枚举大型 k 顶点连通分量图表

Dong Wen ，Lu Qin Xuemin Lin ‡ , Ying Zhang , and Lijun Chang ‡
CAI,澳大利亚悉尼科技大学 ‡澳大利亚新南威尔士大学
dong.wen@student.uts.edu.au； {lu.qin,
ying.zhang}@uts.edu.au; {lxue, ljchang}@cse.unsw.edu.au；
‡

抽象的

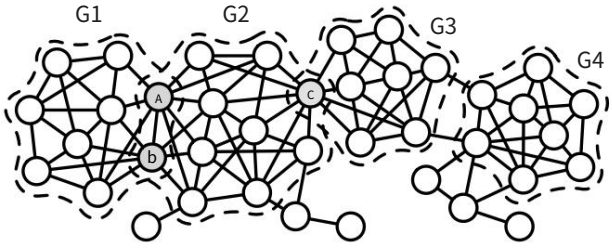
内聚子图检测是一个重要的图问题,广泛应用于许多应用领域,例如社会社区检测、网络可视化和网络拓扑分析。大多数现有的内聚子图指标可以保证良好的结构特性,但可能会导致搭便车效应。在这里,搭便车效应是指一些不相关的子图如果只共享少量的顶点和边,就会合并为一个子图。在本文中,我们研究了 k-vertex connected component (k-VCC),它可以有效地消除搭便车效应,但在文献中研究较少。 k-VCC 是连接子

移除任何 k − 1 个顶点都不会断开子图的图。除了消除搭便车效应外,k-VCC还具有直径有界、内聚性高、有界图重叠、子图数有界等优点。我们提出了一种多项式时间算法,通过将图递归地划分为重叠的子图来枚举图的所有 k-VCC。我们发现改进算法的关键是减少本地连接测试的数量。因此,我们提出了两种有效的优化策略,即邻居扫描和组扫描,以大大减少本地连接测试的数量。我们使用七个大型真实数据集进行了广泛的性能研究,以证明该模型的有效性以及我们提出的算法的效率。

一、简介

图已被广泛用于表示现实世界中实体之间的关系。随着图形应用程序的激增，研究工作已经致力于挖掘和分析图形数据的许多基本问题。最近,内聚子图检测引起了强烈的研究兴趣[22]。此类问题可以广泛应用于许多实际应用中,例如社区检测 [10, 16]、网络聚类 [26]、图形可视化 [1, 35]、蛋白质-蛋白质网络分析 [2] 和系统分析 [34]。

在文献中,已经提出了大量的内聚子图模型。其中,每对顶点都相连的团保证了顶点之间的完美熟悉度和可达性。由于集团的定义太



4 核：{G1∪G2∪G3∪G4} 4-ECC：{G1∪G2∪G3, G4} 4-VCC：{G1, G2, G3, G4}

图 1:图 G 中的内聚子图。

文献中提出了严格的 clique-relaxation 模型,包括 s-clique [19]、s-club [19]、γ-quasi-clique [33] 和 k-plex [4, 23]。然而,这些模型需要指数计算时间并且可能缺乏保证的内聚性。为了克服这个问题,提出了其他模型,例如 k-core [3]、k-truss [9、27、24]、k-mutual-friend subgraph [36] 和 k-ECC（k-edge connected component） [37, 6],这需要多项式计算时间并保证良好的内聚性。例如,k-core 保证每个顶点在子图中的度数至少为 k,而 k-ECC 保证在删除任何 k-1 条边后子图不会断开。

动机。尽管现有的内聚子图模型具有良好的结构保证,但我们发现这些模型中的大多数都不能有效消除搭便车效应。在这里,搭便车效应是指一些不相关的子图如果仅共享少量的顶点和边,则它们会合并为一个结果子图。

为了说明搭便车效应,我们考虑图 1 中所示的图 G,它包括四个子图G1、G2、 G3和G4。四个子图是松散连接的,因为： G1和G2共享一条边 (a, b)； G2和G3共享一个顶点 c； G3和G4不共享任何边或顶点。令k = 4。基于k核模型,只有一个k核,它是四个子图G1、 G2、 G3和G4的并集,以及连接G3和G4的两条边。基于k-ECC模型,有两个k-ECC,分别是G4和三个子图G1、 G3、 G3的并集。

受此启发,我们旨在检测内聚子图并有效消除搭便车效应,即准确检测G1、 G2、 G3和G4作为图 1 中的内聚子图。

在文献中,最近的一项工作 [31] 旨在消除本地社区搜索中的搭便车效应。给定一个查询顶点,[31] 中的算法试图通过根据与查询顶点的接近度对图中的每个顶点进行加权来消除搭便车效应。

基于顶点权重,通过考虑密度和与查询ver的接近度返回一个query-biased子图

特克斯。不幸的是,这种查询偏向的本地社区模型不能用于内聚子图检测。 k-顶点连通分量。顶点连通性,也称为结构内聚性 [20],

是需要移除以断开图形的最小顶点数。它已被证明是评估社会群体凝聚力的出色指标 [20, 29]。我们发现这种社会学概念可用于检测内聚子图并有效消除搭便车效应。给定整数 k,k-顶点连通分量 (k-VCC) 是最大连通子图,其中移除任何 k -1 个顶点都不能断开子图。给定图 G 和参数 k,我们的目标是检测 G 中的所有 k-VCC。在图 1 和 k = 4 中， G 中有四个 k-VCC G1、G2 、 G3和 G4 。

由G1和G2并集形成的子图不是 k-VCC,因为它会通过删除两个顶点 a 和 b 来断开连接。

效力。 k-VCC 通过确保每个 k-VCC 不能通过删除任何 k-1 个顶点断开连接,有效地消除了搭便车效应。此外,k-VCC还具有以下四种良好的结构特性。 · 有界直径。 k-VCC G (V , E) 的直径为 +1,其中 κ(G) 是以 G 为界的顶点连通性。

$$\frac{|V|-2}{k(G)}$$

例如,我们考虑图 1 中具有 9 个顶点的 4-VCC G1。G1的直径以 2 为界。 · 高内聚性。我们可以保证一个k-VCC嵌套在

一个k-ECC和一个k-core中。因此,k-VCC 通常更具凝聚力,并继承了 k-core 和 k-ECC 的所有结构特性。例如,图 1 中的四个 4-VCC 中的每一个也是一个 4-core 和一个 4-ECC。

