软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(12):3823−3835 [doi: 10.13328/j.cnki.jos.005968]

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

面向时序图数据的快速环枚举算法*

潘敏佳[1], 李荣华[1], 赵宇海[2], 王国仁[1]

[1](北京理工大学 计算机科学与技术学院,北京 100081)

[2](东北大学 计算机科学与工程学院,辽宁 沈阳 110819)

通讯作者: 李荣华, Email: lironghuabit@126.com

摘 要: 时序图数据是一类边上带有时间戳信息的图数据.在时序图数据中,时序环是边满足时间戳递增约束的回路.时序环枚举在现实中有着很多应用,它可以帮助挖掘金融网络中的欺诈行为.此外,研究时序环的数量对于刻画不同时序图的特性也有重要作用.基于 2018 年由 Rohit Kumar 等人提出的时序环枚举算法(2SCENT 算法),提出一种通过添加环路信息来削减搜索空间的新型时序环枚举算法.所提出的算法为一个两阶段的算法:1) 首先,通过遍历原图获得所有可能会形成环路的节点,以及相应的时间和长度信息;2) 然后,利用以上信息进行动态深度优先搜索,挖掘所有的满足约束条件的环.在 4 个不同的真实时序图数据集上进行了大规模的实验,并以 2SCENT 算法作为基准对算法进行了对比.实验结果表明,所提出的算法较之前最好的 2SCENT 算法要快 50%以上.

关键词: 时序图;时序环;约束环;剪枝;环枚举算法

中图法分类号: TP311

中文引用格式: 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/5968.htm

Fast Temporal Cycle Enumeration Algorithm on Temporal Graphs

PAN Min-Jia[1], LI Rong-Hua[1], ZHAO Yu-Hai[2], WANG Guo-Ren[1]

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

[2](School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: Temporal graph is a type of graph where each edge is associated with a timestamp. In temporal graphs, a temporal cycle denotes a loop where the timestamps of the edges in the loop follow an increasing order. Temporal cycle enumeration has a number of real-life applications. For example, it can be applied to detect the fraud behavior in temporal financial networks. Additionally, the number of temporal cycles can be used to characterize the topological properties of temporal graphs. Based on the 2SCENT algorithm proposed by Rohit Kumar *et al*. in 2018, a new temporal cycle enumeration algorithm is proposed which uses additional cycle information to prune the search space. Specifically, the proposed algorithm is a two-stage algorithm. First, the algorithm traverses the temporal graph to identify all root nodes that probably form temporal cycles, as well as the corresponding time and length information of the cycles. Second, the algorithm performs a dynamic depth first search using the above information to find all valid temporal cycles. Extensive experiments are conducted on four real-life datasets, using 2SCENT as the baseline algorithm. The result shows that the proposed algorithm reduces the running time over the 2SCENT algorithm by 50 percent.

Key words: temporal graph; temporal cycle; constraint cycle; prune; cycle enumeration algorithm

* 基金项目: 国家自然科学基金(61772346, U1809206, 61772124, 61332006, 61332014, 61328202, U1401256)

Foundation item: National Natural Science Foundation of China (61772346, U1809206, 61772124, 61332006, 61332014, 61328202, U1401256)

收稿时间: 2019-07-18; 修改时间: 2019-09-10; 采用时间: 2019-11-25

在现实世界中,很多图数据上都带有时序信息,例如电信话务网络、金融交易网络、道路交通网络和在线社交网络等等.而环路作为一种特殊的结构,在各种图中都频繁出现,并且拥有着重要的意义.目前,时序环已经开始被应用于多个领域,有了多种实际应用.比如在生物领域,时序环可以表示某种生物反馈机制[1];在金融领域,检测出的时序环可能代表了某一种金融诈骗行为[2].攫取时序图结构,可以帮助我们挖掘未知的图模式.图 1 展示了两个具有不同含义的时序图.

- 图 1(a)为用户之间进行金钱交易的时序图,其中的时序边表示某一用户在某一时间向另一用户进行款项支付.可以看到,买家 1 和买家 2 分别处于不同的时序环中,表明它们参与到了不同的有第三方参与的欺骗行为中,具体可以表现为阿里巴巴中某些不正当的刷单行为;
- 图 1(b)表示细胞之间某些信息通过介质进行传递,最终由细胞 D 传往细胞 A 的时序边表示信息的反馈.

可见:在不同背景的时序图下,时序环拥有不同的意义.但在某些情形中,用户需要的并不是所有的环路,而是某些长度的环路.最朴素的算法需要从每个点进行小于等于该长度的深度优先搜索,在大多数情况下,费时并且会占用大量空间.本文在 2SCENT 算法的基础上提出一种两阶段的环路枚举算法,相对于 2SCENT,它减少了搜索所需的时间,并且在大多数情况下都能显著节约存储空间.

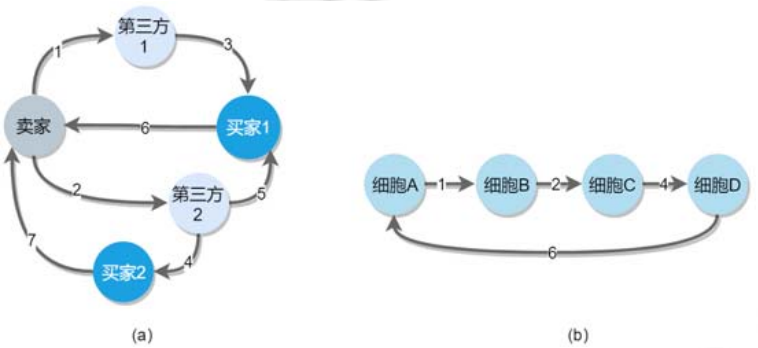


Fig.1 Temporal cycles with different meanings

图 1 具有不同意义的时序环

本文的创新贡献主要包括以下两个方面.

(1) 提出了一种长度限制下的环路枚举新算法.该算法通过引入新型的剪枝技术,允许用户快速获得某一长度以内的所有时序环;

(2) 在 4 个真实的时序图上进行了大量的对比实验,并详细分析了算法起效的原因,以及在不同情况下的表现.实验结果表明,本文提出的算法较过去的 2SCENT 算法在性能上能够提升 50%以上.

