

面向大规模时态图的紧密子图维护算法

车鑫恺¹, 陈雅迪¹, 胡 森^{1,2}, 吴 迪^{1,2*}

(1. 中山大学计算机学院, 广州, 510006; 2. 广东省大数据分析处理重点实验室, 广州, 510006)

摘 要: 大规模图数据中的重要顶点与层级结构对于挖掘复杂网络(如社交网络、交通网络等)中有价值的信息具有重要意义. 提出一种自顶向下的大规模时态图 (k, h) -维护算法, 对时态图中紧密度最高的前 n 层 (k, h) -核, 或满足自定义 k, h 值约束条件的核进行维护. 首先提出识别 (k, h) -最大层的方法. 当时态图中出现新的边时, 为了定位当前时刻可能因新加入边导致核值需要更新的顶点的范围, 提出候选插入子图与部分 (k, h) -核的概念及相应的识别算法. 针对加边情况, 提出自顶向下的时态图 (k, h) -核维护加边算法, 根据部分 (k, h) -核识别核值受加边影响的顶点并对其核值进行更新. 针对当前时刻有已经存在的边被删除的情况, 提出自顶向下的时态图 (k, h) -核维护删边算法, 对上一时刻的 (k, h) -核做最小调整以得到当前时刻的核值. 从理论上证明了算法的正确性, 还在真实的时态图上设计了一系列对比实验. 实验结果表明, 在维护层数较少时下添加边, 提出的核维护算法与其他对比算法相比, 加速比可达几十倍; 删边时, 加速比也在 1~2 倍. 提出的算法有良好的扩展性, 对于增删不同数量的边和不同的 (k, h) 设置, 都能保持较高的效率.

关键词: 大规模图, 核值维护, 紧密子图, 时态图

中图分类号: TP391

文献标志码: A

Cohesive subgraph maintenance algorithms for large-scale temporal graphs

Che Xinkai¹, Chen Yadi¹, Hu Miao^{1,2}, Wu Di^{1,2*}

(1. School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, 510006, China;

2. Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, 510006, China)

Abstract: The important vertices and hierarchies in large-scale graph are of great significance for mining valuable information in complex networks (e.g., social networks, traffic networks). In this paper, we propose a top-down (k, h) -core maintenance approach which can maintain only the top- n layers of the densest (k, h) -cores or the (k, h) -cores satisfying certain constraints. Firstly, we propose a method to identify the densest layer so as to locate the influenced scope of inserted edges. We also propose the concepts of candidate graph and partial (k, h) -core and provide their corresponding detection methods. With the help of partial (k, h) -core, our insertion algorithm can find the minimum incremental graph for each influenced (k, h) -core. For the removal case, our removal algorithm makes the minimum adjustment on previous (k, h) -cores to obtain current (k, h) -cores. We also theoretically prove the correctness of our algorithms. To verify the effectiveness of our methods, we conduct extensive experiments on several real temporal graphs. Experimental results show that our top-down (k, h) -core maintenance algorithms can accelerate dozens of times when adding edges and the number of maintenance layers is small. When deleting edges, the speedup ratio can reach 1 to 2 times. Our algorithm is flexible under various (k, h) settings, and maintains high efficiency for adding and deleting different numbers of edges.

Key words: large-scale graph, core maintenance, cohesive subgraph, temporal graph

基金项目: 国家自然科学基金(U1911201, U2001209), 广州市重点研发计划(202103010004)

收稿日期: 2022-03-21

* 通讯联系人, E-mail: wudi27@mail.sysu.edu.cn

时态图是一种多重边动态图,在时态图中任意两个顶点之间可以存在多条边,并且每条边包含额外的时间信息.时态图中的顶点和边会随时间不断发生变化,新的顶点(边)会被插入或删除,因此时态图在多个时间点拥有不同的快照.现实中很多系统都可以被抽象为时态图,如线上社交网络、航空运输网络、蛋白质相互作用网络等,如何在这些复杂网络中挖掘有价值的信息或区域一直是研究人员密切关心的问题.

紧密子图通常由图中联系较为紧密的顶点和边组成,可以揭示复杂网络的变化规律,帮助研究者挖掘复杂网络的结构特点.在过去的研究中,研究者对紧密子图的观点和看法不尽相同.许多研究对 k -核^[1-2]、 k -truss^[3]、最大团^[4-5]、类似团^[6-7]等进行了分析,然而,大多数紧密子图的识别是NP-难问题,难以将理论研究与实际应用相结合从而产生应用价值.作为其中最具代表性的一种紧密子图, k -核是指在一张图中满足每个顶点都有至少 k 个邻居的最大子图.大规模简单图中的 k -核分解算法的复杂度与图中边的数目呈线性关系,在大规模网络可视化^[8]、复杂网络紧密社区发现^[9-10]、社交网络中的影响力分析^[11-12]、复杂网络拓扑分析^[13-14]等领域有广泛的应用.

由于 k -核无法表示时态图中的多重边特性,不能很好地定义时态图中的紧密子图,研究者是在 k -核的基础上提出了 (k, h) -核的概念^[1,15]. (k, h) -核是一张时态图中满足以下条件的最大子图:子图中每个顶点都存在至少 k 个邻居;同时与每个邻居之间都存在至少 h 条边. k, h 越大, (k, h) -核表示的子图越紧密.

与 k -核类似,对 (k, h) -核的研究也主要集中在以下两个问题: (k, h) -核分解和 (k, h) -核维护.识别时态图中存在的所有 (k, h) -核的问题被称为 (k, h) -核分解问题,而 (k, h) -核维护是指在时态图发生变化时基于上一时刻 (k, h) -核分解的结果做出调整以获得当前时刻的 (k, h) -核.在一些情况下使用适当的 (k, h) -核维护算法^[1,16],计算开销远小于重新做一次 (k, h) -核分解.然而,并不是全部的 (k, h) -核都有相同的应用价值, k, h 越大, (k, h) -核表示的子图越紧密,而许多现实应用如社区搜

索、节点影响力排序等,只关注最紧密的 n 层 (k, h) -核.自顶向下、由紧密到稀疏的 (k, h) -核维护算法可以大大减小计算规模,提高计算效率.

本文提出一种面向大规模时态图的 (k, h) -核维护算法,自顶向下地对时态图中的紧密子图进行维护.当时态图的结构随时间发生改变时,从紧密度最高的 (k, h) -核开始更新,在上一时刻 (k, h) -核的基础上做最小调整以得到当前时刻的 (k, h) -核,并利用 (k, h) -核之间存在的部分嵌套关系,依次对余下的最紧密的前 n 层 (k, h) -核进行维护.

1 相关工作

已有学者对各种各样不同条件定义的紧密子图进行了广泛研究,如最大团^[4]、 n -最大团、 n -clan、 k -plex^[17]等.

在简单图中,对图中所有的 k -核进行识别的问题被称为核值分解问题,Batagelj and Zaveršnik^[18]首先提出一种核值分解算法,主要思想是逐步去除不符合 k -核定义的点及与之相连的边,最终得到的图即为 k -核.该算法的实现过程利用了桶排序的思想,令 m 表示图中边的数目,时间复杂度为 $O(m)$.当图数据规模较大无法一次性完全放入内存时,Cheng et al^[19]提出一种高效的使用外部存储的核分解算法 EMCORE,使用一种图划分策略,每次只将大规模图的一部分载入内存.令 K_{\max} 表示图中所有顶点的最大核值,EMCORE 需要遍历整张图 $O(K_{\max})$ 次.然而,EMCORE 无法确定真正需要的内存的规模,而且将许多边的信息也同时载入了内存.Wen et al^[20]提出一种使用半外存模型的算法.此外,也有学者考虑使用分布式计算的方式解决大规模复杂图核值分解的问题.Montresor et al^[21]提出一种经典的以顶点为中心的分布式核值分解算法,在计算一个顶点的核值时,只与该顶点的邻居顶点进行信息交换,利用其邻居的信息来更新自己的核值.

动态图的核值维护即是对因动态图发生变化导致核值发生改变的顶点进行核值更新.Li et al^[22]首次提出动态图中核值维护问题的一种解决办法,是一种由 Color, RecolorInsert, UpdateInsert 三部分组成的 k -核维护算法.同时,还提出两个

剪枝策略,对于有多条边同时需要进行更新的情况,该算法采用对每一条边按顺序依次进行处理的方式完成全部顶点的核值更新. Sariyüce et al^[23]也提出一种采用单条边依次更新方式的 k -核维护算法,利用顶点周围的邻居及其二层邻居的核值信息,根据一定的策略依次对受影响的顶点进行核值更新.

在一种特殊的动态图——时态图中,任意两个顶点之间可以同时存在多条边,因此时态图的紧密子图 (k, h) -核同时考虑每个顶点邻居的数量和任意两个顶点间边的数量,所以对 (k, h) -核的研究比 k -核更困难. Bai et al^[16]提出一种时态图 (k, h) -核维护算法,采用批量更新的方式,利用 (k, h) -核之间的部分嵌套关系,实现自底向上的全部 (k, h) -核维护.