· 子图重叠。与 k-core 和 k-ECC 不同,k-VCC 模型允许 k-VCC 之间重叠,我们可以保证任何一对 k-VCC 的重叠顶点数小于 k。例如,图 1 中的两个 4-VCC G1和G2与两个顶点和一条边重叠。 · 有界子图数。即使存在重叠,我们也可以将 k-VCC 的数量限制为与图中的顶点数量成线性关系。这表明 k-VCC 中的冗余是有限的。例如,图 1 所示

的图形包含四个 4-VCC,三个顶点 a、b 和 c 重复。

这四个属性的详细信息可以在第 2.2 节中找到。

效率。在本文中,我们提出了一种算法,通过重叠图分区枚举给定图 G 中的所有 k-VCC。简而言之,我们的目标是在 G 中找到少于 k 个顶点的顶点切割。这里,G 的顶点切割是一组顶点,删除这些顶点会断开图。通过顶点切割,我们可以将 G 划分为重叠的子图,每个子图包含切割中的所有顶点及其导出的边。我们递归地划分每个子图,直到不存在这样的切割。以这种方式,我们计算所有 k-VCC。例如,假设图 G 是图 1 中G1和G2的并集,k = 4,我们可以找到一个由两个顶点 a 和 b 组成的顶点割。因此,我们将图划分为两个子图G1和G2 ,它们与两个顶点 a、b 和边 (a, b) 重叠。由于G1和G2都没有任何少于 k 个顶点的顶点切割,我们将G1和G2作为最终的 k-VCC 返回。我们从理论上分析了我们的算法,并证明了 k-VCC 的集合可以在多项式时间内枚举。可以在第 4.3 节中找到更多详细信息。

尽管如此,上述算法还有很大的改进空间。
该算法中最关键的操作称为局部连通性测试,它给定两个顶点 u 和 v,通过从 G 中删除最多 k-1 个顶点来测试 u 和 v 是否可以在两个组件中断开连接。要找到一个顶点切割少于 k 个顶点,我们

在最坏的情况下,需要在源顶点 s 和 G 中的每个其他顶点 v 之间进行局部连通性测试。因此,提高算法效率的关键是减少图中局部连通性测试的数量。给定一个源顶点 s,如果我们可以避免测试 s 和某个顶点 v 之间的局部连通性,我们称之为我们可以扫描顶点 v。我们提出了两种扫描顶点的策略。

· 邻居扫描。如果一个顶点具有某些属性,则可以扫描它的所有邻居。因此,我们称这种策略为邻居扫描。此外,我们为每个顶点维护一个存款值,一旦我们完成测试或扫描一个顶点,我们就会增加其邻居的存款值。如果某个顶点的押金值满足一定条件,则该顶点也可以被扫除。 · 组扫描。我们介绍了一种将图中的顶点划分为不相交组的方法。如果组中的顶点具有某些属性,则可以扫描整个组中的顶点。我们称此策略组扫荡。此外,我

们为每个组维护一个组存款值。一旦我们测试或扫描组中的一个顶点,我们就会增加相应的组存款值。

如果组存款值满足一定条件,也可以扫除整个组内的顶点。

尽管这两种策略是独立研究的,但它们可以一起使用并提高彼此的有效性。通过这两种顶点扫描策略,我们可以显着减少算法中局部连通性测试的数量。实验结果表明我们的扫描策略具有出色的性能。

在第 5 节和第 6 节中可以找到更多详细信息。

贡献。我们在本文中做出以下贡献。

(1) k-VCC有效性的理论分析。我们展示了几个属性来展示 k-vertex 连接组件的卓越质量。尽管在文献中已经研究了顶点连通性的概念来评估 so 的内聚性

据我们所知,这是第一项旨在枚举所有 k-VCC 并考虑消除内聚子图检测中的搭便车效应的工作。

(2) 基于重叠图划分的多项式时间算法。我们提出了一种算法来计算图 G 中的所有 k-VCC。该算法递归地将图划分为重叠的子图,直到每个子图无法进一步划分。我们证明我们的算法在多项式时间内终止。

(3) 两种有效的剪枝策略。我们设计了两种修剪策略,即邻居扫描和组扫描,以大大减少本地连接测试的数量,从而显着加快算法速度。

(4) 广泛的性能研究。我们对 7 个真实的大图进行了广泛的性能研究,以证明 k-VCC 的有效性和我们提出的算法的效率。

大纲。本文的其余部分安排如下。第 2 节正式定义问题并介绍其基本原理。第 3 节给出了计算给定图中所有 k-VCC 的框架。第 4 节给出了框架的基本实现并分析了算法的时间复杂度。第 5 节介绍了几种加速算法的策略。第 6 节使用大量实验评估模型和算法。第 7 节回顾相关工作,第 8 节总结本文。

2. 初步

2.1 问题陈述

在本文中,我们考虑一个无向且未加权的图 G(V, E),其中 V 是顶点集,E 是边集。

我们还使用 V(G) 和 E(G) 分别表示图 G 的顶点集和边集。顶点数和边数用 n = |V | 表示。和 m = |E|分别。为简单起见且不失一般性,我们假设 G 是连通图。我们用 N(u) 表示顶点 u 的邻居集,即 N(u) = {u ∈ V |(u, v) ∈ E},并且 u 的度由 d(u) = |N(u)|。给定两个图 g 和 g， g ⊆ g 表示 g 是 g 的子图。给定一组顶点Vs,导出的子图 G[Vs]是 G 的一个子图,使得 G[Vs] = (Vs, {(u, v) ∈ E|u, v ∈ Vs})。对于G的任意两个子图g和g,我们用g∪g表示g和g的并集,即g∪g = (V(g)∪V(g),E(g)∪E(g)).在陈述问题之前,我们首先给出一些基本定义。 我们用

定义1. （顶点连通性)图 G 的顶点连通性,用 κ(G) 表示,被定义为顶点的最小数量,其移除导致不连通图或平凡图（单顶点图） .

