SIHC：一种高效的时态图上 k-core 查询算法

周军锋[1)] 王春花[1)] 杜 明[1)] 陈子阳[2)]

[1)](东华大学计算机科学与技术学院 上海 201620)

[2)](上海立信会计金融学院信息管理学院 上海 201620)

摘 要 许多实体之间的关系可以建模为时态图，其中每条边都与表示其发生的时间相关联. k-core 是捕获密集子图的基本模型，在近些年得到了广泛研究. 给定时间区间 $I=[s,e]$和 k 值，时态图 G 上的 k-core 子图查询从区间 I 对应的快照图 G_I 中返回相应的 k-core 子图. 针对时态图中的 k-core 子图查询问题，现有方法是基于 PHC 索引(Pruned Historical Core-Index)的算法. 对任意可能的 k 值，PHC 索引维护了所有可能出现在某个时间区间的 k-core 子图中的顶点集 S_k，且为集合中每个顶点存储了一组时间区间，用于判定该点是否属于给定时间区间的 k-core 子图. 基于 PHC 索引查询 k-core 子图时，需要访问 S_k 集合中的所有顶点，并判断每个顶点的可满足性. 由于 S_k 集合对应于最大区间快照图的 k-core 子图里的所有顶点，且实际中用户查询区间对应的快照图往往比最大区间快照图小得多，基于 PHC 索引的查询算法存在许多无效判断，需要对大量不在结果集中的顶点进行检测，且无效检测次数随着查询区间的缩短而增多，从而导致算法效率较低. 针对该问题，本文提出一种新的索引，即最短区间历史核索引 SIHC(Shortest Interval Historical Core Index). SIHC 索引的基本思想是通过维护最短 k 核区间到顶点的倒排表，查询处理时，可基于用户给定的时间区间定位到 SIHC 索引中满足条件的区间，进而直接得到满足条件的 k-core 子图中的顶点，从而避免了基于 PHC 索引进行查询时所需的大量无效判断. 我们从理论上证明了基于 SIHC 索引处理时态图上 k-core 子图查询的正确性，并设计了高效的索引构建算法. 最后，基于真实世界的时态图进行了实验，实验结果表明本文提出的算法比现有算法快 1～2 个数量级.

关键词 图数据管理；时态图；密集子图；k-core；最短 k 核区间

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2024.01045

SIHC: An Efficient Algorithm for k-core Query on Temporal Graphs

ZHOU Jun-Feng[1)] WANG Chun-Hua[1)] DU Ming[1)] CHEN Zi-Yang[2)]

[1)](*School of Computer Science and Technology, Donghua University, Shanghai* 201620)

[2)](*School of Information Management, Shanghai Lixin University of Accounting and Finance, Shanghai* 201620)

Abstract In real applications, the relationships between entities are usually modeled using a temporal graph, where each edge is associated with a timestamp denoting the time of the interaction between two entities at that moment. The k-core is a fundamental model for capturing dense subgraphs and has been widely studied in recent years. Given a temporal graph G and a time interval $I=[s,e]$, the snapshot graph G_I of G consists of all vertices of G and edges with timestamps within I, where edges with the same endpoints are merged into a single edge without timestamp. Based on the snapshot graph G_I, the k-core subgraph query on a temporal graph G returns the corresponding k-core subgraph from G_I. For the k-core subgraph query problem in temporal graphs, the state-of-the-art method is based on the PHC index (Pruned Historical Core-Index). For any possible value of k, the PHC index maintains a set S_k of vertices that may appear in

收稿日期：2023-06-12；在线发布日期：2024-01-23. 本课题得到国家自然科学基金项目(62372101，62272097，61873337)资助.**周军锋**(通信作者)，博士，教授，中国计算机学会(CCF)会员，主要研究领域为大图数据的查询处理技术. E-mail：zhoujf@dhu.edu.cn. **王春花**，硕士研究生，主要研究方向为密集子图挖掘. **杜 明**(通信作者)，博士，教授，中国计算机学会(CCF)会员，主要研究领域为信息查询和数据分析. E-mail：duming@dhu.edu.cn. **陈子阳**，博士，教授，主要研究领域为非结构化数据管理.

the k-core subgraph within any time interval, and for each vertex of S_k, PHC stores a set of time intervals to determine if it belongs to the k-core subgraph of a given time interval. When querying the k-core subgraph based on the PHC index, it is necessary to access all vertices of S_k and determine the satisfiability of each vertex. Since S_k corresponds to all vertices in the k-core subgraph of the snapshot graph corresponding to the largest interval, and the snapshot graph of the given query interval is often much smaller than the snapshot graph of the largest interval, the query algorithm based on the PHC index involves many invalid judgments, requiring detection of a large number of vertices that are not in the result set. Even worse, the number of invalid detections increases as the query interval becomes shorter, resulting in low efficiency. We show that it is the adopted interval which is not the shortest one that results in the low efficiency of PHC index. We theoretically prove the correctness of the k-core subgraph query on temporal graphs based on the shortest intervals, based on which we propose a new index called Shortest Interval History Core (SIHC) index. The basic idea of the SIHC index is maintaining an inverted table from the shortest k-core interval to vertices, based on which we can directly get vertices in the k-core subgraph that meet the requirements by accessing qualified inverted lists according to the query interval. In this way, we can avoid a large number of invalid judgments required by the PHC index. We show that the interval of PHC can be used to compute the shortest interval of ours, based on which we propose the index construction algorithm based on PHC index. To improve the efficiency of index construction, we propose an optimized index construction algorithm. We further prove that constructing SIHC index has the same time complexity with that of PHC, and they both have the same index size. Finally, we conduct rich experiments on real-world temporal graphs, and the experimental results demonstrate that the proposed algorithm is one to two orders of magnitude faster than existing algorithms.

Keywords graph data management; temporal graphs; dense subgraphs; k-core; shortest k-core interval

1 引 言

图可用于对现实世界中的实体及其复杂交互关系进行建模. 由于实体间交互操作频繁发生,实体间关系会随时间的推移不断演变,如社交网络[1]、协作网络[2]中与时间密切相关的关注或者合作等关系. 通过给图中表示交互关系的边加上时间信息,可以得到用于刻画关系随时间演变的时态图. 图 1 所示为一个时态图 G,其中边上的时间信息表示该边只在这个时刻存在于图中.

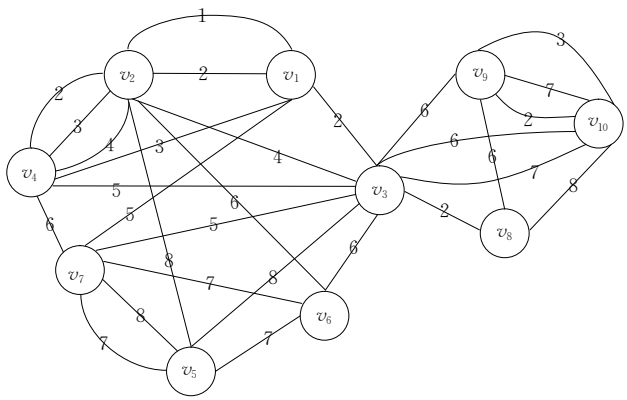

图 1 时态图 G

密集子图挖掘是图数据管理领域的研究热点[3-9]. k-core 是最重要的密集子图模型之一,其中每个点至少有 k 个邻居. k-core 子图挖掘广泛应用于各种实际场景,用于捕获实体间的密切交互关系[3,9-12]. 时态图中的 k-core 子图查询要求系统根据用户给定的区间 $I=[s,e]$和 k 值,从时态图 G 的快照图 G_I中返回相应的 k-core 子图 $C_k(G_I)$. 其中,快照图 G_I是通过截取时态图 G 中位于区间 I 内的边,且将具有相同端点的边合并之后得到的. 例如,图 2 是图 1 对应于时间区间[2,7]的快照图 $G_{[2,7]}$,其中虚线包围的子图 C_2是 2-core 子图,对应于查询 $I=[2,7]$,$k=2$,而虚线表示的 3-core 子图 C_3对应于查

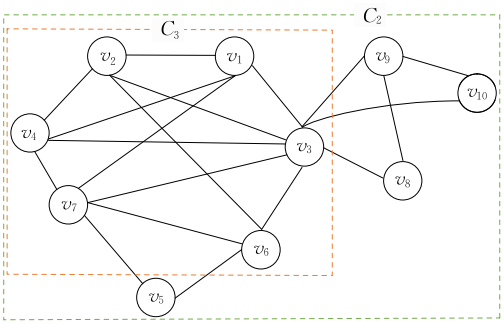

图 2　快照图 $G_{[2,7]}$

询 $I=[2,7]$，$k=3$.

与非时态图上求解所有可能 k 值对应的 k-core 子图(也叫 core 分解)不同，时态图上的 k-core 子图查询用于返回特定快照图 G_I 的 k-core 子图. 其应用包括但不限于以下几个方面：(1) 挖掘 k-core 社区. 由于 k-core 是一种常用的社区模型，如果用户想从科研协作网络(如 DBLP)了解某段时间内紧密协作的研究团队，或者从社交网络(如 Facebook 或者 WeChat)查找某个特殊时间段内人员之间的密切互动情况，或者从交易网络(如淘宝)监控某段时间内的异常交易情况，均可通过在时态图相应时间段的快照图中求解 k-core 子图来得到想要的结果；(2) 协助求解复杂查询问题. 由于 k-core 子图可高效求解，在很多实际应用中，k-core 计算的结果可用于进一步加速其他操作的执行效率，如极大团求解[13-14]、k 点连通分量求解[15]等，当这些问题移植到时态图上后，相应的 k-core 子图查询结果可用于压缩搜索空间，提升查询效率. 另外，如时态图上的 span-core 问题[16]，也可基于 k-core 子图查询来提升查询效率[17]. 因此，时态图上 k-core 子图查询效率的提升具有重要的实际意义和应用价值.

给定时态图 G 和时间区间 I，一种基本的 k-core 求解方法是在线求解方法. 该方法首先根据 I 得到相应的快照图 G_I，然后在 G_I 上使用针对普通图的 k-core 分解算法得到满足条件的结果. 由于不同用户在查询时给定的时间区间和 k 值都不同，每次查询时，该方法均需从时态图中提取快照图 G_I 并计算 k-core 子图 $C_k(G_I)$，单次查询的时间复杂度高达 $O(\log|E|+|E_I|)$，其中 E 表示时态图 G 的边集，E_I 表示快照图 G_I 的边集.

为了从时态图中快速得到满足条件的 k-core 结果，文献[17]提出了一种 PHC 索引，用于从时态图上高效获取 k-core 子图查询的结果，PHC 索引的基本思想是通过维护两种信息来快速判定一个顶点是否为满足条件的顶点，这两种信息包括：(1) 不同 k 值对应的顶点集合，以及(2)每个顶点集合中顶点对应的时间区间，具体来说，对于每个可能的 k 值，维护一个顶点集合 S_k，S_k 中的每个点 u 至少是一个区间快照图 G_I 中 k-core 子图 $C_k(G_I)$ 的顶点，假设时态图 G 中边上时间戳的最小和最大值分别用 1 和 $t_{\max}$ 表示，显然 S_k 表示的是最大区间快照图 $G_{[1,t_{\max}]}$ 中 k-core 子图的顶点集，进一步，对 S_k 中的每个点 u，维护一组时间区间 $L_u=\{I_1,I_2,\cdots,I_n\}$，其中每个时间区间 $I\in L_u$ 表示在对应的快照图 G_I 中，u 是 k-core 子图 $C_k(G_I)$ 中的顶点，给定时间区间 I 和 k 值，当基于 PHC 索引进行 k-core 子图查询时，文献[17]的方法首先根据 k 值定位到顶点集 S_k，然后检测 S_k 中的每个点，判断其是否为 G_I 的 k-core 子图中的点.

从以上介绍可以看出，使用 PHC 索引处理时态图上的 k-core 子图查询时，需要检测 S_k 集合中的所有点才能得到结果，由于 S_k 对应于最大区间快照图 $E_{[1,t_{\max}]}$，而用户查询的时间区间 I 相对于最大时间区间 $[1,t_{\max}]$ 来说通常很小，因而集合 S_k 中满足条件的顶点较少，当基于 PHC 算法进行 k-core 子图查询时，存在大量无效判断，效率较低.

为了进一步提升时态图上 k-core 子图查询的效率，本文提出一种新的索引 SIHC 来加快查询处理的速度，和 PHC 索引维护顶点到区间的倒排表不同，SIHC 索引维护的是区间到顶点的倒排表，且倒排表中的顶点均满足时间区间的要求，查询处理时，可根据用户给定的区间直接确定满足条件的倒排表，进而得到满足条件的结果，和基于 PHC 索引的查询方法存在大量无效判断相比，使用本文提出的 SIHC 索引进行查询时无需检测顶点是否满足条件，可显著提升查询效率，本文的主要贡献如下：

(1) 对 PHC 索引的构造机理和存在问题进行了深入分析，进而得出基于 PHC 索引的区间无法构建从区间到顶点的倒排表来加速查询处理的结论.

(2) 提出针对时态图的 SIHC 索引，以及基于 SIHC 索引进行 k-core 子图查询的高效方法来避免对无效顶点的检测，同时，对基于 SIHC 索引进行 k-core 子图查询处理的正确性进行了证明.

(3) 提出两种构建 SIHC 索引的算法，分别是基本算法和优化算法，基本算法在 PHC 索引构建完毕的基础上，根据 PHC 索引中的信息生成 SIHC 索引，优化算法则无需预先生成 PHC 索引，从而可以避免生成 PHC 索引过程中的冗余操作，提升 SIHC

索引的构建效率.

(4)在真实数据集上对本文方法的高效性进行了验证,实验结果表明,本文提出的算法比现有算法快1～2个数量级,可以显著提高时态图中 k-core 子图查询的效率.