本文第 1 节介绍时序环枚举的相关工作.第 2 节给出一些定义以及问题描述.第 3 节介绍环枚举算法的具体设计.第 4 节在 4 个真实数据集上使用 2SCENT 算法和我们的算法来比较算法的运行效率.第 5 节总结全文并且给出未来的工作.

1 相关工作

时序图[3]是由节点和带有时间戳的边组成的集合,相对于静态图而言,它加入了“时序”的概念,因此两个节点之间的多条边不能简单地视作一条.针对时序图,有很多亟待处理的问题.时序图上的查询处理问题[4]主要有路径问题、可达性问题和精确匹配问题等;时序图挖掘问题主要有最小生成树问题、稠密子图挖掘问题和 k-匿名问题等等.

除此之外,由于在时序图上的子图会随着时间进行演化,而这种演化模式会随着时间推移不断出现,因此出现了对“频繁演化模式”的算法研究[5],以及找到 k 个最有影响力的顶点,使得信息得到最大化传播的影响力最大化算法研究等等[6].可以说,研究时序图中隐含的信息,能够提升我们对信息网络的认知水平.

模体(motif)[7]可以用于衡量某一个模式在不同时间规模下出现的频率,这样的模式具有一定的现实意义.模体对于理解图的结构和系统的功能也有着重要的意义.环路就是一种重要的模体,显然,它是指一系列节点在某一时间段内经过一系列交互回到出发点的行为.因此,寻找时序环也可以被认为是在寻找时序图中的一系列模体.

Paranjape 等人[8]提出了对简单模体进行计数的方法,使用的时间接近于线性.该算法寻找的主要是二元模体和三元模体,其中当然也包括了三元环这种情况.然而,由于环路计数与环路枚举有较大的区别,这个算法虽然对计算小的模体数量具有意义,却不适用于更为广泛的情况.

环路计数和环路检测是关于环路的两个枚举问题:环路计数要求统计图中所有环路的条数;环路检测则是关于如何具体构造这些环路,给出详细的路径信息.通过环路检测,我们一定可以获得环路的计数值;但是反过来,环路计数的值却往往对环路检测的实现起不到什么作用.

在静态图上寻找环路是一个经典的问题,从上世纪 70 年代开始就有了研究[9−11].解决问题的算法主要有 Tiernan 算法[12]、Weinblatt 算法[13]等.目前,解决这个问题的最有效方案是 Johnson 算法[14].Johnson 算法可以在静态有向图中找到所有简单环路,同时保证时间复杂度上限为 $O((n+e)(c+1))$,空间复杂度上限为 $O(n+e)$,其中,n 代表节点个数,e 代表边的条数,c 代表图中简单环路的个数.

Johnson 算法用深度优先搜索的方式搜索有向图,与此同时加入剪枝,尽可能地避免当前节点向不可能形成环路的节点进行搜索,因此大大提升了效率.剪枝主要通过对节点的封锁/解锁来进行处理.如当前的环路源点为 a,若由 a 经过一系列路径搜索到 b 之后,发现 b 并不能通过后续的搜索回到 a,则 b 被暂时封锁.这样一来,就可以避免未来由 b 出发的无效搜索.

但是对于无向图上的环路枚举,Johnson 算法不是最优的.Ferreira 等人提出了一个更优的算法[15]:令 $C(G)$ 为图 G 上的所有环路,$|c|$为 $C(G)$中每一条环路的边的数量,则 Ferreira 算法所需的时间为 $O\left(m+\sum_{c\in C(G)}|c|\right)$.其中,$O(m)$是读图所必须的时间,而 $O\left(\sum_{c\in C(G)}|c|\right)$ 是输入环路所必须的时间.因此,该算法是渐进最优的.

综上所述,类似的算法在静态图上已经有了良好的表现,但是它们并不能被直接用于带有时序的交互网络.这是由于静态图中没有附加其他信息,整个图更为简单;而时序图上的环在普通环的基础上,要求时序递增,提升了复杂度.再者,时序图上两点之间极有可能出现多条带有不同时间戳的时序边,而在静态图上,两个点之间往往可以简化为只有一条边.

由于近年来对时序图网络研究的兴起,以及对时序环重要性认识的不断提高,已经有一些专家学者针对时序环进行了研究.阿里巴巴在 2018 年提出了限制条件下的实时环路检测[16],其中的限制包括长度限制、重要性限制等.该算法针对不断更新的时序图网络,按照用户的要求,将符合条件的环路返回给用户.算法中定义了热点(hotpoint)的概念,表示一些交互活动频繁、重要性较高的点,而这些热点往往是带来较大查询时延的瓶颈.于是,算法为这些热点维护特殊的索引,这样一来,热点带来的查询复杂度就可以降低.然而,这个算法查询的环路并不是严格遵循时序递增的时序环,因此该技术不能用于本文的研究问题.

与本文所研究的问题最相关的工作有两个.

- 2017 年,Kumar 和 Calders 提出了一种较为朴素的时序环枚举算法[17],该算法在研究时序环的同时,提出了一个论断,认为可以使用简单时序环以及它们的数量分布情况来表现时序网络里的信息流动.具体而言,该算法枚举窗口内所有可能的时序路径来寻找时序环,它记录所有当前已经形成的时序路径,并将新的点不断追加到之前的路径上,直到环路形成.该算法容易理解也便于实现,但是并不适用于大图,当路径越来越多,该算法会占用过多的内存;
- 2SCENT[18]算法是 2018 年由 Kumar 等人提出的时序环枚举算法,也是目前为止最快的时序递增环路枚举算法,它借鉴了 Johnson 算法中的思想,是一个适用于时序图的 Johnson 算法衍生.相对于朴素的算法,它达到了 300 倍以上的时间提升,同时能处理的图的大小也获得了巨大的增加.2SCENT 算法是一个两阶段的算法.

- 在第 1 个阶段(可以被称作源检测阶段),它找到所有的候选信息四元组,其中包括环路顶点、开始时间、结束时间、候选节点集合.为了适应更大的图,该算法还引入了布隆过滤器来完成这一阶段的处理;
- 在第 2 个阶段,利用每一组候选信息进行动态深度优先搜索.为了避免不必要的多次重复搜索,引入了时序图中的“关闭时间”这一概念来进行剪枝.为了处理那些有多重边、高度重复活动的网络,该算法还使用“路径束”进行搜索,使效率得到了提升.