定义2. (K-VERTEX CONNECTED)图 G 是 k 个顶点连通的,如果： 1) |V (G)| > k;和 2) 剩余的图在删除任何 (k-1) 个顶点后仍然是连接的。即,κ(G) ⩾ k。

当上下文清楚时,我们使用术语 k-connected 来简称。

很容易看出任何非平凡连通图至少是 1-连通的。基于定义 2,我们定义 k-Vertex Connected Component (k-VCC) 如下。

定义3. (K-VERTEX CONNECTED COMPONENT)给定
图 G,子图 g 是 G 的 k-顶点连通分量 (k VCC),如果： 1) g 是 k-顶点连通;和 2) g 是最大的。
也就是说,g ⊆ G,使得 κ(g) ⩾ k, g ⊆ g 。

问题定义。给定图 G 和整数 k,我们将 G 的所有 k-VCC 的集合表示为 V CCk(G)。在本文中,我们研究了有效枚举G 的所有k-VCC 的问题,即计算V CCk(G) 。

示例1. 对于图 1 中的图形 G,给定参数 k = 4,有四个 4-VCC:V CC4(G) = {G1, G2, G3, G4}。我们不能通过删除任何 3 个或更少的顶点来断开它们中的每一个。子图G1 ∪ G2不是 4-VCC,因为它在移除两个顶点 a 和 b 后将断开连接。

2.2 为什么使用k-Vertex Connected Component?

k-VCC 模型通过确保每个 k-VCC 不能通过删除任何 k-1 个顶点断开连接,有效地减少了搭便车效应。在本小节中,我们展示了 k-VCC 在有界直径、高内聚性、有界重叠和有界分量数方面的其他良好结构特性。 k-core 和 k-Edge Connected Component (k-ECC) 等其他内聚图模型都无法同时实现这四个目标。

直径。在讨论 k-VCC 的直径之前,我们首先引用 Global Menger 定理如下。

定理1. 当且仅当任何一对顶点 u,v 由至少 k 个顶点无关的 uv 路径连接时,图是 k-连通的。 [18]

该定理显示了顶点连通性和顶点独立路径的数量的等价性,这两者都被认为是图内聚的重要属性 [29]。基于此

定理,我们可以限制 k-VCC 的直径,其中图 G 的直径,用 diam(G) 表示,是 G 中任意一对顶点之间的最长最短路径：

$$\text{diam}(G) = \max_{u,v \in V(G)} \text{dist}(u, v, G) \quad (1)$$

这里,dist(u, v, G) 是 G 中顶点对 u 和 v 的最短距离。小直径被认为是 [11] 中良好社区的重要特征。我们给出 k-VCC 的直径上限如下。

定理2. 给定G 的任意 k-VCC Gi ,我们有：

$$\text{直径}(G_i) \leqslant \frac{|V(G_i)| - 1}{2\kappa(G_i)} + 1。 \quad (2)$$

证明。考虑 Gi 中的任意两个顶点 u 和 v ,我们有 d(u, v, Gi) ⩽ diam(Gi)。定理 1 表明在Gi中 u 和 v 之间至少存在 κ(Gi)条顶点不相交的路径,并且在每条路径中,我们至多有 diam(Gi)−1 个内部顶点,因为 dist(u, v, Gi) ⩽直径（Gi）。因此我们在它们之间最多有 κ(Gi)×(diam(Gi)− 1) 个内部顶点。对于两个端点 u 和 v,我们有 2 + κ(Gi) × (diam(Gi) − 1) ⩽ |V (Gi)|。因此 diam(Gi) 的上限是|V (Gi)|−2

$$\frac{}{\kappa(G_i)} + 1。 \quad \square$$

凝聚力。为了进一步研究 k-VCC 的质量,我们引入了惠特尼定理 [30]。给定图g,分析顶点连通性κ(g)、边连通性κ(g)和最小度δ(g)之间的包含关系。该定理如下所示。

定理3. 对于任何图 g,κ(g) ⩽ κ (g) ⩽ δ(g)。

从这个定理,我们知道对于图G,G的每个k-VCC嵌套在G中的一个k-ECC中,G的每个k-ECC嵌套在G中的一个k-core中。因此,k-VCC通常比 k-ECC 和 k-core 更具凝聚力。

重叠。 k-VCC 模型还支持不同 k-VCC 之间的顶点重叠,这在社交网络中尤为重要。我们可以很容易地从 k-VCC 的定义中推导出以下属性来限制重叠大小。

性质1. 给定图 G 中的两个 k-VCC Gi和Gj ， Gi和Gj的重叠顶点数小于 k。即|V (Gi) ∩V(Gj)| < k。

示例2. 在图 1 中,我们发现两个顶点 a 和 b 包含在两个 4-VCC G1和G2 中。对于 k-ECC 和 k-core,如果两个组件有一个共同的顶点,它们将被合并。

比如只有一个4核,就是G1、 G2、 G3 、 G4的并集。

组件编号。一旦我们允许不同组件之间的重叠,组件的数量就很难受到限制。 3个

例如,最大派系的数量达到了 3 个有 n 个顶点的图 [21]。尽管如此,我们发现图 ^{n}
G 中 k-VCC 的数量可以由一个与 G 中的顶点数量成线性关系的函数来界定。即：

$$|VCCk(G)| \leqslant 2^{|V(G)|} \quad (3)$$

方程式的详细讨论。 3 可以在第 4 节中找到。值得注意的是,k-VCC 的线性数量允许我们设计一个多项式时间算法来枚举所有 k-VCC。我们还将在第 4 节中详细讨论这一点。

算法 1 KVCC-ENUM(G, k)

输入:一个图G和一个整数k;
输出:所有k-顶点连通分量;

1: VCCk (G) ← ∅; 2:当
∃u : d(u) < k确实移除 u 和入射边； 3:识别G中的连通分量G = {G1,
G2, ..., Gt}； 4:对于所有连通分量Gi ∈ G做5： S ← GLOBAL-CUT(Gi, k)； 6:
如果S = ∅则7： V CCk(G) ← V CCk(G) ∪ {Gi}； 8:
else 9: Gi ← OVERLAP-PARTITION(Gi, S);
对于所有G j 10: ∈ Gi do
11: V CCk(G) ← V CCk(G) ∪ KVCC-ENUM(G i , k); 12:
返回VCCk
(G)； 13: Procedure OVERLAP-PARTITION(Graph G 14: G
G 中移除 S 中的顶点及其
对于所有连通分量 G 17: G ← G ∪ {G [V (G 18: return G;

相邻边; 16:

j

，顶点切割 S)

G做
) ∪ S]};