本文第2节介绍基础知识和相关工作;第3节分析 PHC 索引存在的问题,基于这些问题;在第4节介绍本文提出的 SIHC 索引,以及相应的查询算法;第5节给出实验结果与分析;最后在第6节总结本文工作.

2 基础知识和相关工作

2.1 相关概念和问题定义

给定时态图 $G=(V,E)$,其中 V 是顶点集合,E 是边集,E 中的每条边是一个三元组 (u,v,t),表示点 u 和点 v 之间的边在时刻 t 存在,G 在给定时间区间 $I=[s,e]$ 上的快照图表示为 $G_I=(V,E_I)$,其中 $E_I=\{(u,v)\mid(u,v,t)\in E,t\in[s,e]\}$,不失一般性,假设 G 中边上最小的时刻为1,最晚时刻为 $t_{\max}$,对快照图 $G_{[1,t_{\max}]}$ 中的每条边 (u,v) 来说,顶点 u 和 v 在时态图中可能存在多条对应于不同时刻的边,因此 $|E_{[1,t_{\max}]}|\leq|E|$,顶点 u 在快照图 G_I 中的邻居顶点用 $|N_u^I|$ 表示,$|N_u^I|=\{v\mid(u,v)\in E_I\}$,顶点 u 在 G_I 中的度用 $deg_I(u)$ 表示,$deg_I(u)=|N_u^I|$,表1是本文常用符号及意义.

表1 本文常用符号及意义

符号	意义
$G=(V,E)$	无向时态图
$I=[s,e]$	时间区间
E_I	$E_I=\{(u,v)\mid(u,v,t)\in E,t\in I\}$
G_I	$G_I=(V,E_I)$
$t_{\max}$	最大时间戳
$d_{\max}$	$G_{[1,t_{\max}]}$ 的最大度
$C_k(G_I)$	G_I 中的 k-core 子图
$core_I(u)$	u 在 G_I 中的核数
$CT_s(u)_k$	u 在时刻 s 的核时间
$k_{\max}$	$G_{[1,t_{\max}]}$ 中最大核数
SI_k	最短 k 核区间集合
V_I	最短区间 I 对应的顶点集合
$\bar{t}$	每个顶点不同 k 值对应的有效 k 核区间数量的平均值

定义1. k 核(k-core). 给定快照图 G_I,正整数 k,G_I 中的 k-core 子图是 G_I 中满足每个点的度大于等于 k 的最大子图,用 $C_k(G_I)$ 表示.

问题定义(时态图上的 k-core 查询). 给定时态图 G,时间区间 I 和正整数 k,返回 G_I 中 k-core 子图的所有点,即返回 $C_k(G_I)$ 中的点.

例1. 给定图1的时态图 G,时间区间 $[2,7]$ 和 $k=3$,则 G 在时间区间 $[2,7]$ 上的快照图 $G_{[2,7]}$ 如图2所示,图2中的 C_3 表示 $G_{[2,7]}$ 上的3-core 子图.

2.2 core 分解

定义2. 核数(Core Number). 给定一个快照图 G_I 和其中的一个顶点 u,u 的核数是包含 u 的所有非空 k-core 中 k 的最大取值,用 $core_I(u)$ 表示,这时,核数 $core_I(u)$ 也称为顶点 u 关于区间 I 的核数.

为了支持高效的 k-core 查询操作,通常需要提前计算每个顶点的核数,计算所有点核数的操作称为核分解,计算快照图中所有点的核数可以采用贪心方法,具体过程如算法1所示[3],该方法依次从图中剥离度数最小的点以及与该点相连的边,在此过程中为顶点计算相应的核数,该算法在第3行使用桶排序,其总的时间复杂度为 $O(|E|)$,计算 k-core 的算法与核分解算法相似,区别在于每次删除的是度数小于 k 的点,直到图中不存在度数小于 k 的点时,剩余子图就是所求的 k-core,文献[18]进一步考虑通过 GPU 来提升核分解操作的效率.

算法1. CoreDecomp.

输入:$G_{[s,e]}=(V,E_{[s,e]})$

输出:$core_{[s,e]}(u)$ for all u

1. $c\leftarrow 0$
2. WHILE $V\neq\varnothing$ DO
3. $\quad u\leftarrow\arg\min_{u\in V} deg_{[s,e]}(v)$
4. $\quad c\leftarrow\max(deg_{[s,e]}(u),c)$
5. $\quad core_{[s,e]}(u)\leftarrow c$
6. $\quad V\leftarrow V\backslash\{u\}$
7. $\quad$ FOREACH $v\in N_u^{[s,e]}$ DO
8. $\quad\quad deg_{[s,e]}(v)\leftarrow deg_{[s,e]}(v)-1$
9. RETURN $core_{[s,e]}(u)$ for all u

例2. 图2是图1的 G 在区间 $[2,7]$ 上的快照图 $G_{[2,7]}$,其中 C_2 为2-core,C_3 为3-core,所以

$$core_{[2,7]}(v_1)=core_{[2,7]}(v_2)=core_{[2,7]}(v_3)=$$
$$core_{[2,7]}(v_4)=core_{[2,7]}(v_6)=core_{[2,7]}(v_7)=3,$$

剩余点的核数皆为2,以图2求解3-core为例,算法首先一次性从图2中删除所有度数小于3的顶点,包括 v_5、v_8、v_{10} 三个顶点,同时修改相关顶点的度,即 v_7 和 v_6 的度减1,v_3 和 v_9 的度同时减2,之后只有 v_9 的度小于3,下次循环时,删除 v_9,然后将 v_3 的度减1,至此,由于没有顶点的度小于2,因此虚线 C_3 中的顶点都是3-core 中的顶点.

此外,考虑到实际应用中存在数据图动态变化的情况,有必要研究数据图发生变化后,如何高效更

新顶点核数的问题，也叫核数维护问题，对于该问题，文献[19]提出了一种在流图中插入或删除边时核数维护的算法，文献[20]基于顶点的简并度顺序提高核数维护的效率，文献[21]提出了一种基于并行顺序的核数维护方法来提升效率，文献[22-23]分别考虑分布式环境和外部存储器中的核数维护，文献[24]中考虑了二分图上的核维护问题并提出了高效的边插入和删除算法.

2.3　时态图上的 k-core 查询

最近几年，时态图上的 k-core 查询问题也得到了学术界的广泛关注和深入研究[11,16-17,25-29]，这些工作中，文献[17]和本文解决的问题相同，都是针对给定的时态图 G，时间区间 I 和正整数 k，返回 G_I 中 k-core 子图的所有顶点，对于该问题，最直观的在线解决方案(表 2 中的Online算法)首先提取快照图 G_I，然后计算其 k-core 子图，复杂度高达 $O(\log|E|+|E_I|)$，针对该问题，文献[17]提出了 PHC 索引来加速查询处理，然而，PHC 索引在处理时态图上的 k-core 查询时，需要检测每个顶点的可满足性，时间代价仍高达 $O(|V|\log t)$[17]，由于满足条件的顶点仅仅是图中顶点集合的一个子集，且查询区间越小，满足条件的顶点越少，因而基于 PHC 索引的查询算法仍然存在大量冗余操作来检测无用顶点的可满足性，本文在第 3 节对其存在的问题进行深入分析，并基于分析的结果提出新的索引来避免检测无用顶点的可满足性，进一步提升查询处理的效率，几种方法的对比如表 2 所示，其中 $|E|$ 为时态图中时态边的数目，$|V|$ 为时态图中顶点数目，m_I 为时态图中位于区间 I 中的边数，$\bar{t}$ 为不同 k 值对应的有效 k 核区间个数的平均值，$m^*=|E_{[1,t_{\max}]}|$ 为快照图 $G_{[1,t_{\max}]}$ 中的边数，$d_{\max}$ 为 $G_{[1,t_{\max}]}$ 中的最大度，$|SI_k|$ 表示 k 值对应的最短 k 核区间数量，$\bar{V}_I$ 为 SIHC 索引中最短 k 核区间顶点集合中顶点数量的平均值，$|SI_q|$ 为查询区间$[s,e]$包含的最短 k 核区间数量.

表 2　不同算法的比较

算法	查询时间	索引规模	索引时间
Online	$O(\log\|E\|+m_I)$	$\varnothing$	$\varnothing$
PHC[17]	$O(\|V\|\log\bar{t})$	$O(\bar{t}\cdot m^*)$	$O(m^*\cdot\bar{t}\cdot d_{\max})$
SIHC(Ours)	$O(\log\|SI_k\|+\|\bar{V}_I\|\cdot\|SI_q\|)$	$O(\bar{t}\cdot m^*)$	$O(m^*\cdot\bar{t}\cdot d_{\max})$

考虑到实际应用中，用户可能不知道所需查询结果对应的精确时间区间 I，文献[25]认为返回 I 对应的所有子区间的 k-core 子图也有其相应的现实意义，该文中，作者提出了时态核分解的操作，并基于一种高效的时态图组织方式 TEL 和三种剪枝技术来高效返回不同子区间对应的 k-core 子图，由于文献[25]返回给定时间区间 I 对应的所有子区间的 k-core 子图，自然也包含本文的返回结果，文献[25]提出的算法首先求解的是最大区间的 k-core 子图，即本文查询的结果，因此，当使用文献[25]的方法来求解本文的查询结果时(即对文献[25]的算法进行修改，在返回第一个结果后立即结束算法的执行)，本质上和表 2 的 Online 算法类似，都是首先得到快照图 G_I，然后得到 G_I 上的 k-core 子图，因而二者具有相同的查询复杂度.

和本文及文献[17]研究的问题相比，文献[16]提出了 span-core 的概念，要求返回结果中的所有边在区间 I 的每个时刻都必须存在，因此，在给定 k 值和时间区间 I 的前提下，文献[16]的查询结果是本文查询结果的子图，从以上讨论可知，本文和文献[17]的查询问题可以认为是对文献[16]查询约束条件进行适度弱化形成的，而文献[25]的查询问题则是在本文和文献[17]查询问题的基础上进一步放宽约束条件得到的，

此外，还有学者研究了时态图上基于更复杂社区模型的查询问题，文献[11]提出了(θ,τ)-persistent k-core 的概念，用于从时态图中找到在不小于 τ 的时间段内，其所有长度为 θ 的子区间均存在的最大 k-core 子图，对文献[11]而言，其涉及的长度为 θ 的快照图与本文的快照图相同，文献[29]和文献[11]基于相同的社区定义，返回包含给定查询点的所有连通 k-core 子图，文献[26]则提出了(k,h)-core 的概念，要求 k-core 子图中任意两点间的时态边数量不小于 h，文献[27]进一步提出基于弱约束条件的 quasi-(k,h)-core，并基于该模型特性提出了当时态图发生变化时支持(k,h)-core 高效维护的解决方案，文献[30]提出了(l,δ)-maximal bursting core ((l,δ)-MBC)模型，用于从时态图中识别短时间内形成的密集子图，和以上工作所处理的时态图不同，文献[28]研究的是加权时态图上的稠密子图挖掘问题.

2.4　其他相关工作

根据实际应用中需求多样化的特点，研究者提出了各种核的变体，包括 (k,p)-core[31]、(k,r)-core[32]、(k,d)-core[33]、(α,β)-cluster[34]、(p,n)-core[35] 等，同时，有研究人员提出 k-truss[36]、k-clique[37]、k-ecc[38] 等模型来获取密集子图.

此外，k-core 相关问题也并非仅限于时态图和一般无向图，研究人员也研究了其他数据图上的 k-core

查询问题,包括不确定图[39]、有向图[40]、二分图[41-42]、超图[43]、符号网络[35]等.

3 PHC 索引分析

本节讨论 PHC 索引[17]并分析其查询处理低效的原因,并以此为基础提出本文的解决方案,PHC 索引是在 HC 索引(Historical Core Index)的基础上提出的,HC 索引的基本思想是:对图中每个顶点 u,存储 u 在所有可能快照图上的核数,如图 3(a)所示,处理查询时,根据查询中的时间区间通过简单的查找操作即可在 $O(1)$时间判断 u 是否为满足条件的顶点,尽管速度快,但 HC 索引的空间复杂度高达 $O(t_{\max}^2 \times |V|)$,这里 $t_{\max}$ 为时态图中时刻的最大值,$|V|$为时态图中的顶点数,为此,需要引入如下概念,以便对 HC 索引进行压缩,降低其空间复杂度.

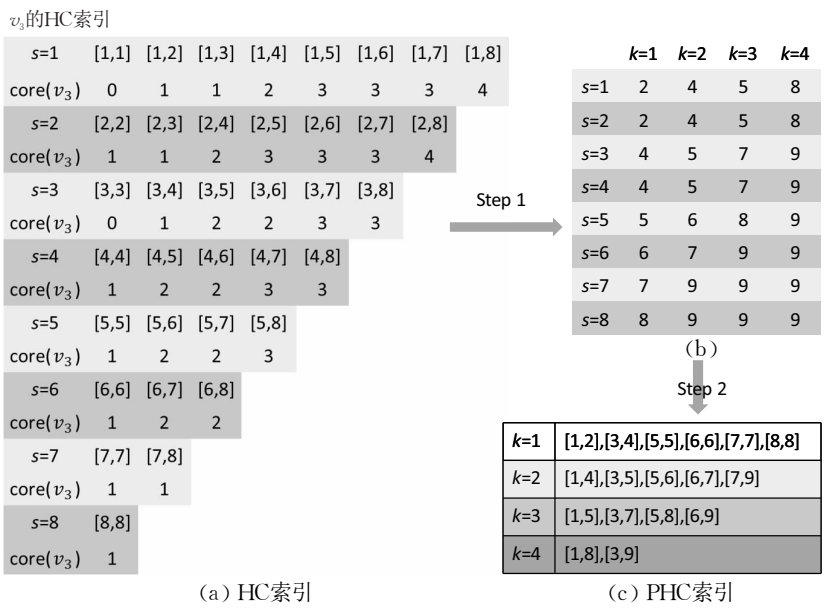

图 3 从 HC 索引到 PHC 索引的压缩过程

定义 3. 核时间(Core Time[17]). 给定时态图 G,点 u,正整数 k 和时刻 s,点 u 针对时刻 s 的核时间表示使 $core_{[s,e]}(u) \geq k$ 成立的最小时间 e,即点 u 变为 k-core 子图 $C_k(G_{[s,e]})$中的点的最小时间,用 $CT_s(u)_k$表示.

