基于滑动时间窗的稠密子图发现算法研究

田朝霞 张 俊 陈 旭 曲贤菲
(大连海事大学信息科学技术学院 辽宁 大连 116026)

摘 要 针对在滑动时间窗中发现稠密子图的问题,提出一种有效的动态算法,结合时间窗将网络时间线划分为 k 个非重叠的间隔,间隔内包含最大密度的子图。算法输入是一个边流,输出是一系列稠密子图及相应的时间间隔。现有技术在图更新时需要迭代整个图,所提算法仅影响图的有限区域,只需要局部更新稠密子图。结合理论分析,证明了该算法比基线 KGOPTDP 和 KGOPTDS 更快。多组数据集上的实验结果表明,该算法具有很高的效率和很好的扩展性,可用于处理大规模时态图。

关键词 时态图 稠密子图 滑动时间窗

中图分类号 TP301.6 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2021.07.047

DENSE SUBGRAPH DISCOVERY ALGORITHM BASED ON SLIDING TIME WINDOW

Tian Zhaoxia Zhang Jun Chen Xu Qu Xianfei
(*School of Information Science and Technology*, *Dalian Maritime University*, *Dalian 116026*, *Liaoning*, *China*)

Abstract Aiming at the problem of finding dense subgraphs in the sliding time window, this paper proposes an effective dynamic algorithm, which combines the time window to divide the network timeline into k non-overlapping intervals, and the interval contained subgraph with the maximum density. The input to the algorithm was an edge stream, and the output was a series of dense subgraphs and corresponding time intervals. In the state-of-the-art solutions, the entire graph needs to be iterated when the graph is updated, our algorithm only affects the limited region of the graph, only the dense subgraph needs to be locally updated. Combined with theoretical analysis, it is proved that the algorithm is faster than the baseline KGOPTDP and KGOPTDS. The experimental results on multiple sets of data sets show that the algorithm has high efficiency and good scalability and can be used to process large-scale temporal graphs.

Keywords Temporal graph Dense subgraph Sliding time window

0 引 言

当今,动态性已经成为建模为图和网络的数据分析系统和应用程序的明显特征,如社交网络分析、生物数据分析、推荐系统和路径规划[1]。因此,动态网络引起了行业和学术界的重视,实际上,通常使用动态网络的各种替代术语,如时态网络、动态图、演化网络、时间依赖图和图流等[1-2]。现实世界的网络本质上是动态变化的,实体(节点)不断建立新的关系(边),移除旧的关系。分析网络的时间维度可以提供有关其结构和功能的有价值的见解,例如,可以揭示时间模式、概念漂移、周期性和时间事件等[3]。

稠密子图发现是一个基本的图挖掘问题,已经成为广泛的数据分析任务中的原语,如社区检测[4]、事件检测[5]、链接垃圾邮件检测[6]和计算生物学[7]等。在理论计算机科学中已经广泛研究了稠密子图发现问题,由于该问题与实际应用的相关性,在社区数据挖掘中引起了相当大的关注。社交网络中紧密连接的用户可能对应于社区,也就是有相似兴趣或与同一组织(例如大学或公司)相关联的用户组。突然在推文中共同出现的城市、个人和公司名称等实体可能表明涉及相应实体的事件正在发生[2]。

在现实应用中,图数据会随时间发生动态变化,也就是所谓的时态图,时态图有两种变化形式:一种是图的拓扑结构变化,图中的节点和边随时间发生插入和

收稿日期:2019-10-21。**田朝霞**,硕士,主研领域:时态数据库,时态稠密子图。**张俊**,教授。**陈旭**,硕士。**曲贤菲**,硕士。

删除,从而导致图的结构发生变化;另一种是图的内容变化,图中的节点和边所表征的数据值或者对象内容发生变化。本文更多关注时态图的拓扑结构变化对稠密子图发现结果的影响。当处理时态网络时,首先要确定如何处理时间维度,即识别哪段时间是可以发现稠密子图的时间间隔。本文算法不是先验地定义这些间隔,而是研究自动识别稠密子图的间隔的问题,因此能够在时态网络中发现一系列稠密子图,捕获网络生命周期中发生的有趣事件的演变。

为了得到更具体的理解,请考虑以下几种场景的事例:

(1) 许多不同机构的一组研究人员正在合作开展一个大型项目,小组成员参加他们各自的日常活动,但每隔几周或几个月,在项目会议或可交付的截止日期之前,小组成员间会参与许多与项目有关的活动。

(2) 一群 Twitter 用户对某一技术产品感兴趣,他们积极地在博客评论并互相评论彼此的帖子,尽管这之间的相互联系非常稀疏,但持续时间长,并且在新产品发布之后显著增强。

(3) 在线社交媒体中的故事识别[8]通过实体之间的稠密子图自动发现新兴故事,了解故事如何随时间的推移而演变。例如,当一个故事消失另一个故事出现时,实体间的一个稠密子图消失,另一个出现。通过将时态网络的时间线分割成区间,并识别每个间隔中的稠密子图,可以捕捉主要故事随时间的演变。

对于时态网络,只有很少的论文提出了发现时间上连续的最稠密子图的方法。与本文工作最相似的是找到所有或 k 个快照中存在的重子图[8],另一个相关工作侧重在时态网络中找到由 k 个散乱区间覆盖的稠密子图[9]。然而这两种方法都只发现了一个最稠密子图。