3. 算法框架

3.1 基于切割的框架为了计算图中的所有 k-VCC,我们引入了

基于切割的框架
本节中的框架 KVCC-ENUM。我们定义顶点切割。

定义4. （顶点切割)给定一个连通图 G,如果从 G 中移除 S 导致断开连接的图,则顶点子集 S ⊂ V 是一个顶点切割。

由定义4可知,对于给定的图G,顶点切割可能不是唯一的,顶点连通性是最小顶点切割的大小。对于完全图,没有顶点切割,因为任何两个顶点都是相邻的。顶点切割的大小是切割中的顶点数。在本文的其余部分,当上下文清楚时,我们使用 cut 来表示顶点切割。

算法。给定一个图 G,我们基于切割的框架的总体思路如下。如果 G 是 k-连通的,则 G 本身是 ak VCC。否则,必然存在一个大小小于k的合格割S。在这种情况下,我们找到这样的割并使用割将 G 划分为重叠的子图。我们重复分区过程,直到每个剩余的子图都是 k-VCC。由定理3可知,一个k-VCC一定是一个k核（最小度数不小于k的图）。因此我们可以提前计算所有 k-cores 以减少图的大小。

我们框架的伪代码在算法 1 中给出。
在第 2 行中,该算法通过迭代删除度数小于 k 的顶点来计算 k-core,并在不存在这样的顶点时终止。然后我们识别输入图 G 的连通分量。对于每个连通分量Gi （第 4 行），我们首先通过调用子程序 GLOBAL-CUT（第 5 行)找到Gi的切割。

在这里,我们只需要找到一个少于 k 个顶点的切割而不是最小切割。后面会介绍GLOBAL-CUT的具体实现。如果没有这样的切割,则意味着Gi是 k-连通的,我们将其添加到结果列表 V CCk(G)（第 6-7 行）。

否则,我们通过调用 OVERLAP-PARTITION（第 9 行)使用切割 S 将图划分为重叠的子图。我们使用相同的过程 KVCC-ENUM（第 11 行)递归地切割每个其他子图,直到所有剩余的子图都是 k-VCC。

接下来介绍子程序OVERLAP PARTITION,通过割S将图划分为重叠的子图。

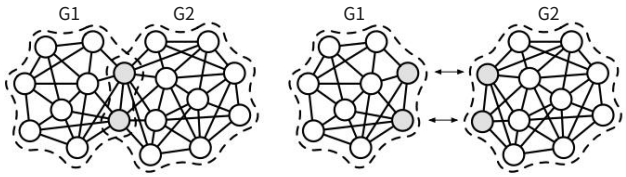


图 2:重叠图分区示例。

重叠图分区。要使用割 S 将图 G 划分为重叠的子图,我们不能简单地删除 S 中的所有顶点,因为这些顶点可能是两个或多个 k-VCC 的重叠顶点。子程序 OVERLAP-PARTITION 显示在

算法 1 的第 13-18 行。我们首先从 G 中删除 S 中的顶点及其相邻边。删除 S 后 G 将断开连接,因为 S 是 G 的顶点切割。我们可以简单地将切割 S 添加到 G 的每个连通分量 G 中,并返回导出子图 G [V (Gi) ∪ S] 作为分区子图（第 17-18 行）。分区子图相互重叠,因为切割 S 在这些子图中是重复的。下面,我们用一个例子来说明分区过程。

示例3. 我们考虑图 2 左侧的图 G。给定输入参数 k = 3,我们可以找到一个顶点切割,其中所有顶点都用灰色标记。这些顶点属于 3-VCC， G1和G2。因此,给定图 G 的切割 S,我们通过复制切割 S 的诱导子图来划分图。如图 2 右侧所示,我们通过复制两个切割顶点获得两个 3-VCC ， G1和G2和它们的内边缘。

3.2 算法正确性

在本节中,我们使用以下方法证明 KVCC-ENUM 的正确性
以下引理。

引理1. KVCC-ENUM 返回的每个子图
k-顶点连接。

证明。我们用反证法证明。假设其中一个结果子图Gi不是 k-连通的。第 5 行中的 GLOBAL-CUT 将找到顶点切割。 Gi会在第9行被分割,无法返回,这与Gi在结果列表中是矛盾的。

□

LEMMA 2. (Completeness) KVCC-ENUM 返回的结果
包含输入图 G 的所有 k-VCC。

证明。假设图 G 被划分为重叠的子图 G = { GG 2, . . .} 使用顶点切割 S。我们首先证明 G 的每个 k-VCC Gi包含在 G 中的至少一个子图中。我们用反证法证明这一点。我们假设Gi不包含在 G 的任何子图中。考虑在我们从 Gi 中删除 S 中的顶点及其相邻边之后 G 的计算， Gi中剩余的顶点包含在 G 中的至少两个图中。

，

这表明 S 是Gi 的顶点切割。自|S| < k， Gi不可能是 k-VCC,这与Gi是 k-VCC 矛盾。因此,我们证明我们不会丢失任何 k-VCC。从引理 1 可知,KVCC-ENUM 返回的每个子图都是 k 连通的。因此,KVCC-ENUM 将返回所有 k-连通的最大子图。换句话说,所有的 k-VCC 都将由 KVCC-ENUM 返回。

□

引理3.（无冗余)不存在 KVCC-ENUM 返回的两个子图Gi和Gj使得Gi ⊆ Gj 。

证明。我们用反证法证明。假设有两个子图Gi和Gj由 KVCC-ENUM 返回,使得Gi ⊆

吉。一方面,我们有 |V (Gi) ∩ V (Gj)| = |V (Gi)| ⩾ k。
另一方面,必须存在一个分区 G = { GG 2, . . .} G 的某个切割 S 使得Gi和Gj包含在 G 中的两个不同的图中。
从划分过程中,我们知道Gi和Gj至多有 k-1 个公共顶点。这与 |V (Gi) ∩ V (Gj)|矛盾⩾ k。因此,引理成立。 □

定理4. KVCC-ENUM 正确计算 G 的所有 k-VCC。

证明。从引理 1,我们知道 KVCC-ENUM 返回的所有子图都是 k-连通的。从引理 2 中,我们知道所有 k-VCC 都由 KVCC-ENUM 返回。由引理 3 可知,KVCC-ENUM 返回的所有 k-连通子图都是最大的,不会产生冗余子图。因此,KVCC-ENUM(算法 1)正确计算了 G 的所有 k-VCC。