定义 4. k 核区间(Core Interval). 给定时态图 G,点 u 和时刻 s,称以 s 为起始值,以点 u 的任一核时间 $CT_s(u)_k$为结束值的区间为 u 的 k 核区间.

例 3. 考虑图 3(a)中展示的 v_3 顶点的区间,可以看出以 2 开始的 7 个区间是[2,2],[2,3],…,[2,8],给定 $s=2$,尽管 v_3 在[2,2],[2,3]对应的快照图中核数都等于 1,根据定义 3,v_3 的核时间等于 2,当 $k=2,3,4$ 时,v_3 的核时间分别为 $CT_2(v_3)_2=4$,$CT_2(v_3)_3=5$,$CT_2(v_3)_4=8$,因此,当 $s=2$ 时,v_3 有四个 k 核时间 2、4、5、8,分别对应 1 核区间[2,2]、2 核区间[2,4]、3 核区间[2,5]以及 4 核区间[2,8].

基于 k 核区间的定义可知,每个非 k 核区间都对应唯一核值相同且被其包含的 k 核区间,因此可安全删除 HC 索引中的非 k 核区间,从而压缩存储空间,相应的压缩规则如下,结果如图 3(b)所示.

压缩规则 1. 删除非 k 核区间.

图 3(b)是在图 3(a)的基础上删除 k 核区间的结果,观察图 3(b)可知,对同一顶点来说,不同起始值的 k 核区间存在包含和被包含的情况,如对于 v_3 和 $k=2$ 来说,[1,4],[2,4]都是 2 核区间,但[2,4]⊂[1,4],为了进一步压缩索引空间,需引入如下概念.

定义 5. 有效 k 核区间(Effective k-Core Interval). 给定时态图 G,点 u 及核数 k,称 u 的 k 核区间中不被其他区间包含的区间为 u 的有效 k 核区间.

文献[17]中进一步证明了,有效 k 核区间可用于正确回答时态图上的 k-core 子图查询问题,因此有如下压缩规则.

压缩规则 2. 删除无效 k 核区间.

根据压缩规则 2,可将图 3(b)每列的无效 k 核区间予以删除,所得结果对应于图 3(c)一行的有效 k 核区间,图 3(c)是 PHC 索引,给定时态图 G,当基于 PHC 索引查询 $G_{[s,e]}$ 中的 k-core 时,需验证 $G_{[1,t_{\max}]}$ 中核数≥k 的所有点,看其是否存在一个有效 k 核区间 I,满足以下两个条件:(1)在所有起始值小于等于 s 的区间中,I 的起始值最大;(2)I 的终止值介

于 s 和 e 之间，基于 PHC 索引回答 k-core 子图查询的时间复杂度为 $O(n_k \times \log\bar{t})$，其中 n_k 为 $G_{[1,t_{\max}]}$ 中 k-core 子图的顶点数量.

假设 S_k 表示快照图 $G_{[1,t_{\max}]}$ 中 k-core 子图的所有顶点构成的集合，则根据 PHC 索引查询 k-core 子图时需验证 S_k 中的所有顶点才能得到结果，由于实际中的时态图时间跨度较大，相比较而言，用户查询的时间区间相对很小，因而集合 S_k 中只有少量满足条件的顶点，因此 PHC 索引用于查询 k-core 子图时存在大量无效判断，如果在 PHC 索引的基础上建立区间到顶点的映射关系，则可能返回错误的结果.

例 4.　图 4(a)是 PHC 索引中 v_4，v_5，v_6 在 $k=2$ 时对应的时间区间，假设 $k=2$，查询区间为[6,8]，则根据 PHC 索引可知图 4(a)中满足条件的顶点是 v_5 和 v_6，如果基于图 4(a)构建区间到顶点的映射关系，则结果如图 4(b)所示，当基于图 4(b)进行查询时，假设使用和 PHC 索引一样的查询处理规则，那么满足条件的区间是[5,7]，表示满足条件的顶点只有 v_6，存在漏解的问题，即 false-negative 问题，而如果将判断条件放松，仅要求 $s\leq 6$ 且 $6\leq e\leq 8$，这时满足条件的区间有[1,6]，[1,7]，[5,6]，[5,7]，由图 4(b)可知，v_4，v_5，v_6 均是满足条件的顶点，这时存在返回错误结果的问题，即 false-positive 问题.

PHC索引

(a)

k=2	v_4	v_5	v_6
	[1,3],[3,5],[5,6],[6,9]	[1,7],[8,9]	[1,6],[5,7],[8,9]

(b)

k=2	[1,3]	[1,6]	[1,7]	[3,5]	[5,6]	[5,7]
	v_4	v_6	v_5	v_4	v_4	v_6

图 4　基于 PHC 索引构建的区间到顶点的倒排表

4　基于 SIHC 索引的查询算法

4.1　SIHC 索引

定义 6.　最短 k 核区间(Shortest k-Core Interval). 给定顶点 u 的 k 核区间 $I=[s,e]$，I 是 u 的最短 k 核区间当且仅当 $core_{[s+1,e]}(u)<k$.

直观来看，u 的最短 k 核区间 I 是使 u 成为非空 k-core 子图中顶点的最短区间.

定理 1.　给定时态图 G，顶点 u 及时间区间[s,e]，则 u 是 k 核子图 $C_k(G_{[s,e]})$ 中的顶点当且仅当 u 存在一个最短 k 核区间[s',e']满足[s',e']⊆[s,e].

定理 1 说明，最短 k 核区间可用于检测顶点是否为相应快照图中核数大于等于 k 的顶点，基于此，本文提出最短区间历史核索引(Shortest Interval Historical Core Index)，简称 SIHC 索引，对每个 k 值，SIHC 索引维护了每个最短 k 核区间到相应顶点的映射关系，该映射关系通过两步建立，首先，得到所有顶点的最短 k 核区间集合 $SI_k=\bigcup_{u\in V}SCI_k(u)$，这里 $SCI_k(u)$ 表示顶点 u 的最短 k 核区间集合，其次，得到每个最短 k 核区间 I 到顶点集合 V_I 的映射关系，这里 $V_I=\{u\mid I\in SCI_k(u)\}$.

例 5.　给定图 1 的时态图 G，图 5 是 G 的 SIHC 索引，对于每个 k 值，其最短区间到顶点集的映射关系用相应的两行信息表示，第一行是最短 k 核区间，每个区间 I 占一列，第二行中和 I 同一列的元素表示顶点集 V_I，是最短 k 核区间 I 对应的顶点集，表示给定时态图 G，查询区间 I 和 k，V_I 中的顶点位于快照图 G_I 的 k-core 子图中，以 v_3 为例，v_3 的最短 2 核区间有[2,4]，[4,5]，[5,6]，[6,7]，那么这四个区间均出现在 $k=2$ 时的最短 k 核区间集合 SI_2 中，如图 5 第一行所示，相应的，v_3 出现在这四个最短 2 核区间对应的顶点集合中，如图 5 第二行所示，给定图 1 的时态图 G，$I=[2,3]$以及 $k=2$，由图 5 的 SIHC 索引可知，和最短 2 核区间[2,3]对应的顶点 v_1，v_2，v_4 都位于快照图 $G_{[2,3]}$ 的 2-core 子图中.

定理 2.　任意顶点 u 的有效 k 核区间和最短 k 核区间具有一一对应关系.

例 6.　给定图 3(c)，v_3 的有效 2 核区间为 $ECI_2(v_3)=\{[1,4],[3,5],[5,6],[6,7],[7,9]\}$，其中[1,4]对应的最短 2 核区间是[2,4]，[3,5]对应的最短 2 核区间是[4,5]，区间[5,6]和[6,7]既是有效 2 核区间，也是最短 2 核区间，区间[7,9]不是有效 2 核区间，仅用于标识起始值大于等于 7 的区间中不存在有效 2 核区间，因此，v_3 的最短 2 核区间集合为 $SCI_2(v_3)=\{[2,4],[4,5],[5,6],[6,7]\}$.

k=2	[2,3]	[2,4]	[2,6]	[3,5]	[3,6]	[4,5]	[4,6]	[5,6]	[6,7]	[6,8]	[7,7]
	v_1,v_2,v_4	v_3	v_8	v_1,v_7	v_9,v_{10}	v_2,v_3,v_4	v_6	v_3,v_4,v_7	v_3,v_9,v_{10}	v_2,v_8	v_5,v_6,v_7
k=3	[2,5]	[2,6]	[2,8]	[4,7]	[5,8]						
	v_1,v_2,v_3,v_4	v_7	v_8,v_9,v_{10}	v_2,v_3,v_4,v_6,v_7	v_3,v_5,v_6,v_7						
k=4	[2,8]										
	$v_1,v_2v_3,v_4,v_5,v_6,v_7$										

图 5　图 1 时态图 G 对应的 SIHC 索引

定理 3. SIHC 索引和 PHC 索引具有相同的索引规模,均为 $O(\bar{t}\times|E_{[1,t_{\max}]}|)$,这里 $\bar{t}$ 为不同 k 值对应的有效 k 核区间个数的平均值,$|E_{[1,t_{\max}]}|$ 为快照图 $G_{[1,t_{\max}]}$ 中的边数.

定理 3 说明,基于最短 k 核区间构建 SIHC 索引不会增大索引规模.

4.2 查询处理

定理 4. 假设 $SI_k=\bigcup_{u\in V}SCI_k(u)$ 表示所有顶点的最短 k 核区间集合,$V_I=\{u\mid I\in SCI_k(u)\}$ 是以 I 为其最短 k 核区间的顶点集,给定查询区间 $[s,e]$ 和 k 值,如果存在 $I\in SI_k$,满足 $I\subseteq[s,e]$,那么 V_I 中的顶点都是 k 核子图 $C_k(G_{[s,e]})$ 中的顶点.

定理 4 说明,基于 SIHC 索引进行时态图上的 k 核子图查询时,只需找到被查询区间包含的最短 k 核区间,即可得到满足条件的顶点,具体过程如算法 2 所示,和 PHC 相比,基于 SIHC 索引进行查询时,无验证所有顶点的可满足性,从而避免大量无效判定的代价.

算法 2. SIHC-Query.

输入:I,k

输出:$C_k(G_I)$

1. $C_k(G_I)\leftarrow\varnothing$ /*k-core 子图中的顶点集*/
2. FOREACH $I'\in SI_k\wedge I'\subseteq I$ DO
3. $C_k(G_I)\leftarrow C_k(G_I)\cup V_{I'}$
4. RETURN $C_k(G_I)$

定理 5. 算法 2 基于 SIHC 索引查询时不存在漏解(false-negative)和返回错误结果(false-positive)的问题.

例 7. 给定图 1 的时态图,假设查询区间为 $[2,7]$,$k=3$,根据图 5 所示 SIHC 索引,可知最短 3 核区间集合 $SI_3=\{[2,5],[2,6],[2,8],[4,7],[5,8]\}$,其中 $[2,5]$,$[2,6]$,$[4,7]$ 是被包含在 $[2,7]$ 中的区间,$[2,5]$,$[2,6]$,$[4,7]$ 所映射的顶点集 $V_{[2,5]}$,$V_{[2,6]}$,$V_{[4,7]}$ 都是快照图 $G_{[2,7]}$ 里 3-core 子图中的顶点,由图 5 可知,当 $k=3$ 时,$V_{[2,5]}=\{v_1,v_2,v_3,v_4\}$,$V_{[2,6]}=\{v_7\}$,$V_{[4,7]}=\{v_1,v_2,v_3,v_4,v_6,v_7\}$,因此,对该查询而言,满足条件的顶点集合为 $V_{[2,5]}\cup V_{[2,6]}\cup V_{[4,7]}=\{v_1,v_2,v_3,v_4,v_6,v_7\}$.

在 SIHC 索引的实现中,针对不同 k 值的最短 k 核区间根据起始值进行了排序,查询时,首先利用折半查找定位首个被查询区间包含的最短 k 核区间,代价是 $\log|SI_k|$,之后顺序找到所有被包含的最短 k 核区间,假设 $|\bar{V}_I|$ 为 SIHC 索引中最短 k 核区间顶点集合中顶点数量的平均值,$|SI_q|$ 为查询区间 $[s,e]$ 包含的最短 k 核区间数量,则访问所有 V_I 集合中顶点的代价为 $O(|\bar{V}_I|\times|SI_q|)$,同时,我们维护了从每个最短 k 核区间 $[s,e]$ 到起始值大于 s 的第一个最短 k 核区间的指针,则基于 SIHC 索引在时态图上进行 k 核子图查询的时间复杂度为 $O(\log|SI_k|+|\bar{V}_I|\times|SI_q|)$.

4.3 索引构建

定理 6. 给定顶点 u 按开始时间升序有序的有效 k 核区间集合 $ECI_k(u)$,并假设 $[s_1,e_1]$,$[s_2,e_2]$ 为 $ECI_k(u)$ 中两个相邻的有效 k 核区间($e_1\neq e_2$),则与 $[s_1,e_1]$ 对应的最短 k 核区间为 $[s_2-1,e_2]$.

定理 6 说明,基于有效 k 核区间求解最短 k 核区间是可行的.