2 问题描述及相关定义

2.1 时序边和时序图

时序边是一条带有时间戳的有向边,表示成三元组(起点,终点,时间戳),其中,时间戳为非负实数,本文中时间戳的单位为秒.一系列时序边的集合被称为时序图 $G(V,E)$.时序图 G 建立在点集合 V 上,由$(u_i,v_i,t_i),i=1,2,\ldots,m$ 构成.$u_i,v_i\in V,m=|E|$,为边集合的大小.

2.2 时序环

时序环是由一组时序边 $p=\langle(u,v_1,t_1),(v_1,v_2,t_2),\ldots,(v_{n-1},u,t_n)\rangle$组成的环路,其中,$t_1<t_2<\ldots<t_n$.在本文中,我们要寻找的环是简单时序环,即环路中不含有重复经过的节点.

2.3 持续时间和时间窗口

对于找到的一条路径 $u\xrightarrow{t_1}v_1\xrightarrow{t_2}v_2\rightarrow\ldots\xrightarrow{t_n}u$,我们定义它的持续时间为 t_n-t_1,它的长度为 n.时间窗口是指我们所要寻找的最大持续时间值.若时间窗口限制为δ,则要求 $t_n-t_1\leqslant\delta$.

在本文中,对于给定的时间窗口,我们只寻找持续时间小于等于时间窗口的简单时序环.

2.4 长度限制的查询

对于某个时间窗口ω,若用户只要求查找长度小于 L 的环路,则将这样的查询称为长度限制为 L 的查询,这样的环路被称为约束时序环.当 L 足够大时,获得的是全部的环路.

2.5 问题定义

本文具体研究的问题是:对于给定的时序图 $G(V,E)$、时间窗口ω、长度限制 L,枚举所有长度 $l<L$、持续时间 $w\leqslant\omega$的简单约束时序环.

3 新型环枚举算法设计

在最朴素的算法下,直接对图上的每一个顶点进行深度优先搜索或广度优先搜索,在很多情况下会产生巨大的损耗,不仅在时间上远远不能满足现实的需求,也极容易造成空间的不足,使得系统崩溃.这主要是由于其中一大部分的搜索是无效的,并不能对形成环路起到作用,但是它们却会被不断存储下来.因此,如果我们能知道环路中会经过的顶点以及大致时间再进行搜索,就可以降低搜索的复杂度.

在寻找环的过程中,常使用两阶段的算法:在第 1 阶段确定可能成为环的结构,在第 2 阶段对这些结构进行进一步确认.本文基于 2SCENT 算法,使用两个阶段的方法来寻找满足要求长度的简单环路:在第 1 个阶段,遍历所有的时序边,获得环路起点、环路的起始时间、环路的结束时间、可能经过的节点集合、环路最小跳数这 5 个信息;在第 2 阶段,利用以上信息,进行动态深度优先搜索,结合剪枝技术,最终获得要求的环路.

3.1 获取五元组算法

产生候选五元组模块的五元组是指⟨环路起点 s,环起始时间 ts,环结束时间 te,候选节点集合 $candidates$,最小跳数 hop⟩.其中,候选节点集合是环路中可能会经过的节点的集合,最小跳数是指通过该五元组进行搜索能获得的环路的最小长度,它可以给出一系列环路的长度下限.

按照时间戳递增的顺序读入时序边.假设边表示信息的传递过程,则由一条输入边 $a \xrightarrow{t} b$ 可得:节点 a 所拥有的信息在时间 t 传递到节点 b.由于在时间 t 之前还可能有一系列的时序边到达点 a,因此 a 所拥有的信息里其实包含了所有在 t 之前传递给 a 的来自其他节点的信息.我们用"到达"来表示某个节点拥有的信息传递到另一个节点,则 $a \xrightarrow{t} b$ 表示 a 可以经由时间 t 到达 b;同时,所有可以到达点 a 的节点也可以到达点 b.定义连通集合 $Arrive(X)$为可以通过一系列时序递增的边到达 X 的节点集,则通过这一条输入边,a 以及 $Arrive(a)$中的节点会加入 $Arrive(b)$.

以此类推,若最终在某个点 j,发现 j 的连通集合中拥有自身,则证明存在一条由 j 出发、经过递增的时序边回到 j 的时序环.但是由于时间窗口的限制,有一些点已经超出时间窗口(时间戳过小),不应该被加入连通集合.因此,除了记录节点编号,同时也记录由该点出发的边的时间戳,以便及时删除应当废弃的信息,避免占用过多内存.为了实现这一点,我们在 *Arrive* 集合的基础上用二元组⟨节点,出发时间⟩来代替原先单纯的节点信息.如 $Arrive(X)$中包含的二元组$\langle n,t\rangle$表示 n 从一条以 n 为起点的、时间戳为 t 的时序边开始,沿着时序递增路径最终可以将信息传递给节点 X,此二元组也可以称为被 X 所拥有.

除了以上的两个信息之外,需要使用最小跳数来帮助我们处理与环路长度有关的信息.跳数的值等于经过的时序边的数量,若点 a 到点 b 的跳数为 n,则代表点 a 经过 n 条边可以到达点 b.因此,最小跳数表示至少要经过的边数.对于每一个⟨节点,出发时间⟩的二元组,附加一个最小跳数的信息,表示该节点到拥有这个二元组的节点的最小距离(至少要经过的边数).

进一步扩展 *Arrive* 集合,除了⟨节点,出发时间⟩之外,加入最小跳数域,我们用三元组⟨*node*,*time*,*hop*⟩来进行表示.对于每个点 s,用集合 S 来表示它所拥有的三元组集合,即:

$$S[s]=\{\langle node_1,time_1,hop_1\rangle,\langle node_2,time_2,hop_2\rangle,\ldots,\langle node_n,time_n,hop_n\rangle\}.$$

用这个集合来表示 $node_i$ 从时间戳为 $time_i$ 的时序边开始,经过最少 hop_i 条时序边可以到达点 s.对于一条边 $a \xrightarrow{t} b$,若 $S[a]$中含有三元组$\langle node_i,time_i,hop_i\rangle$,则在时间 t,$node_i$ 的信息传递到 b,使 b 中也出现一个由 $node_i$ 从 $time_i$ 出发的三元组,而这个三元组的 *hop* 值显然与 a 中三元组的 *hop* 值有关联.我们将点 b 在时间 t 根据 a 的集合 $S[a]$来更新自己的三元组集合 $S[b]$的过程称为"继承".则 b 在继承这个三元组时会出现两种情况.