2 背景知识

表 1 为本文使用的数学符号及其含义.

2.1 时态图 时态图是一种特殊的无环无向无权多重图,其中每条边都带有额外的时间信息,用于指明这条边的生存周期,即边从第一次出现到消失的时间段. 对于时态图 $G=(V(G), E(G))$ 中的任意一条边 (u, v, t_s, t_e) , $u, v \in V(G)$ 代表图中被这条边连接的两个顶点, t_s 是这条边被插入到时态图 G 的时刻,即这条边第一次出现在图 G 中的时刻, t_e 是这条边从图 G 中被删除的时刻.

为了方便表述,本文简化了一些符号. 对于时态图 $G=(V(G), E(G))$ 和图 G 中的任意边 (u, v, t_s, t_e) , 通常用 $u, v \in G$ 代替 $u, v \in V(G)$, 用 $(u, v, t_s, t_e) \in G$ 代替 $(u, v, t_s, t_e) \in E(G)$. 此外,用 $\phi(G, v)$ 表示顶点 v 在时态图 G 中的邻居顶点集合, $\phi(G, u, v)$ 则代表时态图 G 中任意两个顶点 u 和 v 之间的边的集合. 相应地, $|\phi(G, v)|$ 表示顶点 v 在时态图 G 中的邻居顶点的数量, $|\phi(G, u, v)|$ 则代表时态图 G 中任意两个顶点 u 和 v 之间的边的数量. 此外,本文还令 $|\phi(G)|$ 代表时态图 G 中顶点的最大度数, $|\psi(G)|$ 表示时态图 G 中两个邻接顶点之间的最大多重边个数.

为了更好地描述两张时态图之间的联系,还

表 1 文中使用的数学符号表

Table 1 Mathematical symbols used in this article

符号	描述
$E(G)$	时态图 G 的边的集合
$F(k, h)$	$\hat{T}(S_i, G_c, k, h)$ 的候选插入子图
G	一张时态图
G_p	上一时刻的时态图
G_c	当前时刻的时态图
$P(k, h)$	部分 (k, h) -核
S_i	插入子图
S_r	删除子图
$S_i(G_c)$	S_i 在 G_c 上的扩展图
$T(G, k, h)$	时态图 G 的 (k, h) -核
$\hat{T}(S_i, G_c, k, h)$	S_i 在 G_c 上的类似 (k, h) -核
$V(G)$	时态图 G 的顶点的集合
u	时态图中任意一个顶点
v	时态图中任意一个顶点
(u, v, t_s, t_e)	时态图中任意一条边
$\phi(G, v)$	图 G 中任意一个顶点 v 的邻居顶点集合
$\phi(G, u, v)$	图 G 中任意两个顶点 u 和 v 之间边的集合
$ \phi(G, v) $	图 G 中任意一个顶点 v 的邻居顶点的数目
$ \phi(G, u, v) $	图 G 中任意两个顶点 u 和 v 之间边的数目
$ \phi(G) $	图 G 中顶点的最大度数
$ \psi(G) $	图 G 中两个邻接顶点之间最大多重边个数

定义了一些在时态图上的集合操作. 对于给定的两张时态图 G_1 和 G_2 , 如果 G_1 是 G_2 的子图, 那么 $G_1 \subseteq G_2$. $G_1 \cup G_2$ 是 G_1 和 G_2 的并图, $G_1 \cap G_2$ 表示 G_1 和 G_2 的交图, $G_1 \setminus G_2$ 代表 G_1 和 G_2 之间的差异图, 即 $E(G_1 \setminus G_2) = E(G_1) \setminus E(G_2)$.

2.2 (k, h) -核 与 k -核不同的是, (k, h) -核同时考虑了顶点的邻居数目以及顶点和邻居之间的带有时间戳的边的数目, 相应地, 它的结构也更复杂. 一般而言, 对于 (k, h) -核的研究, 要求 $k \geq 1$ 且 $h \geq 1$, 否则得到的 (k, h) -核没有意义. 通常用 K_0 表示一个无意义的 (k, h) -核或一张空图.

定义 1 (k, h) -核 给定一张时态图 G , 它的 (k, h) -核 $T(G, (k, h))$ 是时态图 G 满足以下条件

的最大子图,即对于任意顶点 $u \in T(G, (k, h))$, 有:

$$|\phi(T(G, (k, h)), u)| \geq k$$

对于任意一条边, 有:

$$(u, v, t_s, t_e) \in T(G, (k, h))$$

$$|\psi(T(G, (k, h)), (u, v))| \geq h$$

(k, h) -核之间还具有如图 1a 所示的部分嵌套关系^[16]. 箭头指向的时态核是箭头下方时态核的子图, 反之, 箭头下方的时态核是箭头指向的时态核的父图. 例如, 在 $(2, 2)$ -核中可以识别出 $(2, 3)$ -核和 $(3, 2)$ -核, 而 $(3, 1)$ -核和 $(1, 2)$ -核中也一定包含 $(2, 2)$ -核. 图 1b 中的一行称为一层, 即 k, h 的和相等的所有 (k, h) -核都处在同一层. 实际上, 时态图 (k, h) -核的维护算法中简化了这一关联关系, 通常利用如图 1b 所示的树状核值关系图.

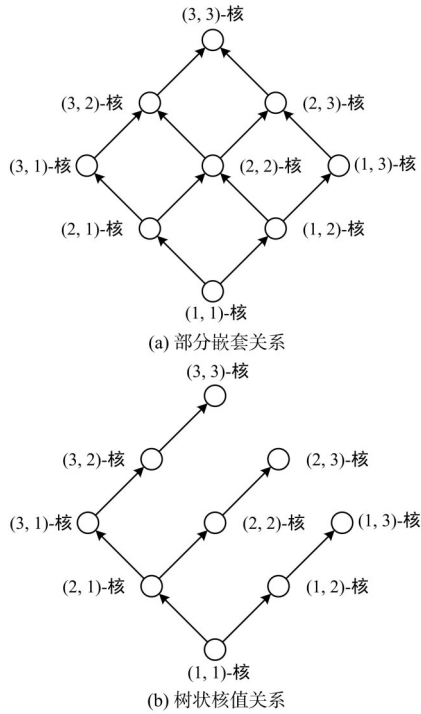


图 1 (k, h) -核之间的关联关系

Fig.1 Correlation between (k, h) -cores

(k, h) -核具有唯一性^[16], 因此 (k, h) -核分解和维护应该得到相同的结果. 在进行 (k, h) -核维护前, 首先要识别需要进行维护的 (k, h) -核. Bai et al^[16] 引入类似 (k, h) -核的概念, 类似 (k, h) -核与 (k, h) -核相似, 但约束条件较后者更宽松. 此外,

与 (k, h) -核不同的是, 类似 (k, h) -核是定义在两张有一定联系的时态图 S 和 G 上的. 为了解释这一概念, 首先需要对扩展图作出定义.

定义 2 扩展图 给定两张时态图 $S = (V(S), E(S))$ 和 $G = (V(G), E(G))$. 如果

$$S(G) = (V(S(G)), E(S(G)))$$

满足

$$V(S(G)) = V(S) \cup \{u: u \in \phi(G, v) \wedge v \in S\}$$

且

$$E(S(G)) = E(S) \cup \{\phi(u, v): u \in \phi(G, v) \wedge v \in S\}$$

则 $S(G)$ 是 S 在 G 上的扩展图. 另外, 如果 $S \cap G = K_0$, 那么 $S(G) = S$.

扩展图具有以下特征: 如果 $S \subseteq G$, 那么对于时态图 S 中任意一条边 $(u, v, t_s, t_e) \in S$, $\phi(S(G), u) = \phi(G, u)$, $\phi(S(G), v) = \phi(G, v)$, $\phi(S(G), u, v) = \phi(G, u, v)$ 都成立.

定义 3 类似 (k, h) -核 给定两张时态图 $S = (V(S), E(S))$ 和 $G = (V(G), E(G))$, 如果 $\hat{T}(S, G, (k, h))$ 是 S 的满足以下条件的最大子图: 令 $\hat{T}(G)$ 是 $\hat{T}(S, G, (k, h))$ 在 G 的扩展图, 对于任意一条边 $(u, v, t_s, t_e) \in \hat{T}(S, G, (k, h))$, $|\phi(\hat{T}(G), u)| \geq k$, $|\phi(\hat{T}(G), v)| \geq k$, 且 $|\phi(\hat{T}(G), (u, v))| \geq h$, 则 $\hat{T}(S, G, (k, h))$ 是 S 在 G 上的类似 (k, h) -核.

值得注意的是, 类似 (k, h) -核和 (k, h) -核一样具有唯一性, 并且不同的 k, h 组合构成的类似 (k, h) -核之间也同样具有部分嵌套关系. 因此, 只需要在 (k, h) -核分解算法的基础上稍加修正, 即可将其应用于解决时态图 S 在时态图 G 上的所有类似 (k, h) -核的分解问题.

