决, 性能会得到进一步的提升.

近来, 研究者试图利用并行化手段解决环路的枚举问题[11–14], 该研究方向也是可以探索研究的. 由于篇幅所限, 这里不做过多介绍.

经过分析, 可以发现: (1) 无论是静态图还是时序图中的环路检测, 多数以深度优先遍历为基础, 使用各种剪枝优化策略, 大大减少了非必需的搜索环节, 降低了检测算法的空间和时间复杂度, 现有方法各自适用于不同规模图中的环路检测计算. (2) 无论是静态图还是时序图中大多数现有环路枚举方法假设环路的起点和终点是同一点, 这样就错失了具有重要信息的环路. 现实中环路的起点和终点并不一定是同一点, 可能是同一族群, 该特点正是本文检测近似环路依赖的第 1 个重要特征. (3) 在时序图中检测时序环路的大多数方法假设环路中的边出现的时序是非递减的, 但是比如具有反侦查意识的洗钱犯罪分子会避开严格时序非递减, 而打乱前后时间顺序, 这样就会在一定程度上避开相关部门的侦查. 因此, 处于一定时间区间内不符合时间非递减性质的时序路径也应该得到重视. 该特点也是本文检测近似环路依赖的第 2 个特征.

1.3 时序图中的可达性查询

可达性查询是图上的一种基本类型的查询, 在许多领域中都有重要的应用[15]. 本节简要回顾其中的一些典型的研究工作[16–18].

Zhu 等人[16]提出了一种研究大型时序图的可达性索引方法. 首先介绍了一个通用的索引框架, 该框架总结了一系列在现有静态图技术中性能最好的可达性索引. 然后, 在提出的框架下, 提出了处理节点插入和删除的通用高效算法. 此外, 还证明了所提更新算法可以用来改进静态图上现有的可达性技术, 并且还提出了一种在所提的框架下从头开始构建可达性索引的方法.

时序图是一种节点具有特定时间点的标注, 两节点之间的边上有持续时间的有向图. 例如, 用户 A 在上午 11 点呼叫用户 B, 通话持续 7 min. 此行为可抽象为: 由节点 A 指向节点 B 的边, 边上的时间戳为上午 11 点, 持续时长为 7 min. 时序图可用于建模许多具有时间相关活动的网络, 但用于分析时序图的有效算法严重不足. Wu 等人[17]研究了时序图中的基本问题, 如回答可达性和基于时间的路径查询, 并提出了一种专门为处理时序图的这些查询而设计的高效索引技术.

可达性查询是图分析中的一个基本问题. 在社交网络和协作网络等应用中, 边总是与时间戳相关联. 关于时序图中可达性查询的大多数现有工作的假设是, 如果两个节点通过具有非递减时间戳 (时间相关) 的路径连接, 则它们是相关的. 这种假设无法捕捉同一组或活动中涉及的没有时间属性的实体之间的关系. Wen 等人[18]定义了一个可达性模型, 称为跨度或者区间可达, 旨在放松时间顺序依赖并识别给定时间段内实体之间的关系. 采用两跳覆盖的思想, 提出了一种基于索引的方法来回答跨度或者区间可达性查询. 为了提高索引构造和查询处理的效率, 还提供了一些优化.

经过分析, 发现可达性查询方法最基本的特点是起点和终点不是同一点, 且可达性查询时一般根据用户的指定给出起止点, 但是近似环路的起止点不是人为指定的, 如何挖掘高概率起止点是需要解决的关键问题.

2 背景知识

用 $G=(V, E, T)$ 来表示有向时序图, 其中 V 代表节点集合, E 表示有向时序边的集合, T 表示时间戳集合. 每个时序边 $\varepsilon \in E$ 表示为一个三元组 $\langle u, v, t\rangle$, 其中 u 和 v 是 V 中的节点, t 是 u 到 v 的联系时间. 不失一般性, 亦将 t 假定为一个整数, 因为实际应用中的时间戳通常为整数. 给定节点 $u\in V$, u 的出节点表示为 $N_{\text{out}}=\{\langle u, t\rangle \mid (u, v, t)\in E\}$, u 的入节点表示为 $N_{\text{in}} = \{\langle u, t\rangle \mid (v, u, t)\in E\}$. 根据上述表示, 则有 u 的出度为 $degr_{\text{out}}(u)=|N_{\text{out}}(u)|$. 类似地, u 的入度

为 $degr_{in}(u)=|N_{in}(u)|$. 给定时间区间 $[t_s, t_e]$, $[t_s, t_e]$ 中 G 的映射图为 $G_{[ts, te]}$, 其中 $V(G_{[ts, te]})=V$, $E(G_{[ts, te]})=\{(u, t) \mid (u, v, t)\in E, t\in[t_s, t_e]\}$. 一个时间区间 $[t_s, t_e]$ 内的长度或者宽度为 t_e-t_s+1. 文中一些重要符号及其说明见表 1.

表 1　文中用到的重要符号

符号	说明	符号	说明
$G=(V, E, T)$	有向时序图	θ	活跃度阈值
V	节点集合	W	时间窗口
$\|V\|$	节点个数	$\|W\|$	时间窗口长度
E	边集合	ω	时间区间
$\|E\|$	边个数	hot	热点集合
C_{se}	起止点对集合	$\|hot\|$	热点个数
$\|C_{se}\|$	起止点对个数	$N_{out}(u)$	节点u的出节点
$\|C_s\|$	起点个数	$N_{in}(u)$	节点u的入节点
$\|C_e\|$	终点个数	$degr_{out}(u)=\|N_{out}(u)\|$	节点u的出度
C_s	起点集合	$degr_{in}(u)=\|N_{in}(u)\|$	节点u的入度
C_e	终点集合	$path$	近似环路集合

定义 3. 环路. 在图 $G=(V, E)$ 中, 其中 V 是节点集合, E 是边集合. 当且仅当满足以下条件时, 路径 $P=(v_0, v_1,\ldots, v_n)$ 被称为环路.

(1) $v_0=v_n$, 即路径的起点和终点相同.

(2) 对于任意 $i\in[0, n-1]$, 边 $(v_i, v_{i+1})\in E$, 即路径上的每对连续节点之间存在一条边.

(3) 对于任意 $i, j\in[1, n-1]$, 如果 $i\neq j$, 则 $v_i\neq v_j$, 即路径上的其他节点只经过一次.

这种路径也被称为简单环路, 因为它除了起点和终点相同外, 其他节点均不重复.

例 1: 以图 3 所示时序图为例 (本例实线、虚线代表不同的数据源, 不适用于其他例子), 在多种数据源的情况下, 可以发现时序环路 a→c→e→a, 而在单一数据源且 e 与 a 是一个小团体的情况下, 只能发现近似环路 a→c→e, 此时已经很接近时序环路, 发现此种模式也很有现实意义.

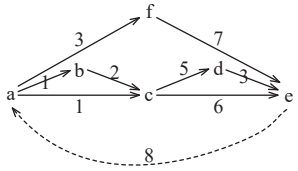


图 3　建模后的时序图中的环路举例

定义 4. 时序环路. 在时序图 $G=(V, E, T)$ 中, 其中 V 是节点集合, $E\subseteq V\times V\times T$ 是带时间戳的有向边集合, T 是时间戳集合. 当且仅当满足以下条件时, 路径 $P=(v_0, v_1,\ldots, v_n)$ 被称为时序环路.

(1) $v_0=v_n$, 即路径的起点和终点相同.

(2) 对于任意 $i\in[0, n-1]$, 边 $(v_i, v_{i+1}, t_i)\in E$, 即路径上的每对连续节点之间存在一条带时间戳的边.

(3) 对于任意 $i, j\in[1, n-1]$, 如果 $i\neq j$, 则 $v_i\neq v_j$, 即路径上的其他节点只经过一次.

(4) 路径上的边的时间戳满足 $t_0\leqslant t_1\leqslant\ldots\leqslant t_{n-1}$, 即路径上的边在时间上是非递减的.

例 2: 以图 3 中所示时序图 (只用实线数据) 为例, 在时间区间 $\omega=[0, 5]$ 可以发现近似环路 a→c→d→e, a→b→c→d→e. 在以往严格按照时间非递减的要求的情况下, 以上两个近似环路会被忽略掉.

定义 5. 活跃度. 在时序图 $G=(V, E, T)$ 中, 给定时间区间 $[t_s, t_e]$, 节点 $v\in V$ 的活跃度 $A(v)$ 是指该节点在时间区间 $[t_s, t_e]$ 内的活动程度, 可以用该节点的出度或入度之一来表示.

(1) 基于出度的活跃度:

$$A_{out}(v) = degr_{out}(v, [t_s, t_e]) \tag{4}$$

其中, $degr_{\text{out}}(v,[t_s,t_e]) = |(v,\text{u},t) \in E \mid t_s \leqslant t \leqslant t_e|$.

(2) 基于入度的活跃度:

$$A_{\text{in}}(v) = degr_{\text{in}}(v,[t_s,t_e]) \tag{5}$$

其中, $degr_{\text{in}}(v,[t_s,t_e]) = |(\text{u},v,t) \in E \mid t_s \leqslant t \leqslant t_e|$.