- 若 $S[b]$中不含有前两个域为$\langle node_i,time_i\rangle$的三元组,则插入$\langle node_i,time_i,hop_i+1\rangle$这样的三元组.显然,这是由于从 a 到 b 又经过了一条边,因此,$node_i$ 到 b 的距离应该为其到 a 的距离加一,即 hop_i+1;
- 另一种情况下,若 $S[b]$中已经含有这样的三元组,则需要将这个三元组中的 *hop* 值与 hop_i+1 进行比较,并取其中较小的那个,以此来确保跳数是最小的.

从集合 S 的定义可以得到获得环路的方法,如在处理一条 $i \xrightarrow{t} s$ 的边后,发现 $S[s]$中出现了三元组$\langle s,t',hop\rangle$,则证明找到了环路.同时,我们还能获得该环路集合的起始时间为 t',结束时间为 t,环路最小长度为 *hop*.

由上述已知可能有环路由 s 从时间 t'出发最终在 t 时刻返回 s,那么对于 $S[s]$中所有的三元组,若其对应的时间戳在时间 t'到 t 之间,则这个三元组中的节点就成为此环路中可能经过的节点.这样的节点们构成了候选节点集合,表示一系列可能出现在环路中的节点的集合.这样的节点集合并不是完全准确的,它是环路中真正经过的节点的超集.为了进一步缩小节点的范围,还需要通过 *hop* 值来排除一部分不会在满足要求的环路中出现的点:对于一条边 $a \xrightarrow{t} b$,若在继承来自于点 a 的三元组时,发现三元组为$\langle b,t',hop\rangle$,这说明可以形成新的环路;但如果生成的新 *hop* 值(即 *hop*+1)已经大于要求的最大长度 L,则不生成五元组,这是因为通过该五元组形成的环必然超长.若发现另一三元组不能成环并且新 *hop* 值大于等于 L,则不进行三元组的继承.显然,通过这个三元组里的点到点 b 的路径均大于等于 L,因此这个三元组里的跳数域最后一定会被更新成更小的合法值或是直接被舍弃.然而,这样的舍弃并不代表这个三元组中的节点不会成为环路的一部分,未来它仍有可能获得更小的 *hop* 值而得以进入候选节点集合.

如图 2 所示,若当前我们想要寻找长度小于 5 的环路.在处理了时间戳 $3 \xrightarrow{3} 4$ 后,我们有 $S[4]=\{\langle 3,3,1\rangle,\langle 2,2,2\rangle,\langle 1,1,3\rangle\}$;但在处理边 $4 \xrightarrow{4} 5$ 时,我们不会继承⟨1,1,3⟩这个三元组,这是由于此时新的 *hop* 值将更新为 4 但仍未成环,因此无需将其保存下来.至此,我们找到所有五元组.需要注意:一个五元组并不代表一条环路,而是

代表拥有同一个起点、起始时间、结束时间的一系列环路.算法 1 中给出了寻找候选五元组函数的伪代码,其中,*S* 集合代表节点拥有的三元组信息,*SearchLimit* 值为人为给定的限制长度值.

同理,在使用布隆过滤器[19]进行处理时,我们仍然可以利用 *hop* 值来降低搜索的复杂度.这是因为在使用布隆过滤器时,为了满足获取候选信息的需要,要对原图进行一次反向遍历和一次正向遍历[18].而在进行正向遍历的过程中,就可以运用三元组进行操作,操作的方式与上述方式是基本一致的.

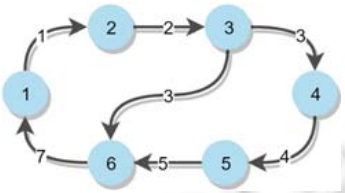


Fig.2 Reduce the size of candidates using *hop*

图 2 通过 *hop* 缩小候选节点集合

算法 1. *set_interaction_exact*(*G*).

输入:读取时序图的输入流.其中,要求图输入的格式为(起始节点,终止节点,时间戳)的三元组,输入时间不递减;

输出:⟨起点,起始时间,结束时间,候选点集,最小跳数⟩的五元组.

```
1.  while (读取时序边⟨from,to,timestamp⟩)
2.    if (self_loop) continue;
3.    S[to]←pair((from,timestamp),1);
4.    if (S[from]不为空)                          //进行继承
5.      for (S[from]中的三元组⟨node,time,hop⟩)
6.        if (该三元组的时间已经在时间窗口外)      //清理
7.          从 S[from]中删除这个三元组;
8.        else
9.          new_hop←hop+1;
10.         if (to==node)                         //找到了环路
11.           if (new_hop≤SearchLimit)
12.             记录五元组信息;
13.           else                                //还要继续进行搜索
14.             if (new_hop<SearchLimit)
15.               if (S[to]中不含有这个三元组)
16.                 S[to]←((node,time),new_hop);
17.               else
18.                 if (原 hop 值>新的 hop 值)
19.                   更新这个三元组的 hop 值;
20.   if (到了程序约定的清理时间)
21.     清除已经过期的三元组;
```

3.2 搜索算法

在这一模块中,我们将沿用 2SCENT 算法中的剪枝方式[18],利用关闭时间和封锁节点的策略来帮助我们进行搜索.虽然通过第 1 步操作我们已经减少了很多不会被搜索到的点,但是当图变得越来越大,搜索的长度越来越长,一些搜索已经超过了要求的长度却仍在进行.这样的搜索增加了寻找环路所需的时间.若想在 2SCENT 算

法的基础上进行改进,按照比较直接的想法,获取小于某一长度 L 的环的办法可以是:从点 s 开始搜索,发现搜索长度大于等于 L 之后,就立刻停止搜索,因为继续搜索下去,获得的环路长度一定超过 L.但是我们可以证明,在2SCENT 的剪枝方法上直接使用这个想法会导致非简单环路的发生.