4.3.1 基础算法

由第 3 节的分析可知,基于 PHC 构建倒排索引会导致查询错误,同时,根据定理 6,可从点 u 的有效 k 核区间求得最短 k 核区间,因此本文构建 SIHC 索引的基本思想是首先将 PHC 索引中的有效 k 核区间替换为最短 k 核区间,然后对 PHC 索引进行一遍扫描来得到 SIHC 索引,如算法 3 所示,算法 3 第 1 行调用文献[17]中的方法构建 PHC 索引,第 2 行初始化核数数组 $CORE$,第 3~9 行根据每个点两个相邻的有效 k 核区间得到最短 k 核区间,其中第 3 行遍历图中每个顶点 u,第 4 行对 u 的每个 k 值,在第 6 行扫描 u 在当前 k 值下对应的所有区间,第 11~12 行对所有的最短 k 核区间按照起始值升序排序,相同起始值按照结束值升序排序.

算法 3. SIHC-B.

输入:$G=(V,E)$

输出:SI

1. Construct PHC index[17]
2. $CORE\leftarrow core_{[1,t_{\max}]}$
3. FOREACH $u\leftarrow V$ DO
4. FOREACH $k\in[1,CORE[u]]$ DO
5. $CI\leftarrow ECI_k(u)$
6. FOREACH $i\in[2,len(CI)]$ DO
7. $[s_1,e_1]\leftarrow CI[i-1]$,$[s_2,e_2]\leftarrow CI[i]$
8. $I\leftarrow[s_2-1,e_1]$
9. $SI_k\leftarrow SI_k\cup\{I\}$, $V_I\leftarrow V_I\cup\{u\}$
10. $k_{\max}\leftarrow$the maximum value in $CORE$
11. FOREACH $k\in[1,k_{\max}]$ DO
12. Sort SI_k in ascending order
13. RETURN SI

定理 7. 算法 3 的时间复杂度为 $O(|E_{[1,t_{\max}]}|\times\bar{t}\times d_{\max})$.

4.3.2 优化算法

虽然算法 3 的思路比较直观,但需要首先构建

PHC 索引，然后才能基于定理 6 得到最短 k 核区间，效率较低，为了提升 SIHC 索引的构建效率，本文提出基于 k 核区间直接求解最短 k 核区间的方法，其正确性可由定理 8 得到保证.

定理 8. 给定顶点 u 的 k 核区间集 $CI_k(u)$，$CI_k(u)$中每个不同的区间结束值对应一个最短 k 核区间.

基于定理 8，本文提出一种三阶段 k 核区间求解方法，如算法 4 所示.

算法 4. SIHC-O.

输入：$G=(V,E)$

输出：SI

1. $core_{[1,t_{\max}]} \leftarrow CoreDecomp(G_{[1,t_{\max}]})$
2. $CT_1 \leftarrow Computer1CT(G, core_{[1,t_{\max}]})$
3. $SI \leftarrow ComputerSI(G, core_{[1,t_{\max}]}, CT_1)$
4. $k_{\max} \leftarrow$ the maximum value in $core_{[1,t_{\max}]}$
5. FOREACH $k \in [1, k_{\max}]$ DO
6. Sort SI_k in ascending order
7. RETURN SI

(1)第一阶段(第 1 行)，求得所有顶点在快照图 $G_{[1,t_{\max}]}$中的核数.

(2)第二阶段(第 2 行)，求解起始值等于 1 时所有顶点在所有 k 值对应的核时间.

(3)第三阶段(第 3～6 行)，求解其它起始时刻所有顶点在所有 k 值对应的最短 k 核区间，得到 SIHC 索引.

这种处理策略的好处是在每个阶段只需处理一次时态图中的边便可得到所需的结果，下面分别介绍每个阶段的处理细节及注意事项.

阶段 1. 求得所有顶点在快照图 $G_{[1,t_{\max}]}$中的核数.

该操作通过调用现有的非时态图上的高效 k 核分解算法来完成，

阶段 2. 求解起始时刻 $s=1$ 时所有顶点在所有 k 值对应的 k 核时间.

具体求解方法是，按照时间从大到小的顺序，依次删除每个时刻 $e(2\leq e\leq t_{\max})$的边，同时更新被删边两端顶点的核数，并根据核数的变化得到每个顶点 u 在快照图 $G_{[1,e-1]}$中的核数，进而得到相应的 k 核区间，需要注意的是，如果删边后 u 的邻居中核数大于等于 k 的顶点少于 k 个，那么 u 核数一定发生变化，为了正确高效处理顶点核数的变化，需要引入区间核邻居的概念，用于记录删边前后复杂状态信息.

定义 7. 区间核邻居(Interval Core Neighbors). 给定时态图 G，时间区间 I 和顶点 u，若 u 的邻居 v 满足 $core_I(v)\geq core_I(u)$，则称 v 是 u 关于 I 的区间核邻居.

假设从 G_I中删除的边是(u,v)，若 v 是 u 关于$[s,e]$的区间核邻居且 v,u 之间仅有一条位于$[s,e]$中的边，则删除该边后，u 的区间核邻居会减少一个，由于时态图中 u,v 之间在$[s,e]$中可能存在多条边，因此对于 u 的每个区间核邻居 v，需要记录边(u,v)在 I 中出现的次数，如果删边之后变为 0，说明 v,u 之间不存在区间 I 内的边，可将其区间核邻居数减一，假设 $x_{(u,v)}=|\{t|(u,v,t)\in E\wedge t\in I\}|$ 表示(u,v)在时间段 I 中出现的次数，我们用哈希表 CN_u记录 u 的区间核邻居集合中每个顶点 v 到 $x_{(u,v)}$的映射关系，$CN_u[v]=x_{(u,v)}$.

算法 5 为计算时刻 1 所有顶点在所有 k 值对应核时间的伪代码，第 1 行对核数组 $CORE$ 和集合 Q 进行初始化，第 2～3 行计算每个顶点 u 相对于$[1,t_{\max}]$的区间核邻居，第 4～16 行从大到小依次删除每个时刻的边，更新顶点的核数，并计算 u 在时刻 1 的核时间(第 11～12 行)，具体来说，第 5 行的作用是求得需要重新计算核数的顶点并将其暂存在集合 Q 中，第 7～16 行求这些顶点的部分核时间及区间核邻居，其中，第 9 行计算每个顶点 u 的核数，第 10 行更新 u 相对于$[1,e-1]$的区间核邻居，第 11～12 行基于 u 的核数改变情况求其核时间，第 13～16 行将需要重新计算核数的顶点加入 Q 中，最后在第 17 行返回起始时刻 $s=1$ 时所有顶点的 k 核时间.

算法 5. Computer1CT.

输入：$G, core_{[1,t_{\max}]}$

输出：CT_1

1. $CORE \leftarrow core_{[1,t_{\max}]}, Q \leftarrow \varnothing$
2. FOREACH $u\in V$ DO
3. $CN_u \leftarrow$ComputeCN($[1,t_{\max}]$, u)
4. FOREACH e: from $t_{\max}$ to 2 DO
5. DelEdgesInterval(e, Q)
6. $I \leftarrow [1, e-1]$
7. WHILE $Q\neq\varnothing$ DO
8. $u \leftarrow$remove a node from Q, $oc \leftarrow CORE[u]$
9. $CORE[u] \leftarrow$ComputeCore(I, u)
10. $CN_u \leftarrow$ComputeCN(I, u)
11. FOREACH k from oc to $CORE[u]+1$ DO
12. $CT_1(u)_k \leftarrow e$
13. FOREACH $v\in N_u^I$ DO
14. IF $u\in CN_v \wedge CORE[u]<CORE[v]$ then
15. delete u from CN_v
16. IF $|CN_v|<CORE[v]$ THEN $Q \leftarrow Q\cup\{v\}$
17. RETURN CT_1
18. Function ComputeCN(I, u)

```
19. FOREACH v∈N_u^I DO
20.   IF v∉CN_v ∧ CORE[v]≥CORE[u] THEN
21.     x_(u,v) ← |{t|(u,v,t)∈E∧t∈I}|
22.     CN_u[v] ← x_(u,v)
23.   RETURN CN_u[v]
24. Function DelEdgesInterval(e,Q)
25.   FOREACH (u,v)∈E_[e,e] DO
26.     IF v∈CN_u DO
27.       CN_u[v] ← CN_u[v]−1
28.       IF CN_u[v]=0 THEN delete v from CN_u
29.       IF |CN_u|<CORE[u] THEN Q←Q∪{u}
30.     IF u∈CN_v DO
31.       对 u 执行 27～29 行相同的操作
32. Function ComputeCore(I,u)
33.   cnt[1…CORE[u]−1] ← 0
34.   FOREACH v∈N_u^I DO
35.     IF CORE[v]<CORE[u] THEN
36.       cnt[CORE[v]] ← cnt[CORE[v]]+1
37.   sum ← |CN_u|
38.   FOREACH k from CORE[u]−1 to 0 DO
39.     sum ← sum+cnt[k]
40.     IF k≤sum THEN RETURN k
```

算法 5 中，函数 ComputeCN(I,u)的作用是根据定义 7 计算顶点 u 关于时间区间 I 的核邻居并统计 u 与每个核邻居之间时态边的数目，其中第 19 行的循环用于遍历快照图 G_I 中 u 的每个邻居 v，第 20 行根据定义 7 判断 v 是否为 u 的区间核邻居，如果 v 是 u 的区间核邻居且 v 没有记录在 u 的区间核邻居哈希表 CN_u 中，则在第 21 行统计 u 和 v 之间时态边的数目，并在第 22 行将其加入 CN_u 中.

函数 DelEdgesInterval()用于在快照图 $G_{[1,e]}$ 上删除时刻为 e 的边，在第 26～29(30～33)行更新顶点 $u(v)$的区间核邻居 $CN_u(CN_v)$并判断顶点 $u(v)$的核数是否改变，并将改变的顶点加入集合 Q 予以返回.

函数 ComputeCore()的作用是更新顶点 u 的核数，其理论依据是：顶点 u 的核数是最大的 k 值，满足 u 的邻居中至少存在 k 个邻居 v 且 v 的核数大于等于 k[44].

阶段 3. 求解其它起始时刻所有顶点在所有 k 值对应的最短 k 核区间.

具体求解方法是，按照时间从小到大的顺序，依次删除每个时刻 s 的边，同时更新被删边两端顶点的核时间，从而得到每个顶点 u 在时刻 $s+1$ 的核时间，对于给定的 k 值，顶点 u 在 s 时刻和 $s+1$ 时刻的核时间可能相同也可能不同，当不同时，则表示得到了 u 的一个最短 k 核区间$[s,CT_s(u)_k]$.

需要注意的是，如果删除时刻为 s 的边后，u 在 $G_{[s+1,e]}$ 的邻居中核时间小于等于 u 的邻居少于 k 个，那么删边后 u 的核时间一定发生变化，且大于 e，为了高效处理顶点核时间的变化，需要引入 k 核时间邻居的概念，以便记录删边前后的复杂状态信息.

定义 8. k 核时间邻居(k-Core Time Neighbors). 给定时态图 G，顶点 u，整数 k 和时刻 s，若 u 在快照图 $G_{[s,CT_s(u)_k]}$ 的邻居 v 满足 $CT_s(v)_k \leq CT_s(u)_k$，则称 v 是 u 在时刻 s 的 k 核时间邻居.

根据 k 核时间邻居的定义，u 在 s 时刻的 k 核时间邻居就是 u 在 G_I 里的 k-core 子图中的邻居，假设 $x_{(u,v)}=|\{t|(u,v,t)\in E\wedge t\in I\}|$ 表示边(u,v)在时间段 $I=[s,CT_s(u)_k]$中出现的次数，我们用哈希表 $CTN_{(u,k)}$ 记录 u 的每个 k 核时间邻居 v 到 $x_{(u,v)}$ 的映射关系，$CTN_{(u,k)}[v]=x_{(u,v)}$.

算法 6 为计算 $s(s\neq 1)$时刻所有顶点在所有 k 值的最短 k 核区间的伪代码，第 1 行对核数组 $CORE$ 和集合 Q 进行初始化，第 2～4 行对核时间数组 CT 进行初始化并计算顶点在 1 时刻的 k 核时间邻居，第 5～16 的作用是以从小到大的时间顺序删除边，并在此过程中求得所有最短 k 核区间，具体来说，第 6 行求得需要重新计算核时间的点 u 及对应的 k 值，并暂存在集合 Q 中，第 7～16 行依次处理 Q 中的元素(u,k)，并求 u 在 s 的核时间、最短 k 核区间和 k 核时间邻居，其中，第 9～10 行求 u 的最短 k 核区间，第 11 行计算 u 在 s 的核时间，第 12 行更新 u 在 s 的 k 核时间邻居，第 13～16 行将需要重新计算核时间的顶点及 k 值加入 Q 中，最后在第 17 行返回最终构建的 SIHC 索引.

算法 6 中，函数 ComputeCTN(s,u,k)的作用是根据定义 8 计算顶点 u 在快照图 $G_{[s,CT_s(u)_k]}$ 中的 k 核时间邻居并统计 u 与每个核邻居之间时态边的数目，其中第 19 行的循环用于遍历快照图 $G_{[s,CT_s(u)k]}$ 中 u 的每个邻居 v，第 20 行根据定义 7 判断 v 是否为 u 的 k 核时间邻居，如果 v 是 u 的 k 核时间邻居且 v 没有记录在 u 的 k 核时间邻居哈希表 $CTN_{(u,k)}$ 中，则在第 21 行统计 u 和 v 之间时态边的数目并将其加入 $CTN_{(u,k)}$ 中.