□

接下来,我们将展示如何按照算法 1 中的框架高效地计算所有 k-VCC。从算法 1 中,我们知道提高算法效率的关键是高效地计算图 G 的顶点切割。下面,我们首先介绍第 4 节中的基本算法在多项式时间内计算图的顶点切割,然后我们探索优化策略以加速第 5 节中顶点切割的计算。

4. 基本解决方案在上一节中,我们提出了一个

名为 KVCC-ENUM 的基于切割的框架来计算所有 k-VCC。算法 1 中的一个关键步骤是 GLOBAL-CUT。在给出 GLOBAL-CUT 的详细实现之前,我们讨论了寻找边切割的技术,这与顶点切割高度相关。在这里,边割是一组边,删除这些边会使图断开连接。我们将证明这些方法不能直接用于查找

顶点切割。

最大流量。找到边缘切割的基本解决方案是最大流算法。对于给定的最大流,我们可以很容易地根据最大流最小切割定理计算出最小边切割。但流算法只考虑每条边的容量,对顶点的容量没有任何限制,显然不适合求顶点割。

最小切边。 Stoer 和 Wagner [25] 提出了一种在无向图中寻找全局最小边割的算法。一般的想法是迭代地找到边切割并合并一对顶点。它返回 n-1 次合并操作后具有最小值的切边。给定一个上限 k,一旦找到少于 k 条边的边切割,算法就会终止。然而,该算法不适合寻找顶点切割,因为我们不知道切割中是否包含顶点。因此,我们不能简单地合并整个过程中的任意两个顶点。

4.1 寻找顶点切割

在介绍这个想法之前,我们给出一些必要的定义
实施 GLOBAL-CUT。

定义5. (最小uv切割)如果删除 S 后 u 和 v 在不相交的子集中,则顶点切割 S 是 uv 切割,如果其大小不大于其他 uv 切割,则它是最小 uv 切割。

定义6. (局部连通性)给定图 G,两个顶点 u 和 v 的局部连通性用 κ(u, v, G) 表示,定义为最小 uv 切割的大小。 κ(u, v, G) = +∞ 如果不存在这样的切割。

算法 2 GLOBAL-CUT(G, k)

输入:一个图G和一个整数k;
输出:少于k个顶点的顶点切割;

1:计算G的稀疏认证SC; 2:选择一个度数最小
的源顶点u; 3:构造SC的有向流图SC; 4:对于所有的
v ∈ V做

5: S ← LOC-CUT(u, v, SC, SC); 6:如果S
= ∅则返回S; 7:对所有va ∈
N(u)做8:对所有vb ∈ N(u)
做

9: S ← LOC-CUT(va, vb, SC, SC);如果S = ∅则
返回S; 10:11:返回∅;

12:程序LOC-CUT(u, v, G, G) 13:如果v ∈
N(u) 或 v = u,则返回∅; 14: λ←计算G中u
到v的最大流量; 15:如果λ ⩾ k则返回∅; 16:计算G中的最
小边切割; 17:返回G中对应的
顶点割;

基于定义 6,我们定义两个局部 k 连通关系如下:

· u ≡ k G v:图G中u和v的局部连通性不小于k,即κ(u, v, G) ⩾ k。

· u ≡ k G v:u和v之间的局部连通性小于k
在图 G 中,即 κ(u, v, G) < k。

我们省略了后缀 G,并使用 u ≡ k v 和 u ≡ v表示
u ≡ k G v and u ≡ v,当上下文清楚时分别为G v。一次
u ≡ k 我们说u和v是k-local连通的。显然,u ≡ k和 v ≡ k 在
你是等价的。

全局切割算法。我们按照 [12] 来实施 GLOBAL-CUT。给定图 G,我们假设 G 包含顶点切割 S,使得 |S| < k。我们考虑任意源顶点 u。只有两种情况:(i) u ∈ S 和 (ii) u ∈ S。算法 GLOBAL-CUT 的总体思路考虑了两种情况。在第一阶段,我们选择一个顶点 u 并测试 u 和 G 中所有其他顶点 v 之间的局部连通性。我们有 (a) u ∈ S 或 (b) 如果每个局部连通性不少于感谢。

在第二阶段,我们考虑 u ∈ S 的情况,并根据引理 4 测试 u 的任意两个邻居之间的局部连通性。
更多细节可以在 [12] 中找到。

引理4. 给定一个非 k 顶点连通图 G 和一个顶点 u ∈ S,其中 S 是顶点切割,|S| < k,存在 v,v ∈ N(u) 使得 v ≡ k 在。

伪代码在算法 2 中给出。此处的优化是计算第 1 行中原始图的稀疏证书。给定图 G(V, E),稀疏证书是边 E ∈ E 的子集,使得子图当且仅当 G 是 k 连通的时,G (V, E) 是 k 连通的。毫无疑问,同样的算法在稀疏图中效率更高。我们将在下一节中介绍稀疏证明的细节。

第一阶段如第 4-6 行所示。一旦找到这样的切割 S,我们将其作为结果返回。同样,第二阶段显示在第 7-10 行中。这里,程序 LOC-CUT 测试 u 和 v 之间的局部连通性,如果 u ≡ v(第 5 行和第 9 行),则返回顶点切割。要调用 LOC-CUT,我们需要将原始图 G 转换为有向流图 G。有关如何构建有向流图的详细信息介绍如下。 k

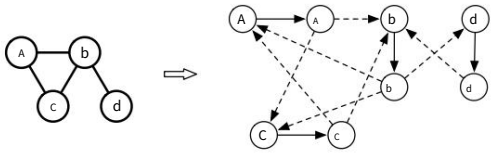


图 3:定向流图构造示例。

有向流图。图G的有向流图G是辅助有向图,用于计算两个顶点之间的局部连通性。给定图 G,我们可以按如下方式构造有向流图。 G 中的每个顶点 u 由有向流图 G 中的有向边eu表示。让 u 和 u 表示eu 的起始顶点和结束顶点。对于 G 中的每条边 (u, v),我们构造两条有向边:一条来自 u

对 v ，另一个是从 v 到 u 。因此,我们得到具有 2n 个顶点和 n + 2m 条边的 G,并且每条边的容量为 1。

示例4. 图 3 给出了定向流图构造的示例。有向流图中的实线表示原图中的顶点,有向流图中的虚线表示原图中的边。原始图包含4个顶点和4条边,有向流图包含8个顶点和12条边。