需要注意的是, 基于出度的活跃度反映了节点 v 在指定时间区间内发送信息或与其他节点交互的频率. 基于入度的活跃度反映了节点 v 在指定时间区间内接收信息或被其他节点交互的频率. 根据具体应用场景, 可以选择使用出度或入度来表示活跃度, 以突出节点的发送或接收能力.

定义 6. 活跃度阈值. 在时序图 $G=(V, E, T)$ 中, 给定时间区间 $[t_s, t_e]$, 活跃度阈值 θ 是一个用于判断节点是否达到一定活跃度水平的界限值. 节点 $v \in V$ 被称为活跃节点, 当且仅当其出度或入度中至少一个达到或超过阈值 θ. 形式化地, 节点 v 在时间区间 $[t_s, t_e]$ 内的活跃度阈值定义为:

$$A_{\text{threshold}}(v,\theta) = (degr_{\text{out}}(v,[t_s,t_e]) \geqslant \theta) \vee (degr_{\text{in}}(v,[t_s,t_e]) \geqslant \theta) \tag{6}$$

例 3: 以图 3 中所示时序图为例 (只用实线数据), 设活跃度阈值 θ=2, 则节点 a、c、e 为活跃节点.

定义 7. 热点. 在时序图 $G=(V, E, T)$ 中, 给定时间区间 $[t_s, t_e]$ 和活跃度阈值 θ, 节点 $v \in V$ 被称为热点, 当且仅当其出度或入度均达到或超过阈值 θ. 形式化地, 热点的定义如下:

$$H(v,[t_s,t_e],\theta) = (degr_{\text{out}}(v,[t_s,t_e]) \geqslant \theta) \wedge (degr_{\text{in}}(v,[t_s,t_e]) \geqslant \theta) \tag{7}$$

热点反映了节点在指定时间区间内的高活动程度, 既包括发送信息的频率 (出度), 也包括接收信息的频率 (入度).

例 4: 以图 3 中所示时序图为例 (只用实线数据), 设活跃度阈值 θ=2, 由于 $degr_{\text{out}}(\text{c}) = |N_{\text{out}}(\text{c})| = 2$, $degr_{\text{in}}(\text{c}) = |N_{\text{in}}(\text{c})| = 2$, 二者均达到阈值 θ=2, 则节点 c 为热点.

问题描述 (时间区间近似环路检测). 给定时序图 G, 时间区间 ω, 活跃度阈值 θ, 发现处于每个时间区间内的近似环路. 需要说明的是后文中使用时间窗口处理时间区间, 所以后文表述中用时间窗口代替时间区间.

3 基线方法

本节介绍时序图中近似环路检测的基线方法 (Baseline). 基线方法的基本思想是将时序图中近似环路检测问题分为两步解决. 第 1 步计算每个时间窗口内的环路, 将其中符合起止点特征的节点作为后续时间窗口内近似环路的起止点. 这里以起止点最为显著的特征: 多出少进、多进少出作为选择起止点的标准. 第 1 步的目的是解决近似环路检测的冷启动问题, 即寻找高概率的起止点. 第 2 步根据第 1 步得出的起止点在后续时间窗口内利用深度优先遍历方法来检测近似环路.

接下来给出选择起止点过程中需要的活跃度阈值与热点阈值计算方法. 活跃度阈值与热点阈值选择与动态调整的具体过程如算法 1 所示. 如果当前时间窗口计数 i 为 1 (第 1 行), 则只利用当前窗口内的数据计算所有节点出入度的均值、标准差 (第 2 行). 如果当前时间窗口计数 i 小于窗口数量阈值 f (第 3 行), 则利用当前所有窗口内的数据计算所有节点出入度的均值、标准差 (第 4 行). 如果当前时间窗口计数 i 不小于窗口数量阈值 f (第 5 行), 则利用从窗口 W_{i-f+1} 到 W_i 内的数据计算所有节点出入度的均值、标准差 (第 6 行). 最后, 根据上述 3 种情况之一计算出当前窗口活跃度阈值 θ 与热点阈值 θ 为 $(\mu + k\sigma)/2$ (第 7 行).

算法 1. 阈值选择与调整 (*ThresholdChooseUpdate*).

输入: 时序图数据 G, 时间窗口 W_i, 窗口数量阈值 f, 窗口计数 i, 标准差数量 k;
输出: 阈值 θ.

1. **if** i==1 **then**
2. compute mean degree μ and standard deviation σ in W_i;

3. **else if** $i<f$ **then**
4. compute mean degree μ and standard deviation σ in W_1 to W_i;
5. **else**
6. compute mean degree μ and standard deviation σ in W_{i-f+1} to W_i;
7. $\theta \leftarrow (\mu + k\sigma)/2$;
8. **return** θ;

基线方法的具体过程如算法 2 所示. 第 1 行初始化各个变量, 包括窗口计数、时间窗口的左右两个边界、起点集合、终点集合、环路、近似环路路径; 对于每个时间窗口内的数据 (第 2 行), 第 3 行利用深度优先遍历的方法寻找当前窗口内的所有环路, 同时调用算法 1 计算当前窗口内的活跃度阈值 θ_i; 第 4–7 行对挖掘出来的环路中的节点进行分类, 找到多出少进的点放入起点集合 C_s、多进少出的点放入终点集合 C_e; 第 8–10 行对于 C_s 和 C_e 中来自之前时间窗口内的点, 以 C_s 中的点为起点, 寻找是否有以 C_e 中的点为终点的路径. 如果有, 则放入近似环路集合 *path* 中; 第 11 行移动时间窗口的左右边界, 进入下一个时间窗口. 接下来进入第 3–11 行的迭代计算.

算法 2. 基线方法 (Baseline).

输入: 时序图 G, 时间区间 (时间窗口)W_i, 窗口长度$|W|$, 活跃度阈值 θ;
输出: 近似环路集合 *path*.

1. $i \leftarrow 1$, $W_i.left \leftarrow$ first timestamp, $W_i.right \leftarrow W_i.left+|W|$, $C_s \leftarrow \{\}$, $C_e \leftarrow \{\}$, $cycle \leftarrow \{\}$, $path \leftarrow \{\}$;
2. **while** $W_i.getData() \neq$ null **do**
3. $cycle.add(DFSFindCycle())$; $\theta \leftarrow ThresholdChooseUpdate()$; //调用算法 1
4. **for** each cycle in *cycle* **do**
5. **for** each node v in each cycle **do**
6. **if** $degr_{out}(v)>\theta$ && $degr_{out}(v)>degr_{in}(v)$ **then** $C_s.add(v, W_i)$;
7. **if** $degr_{in}(v)>\theta$ && $degr_{in}(v)>degr_{out}(v)$ **then** $C_e.add(v, W_i)$;
8. **for** each node v in C_s from W_{i-1} **do**
9. **for** each node u in C_e from W_{i-1} **do**
10. $path.add(DFSFindPath(v, u))$;
11. $W_{i+1}.left \leftarrow W_i.left+|W|$, $W_{i+1}.right \leftarrow W_i.right+|W|$, i++;
12. **return** *path*;

例 5: 以图 4 所示时序图为例, 说明算法 2 的运行过程. 将活跃度阈值设定为 θ=2. 在时间窗口 [0, 5] 内, 发现 5 个环路 (a→b→c→d→e→a, a→b→c→e →a, a→c→d→e→a, a→c→e→a, a→f→e→a), 且环路以 a 作为起止点. 分析发现 a 的出度 $degr_{out}$(a) 大于活跃度阈值 θ=2、其入度 $degr_{in}$(a) 小于 θ=2 (即多出少进), e 的入度大出度小 (即多进少出), 分别放入起止点集合 C_s、C_e 中. 在时间窗口 [5, 10] 内, 以上个时间窗口发现的 a 和 e 分别作为起止点, 发现了 4 个近似环路 (a→b→c→d→e, a→b→c→e, a→c→d→e, a→c→e). 同时在该时间窗口内发现了 2 个环路 (g→c→b→f→g, g→c→d→f→g), 且环路以 g 为起止点. 分析发现 g 多出少进, f 多进少出, 分别放入起点集合 C_s、终点集合 C_e 中. 在时间窗口 [10, 15] 内, 以上 2 个时间窗口发现了 2 个起止点对〈a, e〉、〈g, f〉, 发现了 3 个近似环路 (a→c→ d→f→e, a→c→e, g→c→d→f, g→c→e→b→f). 同时在该时间窗口内发现了 3 个环路 (b→c→e→b, b→c→d→f→ e→b, b→f→ e→ b), 且环路以 b 为起止点. 分析发现 b 多出少进, e 多进少出, 分别放入起止点集合 C_s、C_e 中. 在时间窗口 [15, 20] 内, 根据以上 3 个时间窗口发现的 3 个起止点对〈a, e〉、〈g, f〉、〈b, e〉, 发现了 7 个近似环路 (a→c→ d→e, a→c→e, a→g→c→d→e, a→g→c→e, b→c→e, b→c→d→e, b→f→e). 同时在该时间窗口内未发现环路, 所以无新节点放入起止点集合 C_s、C_e 中.

