有向图上的影响力社区搜索

杜明，胡欣雨，周军锋
（东华大学计算机科学与技术学院，上海 201620）

摘　要：现有社区搜索方法用于从无向图中挖掘满足内聚性和影响力要求的社区，没有考虑有向图中边的方向对社区的影响，导致有向图上社区挖掘的结果出现影响力和内聚性不足的问题。基于此，提出有向图上的影响力社区搜索问题，并设计相应的在线搜索算法；为进一步提升社区挖掘的效率，提出有向图上基于索引的影响力搜索方法及其优化策略。此外，提出一种基于并行思想的索引构建方法，加速索引的构建过程。最后，基于 8 个真实数据集进行验证，实验结果验证了所提算法的有效性和高效性。

关键词：社区搜索；有向图；Truss 模型；影响力
中图分类号：TP311
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2024189

Influence community search on directed graphs

DU Ming, HU Xinyu, ZHOU Junfeng
School of Computer Science and Technology, Donghua University, Shanghai 201620, China

Abstract: Existing community search methods are used to explore communities in undirected graphs that meet cohesion and influence requirements, without considering the impact of edge direction in directed graphs. This oversight leads to insufficient influence and cohesion in the results of community detection on directed graphs. The problem of influence community search on directed graphs was proposed, and a corresponding online search algorithm was designed. To further enhance the efficiency of community mining, an index-based influence search method on directed graphs and its optimization strategies were proposed. In addition, a parallel-based index construction method was proposed to accelerate the index building process. Finally, based on eight real-world datasets, validation is conducted, and the experimental results confirm the effectiveness and efficiency of the proposed algorithm.

Keywords: community search, directed graph, Truss model, influence

0　引言

社区搜索旨在从大型网络中寻找满足特定查询条件的连通子图，是网络分析的基础问题之一[1]。实际应用中的网络，如 Facebook、Twitter 和 Youtube 等，常隐藏着复杂的结构和丰富的信息。基于社区的内聚性[2-9]和影响力[1,10-15]等信息，社区搜索可快速挖掘数据之间的潜在联系，并广泛应用于生物信息学[16]、环境科学[17]、社会科学[18]和交通网络[19]等领域。

现有的社区搜索方法主要分为 2 类：一类仅关注内聚性，另一类则同时考虑内聚性和影响力。在第一类方法中，研究者提出了针对无向图的内聚子图模型 k-core[4]和 k-truss[5]，以及针对有向图的内聚子图模型 D-core[6]、CF-truss[7]、D-truss[8]，但这类方法仅考虑了社区的结构内聚性，并没有考虑节点附带有其他信息的实际情况；第二类方

收稿日期：2024–07–12；修回日期：2024–10–10
通信作者：周军锋，zhoujf@dhu.edu.cn
基金项目：国家自然科学基金资助项目（No.62372101, No.61873337, No.62272097）
Foundation Items：The National Natural Science Foundation of China (No.62372101, No.61873337, No.62272097)

法[12-15]将社区影响力与内聚子图模型相结合，能够从无向图中返回同时满足结构内聚性和影响力约束的结果。

然而在实际应用中的图通常是有向图，比如社交网络[18]中一个用户对其他用户的关注，学术网络中一篇文献对其他文献的引用，生物网络[16]中的信息传递等，现有的影响力社区搜索方法返回的结果无法准确反映实体间的有向关系，需要设计针对有向图的影响力社区搜索方法。以社交网络为例，用户本身具有一定影响力，彼此之间展现出复杂的单向关注关系，这种关系的有向性无法通过无向图准确展现。当新用户希望加入感兴趣的网络社群，或者广告公司在进行产品推广时，他们通常会选择与某人或某个话题密切相关且具有一定影响力的社群。其中，密切相关和一定影响力分别指的是有向图的内聚性和社区的影响力。尽管现有的2类社区搜索方法可以找到关系紧密或高影响力的社群，但却无法保证社群同时满足这2种属性。

针对现有研究的局限性，本文提出了有向图上的影响力社区搜索（IDCS, influential community search in directed graph）问题。具体来说，IDCS问题旨在找到有向图中满足查询点约束、结构内聚性约束和影响力约束的子图。

本文根据定义将IDCS问题的求解过程分为2个阶段，首先查找规模最大的内聚子图，然后查找影响力最大的子图，并提出了相应的在线算法。由于在线算法需多次遍历全图，并逐一检查子图的内聚性，效率较低。为了进一步提高算法效率，本文提出了以下优化策略。

首先，提出基于索引的最大内聚子图查询方法。通过预先计算并记录每条边的支持度，查找内聚子图只需遍历一次索引并检查该边是否满足内聚性约束，从而无须访问原图就能逐一检查子图是否满足内聚性要求，显著提升了查询效率。

其次，提出一种基于顶点权值的优化查询方法。优先从影响力较大的图中查找内聚子图，减少全图搜索的次数。

再次，提出一种基于并行思想的索引构建方法，能够有效加速索引构建。

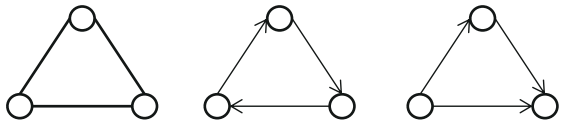
最后，在8个真实数据集上进行对比实验，验证了本文方法的可行性和高效性。

1 问题定义和相关工作

1.1 问题定义

对于给定的有向图 $G=(V_G,E_G,W_G)$，V_G 是顶点集，E_G 是边集，W_G 是每个顶点对应权值的集合，代表顶点的影响力。对于图 G，顶点数和边数分别用 $|V_G|$ 和 $|E_G|$ 表示。如果 $<u,v>$ 表示从顶点 u 指向顶点 v 的有向边，则 u 是 v 的入邻居，v 是 u 的出邻居。$N_G^+(v)$ 和 $N_G^-(v)$ 分别为顶点 v 的入邻居集和出邻居集，$N_G(v)$ 为顶点 v 的邻居集，即 $N_G(v)=N_G^+(v)\cup N_G^-(v)$。因此，顶点 v 的入度为 $\deg_G^+(v)=|N_G^+(v)|$，出度为 $\deg_G^-(v)=|N_G^-(v)|$，总度数 $\deg_G(v)=|N_G(v)|$。

无向图中的三角形 $\mathrm{tri}(u,v,w)$ 是由顶点 u,v,w 和边 (u,v)、(v,w) 和 (u,w) 连接在一起的子图结构[5]，如图1(a)所示，常用于衡量子图的内聚性。区别于无向图，有向图中存在cycle三角形和flow三角形[7]，结构如图1(b)和图1(c)所示。cycle三角形每个顶点有且仅有一个入度和出度，所以cycle三角形的顶点构成一个强连通子图；flow三角形每个顶点的出度分别为0、1和2，是非强连通子图。



(a) 无向图上的三角形 (b) cycle三角形 (c) flow三角形

图1 无向图和有向图中的三角形示例

定义1 边的支持度。给定有向图 $G=(V_G,E_G)$，边 $e=<u,v>$，$e\in E_G$。边 e 的cycle支持度是该边参与的cycle三角形个数，记为 $\mathrm{csup}_G(e)$。边 e 的flow支持度是该边参与的flow三角形个数，记为 $\mathrm{fsup}_G(e)$。

定义2 子图。给定一个有向图 $G=(V_G,E_G)$，若图 $H=(V_H,E_H)$，$V_H\subseteq V_G$ 且 $E_H\subseteq E_G$，则称 H 为 G 的子图，记为 $H\subseteq G$。

定义3 CF-truss子图。给定有向图 $G=(V_G,E_G)$，子图 $H=(V_H,E_H)$，其中 $V_H\subseteq V_G$ 且 $E_H\subseteq E_G$。若任意边 $e\in E_H$ 满足 $\mathrm{csup}_H(e)\geqslant c$ 且 $\mathrm{fsup}_H(e)\geqslant f$，$c,f\in R$，则称 H 是CF-truss子图。

定义4 边的Trussness。给定有向图 $G=$

(V_G,E_G)，若图 G 是 CF-truss，则对于任意边 $e \in E_G$，其 Trussness 为 (c,f)，记为 $T(e)=\{(c,f)\}$。

定义 5　Trussness 支配。给定边 e 的 2 个 trussness 值：(c_1,f_1) 和 (c_2,f_2)，当且仅当满足 $c_1 \geqslant c_2$ 且 $f_1 > f_2$、$c_1 > c_2$ 且 $f_1 \geqslant f_2$ 这 2 个条件之一时，trussness (c_1,f_1) 支配 trussness (c_2,f_2)，表示为 $(c_2,f_2) < (c_1,f_1)$。

定义 6　边的 Skyline Trussness。给定边 e 和 $T(e)=\{(c_1,f_1),(c_2,f_2),\cdots,(c_n,f_n)\}$，如果存在 $T(e) \subseteq T(e)$ 并且 $T(e)$ 中的任意 2 个 trussness 值都不满足支配条件，则 $T(e)$ 为 Skyline Trussness，记为 $\mathrm{ST}(e)=\{(c_i,f_i) \in T(e) : \nexists (c_i,f_i) \in T(e), \text{s.t.}, (c_i,f_i) < (c_j,f_j)\}$。

定义 7　子图影响力。给定 $G=(V_G,E_G,W_G)$，图 G 的影响力为图中所有顶点权值的最小值，记为 $\mathrm{SW}(G)$。

例 1　图 2 是有向图 G 的示例，其中每个顶点都被赋予一个权重值。例如，$W_G(v_1)=9$，表示顶点 v_1 的影响力为 9。图 2 中 H_1、H_2、H_3 均为 G 的子图，其子图影响力分别为 $\mathrm{SW}(H_1)=3$、$\mathrm{SW}(H_2)=8$、$\mathrm{SW}(H_3)=8$。从影响力角度来看，H_2 和 H_3 子图具有相同的影响力，并且都优于 H_1 子图。H_3 子图中的每条边都满足 csup ≥ 1 和 fsup ≥ 1，所以 H_3 子图为(1,1)-truss；而 H_2 子图中的每条边仅满足 csup ≥ 0 和 fsup ≥ 1，所以 H_2 子图为(0,1)-truss。因此，H_3 的子图内聚性要优于 H_2。