本文的目标是研究在滑动时间窗中发现稠密子图的问题,为了实现这一目标,采用动态稠密子图的现有工作设计一个快速的近似算法,结合滑动时间窗将时态网络划分成 k 个非重叠的间隔,使得间隔内能够发现最大稠密子图。

本文的主要思路如下:

(1) 研究了滑动时间窗中的最大稠密子图问题,利用动态稠密子图的现有工作,设计一个快速的动态近似算法。

(2) 结合时间窗大小将时态网络的时间线分割成 k 个非重叠间隔,每个间隔内能够发现具有最大密度的子图。

(3) 在真实数据集上进行实验,验证了本文方法的有效性,并解释结果的合理性。

1　相关工作

在稠密子图中对图进行分区是一个公认的问题,现有研究大部分采用密度定义的平均度概念[10],在此定义下,可以在多项式时间内找到最稠密子图[11]。另外,Charikar [12]开发了一个近似贪婪算法,它在图大小的线性时间内运行。Epasto 等[2]开发了在流式场景中维护平均度最稠密子图的算法。其他密度定义通常难以通过有效的启发式方法得到问题的近似解。

现有研究大都关注动态图,建立节点和边添加或删除的模型,研究网络演化的不同方面,包括稠密集群的演化[13]。另一种建模时态图的经典方法是图快照,分别在每个快照中(或合并先前快照的信息)找到结构,然后总结已发现结构的历史行为[14]。这些方法侧重于快照中发现的稠密结构的时间一致性,并假设已经给出了快照。与此不同,本文工作是将瞬时交互聚合到任意长度的时间轴分区中。

许多动态图研究致力于事件检测问题,Akoglu 等[15]涵盖了该主题的最新研究,大部分研究侧重比较不同的图快照,目的是检测图结构发生重大变化的快照。与其他事件检测问题相比,本文研究主要目标是同时发现事件的子图和相应的时间间隔。与动态图事件检测类似,构建一个包含时态信息的静态图(或一系列静态图)是动态图研究的常用方法, Rosvall 等[16]的动态聚类方法将时间数据嵌入到静态图中,使用历史时态数据学习二跳路径的概率以产生聚类。但发现的聚类在时间上是全局的,没有检测到相关的突发时间间隔,并且没有时间约束。

与本文工作最相似的是 Rozenshtein 等[9]的研究,考虑了在时态网络中发现最稠密子图的问题,然而他们并不旨在创建时间分区,且他们发现的是边出现在 k 个短间隔内的单一的稠密子图。本文工作是划分间隔并仅考虑跨越连续间隔的图。Semertzidis 等[8]考虑一组图快照,并搜索一个或多个间隔诱导的单个重子图,探索持久性重子图问题的不同公式,包括最大平均密度。Bogdanov 等[17]提出了在时间演变网络中挖掘重子图的方法,但其仍然是基于网络快照,因此对边界量化效果很敏感,且发现的重子图的概念基于边权重而不是基于密度。

总之,与已有研究相比,本文希望能够在时态网络中发现一系列稠密子图,捕获网络生命周期中发生的有趣事件的演变。

2 时态稠密子图问题定义

给定一个无向图 $G=(V,E,\mathcal{T})$，有 $n=|V|$ 个顶点和 $m=|E|$ 条边，$\mathcal{T}=\{0,1,\cdots,t_{\max}\}\subseteq \mathbf{N}$，是一个离散的时间域，顶点 $u\in V$ 邻居定义为 $N(v)=\{u|(v,u\in E)\}$，顶点的度定义为 $d(v)=|N(v)|$。给定一个时间间隔 $T=\{t_1,t_2\}\subseteq\mathcal{T}$，令 $H=(V[T],E[T])$ 是由时间边的集合 $E[T]=\{(u,v)|(u,v,t)\in E\wedge t\in T\}$ 诱导形成的子图，$V[T]$ 中顶点 v 的内度定义为 $d(v)=|N(v)\cap V[T]|$。最后，对于顶点的子集 $V[T]\subseteq V$，定义密度 $\rho_{V[T]}$：

$$\rho_{V[T]}=\frac{|E[T]|}{|V[T]|} \tag{1}$$

定义 1 时态稠密子图(Temporal Densest Subgraph)给定一个无向图 $G=(V,E,\mathcal{T})$，稠密子图 H^* 是使密度函数最大化的顶点的集合，即：

$$H^*=\arg\max_{V[T]\subseteq V}\rho_{V[T]} \tag{2}$$

如果有算法 A 计算子集 $V[T]\in V$ 使 $\rho_{V[T]}\geqslant\frac{1}{\alpha}\rho_{H^*}$，其中 $H^*\subseteq V$ 是 G 的稠密子图，则算法 A 计算得到稠密子图的 α-近似解。

下面介绍与稠密子图相关的其他概念：图核、核心分解和顶点的诱导核心子图。

定义 2 j 核(j-core)给定无向图 $G=(V,E,\mathcal{T})$，一个整数 j，G 的 j-core 是顶点 $C\subseteq V$ 的子集，因此每个顶点 $v\subseteq C$ 都有内度 $d_C(v)\geqslant j$。