LOC -CUT程序。通过使用有向流图,我们将顶点连通性问题转化为边连通性问题。为了计算两个顶点 u 和 v 的局部连通性,我们在有向流图上执行最大流算法。

最大流的值是 u 和 v 之间的局部连通性。

LOC-CUT 的伪代码在算法 2 的第 12 行到第 17 行给出。它首先在第 13 行检查 v 是否是 u 的邻居。
如果 u ∈ N(v),我们总有 u ≡ k v 因为引理 5。

引理5. in ≡ k v 如果 (u, v) ∈ E。

然后该过程在第 14 行计算 G 中从 u 到 v 的最大流量 λ。如果 λ ⩾ k,我们有 u ≡ v 并且该过程在第 15 行返回 ∅。否则,我们在第 16 行计算 G 中的边切割。然后我们为边切割中的每条边定位原始图 G 中的相应顶点,并将它们作为 G 的顶点切割返回（第 16-17 行）。

4.2 稀疏证书

我们在本节中介绍稀疏证书[8]的细节。
在第 5 节中,我们将展示稀疏证书不仅可以用于减小图的大小,还可以用于进一步减少本地连接测试。

定义7. （证书） G 的 k 顶点连通性的证书是 E 的子集 E,这样当且仅当 G 是 k 顶点连通时,子图 (V, E) 是 k 顶点连通的。

定义8. （稀疏证书） k 顶点的证书
如果 G 有 O(k·n) 条边,则称 G 的连通性是稀疏的。

从定义中,我们可以看出稀疏证书等同于原始图 wrt k-vertex 连通性。它还可以限制边缘大小。我们根据以下定理计算稀疏证书（算法 2 的第 1 行）。

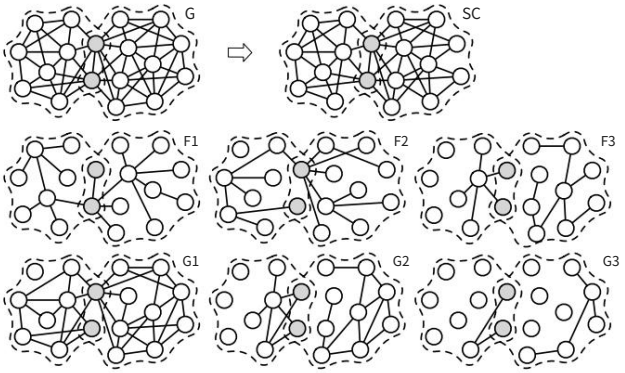


图 4:k = 3 的给定图 G 的稀疏证书

定理5. 设 G(V, E) 为无向图,n 表示顶点数。令k为正整数。对于 k,设Ei是图中Gi－1 = (V, E － (E1 ∪ E2 ∪ ... ∪ Ei－1) 中扫描优先搜索森林 i = 1, 2, ..., Fi的边集)).那么有 k × (n － 1) E1 ∪ E2 ∪ ... ∪ Ek是 G 的 k 个顶点连通性的证明,这个证明最多条边 [8]。

基于定理 5,我们可以简单地使用扫描优先搜索 k 次生成 G 的稀疏证书,每次创建一个扫描优先搜索森林Fi。下面,我们介绍如何进行扫描优先搜索。

扫描第一搜索。在给定图 G 的扫描优先搜索中,对于每个连接的组件,我们通过标记其所有邻居来扫描根顶点。我们每次扫描一个任意标记但未扫描的顶点并标记其所有未访问的邻居。

执行此步骤,直到扫描完所有顶点。生成的搜索林形成 G 的扫描优先森林。显然,呼吸优先搜索是扫描优先搜索的特例。

示例5. 图 4 显示了图 G 的稀疏证书的构造。令 k = 3。对于 i ∈ {1, 2, 3}， Fi表示从Gi-1获得的扫描优先搜索林。 Gi是通过从Gi－1 中去除Fi中的边得到的。 G0是输入图G。得到的稀疏证书SC显示在G的右侧,SC = F1 ∪ F2 ∪ F3。所有移除的边都显示在G3 中。

4.3 算法分析本节分析基本算法。在有向流图

中,所有边的容量都等于 1,并且每个顶点要么有一条边从它发出,要么有一条边进入它。对于这种图,计算最大流量的时间复杂度为 O(n· 1/2m) [14]。请注意,我们不需要在算法中计算准确的流量值。一旦流值达到 k,我们就知道任意两个给定顶点之间的局部连通性至少为 k,我们可以终止最大流算法。流计算的时间复杂度是1/2 O(min(n 个网络,我们可以执行深度优先搜索来找到切割。它花费 O(m + n) 时间。结果,我们有以下引理:

, k)· m)。给定一个流量值和对应的resid

引理6. LOC-CUT 的时间复杂度为 O(min 1/2 (n , k) · m)。

接下来我们讨论一下GLOBAL-CUT的时间复杂度。稀疏证书和有向流图的构建都需要 O(m+n) CPU 时间。令 δ 表示输入图中的最小度数。我们可以很容易地得到以下引理。

引理7. GLOBAL-CUT 调用 LOC-CUT
O(n+d) 次在最坏的情况下。

接下来我们讨论整个算法KVCC-ENUM的CPU时间复杂度。 KVCC-ENUM 迭代地删除第 2 行中度数小于 k 的顶点。这花费 O(m + n) 时间。可以通过采用深度优先搜索（第 3 行)来识别所有连接的组件。这也需要 O(m + n) 时间。为了研究调用 GLOBAL-CUT 所花费的总时间复杂度,我们首先给出以下引理。

引理8. 对于重叠分区创建的每个子图,分区后最多增加 k-1 个顶点和(k-1)(k-2)条边。

证明。顶点切割S包含不超过k -1 个顶点,并且只有这些顶点存在于重叠部分。因此,在(k－1)(k－2)处,入射边被复制。大多数2

□

引理9. 给定一个图 G 和一个整数 k,对于算法 1 中重叠划分得到的每个连通分量 C，|V (C)| ⩾ k + 1。

证明。令 S 表示重叠分区中的顶点切割。
C是在这个分区中获得的连接组件之一。
设H表示V(C)中所有顶点的顶点集,但不在S中,即H={u|u∈V(C),u∈S}。我们有 H = ∅。请注意,图中的每个顶点在 G 中的度数至少为 k（算法 1 中的第 5 行）。 H 中的每个顶点 u 至少存在 k 个邻居,因此根据引理 5,对于 u 的每个邻居 v,我们有 v ∈ C。因此,我们有 |V (C)|。 ⩾ k + 1。