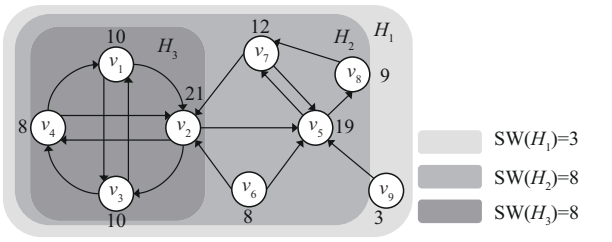


图2　节点带权的有向图示例

给定有向图 $G=(V_G,E_G,W_G)$、查询点 q，以及正整数 c 和 f，有向图上的影响力社区搜索问题旨在找到满足以下条件的子图 $H=(V_H,E_H,W_H)$。

1) 包含查询顶点 q。

2)子图 H 满足 (c,f)-truss。

3)子图 H 的影响力最大。

4)影响力相等时，子图 H 规模最大。

例 2　给定图 2 所示有向图、查询点 $q=v_3$，以及 $c=f=1$，则有向图上的影响力社区搜索返回的子图为由顶点集 $\{v_1,v_2,v_3,v_4,v_5,v_7,v_8\}$ 构成的子图 H，$\mathrm{SW}(H)=8$。虽然图 G 中子图 H_3 满足(1,1)-truss 且 $\mathrm{SW}(H_3)=8$，但 $|V_{H_3}| < |V_H|$，即子图规模未达到最大。因此，返回的结果应为子图 H。

表 1 给出了本文所用符号及其意义。

表 1　本文所用符号及其意义

符号	描述
G	数据图
q	查询点
csup	cycle 支持度
fsup	flow 支持度
$\deg(v)$	顶点 v 的总度数
$\deg^+(v)$	顶点 v 的出度
$\deg^-(v)$	顶点 v 的入度
$<u,v>$	有向边 u 指向 v
(u,v)	无向图边 uv
$W(v)$	顶点 v 的权值
$\mathrm{SW}(G)$	图 G 的影响力

1.2　相关工作

1.2.1　有向图上的内聚子图模型

有向图上的内聚子图模型主要有 D-core[6]、CF-truss[7] 和 D-truss[8-9]。D-core[6] 模型要求子图中所有顶点的入度不小于正整数 k 并且出度不小于正整数 l。有向图中存在着两类特殊顶点：高入度低出度的顶点和低入度高出度的顶点。通常情况下，D-core 模型只能查找包含其中一类特殊顶点的社区。若要考虑查找同时包含这 2 类特殊顶点的社区，则需要将 k 和 l 都设置为较低的值，这会导致社区结构松散，难以准确找到目标社区。Takaguchi 等[7]研究了有向图中的 truss 结构，定义了有向图中的 2 类三角形：cycle 三角形和 flow 三角形，并提出了 CF-truss 模型（也称 (c,f)-truss）。与 D-core 模型相比，CF-truss 模型能够捕捉边的内聚性，从而更有效地识别有向图中的真实社区。在这一基础上，Liu 等[8]进一步提出了 D-truss 模型，该模型综

合考虑了有向图中边的内聚性和顶点结构，更准确地描述了有向图的内聚性。

然而，在有向图上进行影响力社区搜索时，需要综合考虑子图的内聚性和影响力。D-truss模型对内聚性的要求较高，这会限制子图的影响力，导致返回的社区只关注内聚性约束，而忽略了影响力。同时，目前此类社区搜索方法均未考虑社区影响力对查询结果的影响，因此无法解决IDCS问题。

1.2.2 无向图上的影响力社区搜索

Zhou等[13]提出了*k*-weighted core问题，用于查找规模最大且所有顶点的加权度都大于*k*的子图。Li等[14]提出了top-*r* *k*-influential社区问题，寻找前*r*个子图影响力最大的*k*-core子图。杜明等[15]提出了规模受限的影响力社区搜索问题，查找兼顾社区规模、结构特征和属性特征约束的影响力社区。以上影响力社区搜索方法仅考虑了无向图，无法从有向图中返回同时满足结构内聚性和影响力约束的子图，因此不能有效解决IDCS问题。

1.2.3 Skyline Trussness索引

Liu等[8]设计了ST（Skyline Trussness）索引，并提出了基于ST索引的D-truss子图查询方法。该方法消除了在线算法中的重复操作，加快了查询D-truss子图的速度。根据定义5和定义6，构建ST索引主要包含以下2个步骤：找到kc=0至$\mathrm{kc}_{\max}$的所有最大子图；在这些子图中依次寻找kf=0至$\mathrm{kf}_{\max}$的子图。对于(kc,kf)-truss子图中的任意边e，其Trussness为(kc,kf)，即$T(e)=\{(\mathrm{kc},\mathrm{kf})\}$，然后根据定义6记录该边的ST。

ST索引存储了有向图中各边支持度的关键信息，能够用于加速查找有向图上的影响力社区搜索。

2 在线算法

鉴于IDCS问题涉及有向图中的2种社区属性，该问题的解决思路可以简单概括为2个阶段：首先查找规模最大的CF-truss子图，然后查找其中影响力最大的子图。具体实现如算法1所示。首先，查找图中包含查询点q且规模最大的CF-truss子图（第2行FindMaxTruss）。如果存在满足该条件的子图，则查找其中影响力最大的子图（第3~5行InfluentialTruss）。下面分别讨论这2个阶段。

算法1 在线算法框架

输入 有向图G，查询顶点q，正整数c和f

输出 子图影响力最大的CF-truss子图H

1)$H \leftarrow \varnothing$;

2)$G_M \leftarrow \mathrm{FindMaxTruss}(G,q,c,f)$;

3) if $G_M=\varnothing$ then return $\varnothing$;

4) end if;

5) $H \leftarrow \mathrm{InfluentialTruss}(G_M,q,c,f)$;(算法3/算法4)

6) return H;

2.1 最大CF-truss子图查询

为了查找最大CF-truss子图，一种直观的方法是遍历图中各边，移除不符合约束条件的边并更新相邻边的支持度。基于该思想，本文提出了FindMaxTruss算法，算法2中展示了具体实现细节。计算每条边的支持度，若边的支持度满足csup < c或fsup < f，则该边加入删除队列Q（第2~5行）。依次删除Q中的边，并更新邻边的支持度。若更新后支持度满足csup < c或fsup < f，则将该边加入删除队列中（第6~16行）。重复上述过程，直到删除Q中所有边，剩余子图作为结果返回（第6~21行）。

例3 给定图2中有向图G，$q=v_2$，$c=f=1$，计算图G中每条边的支持度，其中边$<v_6,v_2>$、$<v_6,v_5>$、$<v_9,v_5>$的csup < 1，将这3条边加入队列Q。逐一对队列Q中各边执行删除操作。以边$<v_6,v_5>$为例，删除边$<v_6,v_5>$后，重新计算边$<v_6,v_2>$和$<v_2,v_5>$的支持度：$\mathrm{fsup}(<v_6,v_2>)=0<1$，$\mathrm{fsup}(<v_2,v_5>)=1$。此时$<v_6,v_2>$已在队列$Q$中，无须重复加入。删边完成后，图$G$中剩余边都满足csup ⩾ c且fsup ⩾ f。因此，由节点$\{v_1,v_2,v_3,v_4,v_5,v_7,v_8\}$组成子图$G_M$即包含$v_2$的最大(1,1)-truss子图，记为$G_M$：$\{v_1,v_2,v_3,v_4,v_5,v_7,v_8\}$。

算法2 FindMaxTruss算法

输入 有向图G，查询顶点q，正整数c和f

输出 规模最大的CF-truss子图G_M

1) $Q \leftarrow \varnothing$; $G_M \leftarrow \varnothing$;

2) 计算图G中每条边的csup支持度和fsup支持度

3) if (csup(e) < c or fsup(e) < f) then

4) $Q = Q \cap e$;

5) end if;
6) while $Q \neq \varnothing$do
7) pop out $e =< u,v >$ from Q;//边e从队列Q中弹出
8) 在G中删除e;
9) for each vertex $w \in N(u) \cap N(v)$ do
10) for$e' \in \{< w,u >, < w,v >, < u,w >, < v,w >\} \cap E_G$ do
11) 计算边e'的csup支持度和fsup支持度;
12) if (csup$(e') <$ $c$or fsup$(e') < f$) then
13) $Q = Q \cap e'$;
14) end if;
15) end for;
16) end for;
17) if $q \in V_G$ then
18) $G_M \leftarrow G$;
19) end if;
20) end while;
21) return G_M;

边支持度计算是FindMaxTruss算法的核心。以边$e =< u,v >$为例，计算支持度，即计算u、v顶点公共邻居的个数，时间复杂度为$O(\deg(u)\deg(v))$。最坏的情况下，原图中不存在满足约束条件的边，所有边都需删边并更新邻边支持度，时间复杂度为$O\left(\sum_{< u,v > \in E_G} \deg(u)\deg(v)\right) \leqslant O(|E_G|^{1.5})$。因此，FindMaxTruss算法的时间复杂度为$O(|E_G|^{1.5})$。另外，算法需存储原图信息，空间复杂度为$O(|V_G| + |E_G|)$。

2.2 影响力社区搜索

在获取规模最大的CF-truss子图后，需进一步查找其中影响力最大的子图。为此，本文提出了2种基础查找方法：自顶向下的Top-down算法和自底向上的Bottom-up算法。

2.2.1 Top-down算法

Top-down算法采用自顶向下的查找策略：迭代删除权值最小的顶点并维持truss结构，直到图中不存在truss或不含查询点。算法3提供了Top-down算法的具体实现过程。图G_M中所有顶点按权值从小到大排序（第2行），依次删除权值最小的顶点，并检查是否存在CF-truss子图。若存在，则保存子图并重复删点操作，否则算法结束并返回保存的子图（第3~11行）。

例 4　给定图3所示G_M子图，$q = v_2$，$c = f = 1$，删除权值最小的顶点v_4，并保存由顶点$\{v_2,v_5,v_7,v_8\}$构成的(1,1)-truss子图H。继续删除权值最小的顶点v_8，此时不存在满足(1,1)-truss的子图，算法终止。最终返回子图H:$\{v_2,v_5,v_7,v_8\}$，SW$(H) = 9$。