定义 4 部分 (k, h) -核 部分 (k, h) -核 $P(k, h)$ 是当前时刻的 (k, h) -核和上一时刻的 (k, h) -核的差异图, 即:

$$P(k, h) = T(G_c, k, h) \setminus T(G_p, k, h)$$

当前时刻有新的边插入时态图时, 为了估计新插入的边对时态图中顶点核值的影响范围, 还需要引入候选插入子图的概念. 候选插入子图的

定义及其解释详见下文.

此外,给定一张时态图 G ,对于 G 在两个连续时刻的快照 G_p 和 G_c ,那些在上一时刻的快照 G_p 中不存在却在当前时刻的快照 G_c 中出现的边及其对应顶点所构成的子图称为插入子图 S_i , $S_i = G_c \setminus G_p$;相应地,那些在上一时刻的快照 G_p 中存在却在当前时刻的快照 G_c 中消失的边及其对应顶点构成的子图称为删除子图 S_r , $S_r = G_p \setminus G_c$. 显然,插入子图 S_i 与上一时刻的快照 G_p 可能含有相同的顶点,但一定没有相同的边.

3 自顶向下的时态图 (k, h) -核维护加边算法

本文提出一种由紧密到稀疏、自顶向下的时态图 (k, h) -核维护算法,可以对所有的 (k, h) -核进行维护,也可以只维护最紧密的前 n 层 (k, h) -核. 与自底向上的时态图 (k, h) -核维护算法不同,自顶向下的时态图 (k, h) -核维护算法主要面临两个挑战:(1)随着新的边加入时态图,时态图的结构可能会变得更紧密,如何估算最紧密的一层 (k, h) -核对应的 k, h 值;(2)如何自顶向下地利用 $(k+1, h)$ -核和 $(k, h+1)$ -核,尽可能减小维护 (k, h) -核所需的时间. 需要注意的是, (k, h) -核是 $(k+1, h)$ -核和 $(k, h+1)$ -核的父图,因此还需利用时态图中 $(k+1, h)$ -核和 $(k, h+1)$ -核以外的一些顶点的信息,这会使情况变得更复杂.

对于 k 的取值,由于当 $h=1$ 时, $(k, 1)$ -核等同于 k -核,一种寻找最大 k 值的方法是根据“当在图 G 中插入或删除一条边 (u, v) 时,图 G 中所有顶点的核值的改变最多为 1”这一定理^[22],推断在图 G 中插入 m 条非多重边时,图 G 中所有顶点的 k -核的核值变化最多为 m . 只需统计插入子图 S_i 去除多重边后顶点间边数的最大值 m ,将上一时刻最大的 k 值与 m 相加,即可得到当前时刻 (k, h) -核中可能存在的最大的 k . 这种方法适用于自顶向下维护时态图中所有的 (k, h) -核,或时态图中最大的 k 大于等于最大的 h 的情况.

当算法只维护最紧密的 n 层 (k, h) -核或时态图中最大的 k 小于最大的 h 时,由于时态图中 k 最

大的 (k, h) -核可能不在前 n 层 (k, h) -核中,无法根据上一时刻前 n 层 (k, h) -核的信息获得上一时刻最大的 k . 此时算法可以统计当前时刻时态图中顶点最大的度数 K ,根据 (k, h) -核的定义, K 即为当前时刻 (k, h) -核中最大的 k 的上界. 同理,当 $k=1$ 时, $(1, h)$ -核即由时态图中任意两个顶点及其之间的边组成,因此 h 的最大值 H 可以确定为当前时刻时态图中任意两个顶点之间边的数目的最大值. 需要注意的是, (K, H) -核有可能不存在.

对一个受到插入边影响而需要更新的 (k, h) -核进行维护的过程主要分三步:(1)估算所有可能需要更新核值的顶点,即将类似 (k, h) -核扩展为候选插入子图,候选插入子图包含所有可能被类似 (k, h) -核中边影响核值的顶点及这些顶点之间的边;(2)识别核值真正发生改变的顶点,即利用候选插入子图中的信息识别出部分 (k, h) -核;(3)将部分 (k, h) -核与上一时刻的 (k, h) -核合并,即可得到当前时刻的 (k, h) -核,完成 (k, h) -核维护.

定义 5 候选插入子图 令:

$$T = T(G_p, k, h) \cup T(G_c, k+1, h) \cup T(G_c, k, h+1)$$

如果图 $F(k, h)$ 中任意一个顶点或一条边都可以经由满足以下条件的路径访问 $hat T(S_i, G_c, k, h)$ 中的任意顶点或边: $\forall (u, v, t_s, t_e) \notin T$, $|\phi(G_c, u)| \geq k, |\phi(G_c, v)| \geq k$ 且 $|\phi(G_c, (u, v))| \geq h$, 则 $F(k, h)$ 是 $T(G_c, k, h)$ 的候选插入子图.

自顶向下的时态图 (k, h) -核维护算法从联系最紧密的那一层 (k, h) -核开始自顶向下逐层更新,因此在维护当前时刻的 (k, h) -核时,它在当前时刻时态图快照 G_c 上的父图(如 $(k-1, h)$ -核或 $(k, h-1)$ -核)是未知的,无法利用父图缩小关注范围,需要在 G_c 上识别候选插入子图以估算核值可能受到影响的顶点范围. 此外,由于此时已经完成了对当前时刻 $(k+1, h)$ -核与 $(k, h+1)$ -核的更新,而它们又都是 (k, h) -核的子图,因此本文提出一种在当前时刻 $(k+1, h)$ -核与 $(k, h+1)$ -核的基础上识别候选插入子图的方法,以减少计算时间.

根据 (k, h) -核之间的部分嵌套关系,可以发现,如果一条边存在于上一时刻的 (k, h) -核中,或它在当前时刻的 $(k+1, h)$ -核或 $(k, h+1)$ -核中已经出现,那么它一定存在于当前时刻的 (k, h) -核中. 根据定义4,此时这条边不需要加入候选插入子图.

根据定义5可以发现, $F(k, h)$ 中一定含有除 $T(G_p, k, h) \cup T(G_p, k+1, h) \cup T(G_p, k, h+1)$ 外所有由于类似 (k, h) -核中边的插入导致核值可能受到影响的顶点和对应的边. 需要注意的是, $F(k, h)$ 中还可能包含一些冗余的顶点,它们的核值可能并没有真正发生改变,算法下一步会去除这些冗余顶点和相应的边.

算法1介绍了识别候选插入子图的具体方法. 由于该算法只涉及图的遍历操作,因此时间复杂度不会超过 $O(|G_c|)$,其中 $|G_c|$ 是图 G_c 的顶点数目. 另外,由于算法是自顶向下识别最紧密的 n 层类似 (k, h) -核在当前时刻的时态图快照 G_c 上的候选插入子图,而越紧密的 (k, h) -核中顶点的数目越少. 对于一张大规模时态图,当需要维护的层数 n 远小于时态图中存在的所有 (k, h) -核所构成的总层数时,候选插入子图需要遍历的顶点数目通常远小于时态图快照 G_c 中顶点的总数,因此算法的时间复杂度在一般情况下远小于 $O(|G_c|)$.

算法1 候选插入子图识别算法

输入:当前时刻的时态图快照 G_c ;插入子图 S_i 在 G_c 上的类似 (k, h) -核 $\hat{T}(S_i, G_c, k, h)$;由 $T(G_p, k, h)$, $T(G_c, k+1, h)$ 和 $T(G_c, k, h+1)$ 的并集构成的并图 T

输出:候选插入子图 $F(k, h)$

1. 定义一个空的顶点集合 \mathcal{Q}_v ;
2. 将所有 $v \in \hat{T}(S_i, G_c, k, h)$ 标为已访问,并放入 \mathcal{Q}_v ;
3. while \mathcal{Q}_v 不为空 do
4. 从 \mathcal{Q}_v 中取出一个顶点 u ,并将其从 \mathcal{Q}_v 中移除;
5. for all 顶点 $v \in \phi(G_c, u)$ do
6. if $|\phi(G_c, v)| \geq k \wedge |\phi(G_c, (u, v))| \geq h$ then
7. if $\phi(G_c, u, v) \notin T$ then
8. 将 G_c 中顶点 v 及边集 $\phi(G_c, (u, v))$ 插入 $\hat{T}(S_i, G_c, k, h)$

9. if $v \notin T \wedge v$ 未访问 then

10. 将顶点 v 加入集合 \mathcal{Q}_v ;

11. 输出此时的 $\hat{T}(S_i, G_c, k, h)$ 即为候选插入子图 $F(k, h)$.

