软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(12):5526−5543 [doi: 10.13328/j.cnki.jos.007064] [CSTR: 32375.14.jos.007064]

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

面向时序图的季节突发性子图挖掘算法*

张千桢, 郭得科, 赵 翔

(国防科技大学 系统工程学院, 湖南 长沙 410073)

通信作者: 郭得科, E-mail: dekeguo@nudt.edu.cn; 赵翔, E-mail: xiangzhao@nudt.edu.cn

摘 要: 时序图是一类边上带有时间戳信息的图. 在时序图中, 季节突发性子图是在多个时间周期内具有突发性特征的稠密子图, 它可以用于社交网络中的活动发现和群体关系分析. 然而以前大多数的研究主要集中在识别没有时间信息的网络中的稠密子图. 为此, 提出一种极大 (ω,θ)-稠密子图模型对时序图中的季节突发性子图进行建模. 所提模型表示时序图中在至少 ω 个长度不小于 θ 的时间段内快速累积密度的子图. 为了挖掘出时序图中所有的极大 (ω,θ)-稠密子图, 将该类挖掘问题转化为一个混合的整数规划问题, 包括挖掘最稠密子图和寻找突发值最大化时间段集合两个子问题, 并给出有效的解决方案. 进一步基于 key-核模型和动态规划思想设计两种优化策略来提升算法的性能. 实验表明所提模型能够真实地反映现实世界中具有季节突发性的行为模式. 同时在 5 个真实时序网络中验证了所提算法的有效性、效率和可扩展性.

关键词: 时序图; 稠密子图; 季节突发性; 子图挖掘; 时间段

中图法分类号: TP18

中文引用格式: 张千桢, 郭得科, 赵翔. 面向时序图的季节突发性子图挖掘算法. 软件学报, 2024, 35(12): 5526–5543. http://www.jos.org.cn/1000-9825/7064.htm

英文引用格式: Zhang QZ, Guo DK, Zhao X. Mining Method Seasonal-bursting Subgraphs in Temporal Graphs. Ruan Jian Xue Bao/Journal of Software, 2024, 35(12): 5526–5543 (in Chinese). http://www.jos.org.cn/1000-9825/7064.htm

Mining Method Seasonal-bursting Subgraphs in Temporal Graphs

ZHANG Qian-Zhen, GUO De-Ke, ZHAO Xiang

(College of Systems Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: Temporal graph is a type of graph where each edge is associated with a timestamp. Seasonal-bursting subgraph is a dense subgraph characterized by burstiness over multiple time periods, which can applied for activity discovery and group relationship analysis in social networks. Unfortunately, most previous studies for subgraph mining in temporal networks ignore the seasonal or bursting features of subgraphs. To this end, this study proposes a maximal (ω,θ)-dense subgraph model to represent a seasonal-bursting subgraph in temporal networks. Specially, the maximal (ω,θ)-dense subgraph is a subgraph that accumulates its density at the fastest speed during at least ω particular periods of length no less than θ on the temporal graph. To compute all seasonal bursting subgraphs efficiently, the study first models the mining problem as a mixed integer programming problem, which consists of finding the densest subgraph and the maximum burstiness segment. Then corresponding solutions are given for each subproblem, respectively. The study further conceives two optimization strategies by exploiting key-core and dynamic programming algorithms to boost performance. The results of experiments show that the proposed model is indeed able to identify many seasonal-bursting subgraphs. The efficiency, scalability, and effectiveness of the proposed algorithms are also verified on five real-life datasets.

Key words: temporal graph; dense subgraph; seasonal burstiness; subgraph mining; time span

随着大数据时代的到来, 多源异构数据的快速增长已经成为一个开放性问题, 这些数据之间的内在关联通常可以用图的形式来表现. 其中图中的节点表示实体, 节点之间的边表示实体之间的关系, 如 Web 网络[1]、社交网

* 基金项目: 国防基础科研计划 (WDZC20235250412); 国家自然科学基金 (U19B2024, 62272469)
收稿时间: 2022-06-25; 修改时间: 2023-05-26, 2023-08-26; 采用时间: 2023-09-26; jos 在线出版时间: 2024-02-05
CNKI 网络首发时间: 2024-02-09

络[2]、生物蛋白质网络[3]等. 挖掘图数据中的有用知识已经成为一个重要的研究方向, 简称图数据挖掘. 其中, 稠密子图发现是图数据挖掘中一个十分重要的问题[4], 有着广泛的应用. 稠密子图作为具有特殊结构和性质的子图, 可表示现实网络中具有某种特定意义的对象群体. 例如, 在社交网络分析中, 稠密子图对应于具有紧密社交关系的社区[2]; 在 Web 网络中, 稠密子区域可用于发现主题群组和 spam 链接等[1]. 因此, 稠密子图发现问题具有广泛的应用价值和重要的理论意义.

目前, 大量的研究工作给出了不同的稠密子图的定义和相应的查询算法, 例如, k-core[5]、k-truss[6]、极大团[7]和准团[8]等. 然而现有的研究工作主要针对静态图中的稠密子图挖掘问题. 在实际应用中, 图中的边通常随时间变化, 从而演化出时序图. 时序图中每一条边有一个时间戳表示该边出现的时间. 例如在一个面对面接触网络中[9,10], 边 $\langle u,v,t\rangle$ 表示两个实体 u 和 v 之间在时刻 t 存在一次联系; 在一个邮件通信网络中, 每一封邮件包含一个发送方和一个接受方以及这封邮件的发送时间; 在学者协作网络中, 边 $\langle u,v,t\rangle$ 表示协作出版物的两个合作学者 u 和 v 在时刻 t 合作了一篇文章. 值得注意的是, 时序图中的稠密子图挖掘问题尚处于起步阶段, 需要根据具体研究问题的场景来定义时序图中的稠密子图挖掘问题并设计相应的挖掘算法.

近年来越来越多的学者研究时序图中的稠密子图挖掘问题. 例如, 文献 [11] 提出了时序图中具有周期性特征的稠密子图挖掘问题. 文献 [12] 定义了 (l,δ)-最大核模型, 用于挖掘时序图中在不小于 l 的时间段内, 满足每个节点的平均度不小于 δ 的稠密子图. 文献 [13] 提出在加权时序图中通过时间快照计算权值找到一个时间段内的稠密子图. 然而现有的研究工作没有关注时序图中的季节突发性特征. 季节突发性特征是一种常见的交互现象, 反映多个时间段内用户之间的交互规律. 其中, 突发性特征表示短时间内用户之间的交互越来越频繁, 季节性特征表示这种具有突发性特征行为出现的次数. 例如: (1) 来自多个机构的研究人员正在合作一个大型项目, 团队成员的日常活动通常与项目无关. 但每隔几周或几个月 (季节性特征) 在交付期限或项目会议之前, 团队成员之间的交互会变得更加紧密 (突发性特征). (2) 一群 Twitter 用户对技术产品感兴趣, 他们彼此之间会评论对方的帖子. 这些用户之间平时的互动很少, 但在新产品发布后 (季节性特征) 他们之间的互动会显著加强 (突发性特征. 挖掘这种具有季节突发性特征的子图对于理解时序网络中在多个时间段内具有紧密联系的行为至关重要. 在时序图中通常变化最快的部分具有很大的研究价值, 为此, 本文充分考虑了时序图中的时序信息, 将静态图中基于边密度的稠密子图挖掘问题扩展到时序图中, 提出了一种新的稠密子图挖掘问题: 季节突发性子图挖掘问题. 给定一个时序图, 季节突发性子图表示在该时序图中多个时间段内以最快的速率累积密度的子图. 其中, 季节性特征主要体现在时间段的数目; 突发性特征主要体现在子图的密度累积速率, 即子图在多个时间段内的密度累加和与时间段总长度的比值. 这里子图的密度用该子图的边密度来衡量, 即子图的边数与节点数的比值.

由于在时序图中每条边上都有记录时间的标签序列, 且节点之间的联系是随着时间动态变化的, 而查找的稠密时序子图不仅要保证在拓扑结构上的稠密性, 而且要保证子图中的边满足时序约束. 因此, 现有的静态图上的稠密子图查询算法无法直接应用季节突发性子图挖掘. 此外, 现有的时序图中的稠密子图挖掘算法仅考虑了子图的密度, 没有考虑子图密度的累积速率以及速率的变化情况. 由于一个子图可能会在很长的时间段内缓慢地累积到一个比较大的密度, 因此该子图不一定是密度突发性子图. 此外, 这些研究没有考虑子图的季节性特征. 本文首次提出了时序图上的季节突发性子图挖掘问题, 并给出有效的解决方案.

本文的创新贡献主要包括以下 4 个方面.

(1) 提出了一种极大 (ω,θ)-稠密子图模型对季节突发性子图进行建模. 该模型表示在时序图中至少 ω 个长度不小于 θ 的时间段内以最快速率累积密度的子图. 其中参数 ω 用来反映子图的季节性特征, 参数 θ 用来限制由于持续时间短而引发的密度累积速率过高的情形.

(2) 基于极大 (ω,θ)-稠密子图模型, 提出一种基准算法来挖掘时序图中的季节突发性子图. 该算法将季节突发性子图挖掘问题转化成一个混合的整数规划问题: 给定一个初始的时间段集合, 通过迭代求解最稠密子图挖掘和最大突发值时间段集合挖掘两个子问题来发现一个季节突发性子图, 并为每一个子问题设计有效的解决方案.

(3) 为了进一步提升算法的效率, 减少搜索空间, 提出了两种优化策略. 针对基准算法中的第 1 个子问题, 提出了一种近似比为 0.5 的最稠密子图挖掘算法. 该算法基于 k-核结构首先定位可能包含最稠密子图的一些小的稠密

子图, 接着在这些小的稠密子图中计算 k_{max}-核作为最稠密子图. 针对基准算法中的第 2 个子问题, 将其转化为求解最大斜率问题, 并提出一种动态规划算法, 通过维护一个小的数组可以有效地将该部分时间复杂度由 $O(T^2)$ 降为 $O(T)$.

(4) 在 5 个真实的时序网络数据集中进行实验. 实验结果验证了所提算法的有效性、效率和可扩展性. 值得注意的是, 本文所提的优化算法在效率上比基准算法快了近 2 个量级. 同时在 Twitter 数据集中进行了案例分析, 结果表明本文所提算法能够在真实的时序网络中识别出有趣的季节突发性子图.