算法 6. ComputeSI.

```
输入：G, core_[1,t_max], CT_1
输出：SI
1.  CORE ← core_[1,t_max], Q ← ∅
2.  FOREACH ∀u∈V∧k∈CORE[u] DO
3.    CT[u][k] ← CT_1(u)_k
4.    CTN_(u,k) ← ComputeCTN(1,u,k)
5.  FOREACH s: from 2 to t_max DO
```

```
6.  DelEdgesTime (s−1,Q)
7.  WHILE Q≠∅ DO
8.    (u,k)←remove a node from Q
9.    I←[s−1,CT[u][k]]
10.   SI_k←SI_k∪{I},V_I←V_I∪{u}
11.   CT[u][k]←ComputeCT(s,u,k)
12.   CTN_(u,k)←ComputeCTN(s,u,k)
13.   FOREACH v∈N_u^[s,t_max] DO
14.     IF u∈CTN_(v,k)∧CT[v][k]<CT[u][k]
        THEN
15.       delete u from CTN_(v,k)
16.       IF |CTN_(v,k)|<k THEN Q←Q∪{(v,k)}
17.   Return SI
18. Function ComputeCTN(s,u,k)
19.   FOREACH v∈N_u^[s,CT[u][k]] DO
20.     IF v∉CTN_(u,k) ∧CT[v][k]≤CT[u][k]
        THEN
21.       CTN_(u,k)[v]←|{t|(u,v,t)∈E_[s,CT[u][k]]}|
22.   RETURN CTN_(u,k)
23. Function DelEdgesTime(s,Q)
24.   FOREACH (u,v)∈E_[s,s] DO
25.     FOREACH k∈[1,min(CORE[u],CORE[v])
        DO
26.       IF v∈CTN_(u,k) DO
27.         CTN_(u,k)[v]←CTN_(u,k)[v]−1
28.         IF CTN_(u,k)[v]=0 THEN
29.           delete v from CTN_(u,k)
30.           IF |CTN_(u,k)|<k THEN Q←Q∪{(u,k)}
31.       IF u∈CTN_(v,k) THEN
32.         对 u 执行 27～30 行相同的操作
33. Function ComputeCT(s,u,k)
34.   T←∅
35.   FOREACH v∈N_u^[s,t_max] DO
36.     t←the first time occurrence of (u,v) in [s,t_max]
37.     T←T∪{max(t,CT[v][k])}
38.   IF |T|≥k THEN RETURN k-th smallest value
      in T
39.   CORE[u]←min(CORE[u],k−1)
40.   RETURN t_max+1
```

函数 DelEdgesTime()的作用是在快照图 $G[s,t_{\max}]$上删除时刻 s 的边，返回核时间改变的顶点及对应的 k 值，并设置顶点在 $s+1$ 时刻的 k 核时间邻居，

函数 ComputeCT()的作用是更新顶点 u 的核时间，其计算依据是：u 的核时间 $CT_s(u)_k$ 是最小的 t，满足 u 在 $G[s,t]$内的邻居中至少存在 k 个邻居 v 且 v 的核时间 $CT_s(u)_k$ 小于等于 t[17].

定理 9.　给定时态图 G，算法 4 的时间复杂度为 $O(|E_{[1,t_{\max}]}|\times\bar{t}\times d_{\max})$，空间复杂度为 $O(k_{\max}\times|E_{[1,t_{\max}]}|)$.

5　实　验

本文用于实验的台式机配置为 Intel Core 1.80 GHz CPU，24 GB 内存，操作系统为 Ubuntu 16.04.1，用于比较的算法有四个：

(1) 基本的在线求解算法 Online. 该方法首先得到查询区间对应的快照图，然后在快照图上求解相应的 k-core 子图；

(2) TCD 算法[25]. 文献[25]提出了两种算法来求解给定查询区间 I 所有子区间快照图中的 k-core 子图，分别称为 TCD 和优化的算法 OTCD，二者都是从最大区间开始，逐步缩减区间进行求解，因此，其第一个输出的结果是最大区间 I 的快照图中的 k-core 子图，这和本文的输出结果一致，OTCD 相较于 TCD 的优化体现在最大区间快照图之后的计算，在求解本文问题时，OTCD 和 TCD 的做法相同，因此在本文实验中，我们统一用 TCD 代表该文献的方法在求解本文查询问题时的名字，并在返回第一个结果后立即终止执行，以便进行公平的比较.

(3) 基于 PHC 索引的算法[17]. 用 PHC 表示.

(4) 本文提出的基于 SIHC 索引的算法，用 SIHC 表示，当比较索引规模和构建时间时，分别用 SIHC-B 和 SIHC-O 表示基础算法(算法 3)和优化算法(算法 4).

PHC 和 Online 的源代码由文献[17]的作者提供，TCD 的源码从网上下载①，所有的算法都使用 C/C++实现，并使用 g++ 9.4.0 进行编译.

本文算法是对文献[17]算法的改进，因而使用和文献[17]相同的 8 个真实世界的时态网络数据集进行实验，包括 MathOverflow、Ask-Ubuntu、SuperUser、WikiTalk、Youtube、DBLP、Flickr、Wikipedia，前四个数据集来自 SNAP②，其中 MathOverflow、AskUbuntu、SuperUser 分别代表 Math Overflow、Ask Ubuntu 以及 Super User 三个网站上表示用户评论、提问和回答的时态图；WikiTalk 是表示维基百科上用户互相编辑会话页的时态图，后四个数据集来自 KONECT③，其中 Youtube 和 Flickr 分别表示 YouTube 和 Flickr 网站上用户及其好友关系的社交网络时态图；DBLP 是表示 DBLP 网站上科技论文作者之间协作关系的时态

① https://github.com/graphlab-whu/Temporal-k-Core-Query-Project
② http://snap.stanford.edu/
③ http://konect.cc/

图;Wikipedia 是表示维基百科网页之间超链接的时态图,表 3 展示了 8 个数据集的统计信息,其中 $|E_{[1,t_{\max}]}|$ 是快照图 $G_{[1,t_{\max}]}$ 中边的数量,$deg_{[1,t_{\max}]}$ 是时态图在去掉时态信息后所得快照图 $G_{[1,t_{\max}]}$ 中顶点的平均度 $d_{\text{avg}}=|E_{[1,t_{\max}]}|\times 2/|V|$,$d_{\max}$ 是时态图中顶点的最大度,$k_{\max}$ 是 $G_{[1,t_{\max}]}$ 的最大 k 值,$t_{\max}$ 是图中不同时间戳的数量,$\bar{t}$ 是 PHC 索引中每个顶点不同 k 值对应有效/最短 k 核区间数量的平均值,$|\bar{V}_I|$ 是 SIHC 索引中每个最短 k 核区间对应顶点集大小的平均值.

表 3 数据集

Datasets	$\|V\|$	$\|E\|$	$\|E_{[1,t_{\max}]}\|$	$deg_{[1,t_{\max}]}$	$d_{\max}$	$k_{\max}$	$t_{\max}$	$\bar{t}$	$\|\bar{V}\|$
MathOverflow	24818	390441	187986	15.1	7421	78	390157	104.93	4.65
AskUbuntu	159316	726661	455691	5.7	10122	48	725568	30.17	3.62
SuperUser	194085	1108739	714570	7.4	27183	61	1106768	41.94	3.70
WikiTalk	1140149	6100538	2787967	4.9	233954	124	6088535	97.09	5.49
Youtube	3223589	9375374	9375374	5.8	91751	88	5201409	31.27	3.83
DBLP	1824701	29487744	8344615	9.1	7276	286	2612156	9.10	4.59
Flickr	2302926	33140017	22838276	19.8	34174	600	134	8.89	102.45
Wikipedia	1870710	39953145	36532531	39.1	226577	206	2198	35.33	20.47

5.1 查询效率

查询参数包括查询区间和 k 值,其中查询区间的大小是指查询区间的宽度占 $t_{\max}$ 的比例,用 $|I|$ 表示,k 值大小取 $t_{\max}$ 的特定比例,我们将查询区间的大小变化设置为 10%、20%、30%、40%、50%,默认为 30%,k 值的大小变化设置为 10%、30%、50%、70%、90%,默认为 50%,这么设置的原因是,时态图中边上的整体时间跨度较大,相比较而言用户需要了解的通常是很小区间的 k-core 结果,例如,DBLP 数据集中包含了超过 50 年的文献出版信息,但研究热点或作者之间的合作关系相对于整体时间跨度来说,通常在相对较短的时间内会发生变化,例如 5 年左右(仅占整体时间跨度的 10%),因而我们在实验中将时间跨度最大值设为 50%,对于 k 而言,用户通常更关心最密集的 k-core 子图.k 值越大,代表相应的结果中顶点之间的联系越紧密,因此,实验中 k 的最大值取到 90%,对每种组合情况,我们随机生成 1000 个查询,并展示其平均运行时间.

5.1.1 k 值变化时的查询性能比较

图 6 展示了 8 个数据集上四种算法随 k 变化的查询时间,区间大小是默认值 30%,根据图 6 有如下观察:(1)在所有的数据集上,SIHC 查询效率都比 PHC 高,且两个算法的查询时间都随着 k 值的增大而减少,两种方法时间都减少的原因是查询处理的数据量随着 k 值的增大而减少,因此两种方法的用时均减少,二者的性能变化可参考表 4 第 2、3、5、6 列的数据,这四列数据展示的是两种算法在 $k=10\%$,$|I|=30\%$ 以及 $k=90\%$,$|I|=30\%$ 两种极端情况下所处理的基本数据量,可以看出,随着 k 值的增大,二者处理的数据量都明显变少;(2)随着 k 值

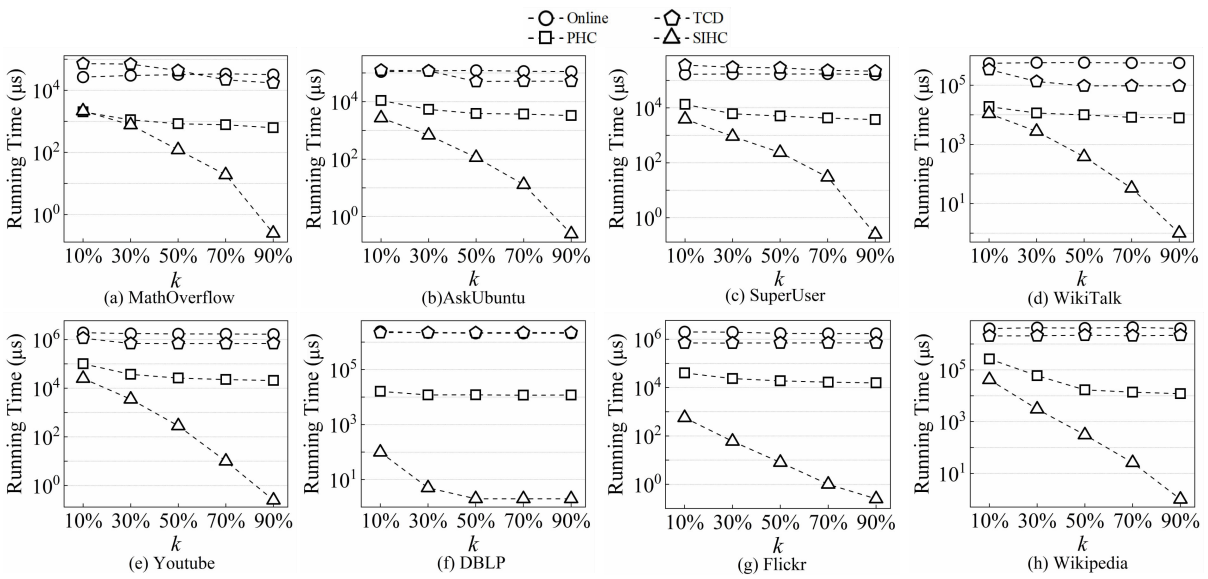

图 6 不同 k 值下的查询时间,$|I|=30\%$

增大，SIHC 方法的查询用时迅速减少，远远少于 PHC 算法，这是因为随着 k 值的增大，满足查询区间的最短 k 核区间数量迅速减少，算法可以在极短的时间内得知结果集并返回，与之对应，PHC 算法需要检测快照图 $G_{[1,t_{\max}]}$ 的 k 核子图中的所有顶点，二者的性能差异可参考表 4 第 2～7 列的数据；(3) 由于使用了索引，SIHC 和 PHC 的性能均远好于 Online 和 TCD，需要注意的是，TCD 和 Online 都是首先得到快照图，然后得到快照图上的 k-core 子图，因而二者具有相同的查询复杂度，从图 6 可以看出，二者的查询性能差异较少.

5.1.2　查询区间变化时的查询性能比较

图 7 展示了四个算法在 8 个数据集上随着查询区间变化时的查询时间对比（k 值大小是默认值 50%），根据图 7 我们有如下观察：(1) PHC 算法运行时间基本稳定，原因在于 PHC 查询时间只与 $G_{[1,t_{\max}]}$ 中 k-core 子图规模和每个点对应的平均有效 k 核区间的数量有关，和查询区间的大小无关；(2) 随着区间不断增大，Online、TCD 以及 SIHC 的运行时间均有所增长，原因是当区间增大时，Online、TCD 需要处理的快照图中的边数增多，同时，由于满足条件的 k-core 子图变大，SIHC 算法需要访问的顶点数量也增多；(3) 由于 PHC 和 SIHC 使用了索引，因而二者均比 Online 和 TCD 快；(4) 和 PHC 相比，SIHC 在所有数据集上都更快，原因是随着区间的增大，尽管 SIHC 访问的数据量有所增加，但仍远少于 PHC 处理的数据量，PHC 和 SIHC 的性能差异可参考表 4 第 8～13 列的数据.