● 时间复杂度分析. 第 1 步寻找环路的时间复杂度为 O((|V|+|E|)(|γ|+1)), 其中|γ|表示环路个数. 第 2 步寻找近似

环路的时间复杂度为 $O((|V|+|E|)|C_s||C_e|)$, 其中$|C_s|$表示起点集合中的元素数目, $|C_e|$表示终点集合中的元素数目.

● 优缺点分析. 寻找近似环路的时间复杂度为 $O((|V|+|E|)|C_s||C_e|)$, 复杂度相对较高, 需要优化. 利用深度优先的遍历方法无形中会遍历一些无效的路径, 因此也需要优化路径寻找方法.

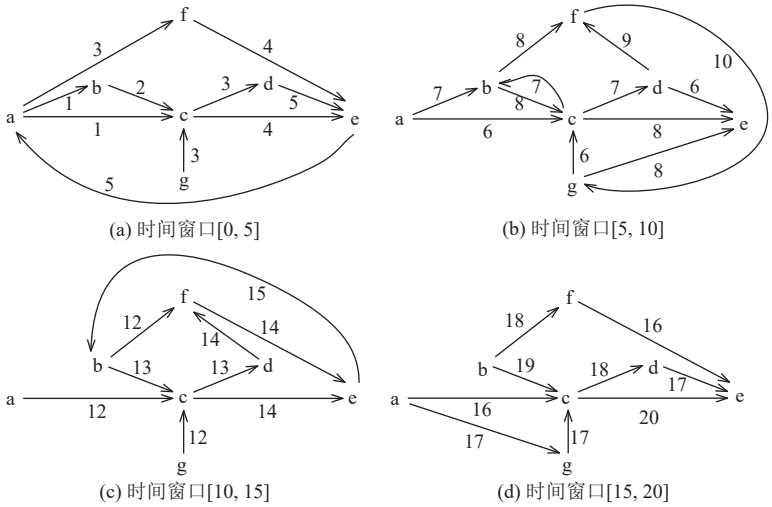


图 4 不同时间窗口内的时序图举例

4 优化方法

由于直接按照基线方法的定义去计算开销较大, 所以本文从起止点组合、起止点间路径索引、近似环路搜索等方面对基线方法进行优化, 简称 Improved.

4.1 起止点组合的优化

由于基线方法需要挖掘所有起点和终点组合之间的可达路径, 所以根据定义计算开销较大. 观察到起止点存在一定的出现概率, 前期出现多次的起止点后期出现的可能性较大, 本文从活跃度方面对起止点进行了分类. 分类标准分为两个: (1) 如果是首个时间窗口, 某起止点在该窗口中出现 θ 次以上, 称该点是活跃起止点. 反之, 称该起止点是非活跃起止点. (2) 如果已经运行多个时间窗口, 某起止点在前 k 个窗口中出现 θk 次以上, 称该起止点是活跃起止点. 反之, 称该起止点是非活跃起止点. 简而言之, 如果某个起止点首次出现, 那么用首个时间窗口的标准 (1) 去计算. 如果某个起止点已经在多个窗口内出现, 那么用多个时间窗口的标准 (2) 去衡量.

起止点组合的具体过程如算法 3 所示. 对于已挖掘出的环路中的每个环路 (第 1 行) 以及每个环路中的每个节点 (第 2 行), 分如下两种情况讨论. (1) 如果节点第 1 次出现 (第 3 行), 若节点出度超过阈值且大于入度, 则将节点作为起始节点 (第 4 行); 若节点入度超过阈值且大于出度, 则将节点作为终止节点 (第 5 行); 若节点出入度均超过阈值, 则将节点作为热点 (第 6 行). (2) 如果节点非第 1 次出现 (第 7 行), 不管大于还是小于 k 次 (第 8 行), 若节点出度超过 $\theta k'$且大于入度, 则将节点作为起始节点 (第 9 行); 若节点入度超过 $\theta k'$且大于出度, 则将节点作为终止节点 (第 10 行); 若节点出入度均超过 $\theta k'$, 则将节点作为热点 (第 11 行). 最后将本窗口内的起止点对放入起止点对集合, 热点放入热点集合 (第 12 行).

算法 3. 起止点配对 (*StartEndPairs*).

输入: 活跃度阈值 θ, 环路集合 *cycle*;
输出: 起止点对集合 $C_{se}\langle c_s, c_e\rangle$.

1. **for** each cycle in *cycle* **do**
2. **for** each node v in each cycle **do**
3. **if** v comes out first time **then**
4. **if** $degr_{out}(v)>\theta$ && $degr_{out}(v)>degr_{in}(v)$ **then** $c_s \leftarrow v$;
5. **if** $degr_{in}(v)>\theta$ && $degr_{in}(v)>degr_{out}(v)$ **then** $c_e \leftarrow v$;
6. **if** $degr_{out}(v)>\theta$ && $degr_{in}(v)>\theta$ **then** $h \leftarrow v$;
7. **else**
8. $f \leftarrow$number of times v has come out, $k' \leftarrow \min(f, k)$;
9. **if** $\mathrm{sum}(degr_{out}(v), k')>\theta k'$ **then** $c_s \leftarrow v$;
10. **if** $\mathrm{sum}(degr_{in}(v), k')>\theta k'$ **then** $c_e \leftarrow v$;
11. **if** $\mathrm{sum}(degr_{out}(v), k')>\theta k'$ && $\mathrm{sum}(degr_{in}(v), k')>\theta k'$ **then** $h \leftarrow v$;
12. $C_{se}.add(\langle c_s, c_e\rangle, W_i)$, $hot.add(h, W_i)$;
13. **return** C_{se};

例 6: 以图 4 所示时序图为例, 说明算法 3 的运行过程. 首先设定单个和多个时间窗口的活跃度的阈值是 θ=2. 时间窗口 [0, 5] 是首个时间窗口, 从例 3 中得到第 1 个时间窗口内的起止点分别为 a 和 e, 且 a 的出度为 3, 超过了阈值 θ, e 的入度为 3, 也超过了 θ, 所以二者分别为活跃起点、活跃终点. 在时间窗口 [5, 10] 内, 发现的起止点对为〈a, e〉、〈g, f〉, 由于〈g, f〉为第 1 次出现, 用单个时间窗口活跃度标准计算, 经过计算, 在下个时间窗口内使用. 由于〈a, e〉为第 2 次出现, 用多个时间窗口标准计算, 由于 a 和 e 在两个时间窗口内都超过活跃度阈值, 所以〈a, e〉也为活跃起止点, 可在本时间窗口内使用. 在时间窗口 [10, 15] 内, 以上 2 个时间窗口发现的 3 个起止点对〈a, e〉、〈g, f〉、〈b, e〉, 由于 a 和 g 的活跃度未超过阈值, 在该时间窗口内为非活跃起止点. 由于〈b, e〉为第 1 次出现, 用单个时间窗口活跃度标准计算, 经过计算, 在下个时间窗口内使用. 在时间窗口 [15, 20] 内, 以上 3 个时间窗口发现的 3 个起止点对〈a, e〉、〈g, f〉、〈b, e〉, 只有 g 的活跃度未超过阈值, 在该时间窗口内为非活跃起止点. 另外两个起止点对〈a, e〉、〈b, e〉为活跃起止点, 可在本时间窗口内使用.

经过活跃度的检测, 减掉部分起止点, 在一定程度上提升了挖掘效率, 在可解释性方面, 也容易解释. 以金融洗钱为例, 洗钱需要大量的转出资金, 起点转出笔数太少, 洗钱的可能性就大大降低. 因此, 用活跃度来区分是否为真正的起止点是比较合理的.

4.2 起止点间路径索引方法

由于基线方法需要按照深度优先遍历的方法对可达路径进行搜索, 期间需要保存较多的中间结果, 其中一定数量的中间结果到最后并不能达到终点, 搜索效率并不高. 尽管可以通过提前剪枝减少不符合条件的分支的遍历, 但是未提前剪枝的分支还存在一部分, 同样会浪费一定的空间与计算资源.

根据上述分析, 直接寻找起止点之间的路径效率不高, 本节给出优化方法. 根据第 4.1 节发现的活跃起止点来建立轻量索引 *Lindex*, 这里采用活跃起点到热点、活跃终点到热点建立轻量索引. 热点的定义标准是节点出入度都超过活跃度阈值 θ. 索引结构为〈from, [timestamp, to], [timestamp, to],…〉.