本文第 1 节介绍时序图子图挖掘的相关工作. 第 2 节给出一些定义以及问题描述. 第 3 节介绍本文所提的基准算法. 第 4 节在基准算法的基础上进一步给出两种优化策略. 第 5 节在 5 个真实数据集上验证所提算法的有效性和效率. 第 6 节总结全文并且给出未来的工作.

1 相关工作

时序图[14]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, 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. 时序图强调在一个时间阈值内数据的变化. 一般情况下时序图可以按照节点间联系持续时间的大小分为以下两种情况. 第 1 种情况为各节点间联系无持续性或其持续时间可忽略不计. 该种情况下图中的边可以用三元组 $\langle v,v',t\rangle$ 表示, 每条边都存在一组时间的序列. 第 2 种情况是时序图中的边在一定的时间段内被激活. 在这种时序图中, 持续时间是一个重要、不可忽视且必须考虑的因素. 本文是基于第 1 种情况下在时序图中进行稠密子图挖掘研究的. 值得注意的是, 时序图与动态图[15]不同, 动态图是指会随着时间实时发生变化的图数据, 包括边的增加和删除操作. 而时序图则研究图中历史数据随时间的演化情形, 其本质上是边带有时序信息的静态图.

时序图中的稠密子图挖掘问题近年来备受关注, 根据应用场景的不同, 现有时序图中的稠密子图可分为权重子图[13,16]、特征子图[11,12,17,18]和动态稠密子图[19,20]. Ma 等人提出 FIDES 算法[13]和 FIDES$^+$算法[16]在加权时序图中通过时间快照计算权值找到一个时间段内具有较大边权重的稠密子图. Li 等人[17]提出在时序图中挖掘具有持久性特征的社区. Qin 等人[11]研究了时序图中的具有周期性特征的社区挖掘问题. Zhang 等人[18]研究了时序图中的季节周期性子图挖掘问题, 考虑在多个时间段内呈现周期性特征的子图. Liu 等人[19]提出了一种新颖的随机算法在时序图中挖掘在较长时间段内仍然具有较高密度的子图. 然而这些工作都没有考虑子图的密度突发性特征.

与本文所研究问题相关的工作有两个. 文献 [12] 提出了 (l,δ)-最大核模型来定义时序图中的密度突发性子图. 该模型表示时序图中在不小于 l 的时间段内, 满足每个节点的平均度不小于 δ 的稠密子图. 然而该模型对突发性特征的定义仅考虑了子图中每个节点的平均度, 没有考虑子图的边密度以及边密度的累积速率; 其次, 该模型没有考虑子图的季节性特征. 文献 [20] 研究了在不同时间段内满足边密度阈值的稠密子图挖掘问题. 该研究仅考虑稠密子图的季节性特征, 没有考虑子图的密度突发性特征, 因此不能应用到本文的研究工作当中.

当给定一个时间段之后, 时序图可以当作一个静态图来进行处理, 其中图上边的时间信息可以看作为边的标签. 因此为了挖掘在该时间段内以最快的速率累积密度的子图, 可以将其转化为静态图上基于边密度的最稠密子图挖掘问题. 给定一个子图 S, 其边密度定义为 m/n, 其中 m 表示该子图中边的数目, n 表示子图中的节点数目. 基于边密度的最稠密子图挖掘问题为挖掘图中边密度最大的子图. 该问题可以通过求解一个参数化最大流问题来解决[21]. Qin 等人[22]设计了有效的算法挖掘图中前 k 个局部密度最大的子图. Epasto 等人[23]研究了演变图中的最稠密子图发现问题. 上述算法为精确的最稠密子图挖掘算法, 其在图规模比较小的时候可以获得比较好的结果. 当图的规模较大时算法的效率显著降低. 因此, 研究人员开发了近似算法, 以获得更高的效率. Charikar 等人[24]提出了一种近似率为 0.5 的启发式算法来挖掘最稠密子图. 这里近似率指的是近似算法返回的最稠密子图的密度与精确算法返回的最稠密子图的密度比值. Bahmani 等人[25]提出了流图环境下一个近似率为 $1/(2+2\varepsilon)$ 的最稠密子图挖掘算法, 该算法的时间复杂度为 $O(m\log(n)/\varepsilon)$. 本文采用文献 [24] 提出的最稠密子图挖掘算法作为基线算法, 并对其进行进一步优化.

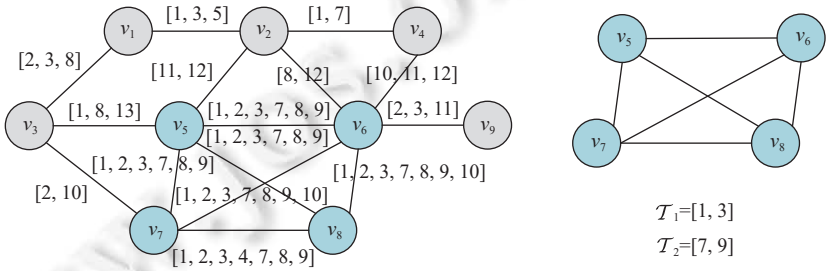
此外, 还有其他研究工作对时序图进行分析. Yang 等人[26]研究了时序图中变化最频繁的子图挖掘问题. Huang 等人[27]研究了时序图中的最小生成树问题. Gurukar 等人[28]提出了一种识别具有相似信息流序列的重复子图的模型. Wu 等人[29]提出了一种时序图上的 (k, h)-核模型, 即子图中每个顶点至少有 k 个邻接点和 h 条时序边, 用于时序图的核心分解. 文献 [30] 提出了时序图中一种长度限制下的环路枚举新算法. 该算法通过引入新型的剪枝技术, 允许用户快速获得某一长度以内的所有时序环.

2 问题描述及相关定义

在本节中, 我们首先介绍了时序图和时序子图的相关定义, 接着给出时序图中季节突发性子图挖掘问题的形式化描述.

2.1 时序图和时序子图

时序图 $\mathcal{G}$ 可以用一个三元组 $(\mathcal{V},\mathcal{E},T)$ 来表示, 其中, $\mathcal{V}$ 表示节点集, $\mathcal{S}=\mathcal{V}\times\mathcal{V}\times T$ 表示时序边集, 每一条边都与时间集 T 中的一个时间戳相关联. 例如时序边 $e=\langle v,v',t\rangle$ $(t\in T)$ 表示节点 v 和 v' 在时刻 t 存在交互. 为了不失一般性, 假设每一个时间戳都为整数, 且所有时间戳都按时间顺序排列, 即 $t_1<t_2<\ldots<t_{|T|}$. 图 1(a) 给出了一个与通信网络相关的采样时序图, 其中图中的节点表示科研工作者, 节点之间的边表示两者在该时刻进行了通信.



(a) 采样时序图 (b) 极大 (2, 3)-稠密子图

图 1 季节突发稠密子图示例

给定一个由时间戳 t_s 到 t_e 的时间段 $\mathcal{T}=[t_s,t_e]$, 其中 t_s 和 t_e 为 $\mathcal{T}$ 中开始时间和结束时间, 该时间段长度为 $|\mathcal{T}|=t_e-t_s+1$. 本文用 $\mathcal{G}(\mathcal{T})$ 表示时序图 $\mathcal{G}$ 在时间段 $\mathcal{T}$ 内的存在性, 这里 $\mathcal{T}$ 用来限制 $\mathcal{G}$ 在时间段 $\mathcal{T}$ 内存在的时序边. 基于此, 时序图 $\mathcal{G}$ 等价于 $\mathcal{G}(\mathcal{T})$. 相似地, 给定一个时间段集合 $\Omega=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_n\}$, 用 $\mathcal{G}(\Omega)$ 表示时序图 $\mathcal{G}$ 在时间段集合 Ω 内的存在性.

定义 1. 时序子图. 给定一个时序图 $\mathcal{G}$, 一个时间段 $\mathcal{T}$, 和一个节点集 $S\in\mathcal{V}$, 时序子图 $\mathcal{G}_S(\mathcal{T})=(S,\mathcal{E}_S(\mathcal{T}))$ 为时序图 $\mathcal{G}(\mathcal{T})$ 的诱导子图, 满足 $\mathcal{E}_S(\mathcal{T})=\{\langle u,v,t\rangle|u,v\in S,t\in\mathcal{T},\langle u,v,t\rangle\in\mathcal{E}\}$. 相似地, 给定一个时间段集合 Ω, $\mathcal{G}_S(\Omega)=(S,\mathcal{E}_S(\Omega))$ 为时序图 $\mathcal{G}(\Omega)$ 中的时序子图.

2.2 季节突发性子图的形式化描述

给定一个时序子图 $\mathcal{G}_S(\Omega)=(S,\mathcal{E}_S(\Omega))$, 本文采用边密度来衡量子图的密度. 因此, 时序子图 $\mathcal{G}_S(\Omega)$ 的密度 $d(\mathcal{G}_S(\Omega))$ 定义为 $d(\mathcal{G}_S(\Omega))=\dfrac{|\mathcal{E}_S(\Omega)|}{|S|}$.

定义 2. 最稠密子图. 给定一个时序图 $\mathcal{G}(\Omega)$, 找到一个节点集 $S\in\mathcal{V}$ 使得 $d(\mathcal{G}_S(\Omega))$ 的值最大.

时序子图 $\mathcal{G}_S(\Omega)$ 的突发值为该子图在时间段集合 Ω 内的密度累积速率. 因此, 其突发值 $B(\mathcal{G}_S(\Omega))$ 定义为 $B(\mathcal{G}_S(\Omega))=\dfrac{d(\mathcal{G}_S(\Omega))}{\sum_{i=1}^{n}|\mathcal{T}_i|}$. 接下来给出本文所提极大 (ω,θ)-稠密子图模型的定义.

定义 3. 极大 (ω,θ)-稠密子图. 给定一个时序图 $\mathcal{G}$, 一个时间段集合 $\Omega=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_n\}$ 和一个节点集 $S\in\mathcal{V}$, 时序子图 $\mathcal{G}_S(\Omega)$ 为极大 (ω,θ)-稠密子图当且仅当: (1) $|\mathcal{T}_i|\geqslant\theta\,(1\leqslant i\leqslant n)$, $n\geqslant\omega$; (2) 不存在其他节点集 $S'\in\mathcal{V}$, 使得 $B(\mathcal{G}_{S'}(\Omega))>B(\mathcal{G}_S(\Omega))$; (3) 不存在其他时间段集合 Ω', 使得 $B(\mathcal{G}_S(\Omega'))>B(\mathcal{G}_S(\Omega))$.