在获得候选插入子图后,还需考虑如何利用候选插入子图识别部分 (k, h) -核的问题. 这一步骤较复杂,根据自顶向下维护 (k, h) -核时已经得到的上一时刻的 (k, h) -核、当前时刻的 $(k+1, h)$ -核与当前时刻的 $(k, h+1)$ -核,将识别部分 (k, h) -核的问题分四种情况讨论:

定理1 如果:

$$T(G_p, k, h) \neq K_0$$

$$T(G_c, k+1, h) \cup T(G_c, k, h+1) \neq K_0$$

令:

$$T = T(G_p, k, h) \cup T(G_c, k+1, h) \cup T(G_c, k, h+1)$$

那么:

$$P(k, h) = (\hat{T}(F(k, h), T, k, h) \cup T(G_c, k+1, h) \cup T(G_c, k, h+1)) \setminus T(G_p, k, h)$$

定理1的证明略.

引理1 如果 $T(G_p, k, h) \neq K_0$,那么 $P(k, h) = \hat{T}(P(k, h), T(G_p, k, h), k, h)$ 成立.

证明 令:

$$\hat{T} = \hat{T}(P(k, h), T(G_p, k, h), k, h), P = P(k, h)$$

根据定义3,可知 $\hat{T} \subseteq P$,因此只需证明 $P \subseteq \hat{T}$ 成立. 假设存在一张非空时态图 $A = P \setminus \hat{T}$,那么一定存在一个顶点 $u \in V(P)$ 或一条边 $(u, v, t_s, t_e) \in E(P)$ 满足在 $T(G_p, k, h)$ 上的类似 (k, h) -核的约束条件却不在 \hat{T} 中,这与定义3矛盾. 因此 A 是一张空图,假设错误, $P \subseteq \hat{T}$ 成立.

综上, $P(k, h) = \hat{T}(P(k, h), T(G_p, k, h), k, h)$

得证.

定理2 如果:

$$T(G_p, k, h) \neq K_0$$

$$T(G_c, k+1, h) \cup T(G_c, k, h+1) = K_0$$

令:

$$T = T(G_p, k, h)$$

那么:

$$P(k, h) = \hat{T}(F(k, h), T, k, h)$$

证明 令:

$$\hat{T} = \hat{T}(F(k, h), T, k, h), P = P(k, h), F = F(k, h)$$

首先证明 $P \subseteq \hat{T}$. 根据定义 5 和定义 4 可知 $P \subseteq F$, 因此 $\hat{T}(P, T, k, h) \subseteq \hat{T}(F, T, k, h)$. 根据引理 1 可知, 当 $T = T(G_p, k, h)$ 时 $P = \hat{T}(P, T, k, h)$. 因此, $P \subseteq \hat{T}$ 得证.

接下来证明 $\hat{T} \subseteq P$.

假设(a): 存在一张非空时态图 $A = \hat{T} \setminus P$, 那么对于任意两个顶点 $u, v \in A$, $|\phi(\hat{T}(T), v)| \geq k$, $|\phi(\hat{T}(T), u)| \geq k$ 及 $|\phi(\hat{T}(T), u, v)| \geq h$ 都成立.

假设(b): $v \notin P$. 根据定义 5 和定义 4, 一定存在一个顶点 $x \in T$, v 和 x 在 T 上互相可达, 但 $|\phi(T, x)| < k$, 这与定义 1 矛盾, 所以不存在这样的顶点 $x \in T$, 假设(b)错误. 因此, 对于任意一个顶点 $v \in A$, $v \in P$ 一定成立.

对于任意一条边 $(u, v, t_s, t_e) \in A$, 假设(c): $(u, v, t_s, t_e) \notin P$. 由于 $u, v \in A$, 那么 $u, v \in P$ 成立, 由于 $|\phi(\hat{T}(T), u, v)| \geq h$, 则 $P \subseteq P \cup (u, v, t_s, t_e)$, 与定义 4 矛盾, 所以假设(c)错误. 因此 A 是一张空图, 假设(a)也错误.

综上, 得出 $\hat{T} \subseteq P$.

综上, $P(k, h) = \hat{T}(F(k, h), T, k, h)$ 得证.

定理 3 如果:

$$T(G_p, k, h) = K_0$$

$$T(G_c, k+1, h) \cup T(G_c, k, h+1) \neq K_0$$

令:

$$T = T(G_c, k+1, h) \cup T(G_c, k, h+1)$$

那么:

$$P(k, h) = \hat{T}(F(k, h), T, k, h) \cup T$$

定理 3 的证明略.

定理 4 如果:

$$T(G_p, k, h) \cup T(G_c, k+1, h) \cup T(G_c, k, h+1) = K_0$$

那么:

$$P(k, h) = T(G_p, k, h) = T(F(k, h), T, k, h)$$

证明 令 $P = P(k, h)$, $F = F(k, h)$. 由于

$T(G_p, k, h) = K_0$, 根据定义 4, $P = T(G_c, k, h)$ 成立, 只需证明 $T(G_c, k, h) = T(F, k, h)$.

首先根据定义 5, 当:

$$T(G_p, k, h) \cup (G_c, k+1, h) \cup T(G_c, k, h+1) = K_0$$

时 $T(G_c, k, h) \subseteq F$ 成立, 所以 $T(G_c, k, h) \subseteq T(F, k, h)$ 成立. 另外, 由于 $F \subseteq G_c$, 根据定义 1, $T(F, k, h) \subseteq T(G_c, k, h)$ 成立.

综上, $T(G_c, k, h) = T(F, k, h)$ 成立.

综上, $P(k, h) = T(G_c, k, h) = T(F(k, h), k, h)$ 得证.

基于以上四个定理, 本文提出一种部分 (k, h) -核识别算法, 迭代地找出候选插入子图中不符合部分 (k, h) -核定义, 即核值没有真正发生改变的那些顶点, 并将这些顶点和它们的对应边从 $F(k, h)$ 中删除, 最终得到 $P(k, h)$. 如算法 2 所示.

算法 2 部分 (k, h) -核识别算法

输入: 候选插入子图 $F(k, h)$; 上一时刻的 (k, h) -核 $T(G_p, k, h)$; 由 $T(G_p, k, h)$, $T(G_c, k+1, h)$ 和 $T(G_c, k, h+1)$ 的并集构成的并图 T

输出: 部分 (k, h) -核 $P(k, h)$

1. if $T \neq K_0$ then
2. 初始化顶点队列 Q_v 为空;
3. for all $v \in F(k, h) \wedge v \notin T$ do
4. if $|\phi(F(k, h), v)| < k$ then
5. 将 v 放入 Q_v , 并将 v 标为已访问;
6. while Q_v 不为空 do
7. 从 Q_v 中取出 v , 并将其从 Q_v 中移除;
8. for all $u \in \phi(F(k, h), v) \wedge u \notin T$ do
9. if $|\phi(F(k, h), v) - 1| < k \wedge u$ 未访问 then
10. 将 u 加入队列 Q_v ;
11. 将 v 及对应边从 $F(k, h)$ 中删除;
12. 此时的 $F(k, h)$ 即为 $P(k, h)$;
13. else
14. 利用 $F(k, h)$, 根据定理 4, 对 $F(k, h)$ 做 (k, h) -核分解得到 $P(k, h)$;
15. 输出部分 (k, h) -核 $P(k, h)$.

最后, 提出一种自顶向下的时态图 (k, h) -核加边维护算法, 如算法 3 所示.

算法3 自顶向下的时态图 (k, h) -核维护加边算法

输入:当前时刻的时态图快照 G_c ;插入子图 S_i ;上一时刻前 n 层 (k, h) -核的集合 \mathcal{T}_c ;需要维护的前 n 层 (k, h) -核的 n 值

输出:当前时刻最紧密的 n 层 (k, h) -核的集合 \mathcal{T}_c

1. 计算所有类似 (k, h) -核 $\hat{T}(S_i, G_c, k, h)$;
2. 初始化不重复队列 \mathcal{Q} 、集合 \mathcal{T}_c 均为空, $L = 0$; // 标识当前 (k, h) -核的最大层数
3. 计算 G_c 中顶点的最大度数 $\phi(G_c)$, 任意两个顶点间多重边数目的最大值 $\psi(G_c)$;
4. 将 $(\phi(G_c), \psi(G_c))$ 放入 \mathcal{Q} ;
5. while \mathcal{Q} 不为空 do
6. 从 \mathcal{Q} 中取出 (k, h) , 并将其从 \mathcal{Q} 中移除;
7. if $T_p(k, h) \neq K_0 \wedge L == 0$ then
8. $L = k + h - 1$; // 确定当前时刻最大层数
9. if $\hat{T}(S_i, G_c, k, h) \neq K_0$ then
10. 令
11. $T = T(G_p, k, h) \cup T(G_c, k + 1, h) \cup T(G_c, k, h + 1)$;
12. 根据算法1计算 $F(k, h)$;
13. 根据算法2计算 $P(k, h)$, 将 $P(k, h)$ 与 T 合并得到 $T(G_c, k, h)$;
14. if $T(G_c, k, h) \neq K_0$ then
15. 将 $T(G_c, k, h)$ 放入 \mathcal{T}_c ;
16. 若 $L == 0$, 令 $L = k + h - 1$;
17. if $L - (k + h - 1) + 1 < n$ then
18. 将 $(k, h - 1)$ 和 $(k - 1, h)$ 放入 \mathcal{Q} ;
19. 输出当前时刻最紧密的 n 层 (k, h) -核的集合 \mathcal{T}_c .