轻量索引创建的具体过程如算法 4 所示. 对于热点集合中的每个热点 h (第 1 行) 以及起止点对集合 C_{se} 的每个起止点对〈c_s, c_e〉(第 2 行), 做以下两项工作. (1) 对于起止点对中的起始点 c_s, 使用深度优先搜索方法搜索 c_s 通向热点 h 的路径并放入 *tmp* 中 (第 3 行). 如果 *Lindex* 索引中前项 *path_bf* 包含该路径, 则更新已存在路径中的时间戳 (第 4 行). 如果 *Lindex* 索引中前项 *path_bf* 不包含该路径, 则将 *tmp* 加入 *path_bf* 中 (第 5 行). 如果 *tmp* 中包含 c_s 到 c_e 的路径, 则将该段路径加入 *path* 中 (第 6 行). (2) 对于起止点对中的终止点 c_e, 使用深度优先搜索方法搜索 c_e 通向热点 h 的路径并放入 *tmp* 中 (第 7 行). 如果 *Lindex* 索引中后项 *path_af* 包含该路径, 则更新已存在路径中的时间戳 (第 8 行). 如果 *Lindex* 索引中后项 *path_af* 不包含该路径, 则将 *tmp* 加入 *path_af* 中 (第 9 行). 如果

tmp 中包含 c_s 到 c_e 的路径, 则将该段路径加入 *path* 中 (第 10 行). 最后, 将其他路径标记为非活跃状态 (第 11 行).

算法 4. 轻量索引创建 (*Lindex*).

输入: 起止点对集合 $C_{se}\langle c_s, c_e\rangle$, 热点集合 *hot*;
输出: *Lindex*.

1. **for** each node *h* in *hot* **do**
2. **for** each pair $\langle c_s, c_e\rangle$ in C_{se} **do**
3. *tmp* ← *DFSFindPath*(c_s, *h*);
4. **if** *path_bf.contain*(*tmp*) **then** update timestamp of *tmp*;
5. **else** *path_bf.add*(*h*, c_s, *tmp*);
6. **if** *tmp.contain*(c_e) **then** *path.add*(c_s, c_e, *tmp*);
7. *tmp*←*DFSFindPath*(c_e, *h*);
8. **if** *path_af.contain*(*tmp*) **then** update timestamp of *tmp*;
9. **else** *path_af.add*(*h*, c_e, *tmp*);
10. **if** *tmp.contain*(c_s) **then** *path.add*(c_s, c_e, *tmp*);
11. marked other paths as inactive;
12. **return** *Lindex*⟨*path_bf*, *path_af*⟩;

例 7: 以图 4 中所示时序图为例, 说明算法 4 (轻量索引 *Lindex* 创建) 的具体过程 (见后文图 5), 其中活跃度阈值 θ 为 2. 在第 1 个时间窗口 [0, 5] 内, 活跃起止点对为⟨a, e⟩, 热点为 c. 起点到热点的索引为⟨a, 1, c⟩, ⟨a, [1, b], [2, c]⟩, 终点到热点的索引为⟨e, 4, c⟩, ⟨e, [5, d], [3, c]⟩, 未经过热点的起点到终点的索引为⟨a, [3, f], [4, e]⟩. 在时间窗口 [5, 10] 内, 活跃起止点对为⟨a, e⟩、⟨g, f⟩, 热点为 c、b. 由于⟨a, e⟩在前一个时间窗口内出现过, 如果有相同路径, 可以在前一个索引的基础上略作改动, 如果没有相同路径, 则重新生成路径索引. 经过简单遍历, 发现路径基本相同, 所以只需要改动其中的时间戳即可, 即起点 a 到热点 c 的索引修改为⟨a, 6, c⟩, ⟨a, [7, b], [8, c]⟩, 终点 e 到热点 c 的索引修改为⟨e, 8, c⟩, ⟨e, [6, d], [7, c]⟩. 由于热点 b 第 1 次出现, 一些索引需要从头创建. 起点 a 到热点 b 的索引为⟨a, 7, b⟩, ⟨a, [6, c], [7, b]⟩, 终点 e 到热点 b 的索引为⟨e, [8, g], [10, f], [8, b]⟩. 由于 a 只有两个出边并且都用过了, 所以起止点间未经过热点的路径不存在. 对于⟨g, f⟩, 起点 g 到热点 c 的索引为⟨g, 6, c⟩, 终点 f 到热点 c 的索引为⟨f, [8, b], [7, c]⟩, ⟨f, [9, d], [7, c]⟩. 起点 g 到热点 b 的索引为⟨g, [6, c], [7, b]⟩, 终点 f 到热点 b 的索引为⟨f, 8, b⟩, ⟨f, [9, d], [7, c], [8, b]⟩. 在时间窗口 [10, 15] 内, 由于 a 和 g 的活跃度未超过阈值, 将⟨a, e⟩、⟨g, f⟩在上一个周期内创建的索引放入非活跃索引中. 活跃起止点对只剩下⟨b, e⟩, 热点为 c. 对于第 1 次出现的⟨b, e⟩, 预先创建索引. 起点 b 到热点 c 的索引为⟨b, 13, c⟩, 终点 e 到热点 c 的索引为⟨e, 14, c⟩, ⟨e, [14, f], [14, d], [13, c]⟩. 起止点间未经过热点的索引为⟨b, [12, f], [14, e]⟩. 在时间窗口 [15, 20] 内, 活跃起止点对为⟨a, e⟩、⟨b, e⟩, 热点为 c. 将前期创建好的⟨a, e⟩间的非活跃节点置为活跃索引, 起点 a 到热点 c 的索引修改为⟨a, 16, c⟩, ⟨a, [17, g], [17, c]⟩, 终点 e 到热点 c 的索引修改为⟨e, 20, c⟩, ⟨e, [17, d], [18, c]⟩. 起止点⟨a, e⟩间未经过热点的路径不存在. 起点 b 到热点 c 的索引为⟨b, 19, c⟩, 终点 e 到热点 c 的索引为⟨e, 20, c⟩, ⟨e, [17, d], [18, c]⟩. 起止点间未经过热点的索引为⟨b, [18, f], [15, e]⟩.

4.3 起止点间路径搜索方法

根据起止点与轻量索引搜索近似环路的核心思想是从索引结构中找到起点的路径索引、终点的路径索引, 二者之间做连接操作, 拼成完整路径. 如果二者数量都很多, 进行笛卡尔积操作会十分耗时, 所以在创建轻量索引结构时建议对中间做连接的节点也做索引, 这样连接速度会快很多.

起止点间路径搜索的具体过程如算法 5 所示. 对于热点集合 *hot* 中的每个热点 *h* (第 1 行) 以及起止点对集合 C_{se} 中的每个起止点对<c_s, c_e> (第 2 行), 从轻量索引 *Lindex* 的前项 *path_bf* 中获取 *h* 到 c_s 的路径, 同时从轻量索

引 *Lindex* 的后项 *path_af* 中获取 h 到 c_e 的路径 (第 3 行), 接着将二者合并加入 *path*, 同时将 *Lindex* 中以<c_s, c_e> 为起止点且不经过热点的路径加入 *path* (第 4 行).

时间窗口	活跃起点	起点到热点的索引	热点	终点到热点的索引	活跃终点	起止点间其他索引
[0, 5]	a	〈a, 1, c〉 〈a, [1, b], [2, c]〉	c	〈e, 4, c〉 〈e, [5, d], [3, c]〉	e	〈a, [3, f], [4, e]〉
[5, 10]	a	〈a, 6, c〉 〈a, [7, b], [8, c]〉 〈a, 7, b〉 〈a, [6, c], [7, b]〉	c b	〈e, 8, c〉 〈e, [6, d], [7, c]〉 〈e, [8, g], [10, f] , [8, b]〉	e	
	g	〈g, 6, c〉 〈g, [6, c], [7, b]〉		〈f, [8, b], [7, c]〉 〈f, [9, d], [7, c]〉 〈f, 8, b〉 〈f, [9, d], [7, c], [8, b]〉	预先创建 f	
[10, 15]	a g	标记为非活跃索引 〈a, 6, c〉 〈a, [7, b], [8, c]〉 〈a, 7, b〉 〈a, [6, c], [7, b]〉 〈g, 6, c〉 〈g, [6, c], [7, b]〉	c b	〈e, 14, c〉 〈e, [6, d], [7, c]〉 〈e, [8, g], [10, f] , [8, b]〉 标记为非活跃索引 〈f, 8, b〉 〈f, [9, d], [7, c], [8, b]〉 〈f, [8, b], [7, c]〉 〈f, [9, b], [7, c]〉	e f	
	b	〈b, 13, c〉	c	〈e, [14, f], [14, d], [13, c]〉	e	〈b, [12, f], [14, e]〉
[15, 20]	a	〈a, 16, c〉 〈a, [17, g], [17, c]〉	c	〈e, 20, c〉 〈e, [17, d], [18, c]〉	e	
	g	非活跃索引 〈a, [7, b], [8, c]〉 〈a, 7, b〉 〈a, [6, c], [7, b]〉 〈g, 6, c〉 〈g, [6, c], [7, b]〉	b	〈e, [8, g], [10, f] , [8, b]〉 〈f, 8, b〉 〈f, [9, d], [7, c], [8, b]〉 〈f, [8, b], [7, c]〉 〈f, [9, b], [7, c]〉	f	
	b	〈b, 19, c〉	c	非活跃索引 〈e, [14, f], [14, d], [13, c]〉	e	〈b, [18, f], [15, e]〉

图 5 轻量级索引结构举例 (例 7)

算法 5. 搜索连接路径 (*SearchMerge*).