定义 3 核心分解(Core Decomposition)一个图 $G=(V,E,\mathcal{T})$ 的核心分解是核心的嵌套序列 $\{C_i\}$：

$$V=C_0\supseteq C_1\supseteq\cdots\supseteq C_{\not}\supseteq\varnothing \tag{3}$$

其中每个 C_i 都是一个 j-core。

定义 4 核数(Core number)给定一个图 $G=(V,E,\mathcal{T})$ 的核心分解 $V=C_0\supseteq C_1\supseteq\cdots\supseteq C_{\not}\supseteq\varnothing$，一个顶点的核数 $\kappa(v)$ 是最大的 j，使得 $v\in C$ 和 C 是一个 j-core。通过重写符号，核心 C 的核数 $\kappa(C)$ 是最大的 j，其中 C 是 j-core。

另外，使用 $\kappa_H(v)$ 表示由 H 诱导的子图中顶点 v 的核数，$G(H)=(H,E(H))$ 的子图的最大核(或主核)用 $C_{\not}(H)$ 表示，G 的主核仅用 $C_{\not}$ 表示。

定义 5 诱导核心子图(Induced Core Subgraph)给定一个图 $G=(V,E,\mathcal{T})$ 和一个顶点 $v\in V$，由 $\prod(v)$ 表示的顶点 v 的诱导核心子图，是包含顶点 v 的最大连通子图，$\prod(v)$ 中所有顶点的核数等于 $\kappa(v)$。

换言之，诱导子图包含可从 v 到达并具有相同核数 $\kappa(v)$ 的所有顶点。

本文在滑动时间窗边流中处理图，根据这个模型，问题的输入是边流形式，边 e_i 是流的第 i 个元素，即边 e_i 有时间戳 i，滑动窗口 $W_t(x)$ 定义为在时刻 t 长度为 x 的 e_{t-x+1} 和 e_t 间到达的所有边的集合：

$$W_t(x)=\{e_i,i\in[t-x+1,t]\} \tag{4}$$

对于每条边 $e_i=(u,v)$，u 和 v 出现在时刻 i，使用 $V_t(x)$ 表示时刻 t 出现在长度为 x 的滑动窗口中的顶点集，在时刻 t 长度为 x 的滑动窗口中的图定义为 $G_t(x)=(V_t(x),W_t(x))$。

下面定义本文研究的问题，即以滑动时间窗进行分割，在时态网络中找到一系列稠密子图。

给定一个时态图 $G=(V,E,\mathcal{T})$ 和一个图流 $\{e_i\}$，找到 k 个间隔的集合 $S=\{(I_e,H_e)\}$，$e=1,2,\cdots,k$ 使得 $\{I_e\}$ 是不相交的间隔并且 $\sum_{e=1}^{k}d(H_e)$ 是最大的。

3 时态稠密子图发现算法

本文算法的目标是在线更新稠密子图，但动态流更新稠密子图有两个问题：① 如何减少稠密子图的搜索空间；② 如何分割整个图得到子图。

为了解决上述问题，本文提出两个假设性判断。首先，由于稠密子图由相对高度的顶点形成，可以通过仅跟踪这些高度顶点找到稠密子图；其次，这些子图可以在图更新时进行局部更新，而不影响图的其他部分。

基于这两点，本文提出一种新算法，通过仅考虑高度节点减少搜索空间，并将整个图划分为更小的子图，从每个子图中找到稠密子图。

3.1 存储高度顶点的数据结构

提出一个增量数据结构，存储一个强连接的子图，用 D 表示，子图维护以下不变量：① D 内的每个顶点 $v\in D$ 的核数 $\kappa_D(v)$ 等于 D 的主核($C_e(D)$)；② D 内所有的顶点是连通的。这些不变量确保 D 中的所有顶点具有相同的核数，根据定义这是 D 的主核。在处理图更新时，D 会维护这些不变量。

图中的高度顶点被分配给 D，由于这些顶点没有紧密连接，最终可能会出现在不同的 D 中，将这些不连接的 D 存储在一个被称为包的数据结构中，用 B 表示。包确保每个 D 是顶点不相交的，此外，包提供了额外操作：在一组 D 中提取最稠密的 D，提取 D 的密度是最大密度，最大密度是将高度顶点与低度顶点分开的阈值，这个最大密度的估计值表示为 $\tilde{\rho}_{H^*}$。包是一个超图，包含一组 D 之间的所有边。本文利用核心

分解算法维护包中顶点的所有核数,包中每个节点的核数用于确保每个顶点具有最大可能的核数。图 1 表明了包需要保持顶点的核心数。图显示包中包含两个 D,可以通过核心分解产生具有更大主核数的 D。接下来,定义算法在图更新时同步更新此数据结构。

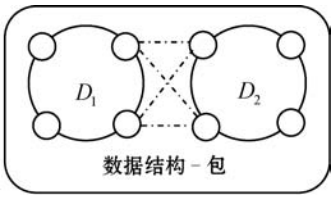


图 1　存储高度顶点的数据结构图

本文算法需要访问每个顶点的邻域信息。在给定顶点上执行以下三个查询:① 提取顶点度。② 提取顶点的所有邻居顶点,解决这两个查询需要建立顶点图,存储所有顶点的邻域信息,将信息存储在散列图中,其中:键是顶点标识符;值是每个顶点的邻居。顶点图以恒定时间执行搜索和更新操作。③ 给定密度 $\tilde{\rho}_{H^*}$,提取度大于给定密度的所有顶点。对于此查询,需要按度对顶点进行排序,使用 bin 排序对顶点进行排序,在恒定时间内提取度大于密度 $\tilde{\rho}_{H^*}$ 的顶点。