算法3第1行利用QTCDNG算法^[16]计算 S_i 在当前时刻时态图快照 G_c 上的所有类似 (k, h) -核, 第9~17行根据算法1计算类似 (k, h) -核在 G_c 的扩展图, 再根据算法2计算部分 (k, h) -核并最终获得当前时刻的 (k, h) -核. 此外, 算法3第2~8行及13~17行展示了寻找当前时刻时态核的最大层数以及只维护最紧密的前 n 层时态核的方法.

自顶向下的时态图 (k, h) -核维护加边算法在最坏情况下的时间复杂度由几部分组成. 令 $T = T(G_p, k, h) \cup T(G_c, k + 1, h) \cup T(G_c, k, h + 1)$. 首先, QTCDNG算法^[16]的时间复杂度不超过 $O\left(\sum_{\hat{T}(S_i, G_c, k, h) \in \hat{\mathcal{T}}} 3|\hat{T}(S_i, G_c, k, h)|\right)^{[16]}$. 算法1计算

扩展图 $F(k, h)$ 的时间复杂度不超过 $O(|F(k, h)|)$, 算法2计算部分 (k, h) -核的时间复杂度也不超过 $O(|T|)$. 两次图合并操作的时间复杂度分别为 $O(|T|)$ 及 $O(|\hat{T}(F(k, h), T, k, h)|)$. 因此自顶向下的时态图 (k, h) -核维护加边算法在最坏情况下的时间复杂度不超过 $O(N \times |G_c|)$, 其中 N 表示当前时刻前 n 层中 (k, h) -核的数目.

图2展示了自顶向下的时态图 (k, h) -核加边算法的具体过程. 为了简洁直观, 隐藏了时态图边上带有的时间信息, 并以不同颜色标注算法中使用到的不同子图.

4 自顶向下的时态图 (k, h) -核维护删边算法

当有上一时刻时态图中存在的边在当前时刻被删除时, 删除子图 S_r 只会对上一时刻已经存在的 (k, h) -核的结构产生影响, 而不会像有新的边插入时一样产生新的 (k, h) -核, 省去了寻找扩展图和部分 (k, h) -核的步骤, 因此删边算法更简单. 同样, 对于如何寻找新的最紧密的一层 (k, h) -核中 (k, h) 取值的问题, 也可以直接利用上一时刻最紧密的时态核层数, 自顶向下、自左向右一次性寻找当前时刻最紧密的 n 层时态核.

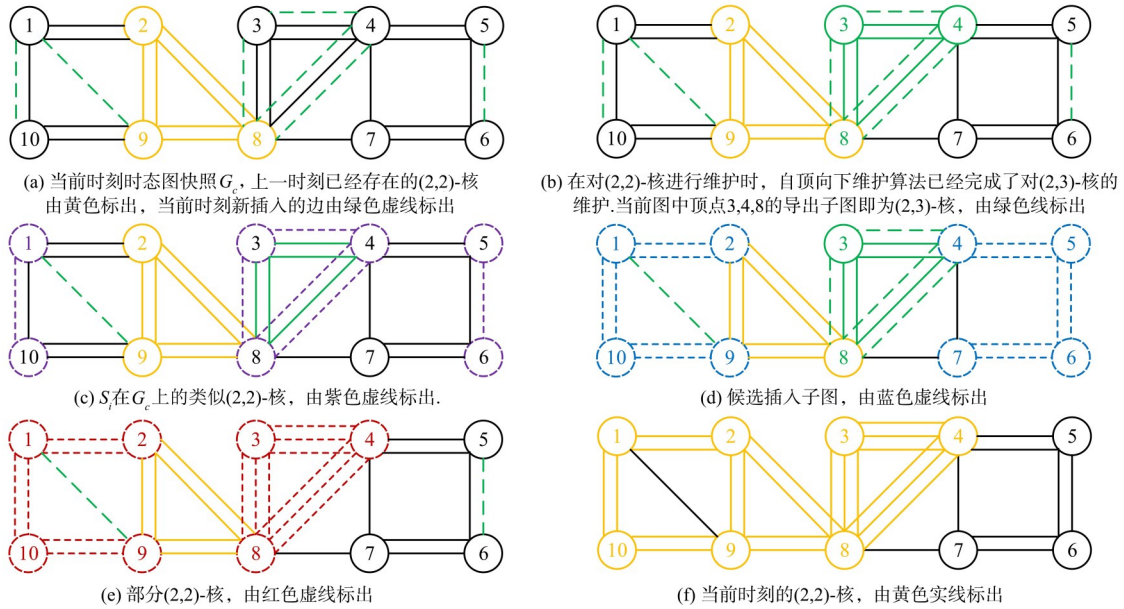
算法4展示了对于单独一个 (k, h) -核进行维护的删边算法. 该算法利用类似 (k, h) -核, 首先将类似 (k, h) -核中的边从上一时刻的 (k, h) -核中移除, 然后迭代地从上一时刻的 (k, h) -核中移除那些受删除边影响且在当前时刻不再满足 (k, h) -核定义的顶点及对应的边, 完成对当前时刻 (k, h) -核的维护.

算法4 单一 (k, h) -核维护删边算法

输入:类似 (k, h) -核 $\hat{T}(S_r, G_c, k, h)$; 上一时刻的 (k, h) -核 $T(G_p, k, h)$

输出:当前时刻的 (k, h) -核

1. 定义一个空的用于存储顶点的队列 \mathcal{Q}_v ;
2. for all $(u, v, t_s, t_e) \in T(G_p, k, h) \cap \hat{T}(S_r, G_p, k, h)$ do
3. 将 (u, v, t_s, t_e) 从 $T(G_p, k, h)$ 中删除;

图 2 当插入 S_i 时自顶向下维护时态图 $(2,2)$ -核的过程Fig.2 The process of top-down maintenance of the temporal graph $(2,2)$ -core when inserting S_i

4. if $|\phi(T(G_p, k, h), u, v)| < h$ then
5. 将 $T(G_p, k, h)$ 中 u, v 之间的边全部删除;
6. if $|\phi(T(G_p, k, h), u)| < k$ then
7. 将顶点 u 加入队列 Q_v ;
8. if $|\phi(T(G_p, k, h), v)| < k$ then
9. 将顶点 v 加入队列 Q_v ;
10. while Q_v 不为空
11. 从 Q_v 中取出顶点 u , 并将其从 Q_v 中删除;
12. for all $v \in \phi(T(G_p, k, h), u)$ do
13. 删除 $T(G_p, k, h)$ 中 (u, v) 间的所有边;
14. if $|\phi(T(G_p, k, h), v)| < k$ then
15. 将顶点 v 加入队列 Q_v ;
16. 将顶点 u 从 $T(G_p, k, h)$ 中删除;
17. 输出当前的 $T(G_p, k, h)$ 作为 $T(G_c, k, h)$.

在算法 4 的基础上, 自顶向下的时态图 (k, h) -核维护删边算法可以自顶向下维护全部层或前 n 层 (k, h) -核, 如算法 5 所示. 与加边算法类似, 删边算法第 1 行同样是首先利用 QTCDNG 算法^[16]计算 S_i 在当前时刻时态图快照 G_c 上的所有类似 (k, h) -核, 算法第 3~4 行展示了寻找当前时刻 (k, h) -核的最大层数的方法.

算法 5 自顶向下的时态图 (k, h) -核维护删边算法

输入: 当前时刻的时态图快照 G_c ; 删除子图 S_i ; 上一时刻最紧密的 n 层 (k, h) -核的集合 \mathcal{T}_p ; 需要维护的最紧密的 n 层 (k, h) -核的 n 值

输出: 当前时刻最紧密的 n 层 (k, h) -核的集合 \mathcal{T}_c

1. 计算所有的类似 (k, h) -核 $\hat{T}(S_i, G_c, k, h)$;
2. 定义一个空的用于存储 (k, h) 二元组的不重复队列 Q ;
3. 将所处层数与上一时刻时态核层数的最大值相同的所有 (k, h) 放入 Q ;
4. 初始化 $L = 0$; // 标识当前 (k, h) -核的最大层数
5. while Q 不为空
6. 从 Q 中取出 (k, h) , 并将其从 Q 中移除;
7. if \mathcal{T}_p 均已被访问过 then
8. 用 (k, h) -核分解算法从 G_c 中分解出 $T(G_c, k, h)$;
9. if $T(G_p, k, h) \neq K_0$ then
10. 利用算法 4 计算 $T(G_c, k, h)$;
11. if $T(G_c, k, h) \neq K_0$ then
12. 将 $T(G_c, k, h)$ 放入 \mathcal{T}_c ;
13. 若 $L = 0$, 令 $L = k + h - 1$; // 确定当前时刻时态核最大层数
14. if $L - (k + h - 1) + 1 < n$ then
15. 将 $(k, h - 1)$ 和 $(k - 1, h)$ 放入队列 Q ;
16. 输出当前时刻最紧密的 n 层 (k, h) -核的集合 \mathcal{T}_c .