条件 (1) 中参数 ω 保证了子图的季节性特征, 时间段大小约束 θ 可以排除掉那些由于持续时间太短导致突发值较大的无意义子图, 例如一个科研团队在 1 天内的正常邮件交互可以使得突发值很大, 但该结果是由于持续时间短导致, 没有实际意义; 然而如果该团队持续 5 天仍然产生很大的突发值, 说明该团队正在集中攻关一个科研项目, 期间内团队内的联系持续保持紧密; 条件 (2) 保证了子图 S 为局部最稠密子图; 条件 (3) 保证了子图 S 的突发值最大.

图 1(b) 展示了在图 1(a) 所示的通信网络挖掘出的季节突发稠密子图示例. 给定 $\omega=2$, $\theta=3$, 节点 $S=\{v_5, v_6, v_7, v_8\}$在时间段 $\mathcal{T}_1=[1,3]$ 和 $\mathcal{T}_2=[7,9]$ 形成时序子图的突发值 $B(\mathcal{G}_S(\Omega))=\frac{9}{6}=1.5$ $(\Omega=\{\mathcal{T}_1,\mathcal{T}_2\})$. 由于时序子图 $\mathcal{G}_S(\Omega)$ 为时序图 $\mathcal{G}(\Omega)$ 中的最稠密子图, 因此不存在其他节点集 $S'\in\mathcal{V}$ 使得 $B(\mathcal{G}_{S'}(\Omega))>B(\mathcal{G}_S(\Omega))$. 进一步, 从图 1(a) 中可以明显地看到不存在其他时间段集合 Ω', 使得 $B(\mathcal{G}_S(\Omega'))>B(\mathcal{G}_S(\Omega))$, 因此时序子图 $\mathcal{G}_S(\Omega)$ 为极大 (2, 3)-稠密子图.

2.3 问题定义

本文具体研究的问题是: 给定一个时序图 $\mathcal{G}$、参数 ω 和 θ, 枚举时序图中所有的极大 (ω,θ)-稠密子图. 为了便于描述, 接下来将极大 (ω,θ)-稠密子图缩写为 SBS.

问题挑战: 由于时序图本身是带有时间戳的静态图, 需要考虑的问题是传统的静态图中稠密子图挖掘方法能否用于挖掘极大 (ω,θ)-稠密子图. 答案是否定的, 这是因为传统的静态图中的稠密子图挖掘算法仅考虑了子图的密度, 没有考虑子图密度的累积速率以及速率变化的季节性特征, 因此无法直接应用到挖掘极大 (ω,θ)-稠密子图.

3 SBS 挖掘算法

根据定义 3, 给定一个 SBS $\mathcal{G}_S(\Omega)=(S,\mathcal{E}_S(\Omega))$, 发现: (1) S 是在时间段集合 Ω 约束条件下使得突发值 $B(\mathcal{G}_S(\Omega))$ 形成局部极值的节点集; (2) Ω 是在节点集 S 约束条件下使得 $B(\mathcal{G}_S(\Omega))$ 形成局部极值的时间段集合. 因此, 可以通过求解一个混合的整数规划 (MIP) 问题来挖掘时序图中所有的 SBS.

根据定义 3, 一个 SBS $\mathcal{G}_S(\Omega)=(S,\mathcal{E}_S(\Omega))$ 为以下 MIP 问题中的一个局部最大点:

$$\arg\max_{(S,\Omega)} B(\mathcal{G}_S(\Omega)),\ \text{s.t. } S\in\mathcal{V},\Omega=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_n\},|\mathcal{T}_i|\geqslant\theta,n\geqslant\omega \tag{1}$$

因此, 给定一个初始的时间段集合 Ω, 可以通过迭代地更新 S 和 Ω 来增加突发值 $B(\mathcal{G}_S(\Omega))$, 从而获取一个局部最大点. 基于此, 我们将该 MIP 问题分解为以下两个子问题.

问题 1. 寻找最优节点集合. 给定一个时间段集合 Ω, 在时序图 $\mathcal{G}(\Omega)$ 寻找一个节点集 S, 使得 $B(\mathcal{G}_S(\Omega))$ 的值最大.

问题 2. 寻找最优时间段集合. 给定一个节点集 S, 在时序图寻找一个时间段集合 $\Omega=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_n\}$, 使得 $B(\mathcal{G}_S(\Omega))$ 的值最大且满足 (1) $|\mathcal{T}_i|\geqslant\theta$ $(1\leqslant i\leqslant n)$; (2) $n\geqslant\omega$.

综上, 时序图中季节突发性子图挖掘算法 findSBS 的伪代码如算法 1 所示.

算法 1. findSBS($\mathcal{G},\omega,\theta$).

输入: 读取时序图的输入流 $\mathcal{G}$, 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; 参数 ω 和 θ;

输出: 所有季节突发性子图 SBS.

1. $\mathcal{A}\leftarrow initializeSBS(\mathcal{G},\omega,\theta)$; //获取初始时间段集合
2. for each Ω in $\mathcal{A}$
3. repeat
4. $S\leftarrow\arg\max_S B(\mathcal{G}_S(\Omega))$;
5. $\Omega\leftarrow\arg\max_{\Omega}\frac{d(\mathcal{G}_S(\Omega))}{\sum_{i=1}^{n}|\mathcal{T}_i|}$, s.t. $|\mathcal{T}_i|\geqslant\theta$, $n\geqslant\omega$; //挖掘突发值最大时间段集合

6. until the value of $B(\mathcal{G}_S(\Omega))$ does not increase;
7. if *Coverage*(*S*) = TRUE //判断子图是否出现过
8. $ResultSet \leftarrow ResultSet \cup \{\mathcal{G}_S(\Omega)\}$;
9. end if
10. end for
11. return *ResultSet*;

算法 1 首先获取初始的时间段集合候选集 $\mathcal{A}$ (第 1 行). 对于候选集 $\mathcal{A}$ 中的每一个时间段集合, 通过迭代地解决问题 1 和问题 2 直到密度突发值 $B(\mathcal{G}_S(\Omega))$ 不再增加为止 (第 2–6 行). 接下来判断是否 *Coverage*(*S*) = TRUE; 若成立, 则将时序子图 $\mathcal{G}_S(\Omega)$ 加入 *ResultSet* 中 (第 7–9 行). 这里 *Coverage*(*S*) = TRUE 表明在 *ResultSet* 中不存在一个 SBS 使得其节点集与 *S* 相同.

给定一个初始的时间段集合, 算法 1 最多可以识别 $O(|\mathcal{V}|\times|\mathcal{E}|)$ 个独一无二的时序子图. 因此, 算法可以保证在最多迭代 $O(|\mathcal{V}|\times|\mathcal{E}|)$ 次后停止, 从而保证算法的有效性. 算法 1 主要包含 3 个核心步骤, 分别为: 初始化时间段集合、寻找最优节点集和寻找最优时间段集合, 接下来分别给出相应的解决方案.

3.1 算法初始化

算法 1 中返回结果的完整性与选取的初始时间段集合候选集密切相关. 给定一个时序子图 $\mathcal{G}_S(\Omega)$, 其中 $\Omega=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_n\}$, 若 $\mathcal{G}_S(\Omega)$ 为 SBS 且密度为 d^*, 则在 Ω 中必然存在一个时间段 $\mathcal{T}_i$, 使得在时序图 $\mathcal{G}(\mathcal{T}_i)$ 中由节点集 *S* 产生的时序子图 $\mathcal{G}_S(\mathcal{T}_i)$ 满足其密度 $d(\mathcal{G}_S(\mathcal{T}_i))\geqslant\frac{d^*}{n}$. 基于此, 算法 1 可以选择所有长度不小于 θ 的单个时间段作为初始候选集.

由于在时序图中存在 $O(|T|^2)$ 个长度不小于 θ 的时间段, 当 *T* 的值较大时, 算法 1 无法在有效的时间内返回所有的 SBS. 因此, 为了提升算发的效率, 可以从候选集 $\mathcal{A}$ 中随机选择 $\mathcal{J}$ 个时间段来运行迭代算法从而获取所有可能的结果. 实验中会给出 $\mathcal{J}$ 的值对时序图中发现的季节突发性子图数目的影响.

3.2 寻找最优节点集

由于在问题 1 中已经给定了时间段集合 Ω, 因此可以进一步将问题 1 转化为在时序图 $\mathcal{G}(\Omega)$ 中挖掘最稠密子图, 即寻找一个节点集 *S* 来最大化时序子图密度 $d(\mathcal{G}_S(\Omega))$. 值得注意的是, 可以将限定了时间段范围的时序图当成一个静态图来处理, 因此问题 1 可以用现有的关于静态图工作中的最稠密子图挖掘技术来解决. 由于精确的最稠密子图挖掘算法不适用于图规模较大的情形, 因此, 本文采用文献 [24] 提出的一种启发式算法 findDSS 来求解该问题. 该算法可以保证 0.5 的近似率, 即该近似算法返回的最稠密子图的密度与精确算法返回的最稠密子图的密度比值不小于 0.5. 算法 findDSS 的伪代码如算法 2 所示.

算法 2. findDSS($\mathcal{G}(\Omega)$).

输入: 读取时序图 $\mathcal{G}$ 在时间段集合 Ω 内的快照 $\mathcal{G}(\Omega)$;
输出: 使得时序子图密度最大的最优节点集 *S*.

1. $\bar{S}\leftarrow\mathcal{G}(\Omega)$, $S\leftarrow\varnothing$;
2. *computeDegree*($\bar{S}$); //获取 $\bar{S}$ 中每一个节点的度
3. while $\bar{S}\neq\varnothing$
4. $v\leftarrow minDegree(\bar{S})$; //获取 $\bar{S}$ 中度最小的节点
5. $\bar{S}\leftarrow\bar{S}\backslash v$; //将 *v* 从 $\bar{S}$ 中删除
6. if $d(\bar{S})\geqslant d(S)$
7. $S\leftarrow\bar{S}$;

8. end if
9. end while
10. return S;

假设时序图$\mathcal{G}(\Omega)$的节点数为n, 算法 2 将进行n次迭代计算. 每一次迭代计算算法 2 移除$\mathcal{G}(\Omega)$中度最小的节点, 并重新计算当前时序图的密度. 当所有的迭代完成之后, 算法 2 即可返回最稠密子图S.