输入: *Lindex*〈*path_bf*, *path_af*〉;
输出: 近似环路集合 *path*.

1. **for** each node h in *hot* **do**
2. **for** each pair 〈c_s, c_e〉 in C_{se} **do**
3. $p_{bf} \leftarrow path_bf.get(h, c_s)$, $p_{af} \leftarrow path_af.get(h, c_e)$;
4. $path.add(\langle c_s, c_e\rangle, p_{bf}+p_{af})$, $path.add(Lindex.get(\langle c_s, c_e\rangle))$;
5. **return** *path*;

例 8: 以图 4 中所示时序图为例, 说明算法 5 利用轻量索引进行搜索合并的具体过程 (见图 6). 图 6 中 hash() 表示哈希函数. 在第 1 个时间窗口 [0, 5] 内, 活跃起止点对为〈a, e〉, 热点为 c. 由于只有一个热点和一个起止点对, 只需对起点路径索引、终点路径索引做全连接即可. 同时加上未经过热点的 1 个近似环路, 这样得到 5 个近似环

路. 在时间窗口 [5, 10] 内, 活跃起止点对为〈a, e〉、〈g, f〉(本窗口不使用), 热点为 c、b. 由于只需计算一个起止点对和两个热点间的路径, 只需分别对每个热点所对应的起止点路径索引做全连接即可. 本窗口内没有未经过热点的近似环路, 最终得到 6 个近似环路. 如果没有优化, 即不区分热点, 会进行 12 次连接. 尽管得到的近似环路数量一样, 但是会有一些不必要的计算. 在时间窗口 [10, 15] 内, 活跃起止点对为〈b, e〉, 热点为 c. 由于只有一个热点, 按全连接处理即可. 同时加上未经过热点的 1 个近似环路, 这样得到 3 个近似环路. 在时间窗口 [15, 20] 内, 活跃起止点对为〈a, e〉、〈b, e〉, 热点为 c. 尽管有两个起止点对, 由于只有一个热点, 只需对起点路径索引、终点路径索引 (包含去重操作) 进行全连接处理即可. 同时加上未经过热点的 1 个近似环路, 这样得到 7 个近似环路.

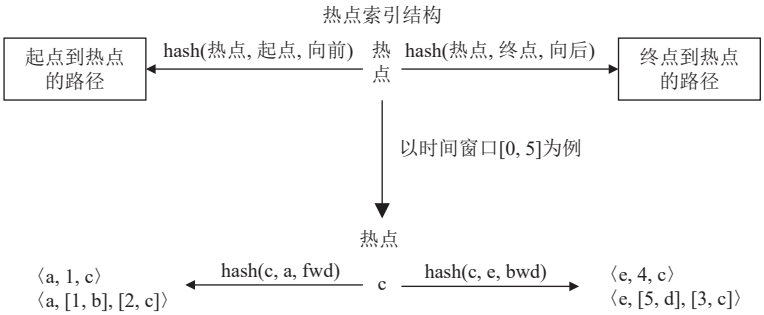


图 6 热点指向路径的索引结构举例

4.4 优化后的算法

优化后算法的基本思想是: 将时序图中近似环路检测问题分为两步解决. 第 1 步在基线方法的基础上, 每个窗口内挖掘出起止点后, 都要做检验. 检验包含两种情况: (1) 如果为首个窗口, 检验单个窗口中每个起止点的活跃度是否超过 θ. (2) 如果为非首个窗口, 检验包含此窗口在内的之前的 k 个窗口中每个起止点是否超过 θk 次. 如果未超过, 则删除该起止点. 第 2 步在基线方法的基础上, 利用第 4.2、4.3 节的活跃路径所建轻量级索引 *Lindex*, 来搜索第 1 步精简后的起止点间的可达路径.

优化后方法的具体过程如算法 6 所示. 首先初始化各个变量, 包括窗口计数、时间窗口的左右两个边界、起点集合、终点集合、环路、近似环路路径、轻量索引 *Lindex* 前项、轻量索引 *Lindex* 后项 (第 1 行); 对于每个时间窗口内的数据 (第 2 行), 利用深度优先遍历方法搜索环路 (第 3 行); 基于上一步搜索到的环路, 调用算法 3 进行起止点的配对 (第 4 行); 接着利用上一步发现的起止点对和热点, 调用算法 4 创建轻量索引 *Lindex* (第 5 行); 然后利用上一步创建好的轻量索引, 调用算法 5 进行近似环路的搜索和连接 (第 6 行); 最后移动时间窗口的左右边界, 进入下一个时间窗口 (第 7 行), 接下来进入第 3–7 行的迭代计算.

算法 6. 优化方法 (Improved).

输入: 时序图 G, 时间窗口 W, 窗口长度$|W|$, 活跃度阈值 θ;
输出: 近似环路集合 *path*.

1. $i \leftarrow 1$, $W_i.left \leftarrow$ first timestamp, $W_i.right \leftarrow W_i.left+|W|$, $C_s \leftarrow \{\}$, $C_e \leftarrow \{\}$, $cycle \leftarrow \{\}$, $path \leftarrow \{\}$, $path_bf \leftarrow \{\}$, $path_af \leftarrow \{\}$;
2. **while** $W_i.getData() \neq$ null **do**
3. $cycle.add(DFSFindCycle())$;
4. $StartEndPairs(\theta, cycle)$; //算法 3
5. $Lindex(C_{se}\langle c_s, c_e\rangle, hot)$; //算法 4
6. $path.add(SearchMerge(Lindex\langle path_bf, path_af\rangle))$; //算法 5
7. $W_{i+1}.left \leftarrow W_i.left+|W|$, $W_{i+1}.right \leftarrow W_i.right+|W|$, i++;
8. **return** *path*;

4.5 理论分析

本节主要从理论方面证明所提出方法的高效性与有效性.

定理 1. 在未指定起止点的情况下, 基于深度优先遍历方法的时间区间近似环路的检测方法的时间复杂度为 $O((|V|+|E|)|V|^2)$.

证明: 设时序图中有$|V|$个节点, $|E|$条边. (1) 在每次深度优先遍历中, 每个节点只会被访问一次. 对于每个节点, 需要遍历其邻居节点, 并递归地继续深入搜索. 因此, 深度优先遍历的时间复杂度可以表示为 $O(|V|+|E|)$. 这是因为在最坏情况下, 需要遍历所有的节点和边. 对于节点的遍历, 时间复杂度为 $O(|V|)$, 对于边的遍历, 时间复杂度为 $O(|E|)$. (2) 由于未指定近似环路的起止点, 所以$|V|$个节点都有可能是起止点, 因此$|V|$个起点到$|V|$个终点的基于深度优先遍历方法的时间区间近似环路检测的时间复杂度为 $O((|V|+|E|)|V|^2)$.

定理 2. 指定起止点的时间区间近似环路的检测方法 (基线方法) 的时间复杂度为 $O((|V|+|E|)|C_s||C_e|)$.

证明: 设时序图中有$|V|$个节点, $|E|$条边, 起点数量为$|C_s|$, 终点数量为$|C_e|$. 由定理 1 可知, 对于每次深度优先遍历, 时间复杂度为 $O(|V|+|E|)$. 对于$|C_s|$个起点到$|C_e|$个终点的基于深度优先遍历的时间区间近似环路检测的时间复杂度为 $O((|V|+|E|)|C_s||C_e|)$.

定理 3. 起止点到热点的索引创建方法 (优化方法之索引) 的时间复杂度为 $O((|V|+|E|)|C_{se}||hot|)$.

证明: 设时序图中有$|V|$个节点, $|E|$条边, 起止点对数量为$|C_{se}|$, 热点数量为$|hot|$. 因为从图中选出$|C_{se}|$个起止点对到$|hot|$个热点路径的时间复杂度为 $O((|V|+|E|)|C_{se}||hot|)$, 所以起止点到热点的索引创建方法 (优化方法之索引) 的时间复杂度为 $O((|V|+|E|)|C_{se}||hot|)$.

定理 4. 基于起止点到热点的索引搜索合并方法 (优化方法之搜索合并) 的时间复杂度为 $O(|C_{se}||hot|)$.

证明: 设起止点对数量为$|C_{se}|$, 热点数量为$|hot|$. 对于每个起点、终点、热点所索引路径的搜索时间复杂度都为 $O(1)$. 因为需要找出所有起止点对 C_{se} 到热点 hot 的路径, 所以总的时间复杂度为 $O(|C_{se}||hot|)$.

定理 5. 给定$|C_s|$个起点、$|C_e|$个终点、$|C_{se}|$个起止点对、超过活跃度阈值的起止点对平均比例 α, 每个起止点对间平均路径数$|p|$, 使用活跃度度量起止点后, 起止点对最多减少$|C_s||C_e|-\alpha|C_{se}|$, 近似环路的数量最多减少 $(|C_s||C_e|-\alpha|C_{se}|)|p|$.