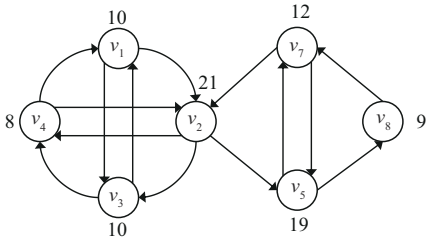


图3　规模最大的(1,1)-truss子图G_M

算法3　Top-down算法
输入　有向图G_M，查询顶点q，正整数c和f
输出　子图影响力最大的CF-truss子图H
1) $H \leftarrow G_M$;
2) 将图G_M中所有的顶点按$W(v)$升序排列
3) while SW$(G_M) <$ $W(q)$do
4) 将顶点v设为图G_M中权值最小的顶点;
5) 删除顶点v及其关联边，并维持CF-truss;
6) 将G_M设为步骤5)的结果;
7) if $G_M \neq \varnothing$ then $H \leftarrow G_M$;
8) else break;
9) end if;
10)end while;
11) return H;

每轮迭代中，Top-down算法删点并维持truss的时间复杂度为$O(|E_{G_M}|^{1.5})$。因此，整个Top-down算法的时间复杂度为$O(\alpha|E_{G_M}|^{1.5})$，其中α是迭代的次数。考虑到该操作中需要保留上次的计算结果，Top-down算法的空间复杂度为$O(|V_{G_M}| + |E_{G_M}|)$。

2.2.2 Bottom-up算法

Bottom-up算法采用自底向上的查找策略：从

距离查询点最近的顶点开始逐层构建新图，并寻找其中影响力最大的CF-truss子图。若迭代中得到的最大影响力小于前一次迭代中最大影响力，则前一次迭代中的搜索结果即影响力最大的CF-truss子图。

定理1 给定2个不同的最大CF-truss子图H_1:(c_1,f_1) - truss和H_2:(c_2,f_2) - truss，如果$c_1 \geqslant c_2$且$f_1 \geqslant f_2$，则$H_1 \subseteq H_2$，$\mathrm{SW}(H_1) \geqslant \mathrm{SW}(H_2)$。

证明 H_1是(c_1,f_1) - truss，H_2是(c_2,f_2) - truss，且$c_1 \geqslant c_2$、$f_1 \geqslant f_2$，根据CF-truss的定义，可以得出H_1中的每条边和每个顶点都满足H_2中的条件。因此，H_1中的边和顶点也属于H_2，并且H_1的子图影响力一定大于或等于H_2的子图影响力，即$H_1 \subseteq H_2$，$\mathrm{SW}(H_1) \geqslant \mathrm{SW}(H_2)$。证毕。

根据定理1，若原图中存在CF-truss子图，逐层建图必然可以找到影响力最大的CF-truss子图。

算法4中给出了Bottom-up算法的主要实现过程。从查询点开始，步长为1加入一批新的顶点，构建子图G_i（第3~5行）。图H记录G_i中影响力最大且满足CF-truss的子图G'_i（第6~10行）。步长加1，重复上述步骤，直到找到影响力最大的CF-truss子图（第7~16行）。

例5 给定图3所示G_M子图，$q=v_2$，$c=f=1$，从查询点v_2开始，步长为1，加入顶点$\{v_1,v_3,v_4,v_5,v_6,v_7\}$，构建子图$G_1$:$\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$，由顶点$\{v_1,v_2,v_3,v_4\}$构成的子图$G'_1$是影响力最大的(1,1)-truss子图；步长增加至2，加入顶点$\{v_8,v_9\}$，构建G_2:$\{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8,v_9\}$，由顶点$\{v_2,v_5,v_7,v_8\}$构成的子图$G'_2$是影响力最大的(1,1)-truss子图；步长增加至3，没有新顶点加入，算法终止，返回子图H:$\{v_2,v_5,v_7,v_8\}$，$\mathrm{SW}(H)=9$。

算法4 Bottom-up算法

输入 有向图G_M，查询顶点q，正整数c和f

输出 子图影响力最大的CF-truss子图H

1) $i \leftarrow 0$; $V_0 \leftarrow \{q\}$; $H \leftarrow G_M$;
2) do
3) $i \leftarrow i+1$;
4) $V_i = V_{i-1} \cup \{N(v) | v \in V_{i-1}\}$;
5) 由顶点集V构建图G_M的诱导子图G_i;
6) 删除权值最小的顶点并维持CF-truss;
7) G'_i存储步骤6)的结果;
8) if $G'_i = \varnothing$then
9) 跳转到步骤3);
10) end if;
11) if $\mathrm{SW}(G'_i) < \mathrm{SW}(H)$then break;
12) $H \leftarrow G'_i$;
13) end if;
14) while V_{i-1} is not equal to V_i
15) end while;
16) return H;

Bottom-up算法主要包括2个步骤：逐层建图和寻找影响力最大的CF-truss子图。逐层建图需遍历最大CF-truss子图G_M，时间复杂度为$O(\left|E_{G_M}\right|)$。查找影响力最大的CF-truss子图，即删除权值最小的顶点并维持truss，时间复杂度为$O(\left|E_{G_M}\right|^{1.5})$。所以，Bottom-up算法的时间复杂度为$O(\left|E_{G_M}\right| + \beta\left|E_{G_M}\right|^{1.5})$，其中$\beta$是迭代的次数。考虑到Bottom-up算法需保留每轮迭代的最大影响力子图，因此空间复杂度为$O(\left|V_{G_M}\right| + \left|E_{G_M}\right|)$。

3 优化策略

第2节中提出的在线算法存在以下2个缺陷。

首先，FindMaxTruss算法需遍历全图，计算支持度并删除不满足约束条件的边，搜索最大CF-truss子图的效率不高。实际上，给定有向图G、查询点q以及约束条件c和f，如果存在包含查询点q的CF-truss子图，那么包含查询点q且规模最大的CF-truss是唯一且确定的。因此，预先计算并存储各边支持度，可有效加速查找规模最大的CF-truss子图。

其次，Top-down算法和Bottom-up算法中存在冗余操作，搜索影响力最大的子图效率不高。Top-down算法在全图范围内删除权值最小的顶点并维持truss，效率较低。Bottom-up算法虽通过逐层建图限制了搜索范围，但建图时未考虑顶点权值，多次迭代才能确保算法的准确性。实际上，在搜索影响力最大的子图时，若优先考虑顶点权值，从影响力较大的子图中查找CF-truss子图，可有效限制搜索范围并减少冗余迭代。

3.1　基于索引的最大 CF-truss 子图查询

根据定义4，对于给定边 e，若 e 包含在 CF-truss 子图中，则 e 的 Trussness 为 (c,f)。考虑到每条边可能属于多个 D-truss，存储 Trussness (e) 的空间成本较高，定义5引入 Trussness 支配的概念，并基于此给出 ST 索引的定义：给定边 e 和 Trussness 值 $T(e)=\{(c_1,f_1),(c_2,f_2),\cdots,(c_n,f_n)\}$，若存在 $T(e)\subseteq T(e)$ 并且 $T(e)$ 中的任意2个 Trussness 值都不满足支配条件，则 $T(e)$ 为 Skyline Trussness，记为 $\mathrm{ST}(e)=\{(c_i,f_i)\in T(e):\nexists(c_i,f_i)\in T(e),\text{s.t.}(c_i,f_i)<(c_j,f_j)\}$。ST 索引仅存储各边支持度的关键信息，既保证了信息的完整性，又有效减少了空间开销。

对于边 e，若 ST(e) 中存在 Trussness 值 (c_i,f_i)，满足 (c_i,f_i) 支配 (c,f)，那么边 e 一定包含在规模最大的 CF-truss 子图中。因此，本文提出基于索引的最大 CF-truss 子图查询，其基本思想如下：从查询点 q 出发，依次遍历图中各边及其索引，找到最大 CF-truss。

具体算法实现如算法5所示。将查询点 q 作为候选队列 Q 和结果子图 G_M 中的初始顶点（第1~2行）。依次检查队列中各顶点的入边和出边（第3~5行）。对于边 $e=<u,v>$，若 ST(e) 中存在 (c_i,f_i)，满足 (c_i,f_i) 支配 (c,f)，则将边 e 加入图 G_M 中，将未访问的顶点 u、v 加入队列 Q 中（第6~13行）。队列中所有顶点都遍历完成后返回图 G_M（第3~16行）。

例6　给定图4所示有向图 G，$q=v_2$，$c=f=1$，从查询点 v_2 开始，依次处理 v_2 的入边和出边。边 $<v_2,v_4>$、$<v_2,v_3>$、$<v_2,v_5>$、$<v_4,v_2>$、$<v_1,v_2>$、$<v_7,v_2>$ 满足支配条件，加入图 G_M，并将未访问顶点 v_4、v_3、v_5、v_1、v_7 加入队列。其中边 $<v_4,v_1>$、$<v_3,v_4>$、$<v_3,v_1>$、$<v_1,v_3>$、$<v_5,v_7>$、$<v_7,v_5>$、$<v_5,v_8>$、$<v_8,v_7>$ 满足支配条件，加入图 G_M，并将顶点 v_8 加入队列。此时不存在满足支配条件的边，算法结束。因此，由节点 $\{v_1,v_2,v_3,v_4,v_5,v_7,v_8\}$ 组成的子图 G_M 是包含 v_2 的最大(1,1)-truss 子图，记为 G_M：$\{v_1,v_2,v_3,v_4,v_5,v_7,v_8\}$。

算法5　基于索引的最大 CF-truss 子图查询

输入　有向图 G，查询顶点 q，正整数 c 和 f

输出　最大的 CF-truss 子图 G_M

1) $G_M\leftarrow\varnothing$;add vertex q into G_M;
2) $Q\leftarrow q$; mark q as visited;
3) while $Q\neq\varnothing$ do
4)　pop out v from Q;//边 e 从队列 Q 中弹出
5)　for 未访问 edge $e=<u,v>$ or $<v,u>$ do
6)　　将边 e 设为已访问;
7)　　for each $(c',f')\in\mathrm{ST}(e)$ do
8)　　if $(c,f)\leqslant(c',f')$ then
9)　　在图 G_M 中加入边 e 和顶点 u;
10)　　$Q\leftarrow Q\cup\{u\}$;
11)　　将顶点 u 设为已访问; break;
12)　　end if;
13)　end for;
14)　end for;
15) end while;
16) return G_M;