3.2　添加顶点/边操作

1) 添加顶点操作。如第 2 节所述,更新以边流的形式出现,定义了添加顶点算法,作为添加边算法的子程序,当添加边中至少一个顶点是高度顶点时触发此算法,需要考虑两种情况:(1) 包已经包含高度顶点;(2) 包不包含高度顶点。在这两种情况下,目标是将新顶点添加到一个 D。

算法 1 描述了添加顶点的算法,对于第一种情况,算法扫描包找到包含顶点的 D 并返回它,对于第二种情况,算法首先识别候选 D,然后将顶点分配给其中一个候选 D,候选 D 具有小于或等于新顶点内度的主核数($\kappa_u(D) \geqslant C_e(D)$),在候选 D 中新顶点被分配给具有最大内度 $d_u(D)$ 的 D。

顶点加入到 D 中,D 的核数可能会增加,需要删除核数低于 D 的主核的顶点,通过类似 bin 排序的方法基于度对顶点进行排序,可以在线性时间内有效更新稠密子图。

算法 1　添加顶点算法

```
1.  H* ← ∅;
2.  for D_i ∈ B do                                  /* 识别候选 D */
3.  if u ∈ D_i then                                 /* 找到包含顶点的 D */
4.     return D_i;
5.   if (d_{D_i}(u) ≥ C_e(D_i) and ρ_{H*} < ρ_{D_i}) then
6.     H* ← D_i;
                  /* 候选 D 具有小于或等于新顶点内度的主核数 */
7.  if H* = ∅ then
8.     H* ← {u};
9.  else H* ← H* ∪ {u}
10.  return H*;
```

2) 添加边操作。在这种情况下,只有当添加边的至少一个顶点是高度顶点时,才会影响包的状态,需要考虑两种情况:(1) 只有一个顶点是高度顶点;(2) 两个顶点都是高度顶点。对于第一种情况,算法利用添加顶点的方法把顶点添加到 D 的包中;对于第二种情况,当两个顶点都添加到 D 的包中时,算法验证主核是否存在于包中,当两个顶点添加到同一个 D 时,算法将新边添加到同一个 D 并确保不变量保持不变。相反,当两个顶点添加到不同的 D 时,算法验证顶点是否存在于彼此的诱导子图中,并修复两个顶点的主核。算法 2 描述了添加边的过程。

算法 2　添加边算法

```
1.  if ((d(u) < ρ̃_{H*}) and (d(v) < ρ̃_{H*})) then
2.     return;
3.  else if ((d(u) ≥ ρ̃_{H*}) and (d(v) < ρ̃_{H*})) then
                                        /* 只有一个顶点是高度顶点 */
4.     D_u ← AddToBag(u);
5.  else if ((d(u) < ρ̃_{H*}) and (d(v) ≥ ρ̃_{H*})) then
6.     D_v ← AddToBag(v);
7.  else                                    /* 两个顶点都是高度顶点 */
8.     D_u ← AddToBag(u);
9.     D_v ← AddToBag(v);
10.    if D_u = D_v then
11.       D_u ← D_u ∪ {u,v}
```

3.3　移除顶点/边操作

1) 移除顶点操作。与添加顶点操作类似,首先定义从 D 的包中移除顶点的过程,移除顶点的方法用作移除边过程的子程序。只有当顶点的度低于最大密度时才会从包中移除顶点,移除顶点不能是主核的一部分,算法从包中的 D 里移除顶点而不执行其他操作。

2) 移除边操作。利用包确保存在一个核数等于图的主核的 D,当其中一个顶点是低度顶点或者两个顶点属于不同的 D 时,包不需要任何更新。因此考虑其中一个顶点位于高度顶点边界的情况,移除边将顶点从高度移到低度,在这种情况下,算法仅需要从包中移除顶点,而不执行其他操作。如果移除边的两个顶点都是高度的并且属于同一个 D,算法从 D 中移除边,此外,验证并更新影响的顶点的核数,移除边可能会降低最大密度,因此需要向包中添加其度大于新最大密度的顶点。算法 3 描述了移除边的过程。

算法 3　移除边算法

1. if ((d(u) < $\tilde{\rho}_{H^*}$) and (d(v) < $\tilde{\rho}_{H^*}$)) then
/* 其中一个顶点是高度顶点 */
2. return;
3. if ((d(u) < $\tilde{\rho}_{H^*}$) and (d(u) +1 ≥ $\tilde{\rho}_{H^*}$)) then
/* 其中一个顶点位于高度顶点边界 */
4. RemoveVertex(u);
5. return;
6. if ((d(v) < $\tilde{\rho}_{H^*}$) and (d(v) +1 ≥ $\tilde{\rho}_{H^*}$)) then
7. RemoveVertex(v);
8. return;
9. for $D_i \in B$ do
/* 两个顶点都是高度并且属于同一个 D */
10. if ($D_i \cap (u,v) \neq \varnothing$) then
11. $D_i \leftarrow D_i \setminus (u,v)$; /* 从 D_i 中移除(u,v)这条边 */