证明: 按照笛卡尔积方法, 起止点两两配对后的数量为$|C_s||C_e|$. 由于其中部分配对不存在, 所以起止点对的上界为$|C_s||C_e|$. 已知实际的起止点对数量为$|C_{se}|$, 超过活跃度阈值的起止点对平均比例 α, 那么超过活跃度阈值的起止点对数量为 $\alpha|C_{se}|$. 因此, 在对起止点配对优化后, 起止点对数量最多减少$|C_s||C_e|-\alpha|C_{se}|$. 已知每个起止点对间平均路径数为$|p|$, 起止点对间的路径即为近似环路, 那么在对起止点配对优化后, 近似环路的数量最多减少 $(|C_s||C_e|-\alpha|C_{se}|)|p|$.

定理 6. 优化方法的时间复杂度为 $O((|V|+|E|)|C_{se}||hot|)$.

证明: 优化方法的时间复杂度由两部分组成, 分别是索引创建的时间复杂度、搜索合并的时间复杂度. 根据定理 3 和 4 可以得出, 优化方法的时间复杂度为 $O((|V|+|E|)|C_{se}||hot|)+O(|C_{se}||hot|)$. 合并约简后, 得到优化方法的时间复杂度为 $O((|V|+|E|)|C_{se}||hot|)$.

5 实 验

本节将进行广泛的实验来评估所提方法的高效性与有效性. 首先, 介绍实验环境、数据集统计信息、比较方法、评价指标等实验设置信息. 然后, 从运行时间、占用内存、可扩展性等方面对所提方法与基线方法进行比较.

5.1 实验设置

5.1.1 实验环境

所提算法使用 Java 语言实现, 实验机器为安装有 Windows 操作系统的笔记本, 搭载 Intel(R) Core(TM) i5-6267U CPU @ 2.90 GHz 2.80 GHz, 配置有 8 GB 内存.

5.1.2 数据集

实验数据主要来自斯坦福大学的图数据库[19]以及 IBM 公司人工生成的金融交易时序图数据[20–22]. 实验中

用到的 6 个主要时序图数据集在节点数、边数、持续时间、下载地址见表 2.

表 2 实验数据集

数据集	节点数	边数	持续时间	下载地址
slashdot-threads	51 083	140 778	977 d	http://konect.cc/files/download.tsv.slashdot-threads.tar.bz2
MathOverflow	24 818	506 550	2 350 d	http://konect.cc/files/download.tsv.sx-mathoverflow.tar.bz2
higgs-twitter	456 626	14 855 842	7 d	https://snap.stanford.edu/data/higgs-activity_time.txt.gz
StackOverflow	2 464 606	17 823 525	2 774 d	https://snap.stanford.edu/data/sx-stackoverflow-a2q.txt.gz
USElection	2 338 000	1 000 000	10 h	https://zenodo.org/record/1154840#.Y0kOYP1Bz3g
HI-Small_Trans	428 626	1 048 575	8 d	https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml

实验所用 6 个数据集的简要情况如下.

(1) slashdot-threads: 该数据集是科技网站 Slashdot 上的一个用户间的回复网络.

(2) MathOverflow: 该数据集是堆栈交换网站 Math Overflow 上的一个临时交互网络.

(3) higgs-twitter: 在 2012 年 7 月 4 日宣布发现一种具有难以捉摸的希格斯玻色子特征的新粒子之前、期间和之后, 在推特上监测传播过程后建立的数据集.

(4) StackOverflow: 这是堆栈交换网站 Stack Overflow 上的一个临时交互网络.

(5) USElection: 该数据集包含 2016 年美国总统大选前后两个时间段的转发记录.

(6) HI-Small_Trans: 该数据集是 IBM 公司人工生成的一个金融交易网络, 其中包含洗钱等非法交易行为.

5.1.3 比较方法

比较的方法主要是基线方法 (Baseline)、2SCENT、优化方法 (Improved).

本文提出的两种方法都分为两个阶段: 高概率起止点的发现、近似环路的检测. 对于第 1 阶段, 主要对不同数据集、不同时间窗口下发现高概率起止点的耗时进行比较. 对于第 2 阶段, 主要比较有无索引情况下和有无起止点配对情况下, 每个时间窗口内的近似环路检测时间、占用内存, 使用活跃度前后起止点与近似环路数量, 以及算法在不同情形下的可扩展性. 由于方法目的有差别, 所以仅与现有方法 2SCENT 比较可扩展性.

5.1.4 评价指标

主要评价指标是: 运行时间、占用内存、近似环路数量等. 需要说明的是, 因为所比较的算法都能发现所有环路, 所以这里只给出了近似环路数量, 并没有给出准确度评价指标的定义及其比较.

(1) 运行时间 (running time): 指算法或程序执行完成所需的时间, 包括两种类型: 算法的时间复杂度、程序的实际运行时间. 本实验采用后者, 主要比较每个时间窗口数据的平均执行时间、整个数据集的平均总体执行时间.

(2) 占用内存 (memory usage): 指算法或程序在执行过程中所使用的内存空间大小, 包括两种类型: 算法的空间复杂度、程序的实际内存占用. 本实验采用后者, 主要比较未用索引时数据占用的内存大小, 使用索引后数据占用的内存大小等. 它通常以字节为单位表示, 这里用兆字节 (MB) 来表示.

(3) 近似环路数量 (number of AC): 指算法或程序在执行完成后检测出的近似环路的绝对数量.

5.2 运行时间与占用内存以及结果数量的比较

5.2.1 高概率起止点聚类时间的比较

本节实验主要目的是证明每个时间窗口额外计算起止点、热点等内容, 不会降低检测的性能, 反而可以利用前期时间窗口信息解决近似环路检测的冷启动问题.

由于 Baseline 和 Improved 算法前期挖掘时序环路的算法基本一致, 所以不区分具体用二者中的哪个算法. 在 6 个数据集 (见第 5.1.2 节) 和 4 个不同时间窗口$|W|$ (1 h、10 h、1 d、1 w) 下运行起止点聚类算法, 除了无聚类结果不作计时的情况外, 其他大部分执行时间都是毫秒级的, 具体情况见表 3, 聚类基本不占用时间, 在用户的时间容忍范围内, 可见这个做法是可行的.

表 3　不同时间窗口下, 高概率起止节点聚类的耗时比较 (ms)

数据集	1 h	10 h	1 d	1 w
slashdot-threads	—	64	72	111
MathOverflow	70	58	16	91
higgs-twitter	281	991	—	—
StackOverflow	—	4357	4007	—
USElection	65	108	106	—
HI-Small_Trans	424	934	1582	6143

注: 活跃度阈值、热点阈值均为6, “—”代表无结果不计时

5.2.2　有无索引的比较

本节实验主要测试在有无使用索引的情况下检测环路的运行时间和占用内存. 在 6 个数据集 (见第 5.1.2 节) 和 2 个不同时间窗口$|W|$ (1 h、10 h) 下运行基于深度优先遍历的近似环路检测算法 (Baseline 中的一个环节)、基于轻量级索引的近似环路检测算法 (Improved 中的一个环节), 且测试的运行时间和占用内存都是多个时间窗口的平均值. 除数据集 higgs-twitter 和 HI-Small_Trans 外, 其他数据集中节点间交互相对较少, 计算密度较低, 所以上述 4 个数据集的检测时间相对较小. 在计算密度较大, 即节点出入度相对较大的情况下, Baseline 方法对数据集 higgs-twitter 的检测导致内存溢出, 而对数据集 HI-Small_Trans 的检测时间较长, 显示出优化算法 Improved 的必要性与优势. 内存占用情况也基本与运行时间呈正相关关系, 具体情况见表 4. Baseline 方法一旦遇到计算密度较大的数据集, 就会出现占用内存比 Improved 方法多的情形. 这是因为 Baseline 方法采用从起点到终点的深度优先搜索策略, 其搜索过程中会产生更多回溯操作, 并且需要缓存大量中间结果, 因而导致开销较大; 而 Improved 方法是双向搜索和多段连接, 起点较多, 回溯较少, 缓存较少, 所以在计算密度较大情况下 Improved 方法显示出其独特的优势. 本实验也启示研究者后期可以探索将二者结合, 根据具体计算密度在两个方法中灵活切换.

表 4　有无索引的窗口检测时间与占用内存

数据集	$\|W\|$ (h)	时间 (ms)		内存 (MB)	
		Baseline	Improved	Baseline	Improved
slashdot-threads	1	—	—	—	—
	10	**1**	5	40	**38**
MathOverflow	1	**2**	6	**26**	27
	10	**1**	5	84	**84**
higgs-twitter	1	crashed	**105**	crashed	**616**
	10	crashed	**116**	crashed	**366**
StackOverflow	1	—	—	—	—
	10	**1**	12	**83**	1240
USElection	1	**1**	4	424	**418**
	10	**3**	5	**417**	442
HI-Small_Trans	1	324	**221**	478	**445**
	10	19605	**117**	813	**410**

注: “—”代表没有活跃的起止点, 无相应操作; “crashed”代表内存溢出; 较优耗时和内存数值加粗表示

5.2.3　有无起止点配对的比较

本节实验主要测试起止点配对这个优化的效果. 需要说明的是, Baseline 方法在得到起点集合、终点集合后, 使用笛卡尔积方法组合起止点, 给出了所有组合, 但是其中一些组合在现实中也许不会出现. 因此, Improved 方法根据统计信息给出起止点组合, 列出的都是出现可能性较大的节点组合, 去除了不必要的节点组合. 在 Baseline 方法中, 使用笛卡尔积方法配对 (无起止点优化配对) 且无索引辅助计算, 其运行时间和第 5.2.2 节中 Baseline 方法的检测时间是一致的. 相反, Improved 方法有起止节点配对优化和索引辅助计算, 其检测时间大大减少. 具体情况见表 5. 本实验也启示研究者在计算中应尽量采用索引来提升计算性能.