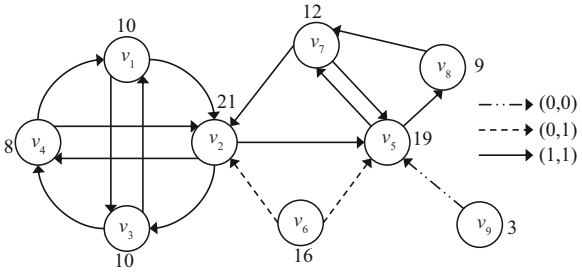


图4　构建ST索引示例

查询时，遍历各边的 ST 索引不会超过 $O(\min\{c_{\max},f_{\max}\})$，所以基于索引的最大 CF-truss 子图查询的时间复杂度为 $O\left(\min\left\{c_{\max},f_{\max}\right\}\left|E_G\right|\right)$，其查询效率远优于 FindMaxTruss 算法。

3.2　影响力社区搜索优化策略

Top-down 算法在全图范围内删除权值最小的顶点并维持 truss，效率较低。而 Bottom-up 算法虽通过逐层建图限制了搜索范围，但建图时未考虑顶点权值，多次迭代才能确保算法的准确性。

因此，为了快速查找影响力最大的 CF-truss 子图，本文提出了基于影响力优先策略的 WI（weight-based insertion）算法。根据顶点权值，优先构建影响力较大的子图。若影响力较大的图中存在 CF-truss，则无须遍历影响力较小的图，有效限制了查询范围。同时，在构建子图时，算法将具有相同权值的顶点一并加入图中。根据定理1，若图中存在 CF-truss 子图，则该子图一定是影响力最大

且规模最大的。

具体算法实现如算法 6 所示。初始化顶点集 V_{base}，将图 G_M 中权值大于或等于查询点的顶点加入 V_{base} 中，构建图 G_{base}（第 1~2 行）。若 G_{base} 中存在 CF-truss，则算法终止并输出 CF-truss 子图（第 3~5 行）。否则，剩余节点 V_{rest} 按权值由大到小的顺序依次加入顶点集 V_{add}，构建图 G_{add}（第 6~11 行）。若 G_{add} 中存在 CF-truss 子图，则算法终止并输出该子图（第 6~15 行）。

例 7 给定图 3 所示 G_M 子图，$q = v_5$，$c = f = 1$，将图 G_M 中权值大于或等于查询点的顶点 $\{v_2\}$ 加入顶点集 V_{base}，构建图 G_{base}:$\{v_2,v_5\}$。G_{base} 中不存在(1,1)-truss，算法继续。剩余的顶点按权值降序 $\{v_7\}$、$\{v_1,v_3\}$、$\{v_8\}$、$\{v_4,v_6\}$ 依次加入顶点集，并构建图 G_{add}。当加入顶点 $\{v_7,v_1,v_3,v_8\}$ 时，由顶点 $\{v_2,v_5,v_7,v_8\}$ 构成的子图 H 满足(1,1)-truss。算法终止并返回子图 H:$\{v_2,v_5,v_7,v_8\}$，$\text{SW}(H) = 9$。

算法 6 WI 算法

输入 有向图 G_M，查询顶点 q，正整数 c 和 f

输出 子图影响力最大的 CF-truss 子图 H

1) $V_{\text{base}} = \{ v | W(v) \geqslant W(q) \text{ and } v \in G_M \}$;

2) 由顶点集 V_{base} 构建图 G_M 的诱导子图 G_{base};

3) $H \leftarrow \text{FindMaxTruss}(G_{\text{base}},q,c,f)$

4) if $H \neq \varnothing$ then

5) return H;

6) $V_{\text{rest}} \leftarrow V_{G_M} - V_{\text{base}}$; $V_{\text{add}} \leftarrow V_{\text{base}}$;

7) 顶点集 V_{rest} 按权值降序排列;

8) while $V_{\text{rest}} \neq \varnothing$ do

9) V_{high} 设为权值最高顶点的集合;

10) $V_{\text{add}} = V_{\text{add}} \cup V_{\text{high}}$; $V_{\text{rest}} = V_{\text{rest}} - V_{\text{high}}$;

11) 由顶点集 V_{add} 构建图 G_M 的诱导子图 G_{add};

12) $H \leftarrow \text{FindMaxTruss}(G_{\text{add}},q,c,f)$;

13) if $H \neq \varnothing$ then

14) break;

15) return H;

WI 算法首先将权值大于或等于查询点的顶点加入 V_{base}，构建图 G_{base}。若图 G_{base} 中存在 CF-truss 子图，则算法终止，时间复杂度为 $O(\left|E_{G_{\text{base}}}\right| + \left|E_{G_{\text{base}}}\right|^{1.5})$。若不存在，算法继续，剩余顶点按权值由大到小依次加入图中，并检查其中是否存在 CF-truss 子图，时间复杂度为 $O(\left|E_{G_M}\right| + \left|E_{G_{base}}\right|^{1.5} + w\left|E_{G_M}\right|^{1.5})$，其中 w 表示图中最小权值与查询点权值之间不同权值的个数。另外，WI 算法的空间复杂度为 $O(\left|V_{G_M}\right| + \left|E_{G_M}\right|)$。

4 构建索引优化方法

4.1 问题分析

3.1 节中提出一种基于索引的最大 CF-truss 查询方法。该方法将查询点作为种子节点，遍历各边索引搜索图中满足约束条件的边。

本文复现了文献[8]中索引的构建方法，结果显示该方法在处理大型图时效率较低。以 LiveJournal 数据集为例，该数据集记录了 4 847 571 个用户之间的 68 993 773 条关系，构建其 ST 索引需要长达 817 h，严重影响了后续查询操作。

当前索引构建方法可分为 2 个阶段：第一阶段查找 $c = 0$ 至 $c_{\max}$ 的所有最大 $(c,0)$ - truss 子图；第二阶段按 fsup 降序对所有 $(c,0)$ - truss 子图执行删边操作，并记录 ST 索引。其中查找 $(c,0)$ - truss 的方法与 FindMaxTruss 算法类似，即遍历全图并删除 csup < c 的边。为了加快构建索引，本节根据 cycle 三角形和 flow 三角形的定义，提出一种新的删边更新策略，加快查找 $(c,0)$ - truss；然后提出并行构建索引方法，降低构建索引的时间复杂度。

4.2 删边更新策略

以边 $e =< u,v >$ 为例，当前删边更新的步骤如下：首先删除边 e，然后查找受影响边，最后更新受影响边的支持度。

定理 2 在有向图中，若给定三角形一条边的方向，其余边方向随机，则存在 4 个不同的三角形。

证明 已知三角形一条边的方向，其余边方向随机，则其余 2 条边的方向各有 2 种可能，可构成 4 个不同的三角形，如图 5 所示，其中包括一个 cycle 三角形和 3 个不同的 flow 三角形。证毕。

定理 3 给定一个有向图 G，对于任意边 $e =< u,v >$，如果移除 e，那么对于任何受到影响的边 e_{aff}，$\text{csup}(e_{\text{aff}})$ 至多减 1，$\text{fsup}(e_{\text{aff}})$ 至多减 2。

证明　根据图 5 中 cycle 三角形和 flow 三角形的实例，如果给定 2 条邻边，则至多存在 2 条边能与之构成三角形，且构成的三角形只有 2 种可能：一个 cycle 三角形和一个 flow 三角形；2 个 flow 三角形。所以删除其中任意一条边，其余边支持度 $\mathrm{csup}(e_{\mathrm{aff}})$ 至多减 1，$\mathrm{fsup}(e_{\mathrm{aff}})$ 至多减 2。证毕。

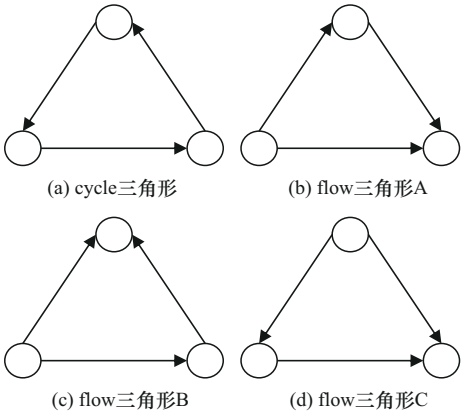


图 5　cycle 三角形和 flow 三角形的示意

根据定理 2 和定理 3，寻找受影响边时可同步更新边支持度，加快删边更新速度。基于此，本文设计了一种新的删边更新策略：删除边 e，通过公共邻居，查找与该边共同构成各类三角形的邻边，并将边支持度对应减 1。具体实现细节如算法 7 所示。删除边 e，根据图 5 查找各类三角形对应的公共邻居，并将其中各边支持度对应减 1，E_{update} 记录支持度更新的边（第 2~19 行）。以 cycle 三角形为例，公共邻居 w 为顶点 u 的入邻居和顶点 v 的出邻居的交集（第 3 行），更新时将边 $<v,w>$ 和边 $<w,u>$ 的 csup 减 1（第 4 行）。

例 8　给定图 6 所示有向图 G、删除边 $e=<v_1,v_2>$，根据图 5(a)，cycle 三角形对应的公共邻居为 v_4，更新后 $\mathrm{csup}(<v_2,v_4>)=0$、$\mathrm{csup}(<v_4,v_1>)=0$；根据图 5(b)，flow 三角形对应的公共邻居为 v_3，更新后 $\mathrm{fsup}(<v_1,v_3>)=1$、$\mathrm{fsup}(<v_3,v_2>)=0$；根据图 5(c)，flow 三角形对应的公共邻居为 v_3，更新后 $\mathrm{fsup}(<v_1,v_3>)=0$、$\mathrm{fsup}(<v_1,v_3>)=0$、$\mathrm{fsup}(<v_2,v_3>)=0$；图 G 中不存在图 5(d)所示 flow 三角形，算法结束，返回 $E_{\mathrm{update}}=\{<v_2,v_4>,<v_4,v_1>,<v_1,v_3>,<v_3,v_2>,<v_2,v_3>\}$。