需要注意,当只有自顶向下维护最紧密的前 n 层时态核时,令上一时刻最大层数为 N ,此时已知上一时刻第 N 至 $N-n+1$ 层时态核.假设上一时刻第 N 层时态核受删除边影响而消失,此时就要维护当前时刻第 $N-1$ 至 $N-n$ 层时态核.而上一时刻第 $N-n$ 层时态核的信息是未知的,所以需要直接从 G_c 中分解出第 $N-n$ 层 (k, h) -核,算法5第7~8行展示了这一过程.时态图越复杂,第 N 层中时态核的个数越多,因此在大规模时态图中只有小部分边被删除时,第 N 层中全部时态核都变为空图的可能性较小.

接下来,算法5第10~12行利用算法4完成删边时对每一个 (k, h) -核的维护.最后,算法5第13~16行通过对被放入维护队列的 (k, h) 进行约束,实现只维护最紧密的前 n 层 (k, h) -核.

自顶向下的时态图 (k, h) -核删边算法在最坏情况下的时间复杂度由两部分组成.令 $T=$

$$T(G_p, k, h) \cup T(G_c, k+1, h) \cup T(G_c, k, h+1).$$

首先,利用QTCDNG算法^[16]识别出 S_i 在当前时刻时态图快照 G_c 上的所有类似 (k, h) -核,在最坏情况下的时间复杂度不超过 $\sum_{\hat{T}(S_r, G_c, k, h) \in \mathcal{T}} O(3|\hat{T}(S_r, G_c, k, h)|)^{[16]}$.而算法4在最坏情况下会访问整张上一时刻的 (k, h) -核,因此其时间复杂度为 $O(|\mathcal{T}_p|)$,其中 \mathcal{T}_p 为上一时刻最紧密的前 n 层 (k, h) -核的集合.因此自顶向下的时态图 (k, h) -核维护删边算法时间复杂度不会超过 $O(3N \times |S_r| + |\mathcal{T}_p|)$,其中 N 表示当前时刻前 n 层中 (k, h) -核的数目.

图3展示了自顶向下的时态图 (k, h) -核维护删边算法的具体过程.为了简洁直观,示例图隐藏了时态图边上带有的时间信息,同时以不同颜色标注算法中使用到的不同子图.

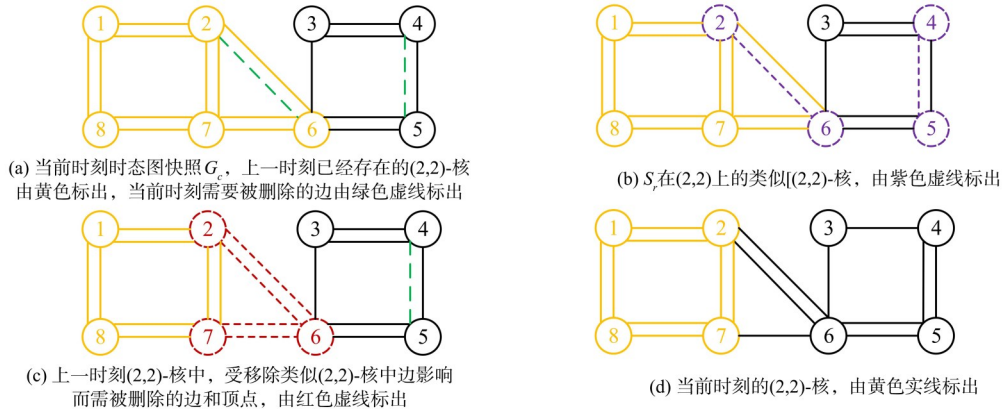


图3 当删除 S_r 时自顶向下维护时态图 $(2,2)$ -核的过程

Fig.3 The process of top-down maintenance of temporal graph $(2,2)$ -core when removing S_r .

5 实验结果与分析

实验平台: CPU Xeon E5-2683v4 @ 2.1 GHz \times 2, 内存 128 GB, 操作系统 Linux.

使用的数据集均来自 Network Repository 和 SNAP, 选用 10 个真实存在的时态图数据集, 分别为 ia-contact (IC), ia-contacts-dublin (ICD), ia-digg-reply (IDR), ia-enron-employees (IEE), ia-escorts-dynamic (IED), ia-enron-email-all (IEEA), sx-askubuntu (SA), ia-stackexch-user-marks-post-und (ISUMPU), rec-amazon-ratings

(RAR), rec-stackoverflow (RS). 表2详细介绍了这些图数据集的关键数据, 其中 $|V|$ 和 $|E|$ 分别表示图中所有点和边的数目. 此外, $|\phi(G)|$ 表示时态图 G 中顶点的最大度数, $|\psi(G)|$ 则是时态图 G 中任意两个顶点间多重边数目的最大值. 为了后续更好地解释 (k, h) -核维护算法的性能, 表2还给出了每张图中其所有存在的 (k, h) -核中 k 的最大值 K 和 h 的最大值 H . 根据 (k, h) -核的定义可以发现, $(K, 1)$ -核与 $(1, H)$ -核一定存在, 但需要注意的是, (K, H) -核不一定存在.

接下来将这些图按照图的规模分为两组:规模较小的一组(IC, ICD, IDR, IEE, IED)用于测试算法的性能;规模较大的一组(IEEA, ISUM-PU, RAR, SA, RS)用于测试算法的可扩展性. 观察表 2 可以发现, $K(H)$ 的值和图的紧密程度没有明显的关系. 虽然一些图, 如 RAR 中顶点的最大度数 $|\phi(G)|$ 非常大, 但它的 K 相比而言很小. 此外, 顶点间多重边数目的最大值 $|\psi(G)|$ 始终等于 H . 这是由 (k, h) -核的定义决定的.

表 2 真实时态图

Table 2 Real temporal graphs

G	$ V $	$ E $	$ \phi(G) $	$ \psi(G) $	K	H
IC	274	28244	101	168	39	168
ICD	10972	415912	64	345	18	345
IDR	30,360	86203	283	25	9	25
IEE	150	24705	74	1117	14	1117
IED	10106	50412	311	39	11	39
IEEA	86978	1134990	1726	1353	53	1353
SA	157222	726639	5401	215	48	215
ISUMPU	545196	1302234	6121	4	24	4
RAR	2146057	5776660	12204	28	29	28
RS	545196	1301923	6121	2	24	2

为了更快地访问图中任意一个顶点或任意一条边, 使用哈希表的方式来存储时态图, 图 4 分别展示了每张图所需的内存用量. 此外, 还在图 5 中统计了每张图中实际存在的所有 (k, h) -核的数目. 这一数目与时态图自身的结构有关, 因此即使图 5b 中的图顶点和边的规模较大, (k, h) -核的数目也不一定更大.

为了体现时态图动态变化的过程, 对于有新的边插入的情况, 对实验所用时态图的所有边按照第一次出现的时间戳进行升序排序, 使用排在最后的 n 条边及其对应顶点来构造 S_t , 使用全部的边及顶点构成当前时刻的时态图快照 G . 相应地, 对于已经存在的边被删除时, 则将时态图的所有边按照其消失的时间戳进行升序排序, 使用排在最前面的 n 条边及其对应顶点来构造删除子图 S_r , 而当前时刻的时态图快照 G 则由除去前 n 条边外余下的边及其对应顶点构成.