□

引理10. 给定图 G 和整数 k,算法 KVCC-ENUM 期间重叠分区的总数不大于 。

证明。假设 λ 是整个算法 KVCC-ENUM 中重叠分区的总数。这至少会生成 λ + 1 个连通分量。我们从引理 9 得知,每个连通分量至少包含 k + 1 个顶点。因此,我们总共至少有 (λ + 1)(k + 1) 个顶点。

另一方面,根据引理 8,我们在通过重叠分区获得的每个子图中最多增加 k-1 个顶点。
因此,最多添加 λ(k -1) 个顶点。我们得到以下公式。

(λ + 1)(k + 1) ⩽ n + λ(k − 1)

重新排列公式,我们有 λ ⩽ □

接下来,我们证明 k-VCC 数量的上限。

定理6. 给定一个图 G 和一个整数 k,有最多 k-VCC,即|V CCk(G)| <

证明。与引理10的证明类似,令λ为整个算法KVCC-ENUM中重叠分区的次数。最多增加 λ(k − 1) 个顶点。令 σ 为在所有分区中获得的连接组件的数量。我们有 σ > λ。

根据引理 9,每个连通分量至少包含 k + 1 个顶点。请注意,每个连通分量要么是 k-VCC,要么是不包含任何 k-VCC 的图。否则,连通分量将被进一步分割。令 x 为 k-VCC 的数量,y 为不包含任何 k-VCC 的连通分量的数量,即 x + y = σ。我们知道一个 k-VCC 至少包含 k + 1 个顶点。因此,在完成所有分区后,至少有 x(k + 1) + y(k + 1) 个顶点。我们有以下公式。

x(k + 1) + y(k + 1) ⩽ n + λ(k − 1)

由于 λ < σ 且 σ = x+y,我们重新整理公式如下。

x(k + 1) + y(k + 1) < n + x(k − 1) + y(k − 1) x(k + 1) < n + x(k − 1)

因此,我们有 x < □

定理7. KVCC-ENUM 的总时间复杂度是1/2 O(min(n , k) · m · (n + d) · 名词)。

证明。 KVCC-ENUM 的总时间复杂度取决于调用 GLOBAL-CUT 的次数。假设在整个KVCC-ENUM算法中GLOBAL-CUT被调用了p次,整个KVCC-ENUM算法中重叠分区的个数为p1 ,k-VCC的总数为p2。很容易看出 p = p1 + p2。从引理 10，我们知道p1 ⩽ 从定理 6,我们知道 < p2 < 2 。因此,我们有 p = p1 + p2 < n。根据引理 6 和引理 7,1 /2 KVCC-ENUM 的总时间复杂度为 O(min(n

, k) · m · (n + d) · 名词)。 □

讨论。定理 7 表明所有 k-VCC 都可以在多项式时间内枚举。虽然时间复杂度仍然很高,但在实践中表现要好得多。请注意,时间复杂度是三部分的乘积： 1/2 · 第一部分 O(min(n , k) · m) 是LOC-CUT测试是否存在尺寸小于k的顶点切割的时间复杂度。在实践中,要测试的图比原始图 G 小得多,因为 (1) 使用 k-core 技术和稀疏证明技术对要测试的图进行了修剪。 (2) 由于图划分方案,输入图被划分成许多更小的图。

· 第二部分 O(n + δ) 是算法 GLOBAL-CUT 调用 LOC-CUT（本地连接测试)的次数。我们将在第 5 节中讨论如何显着减少本地连接测试的数量。 · 第三部分 O(n) 是调用 GLOBAL-CUT 的次数。在实践中,由于 k-VCC 的数量通常比 k-VCC 的数量小得多,因此可以显着减少数量。在下一节中,我们将探索几种搜索减少技术来加速算法。

5.搜索减少

在上一节中,我们介绍了我们的基本算法。回想一下,在最坏的情况下,我们需要使用 GLOBAL-CUT 中的 LOC-CUT 测试源顶点 u 和 G 中所有其他顶点之间的局部连通性,我们还需要测试 u 的每对邻居的局部连通性。对于每一对顶点,我们需要计算有向流图中的最大流。因此,改进算法的关键是减少本地连通性测试（LOC-CUT)的数量。在本节中,我们提出了几种技术来避免不必要的测试。如果我们可以保证,我们可以避免测试顶点对 (u, v) 的局部连通性

u ≡ v. 我们称这样的操作为扫描操作。下面,我们分别在 5.1 节和 5.2 节中介绍两种有效修剪不必要测试的方法,即邻居扫描和组扫描。

5.1 邻居扫描

在本节中,我们提出了一种邻居扫描策略,以在第一阶段修剪不必要的本地连接测试（LOC-CUT）

全球切割。一般来说,给定一个源顶点u,对于任何顶点v,我们的目标是根据v的邻居信息跳过（u,v)的局部连通性测试。下面,我们探索两种邻居扫描策略,即邻居扫描使用顶点沉积使用侧顶点和邻居扫描。

5.1.1 Neighbor Sweep using Side-Vertex我们首先定义side-vertex如下。

定义9. (SIDE-VERTEX)给定图 G 和整数 k,如果不存在满足 |S| 的顶点切割 S,则顶点 u 称为边顶点< k 且 u ∈ S。

基于定义9,我们给出以下引理来证明
关于局部 k 连通性关系的传递性 ≡ ^{k}.

引理11. 给定图 G 和整数 k,如果 b 是边顶点,则假设 a ≡ k b 和 b ≡ c。 ^{k}
c, 我们有一个 ≡ ^{k}

证明。我们用反证法证明。假设 b 是边k顶点且 a ≡ c。在 a 和 c 之间存在具有 k − 1 或更少顶点的顶点切割。 b 不在任何这样的 cut 中,因为它是 c 边。这与

我们有 b ≡ a 或 b ≡ k前提条件 a ≡ b 和 b ≡ ^{k} ^{k}顶点。然后
^{k} c。 □

在引理 11 中使用局部连通关系的传递属性的明智方法可以大大减少不必要测试的数量。考虑在算法 GLOBAL-CUT 中选择的源顶点 u。我们假设 LOC-CUT（第 5 行)为顶点 v 返回 ∅,即 u ≡ v。我们从引理 11 知道,如果 (i) w,则可以跳过顶点对 (u, w) 进行局部连通性测试(ii) v 是一个边顶点。对于条件（i) ,我们可以根据引理5使用一个简单的必要条件,即对于任何w如果（v,w)∈E。在下面,我们将顶点v和w,v ≡集中在条件（ii) 并寻找必要的条件来有效地检查一个顶点是否是一个边顶点。
^{k}
至v ≡
^{k}