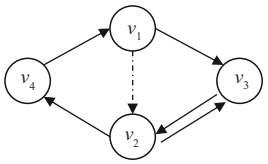


图 6　CF-truss 删边更新策略示例

算法 7　UpdateSEdge 算法

输入　有向图 G，需删除的边

输出　支持度更新的边的集合 E_{update}

1) $E_{\mathrm{update}} \leftarrow \varnothing$;
2) 删除图 G 中的边 e;
3) for $w \in N^+(u) \cap N^-(v)$ do
4)　$\mathrm{csup}(e^c)=\mathrm{csup}(e^c)-1, \forall e^c \in \{<v,w>,<w,u>\}$;
5)　$E_{\mathrm{update}} =< v,w > \cup < w,u > \cup E_{\mathrm{update}}$;
6) end for;
7) for $w \in N^-(u) \cap N^+(v)$ do
8)　$\mathrm{fsup}(e^f)=\mathrm{fsup}(e^f)-1, \forall e^f \in \{<w,v>,<u,w>\}$;
9)　$E_{\mathrm{update}} =< w,v > \cup < u,w > \cup E_{\mathrm{update}}$;
10)end for;
11)for $w \in N^+(u) \cap N^-(v)$ do
12)　$\mathrm{fsup}(e^f)=\mathrm{fsup}(e^f)-1, \forall e^f \in \{<w,v>,<w,u>\}$;
13)　$E_{\mathrm{update}} =< w,v > \cup < w,u > \cup E_{\mathrm{update}}$;
14)end for;
15)for $w \in N^-(u) \cap N^-(v)$ do
16)　$\mathrm{fsup}(e^f)=\mathrm{fsup}(e^f)-1, \forall e^f \in \{<v,w>,<u,w>\}$;
17)　$E_{\mathrm{update}} =< v,w > \cup < u,w > \cup E_{\mathrm{update}}$;
18) end for;
19) return E_{update};

本文提出的删边更新策略将查找受影响边和更新边支持度合并为一个步骤，删边更新过程中只执行一次公共顶点查找。该策略的时间复杂度为 $O(\deg(u)+\deg(v)+\deg(u)\deg(v))$，优于当前的删边更新方法。

4.3　并行构建算法

当前索引构建方法分为以下 2 个阶段。

第一个阶段查找并存储 c=0 至 $c_{\max}$ 的所有最大 $(c,0)$ - truss 子图。根据定理 1，若 H_1 为 $(c,0)$ - truss

子图，H_2为$(c-1,0)$ - truss 子图，则$H_1 \subseteq H_2$。因此，该阶段中$(c-1,0)$ - truss 和$(c,0)$ - truss 之间存在依赖关系。

第二个阶段按 fsup 降序对所有$(c,0)$ - truss 子图执行删边操作，并记录 ST 索引。由于第一个阶段中已找到并存储了所有最大$(c,0)$ - truss 子图，因此执行第二个阶段时，$(c-1,0)$ - truss 和$(c,0)$ - truss 之间不存在依赖关系，可并行实现各个$(c,0)$ - truss 子图的删边操作，即每个线程中只执行一个$(c,0)$ - truss 子图的删边操作。

因此，本文提出并行索引构建 PST（Parallel Skyline Trussness）算法，思路如下：第一个阶段，查找并存储 c=0 至 $c_{\max}$ 的所有最大子图；第二个阶段，对所有$(c,0)$ - truss 子图多线程并行执行删边记录索引操作。

PST 索引的具体实现方法如算法 8 所示。初始化删除队列Q和结果子图H（第 1 行）。计算图中各边的支持度（第 2 行）。当$f=0$时，规模最大的$(c,0)$ - truss 子图可以用$D(c,0)$表示，$D(0,0)$为原图G。从c=1 开始，删除所有 csup < c 的边，查找并存储$D(0,0)$至$D(c_{\max},0)$的所有子图（第 3~20 行）。然后，多线程并行对所有$D(c,0)$子图执行 fsup 降序删边操作，并行构建 ST 索引（第 21~37 行）。以$D(c,0)$子图为例，将 fsup 最小的边存入队列Q（第 24~25 行）。逐一处理队列中各边：首先，根据 ComputeST 函数判断(c,f)可否加入 ST(e)，然后执行删边更新操作，最后将更新后满足 csup < c 或 fsup < f的边加入队列Q（第 28~32 行）。重复上述步骤，直到$D(c,0)$中所有边都删除（第 23~35 行）。

函数 ComputeST 的作用是判断(c,f)能否加入 ST 索引，如算法 9 所示。具体来说，如果 ST(e)中存在索引能被(c,f)支配，则删除被支配的索引并将(c,f)加入 ST(e)中（第 2~4 行）；如果 ST(e)中的所有索引都不能支配(c,f)，则将(c,f)加入 ST(e)中，否则不做任何操作（第 5~9 行）。

算法 8 PST 算法

输入 有向图G

输出 ST 索引

1) $Q \leftarrow \varnothing$; $H \leftarrow \varnothing$;
2) 计算图G中各边的 csup 支持度和 fsup 支持度;
3) $c \leftarrow 0$; $f \leftarrow 0$; $D(0,0) \leftarrow G$;
4) for $c \leftarrow 1$ to $c_{\max}$ do
5) for $e \in E_G$ do
6) if $\text{csup}_G(e) < c$ then
7) Q.push(e);
8) end if;
9) end for;
10) while $Q \neq \varnothing$ do
11) pop out $e =< u,v >$from Q;//边 e 从队列 Q 中弹出
12) $E_{\text{update}} \leftarrow$ UpdateSEdge(G,e);
13) for e in E_{update} then
14) if csup$(e) < c$ then
15) Q.push(e);
16) end if;
17) end for;
18) end while;
19) $D(c,0) \leftarrow G$;
20) end for;
21) 并行执行 for $c \leftarrow 0$ to $c_{\max}$ do
22) $H \leftarrow D(c,0)$; $Q \leftarrow \varnothing$;
23) while $H \neq \varnothing$ do
24) Q中存储图H中 fsup 最小的边;
25) f储存图H中最小的 fsup;
26) while $Q \neq \varnothing$do
27) pop out $e =< u,v >$from Q;//边 e 从队列 Q中弹出
28) ComputeST(e,c,f);
29) $E_{\text{update}} \leftarrow$ UpdateSEdge(G,e);
30) for e in E_{update} do
31) if (csup$(e) < c$or fsup$(e) < f$) then
32) Q.push(e);
33) end if;
34) end for;
35) end while;
36) end while;
37) end for;

算法 9 ComputeST(e,c,f)

1) for$(c',f') \in$ ST(e) do
2) if$(c',f') < (c,f)$ then
3) 用(c,f)代替 (c',f');

4)　end if;

5)　if $(c',f')>(c,f)$ then

6)　　return;

7)　end if;

8) end for;

9)$\mathrm{ST}(e)\leftarrow \mathrm{ST}(e)\cup\{(c,f)\}$;

例 9　给定图 2 所示有向图 G，首先计算每条边的支持度，然后找到 c=0 至 $c_{\max}$ 的所有 $D(c,0)$ 子图，其中 $D(0,0)$ 为顶点 $\{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8,v_9\}$ 构成的子图，$D(1,0)$ 为顶点 $\{v_1,v_2,v_3,v_4,v_5,v_7,v_8\}$ 构成的子图。接着，多线程并行对所有 $D(c,0)$ 子图执行 fsup 降序删边操作，计算 ST 索引。以 $D(1,0)$ 子图为例，删除边 $<v_5,v_8>$，由于 $(1,1)$ 支配 $\mathrm{ST}(<v_5,v_8>)=(0,0)$，因此删除 $(0,0)$ 并将 $(1,1)$ 加入 $\mathrm{ST}(<v_5,v_8>)$。更新 $<v_5,v_7>$、$<v_7,v_5>$、$<v_8,v_7>$ 的支持度，由于 $\mathrm{fsup}(<v_8,v_7>)=\mathrm{fsup}(<v_5,v_7>)=0<1$，因此继续删除边 $<v_5,v_7>$ 和 $<v_8,v_7>$。重复以上操作，直到 $D(1,0)$ 中所有边都被删除，最终构建结果与原构建方法结果相同，如图 4 所示。

第一个阶段中，查找 $D(c,0)$ 子图的时间复杂度为 $O(|E_G|^{1.5})$。第二个阶段中，多线程并行计算 ST 索引的时间复杂度为 $O\left(\frac{(c_{\max}+1)|E_G|^{1.5}}{t}\right)$，$t$ 为线程数。因此，PST 算法的时间复杂度为 $O\left(|E_G|^{1.5}+\frac{(c_{\max}+1)|E_G|^{1.5}}{t}\right)$。并行操作需确保各线程相互独立，因此索引构建速度存在下限。具体来说，速度下限由 $D(c,0)$ 子图的查询时间和 $D(0,0)$ 子图的删边时间决定。因此，PST 算法时间复杂度至少为 $O(|E_G|^{1.5})$。另外，索引构建中需存储所有 $D(c,0)$ 子图，空间复杂度为 $O\left((c_{\max}+1)(|V_G|+|E_G|)\right)$。

5　实验结果与分析

5.1　实验设置

实验运行环境为 Linux 服务器，CPU 为 Intel Xeon Gold 6142 @2.67 GHz，内存为 120 GB，编码语言为 C++。

实验采用了以下 8 个真实数据集：DB（Dblp-cite），EM（Email），TW（Twitter），BS（Berk-Stan），WT（WikiTalk），TC（Topcats），PK（Pokec），LJ（LiveJournal）。其中，DB 来自网络数据集库，其余数据集均来自斯坦福网络分析项目。具体来说，DB 数据集是基于 DBLP 的学术引用网络，EM 是电子邮件网络，TW 是 Twitter 社交网络，BS 是 berkeley 和 stanford 域下的超链接网络，WT 是维基百科的会话网络，TC 是维基百科超链接网络，PK 是 Pokec 社交网络，LJ 是 LiveJournal 社交网络，数据集详细情况如表 2 所示。

表 2　　**数据集详细情况**

数据集	$\|V\|$	$\|E\|$	$c_{\max}$	$f_{\max}$	三角形个数/个
DB	12 591	49 743	1	8	133 900
EM	265 214	420 045	12	39	267 313
TW	81 306	1 768 149	41	139	13 082 506
BS	685 230	7 600 595	161	306	64 690 980
WT	2 394 385	5 021 410	28	91	9 203 519
TC	1791 489	28 508 141	36	51	52 106 893
PK	1 632 803	30 622 564	18	54	32 557 458
LJ	4 847 571	68 993 773	247	750	285 730 264

需要注意的是，除 DB 外，其余数据集均为不带权的有向图。为了评估算法在不同权值条件下的表现，本文根据不同的权值分布为数据集赋予了 0~100 的权值。EM、TW 采用幂律分布，BS、WT 采用正态分布，而 TC、PK 和 LJ 则采用随机分布。