3.3 寻找最优时间段集合

问题 2 为突发值最大时间段集合发现问题, 该问题为 NP-难问题. 证明: 为了计算使得该子图突发值最大的时间段集合, 首先需要获取所有长度不小于θ的时间段, 接着在这些时间段集合中寻找至少ω个互不相交的时间段使得该子图的突发值最大. 可以发现传统的集合覆盖问题为问题 2 的一种特例, 由于集合覆盖问题为 NP-难问题, 因此可以证明问题 2 也为 NP-难问题.

基于集合覆盖问题的求解思想, 本文提出了一种启发式算法来求解突发值最大时间段集合发现问题, 主要步骤如下.

步骤 1. 给定一个节点集S, 初始化其时间段集合Ω为空集.

步骤 2. 在时间集T中选择一个长度不小于θ的时间段$\mathcal{T}$来最大化该时序子图的突发值, 即最大化:

$$B(\mathcal{G}_S(\Omega\cup\mathcal{T}))=\frac{d(\mathcal{G}_S(\Omega))+d(\mathcal{G}_S(\mathcal{T}))}{len(\Omega)+|\mathcal{T}|} \tag{2}$$

其中, $len(\Omega)$表示在时间段集合Ω中所有时间段的长度之和. 接着将$\mathcal{T}$加入Ω中.

步骤 3. 判断算法是否终止, 若不终止, 则将$\mathcal{T}$从时间集T中删除, 并重复步骤 2; 若算法终止则返回Ω作为最优时间段集合.

步骤 3 中算法终止的条件可以通过公式 (2) 的单调性获得. 研究发现, 公式 (2) 具有反单调性, 即$B(\mathcal{G}_S(\Omega\cup\mathcal{T}))<B(\mathcal{G}_S(\Omega))$. 证明: 给定两个时间段$\mathcal{T}_1$和$\mathcal{T}_2$且满足$\frac{d(\mathcal{G}_S(\mathcal{T}_1))}{|\mathcal{T}_1|}>\frac{d(\mathcal{G}_S(\mathcal{T}_2))}{|\mathcal{T}_2|}$. 根据等量变化可得$\frac{d(\mathcal{G}_S(\mathcal{T}_2))}{|\mathcal{T}_2|}<\frac{d(\mathcal{G}_S(\mathcal{T}_1))+d(\mathcal{G}_S(\mathcal{T}_2))}{|\mathcal{T}_1|+|\mathcal{T}_2|}<\frac{d(\mathcal{G}_S(\mathcal{T}_1))}{|\mathcal{T}_1|}$. 假设$\Omega=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k\}$, 根据上述不等式可得$B(\mathcal{G}_S(\Omega))>B(\mathcal{G}_S(\mathcal{T}_k))$. 当加入一个新的时间段$\mathcal{T}$后, 由于$B(\mathcal{G}_S(\mathcal{T}_k))>B(\mathcal{G}_S(\mathcal{T}))$, 从而可得$B(\mathcal{G}_S(\Omega))>B(\mathcal{G}_S(\mathcal{T}))$, 根据不等式性质进而得证$B(\mathcal{G}_S(\Omega\cup\mathcal{T}))<B(\mathcal{G}_S(\Omega))$. 因此, 为了获取突发值最大化时间段集合, 步骤 3 中算法的终止条件设为当$|\Omega|=\omega$时停止.

进一步研究发现, $\frac{d(\mathcal{G}_S(\mathcal{T}))}{|\mathcal{T}|}$的值越大, $B(\mathcal{G}_S(\Omega\cup\mathcal{T}))$的值也就越大. 因此, 步骤 2 可以进一步转化为在时间集T中选择一个长度不小于θ的时间段$\mathcal{T}$来最大化突发值$B(\mathcal{G}_S(\mathcal{T}))$. 为了解决该问题, 可以首先获取时序图中所有长度不小于θ的时间段, 接着选取使得突发值$B(\mathcal{G}_S(\mathcal{T}))$最大的时间段.

算法分析: 算法 2 通过迭代地删除时序图$\mathcal{G}(\Omega)$中度最小的节点来最终获取密度最大的稠密子图. 假设$\mathcal{G}(\Omega)$中边的数目为m, 算法 2 的主要的计算开销来自枚举$\mathcal{G}(\Omega)$中时序边的数目. 因此其时间复杂度和空间复杂度均为$O(m)$.

在寻找最优时间段集合的过程中, 主要计算开销主要来自步骤 2. 由于步骤 2 在计算使得$B(\mathcal{G}_S(\mathcal{T}))$值最大的时间段$\mathcal{T}$时, 需要考虑所有长度不小于$\theta$的时间段, 因此该部分时间复杂度和空间复杂度均为$O(|T|^2)$.

4 两种优化策略

算法 1 虽然可以挖掘出时序图中的季节突发性子图, 但仍然面临以下两种挑战.

挑战 1. 算法 1 在计算最优节点集的过程中, 虽然可以在线性的时间复杂度内完成, 但是存在大量不必要的计算, 严重影响算法效率. 这是因为在最初的几轮迭代计算过程中, 返回的图里包含大量度很小的节点, 这些图不会成为密度最大的稠密子图, 但是算法 2 仍然需要计算这些图的密度.

挑战 2. 算法 1 在每一轮计算使得 $B(\mathcal{G}_S(\mathcal{T}))$ 值最大的时间段 $\mathcal{T}$ (即最优时间段) 的过程中需要考虑所有长度不小于 θ 的时间段. 即使在 $|T|$ 比较小的情况下, 例如 2 000, 仍然需要进行 2^6 次计算来返回使得突发值最大的时间段, 严重影响算法的效率.

本文基于上述两种挑战, 分别提出了相应的优化策略来改进算法的效率.

4.1 基于 *k*-核的最优节点集发现算法

为了解决挑战 1, 本文提出了一种基于 k-核的算法来快速地定位到可能包含最稠密子图的一些小的稠密子图中, 且只需要对这些小的稠密子图进行计算即可.

定义 4. k-核. 给定时序图 $\mathcal{G}(\Omega)$ 和一个整数 k ($k \geqslant 0$), k-核为 $\mathcal{G}(\Omega)$ 中的一个极大子图, 表示为 $\mathcal{H}_k$, 满足 $\forall v \in \mathcal{H}_k$, $deg(v) \geqslant k$. 其中, $deg(v)$ 表示节点 v 的度, k 表示 $\mathcal{H}_k$ 的阶数.

给定 $\mathcal{G}(\Omega)$ 中的一个节点 v, 其核心数定义为所有包含节点 v 的 k-核中阶数的最大值, $k_{\max}$ 表示 $\mathcal{G}(\Omega)$ 中所有节点的核心数的最大值. 接下来分析 k-核的相关性质.

定理 1. 给定一个时序图 $\mathcal{G}(\Omega)$ 和 $\mathcal{G}(\Omega)$ 中的一个 k-核 $\mathcal{H}_k$. $\mathcal{H}_k$ 的密度满足:

$$\frac{k}{2} \leqslant d(\mathcal{H}_k) \leqslant k_{\max} \tag{3}$$

为了证明上述定理, 给出了以下引理.

引理 1. 给定时序图 $\mathcal{G}(\Omega)$ 中的一个最稠密子图 $\mathcal{G}_S(\Omega) = (S, \mathcal{E}_S(\Omega))$, 由 S 中的任一子集 U 组成的连通子图密度相同.

证明: 可以直观地用反证法证明.

引理 2. 给定时序图 $\mathcal{G}(\Omega)$ 中的一个最稠密子图 $\mathcal{G}_S(\Omega) = (S, \mathcal{E}_S(\Omega))$, 移除 S 中的任一子集 U 会导致从 $\mathcal{G}_S(\Omega)$ 中移除至少 $d(\mathcal{G}_S(\Omega)) \times |U|$ 条边.

证明: 同样可以采用反证法证明. 假设 $\mathcal{G}_S(\Omega)$ 为最稠密子图且从 S 中移除 U 导致移除少于 $d(\mathcal{G}_S(\Omega)) \times |U|$ 条边. 将 U 从 S 中移除之后, 剩余图 $\mathcal{G}_{S\setminus U}(\Omega)$ 的密度变为:

$$d(\mathcal{G}_{S\setminus U}(\Omega)) = \frac{\mathcal{E}_{S\setminus U}(\Omega)}{|S|-|U|} > \frac{d(\mathcal{G}_S(\Omega))|S| - d(\mathcal{G}_S(\Omega))|U|}{|S|-|U|} = d(\mathcal{G}_{S\setminus U}(\Omega)) \tag{4}$$

与假设 $\mathcal{G}_S(\Omega)$ 为密度最大稠密子图相矛盾, 因此该引理成立.

根据上述引理, 可以得出 $d(\mathcal{G}_S(\Omega))$ 的上界.

引理 3. 给定 $\mathcal{G}(\Omega)$ 中的一个最稠密子图 $\mathcal{G}_S(\Omega)$ 和所有节点中的核心数最大值 $k_{\max}$, 可以得到:

$$d(\mathcal{G}_S(\Omega)) \leqslant k_{\max} \tag{5}$$

证明: 可以采用反证法证明. 假设 $d(\mathcal{G}_S(\Omega)) > k_{\max}$. 根据引理 2 可以得到从 S 中移除任一节点会导致从 $\mathcal{G}_S(\Omega)$ 中移除至少 $d(\mathcal{G}_S(\Omega))$ 条边或者超过 $k_{\max}$ 条边. 也就是说每一个节点至少与 $k_{\max}+1$ 条边相关. 这与 $k_{\max}$ 为核心数最大值相矛盾, 因此 $d(\mathcal{G}_S(\Omega))$ 的值最大为 $k_{\max}$.

接下来给出定理 1 的证明过程: 假设 h 为 $\mathcal{H}_k$ 中节点的数目. 由于 $\mathcal{H}_k$ 为 k-核, $\mathcal{H}_k$ 中的每一个节点 u 有至少 k 条边. 因此 $\mathcal{H}_k$ 中有至少 $\frac{k \times h}{2}$ 条边, 从而得到 $d(\mathcal{H}_k) \geqslant \frac{k}{2}$. 由于 $d(\mathcal{H}_k) \leqslant d(\mathcal{G}_S(\Omega))$, 因此 $d(\mathcal{H}_k) \leqslant k_{\max}$.

基于定理 1, 本文将时序图 $\mathcal{G}(\Omega)$ 中的 $k_{\max}$-核作为图中的最稠密子图, 接下来给出使用该方法获得结果的近似比.