5.1 对比算法与评价指标 对于时态图核值的

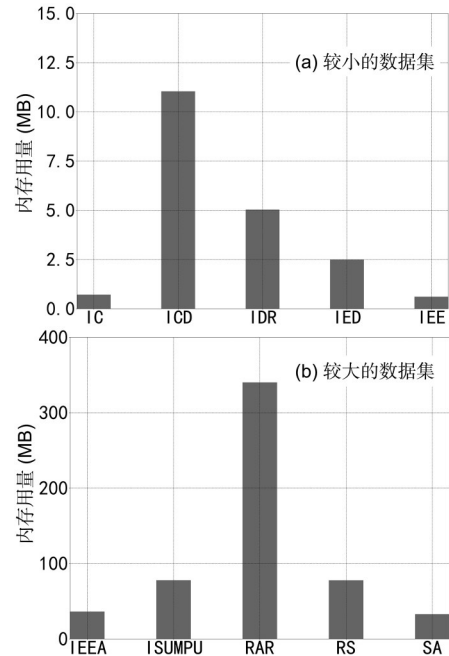
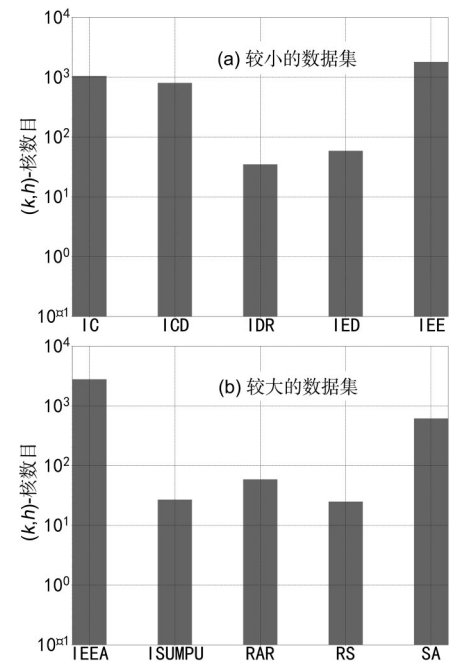


图 4 加载图所需的内存用量

Fig. 4 The amount of memory required to load the graph

图 5 图中 (k, h) -核的数目Fig. 5 The number of (k, h) -cores in the graph

分解和维护, 文献[19, 22—23]已证明获得当前时刻核值结果的执行时间是重要的评价指标. 使用直接的 (k, h) -核分解算法和自底向上的 (k, h) -

核维护算法^[16]作为对比算法,图6展示了直接分解算法的处理时间,而自底向上维护算法的处理时间与实验设置有关,下面的实验使用本文算法与这两种算法处理时间的加速比作为评价指标。

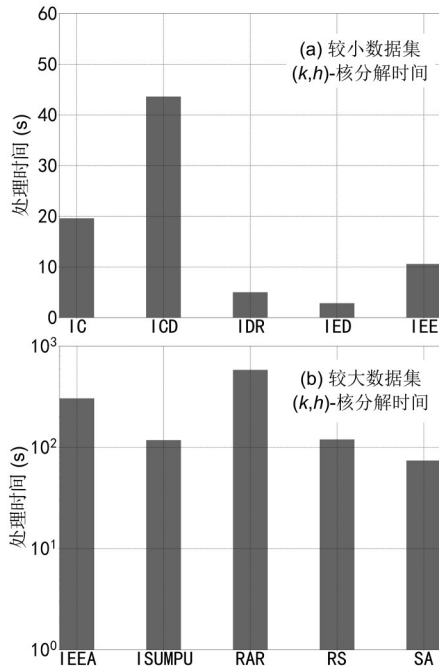


图6 直接的 (k, h) -核分解算法的处理时间

Fig.6 Processing time of direct (k, h) -core decomposition algorithm

5.2 维护层数可扩展性

5.2.1 自顶向下的时态图 (k, h) -核维护加边算法

首先固定时态图中发生变化的边数为100条,研究需要维护的层数 n 不同时,自顶向下加边维护算法的效率变化规律,实验结果如图7所示.由于直接进行 (k, h) -核分解或自底向上进行 (k, h) -核维护,都能得到图中存在的所有 (k, h) -核,与层数 n 无关,因此在实验中直接分解算法和自底向上维护算法所需时间固定不变.由图可见,在需要维护的层数增多时,自顶向下维护算法所需的时间逐渐增多.与这两种对比算法相比,所有被测数据集在自顶向下维护前40层时的加速比都大于1,自顶向下的维护算法效率更高。

$n > 30$ 时,自顶向下维护算法在IDR和IED上的效率与自底向上维护算法几乎相同,这是由于IDR和IED的 (k, h) -核中 k 和 h 的最大值都较

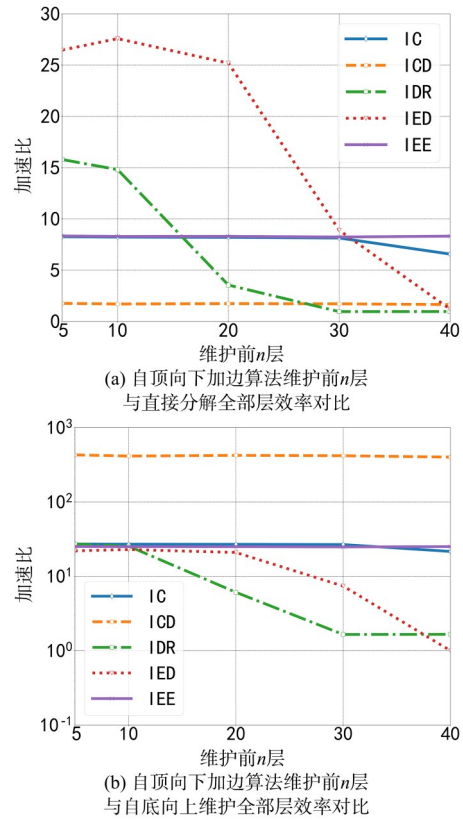


图7 加边情况下维护全部层的效率

Fig.7 The algorithm efficiency of maintaining full layers when inserting edges

小, IDR只有33层,而IED则有49层.由于本文提出的自顶向下算法在维护当前时刻 (k, h) -核时无法获得它的父图 $(k-1, h)$ -核与 $(k, h-1)$ -核的信息,只能在整张当前时刻的时态核快照内寻找候选插入子图.因此,当需要维护的层数占总层数比例较大时,寻找候选插入子图的过程可能会消耗较多时间.但在维护前20层时,自顶向下维护算法的效率可以达到直接分解算法的几十倍,与自底向上算法相比,加速比也可以达到5~30倍。

5.2.2 自顶向下的时态图 (k, h) -核维护删边算法

接下来,固定时态图中发生变化的边数为100,研究当需要维护的层数 n 不同时自顶向下删边维护算法的效率变化.如图8所示,由于被删除边对时态核结构的影响范围不会扩大到上一时刻已经存在的 (k, h) -核之外,因此自顶向下删边维护算法与自顶向上的删边算法相比优势不如加边算法大,但自顶向下删边维护算法的效率依然更高。

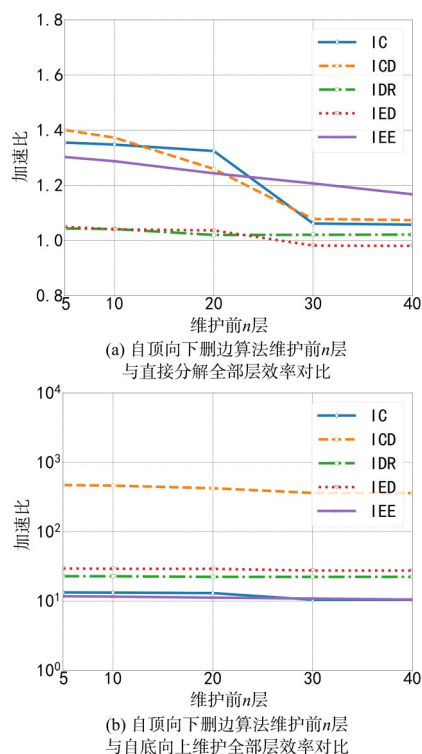


图8 删边情况下维护全部层的效率

Fig.8 The algorithm efficiency of maintaining full layers when removing edges

5.3 变化边数可扩展性 研究时态图中发生变化的边的数目对自顶向下的时态图 (k, h) -核维护算法的效率的影响时,使用图规模较大的五个数据集. 固定需要维护的 (k, h) -核层数为5,分别使用数据集中按时间排序后的后(前)100, 500, 1000, 5000, 10000条边构造插入子图(删除子图).

5.3.1 自顶向下的时态图 (k, h) -核维护加边算法

图9a展示了新插入边数不同时自顶向下维护前五层 (k, h) -核与直接分解算法的加速比. 随着插入边数的增多,插入边数占整张时态图中总边数的比例逐渐增大,受插入边影响的 (k, h) -核比例也随之增大,自顶向下维护算法的开销越来越大,加速比逐渐降低. 实验结果显示,新插入边的数目在100~10000,自顶向下算法始终具有更高的效率,且在变化边数较小时有明显优势.

图9b展示了插入边数不同时自顶向下维护前五层与自底向上维护全部层的效率对比. 实验显示,加速比随着插入边数的增多整体呈降低趋势,但加速比始终大于1,意味着当插入边数不超

过10000时,自顶向下维护前五层的效率始终比自底向上维护全部层高. 此外,对比图9a和图9b,可以发现对于图IEEA和图SA,和直接重新进行 (k, h) -核分解相比,自顶向下维护算法与自底向上维护算法之间的效率差别更大,自顶向下的时态图 (k, h) -核维护加边算法的优势更明显. 这是由于有较大规模的边同时被插入到时态图时,作为对比算法的自底向上的 (k, h) -核维护算法的效率明显降低,不适合处理新插入边占整体图比例较大的情况,而本文的自顶向下的时态图 (k, h) -核维护算法则没有这一缺点.