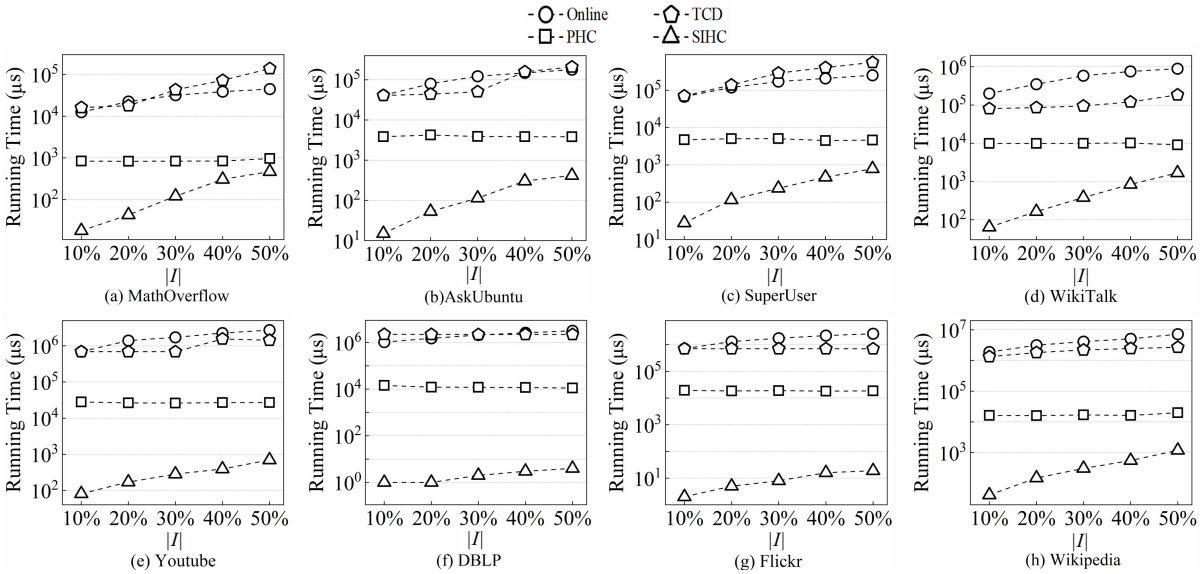

图 7　不同区间宽度下的查询时间，$k=50\%$

表 4　不同算法处理的基本数据量对比　　（单位：个）

Datasets	$k=10\%, \lvert I\rvert=30\%$			$k=90\%, \lvert I\rvert=30\%$			$k=50\%, \lvert I\rvert=10\%$			$k=50\%, \lvert I\rvert=50\%$		
	PHC	SIHC	PHC/SIHC	PHC	SIHC	PHC/SIHC	PHC	SIHC	PHC/SIHC	PHC	SIHC	PHC/SIHC
MathOverflow	688162	177954	3.9	14771	0	—	160100	0	—	160100	44378	3.6
AskUbuntu	716851	160658	4.5	11862	0	—	128049	2098	61.0	128049	37990	3.4
SuperUser	1121427	237195	4.7	24611	0	—	255081	381	669.3	255081	67211	3.8
WikiTalk	3258159	854763	3.8	124493	0	—	1072881	0	—	1072881	189277	5.7
Youtube	6663267	1357740	4.9	70019	0	—	1470219	2877	510.9	1470219	75145	19.6
DBLP	271084	5550	48.8	1382	11	125.6	1382	11	125.6	1382	372	3.7
Flickr	902721	64979	13.9	5782	27	213.0	116873	35	3295.9	116873	2054	56.9
Wikipedia	25146933	3766594	6.7	46164	0	—	635701	0	—	635701	108841	5.8

注：表中“—”表示 SIHC 没有实际访问顶点.

在整体上比较分析图 6 和图 7 的结果后，下面解释 MathOverflow 和 DBLP 两个数据集上出现的特殊现象.

对于 MathOverflow 数据集，当 $k=10\%, \lvert I\rvert=30\%$ 时（下文用 Case-1 表示），PHC 处理的数据量是 SIHC 的 3.9 倍（表 4），对应图 6(a) x 轴 $k=10\%$ 的查询时间显示，二者的查询时间相差不明显，造成这一现象的原因如第 4.2 节最后一段所述：SIHC

索引将不同 k 值的最短 k 核区间按照起始值进行了排序，同时维护了从每个最短 k 核区间$[s,e]$到起始值大于 s 的第一个最短 k 核区间的指针，查询处理时，如果以 s 为起始值的区间不是查询区间的子区间，则需要借助指针找到起始值比 s 大的第一个区间，即在使用 SIHC 索引处理查询的过程中，数据并非顺序扫描的，存在内存数据随机存取的问题，这会一定程度上造成 CPU cache 中数据存取时命中率下降的问题，而 PHC 索引需要处理快照图 $G_{[1,t_{\max}]}$ 中 k-core 子图的所有顶点，数据以顺序扫描的方式进行处理，这和提升命中率的预取机制吻合，在被处理数据量相同时，PHC 效率更高，因而造成 Case-1 情况下 PHC 处理的数据量是 SIHC 的 3.9 倍，但二者查询时间相差不明显的现象.

当 $k=50\%$，$|I|=50\%$时(下文用 Case-2 表示)，PHC 处理的数据量是 SIHC 的 3.6 倍，对应图 7(a) x 轴 $|I|=50\%$的查询时间显示，二者的查询时间看起来也不太明显，原因是 y 轴的查询时间是 log 方式展示的，实际上 PHC 所需的运行时间是 SIHC 的两倍.

进一步，对于两种情况下二者访问数据量的比值近似，但查询时间的比值相差 2 倍的问题，我们统计了 SIHC 查询处理过程中访问的区间数量，对 Case-1，SIHC 处理的区间数量是 36 357，而对 Case-2，SIHC 处理的区间数量是 5091，结合表 4 的统计数据可知以下几点事实：(1) 前后两种参数配置下，PHC 处理的数据量下降了 77%，SIHC 处理的数据量下降了 75%；(2) 前后两种参数配置下，SIHC 处理的区间数量下降了 86%；(3) 当 Case-1 时，SIHC 每个区间对应的顶点数量是 4.9(=177 954/36 357)，而当 Case-2 时，SIHC 每个区间对应的顶点数量是 8.7(=44 378/5091).

从第(1)点来看，二者的查询时间差不应变化太大，但从第(2)点来看，SIHC 处理的“区间”数量比其处理的数据量下降得更多，意味着等概率存取数据时，Case-2 的 cache 命中率更高，从第(3)点来看，Case-2 时每个区间对应的顶点数量更多，更适合命中率高情况下的预取机制，结合以上 3 点来看，Case-2 时查询性能提升的比率大于Case-1 是合理的.

尽管 SIHC 访问的数据量比 PHC 少，由于 SIHC 存在 cache 命中率低的问题，因此无论对于哪种情况，SIHC 相对于 PHC 运行时间加速比都小于访问数据量的比值，这里需要说明的是，虽然命中率会影响查询性能，且对不同查询而言，命中率会因被处理数据的分布不同而有所不同，进而对不同查询的处理时间也会产生不同程度的影响，但命中率不是影响查询性能的主要指标，对查询处理而言，影响其性能的主要指标是查询处理过程中访问的数据量，这可从查询处理的时间复杂度以及图 6 和图 7 中多个数据集的整体变化趋势上得以印证.

对于 DBLP 数据集，SIHC 在图 6 的变化曲线表现出了与其他数据集不同的趋势，当 $I=30\%$，k 值从 10%变为 20%时，SIHC 访问的数据量减少了 90%多，之后 SIHC 访问的数据量下降率小于 80%或基本不变，相比较而言，SIHC 访问的数据量在其他数据集上的变化率通常在 10 倍左右，因而体现在图 6 上，SIHC 在 DBLP 上的变化曲线和其他数据集有所不同.

5.1.3 不同数据集上查询性能比较

从图 6 可以看出，在不同数据集上，SIHC 相对于 PHC 的加速比不同，例如，当 $k=10\%$，$|I|=30\%$时，WikiTalk 上二者基本相同，但 DBLP 和 Flickr 上 SIHC 比 PHC 快很多，造成这种现象的原因有三点：(1) 基于 PHC 索引的查询方法通过检测每个顶点的有效 k 核区间来判断该顶点的可满足性，因此影响其查询性能的主要指标是查询处理的区间数量，与之对应，基于 SIHC 索引的查询方法直接找到被查询区间包含的最短 k 核区间，之后从与之对应的顶点集中输出顶点即可，因此影响其查询性能的主要指标是访问的顶点数量，表 4 是图 6 和图 7 中 8 个数据集在四种极端情况下访问的基本数据量的对比，从表 4 可以看出，当 $k=10\%$，$I=30\%$时，对 WikiTalk 而言，PHC 访问的数据量是 SIHC 的 3.8 倍；而对于 DBLP 和 Flickr 而言，PHC 访问的数据量分别是 SIHC 的 48.8 倍和 13.9 倍，因而在 WikiTalk 上二者查询性能差别不大，但在 DBLP 和 Flickr 上，性能差异较大；(2) PHC 处理的基本数据是时间区间，包含两个数字，处理时需要比较两次才能确定其是否满足条件，而 SIHC 处理的基本数据是单个顶点，仅包含一个数字，而且无需任何处理，直接输出，因而 SIHC 的性能优势可能在数据量差异的基础上进一步得到放大；(3) 如表 9 所示，SIHC 的索引规模比 PHC 小，例如在 Flickr 数据集上，SIHC 索引还不到 PHC 索引的一半，这会进一步放大基于 SIHC 索引进行查询的性能优势.

另外，从表 4 可以看出，当 k 值增大到 90%或者区间 $|I|$ 变小到 10%时，对应的数据图中可能不

存在相应的 k-core 子图，这时，基于 SIHC 索引进行查询时，索引中不存在被包含的最短区间，这种情况下不需要访问任何顶点，即访问的数据量为 0，与之相比，基于 PHC 索引处理查询时，即使没有满足条件的结果，所需访问的数据量也可能很大.

图 7 中 8 个数据集上两种极端情况下的性能差异也可用表 4 的数据进行辅助解释，原因和图 6 上性能差异的原因类似，这里不再赘述.

以上展示并分析了几种算法在查询区间宽度，k 值和数据集变化情况下的主要特点，针对不同区间宽度，不同 k 值和不同数据集取值组合多的问题，我们在表 5 至表 8 进一步展示当区间宽度 $|I|=10\%$、20%、40%、50%四种情况下 PHC 和 SIHC 两种算法查询时间的全部对比数据，供读者做进一步参考，注意 $|I|=30\%$的数据已通过图 6 予以展示，这里不再赘述.

表 5　PHC 和 SIHC 算法在 $|I|=10\%$时不同 k 值对应的查询时间对比　（单位：μs）

k	MathOverflow		AskUbuntu		SuperUser		WikiTalk		Youtube		DBLP		Flickr		Wikipedia	
	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC
10%	2458	571	11648	765	119927	9512	16981	3201	92559	6357	15519	33	40747	43	256384	5780
30%	1311	66	4928	97	52762	1456	10637	392	36139	723	10652	2	23590	13	53916	436
50%	872	17	3909	14	42101	322	8930	90	25708	69	10361	1	19631	3	16261	42
70%	676	6	4081	4	38650	71	8042	27	21171	4	10882	0	16766	0	12335	6
90%	627	0	3258	0	35728	7	7534	1	20076	0	11094	0	14598	0	11370	0

表 6　PHC 和 SIHC 算法在 $|I|=20\%$时不同 k 值对应的查询时间对比　（单位：μs）

k	MathOverflow		AskUbuntu		SuperUser		WikiTalk		Youtube		DBLP		Flickr		Wikipedia	
	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC
10%	3023	1382	12294	1662	125701	37956	16969	5832	107326	13589	16207	69	44794	264	242810	23769
30%	1137	315	4998	263	57742	9094	10726	1025	37067	1836	11481	3	25219	33	51164	1293
50%	941	42	3888	63	47703	2265	8813	124	26022	178	11179	1	18926	5	16053	150
70%	751	10	3450	7	42380	296	7985	24	21325	9	11221	1	17272	0	11843	13
90%	609	0	3271	0	38218	8	7401	1	19674	0	11988	1	14916	0	10924	1

表 7　PHC 和 SIHC 算法在 $|I|=40\%$时不同 k 值对应的查询时间对比　（单位：μs）

k	MathOverflow		AskUbuntu		SuperUser		WikiTalk		Youtube		DBLP		Flickr		Wikipedia	
	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC
10%	2036	3147	11878	3998	124960	54798	17109	13752	93914	29915	16492	128	58514	1105	256971	74109
30%	1069	982	4637	788	57959	15284	10916	4208	37145	5271	11552	8	24481	108	55112	5872
50%	893	340	3746	265	46455	5329	9076	789	25855	451	11437	3	18778	14	16311	555
70%	847	27	3448	20	41005	592	7930	64	21360	19	25890	3	16995	1	12483	73
90%	708	0	3213	0	41196	10	7382	2	19682	0	10829	3	14671	0	11464	1

表 8　PHC 和 SIHC 算法在 $|I|=50\%$时不同 k 值对应的查询时间对比　（单位：μs）

k	MathOverflow		AskUbuntu		SuperUser		WikiTalk		Youtube		DBLP		Flickr		Wikipedia	
	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC	PHC	SIHC
10%	2357	3753	11848	5303	148967	73006	17006	17345	94507	47291	16296	161	47882	1802	278917	80581
30%	1096	1377	4789	1117	68076	22918	10766	6004	37209	7224	11321	9	27918	218	63291	11469
50%	937	453	3712	425	51090	8727	8941	1556	27571	654	11039	4	18931	21	19948	1194
70%	733	80	3484	64	41117	1662	8112	179	21389	21	11127	3	16857	1	14152	192
90%	591	0	3281	1	38169	11	7404	2	19825	0	11040	3	14602	0	11658	1

5.2　索引构建

5.2.1　索引规模

表 9 中 SIHC、PHC 列展示了 SIHC 索引，PHC 索引的大小，单位为 MB，可以看出，在所有数据集上，SIHC 索引的规模都比 PHC 索引小，在 Flickr 上，SIHC 索引比 PHC 索引的一半还要小，原因在于，SIHC 索引存储的顶点数量与 PHC 索引中存储的区间数量一致，但 SIHC 存储顶点只需要存储一个

表 9　索引规模

Datasets	SIHC	PHC
MathOverflow	110.16	135.58
AskUbuntu	62.91	73.14
SuperUser	151.16	173.15
WikiTalk	972.88	1275.94
Youtube	1311.32	1530.79
DBLP	411.67	566.26
Flickr	660.28	1454.49
Wikipedia	5243.63	9420.54

值，而 PHC 存储区间需要存储两个值，另外，PHC 索引中还存储了以 $t_{max}+1$ 为结束值的区间，这种区间在 SIHC 索引中没有对应的最短 k 核区间.