定理 2. 给定一个时序图 $\mathcal{G}(\Omega)$, 其 $k_{\max}$-核 $\mathcal{H}_{k_{\max}}$ 的密度与精确算法返回的密度最大稠密子图 $\mathcal{G}_S(\Omega)$ 的密度比值不小于 0.5.

证明: 根据定理 1 可以得到 $\frac{k_{\max}}{2} \leqslant d(\mathcal{H}_{k_{\max}}) \leqslant k_{\max}$. 由于 $d(\mathcal{G}_S(\Omega)) \leqslant k_{\max}$, 则:

$$\frac{d(\mathcal{H}_{k_{\max}})}{d(\mathcal{G}_S(\Omega))} \geqslant \frac{k_{\max}/2}{k_{\max}} = 0.5 \tag{6}$$

$k_{\max}$-核可以直接使用文献 [31] 中提出的 k-核分解算法计算得到, 该算法可以在线性的时间复杂度内计算所

有节点的核心数. 为了进一步提升算法的效率, 本文提出了一种新的 $k_{\max}$-核挖掘算法 findDSS$^+$. findDSS$^+$依赖一个关键发现: $k_{\max}$-核通常为度较大的节点构成的子图. 因此提出从一系列由度较大的节点构成子图中来发现 $k_{\max}$-核. 此外, 如果找到了一个具有较大阶数的 k-核, 则可以利用该阶数直接剪枝掉那些节点度较小的子图. 算法 findDSS$^+$的伪代码如算法 3 所示.

算法 3. findDSS$^+$($\mathcal{G}(\Omega)$).

输入: 读取时序图 $\mathcal{G}$ 在时间段集合 Ω 内的快照 $\mathcal{G}(\Omega)$;
输出: 最优节点集 S.

1. *computeDegree*($\mathcal{G}(\Omega)$); //计算 $\mathcal{G}(\Omega)$ 中每一个节点 v 的度
2. *arrangeDegree*($\mathcal{G}(\Omega)$); //将 $\mathcal{G}(\Omega)$ 中节点按度的大小降序排列
3. $W \leftarrow W \cup \{v_{\max}\}$, $k_{\max} \leftarrow 0$, $S \leftarrow \varnothing$;
4. while $\max\limits_{v \in \mathcal{V} \setminus W} deg(v) \geqslant k_{\max}$
5. for each $v \in W$
6. *computeDegree*($\mathcal{G}_W(\Omega)$) //计算时序子图 $\mathcal{G}_W(\Omega)$ 中每一个节点的度
7. end for
8. $k_l \leftarrow \min\limits_{v \in W} deg(v)$, $k_u \leftarrow \max\limits_{v \in W} deg(v)$;
9. $k \leftarrow \max\{k_l, k_{\max}+1\}$;
10. while $k \leqslant k_u$ and $|W| > 0$
11. while $\exists v \in W, deg_W(v) < k$
12. $W \leftarrow W \setminus \{v\} \infty$, *updateDegree*($\mathcal{G}_W(\Omega)$); //更新时序子图 $\mathcal{G}_W(\Omega)$ 中的节点度
13. end while
14. if $|W| > 0$
15. if $k > k_{\max}$
16. $k_{\max} \leftarrow k$, $S \leftarrow \mathcal{G}_W(\Omega)$;
17. end if
18. $k \leftarrow k+1$;
19. end if
20. end while
21. *updateVertex*($W, \mathcal{G}(\Omega)$); //更新 W 为 $\mathcal{G}(\Omega)$ 中前 $2\times|W|$ 个度最大的节点
22. end while
23. return S;

算法 3 首先计算 $\mathcal{G}(\Omega)$ 中每一个节点的度, 并按照值的大小按照递减顺序排列 (第 1, 2 行). 然后初始化 3 个变量 W, $k_{\max}$ 和 S. 其中, W 表示 $\mathcal{V}$ 中度最大的节点, S 用来获取 $k_{\max}$-核 (第 3 行). 接下来用一个 while 循环来获取 $k_{\max}$-核 (第 4–23 行), 过程如下. 首先计算由 W 构成的时序子图中每一个节点的度, 并记录所有度中的最小值和最大值 (第 8, 9 行). 接下来利用区间 $[k_l, k_u]$ 中的值作为阶数对子图 $\mathcal{G}_W(\Omega)$ 进行核心分解 (第 10–20 行), 从而更新核心数最大值 $k_{\max}$ 和 $k_{\max}$-核 (第 15, 16 行). 之后将 W 的数目变为 $\mathcal{G}(\Omega)$ 中前 $2\times|W|$ 个度最大的节点进入下一轮迭代 (第 21 行). 最终算法 3 返回 $k_{\max}$-核的节点集 S (第 23 行). 值得注意的是, $k_{\max}$ 的值在每一轮迭代的过程中将会更新. 对于新一轮迭代产生的子图 $\mathcal{G}_W(\Omega)$, 算法 3 专注于发现阶数大于上一轮 $k_{\max}$ 的 k-核 (第 9 行).

算法分析: 算法 3 从局部时序图 $\mathcal{G}(\Omega)$ 中一些由度较大的节点 W 构成的子图中计算 $k_{\max}$-核. 对每一个小的子图 $\mathcal{G}_W(\Omega)$, 算法 3 通过运行核心分解过程来得到具有最大阶数的 k-核. While 循环的停止条件 (第 4 行) 可以确保正确地计算出 $k_{\max}$-核, 这是因为剩余节点, 即 $\mathcal{V} \setminus W$ 的最大度小于 $k_{\max}$, 因此这些节点组成的 k-核的阶数也会小于 $k_{\max}$,

从而保证正确性.

算法 3 的时间复杂度和空间复杂度均为 O(m), 证明如下. 假设算法 3 迭代的次数为 t. 由于算法 3 采用了子图节点数目指数增长的策略, 因此每一轮迭代过程中获得子图的节点数目的最大值分别为 $\left(\frac{1}{2}\right)^{t-1}\times n, \left(\frac{1}{2}\right)^{t-2}\times n,\ldots,n$, 构成一个几何序列. 在第 i 轮迭代过程中, 算法 3 花费了 $\mathrm{O}\left(\left(\frac{1}{2}\right)^{t-i}\times m\right)$ 时间和 O(m) 空间来执行核心分解过程. 因此算法 3 总的时间复杂度和空间复杂度均为 O(m).

与算法 2 对比, 虽然两者最差情况下的时间复杂度和空间复杂度相同, 但是算法 3 的运行时间要远远低于算法 2. 这是因为最稠密子图 $\mathcal{G}_S(\Omega)$ 的节点数目远远小于 $\mathcal{G}(\Omega)$ 的节点数目, 因此在迭代的过程中只有很少的子图会被检查.

4.2 基于动态规划的最优时间段发现算法

为了解决挑战 2, 本文将最优时间段发现问题进一步转化为最大斜率计算问题, 并提出一种动态规划算法可以在线性时间复杂度内计算出最优时间段.

给定时序图 $\mathcal{G}$ 中的一个节点集 S, 定义 $CDC[S][t_i]=d(\mathcal{G}_S([t_1,t_i]))$ 为时间段 $[t_1,t_i]$ 内的时序子图密度. 为了不失一般性, 假设 t_0 和 $CDC[S][t_0]$ 为 0. 则点集 $\{(t_0,CDC[S][t_0]),\ldots,(t_{|T|},CDC[S][t_{|T|}])\}$ 可以在笛卡尔坐标系统中绘制成一个曲线, 定义为 $CDC[S]$. 根据该曲线, 在时间段 $\mathcal{T}=[t_s,t_e]$ 内的密度突发值 $B(\mathcal{G}_S(\mathcal{T}))$ 可以进一步重写为:

$$B(\mathcal{G}_S(\mathcal{T}))=\frac{CDC[S][t_e]-CDC[S][t_s-1]}{t_e-(t_s-1)} \tag{7}$$

与曲线 $CDC[S]$ 中两个点 $(t_s-1,CDC[S][t_s-1])$ 和 $(t_e,CDC[S][t_e])$ 之间的斜率等价, 表示为 $slope(t_b,t_e)$ $(t_b=t_s-1)$.

定义 5. 极大 t_r-截断 θ-斜率. 给定一个密度曲线 $CDC[S]$ 和一个时间戳 t_r $(t_r\in[t_\theta,t_T])$, 极大 t_r-截断 θ-斜率 $MS(t_r)=\{\max(slope(t_b,t_r))|t_b\in[t_0,t_r-\theta]\}$.

根据定义 5, $MS(t_r)$ 表示在曲线 $CDC[S]$ 中以 t_r 作为结束时间且相应时间段不小于 θ 的最大斜率. 为了便于描述, 用 MS 表示所有 $MS(t_r)$ 的集合, 即 $MS=\{MS(t_r),t_r\in[t_\theta,t_{|T|}]\}$. 基于此, 可以将最大化突发值 $B(\mathcal{G}_S(\mathcal{T}))$ 题转化为寻找集合 MS 中的最大斜率问题, 因此需要获取集合 MS 中的所有元素值. 本文提出一种动态规划算法: 已知 $MS(t_r-1)$, 当计算 $MS(t_r)$ 时, 在曲线 $CDC[S]$ 中计算一个以时间戳 $t_r-\theta$ 作为结束时间的下凸包, 表示为 ST. 特别地, $MS(t_r)$ 为在 ST 中的点与曲线上的点 $(t_r,CDC[S][t_r])$ 之间的斜率值. 通过动态地维护 ST, 可以有效地获取 MS 中的所有元素值. 本文基于以下两个重要发现来对 ST 进行动态维护.

发现 1. 假设曲线 $CDC[S]$ 上的两个点 $(t_a,CDC[S][t_a])$ 和 $(t_b,CDC[S][t_b])$ 在维护的下凸包 ST 中, 且存在时间戳 $t_a<t_b<t_c$. 如果满足斜率 $slope(t_b,t_c)<slope(t_a,t_b)$, 则点 $(t_b,CDC[S][t_b])$ 可以从数组 ST 中移除.

发现 2. 假设曲线 $CDC[S]$ 上的两个点 $(t_a,CDC[S][t_a])$ 和 $(t_b,CDC[S][t_b])$ 在维护的下凸包 ST 中, 且 $t_a<t_b$. 若 $slope(t_b,t_r)\geqslant slope(t_a,t_b)$, 则点 $(t_a,CDC[S][t_a])$ 可以从数组 ST 中移除.