5.2　算法效能评估

本文在现有内聚子图和影响力社区的基础上将两者融合，深入研究了内聚性和影响力对社区搜索问题的共同影响。为了更好地对比现有算法和本文提出的算法所求社区结构的差异，本文以 DBLP 为例进行说明。DBLP 是一个涵盖了计算机科学领域的学术文献数据库，提供了大量有关计算机科学的文献元数据、引用和作者信息。本文收集了 DBLP 中与文献[4]相关的文献，并构建了如图 7(a)所示的学术网络。

图 7(a)中每个顶点代表一篇文献，顶点权值为该文献的被引用数，有向边 $<u,v>$ 代表文献 u 引用文献 v。根据有向图上影响力社区搜索问题的定义，本文设定查询需求为查找与文献[4]关系紧密的高

引用量文献社区。由于文献之间的引用关系无法构成一个环，即不存在cycle三角形，因此将查询条件设置为q=1、c=0、f=10。

图7展示了本文算法WI（图7(b)）、CF-truss搜索算法[7]（图7(c)）以及影响力社区搜索算法IC[14]（图7(d)）的查询结果。表3详细展示了不同算法的社区结构。其中，f_{min}表示图中各边构成的最小flow三角形个数，由于文献之间的引用关系无法构成cycle三角形，因此仅用f_{min}来衡量社区的内聚性，数值越大表示文献之间的关联性越强；SW表示图7中所有文献的最小引用量；ASW表示图7中所有文献引用量的平均值。

首先，比较CF-truss搜索算法和本文提出的WI算法，结果如图7(b)、图7(c)及表3前两行所示。可以看出，尽管CF-truss搜索得到的子图规模稍大于本文提出的算法，但由于CF-truss搜索算法仅关注内聚性，社区中存在许多低引用量的顶点，如顶点13~顶点16。相比之下，WI算法综合考虑了内聚性和影响力，删除了低引用量的顶点，具有更高的社区影响力。

其次，比较IC算法和本文提出的WI算法，结果如图7(b)、图7(d)及表3第一行、第三行所示。可以看出，IC算法专注于影响力，结果使社区具有最大的子图影响力。然而，IC算法得到的结果存在以下2个方面的问题。一方面，部分顶点之间关系并不紧密，比如顶点4仅与顶点6、9、

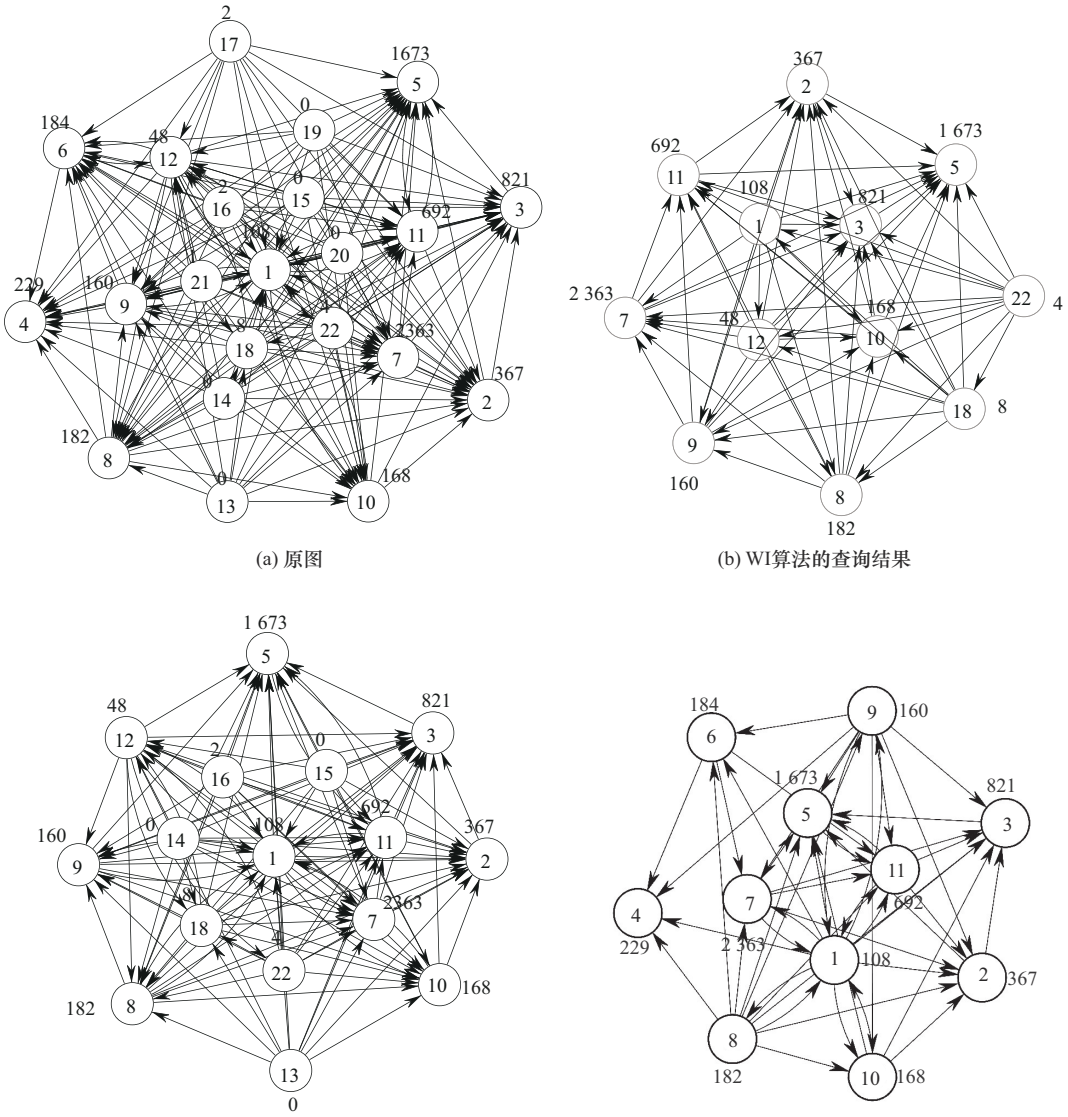


图7　效能评估

1、8之间存在引用关系。相比之下，本文提出的算法得到的结果中顶点之间联系更加紧密。另一方面，由于没有考虑方向，IC算法通常更关注时间久但引用量大的文献，忽略了近几年发表但引用量相对较少的文献。如表3中SW列所示，本文提出的算法所得社区中顶点的最小影响力为4，IC算法所得社区中顶点的最小影响力为108，然而IC算法中最新的文献发表于2019年，相比较而言，本文提出的算法中最新的文献发表于2023年，具有更好的时效性。最后，根据表3的ASW列可知，尽管两者所得社区中最小影响力的值差异较大，但社区中顶点的平均影响力相差并不大，这说明本文算法所得社区既关注了内聚性，也关注了方向所导致的时效性。

表3　　不同算法的社区结构

算法	$\|V\|$	$\|E\|$	$f_{\min}$	SW	ASW
WI	12	66	10	4	535
CF-truss	16	116	10	0	412
IC	11	51	0	108	631

5.3　算法性能比较

实验的评价指标为查询时间和构建索引时间。为了更全面地评估不同查询情况下各种算法的性能，实验中改变了查询点q、约束条件c、f等相关参数。每个实验各运行200次查询，最终的实验结果为这200次查询时间的平均值。若索引的构建时间超过24 h，则记录为24 h。若算法的查询时间超过24 h，则将总查询时间记录为24 h，平均查询时间记录为432 000 ms。

实验中用于比较的算法包括Top-down算法（TD）、Bottom-up算法（BU）、Weight-based Insertion算法（WI）、基于索引优化的Top-down算法（I-TD）、基于索引优化的Bottom-算法（I-BU）、基于索引优化的WI算法（I-WI）以及索引构建方法ST、PST、PST+。其中，ST是文献[8]中提出的ST构建方法，PST和PST+都是本文中提出的基于删边更新策略和并行思想的索引构建方法，PST的线程数为1，PST+的线程数为10。

5.3.1　查询时间

图8展示了TD、BU、WI、I-TD、I-BU、I-WI这6种算法在8个数据集上的查询时间。通过对比图8的实验结果可知，本文提出的I-WI算法在查询效率和延展性方面表现最优。在LJ数据集中，TD、BU、WI、I-TD、I-BU的总查询时间均超过24 h，而I-WI算法总查询时间为3 h。

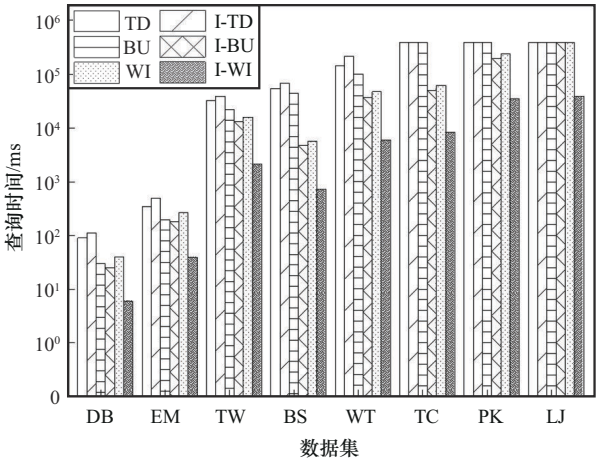


图8　6种算法在8个数据集上的查询时间

相比较而言，TD、BU、WI这3种算法未采用索引优化策略，其性能主要取决于删边次数。随着图规模的增大，删边操作次数急剧增加，导致这3种算法的查询时间急剧恶化，无法在24 h内完成TC、PK、LJ数据集的查询。与这3种算法相比，I-TD算法和I-BU算法能够在规定时间内完成TC和PK数据集的查询，但仍无法完成LJ数据集的查询。这是因为I-TD算法需要遍历全图搜索影响力最大的CF-truss，I-BU算法虽然限制了搜索范围，但存在冗余迭代，这2种算法在大规模图上查询速度较慢。相较于这2种算法，I-WI算法采用了影响力优先策略，有效限制了搜索范围并减少了冗余迭代，能够在24 h内完成大规模图查询，并且查询效率远优过其余算法。比如在BS数据集上，I-WI算法的查询速度相较于其他算法有着显著提升：与TD算法相比提升了81倍，与BU算法相比提升了105倍，与WI算法相比提升了60倍，与I-TD算法相比提升了6倍，与I-BU算法相比提升了8倍。