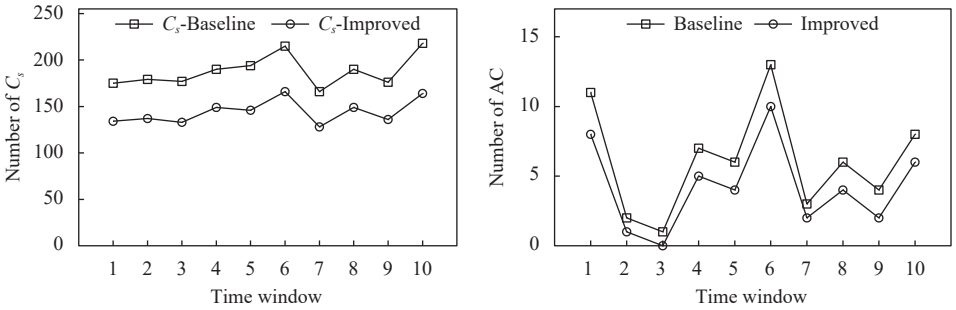
表 5 有无起止点配对的耗时比较

数据集	|W| (h)	时间 (ms)	
		Baseline	Improved
slashdot-threads	1	—	—
	10	1	1
MathOverflow	1	2	**1**
	10	1	1
higgs-twitter	1	crashed	**1**
	10	crashed	**1**
StackOverflow	1	—	—
	10	1	1
USElection	1	1	1
	10	3	**1**
HI-Small_Trans	1	324	**1**
	10	19605	**1**

注: 活跃度阈值、热点阈值为6, “—”代表无结果不计时

5.2.4 使用活跃度前后起止点与近似环路数量的比较

本节实验主要测试起止点配对这个优化前后对起止点数量、近似环路数量的影响效果. 数据集为 HI-Small_Trans, 时间窗口|W|为 2 h, 窗口编号为随机挑选的 10 个不同窗口, 实验结果的具体趋势走向见图 7. 在使用活跃度这个优化之后, 不管是对起点数量, 还是对挖掘出来的近似环路数量, 都有少许减少, 不过在可接受范围之内. 总体来说, 以少量结果的减少来换取整体性能的提升, 这个折中还是值得的.



(a) 对起点数量的影响 (b) 对近似环路数量的影响

图 7 活跃度的影响比较 (HI-Small_Trans, |W|=2 h, θ=6)

5.3 可扩展性比较

5.3.1 时间窗口对扩展性的影响

本节实验主要验证时间窗口大小对所提方法在扩展性方面的影响. 数据集为 HI-Small_Trans, 节点活跃度阈值 θ 为 6, 4 个不同时间窗口|W|分别为 1 h、10 h、1 d、1 w, 实验结果的具体数值见后文表 6, 具体趋势走向见后文图 8(a). Baseline 方法在 1 h 的时间窗口下, 运行时间比 Improved 方法稍高, 其他 3 种情况下时间陡增. 相反, 2SCENT 方法与 Improved 方法展现出良好的可扩展性, 总体运行时间基本没有增减. 而 Improved 方法与 2SCENT 方法相比扩展性相对更好.

5.3.2 活跃度阈值对扩展性的影响

本节实验主要验证活跃度阈值对所提方法在扩展性方面的影响. 数据集为 HI-Small_Trans, 时间窗口|W|为 10 h, 4 个不同的活跃度阈值分别为 4、6、8、10, 实验结果的具体数值见表 7, 具体趋势走向见图 8(b). 两种方法运行时间的总体趋势是在活跃度阈值较小时耗时较多, 活跃度阈值较大时耗时较少, 这是因为较大的阈值过滤掉了多数节点, 计算量大大减少. 与 Baseline 方法相比, Improved 方法显示出更好的可扩展性.

表 6　HI-Small_Trans 数据集上不同时间窗口下的总耗时 (s)

方法	1 h	10 h	1 d	1 w
Baseline	25.014	277.521	>43 200	>86 400
2SCENT	16.060	28.948	27.666	30.131
Improved	19.582	17.779	19.484	18.541

注: 活跃度阈值、热点阈值为6

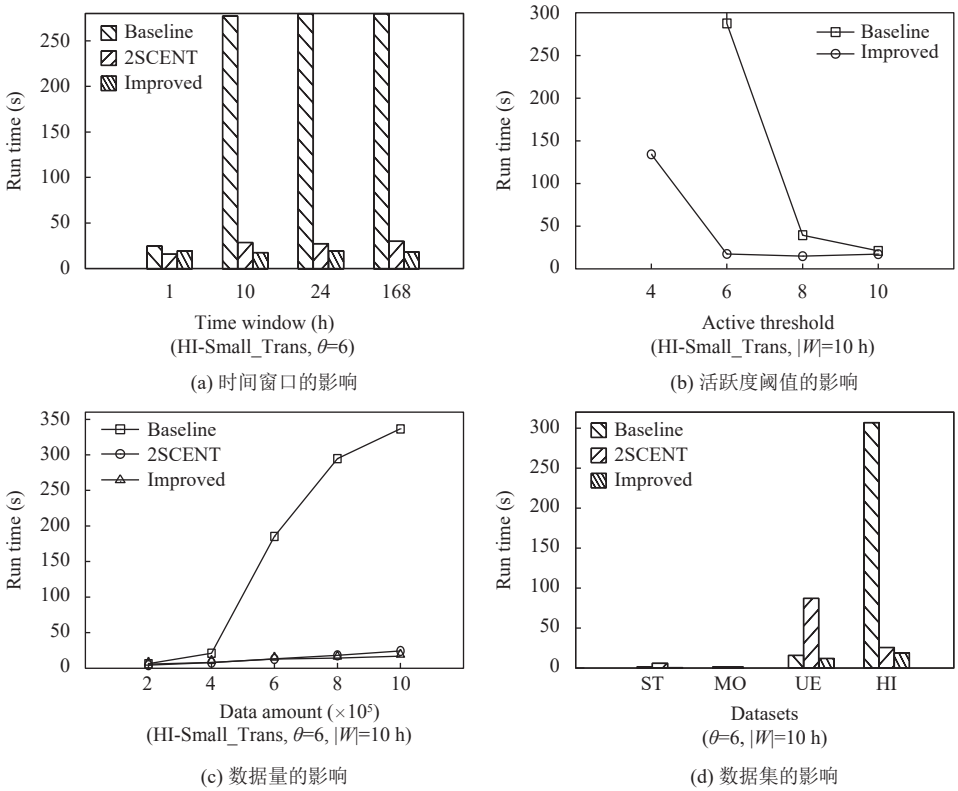


图 8　不同情况下的扩展性比较

表 7　HI-Small_Trans 数据集上不同活跃度阈值下的总耗时 (s)

方法	4	6	8	10
Baseline	8 192.647	287.573	39.459	21.045
Improved	134.091	17.521	14.966	17.231

注: 时间窗口为10 h

5.3.3　数据量对扩展性的影响

本节实验主要验证数据量对所提出方法在扩展性方面的影响. 数据集为 HI-Small_Trans, 节点活跃度阈值为 6, 时间窗口$|W|$为 10 h, 7 个不同的数据量分别为$\leqslant 1\times10^5$、2×10^5、4×10^5、6×10^5、8×10^5、10×10^5、$>1\times10^6$, 实验结果的具体数值见表 8, 具体趋势走向见图 8(c). 实验结果表明: (1) 小规模数据集 (行数$\leqslant 1\times10^5$): 在此范围内, 由于数据量相对较小, Baseline 方法、2SCENT 方法、Improved 方法的运行时间差异不显著. 这主要是因为小规模数据集的计算复杂度较低, 这 3 种方法的性能差异尚未充分显现. (2) 中等规模数据集 (行数为 1×10^5–1×10^6): 随着数据规模的增长, 这 3 种方法的性能差异开始显著. 2SCENT 方法与 Improved 方法在处理中等规模数据集时展现出明显优势, 运行时间较 Baseline 方法大幅减少, 且可视化效果更佳, 差异趋势愈发明显. (3) 大规模数据集 (行数$>1\times10^6$): 基于现有趋势预测, 随着数据规模进一步扩大至 10^7 及以上, Improved 方法的优势将更为突出. 然而, 受

限于当前可用数据集的最大规模 (1.048575×10^6 行), 无法进一步验证这一预测. 鉴于现有数据的可视化效果已能清晰展示优化方法的优越性, 优先选择当前可选数据集中的最优结果进行展示.