基于上述两个发现, 寻找最优时间段算法 findOTS 的伪代码如算法 4 所示.

算法 4. findOTS($\mathcal{G},S$).

输入: 时序图 $\mathcal{G}(\Omega)$, 节点集;
输出: 最优时间段 $\mathcal{T}$.

```
1. CDC[S][0] ← 0, CDC[S] ← ∅;
2. for each t_r in T
3.     computeCDC(S, t_r, G(Ω)); //计算 CDC[S][t_r]
4. end for
5. ST ← ∅, MS ← ∅, i_s ← 0, i_e ← −1;
```

6. for each t_r in $t_\theta : t_{|T|}$
7. while $i_s < i_e$ and $slope(ST[i_e], t_r - \theta) \leqslant slope(ST[i_e - 1], ST[i_e])$
8. $i_e \leftarrow i_e - 1$;
9. end while
10. $ST[++i_e] \leftarrow t_r - \theta$
11. while $i_s < i_e$ and $slope(ST[i_s], t_r) \geqslant slope(ST[i_s], ST[i_s + 1])$
12. $i_s \leftarrow i_s + 1$;
13. end while
14. $MS(t_r) \leftarrow slope(ST[i_s], t_r)$;
15. $MS \leftarrow MS \cup \{(MS(t_r), [ST[i_s], t_r])\}$
16. end for
17. $\mathcal{T} \leftarrow computeTimeSpan(MS)$; //计算 MS 中最大值对应的时间段
18. return $\mathcal{T}$;

算法 4 首先计算时间集中的每一个时间戳 t_r 处的时序子图密度 $CDC[S][t_r]$ (第 2–4 行). 接下来维护一个数组 ST 来存储可能与 t_r 处的点形成最大斜率 $MS(t_r)$ 的开始时间戳, 其中 i_s 记录 ST 的起始索引, i_e 记录 ST 的结束索引 (第 5 行). 对于时间段 $[t_\theta, t_{|T|}]$ 中的每一个时间戳 t_r, 算法 4 动态维护 ST, 并计算相应的 $MS(t_r)$ (第 6–16 行). 根据发现 1, 在第 1 个 while 循环中, 若 $i_s < i_e$ 且 $slope(ST[i_e], t_r - \theta) \leqslant slope(ST[i_e - 1], ST[i_e])$, 则将 i_e 的值减 1 (第 7–9 行). 若不存在该点则设置 $ST[++i_e] \leftarrow t_r - \theta$ (第 10 行). 接下来根据发现 2, 在第 2 个 while 循环中若 $i_s < i_e$ 且 $slope(ST[i_s], t_r) \geqslant slope(ST[i_s], ST[i_s + 1])$, 则将起始索引 i_s 的值加 1 (第 11–13 行). 当所有的迭代进行完之后即可获得时间段 $[t_\theta, t_{|T|}]$ 中的每一个时间戳 t_r 对应的 $MS(t_r)$ 以及相应的时间段, 并获取斜率值最大的元素对应的时间段 $\mathcal{T}$ (第 14–18 行).

算法分析: 算法 4 的第 11 行到第 15 行根据发现 2 可以正确的计算出以 t_r 作为结束时间且满足斜率最大的时间段, 因此可以正确地返回整个时间序列中长度不小于 θ 且斜率最大的时间段, 从而保证算法 4 的准确性. 算法 4 的时间复杂度和空间复杂度均为 O(|T|), 证明如下. 首先算法 3 需要 O(|T|) 时间来获取曲线 CDC 中所有点的值. 对于每一个时间戳 t_r, 索引 i_e 从 t_r 减少到 i_s; 索引 i_s 则从 t_θ 增加到 $t_{|T|}$. 考虑到算法中的所有 while 循环, 为索引 i_s 和 i_e 分配的时间均为 O(|T|). 因此算法 4 的时间复杂度为 O(|T|). 算法 4 主要存储 CDC、 ST 和 MS, 因此其空间复杂度为 O(|T|).

5 实验与分析

5.1 实验算法与平台

实验对所提算法 findSBS、findSBS-T 和 findSBS$^+$的性能进行分析. 其中, 算法 findSBS 为本文所提的基准算法, 该算法在挖掘季节突发性子图的过程中采用文献 [23] 提出的方法 findDSS 来寻找最优节点集. findSBS-T 在基准算法的基础上采用基于 key-核的最优节点集发现算法 findDSS$^+$来提升算法整体的效率. findSBS$^+$在 findSBS-T 算法的基础上采用 findOTS 算法来计算最优时间段集合. 实验使用一台 PC 设备, 配有 Linux 64 位操作系统, 32 GB 内存, 主频 3.50 GHz, 所有编程用 C++语言实现.

5.2 实验数据

本节采用的数据来自文献 [11], 共 5 个真实数据集, 包括 Lkml、 Enron、Twitter、 DBLP 和 WT. 其中, Lkml 为 2001–2011 年之间的 Linux 内核通信网络; Enron 为 1999–2003 年之间安然公司员工之间的通信网络; Twitter 跟踪了 2010 年 8 月–2010 年 10 月赫尔辛基的 Twitter 用户的活动; DBLP 为 1940–2018 年 2 月之间在 DBLP 网站中科研工作者之间的协作网络; WikiTalk (WT) 是一个代表了维基百科用户之间的交互的时序网络. 关于数据集的具体内在特征, 可见表 1 所示.

表 1 数据集特征

数据集	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert T\rvert$	跨度
Lkml	26 885	328 092	96	月
Enron	86 978	499 983	48	月
Twitter	4 605	23 942	93	天
DBLP	1 729 816	12 007 380	78	年
WT	1 094 018	4 010 611	2 321	天

5.3 实验设置

算法的效率取决于算法的运行时间, 而检测的子图质量可以用算法返回季节突发性子图的平均密度突发值 avgEDB 来衡量. 直观地, avgEDB 越大, 返回子图的质量也就越高.

实验中用到两个参数, 分别为 ω (季节数) 和 θ (时间段长度). 其中, ω 的值从 1, 2, 3, 4 中选择, 默认值为 2. θ 的值从 3, 4, 5, 6 中选择, 默认值为 4. 如果没有特别说明, 当改变某个参数时, 其他参数的值将被设置为他们的默认值.

5.4 算法有效性测试

为了考察随机选择的初始时间段数目对实验结果完整性的影响, 在数据集 Lkml 和 Twitter 中测试了不同初始时间段数目下, 算法挖掘出季节突发性子图数目的变化情况. 同时给出了采用所有可能的时间段作为初始集后算法返回的真实季节突发性子图数目. 实验结果如图 2 所示: 随着初始时间段数目的增加, 算法返回的季节突发性子图数目也会相应增加. 当随机选择的时间段数目达到 1 000 后, 返回的季节突发性子图数目已经接近真实值. 由此可见, 并不是将所有可能的时间段作为初始集都是必要的. 为了平衡算法运行时间和返回结果的质量带来的影响, 接下来的实验中随机选择 600 个时间段对算法进行初始化.

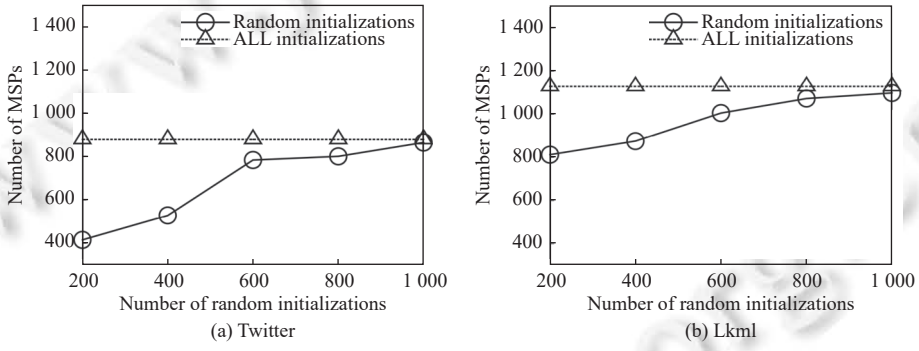


图 2 随机初始化数目对结果的影响

图 3 给出了在默认参数设置下, 算法 findSBS 和 findSBS-T 返回子图的质量对比. 在其他参数设置下也可以得到类似的结果.

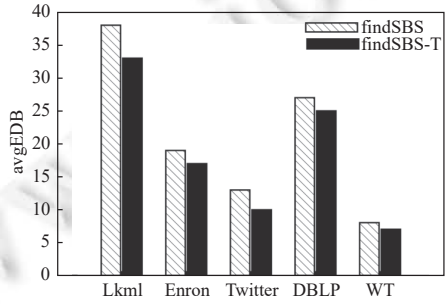


图 3 不同数据集下算法的有效性对比

从图 3 可以看出, 算法 findSBS-T 返回子图的 avgEDB 值略低于算法 findSBS 返回子图的 avgEDB 值. 这是因为 findSBS-T 在寻找最优节点集阶段利用 key-核来定位可能包含密度最大稠密子图的一些小的稠密子图中, 然后在小的稠密子图中进行挖掘, 导致算法的近似率有所降低, 挖掘出子图的 avgEDB 值也会相应降低.

图 4 和图 5 给出了当参数发生变化时, 算法 findSBS 和 findSBS-T 在数据集 Enron 和 DBLP 中返回子图的质量变化趋势. 在其他数据集中也可以得到类似的结果. 其中, 当改变一个参数时, 将保持其他参数为默认值. 实验结果如下: avgEDB 值随着 ω 或者 θ 的增加而减少. 这是因为在计算最优时间段集合的过程中, 随着 ω 或者 θ 值的增大, 子图的密度突发值呈递减趋势, 从而使得最终计算出所有子图的 avgEDB 值也相应地减少.