这个错误发生的原因,是 2SCENT 算法中关闭时间的设置.如图 3 所示:若我们想要所有长度小于 5 的环路,当前搜索路径为 $1\xrightarrow{1}2\xrightarrow{4}3\xrightarrow{5}4\xrightarrow{7}5$ 时,我们发现路径长度已经为最大值 4,但是仍然没有成环,说明可以停止对这段路径接下来的搜索.此时,由于搜索停止,节点 4 会加入到节点 5 的阻塞列表中.以此类推,最终节点 2 加入节点 3 的阻塞列表.当搜索程序回退,当前搜索路径成为 $1\xrightarrow{1}2\xrightarrow{5}5\xrightarrow{9}6$ 时,我们发现了一条长度为 4 的环路.此时,由于环路成立,节点开始解锁,第 1 个解锁的节点是节点 6.由于节点的解锁是一系列的级联反应,最终节点 2 的关闭时间被更新为 4.此时,节点 2 的关闭时间就已经开始出现了问题,节点 2 明明在当前道路中,关闭时间却异常地变大了.这样会导致未来其他节点通往节点 2 时,由于关闭时间的不正常放大,节点 2 可能会再一次出现在搜索路径中,形成非简单环路.

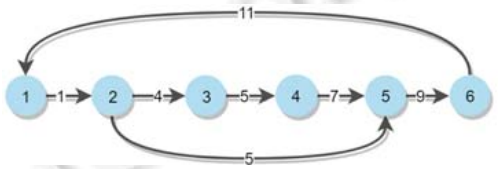


Fig.3 An example of wrong closing time

图 3 错误的关闭时间举例

根据关闭时间的定义可知,关闭时间是节点无法通往环路起点的最小时间值,即大于等于关闭时间的时间戳都无法通往环路起点.一个点的关闭时间来自于由该点出发的时序边上的时间戳,而这个时间戳的选择是由后续的搜索来决定的.对于一个点 i,若有一系列时序边通向 i,则只有时间戳小于点 i 关闭时间的边可以被当作是合法的前序路径.因此,如果从某个点 j 指向点 i 的所有边的时间戳都大于等于 i 的关闭时间,则点 j 没有必要向 i 进行搜索.显然,某个点的关闭时间越大,则通往该点的时间戳能通过的值越大.因此,由关闭时间的性质可知:偏大的关闭时间会导致一部分无效的搜索;而偏小的关闭时间却会导致环路数量的错误,本不应该存在的环路由于关闭时间的错误而产生.为了防止这样的错误发生,同时尽可能地利用剪枝搜索带来的好处,我们在搜索过程中这样处理:当搜索到达要求的长度时,不通过完整的搜索我们无法得知最终是否能形成环路,但对于这样的点,我们都默认可以形成环路,以此防止图 3 中的错误发生,同时尽可能保留剪枝技术.这样的处理会导致一部分剪枝失效(在较差的情况下甚至起效很少,仅可以防止环路中节点的重复搜索),但是由于长度限制对搜索时间的减少很大,虽然剪枝的作用有所降低,总体而言仍是一个合适的策略.

如图 4 所示,若要求找到所有长度不超过 5 的约束环.当前搜索路径为 $1\xrightarrow{1}2\xrightarrow{4}3\xrightarrow{6}4\xrightarrow{8,11}5$,此时,点 5 向点 7 进行搜索,发现长度已经为 5,则搜索终止,并将 5 的关闭时间设置为 9,向点 6 进行搜索时同理.由于并没有从点 6 和点 7 开始进行搜索,因此这两个点的关闭时间值仍然为程序初始化时设置的最大值.下次从点 4 向点 7 进行搜索,即搜索前缀为 $1\xrightarrow{1}2\xrightarrow{4}3\xrightarrow{6}4\xrightarrow{9}7$ 时,搜索可以继续进行,最终获得合法的环路.

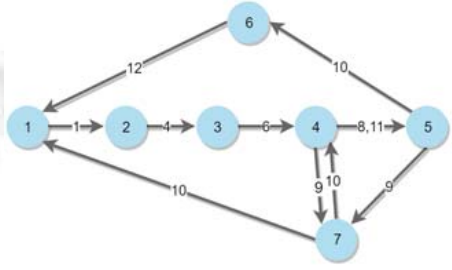


Fig.4 An example of constrained search

图 4 限制下的搜索举例

以上是伴随剪枝的搜索算法的大致流程,为了更清楚地描述这些步骤,我们给出了伪代码.算法 2 为搜索算法的主体,其中的 *subgraph* 是根据第一阶段获得的五元组构建的子图,*SearchLimit* 值是人为给定的限制长度,*ct* 值为关闭时间值.*Extend* 函数和 *UnBlock* 函数用于封锁节点和解锁节点[18].

算法 2. *ALLPATH*(*s*,*vcur*,*timestamps*,*stk*).

输入:环路起始点 *s*,当前节点 *vcur*,当前时间戳向量 *timestamps*,先前路径束的堆栈 *stk*;

输出:所有要求内的路径.

```
1.  tcur←timestamps 中的最小值;
2.  ct[vcur]←tcur;
3.  lastp←0;
4.  ts←{(vcur,x,t)∈subgraph|tcur<t≤ct(x)}
5.  for (ts 中的节点值 to)
6.    if (to==s)                                  //找到环路
7.      T←{t|(vcur,to,t)∈ts};
8.      maxT←max(T);
9.      lastp←max(lastp,maxT);
10.     扩展 new_stk;                             //将新的路径加入到之前搜索的路径中
11.   else
12.     ret←0;
13.     tss←{t∈ts|t<ct(x)};
14.     if (tss 不为空)
15.       扩展 new_stk;
16.       if (new_stk.size(·)<SearchLimit)
17.         ret←ALLPATH(s,to,tss,new_stk);
18.       Else                                    //最大的时间戳
19.         ret←new_stk.back(·).timestamps.back(·)+1;
20.     num←min{t∈ts|t≥ret}
21.     Extend(to,vcur,num);
22.     if (ret)
23.       num←max{t∈ts|t<num};
24.       lastp←max(lastp,num);
25. if (lastp>0)                                  //更新关闭时间
26.   UnBlock(vcur,lastp);
27. return lastp;
```

4 实验与分析

4.1 实验设置

4.1.1 实验环境

我们使用 C++来实现此算法.所有的实验都运行在一台 Intel Corei5-8400 CPU@2.80GHz CPU 和 8GB 内存的台式机上,操作系统为 Windows 系统.

4.1.2 实验数据集

为了验证算法的效率与实用性,本文使用以下 4 个真实的时序图数据集来进行对比实验,这 4 个数据集均来自 SNAP 数据集[20](见表 1).