表 8 HI-Small_Trans 数据集上不同数据量 (行) 下的总耗时 (s)

方法	$\leqslant1\times10^5$	2×10^5	4×10^5	6×10^5	8×10^5	10×10^5	$>1\times10^6$
Baseline	<3.0	6.377	21.003	185.357	294.60	336.45	>>337.0
2SCENT	<3.0	4.315	7.812	13.038	18.218	24.292	>25.0
Improved	<3.0	5.873	8.284	13.060	14.300	17.135	>18.0

注: 活跃度阈值、热点阈值为6, 时间窗口为10 h

5.3.4 数据集对扩展性的影响

本节实验主要验证不同数据集对所提出方法在扩展性方面的影响. 所采用 4 个数据集分别为 slashdot-threads (ST)、MathOverflow (MO)、USElection (UE)、HI-Small_Trans (HI). 节点活跃度阈值为 6, 时间窗口$|W|$为 10 h, 实验结果的具体数值见表 9, 具体趋势走向见图 8(d). Improved 方法与 Baseline 方法两种方法在 slashdot-threads、MathOverflow 数据集下耗时都很少, 且数值基本相同. 而在 USElection、HI-Small_Trans 数据集下耗时都有不同程度增长, Baseline 方法的耗时增长速度更快. 另外, 2SCENT 方法表现不太稳定, 在 4 个数据集上的表现都不如本文优化方法, 尤其在 MathOverflow 数据集上由于内存溢出导致无运行时间. 总体来说, 与 Baseline 方法和 2SCENT 方法相比, Improved 方法显示出更好的可扩展性.

表 9 不同数据集下的总耗时 (s)

方法	slashdot-threads	MathOverflow	USElection	HI-Small_Trans
Baseline	1.711	1.482	16.080	307.182
2SCENT	6.283	crashed	87.322	25.916
Improved	1.042	1.507	12.249	18.922

注: 活跃度阈值、热点阈值为6, 时间窗口为10 h, “crashed”代表内存溢出

6 总 结

本文研究了大规模时序图中处于多个自定义时间区间内的近似环路发现问题, 即由于数据单一或者数据缺失等原因使得未能发现环路而事实上已经组成环路的问题. 首先提出一种基线方法, 其主要包括高概率起止点聚类方法、起止点间可达路径搜索方法; 接着提出一种优化方法对基线方法存在的主要问题进行了优化, 利用节点的活跃度对高概率起止点进行进一步聚类、利用轻量索引对可达路径搜索进行优化、利用活跃路径对路径连接的早晚进行优化. 实验结果表明, 与基线方法相比, 优化方法可以快速、高效地解决大规模时序图中处于一定时间区间内的近似环路检测问题.

在未来的工作中将会做如下深入的研究: (1) 为弥补单一数据源带来的事实上已经组成环路而未能挖掘出环路的问题, 寻找多模态数据来提升环路检测的数量与精确度; (2) 从时序图中挖掘除环路之外的其他欺诈模式, 比如扇出、扇入、聚集-散射、分散-聚集、随机、二部图、堆栈等模式. (3) 融合机器学习和大模型来提升时序图中欺诈行为识别的精准度与高效性.

致谢 在此向百忙之中评审我们论文的专家学者以及对本文的工作给予支持和建议的同行表示感谢.

References

[1] Holme P, Saramäki J. Temporal networks. Physics Reports, 2012, 519(3): 97–125. [doi: 10.1016/j.physrep.2012.03.001]

[2] Wang YS, Yuan Y, Liu M, Wang GR. Survey of query processing and mining techniques over large temporal graph database. Journal of Computer Research and Development, 2018, 55(9): 1889–1902 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2018.20180132]

[3] Ultipa. The Essential Criteria of Graph Database. Beijing: China Machine Press, 2022 (in Chinese).
[4] Tiernan JC. An efficient search algorithm to find the elementary circuits of a graph. Communications of the ACM, 1970, 13(12): 722–726. [doi: 10.1145/362814.362819]
[5] Weinblatt H. A new search algorithm for finding the simple cycles of a finite directed graph. Journal of the ACM, 1972, 19(1): 43–56. [doi: 10.1145/321679.321684]
[6] Johnson DB. Finding all the elementary circuits of a directed graph. SIAM Journal on Computing, 1975, 4(1): 77–84. [doi: 10.1137/0204007]
[7] Kumar R, Calders T. Finding simple temporal cycles in an interaction network. In: Proc. of the 2017 Co-located with the European Conf. on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD'17). Berlin: Springer, 2017. 3–6.
[8] Kumar R, Calders T. 2SCENT: An efficient algorithm for enumerating all simple temporal cycles. Proc. of the VLDB Endowment, 2018, 11(11): 1441–1453. [doi: 10.14778/3236187.3236197]
[9] Pan MJ, Li RH, Zhao YH, Wang GR. Fast temporal cycle enumeration algorithm on temporal graphs. Ruan Jian Xue Bao/Journal of Software, 2020, 31(12): 3823–3835 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5968.htm [doi: 10.13328/j.cnki.jos.005968]
[10] Qiu XF, Cen WB, Qian ZP, Peng Y, Zhang Y, Lin XM, Zhou JR. Real-time constrained cycle detection in large dynamic graphs. Proc. of the VLDB Endowment, 2018, 11(12): 1876–1888. [doi: 10.14778/3229863.3229874]
[11] Sun SX, Chen YH, He BS, Hooi B. PathEnum: Towards real-time hop-constrained s-t path enumeration. In: Proc. of the 2021 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). New York: ACM, 2021. 1758–1770. [doi: 10.1145/3448016.3457290]
[12] Blanusa J, Ienne P, Atasu K. Scalable fine-grained parallel cycle enumeration algorithms. In: Proc. of the 34th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA). Philadelphia: ACM, 2022. 247–258. [doi: 10.1145/3490148.3538585]
[13] Blanuša J, Atasu K, Ienne P. Fast parallel algorithms for enumeration of simple, temporal, and hop-constrained cycles. ACM Trans. on Parallel Computing, 2023, 10(3): 15. [doi: 10.1145/3611642]
[14] Qing Z, Yuan L, Chen Z, Lin JJ, Ma GJ. Efficient parallel cycle search in large graphs. In: Proc. of the 25th Int'l Conf. on Database Systems for Advanced Applications (DASFAA). Jeju: Springer, 2020. 349–367. [doi: 10.1007/978-3-030-59416-9_21]
[15] Gandhi S, Simmhan Y. An interval-centric model for distributed computing over temporal graphs. In: Proc. of the 36th Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 1129–1140. [doi: 10.1109/ICDE48307.2020.00102]
[16] Zhu AD, Lin WQ, Wang SB, Xiao XK. Reachability queries on large dynamic graphs: A total order approach. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). Snowbird: ACM, 2014. 1323–1334. [doi: 10.1145/2588555.2612181]
[17] Wu HH, Huang YZ, Cheng J, Li JF, Ke YP. Reachability and time-based path queries in temporal graphs. In: Proc. of the 32nd Int'l Conf. on Data Engineering (ICDE). Helsinki: IEEE, 2016. 145–156. [doi: 10.1109/ICDE.2016.7498236]
[18] Wen D, Huang YL, Zhang Y, Qin L, Zhang WJ, Lin XM. Efficiently answering span-reachability queries in large temporal graphs. In: Proc. of the 36th Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 1153–1164. [doi: 10.1109/ICDE48307.2020.00104]
[19] Leskovec J, Krevl A. SNAP datasets: Stanford large network dataset collection. 2014. http://snap.stanford.edu/data
[20] Altman E, Blanuša J, von Niederhäusern L, Egressy B, Anghel A, Atasu K. Realistic synthetic financial transactions for anti-money laundering models. In: Proc. of the 37th Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2023. 1300.
[21] IBM. AML-data-public. 2021. https://ibm.box.com/v/AML-Anti-Money-Laundering-Data
[22] Kaggle. IBM transactions for anti money laundering (AML). 2022. https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml

附中文参考文献

[2] 王一舒, 袁野, 刘萌, 王国仁. 大规模时序图数据的查询处理与挖掘技术综述. 计算机研究与发展, 2018, 55(9): 1889–1902. [doi: 10.7544/issn1000-1239.2018.20180132]
[3] 赢图团队. 图数据库原理、架构与应用. 北京: 机械工业出版社, 2022.
[9] 潘敏佳, 李荣华, 赵宇海, 王国仁. 面向时序图数据的快速环枚举算法. 软件学报, 2020, 31(12): 3823–3835. http://www.jos.org.cn/1000-9825/5968.htm [doi: 10.13328/j.cnki.jos.005968]

作者简介

姜涛, 博士, 副教授, CCF 专业会员, 主要研究领域为时序图数据挖掘, 大数据管理与分析.
李战怀, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据管理, 数据挖掘, 大数据分析, 信息检索.