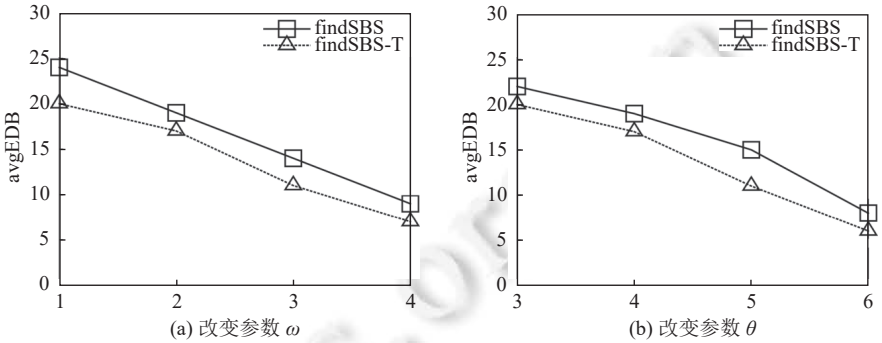


图 4 在 Enron 数据集中参数变化对算法有效性影响

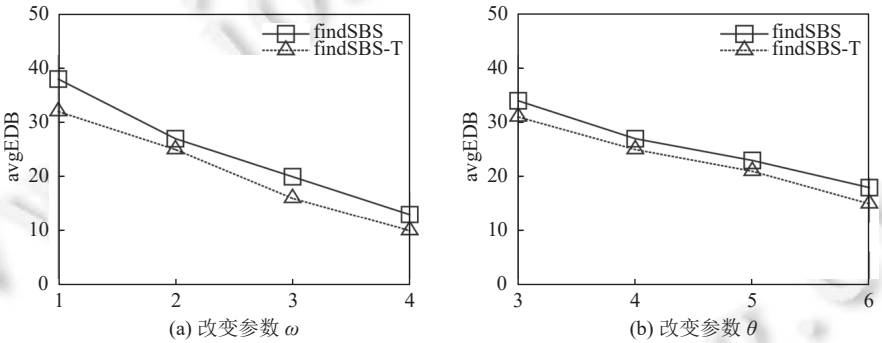


图 5 在 DBLP 数据集中参数变化对算法有效性影响

5.5 算法有效性测试

图 6 给出了算法 findSBS、findSBS-T 和 findSBS$^+$在默认参数下的运行时间对比. 在其他参数设置下也具有类似的结果. 从图 6 可以看出, 算法 findSBS$^+$的时间效率与 findSBS 和 findSBS-T 相比得到了较大的提升. 例如在 WT 数据集中, findSBS$^+$算法的时间效率比 findSBS-T 提升了超过 2 个数量级; 在 DBlP 数据集中, findSBS-T 的时间效率比 findSBS 提升了近 6 倍. 这是因为 findSBS-T 算法采用了本文所提的基于 key-核的最稠密子图挖掘算法, 该算法可以快速地定位到可能包含密度最大稠密子图的一些小的稠密子图中, 从而减小搜索空间, 提升了算法的效率, 在规模较大的数据集中体现得更加明显. findSBS$^+$算法则进一步采用本文提出的动态规划算法来寻找最优时间段集合, 可以有效地将该部分算法的时间复杂度从 $O(|T^2|)$ 减少到 $O(|T|)$. 由此可见, 新算法 findSBS$^+$在时间效率上取得了良好的领先结果.

图 7 和图 8 给出了当参数发生变化时, 算法 findSBS-T 和 findSBS$^+$在数据集 Enron 和 DBLP 中时间效率的变化趋势. 其中, 当改变一个参数时, 保持其他参数为默认值. 在其他数据集中也具有类似的结果. 从图中可以看出算法 findSBS-T 和 findSBS$^+$的时间效率随着 ω 或者 θ 的增加而增加. 这是因为当 ω 或者 θ 的值增加时, 计算最优节点集过程中局部时序图的规模也就越大, 算法的运行时间也就越长.

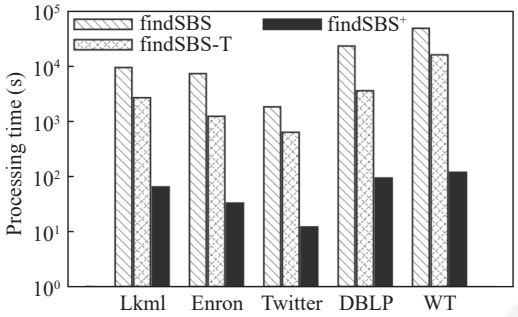


图 6 不同数据集下算法的时间效率对比

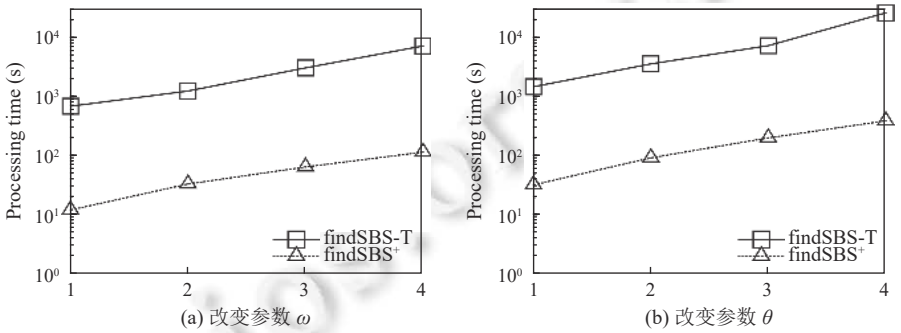


图 7 在 Enron 数据集中参数变化对算法运行时间的影响

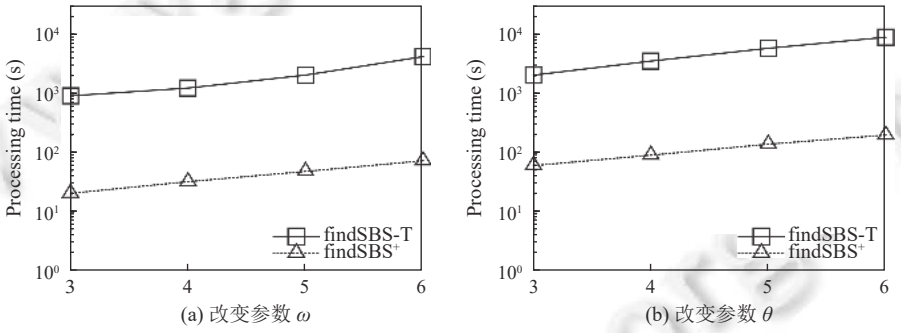


图 8 在 DBLP 数据集中参数变化对算法运行时间的影响

5.6 算法扩展性测试

后文图 9 给出了算法 findSBS-T 和 findSBS$^+$在默认参数下在数据集 WT 上的可扩展性. 在其他数据集中也可以得到类似的结果. 实验通过随机抽取 10%–100% 的时序边以及 10%–100% 的时间戳来分别生成 10 个时序子图, 并评估算法 findSBS-T 和 findSBS$^+$在这些子图上的运行时间. 从图 9 可以看出, 只有算法 findSBS$^+$的运行时间随着时序边或时间戳数目的增加而平稳增加, 这表明 findSBS$^+$在处理大型时序网络时具有良好的扩展性.

5.7 内存消耗测试

表 2 给出了算法 findSBS-T 和 findSBS$^+$在默认参数下的内存消耗情况. 从表中可以看出 findSBS-T 和 findSBS$^+$的内存消耗都大于原始是时序图的大小, 且 findSBS$^+$的内存消耗大于 findSBS-T. 这是因为算法 findSBS-T 需要存储时序图中每个节点的度来计算 $k_{\max}$-核, findSBS$^+$则需要进一步的存储 *MS* 和 *CDC* 来计算最优时间段集合. 然而在实际实验的过程中, 算出一个季节突发子图之后, 可以释放掉计算该子图过程中 *MS* 和 *CDC* 占用的内存, 而减

少内存开销. 因此在规模比较大的数据集中, findSBS$^+$的内存开销通常要小于原始图大小的 3 倍. 该实验结果证明算法 findSBS$^+$可以获得接近线性的空间复杂度.

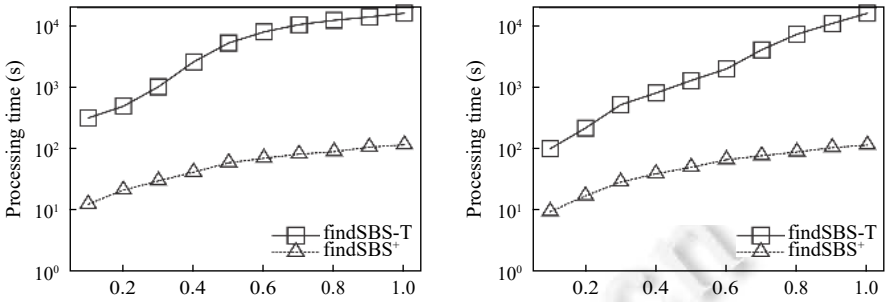


图 9 在 WT 数据集中算法可扩展性测试

表 2 内存开销对比

数据集	Graph	findSBS-T	findSBS$^+$
Lkml	20.1 MB	32.5 MB	54.1 MB
Enron	53.3 MB	85.3 MB	122.4 MB
Twitter	7.2 MB	10.1 MB	18.3 MB
DBLP	1 089.5 MB	1.4 GB	2.1 GB
WT	324.5 MB	532.7 MB	872.8 MB

5.8 案例分析

图 10 给出了在默认参数设置下从 Twitter 数据集中挖掘出的一个包含来自阿尔托创业协会 (@aaltoes) 和阿尔托设计工厂 (@AaltoGarage) 成员的团体. 该团体包含 9 个 Twitter 用户, 图中每一个子图对应一个图快照, 反映在该时刻这些用户之间的交互关系. 从图 10 中可以看出在时间段 2010.8.12–8.15 和时间段 2010.9.25–9.28 内这些 Twitter 用户之间的联系会越来越紧密, 表明这两个组织在该时间段集合内具有商业合作, 引起成员之间的联系会更加密切.

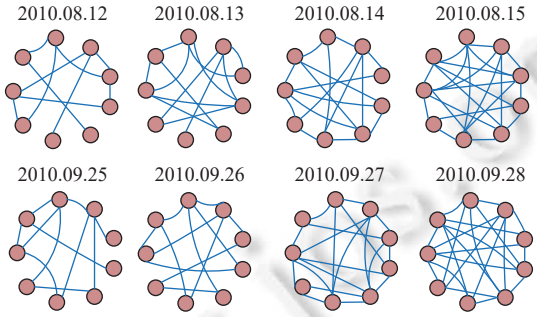


图 10 在 Twitter 数据集中找到的 SBS

本节同时在 Enron 数据集中进行了案例分析. Enron 包括 1999–2003 年安然公司员工之间发送的电子邮件情况. 安然是一家位于美国德克萨斯州休斯顿的能源服务公司, 犯下了有史以来最大的财务造假骗局之一. 在这场骗局中, 安然的高管们利用会计手段虚增公司的收入, 欺骗了投资者和监管机构, 在 2001 年 7 月欺诈行为被曝光, 并于 2001 年 12 月 2 日申请破产. 在默认参数设置下, 图 11 给出了挖掘出的一个关于安然高管之间交互行为的稠密子图, 该子图反映了安然的高管们在时间段 2001.7–2001.10 和时间段 2002.1–2002.4 期间联系比较紧密. 这是因为 2001 年 7 月欺诈行为曝光后, 安然高管们紧急对该欺诈事件进行相应处理. 此后, 在 2001 年 12 月 2 日公司申

请破产, 安然高管开始处理破产后续的相关事宜. 根据挖掘出的季节突发子图可以发现那些需要为安然公司倒闭负责的人.