此外，从理论角度来说，SIHC 和 PHC 两种索引的规模均是 $O(\bar{t}\times|E_{[1,t_{max}]}|)$，其占用的空间不但和图中的边数量相关，而且和不同 k 值对应的有效/最短 k 核区间个数的平均值 $\bar{t}$ 相关，由表 3 可知，不同数据图上 $\bar{t}$ 值和数据集本身承载的信息相关，并非常量，因此索引大小相对图规模来说并非线性相关，但在一些实际应用中，查询区间可能会得到适当程度的简化，例如，我们可能只需要支持带有明确语义的时间区间，比如在以日期为时间戳的时态图中，我们可能只对按月份或者自然年的查询区间感兴趣，这种情况下，SIHC 和 PHC 索引仍然可以正常工作，但将日期转化为月份或者自然年后，索引规模将得以显著降低.

5.2.2 索引构建

表 10 展示的是不同方法构建索引的时间和最大内存消耗，从表 10 可以看出，SIHC-B 算法的运行时间比构建 PHC 索引的算法运行时间长一点，但相差不大，原因在于，在 PHC 索引的基础上构建 SIHC 索引，只需要将索引中的数据项顺序扫描一遍即可，优化算法 SIHC-O 运行时间比基础算法 SIHC-B 运行时间快，是因为 SIHC-O 无需计算出完整的 PHC 索引.

表 10 索引构建时间 （单位：s）

Datasets	PHC		SIHC-B		SIHC-O	
	Time	Peek Memory/MB	Time	Peek Memory/MB	Time	Peek Memory/MB
MathOverflow	1429	245.8	1439	361.2	948	260.3
AskUbuntu	519	266.0	524	266.0	360	429.3
SuperUser	1612	464.2	1625	511.2	1031	678.1
WikiTalk	46409	2554.4	46519	3270.1	28568	3148.3
Youtube	22816	3948.8	22949	4230.8	14067	5487.7
DBLP	4616	2786.5	4658	2786.6	3086	6424.5
Flickr	61747	5335.7	61871	5335.7	40231	11413.5
Wikipedia	251536	11249.4	252443	18340.6	122506	11253.4

同时，由于 SIHC-B 算法在 PHC 索引构建完成的基础上构建 SIHC 索引，因此内存占用会适度大于 PHC 算法，而 SIHC-O 算法在构建 SIHC 索引时，需要缓存的数据量也略多于 PHC，因此内存开销同样会略大于 PHC，和表 9 中的索引大小相比，由于索引构建过程中需要缓存中间结果，因此在表 10 中，三种算法在索引构建过程中的内存峰值会比索引规模大.

另外，由表 3 可知 Youtube 和 DBLP 两个数据集的 $|E_{[1,t_{max}]}|$ 值相差不大，但根据表 10，二者在索引构建时间方面差别较大，其原因在于两个方面：首先，由定理 7 及定理 9 可知，PHC、SIHC-B 以及 SIHC-O 的时间复杂度都是 $O(|E_{[1,t_{max}]}|\times\bar{t}\times d_{max})$，即影响这三个算法运行时间的因素除了 $|E_{[1,t_{max}]}|$，还有 $\bar{t}$ 以及 d_{max}，其次，虽然 Youtube 和 DBLP 的 $|E_{[1,t_{max}]}|$ 相差不大，但根据表 3，二者的另外两个参数值相差较大，例如，对 Youtube 来说，$\bar{t}$ 和 d_{max} 的值分别是 31 和 91751，而对 DBLP 来说，这两个值分别是 9 和 7276，均远小于 Youtube 的相应数值.

6 结 论

针对现有方法在时态图中进行 k-core 查询时的低效性问题，本文对其工作机理进行了深入分析，得出基于现有索引的区间无法构建从区间到顶点的倒排表来加速查询处理的结论. 进一步，本文证明了基于最短 k 核区间处理时态图中 k-core 查询的正确性，并提出了一种基于最短 k 核区间的高效索引 SIHC. SIHC 索引通过存储最短 k 核区间到顶点的映射关系，查询处理时，可根据用户给定的时间区间轻松确定 SIHC 索引中的子区间，进而基于这些子区间得到相应倒排表中的顶点，并将其作为结果直接返回，避免了现有方法需要执行大量无用顶点检测的问题. 基于真实数据集的实验表明，本文提出的 SIHC 索引在具备同等索引构建效率及更小索引规模的前提下，比现有算法快 1～2 个数量级，能够显著提高时态图中 k-core 子图查询的效率.

最后，对于本文提出的 SIHC 索引和文献[17]提出的 PHC 索引来说，构建索引过程中使用的内存空间及索引规模不但和图本身的规模相关，而且受制于不同 k 值对应的有效/最短 k 核区间个数的平均值 $\bar{t}$，由于不同数据图上 $\bar{t}$ 值相对图规模来说并非线性相关，因此可能造成索引规模过大的问题. 我们计划未来对该问题进行深入研究，从数据压缩和索引设计的角度同时考虑，以期降低索引规模，从而得

以在内存受限的情况下支持更大规模时态图上的高效 k-core 查询.

参 考 文 献

[1] Yu Dingjiu, Zhao Yu. Information diffusion model based on k-core for social networks//Proceedings of the 7th International Conference on Internet Multimedia Computing and Service. Zhangjiajie, China, 2015: 11:1-11:6

[2] Moradi-Jamei B, Kramer B L, Calderon J B S, Korkmaz G. Community formation and detection on GitHub collaboration networks//Proceedings of the ASONAM21: International Conference on Advances in Social Networks Analysis and Mining. Virtual Event, The Netherlands, 2021: 244-251

[3] Sariyüce A E, Gedik B, Jacques-Silva G, et al. Incremental k-core decomposition: Algorithms and evaluation. The VLDB Journal, 2016, 25(3): 425-447

[4] Chen Zi, Yuan Long, Han Li, Qian Zhengping. Higher-order truss decomposition in graphs. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(4): 3966-3978

[5] Yu Dongxiao, Zhang Lifang, Luo Qi, et al. Maximal clique search in weighted graphs. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(9): 9421-9432

[6] Chang Lijun, Yu Jeffrey Xu, Qin Lu, et al. Efficiently computing k-edge connected components via graph decomposition //Proceedings of the ACM SIGMOD International Conference on Management of Data. New York, USA, 2013: 205-216

[7] Nalam C, Saranurak T. Maximal k-edge-connected subgraphs in weighted graphs via local random contraction//Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms. Florence, Italy, 2023: 183-211

[8] Saranurak T, Yuan W. Maximal k-edge-connected subgraphs in almost-linear time for small k//Proceedings of the 31st Annual European Symposium on Algorithms. Amsterdam, The Netherlands, 2023: 92:1-92:9

[9] Shin K, Eliassi-Rad T, Faloutsos C. CoreScope: Graph mining using k-core analysis—Patterns, anomalies and algorithms// Proceedings of the IEEE 16th International Conference on Data Mining. Barcelona, Spain, 2016: 469-478

[10] Li Rong-Hua, Qin Lu, Ye Fanghua, et al. Skyline community search in multi-valued networks//Proceedings of the 2018 International Conference on Management of Data. Houston, USA, 2018: 457-472

[11] Li Rong-Hua, Su Jiao, Qin Lu, et al. Persistent community search in temporal networks//Proceedings of the 34th IEEE International Conference on Data Engineering. Paris, France, 2018: 797-808

[12] Yu Dongxiao, Zhang Lifang, Luo Qi, et al. Fast skyline community search in multi-valued networks. Big Data Mining and Analytics, 2020, 3(3): 171-180

[13] Chang Lijun. Efficient maximum clique computation and enumeration over large sparse graphs. The VLDB Journal, 2020, 29(5): 999-1022

[14] Li Rong-Hua, Dai Qiangqiang, Wang Guoren, et al. Improved algorithms for maximal clique search in uncertain networks// Proceedings of the 35th IEEE International Conference on Data Engineering. Macao, China, 2019: 1178-1189

[15] Wen Dong, Qin Lu, Zhang Ying, et al. Enumerating k-vertex connected components in large graphs//Proceedings of the 35th IEEE International Conference on Data Engineering. Macao, China, 2019: 52-63

[16] Galimberti E, Barrat A, Bonchi F, et al. Mining (maximal) span-cores from temporal networks//Proceedings of the 27th ACM International Conference on Information and Knowledge Management. Torino, Italy, 2018: 107-116

[17] Yu M, Wen Dong, Qin Lu, et al. On querying historical k-cores. Proceedings of the VLDB Endowment, 2021, 14(11): 2033-2045

[18] Ahmad A, Yuan Lyuheng, Yan Da, et al. Accelerating k-core decomposition by a GPU//Proceedings of the 39th IEEE International Conference on Data Engineering. Anaheim, USA, 2023: 1818-1831

[19] 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

[20] Zhang Yikai, Yu Jeffrey Xu, Zhang Ying, Qin Lu. A fast order-based approach for core maintenance//Proceedings of the 33rd IEEE International Conference on Data Engineering. San Diego, USA, 2017: 337-348

[21] Guo Bin, Sekerinski E. Parallel order-based core maintenance in dynamic graphs//Proceedings of the 52nd International Conference on Parallel Processing. Salt Lake City, USA, 2023: 122-131

[22] Wen Dong, Qin Lu, Zhang Ying, et al. I/O efficient core graph decomposition at web scale//Proceedings of the 32nd IEEE International Conference on Data Engineering. Helsinki, Finland, 2016: 133-144

[23] Aksu H, Canim M, Chang Yuan-Chi, et al. Distributed k-core view materialization and maintenance for large dynamic graphs. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(10): 2439-2452

[24] Luo Wensheng, Yang Qiaoyuan, Fang Yixiang, Zhou Xu. Efficient core maintenance in large bipartite graphs. Proceedings of the ACM on Management of Data, 2023, 1(3): 208:1-208:26

[25] Yang Junyong, Zhong Ming, Zhu Yuanyuan, et al. Scalable time-range k-core query on temporal graphs. Proceedings of the VLDB Endowment, 2023, 16(5): 1168-1180

[26] Wu Huanhuan, Cheng James, Lu Yi, et al. Core decomposition in large temporal graphs//Proceedings of the 2015 IEEE International Conference on Big Data. Santa Clara, USA, 2015: 649-658

[27] Bai Wen, Chen Yadi, Wu Di. Efficient temporal core maintenance of massive graphs. Information Sciences, 2020, 513: 324-340

[28] Ma Shuai, Hu Renjun, Wang Luoshu, et al. An efficient approach to finding dense temporal subgraphs. IEEE Transactions on Knowledge and Data Engineering, 2020, 32(4): 645-658

[29] Li Yuan, Liu Jinsheng, Zhao Huiqun, et al. Efficient continual cohesive subgraph search in large temporal graphs. World Wide Web, 2021, 24(5): 1483-1509

[30] Qin Hongchao, Li Ronghua, Yuan Ye, et al. Mining bursting core in large temporal graph. Proceedings of the VLDB Endowment, 2022, 15(13): 3911-3923

[31] Zhang Chen, Zhang Fan, Zhang Wenjie, et al. Exploring finer granularity within the cores: Efficient (k,p)-core computation //Proceedings of the 36th IEEE International Conference on Data Engineering. Dallas, USA, 2020: 181-192

[32] Zhang Fan, Zhang Ying, Qin Lu, et al. When engagement meets similarity: Efficient (k,r)-core computation on social networks. Proceedings of the VLDB Endowment, 2017, 10(10): 998-1009

[33] Lee Pei , Lakshmanan L V S, Milios E E. CAST: A context-aware story-teller for streaming social content//Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. Shanghai, China, 2014: 789-798

[34] Mishra N, Schreiber R, Stanton I, Tarjan R E. Finding strongly knit clusters in social networks. Internet Mathematics, 2008, 5(1): 155-174

[35] Kim J, Lim S. (p,n)-core: Core decomposition in signed networks//Proceedings of the 27th International Conference on Database Systems for Advanced Applications. Hyderabad, India, 2022: 543-551

[36] Wang Jia, Cheng James. Truss decomposition in massive networks. Proceedings of the VLDB Endowment, 2012, 5(9): 812-823

[37] Cheng James, Ke Yiping, Fu Ada Wai-Chee, et al. Finding maximal cliques in massive networks. ACM Transactions on Database Systems, 2011, 36(4): 21:1-21:34

[38] Zhou Rui, Liu Chengfei, Yu Jeffrey Xu, et al. Finding maximal k-edge-connected subgraphs from a large graph//Proceedings of the 15th International Conference on Extending Database Technology. Berlin, Germany, 2012: 480-491

[39] Dai Qiangqiang, Li Rong-Hua, Wang Guoren, et al. Core decomposition on uncertain graphs revisited. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(1): 196-210

[40] Chen Yankai, Zhang Jie, Fang Yixiang, et al. Efficient community search over large directed graph: An augmented index-based approach//Proceedings of the 29th International Joint Conference on Artificial Intelligence. Yokohama, Japan, 2020: 3544-3550

[41] Hua Zhengyu, Zhang Yihao, Yuan Long. Efficient distance generalized (α,β)-core decomposition on bipartite graphs//Proceedings of the ACM Turing Award Celebration Conference China 2023. Wuhan, China, 2023: 162-163

[42] Liu Qing, Liao Xuankun, Huang Xin, et al. Distributed (α,β)-core decomposition over bipartite graphs//Proceedings of the 39th IEEE International Conference on Data Engineering. Anaheim, USA, 2023: 909-921

[43] Arafat N A, Khan A, Rai A K, Ghosh B. Neighborhood-based hypergraph core decomposition. Proceedings of the VLDB Endowment, 2023, 16(9): 2061-2074

[44] Montresor A, De Pellegrini F, Miorandi D. Distributed k-core decomposition. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(2): 288-300

附录 1.