图9展示了根据度数改变查询点对查询时间的影响。将顶点按照其总度数由小到大进行排序，然后依次随机选取总度数前20%、20%~40%、40%~60%、60%~80%、80%~100%的顶点作为查询点。需注意的是，本次实验中约束条件为c=1、f=0。通过对比图9(a)和图9(b)的实验结果可知，在BS和WT数据集上，随着总度数的增加，3种算法的

查询时间均呈上升趋势，且I-TD算法和I-BU算法增长幅度明显大于I-WI算法。其原因有2个：首先，根据CF-truss的定义，边支持度和顶点总度数之间存在相关性。如果查询点的总度数增加，则删边次数也随之增加，从而导致查询时间增加；其次，算法采用了不同的搜索策略，查询点总度数对查询时间的影响程度也各不相同。具体来说，I-TD算法多次遍历全图查找影响力最大的CF-truss，查询点总度数越大，遍历次数越多；I-BU算法采用逐层建图的方式来寻找最大影响力子图，查询点的总度数越大，构建的子图规模也越大。相比之下，I-WI算法采用影响力优先策略，每次从影响力较大的子图入手查找CF-truss子图，受邻居节点的影响较小，因此随着查询点总度数的增长，查询时间缓慢增长。另外，此实验中I-WI算法的查询速度依然是最快的。在WT数据集上，I-WI算法的查询时间比I-TD算法快77%，比I-BU算法快80%。

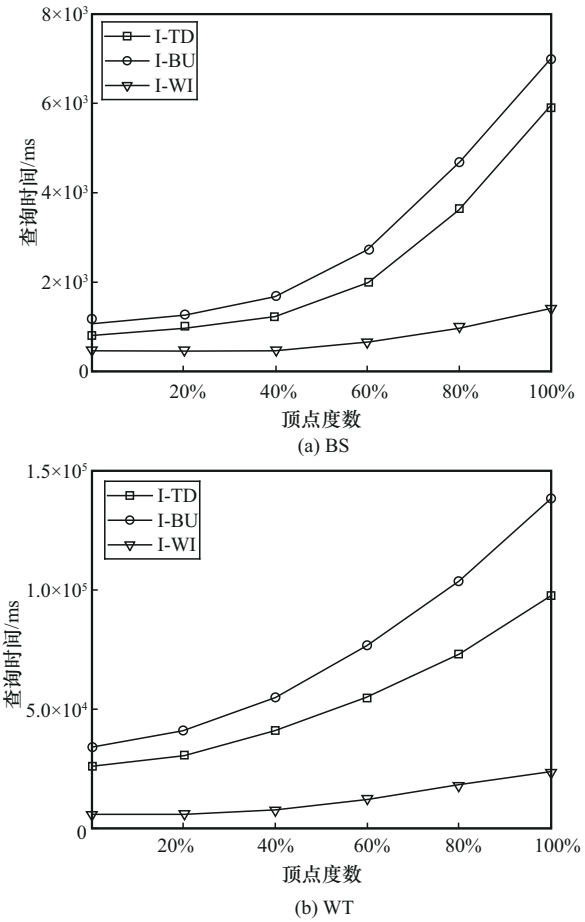


图9　根据度数改变查询点对查询时间的影响

图10根据权值改变查询点对查询时间的影响。其中，顶点按权值由小到大排序，然后依次随机选取权值为0~20、20~40、40~60、60~80、80~100的顶点作为查询点。通过对比图10(a)和图10(b)的实验结果可知，在BS和WT数据集上，随着查询点权值的增加，I-TD、I-BU和I-WI算法的查询时间都趋于平稳。其主要原因是CF-truss与边支持度有关，与顶点权值无关。尽管I-TD、I-BU、I-WI算法主要通过删点扩大子图影响力，但有向图上的影响力社区搜索问题是在CF-truss的基础上进一步寻找影响力最大的子图，因此查询点权值对查询时间的影响有限。另外，此实验中I-WI算法的查询速度也是最快的。

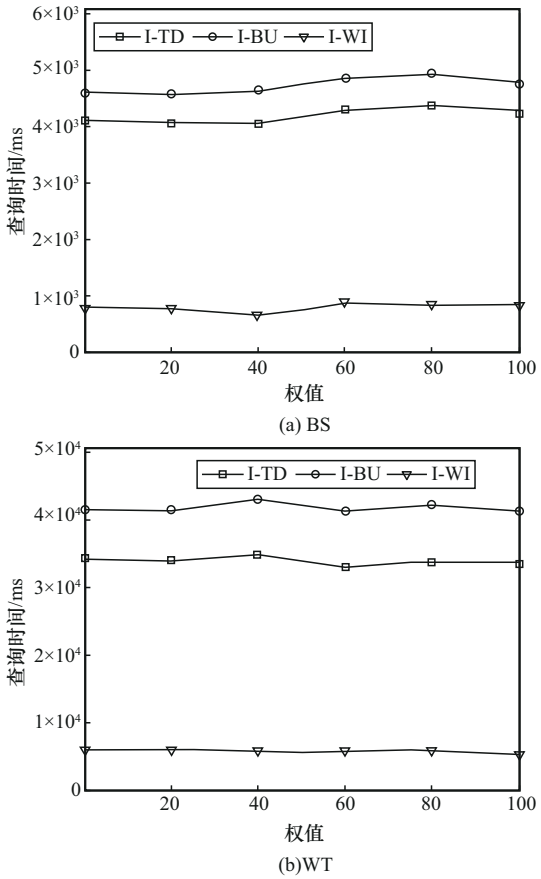
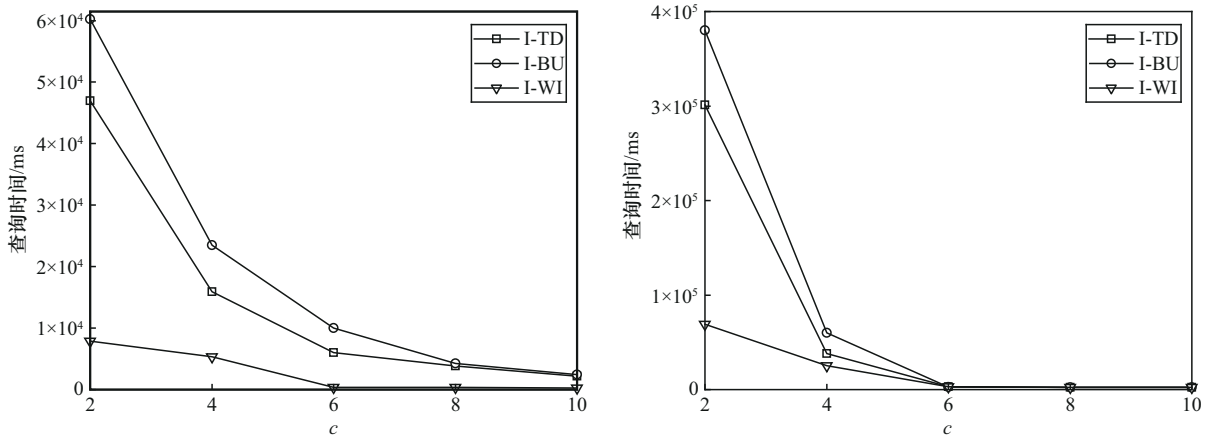


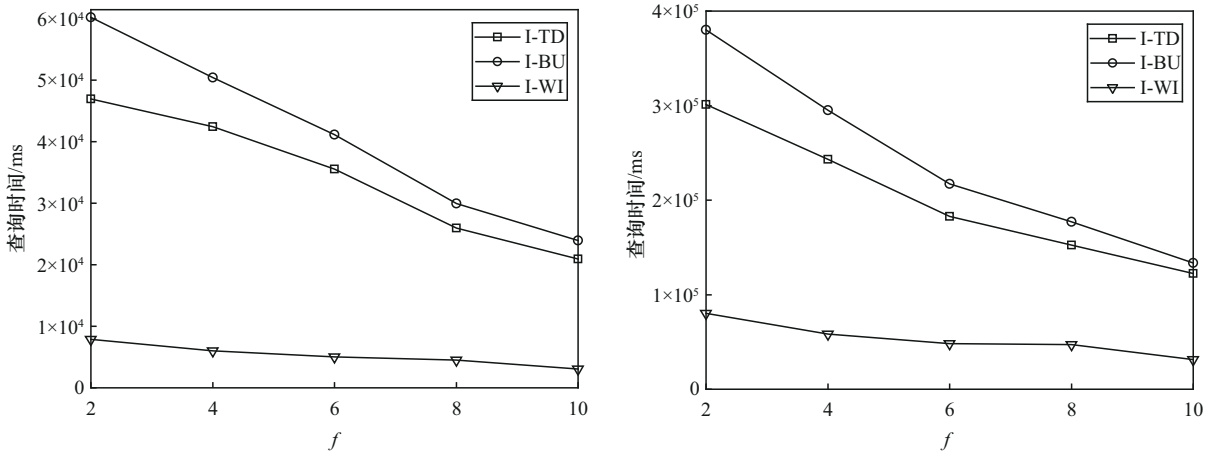
图10　根据权值改变查询点对查询时间的影响

图11和图12分别展示了改变参数c和f对查询时间的影响。在图11中，当c从2增长至10时，TW、PK数据集上4种算法的查询时间均呈明显下降趋势；在图12中，当f从2增长至10时，TW、PK数据集上4种算法的查询时间同样呈现



(a) TW　　(b) PK

图11　c变化对查询时间的影响



(a)TW　　(b) PK

图12　f变化对查询时间的影响

明显的下降趋势，并且图11中的下降幅度明显大于图12中的下降幅度。其原因是当c或f增大时，最大CF-truss子图就越紧密，从而加快了算法的查询速度。此外，根据表2中的相关信息可知，每个数据集的f_{max}均大于c_{max}。因此，$(c+1,f)$-truss子图的规模小于$(c,f+1)$-truss子图的规模，查询速度也更快。

5.3.2　构建索引时间及索引大小

图13展示了不同数据集上各算法的索引构建时间，表4列出了不同数据集上ST索引的大小。通过对图13中各算法的查询时间进行分析可知，在所有数据集上，本文提出的PST算法构建索引的速度均优于基础构建方法。此外，10线程并行构建索引的速度快于单线程构建索引的速度，特别是当数据集规模较大时，并行构建索引的效果更加显著。