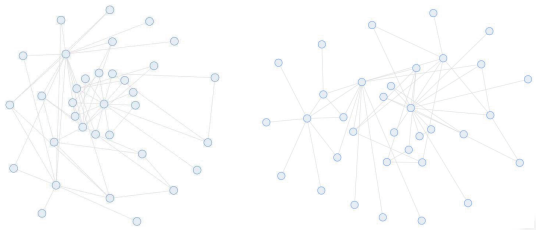


图 11 在 Enron 数据集中找到的 SBS

6 总 结

本文研究了时序图上的季节突发性子图挖掘问题, 并提出了一种极大 (ω,θ)-稠密子图模型对季节突发性子图进行建模. 为了挖掘出时序图中的所有季节突发性子图, 本文将该类挖掘问题转换成一个混合的整数规划问题, 包括挖掘最稠密子图和寻找突发值最大化时间段集合两个子问题, 并给出了有效的解决方案. 接着提出一种基于 k-核的改进算法来定位最稠密子图可能存在的位置, 然后在这些小的稠密子图中挖掘最稠密子图. 进一步提出一种动态规划算法来寻找突发值最大化时间段集合, 有效地提升算法性能. 最后, 在真实的时序网络进行了综合实验, 实验结果证明了所提算法的效率、可扩展性和有效性.

当前的研究没有考虑时序图动态更新的场景, 在接下来的研究工作中将进一步考虑率在时序图动态更新的场景下挖掘季节突发性子图. 其次, 人们往往对前 k 个突发值最大的结果感兴趣, 接下来会进一步将本文的工作扩展到 top-k 季节突发性子图挖掘中.

References:

[1] Broder A, Kumar R, Maghoul F, Raghavan P, Rajagopalan S, Stata R, Tomkins A, Wiener J. Graph structure in the Web. Computer Networks, 2000, 33(1–6): 309–320. [doi: 10.1016/S1389-1286(00)00083-9]

[2] Boyd DM, Ellison NB. Social network sites: Definition, history, and scholarship. Journal of Computer-mediated Communication, 2007, 13(1): 210–230. [doi: 10.1111/j.1083-6101.2007.00393.x]

[3] Rual JF, Venkatesan K, Hao T, *et al*. Towards a proteome-scale map of the human protein-protein interaction network. Nature, 2005, 437(7062): 1173–1178. [doi: 10.1038/nature04209]

[4] Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In: Proc. of the 19th ACM Int'l Conf. on Knowledge Discovery and Data Mining. Chicago: ACM, 2013. 104–112. [doi: 10.1145/2487575.2487645]

[5] Cheng J, Ke YP, Chu SM, Özsu MT. Efficient core decomposition in massive networks. In: Proc. of the 27th IEEE Int'l Conf. on Data Engineering. Hannover: IEEE, 2011. 51–62. [doi: 10.1109/ICDE.2011.5767911]

[6] Uno T. An efficient algorithm for solving pseudo clique enumeration problem. Algorithmica, 2010, 56(1): 3–16. [doi: 10.1007/s00453-008-9238-3]

[7] Tsourakakis C. The K-clique densest subgraph problem. In: Proc. of the 24th Int'l Conf. on World Wide Web. Florence: Int'l World Wide Web Conf. Steering Committee, 2015. 1122–1132. [doi: 10.1145/2736277.2741098]

[8] Mitzenmacher M, Pachocki J, Peng R, Tsourakakis C, Xu SC. Scalable large near-clique detection in large-scale networks via sampling. In: Proc. of the 21th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Sydney: ACM, 2015. 815–824. [doi: 10.1145/2783258.2783385]

[9] Vanhems P, Barrat A, Cattuto C, Pinton JF, Khanafer N, Régis C, Kim BA, Comte B, Voirin N. Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. PLoS One, 2013, 8(9): e73970. [doi: 10.1371/journal.pone.0073970]

[10] Fournet J, Barrat A. Contact patterns among high school students. PLoS One, 2014, 9(9): e107878. [doi: 10.1371/journal.pone.0107878]

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

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

[13] Ma S, Hu RJ, Wang LS, Lin XL, Huai JP. Fast computation of dense temporal subgraphs. In: Proc. of the 33rd IEEE Int'l Conf. on Data Engineering. San Diego: IEEE, 2017. 361–372. [doi: 10.1109/ICDE.2017.95]

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

[15] Xu J, Zhang QZ, Zhao X, Lü P, Li TS. Survey on dynamic graph pattern matching technologies. Ruan Jian Xue Bao/Journal of Software, 2018, 29(3): 663–688 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5444.htm [doi: 10.13328/j.cnki.jos.005444]

[16] Ma S, Hu RJ, Wang LS, Lin XL, Huai JP. An efficient approach to finding dense temporal subgraphs. IEEE Trans. on Knowledge and Data Engineering, 2020, 32(4): 645–658. [doi: 10.1109/TKDE.2019.2891604]

[17] Li RH, Su J, Qin L, Yu JX, Dai QQ. Persistent community search in temporal networks. In: Proc. of the 34th IEEE Int'l Conf. on Data Engineering. Paris: IEEE, 2018. 797–808. [doi: 10.1109/ICDE.2018.00077]

[18] Zhang QZ, Guo DK, Zhao X, Li XY, Wang X. Seasonal-periodic subgraph mining in temporal networks. In: Proc. of the 29th ACM Int'l Conf. on Information & Knowledge Management. ACM, 2020. 2309–2312. [doi: 10.1145/3340531.3412091]

[19] Liu XM, Ge TJ, Wu YH. Finding densest lasting subgraphs in dynamic graphs: A stochastic approach. In: Proc. of the 35th IEEE Int'l Conf. on Data Engineering. Macao: IEEE, 2019. 782–793. [doi: 10.1109/ICDE.2019.00075]

[20] Rozenshtein P, Bonchi F, Gionis A, Sozio M, Tatti N. Finding events in temporal networks: Segmentation meets densest-subgraph discovery. In: Proc. of the 2018 IEEE Int'l Conf. on Data Mining. Singapore: IEEE, 2018. 397–406. [doi: 10.1109/ICDM.2018.00055]

[21] Gallo G, Grigoriadis MD, Tarjan RE. A fast parametric maximum flow algorithm and applications. SIAM Journal on Computing, 1989, 18(1): 30–55. [doi: 10.1137/0218003]

[22] Qin L, Li RH, Chang LJ, Zhang CQ. Locally densest subgraph discovery. In: Proc. of the 21th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Sydney: ACM, 2015. 965–974. [doi: 10.1145/2783258.2783299]

[23] Epasto A, Lattanzi S, Sozio M. Efficient densest subgraph computation in evolving graphs. In: Proc. of the 24th Int'l Conf. on World Wide Web. Florence: Int'l World Wide Web Conf. Steering Committee, 2015. 300–310. [doi: 10.1145/2736277.2741638]

[24] Charikar M. Greedy approximation algorithms for finding dense components in a graph. In: Proc. of the 3rd Int'l Workshop on Approximation Algorithms for Combinatorial Optimization. Saarbrücken: Springer, 2000. 84–95. [doi: 10.1007/3-540-44436-X_10]

[25] Bahmani B, Kumar R, Vassilvitskii S. Densest subgraph in streaming and MapReduce. Proc. of the VLDB Endowment, 2012, 5(5): 454–465. [doi: 10.14778/2140436.2140442]

[26] Yang YJ, Yu JX, Gao H, Pei J, Li JZ. Mining most frequently changing component in evolving graphs. World Wide Web, 2014, 17(3): 351–376. [doi: 10.1007/s11280-013-0204-x]

[27] Huang SL, Fu AWC, Liu RF. Minimum spanning trees in temporal graphs. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Melbourne: ACM, 2015. 419–430. [doi: 10.1145/2723372.2723717]

[28] Gurukar S, Ranu S, Ravindran R. COMMIT: A scalable approach to mining communication motifs from dynamic networks. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Melbourne: ACM, 2015. 475–489. [doi: 10.1145/2723372.2737791]

[29] Wu HH, Cheng J, Lu Y, Ke YP, Huang YZ, Yan D, Wu HJ. Core decomposition in large temporal graphs. In: Proc. of the 2015 IEEE Int'l Conf. on Big Data. Santa Clara: IEEE, 2015. 649–658. [doi: 10.1109/BigData.2015.7363809]

[30] Pan MJ, Li RH, Zhao YH, Wang GR. Fast temporal cycle enumeration algorithm on temporal graphs. Ruan Jian Xue Bao/Journal of Software, 2020, 31(12): 3823–3835 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5968.htm [doi: 10.13328/j.cnki.jos.005968]

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

附中文参考文献:

[14] 王一舒, 袁野, 刘萌, 王国仁. 大规模时序图数据的查询处理与挖掘技术综述. 计算机研究与发展, 2018, 55(9): 1889–1902. [doi: 10.7544/issn1000-1239.2018.20180132]

[15] 许嘉, 张千桢, 赵翔, 吕品, 李陶深. 动态图模式匹配技术综述. 软件学报, 2018, 29(3): 663–688. http://www.jos.org.cn/1000-9825/5444.

htm [doi: 10.13328/j.cnki.jos.005444]
[30] 潘敏佳, 李荣华, 赵宇海, 王国仁. 面向时序图数据的快速环枚举算法. 软件学报, 2020, 31(12): 3823–3835. http://www.jos.org.cn/1000-9825/5968.htm [doi: 10.13328/j.cnki.jos.005968]

张千桢(1992－), 男, 博士, 讲师, 主要研究领域为子图搜索, 图数据挖掘, 大图数据管理.

赵翔(1986－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为知识图谱, 先进数据分析.

郭得科(1980－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为网络计算与系统, 分布式计算与系统.