定理 1 证明. 我们从必要性和充分性两方面进行证明.

必要性. 根据定义 4,$[s',e']$是 u 的最短 k 核区间,则$[s',e']$也是 u 的 k 核区间,且$core_{[s',e']}(u)\geq k$. 同时,由于$[s',e']\subseteq[s,e]$,可知 $core_{[s,e]}(u)\geq core_{[s',e']}(u)$,因此 u 在 $G_{[s,e]}$ 中的 k 核子图中.

充分性. 假设对 u 来说,不存在被$[s,e]$包含的最短 k 核区间,则 $core_{[s,e]}(u)<k$,这与 u 在 k 核子图中矛盾,从而和假设矛盾. 得证. 证毕.

定理 2 证明. HC 索引包含任意顶点 u 的所有 k-core 区间. 这些 k-core 区间可以分为多个子集,每个子集 γ 中的所有 k-core 区间具有相同的结束值 e. 根据 k-core 区间定义,e 是使这些区间成为 k-core 区间的最小值. 从而可知不同子集中的 k-core 区间不存在包含关系. 同时,γ 中起始值最小的 k-core 区间是 γ 中唯一的有效 k-core 区间,γ 中起始值最大的 k-core 区间是其中唯一最短 k-core 区间. 因此,两种区间存在一一对应关系. 证毕.

定理 3 证明. 由定理 2 可知,对 PHC 索引中的每个“(顶点,有效 k-core 区间)”对,在 SIHC 索引中存在唯一对应的“(最短 k-core 区间,顶点)”对. 由于最短 k-core 区间总数为$\sum_{u\in V}\sum_{k\in[1,core_I(u)]}|SCI_k(u)|=\sum_{u\in V}\sum_{k\in[1,core_I(u)]}|ECI_k(u)|=\sum_{u\in V}core_I(u)\times\bar{t}\leq\bar{t}\times\sum_{u\in V}deg_I(u)=2|E_{[1,t_{\max}]}|\times\bar{t}$,其中 $I=[1,t_{\max}]$. 因此 SIHC 索引和 PHC 索引具有相同的索引规模 $O(\bar{t}\times|E_{[1,t_{\max}]}|)$. 证毕.

定理 4 证明. 对于 V_I 中任意顶点 u 来说,I 是 u 的最短 k 核区间. 从而,I 是 u 的 k 核区间,根据定义 4,$core_I(u)\geq k$,同时,由于 $I\subseteq[s,e]$,所以 $core_{[s,e]}(u)\geq k$,因此结论成立. 证毕.

定理 5 证明. 对给定时间区间 I 和 k,算法 2 根据 I 从

SIHC 索引中找到与 k 对应的所有最短区间，并从中找到被 I 包含的所有最短区间，最后返回这些最短区间对应顶点集合的并集作为查询结果. 根据定理 4，每个被 I 包含的最短区间对应的顶点集中的顶点都是 k 核子图 $C_k(G_{[s,e]})$ 中的顶点，因此算法 2 不存在返回错误结果(false-positive)的问题. 另外，根据最短 k 核区间的定义，如果一个顶点的最短 k 核区间集合中不存在被 I 包含的区间，则说明该顶点一定不属于 G_I 中 k-core 子图的顶点，因而算法 2 不存在漏解(false-negative)的问题. 证毕.

定理 6 证明.　首先证明 u 在 $s(s_1\leq s<s_2)$ 时刻的核时间等于 e_1. 如果 u 在 $s(s_1<s<s_2)$ 的核时间 $e'\neq e_1$，则 $[s,e']$ 是 k-core 区间. 因此必然存在一个以 e' 为结束值的有效 k-core 区间 $[s',e']$ 满足 $s_1<s'\leq s$. 故 $[s_1,e_1]$ 后面的相邻区间是 $[s',e']$. 又因为 $[s_2,e_2]$ 是 $[s_1,e_1]$ 后面的相邻区间，且 $[s',e']$ 和 $[s_2,e_2]$ 都是有效 k-core 区间，所以 $s'=s_2\wedge e'=e_2$. 从而得到矛盾 $s_2=s'\leq s<s_2$. 因此给定 k 值，u 在 $s(s_1\leq s<s_2)$ 时刻的核时间等于 e_1.

由于 $[s,e_1](s_1\leq s<s_2)$ 是 k-core 区间，且 s 的上界是 s_2-1，因此与 $[s_1,e_1]$ 对应的最短 k-core 区间为 $[s_2-1,e_1]$. 证毕.

定理 7 证明.　由定理 3 可知，PHC 索引的规模是 $O(\bar{t}\times|E_{[1,t_{\max}]}|)$，因此算法 3 在第 3～9 行扫描 PHC 索引的代价为 $O(\bar{t}\times|E_{[1,t_{\max}]}|)$. 在第 11～12 行，算法 3 基于计数排序方法对最短 k-core 区间排序，所需时间代价同样为 $O(\bar{t}\times|E_{[1,t_{\max}]}|)$. 同时，由文献[17]可知算法 3 在第 1 行计算 PHC 索引的代价为 $O(|E_{[1,t_{\max}]}|\times\bar{t}\times d_{\max})$. 因此，算法 3 构建 SIHC 索引的时间复杂度为 $O(|E_{[1,t_{\max}]}|\times\bar{t}\times d_{\max})$. 证毕.

定理 8 证明.　由定理 2 的证明可知，$CI_k(u)$ 可根据区间结束值得到一个划分 $S_1,S_2,\cdots,S_n$，同一子集 S_i 中的 k 核区间具有相同的结束值 e. 根据 k-core 区间的定义，e 是使得 S_i 中的区间满足 k-core 的最小值. 假设 $I=[s,e]$ 在 S_i 中起始值最大且不是最短 k-core 区间. 那么必然存在一个更短的 k-core 区间 $I'=[s',e]$ 满足 $s'>s$，即 $core_{[s',e]}(u)\geq k$. 因而 $I'\in S_i$. 这和假设矛盾. 因此，I 一定是 S_i 中的最短 k 核区间. 从而可知结论成立. 证毕.

定理 9 证明.　首先考虑时间复杂度. 算法 4 调用了算法 5 和算法 6. 对于算法 5，第 5 行删边的总时间为 $O(|E|)$，第 7～16 行更新顶点核数、区间核邻居和核时间时都需要遍历 u 的邻居 $|N_u^I|$，其时间代价为 $O(|N_u^{[1,t_{\max}]}|)$. 由于顶点 u 的核数最多改变 $core_{[1,t_{\max}]}(u)$ 次，算法 5 从 $G_{[1,t_{\max}]}$ 中依次移除最大时刻边的过程中，所有顶点被加入集合 Q 的次数等于第 7～16 行执行的次数，因此算法 5 第 7～16 行的总代价为 $O\left(\sum_{u\in V}core_{[1,t_{\max}]}(u)\times|N_u^{[1,t_{\max}]}|\right)\leq O\left(\sum_{u\in V}k_{\max}\times|N_u^{[1,t_{\max}]}|\right)=O(k_{\max}\times|E_{[1,t_{\max}]}|)$. 从而算法 5 的时间复杂度为 $O(k_{\max}\times|E_{[1,t_{\max}]}|)$. 对于算法 6，第 7～16 行更新顶点核时间、k 核时间邻居的时间代价为 $O(|N_u^{[1,t_{\max}]}|)$. 第 7～16 行执行次数等于最短 k 核区间总数，即 $\sum_{u\in V}\sum_{k\in[1,core_I(u)]}|SCI_k(u)|=\sum_{u\in V}\sum_{k\in[1,core_I(u)]}|ECI_k(u)|=\sum_{u\in V}core_I(u)\times\bar{t}\leq\bar{t}\times\sum_{u\in V}deg_I(u)=2|E_{[1,t_{\max}]}|\times\bar{t}$，其中 $I=[1,t_{\max}]$. 由于 $|N_u^{[1,t_{\max}]}|\leq d_{\max}$，算法 6 的时间复杂度为 $O(|E_{[1,t_{\max}]}|\times\bar{t}\times d_{\max})$. 二者相比较，算法 4 的时间复杂度为 $O(|E_{[1,t_{\max}]}|\times\bar{t}\times d_{\max})$.

其次考虑空间复杂度. 对于算法 5，保存核时间的空间代价为 $O\left(\sum_{u\in V}core_{[1,t_{\max}]}(u)\right)\leq O\left(\sum_{u\in V}deg_{[1,t_{\max}]}(u)\right)=O(|E_{[1,t_{\max}]}|)$. 保存区间核邻居的空间代价为 $O\left(\sum_{u\in V}|CN_u|\right)\leq O\left(\sum_{u\in V}deg_{[1,t_{\max}]}(u)\right)=O(|E_{[1,t_{\max}]}|)$，保存核数的代价为 $O(|V|)$. 由于 $|V|$ 小于 $|E_{[1,t_{\max}]}|$，算法 5 的空间代价为 $O(|E_{[1,t_{\max}]}|)$. 对算法 6，保存 k 核时间邻居的代价为 $O\left(\sum_{u\in V}\sum_{k\in[1,core_{[1,t_{\max}]}(u)]}|CTN_{(u,k)}|\right)\leq O(k_{\max}\times|E_{[1,t_{\max}]}|)$. 保存顶点的核数、核时间的空间代价分别为 $O(|V|)$ 和 $O(|E_{[1,t_{\max}]}|)$，保存最短 k 核区间的空间代价等于 SIHC 索引的大小 $O(\bar{t}\times|E_{[1,t_{\max}]}|)$，因此算法 6 的空间代价为 $O(k_{\max}\times|E_{[1,t_{\max}]}|)$. 综上，算法 4 的空间复杂度为 $O(k_{\max}\times|E_{[1,t_{\max}]}|)$. 证毕.

ZHOU Jun-Feng, Ph. D., professor. His main research interests focus on query processing techniques on large graphs.

WANG Chun-Hua, M. S. candidate. Her research interests focus on mining dense subgraphs.

DU Ming, Ph. D., professor. His main research interests include information retrieval and data analysis.

CHEN Zi-Yang, Ph. D., professor. His main research interests focus on unstructured data management.

Background

Dense subgraph mining is a hot research issue in the field of graph data management. The k-core is one of the most important models for dense subgraphs, where each vertex has at least k neighbors. In the temporal graph, querying k-core

subgraphs requires the system to return the corresponding k-core subgraph $C_k(G_I)$ from the snapshot graph G_I of the temporal graph G, based on the given query interval $I=[s,e]$ and the value of k.

Given a temporal graph G and a time interval I, one basic method for solving the k-core problem is the online approach. The problem is high query cost due to graph traversal for each query. Another method is the index-based approach. The latest method is the PHC index[17]. The basic idea is to maintain two types of information to quickly determine whether a vertex satisfies the given conditions. These two types of information include: (1) the sets of vertices corresponding to different values of k, and (2) the time intervals associated with each vertex in the vertex set. Specifically, for each possible value of k, a vertex set S_k is maintained, where each vertex u in S_k is a vertex in the k-core subgraph $C_k(G_I)$ of an interval snapshot graph G_I. For each vertex u in S_k, a set of time intervals $L_u=\{I_1,I_2,\cdots,I_n\}$ is maintained, where each time interval $I\in L_u$ represents the intervals in the corresponding snapshot graph G_I and u is a vertex in the k-core subgraph $C_k(G_I)$. Given a time interval I and a value of k, when conducting a k-core subgraph query, the PHC index-based approach first locates the vertex set S_k based on the value of k, and then checks each vertex in S_k to determine if it is a vertex in the k-core subgraph of G_I.

From the above description, it can be observed that when using the PHC index to handle k-core subgraph queries on temporal graphs, it is necessary to check all the vertices in S_k to obtain the results. Since S_k corresponds to the maximum interval snapshot graph $G_{[1,t_{\max}]}$, and the user's query time interval I is typically much smaller than the maximum time interval $[1,t_{\max}]$, the number of vertices satisfying the conditions in S_k is relatively small. Therefore, when conducting k-core subgraph queries based on the PHC algorithm, there are a large number of invalid judgments, resulting in lower efficiency.

To improve the efficiency of k-core subgraph queries on temporal graphs, this paper proposes a new index called SIHC to accelerate queries processing. Unlike the PHC index that maintains an inverted list from vertices to intervals, the SIHC index maintains an inverted list from intervals to vertices, where all vertices in the inverted table are satisfied ones. During query processing, the inverted list that satisfies the given interval can be directly determined based on the given query interval, and thus obtain the results that meet the requirements immediately. Compared to PHC, query processing based on the SIHC index does not require checking whether the vertices satisfy the conditions, resulting in a significant improvement in query efficiency.

This work was partly supported by grants from the National Natural Science Foundation of China (Nos. 62372101, 62272097, 61873337). This study aims to improve the efficiency of k-core queries on temporal graphs and further enhance and develop graph data management techniques.