(1) Wiki-talk 数据集.该数据集记录了 Wikipedia 用户修改其他用户的讨论页的行为;

(2) CollegeMsg 数据集.该数据记录了加州大学欧文分校在线社交网络中的信息发送情况;

(3) Higgs-twitter 数据集.该数据集记录了 twitter 上用户的信息往来行为;

(4) Stack-overflow 数据集.该数据集记录了 stackoverflow 网站上的用户往来行为.

Table 1 Experiment datasets

表 1 实验数据集

数据集	点数	边数	持续时间(天)
Wiki-talk	1 140 149	7 833 140	2 320
CollegeMsg	1 899	20 296	193 000
Higgs	304 691	563 069	7
Stackoverflow	2 464 606	17 823 525	2 774

4.1.3 实验设计

由于朴素的搜索算法效率过低且目前没有类似的约束时序环搜索算法,我们以 2SCENT 算法为基准来进行对比.2SCENT 算法在搜索到环路后可以舍弃不符合要求的环路,最终达到本算法所要的输出结果.实验将以时间窗口以及限制长度作为变量,针对算法运行所需要的时间(包括获得五元组时间和深度优先搜索时间)、内存等方面进行比较.为了保证其他变量不对实验造成影响,在设置布隆过滤器时,我们使用统一的参数:我们设置布隆过滤器的单边错误发生率为 0.01%,设置一个布隆过滤器中可能存的数为 1 000 个.

4.2 元组数量比较

在算法中,我们通过候选节点集合先对五元组进行了初步筛选(2SCENT 算法获得的其实为四元组,为了叙述方便,之后不再特意指出),去掉了只有两个候选点(即最多形成二元环路)的五元组.从表 2 可以看出:通过最小跳数值进行筛选,通过这些数据集获得的元组数量都有所减少,但是缩减的程度根据图的不同有较大的变化,最小的变化量只有几十,大的缩减可以上千.这也可以侧面说明,在不同结构的图中,同一长度的环数量差别较大.

Table 2 Comparison between number of tuples

表 2 元组数量比较表

数据集	窗口及最大长度		2SCENT	本文
Wiki-talk	24h	4	159 313	158 109
	36h	4	174 354	171 465
CollegeMsg	40h	5	5 147	4 870
	50h	5	5 663	5 221
Higgs	10h	4	2 526	2 496
	20h	4	2 800	2 742
Stackoverflow	8day	4	4 578	3 486
	10day	4	5 756	4 176

4.3 运行时间比较

4.3.1 获取五元组时间

表 3 列出了 2SCENT 算法和本文算法的两种获取元组方式的大致运行时间.首先,我们横向比较同一算法的两种获取五元组方式的运行时间.从表中可见:对于大多数图,使用布隆过滤器的方式更加耗时.产生这个现象主要原因是:使用布隆过滤器的方式需要对原图进行正反两次遍历,毫无疑问,这增加了算法运行所需时间.但是并不是所有图都有这样的现象.对于一些图而言,若图本身较小,或者图中需要继承的三元组过多,则布隆过滤器会更优,这是因为布隆过滤器进行的是位向量的并操作,运行速度快.

另一方面,我们纵向比较不同数据集,同一方式之间的时间.由于在 2SCENT 算法中无法预知环路的长度,因此想获得某一长度的环路也必须先对所有元组进行查找.但是由于 *hop* 的存在,我们的算法在查找小于某一长度的时序环时显然更占优势.

从表格里可以看到:使用 *hop* 进行筛选有时,可以节约原时间的 70%以上.时间的减少主要来自两方面:第 1

点是合并五元组所需的时间变少,这一点可以结合表 2 得到;第 2 点是 *hop* 值避免了大量不必要的三元组更新与继承.在大多数图里,第 2 点节约的时间超过第 1 点.

Table 3 Comparison between time of getting tuples

表 3 获取元组时间比较

数据集	窗口及最大长度		2SCENT		本文	
			Exact	Bloom	Exact	Bloom
Wiki-talk	24h	4	136.0	152.5	92.4	120.8
	36h	4	294.4	263	142.6	159.6
CollegeMsg	40h	5	18.0	23.4	6.5	8.4
	50h	5	30.2	41.3	10	12.9
Higgs	10h	4	31.7	11.0	9.3	7.7
	20h	4	63.8	17.5	13.4	8.9
Stackoverflow	8day	4	131.4	218.3	101.5	191.8
	10day	4	169.4	256	111.4	203.0

4.3.2 寻找环路时间

表 4 展示了利用不同方式获得的元组来进行动态深度优先搜索所消耗的时间.首先进行横向比较,即同一算法下,不使用布隆过滤器和使用布隆过滤器获得的元组,它们在进行搜索时消耗的时间也会有不同.从表中可见:对类似 CollegeMsg,Wiki-talk 和 Stackoverflow 的图来说,两种方式下搜索的时间大致相同.但是在 higgs 图中却有较大的区别.产生这样的区别的原因在于:采用布隆过滤器的方式有时也能进一步缩小候选节点集合,但是缩小的程度因图而异.

Table 4 Comparison between time for dynamic bfs

表 4 动态深度优先搜索算法时间比较

数据集	窗口及最大长度		2SCENT		本文	
			Exact	Bloom	Exact	Bloom
Wiki-talk	24h	4	35.1	34	29.7	28.8
	36h	4	60.7	60.5	41.3	41.7
CollegeMsg	40h	5	44.9	44.3	18.9	18.9
	50h	5	260.5	257.9	116.2	112.6
Higgs	10h	4	8.5	3.8	3.9	2.2
	20h	4	66.9	50.0	15.8	11.8
Stackoverflow	8day	4	53.7	53.0	44.0	43.1
	10day	4	81.0	78.0	46.6	46.0

接下来比较 2SCENT 算法与本文算法,显然,我们的算法在获取小于等于某一长度的时序环时消耗的时间比原算法少,有时减少的时间可以达到之前的 70%以上.算法起效的原因主要有 3 个方面:第一,一般而言,长环路的搜索更为复杂,搜索时间更长,因此我们减少的五元组可以使搜索时间有一定的提升;第二,我们的算法进一步减少了候选节点集合的大小,这避免了一部分初始化工作;第三,我们在达到限制长度时及时地停止了搜索,避免为产生不符合要求的环消耗时间.

4.4 运行内存比较

表 5 为获取候选元组(不包括元组合并)阶段,算法在运行过程中所占用的最大内存.首先,我们可以发现:使用布隆过滤器相对于原始的方式总是可以节约大量内存;且随着时间窗口的增大,内存的增长并不大,这体现了布隆过滤器在节约内存方面的巨大作用.