3.4 完整的时态稠密子图发现算法

时态图是随时间变化的,边的添加和删除是不断进行的,算法需要保持稠密子图的近似密度 $\tilde{\rho}_{H^*}$,本文提出一个完全动态的算法 TDSubgraph 来查找滑动时间窗中的稠密子图(见算法 4)。算法 TDSubgraph 可以在准线性时间内解决稠密子图发现问题,得到增量稠密子图的最佳近似解。

算法 4 TDSubgraph

输入:$G=(V,E,\mathcal{T})$,$\{e_i\}$,$e=1,2,\cdots,k$。

输出:H^*。

1. $H^* \leftarrow G=(V,E,\mathcal{T})$;
2. Wait for update(Op,u,v),Op ∈ {Add,Remove};
/* 选择添加或者移除 */
3. if Op = Add then;
4. if Add = Vertex; 算法 1;
5. else 算法 2;
6. else 算法 3;
7. return H^*;

完整算法保证在图更新后包中顶点的度大于 $\tilde{\rho}_{H^*}$,图更新包括边添加和移除。算法必须考虑添加边操作,它可能通过合并多个子图影响 $\tilde{\rho}_{H^*}$ 的值。同样,必须考虑移除边操作,它可能通过减小任何稠密子图的密度、合并多个子图或分割一个子图影响 $\tilde{\rho}_{H^*}$ 的值。

下面给出本文算法的复杂度分析,设 A 为添加的边数,即 A 包含初始图的边和添加的边,R 为移除的边数,得到 $A+R=O(A)$,本文将一系列操作的总开销限制在移除边的随机选择定义的概率空间中。本文算法的添加边操作需要 $O(\log(A)\log^2(n))$ 的时间,移除边操作需要 $O(\log(A)\log^3(n))$ 的时间,因此算法总的时间复杂度为 $O(A\log(A)\log^2(|V|)+R\log(A)\log^3(|V|))$,同时算法需要 $O(n^2 poly(A)\log n)$ 的空间。

4 实验与结果分析

为了评估本文算法 TDSubgraph,使用社交网络,检查算法的运行时间,分析发现稠密子图的结构特征。实验中使用的所有数据集都是公开可用的。

4.1 数据集

本文所使用六个数据集都是来自社交网络,其中 FacebookL 和 TwitterL 数据集更大,用来评估算法的可扩展性,表 1 给出了各个数据集所包含的信息,包括节点个数 n,边的条数 m,数据集的时间跨度 $\mathcal{T}$(以天为单位),拓扑网络最稠密子图的密度 $d(H)$。

表 1 数据集包含的信息

数据集	|V|	|E|	|T|	d(H)
Facebook	4 117	10 000	104	5.29
Twitter	4 605	11 868	93	10.11
Students	889	9 837	120	11.29
Enron	1 143	6 245	8 080	14.38
FacebookL	45 813	10 000 000	104 000	27.36
TwitterL	162 207	10 000 000	78 362	21.24

(1) Facebook。该数据集是新奥尔良地区社区中三个月 Facebook 活动的子集,包含匿名墙贴的列表,子集涵盖 2006 年 5 月 9 日至 2006 年 8 月 6 日的时间段。

(2) Twitter。该数据集在 2010 年 8 月至 2010 年 10 月期间跟踪赫尔辛基地区 Twitter 用户的活动。由于用户交互考虑了包含其他用户评论的推文。

(3) Students。该数据集是加州大学欧文分校学生在线社区的活动日志。节点代表学生,边代表消息。使用该数据集的一个子集,涵盖时间范围是 2004 年 6 月 27 日至 2004 年 10 月 26 日。

(4) Enron。该数据集包含从 1980 年开始超过 20 多年的大公司高级管理层的电子邮件通信信息。

(5) FacebookL。该数据集由 Facebook 数据集顺序连接生成,包含 1 000 万条记录。

(6) TwitterL。该数据集由 Twitter 数据集连接生成,包含 1 000 万条记录。

4.2 实验设置

评估本文算法 TDSubgraph 的性能和效率,主要包括:通过算法发现的稠密子图密度评估性能,运行时间评估算法的效率。其中运行时间是移动滑动窗口所需的平均时间,包括添加新的边、移除旧的边、更新稠密子图。

本文实验的硬件配置是 Intel(R) Core(TM) i5 - 4590 3.30 GHz 处理器和 8.00 GB 内存,所有的算法都用 Java 实现,每次运行使用的内存不到可用主内存的 10%。

本文考虑了两种常用的流排序方案。

(1) BFS。排序是从随机顶点开始的广度优先搜索的结果。

(2) DFS。排序是从随机顶点开始的深度优先搜索的结果。

4.3　最优基线

算法的自然基线 Optimal[18] 结合了精确的动态编程,并为每个候选区间找到最佳稠密子图。由于 Optimal 的时间复杂度很高,生成一个包含 60 个时间戳的小数据集,其中每个时间戳包含一个带有 3 ~ 6 个顶点和随机密度的随机图。改变区间数 k 的值,图 2 给出算法结果和运行时间。在这个小数据集上,本文算法能够找到接近最佳的稠密子图,同时明显快于 Optimal。