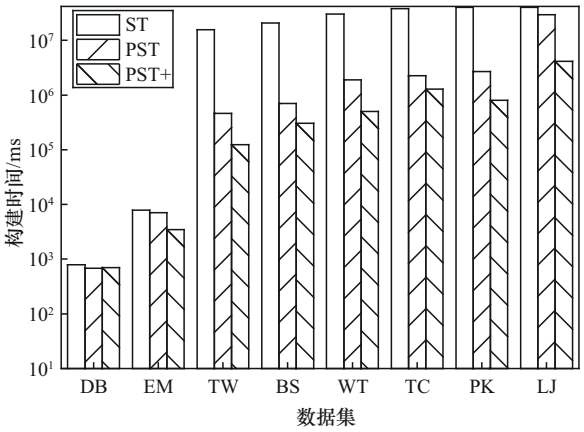


图13　不同数据集上各算法的索引构建时间

具体来说，对于LJ数据集，ST算法运行时间超过24 h，PST算法运行时间为8 h，PST+算法运行时间少于1.5 h。在TW、BS、PK和LJ这4个数据集上，PST算法相较于ST算法至少提升了15倍，

在采用10线程并行的情况下，速度相较于单线程提升了3倍以上。值得一提的是，在LJ数据集上，采用10线程并行构建索引的速度相较于单线程提升了7倍。其原因是PST算法对ST算法中的删边方法以及实现架构进行了调整，提出了删边更新策略和并行构建索引方法，所以PST算法的索引构建时间优于ST算法的构建时间，且线程数越多，构建速度越快。

表4　　不同数据集上ST索引的大小

数据集	索引大小/MB
DB	1
EM	3
TW	38
WT	48
BS	64
TC	230
PK	257
LJ	1 380

需要说明的是，鉴于DB数据集规模小，且各边最大的cycle三角形个数仅为1，难以有效展现并行算法的改进效果。因此，在后续实验中决定排除该数据集，仅在除DB数据集外的其他数据集上进行多线程的比较。

图14展示了不同线程数对索引构建时间的影响。随着线程数从1增加至10，所有数据集上的加速比都呈上升趋势。在EM、WT、TC数据集上，加速比均未超过2.5，并且在4线程时加速比都达到了最大值，其中EM数据集的最大加速比为2；WT数据集的最大加速比为2.3；TC数据集加速效果最差，加速比仅为1.5；TW、BS、PK、LJ数据集的加速比均大于3。例如，TW数据集在7线程时达到最大加速比3.7，BS数据集在9线程时达到最大加速比3.8，PK数据集在10线程时达到最大加速比3.4，LJ数据集在10线程时最大加速比达到了7。以上结果主要与PST算法的构建方法有关。并行构建索引需确保各线程相互独立，因此索引构建速度存在下限，且速度下限由$D(c,0)$子图的查询时间和$D(0,0)$子图的删边时间决定。通过速度下限，计算得到每个数据集可加速时间的占比：EM为57%，TW为79%，BS为79%，WT为64%，TC为50%，PK为76%，LJ为90%。将数据集按可加速时间占比降序排列，得到以下顺序：LJ、BS、TW、PK、WT、EM、TC，与图14中的加速趋势完全一致。以TC数据集为例，其可加速时间仅占查询时间的50%，如图14(e)所示，所以TC数据集的最大加速比仅为1.5。

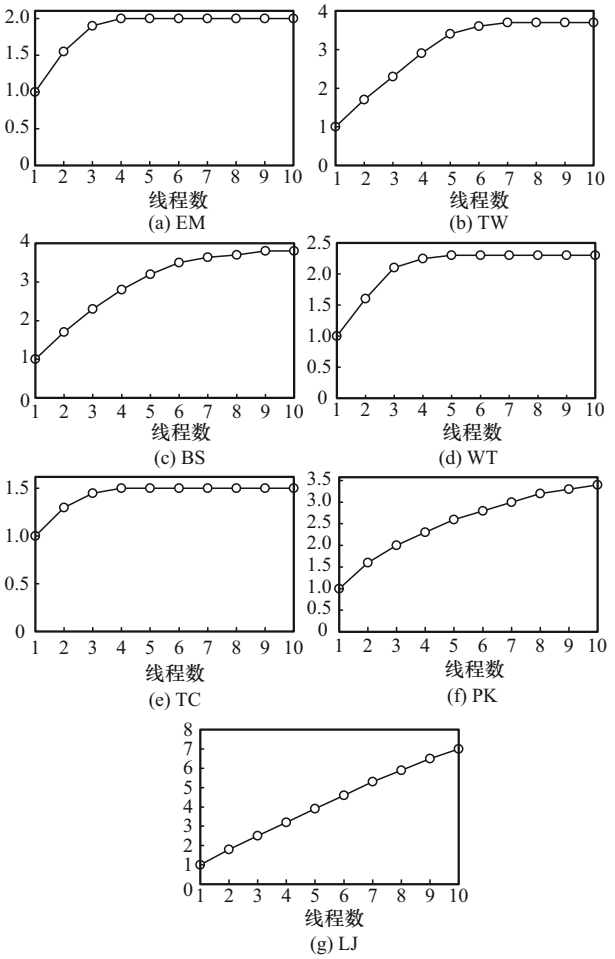


图14　不同线程数对索引构建时间的影响

6　结束语

本文提出了有向图上的影响力社区搜索问题，并设计了在线搜索算法。为了加快查询速度，进一步提出了基于索引和影响力优先策略的优化算法I-WI。该算法通过索引存储各边的ST，并结合影响力优先搜索策略，有效限制了搜索范围，提升了查询效率。此外，针对索引构建缓慢的问题，本文分析了索引构建的各个环节，并将其中的分解过程转换成并行，从而显著缩短了构建时间。实验结果显示，本文提出的算法能够快速并有效地查找有向图中的影响力社区。以BS数据集为例，I-WI算法的

查询效率较TD算法提升了80倍，较BU算法提升了102倍。此外，实验还证明了并行索引构建算法能够显著缩短索引构建时间。以LJ数据集为例，采用10线程构建索引的速度比单线程快了7倍，比基础构建方法ST算法快了400倍左右。

参考文献：

[1] ZHOU Y L, FANG Y X, LUO W S, et al. Influential community search over large heterogeneous information networks[J]. Proceedings of the VLDB Endowment, 2023, 16(8): 2047-2060.

[2] ZHANG F, GUO H C, OUYANG D, et al. Size-constrained community search on large networks: an effective and efficient solution[J]. IEEE Transactions on Knowledge and Data Engineering, 2024, 36(1): 356-371.

[3] LI Q Q, MA H F, LI Z X, et al. Multiresolution local spectral attributed community search[J]. ACM Transactions on the Web, 2024, 18(1): 1-28.

[4] MALLIAROS F D, GIATSIDIS C, PAPADOPOULOS A N, et al. The core decomposition of networks: theory, algorithms and applications[J]. The VLDB Journal, 2020, 29(1): 61-92.

[5] ZONG C Y, GONG P C, ZHANG X, et al. Efficient size-constrained (k, d) -truss community search[C]//International Conference on Advanced Data Mining and Applications. Berlin: Springer, 2023: 405-420.

[6] FANG Y X, WANG Z R, CHENG R, et al. Effective and efficient community search over large directed graphs[J]. IEEE Transactions on Knowledge and Data Engineering, 2019, 31(11): 2093-2107.

[7] TAKAGUCHI T, YOSHIDA Y. Cycle and flow trusses in directed networks[J]. Royal Society Open Science, 2016, 3(11): 160270.

[8] LIU Q, ZHAO M J, HUANG X, et al. Truss-based community search over large directed graphs[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 2183-2197.

[9] TIAN A X, ZHOU A, WANG Y, et al. Maximal D-truss search in dynamic directed graphs[J]. Proceedings of the VLDB Endowment, 2023, 16(9): 2199-2211.

[10] HAJIBABAEI H, SEYDI V, KOOCHARI A. Community detection in weighted networks using probabilistic generative model[J]. Journal of Intelligent Information Systems, 2023, 60(1): 119-136.

[11] KOSMANOS K, KALNIS P, PAPADOPOULOS A. Incremental influential community detection in large networks[C]//Proceedings of the 34th International Conference on Scientific and Statistical Database Management. New York: ACM Press, 2022: 1-12.

[12] YU D X, ZHANG L F, LUO Q, et al. Maximal clique search in weighted graphs[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(9): 9421-9432.

[13] ZHOU W, HUANG H, HUA Q S, et al. Core decomposition and maintenance in weighted graph[J]. World Wide Web, 2021, 24(2): 541-561.

[14] LI R H, QIN L, YU J X, et al. Influential community search in large networks[J]. Proceedings of the VLDB Endowment, 2015, 8(5): 509-520.

[15] 杜明, 宋嘉祎, 周军锋. 规模受限的影响力社区搜索[J]. 电子学报, 2023, 51(5): 1207-1214.
DU M, SONG J Y, ZHOU J F. Size-constrained influential community search[J]. Acta Electronica Sinica, 2023, 51(5): 1207-1214.

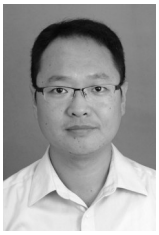
[16] DORANTES-GILARDI R, GARCÍA-CORTÉS D, HERNÁNDEZ-LEMUS E, et al. K-core genes underpin structural features of breast cancer[J]. Scientific Reports, 2021, 11(1): 16284.

[17] WILLIAMS H T P, MCMURRAY J R, KURZ T, et al. Network analysis reveals open forums and echo chambers in social media discussions of climate change[J]. Global Environmental Change, 2015, 32: 126-138.

[18] KIM J, GUO T, FENG K Y, et al. Densely connected user community and location cluster search in location-based social networks[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 2199-2209.

[19] LI Q Y, ZHU Y Y, YE J H, et al. Skyline group queries in large road-social networks revisited[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(3): 3115-3129.

[作者简介]



杜明（1975– ），男，黑龙江鸡西人，博士，东华大学教授，主要研究方向为图数据处理技术、自然语言处理等。

胡欣雨（1998– ），女，上海人，东华大学硕士生，主要研究方向为社区搜索算法。

周军锋（1977– ），男，陕西西安人，博士，东华大学教授、博士生导师，主要研究方向为大图数据的查询处理技术、推荐系统关键技术等。