在表格中,除了 stackoverflow,在其他图上,本文算法的内存占用总是优于原算法.在该图上,内存消耗反而增加.为了分析这一现象可能发生的原因,我们将 wiki-talk 与 stackoverflow 作对比可以看到:两者的时间下降比例较为相似,但是在内存表现上却差距较大.我们通过实验观察了这两个图在算法运行过程中所避免继承的三元组数量的情况,我们发现:在 wiki-talk 中,这个值大约为 3.4×10^7;而在 stackoverflow 中,这个值约为 4.6×10^6,其中存在着较大的差距.这是由于相对于 2SCENT 算法,我们的算法由于引入了 *hop* 值增加了一定的内存占用,但是也因为 *hop* 避免了一部分三元组继承以减少内存占用.因此,总体内存相对于 2SCENT 算法的增减来自于增

加的内存和减少的内存的差值.但是在使用布隆过滤器的算法内均没有出现内存增加的现象,这是因为布隆过滤器的正向遍历阶段已经将范围缩小的很多,导致继承的数量变少,*hop* 值占用也变少.

虽然不使用布隆过滤器的算法在较坏的情况下会出现一定的内存增加,但是由于 *hop* 值本身占用内存较小,因此预计增加的内存会在可接受的范围内.表格中有些情况下,2SCENT 算法和本文算法占用的内存相同.这是由于在使用布隆过滤器时,正向遍历时所占的内存比反向遍历时所占的小,*hop* 值会被用于正向遍历;而我们记录的是两阶段中占用内存的最大值.

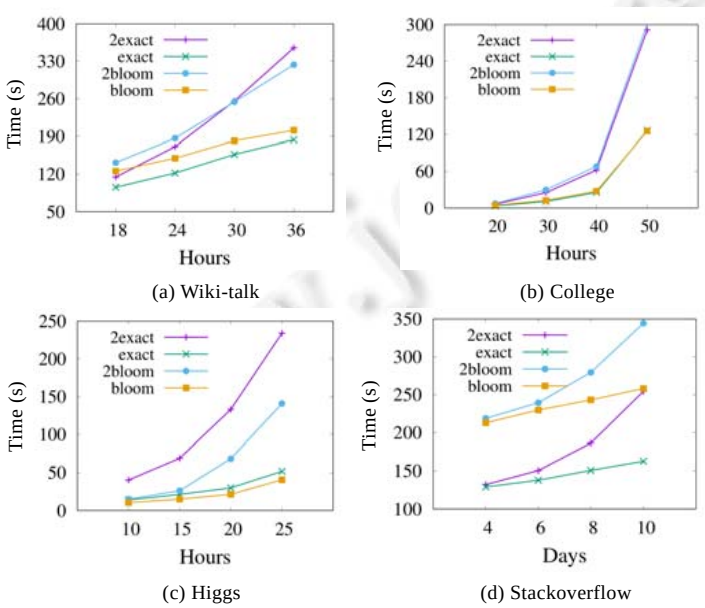
Table 5 Comparison between memory usage

表 5 获取元组算法内存比较表

数据集	窗口及最大长度		2SCENT		本文	
			Exact	Bloom	Exact	Bloom
Wiki-talk	24h	4	1324	124	1143	120
	36h	4	2787	230	1704	125
CollegeMsg	40h	5	31	38	17	16
	50h	5	56	70	25	30
Higgs	10h	4	254	421	84	421
	20h	4	529	521	125	521
Stackoverflow	8day	4	1049	377	1219	377
	10day	4	1228	390	1338	390

4.5 总体比较

图 5 为算法在 4 个图上的总体运行时间比较图,其中,4 个图的长度限制分别为 4,5,4,4.我们取不同的窗口大小作为横坐标,来衡量算法的时间表现.其中,2exact 表示 2SCENT 算法不使用布隆过滤器来获得四元组,2bloom 表示 2SCENT 算法用布隆过滤器获得四元组,同理,exact 和 bloom 对应本文算法获取五元组的方式.由图可见:在获取小于一定长度的环路时,本文算法的运行时间总是比 2SCENT 算法的运行时间短.且从折线的斜率变化可得:图越大,2SCENT 算法所增加的时间越大,而本文算法可以使消耗时间的增加保持一个相对较为平稳的趋势.也就是说,这样的时间差距会随着图的增大变得更为显著.因此,我们的算法对于图的扩大具有良好的稳定性.



(a) Wiki-talk (b) College

(c) Higgs (d) Stackoverflow

Fig.5 Comparison with different time windows

图 5 不同时间窗口比较图

图 6 为本文算法在 4 个图上以长度限制为变量的搜索时间变化图,其中,exact 是指本算法不使用布隆过滤器获得信息后进行搜索的深度优先搜索时间,bloom 是指本算法使用布隆过滤器获得信息后进行搜素的深度优先搜索时间,cycles 为使用路径束记录的符合要求的环路总数量.图中横坐标为最大的环路长度.从图上我们可以看出:在不同的图里,不同长度的环路数量差异很大.在 wiki-talk 中,小于 4 的环占了大多数.但在其他图中,环的数量随着长度有较大的增幅.这说明:在类似维基百科的网络中,用户之间的修改行为是较为随机的,因此缺少较长的环;而在其他更具有社交性质的环中,人与人之间的交流联系更紧密,倾向于长的沟通环路,用户发送的信息可能被多次传递,最后又传递给了最初的发送人.

从柱状图的变化趋势和算法搜索时间的趋势我们也可以发现:当数量增幅较大时,折线的斜率也有变大的倾向.这是由于巨大的环路增幅会导致搜索消耗时间也出现突然增长,这符合我们的预期结果.