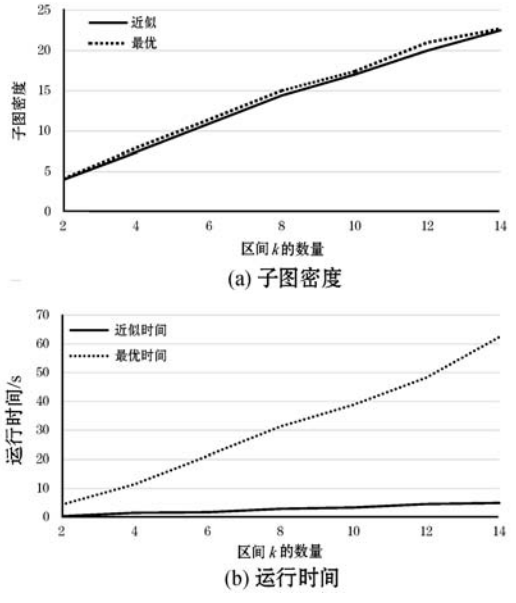


图 2　最优和近似算法的比较

4.4　真实数据集上的结果

由于基线算法 Optimal 在真实数据集上没有很好的可扩展性,本文将算法 TDSubgraph 与基线 kGoptDP[2] 和 kGoptDS[11] 进行比较。kGoptDP 算法执行精确的动态编程,但使用近似增量算法搜索稠密子图;kGoptDS 算法执行近似动态编程,同时为每个候选区间计算稠密子图。kGoptDP 具有 $2(1+\varepsilon_{DP})^2$ 近似保证,kGoptDS 具有 $(1+\varepsilon_{DS})$ 近似保证。然而这两个算法实际运行也非常慢,本文使用 Students 和 Enron 数据集的 1 000 条记录的子集进行比较。

为了确保公平性,本文给出算法发现的区间内最佳稠密子图的总密度。

本文用近似稠密子图搜索(ε_{DP})和近似动态编程(ε_{DS})的不同参数进行实验。表 2 给出算法 TDSubgraph、kGoptDP 和 kGoptDS 和发现的稠密子图的密度及算法的运行时间。对于这两个数据集,kGoptDS 发现了最佳稠密子图,因为该算法具有最佳近似因子。此外,kGoptDS 算法的运行时间最长,随着参数 ε_{DS}的增大,算法运行时间减少,即使是最大的参数值($\varepsilon_{DS}=2$),运行时间仍能达到一个多小时。kGoptDP 算法发现了第二好的稠密子图,只是略微优于本文算法(例如 $\varepsilon_{DP}=0.1$),从表 2 看出,随着 ε_{DP}的增大,得到的稠密子图的密度减小。对于这三个算法,随着近似参数增大,发现稠密子图的密度减小,然而密度减小并非像最坏情况表明得那样明显,使用这样的近似参数加快运行时间,TDSubgraph 为近似参数提供最快的高性能估计,从表 2 可以看出,本文算法对参数 ε_{DP}影响的稠密子图密度的变化更敏感。

表 2　TDSubgraph、kGoptDP 和 kGoptDS 的比较结果

数据集	稠密子图密度							运行时间/s						
Students			ε_{DP}				kGoptDS			ε_{DP}				kGoptDS
		TDS	0.01	0.1	1	2			TDS	0.01	0.1	1	2	
	ε_{DS}	0.01	4.27	4.27	3.85	3.85	6.30	ε_{DS}	0.01	0.52	0.52	0.53	0.54	23 678
		0.1	4.27	4.27	3.85	3.85	6.22		0.1	0.13	0.13	0.14	0.13	11 877
		1	4.27	4.27	3.85	3.85	5.76		1	0.03	0.26	0.03	0.03	3 394
		2	4.27	4.27	3.85	3.85	5.61		2	0.26	0.10	0.10	0.26	3 769
	kGoptDP		5.73	5.73	3.82	3.82		kGoptDP		162.0	43.5	29.5	29.5	
Enron			ε_{DP}				kGoptDS			ε_{DP}				kGoptDS
		TDS	0.01	0.1	1	2			TDS	0.01	0.1	1	2	
	ε_{DS}	0.01	10.7	10.7	10.3	10.8	11.3	ε_{DS}	0.01	54.4	53.5	40.3	29.8	25 788
		0.1	10.7	10.8	10.3	10.6	11.0		0.1	2.02	1.85	1.07	0.70	16 070
		1	9.57	9.57	8.83	9.86	11.0		1	0.33	0.34	0.19	0.18	7 834
		2	7.37	7.37	7.37	7.37	10.8		2	0.12	0.12	0.13	0.13	3 469
	kGoptDP		10.5	11.0	10.4	10.4		kGoptDP		165.0	61.15	17.82	6.07	

实验使用边流的 BFS 和 DFS 两种排序方案，评估不同方案对算法运行时间的影响，设置滑动窗口的大小为 $x=100k$。图 3 给出算法采用不同流排序方案的运行时间，可以看出，本文算法采用这两种排序方案处理所有数据集时，BFS 略优于 DFS，因为 DFS 策略占内存少但速度较慢，BFS 策略占内存多但速度较快，整体比较运行时间仍然都优于其他两个算法。