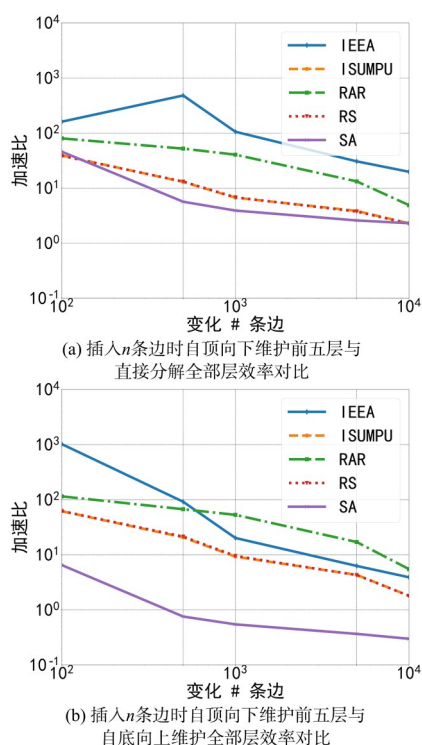


图9 加边情况下维护前五层的效率

Fig.9 The algorithm efficiency of maintaining the first five layers layers when inserting edges

5.3.2 自顶向下的时态图 (k, h) -核维护删边算法

接下来讨论删除边数目对自顶向下的时态图 (k, h) -核维护删边算法效率的影响. 图10a显示,当删除边增多时维护算法所需时间增多,与直接分解算法相比,加速比逐渐降低;但加速比始终很大,在删除边数量不断增加时,所有图的加速比仍然都大于1. 而且,自顶向下的时态图 (k, h) -核维护删边的算法效率依然更高.

此外,对比图 10a 和图 10b,当变化边数较少时,和加边时的情况相比,自顶向下的 (k, h) -核维护删边算法与直接分解算法的加速比大很多,当删除边数为 100 时,在图 RAR 上的加速比可以达到 10^3 . 这是由于被删除边只对上一时刻已经存在的 (k, h) -核产生影响,对时态核结构的影响范围是固定的,与加边算法相比减少了寻找候选插入子图和部分 (k, h) -核的过程,因此计算时间少很多. 自顶向下的时态图 (k, h) -核维护删边算法与直接分解算法相比,优势更明显.

图 10b 中,自顶向下的时态图 (k, h) -核维护删边算法与自底向上维护算法的加速比则较为复杂. 由于删边维护算法比较相似,此时图的自身结构会对实验结果产生较大影响.

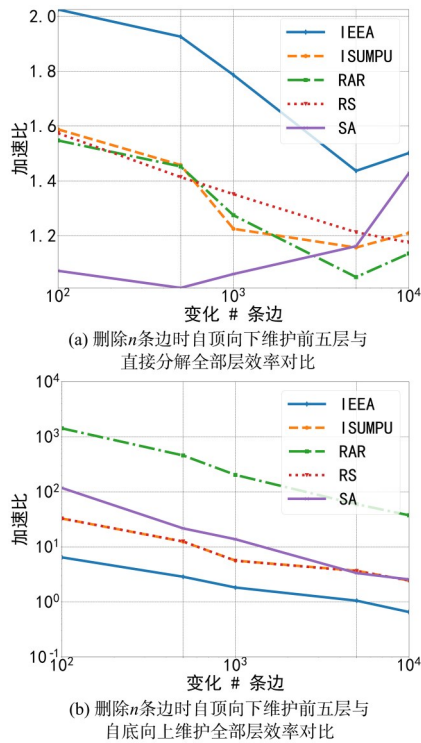


图 10 删边情况下维护前五层的效率

Fig.10 The algorithm efficiency of maintaining the first five layers layers when removing edges

6 结论

本文对时态图中的紧密子图维护问题进行研究,提出一种自顶向下的时态图 (k, h) -核维护算法,根据部分 (k, h) -核识别出核值受加边影响的

顶点并对其核值进行更新. 针对当前时刻已有存在的边被删除的情况,本文提出自顶向下的时态图 (k, h) -核维护删边算法,对上一时刻的 (k, h) -核做最小调整以得到当前时刻的核值. 此外,基于 10 个存在紧密子图的真实时态图数据集和两种对比算法,对本文提出的算法进行了一系列实验. 实验结果表明,与其他对比算法相比,提出的自顶向下的时态图 (k, h) -核维护算法的各方面性能都有很强的优势,不仅显著减小计算规模,减少计算时间,能直接挖掘最紧密的前 n 层 (k, h) -核,同时还具有较好的扩展性.

参考文献

- [1] Wu H H, Cheng J, Lu Y, et al. Core decomposition in large temporal graphs//Proceedings of 2015 IEEE International Conference on Big Data. Santa Clara, CA, USA:IEEE, 2015:649—658.
- [2] Wen D, Qin L, Zhang Y, et al. I/O efficient core graph decomposition: Application to degeneracy ordering. IEEE Transactions on Knowledge and Data Engineering, 2019, 31(1):75—90.
- [3] Huang X, Cheng H, Qin L, et al. Querying k -truss community in large and dynamic graphs//Proceedings of 2014 ACM SIGMOD International Conference on Management of Data. Snowbird, UT, USA:ACM, 2014:1311—1322.
- [4] Lu C, Yu J X, Wei H, et al. Finding the maximum clique in massive graphs. Proceedings of the VLDB Endowment, 2017, 10(11):1538—1549.
- [5] Wang Y Y, Cai S W, Yin M H. New heuristic approaches for maximum balanced biclique problem. Information Sciences, 2018(432):362—375.
- [6] Lin L L, Yuan P P, Li R H, et al. Mining stable quasi-cliques on temporal networks. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2021, DOI:10.1109/TSMC.2021.3071721.
- [7] Miao Z Q, Balasundaram B. An ellipsoidal bounding scheme for the quasi-clique number of a graph. INFORMS Journal on Computing, 2020, 32(3):763—778.
- [8] De Ridder M, Klein K, Kim J. Adapted k -core decomposition and visualization for functional magnetic resonance imaging connectivity networks//

- Proceedings of 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Honolulu, HI, USA: IEEE, 2018; 4134–4137.
- [9] Peng C B, Kolda T G, Pinar A. Accelerating community detection by using k -core subgraphs. 2014, arXiv:1403.2226.
- [10] Li R H, Qin L, Yu J X, et al. Influential community search in large networks. Proceedings of the VLDB Endowment, 2015, 8(5): 509–520.
- [11] 李阅志, 祝园园, 钟鸣. 基于 k -核过滤的社交网络影响最大化算法. 计算机应用, 2018, 38(2): 464–470. (Li Y Z, Zhu Y Y, Zhong M. k -core filtered influence maximization algorithms in social networks. Journal of Computer Applications, 2018, 38(2): 464–470.)
- [12] 曹玖新, 董丹, 徐顺, 等. 一种基于 k -核的社会网络影响最大化算法. 计算机学报, 2015, 38(2): 238–248. (Cao J X, Dong D, Xu S, et al. A k -core based algorithm for influence maximization in social networks. Chinese Journal of Computers, 2015, 38(2): 238–248.)
- [13] Zhang H H, Zhao H, Cai W, et al. Using the k -core decomposition to analyze the static structure of large-scale software systems. The Journal of Supercomputing, 2010, 53(2): 352–369.
- [14] He X, Zhao H, Cai W, et al. Analyzing the structure of earthquake network by k -core decomposition. Physica A: Statistical Mechanics and its Applications, 2015(421): 34–43.
- [15] Liu J X, Xu C, Yin C, et al. K -core based temporal graph convolutional network for dynamic graphs. IEEE Transactions on Knowledge and Data Engineering, 2020, DOI: 10.1109/TKDE. 2020. 3033829.
- [16] Bai W, Chen Y D, Wu D. Efficient temporal core maintenance of massive graphs. Information Sciences, 2020(513): 324–340.
- [17] Balasundaram B, Butenko S, Hicks I V. Clique relaxations in social network analysis: The maximum K -plex problem. Operations Research, 2011, 59(1): 133–142.
- [18] Batagelj V, Zaversnik M. An $O(m)$ algorithm for cores decomposition of networks. 2003, arXiv: cs/0310049.
- [19] Cheng J, Ke Y P, Chu S M, et al. Efficient core decomposition in massive networks//Proceedings of 2011 IEEE 27th International Conference on Data Engineering. Hannover, Germany: IEEE, 2011: 51–62.
- [20] Wen D, Qin L, Zhang Y, et al. I/O efficient Core Graph Decomposition at web scale//Proceedings of 2016 IEEE 32nd International Conference on Data Engineering. Helsinki, Finland: IEEE, 2016: 133–144.
- [21] Montresor A, De Pellegrini F, Miorandi D. Distributed k -core decomposition. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(2): 288–300.
- [22] Li R H, Yu J X, Mao R. Efficient core maintenance in large dynamic graphs. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(10): 2453–2465.
- [23] Sariyüce A E, Gedik B, Jacques - Silva G, et al. Streaming algorithms for k -core decomposition. Proceedings of the VLDB Endowment, 2013, 6(6): 433–444.

(责任编辑 杨可盛)