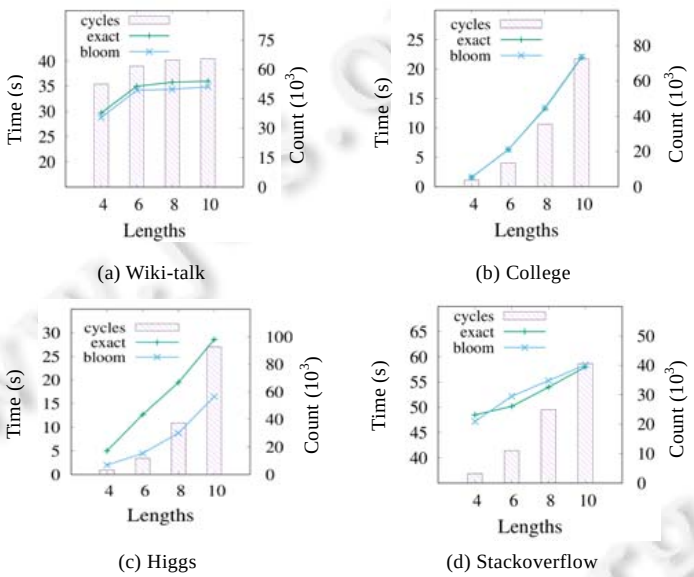


Fig.6 Comparison with different lengths

图 6 以长度为变量的比较图

5 总结与展望

本文在 2SCENT 算法的基础上,提出了一种新型的两阶段环路枚举算法.相对于 2SCENT,它减少了搜索所需的时间,并且在大多数情况下都能节约一定的空间.本文主要完成的工作是:在算法的第 1 阶段,即五元组获取阶段,获得更多有关环路的信息,具体而言就是环路的最小跳数;并且利用最小跳数,帮助我们在算法的第 2 阶段节省更多的时间.

由于时间和技术难度等因素的限制,本算法对环路枚举速度的改进是有限的.最小跳数域可以获得某一五元组的最小环路长度,反过来,也可以将这个域设置成最大跳数域.但是在计算最大跳数时,由于跳数的不准确性,导致预计的环路长度容易出现过大的情况.在较为极端的情况下,这样的增长可能会导致跳数域失效.然而,精确地计算出跳数必然会引入不能忽略的时间消耗.希望未来能够有更加准确而快速的最大跳数计算方式.

在计算环路时,仍然会有相同子路径被多次重复枚举的情况.若能尽可能地降低这样的重复,必定能大大降低时间复杂度.然而在目前的剪枝情况下,已经尽可能地避免了浪费,因此,提出进一步地优化显得更为困难.

References:

[1] Dong CY, Shin D, Joo S, Nam Y, Cho KH. Identification of feedback loops in neural networks based on multi-step granger causality. Bioinformatics, 2012,28(16):2146−2153.

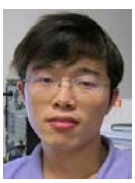
[2] Hoffmann F, Krasle D. Fraud detection using network analysis. 2015. EP Patent App. EP20,140,003,010.
[3] Holme P, Sarama¨ki J. Temporal networks. Physics Reports, 2012,519(3):97−125.
[4] Wang YS, Yuan Y, Liu M. Survey of query processing and mining techniques over large temporal graph database. Journal of Computer Research and Development, 2018,55(9):1889−1902 (in Chinese with English abstract).
[5] Jiang ZH. Minging frequent evolution pattern on temporal network. Modern Computer, 2019,638(2):15−19 (in Chinese with English abstract).
[6] Wu AB, Yuan Y, Qiao BY, Wang YS, Ma YL, Wang GR. The influence maximization problem based on large-scale temporal graph. Chinese Journal of Computers, 2019:1−18 (in Chinese with English abstract).
[7] Kovanen L, Karsai M, Kaski K, Kert´esz J, Sarama¨ki J. Temporal motifs in time-dependent networks. Journal of Statistical Mechanics: Theory and Experiment, 2011(11):P11005, 2011.
[8] Paranjape A, Benson AR, Leskovec J. Motifs in temporal networks. In: Proc. of the 10th ACM Int'l Conf. on Web Search and Data Mining. ACM, 2017. 601−610.
[9] Rao VVB, Murti VGK. Enumeration of all circuits of a graph. Proc. of the IEEE, 1969,57(4):700−701.
[10] Ponstein J. Self-Avoiding paths and the adjacency matrix of a graph. SIAM Journal on Applied Mathematics, 1966,14(3):600−609.
[11] Mateti P, Deo N. On algorithms for enumerating all circuits of a graph. SIAM Journal on Computing, 1976,5(1):90−99.
[12] Tiernan JC. An efficient search algorithm to find the elementary circuits of a graph. Communications of the ACM, 1970,13(12): 722−726.
[13] Weinblatt H. A new search algorithm for finding the simple cycles of a finite directed graph. Journal of the ACM (JACM), 1972, 19(1):43−56.
[14] Johnson DB. Finding all the elementary circuits of a directed graph. SIAM Journal on Computing, 1975,4(1):77−84.
[15] Birmelé E, Ferreira R, Grossi R, *et al*. Optimal listing of cycles and st-paths in undirected graphs. In: Proc. of the 24th Annual ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2013. 1884−1896.
[16] Qiu X, Cen W, Qian Z, *et al*. Real-Time constrained cycle detection in large dynamic graphs. Proc. of the VLDB Endowment, 2018, 11(12):1876−1888.
[17] Kumar R, Calders T. Finding simple temporal cycles in an interaction network. In: Proc. of the Workshop on Large-Scale Time Dependent Graphs (TD-LSG 2017), Co-Located with the European Conf. on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2017). Skopje, 2017. 3−6.
[18] Kumar R, Calders T. 2SCENT: An efficient algorithm for enumerating all simple temporal cycles. Proc. of the VLDB Endowment, 2018,11(11):1441−1453.
[19] Bloom BH. Space/Time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970,13(7):422−426.
[20] Leskovec J, Krevl A. SNAP datasets: Stanford large network dataset collection. 2014. https://snap.stanford.edu/data

附中文参考文献:

[4] 王一舒,袁野,刘萌.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.
[5] 蒋志恒.时序网络的频繁演化模式挖掘.现代计算机(专业版),2019,638(2):17−21.
[6] 吴安彪,袁野,乔百友,王一舒,马玉亮,王国仁.大规模时序图影响力最大化的算法研究.计算机学报,2019:1−18.

潘敏佳(1996－),女,硕士,CCF 学生会员,主要研究领域为图数据挖掘.

赵宇海(1975－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,数据挖掘,机器学习,软件工程,生物信息学.



李荣华(1985－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为图数据管理,图数据挖掘,社交网络分析,图机器学习,图计算系统.

王国仁(1966－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为不确定数据管理,数据密集型计算,可视媒体数据管理与分析,非结构化数据管理,分布式查询处理与优化技术(主要包括传感器网络和 P2P 对等计算),生物信息学.