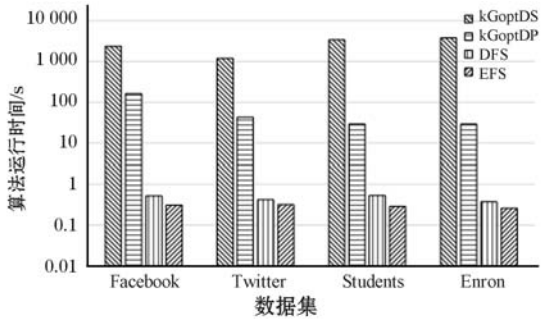
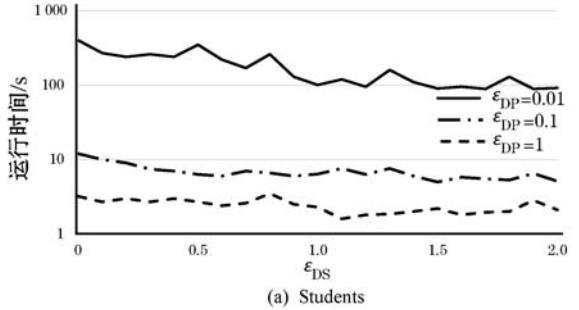


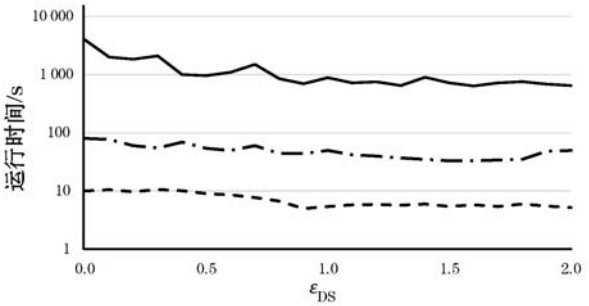
图 3 不同流排序方案对算法的影响

4.5 运行时间与可扩展性

图 4 给出近似参数 ε_{DS} 和 ε_{DP} 的改变对 TDSubgraph 运行时间的影响。图中结果表明参数 ε_{DP} 对运行时间有重大影响，同时参数 ε_{DS} 在这两个数据集上有很好的扩展性。



(a) Students



(b) Twitter

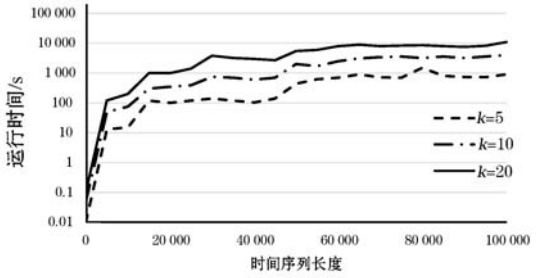
图 4 不同近似参数对算法的影响

实验还使用不同大小的滑动时间窗执行算法，即 $x=10k,100k,1M,10M$，实验中设置 $k=10$，评估滑动时间窗大小对算法的影响，使用具有 DFS 排序的 FacebookL 和 TwitterL 数据集。表 3 给出不同大小滑动时间窗的运行时间。增大滑动时间窗不会显著地影响算法运行时间，这一结果表明大多数的更新都是子图局部更新，并不需要遍历整个图。

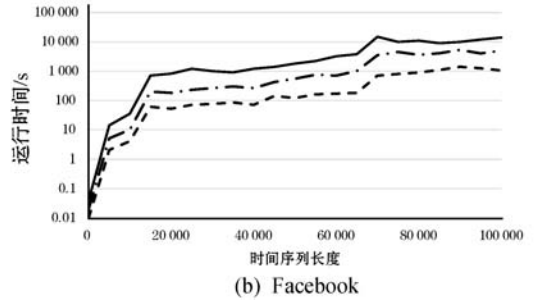
表 3 不同大小的滑动时间窗对算法运行时间的影响 ms

数据集	$x=10k$	$x=100k$	$x=1M$	$x=10M$
FacebookL	0.77	88.11	87.23	90.28
TwitterL	4.90	5.65	7.25	28.11

图 5 是在 Facebook 和 Twitter 数据集中增加交互次数，运行时间的变化显示出算法的可扩展性。实际上，运行时间在前 1 000 次交互中快速增长，然后饱和到线性依赖，因为网络初期顶点数量增长比较快，而且可能出现比之前更稠密的子图，必须频繁计算近似稠密子图。图 5 表明区间 k 的数量对算法的运行时间也有影响，随着 k 的增加运行时间增加。



(a) Twitter



(b) Facebook

图 5 算法的可扩展性

5 结 语

本文研究了在时态网络中基于滑动时间窗找到一系列稠密子图的问题，并提出一种有效的动态算法。现有算法在图更新时迭代整个图，本文算法仅影响图的有限区域，因此只需要局部更新稠密子图。结合滑动时间窗将网络时间线分割为 k 个非重叠间隔，使得间隔内能够发现具有最大密度的子图。结合理论分析，并验证该算法比文献[3]的算法更快。真实数据集上的实验验证本文算法是有效的，且可以得到高质量的结果。

今后的研究中，将考虑子图的频率、子图的统计非随机性对网络进行分割，另一种扩展是每个间隔内发现多个稠密子图可能是更有意义的。还可以考虑是否有可能在高度顶点的阈值上实现更强的界限，是否有可能为问题实现更好的近似保证，进一步改进算法，使

其成为许多实际应用的有用工具。

参 考 文 献

[1] Ma S, Hu R, Wang L, et al. Fast computation of dense temporal subgraphs[C]// 2017 IEEE 33rd International Conference on Data Engineering (ICDE),2017.

[2] Epasto A, Lattanzi S, Sozio M. Efficient densest subgraph computation in evolving graphs[C]//World Wide Web Conference (WWW),2015.

[3] Rozenshtein P, Bonchi F, Gionis A, et al. Finding events in temporal networks: Segmentation meets densest-subgraph discovery[C]//2018 IEEE International Conference on Data Mining (ICDM),2018.

[4] Chen J, Saad Y. Dense subgraph extraction with application to community detection[J]. IEEE Transactions on Knowledge and Data Engineering,2012,24(7):1216 - 1230.

[5] Angel A, Koudas N, Sarkas N, et al. Dense subgraph maintenance under streaming edge weight updates for real-time story identi? cation[J]. The VLDB Journal, 2014,23:175 - 199.

[6] Gibson D, Kumar R, Tomkins A. Discovering large dense subgraphs in massive graphs [C]//31st VLBD Conference, 2005.

[7] Saha B, Hoch A, Khuller S, et al. Dense subgraphs with restrictions and applications to gene annotation graphs[C]// Annual International Conference on Research in Computational Molecular Biology,2010.

[8] Semertzidis K, Pitoura E, Terzi E, et al. Best friends forever (BFF): Finding lasting dense subgraphs[J]. Data Mining and Knowledge Discovery,2019,33:1417 - 1445.

[9] Rozenshtein P, Tatti N, Gionis A. Finding dynamic dense subgraphs[J]. ACM Transactions on Knowledge Discovery from Data,2017,11(3):1 - 30.

[10] Tsourakakis C E. A novel approach to finding near-cliques: The triangle-densest subgraph problem[EB]. arXiv:1405.1477, 2014.

[11] Goldberg A V. Finding a maximum density subgraph[D]. University of California Berkeley,1984.

[12] Charikar M. Greedy approximation algorithms for finding dense components in a graph[C]// Third International Workshop on Approximation Algorithms for Combinatorial Optimization,2000:84 - 95.

[13] Myers S A, Leskovec J. The bursty dynamics of the twitter information network[C]// 23rd International Conference on World Wide Web,2014:913 - 924.

[14] Berlingerio M, Pinelli F, Calabrese F. ABACUS: Frequent pattern mining-based community discovery in multidimensional networks[J]. Data Mining and Knowledge Discovery, 2013,27:294 - 320.

[15] Akoglu L, Tong H, Koutra D. Graph-based anomaly detection and description: A survey[J]. Data Mining and Knowledge Discovery,2014,29:626 - 688.

[16] Rosvall M, Esquivel A V, Lancichinetti A, et al. Memory in network flows and its effects on spreading dynamics and community detection[J]. Nature Communications,2014,5:113 - 120.

[17] Bogdanov P, Mongiovì M, Singh A K. Mining heavy subgraphs in time-evolving networks[C]//2011 IEEE 11th International Conference on Data Mining,2011.

[18] Galbrun E, Gionis A, Tatti N. Top-k overlapping densest subgraphs[M]. Kluwer Academic Publishers,2016.

(上接第 257 页)

相对于 WM 和 DHSWM 算法来说,减少了每次参与匹配模式串匹配的数量,也减少了每个模式串比较字符次数,并在失配时加快了模式串向后移动的速度,因此,时间性能得到了一定幅度的提高。

参 考 文 献

[1] 赵晓,何立风,王鑫,等. 一种高效的模式串匹配算法[J]. 陕西科技大学学报(自然科学版),2017,35(1):183 - 187.

[2] 董世博,李训根,殷珍珍. 一种改进的字符串多模式匹配算法[J]. 计算机工程与应用,2013,49(8):133 - 137.

[3] 刘卫国,胡勇刚. DHSWM:一种改进的 WM 多模式匹配算法[J]. 中南大学学报(自然科学版),2011,42(12):3765 - 3771.

[4] 唐君,杨云. 基于多模式匹配算法的计算机网络入侵检测研究[J]. 科技通报,2014,30(4):218 - 221.

[5] 蒋晓鸽,武小年,张昭,等. 基于后缀 WM 匹配算法的改进算法[J]. 计算机与数字工程,2013,41(4):608 - 610.

[6] 贾博威,吴志刚,张树壮. 基于 Wu-Manber 算法的大规模 URL 模式串匹配算法[J]. 智能计算机与应用,2017,7(5):4 - 9.

[7] 王一霈,石春,戴上静,等. 一种改进的针对中文编码的 Wu-Manber 多模式匹配算法[J]. 小型微型计算机系统,2015,36(4):778 - 781.

[8] 戴胜冬,杨昆. 计算 DNA 序列模式特征的匹配算法[J]. 杭州电子科技大学学报,2015,35(1):88 - 92.

[9] 刘燕兵,邵妍,王勇,等. 一种面向大规模 URL 过滤的多模式串匹配算法[J]. 计算机学报,2014,37(5):1159 - 1169.

[10] 董迎亮,玄雪花,王德民. 基于 WM 算法改进的多模式匹配算法[J]. 吉林大学学报(信息科学版),2011,29(4):383 - 387.