侧顶点检测。为了检查一个顶点是否是边顶点,我们可以很容易地根据定义9得到以下引理。

引理12. 给定图 G,顶点 u 是边顶点当且仅当∀v, v ∈ N(u), v ≡
^{k} 在。

回想一下,如果两个顶点彼此相邻,则它们是 k-local 连接的。对于非连通顶点的 k-local 连通性,我们在下面给出另一个必要条件。

引理13. 给定两个顶点 u 和 v,u ≡ ^{k} v 如果 |N(u) ∩ N(v)| ⩾ k。

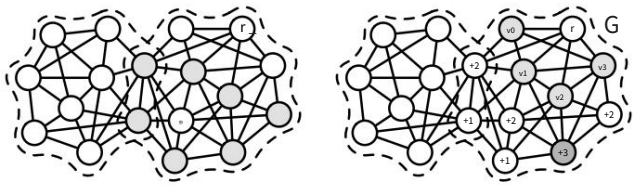
证明。 u 和 v 在删除任何 k − 1 个顶点后不能不相交,因为它们至少有 k 个共同的邻居。因此 u 和 v 必须是 k-local 连接的。
□

结合引理 12 和引理 13,我们得出以下
检查顶点是否为边顶点的必要条件。

定理8. 顶点 u 是边顶点如果∀v, v ∈ N(u), ei
然后 (v, v) ∈ E 或 |N(v) ∩ N(v)| ⩾ k。

证明。使用引理 5、引理 12 和引理 13 可以很容易地验证该定理。
□

DEFINITION 10. (STRONG SIDE-VERTEX)顶点 u 被称为
如果它满足定理 8 中的条件,则它是一个强边顶点。



(a) 强边顶点 s (b) 顶点沉积

图 5:当 k = 3 时,强侧顶点和顶点沉积

使用强边顶点,我们可以定义我们的第一个邻居扫描规则如下。

（邻居扫描规则 1)给定一个图 G 和一个整数 k,设 u 是算法 GLOBAL-CUT 中选定的源顶点,v 是图中的强边顶点。如果我们有 u ≡ ,我们可以跳过所有 (u, w) 对的本地连通性测试
^{k} v 和 w ∈ N(v)。

下面我们举例说明邻居扫描规则 1。

示例6. 图 5 (a) 显示了图 G 中的强边顶点 s,而参数 k = 3。假设 r 是源顶点。 s 的任意两个邻居要么由一条边相连,要么至少有 3 个公共邻居。如果首先测试 rs 之间的局部连通性,我们可以安全地扫描 s 的所有邻居,这些邻居在图 5 (a) 中用灰色标记。

和 s 和 r ≡ ^{k}

下面,我们将讨论在整个算法中如何在图被分割的同时有效地检测强边顶点并保持强边顶点。

强侧顶点计算。根据定理 8,我们可以提前计算所有强边顶点 v,并在 v 与源顶点 k 连接后跳过 v 的所有邻居（GLOBAL-CUT 中的第 5 行）。我们可以推导出以下引理。

引理14. 计算图 G 中所有强边顶点的时间复杂度为 O($\sum_{w\in V(G)} d(w)^{2}$).

证明。为了计算图 G 中的所有强边顶点,我们首先检查每个顶点 u 的所有 2 跳邻居 v。由于 v 和 u 共享 1-hop 邻居的公共顶点,我们可以轻松获得与 u 有 k 个公共邻居的所有顶点。任何顶点 w 都被认为是其他顶点 ud(w) 次的一跳邻居。

我们使用 d(w) 步来获得 u 的 2 跳邻居,它们与 u 共享一个公共顶点 w。这个阶段花费 O( ) 时间。 $\sum_{w\in V(G)} d(w)^{2}$

现在对于每个给定的顶点 u,我们有所有顶点 v 与其共享 k 个公共邻居。对于每个顶点 w,我们检查 w 的任意两个邻居是否有 k 个共同邻居。这个阶段也花费 O( ) 时间。因此,总时间
$\sum_{w\in V(G)} d(w)^{2}$
复杂度为 O($\sum_{w\in V(G)} d(w)^{2}$). □

在计算完原始图 G 的所有强边顶点后,我们不需要从头开始重新计算分区图中所有顶点的强边顶点。相反,我们可以找到可能的方法来减少强侧顶点检查的数量,方法是利用 G 中已经计算出的强侧顶点。我们可以基于引理 15 和引理 16 来做到这一点

分别有效地检测非强侧顶点和强侧顶点。

引理15. 设 G 是一个图， Gi是通过算法 1 中的 OVERLAP-PARTITION 对 G 进行分区得到的图之一,如果一个顶点是 Gi 中的强边顶点,则该顶点是 G 中的强边顶点。

证明。强侧顶点 u 需要 u 的任意两个邻居之间至少有 k 个共同邻居。引理是显而易见的,因为 G 包含Gi 中的所有边和顶点。

□

从引理 15,我们知道如果一个顶点不是 G 中的强边顶点,那么它不是Gi中的强边顶点。这个属性允许我们检查 Gi 中有限数量的顶点,即强边顶点集在 G.

引理16. 令 G 是一个图， Gi是通过算法 1 中的 OVERLAP-PARTITION 对 G 进行分区得到的图之一,S 是 G 的顶点切割,对于任何顶点 v ∈ V (Gi),如果 v是 G 中的强边顶点且 N(v) ∩ S = ∅,则 v 也是Gi中的强边顶点。

证明。顶点 v 的强边顶点的限定需要关于 v 的两跳邻居的信息。在划分图时 S 中的顶点是重复的。给定G中的一个强边顶点v,如果N(v)∩S=∅,v的两跳邻居不受分区操作的影响,因此N(v)中顶点之间的关系不受分区操作的影响分区操作。因此,根据定义 10， v 仍然是Gi中的强边顶点。

□

由引理15和引理16,在图G被顶点切割S划分的图Gi中,我们可以将强边顶点检查的范围从整个图Gi中的顶点缩小到同时满足以下两个条件的顶点u：

· u 是G 中的强边顶点;和 · N(u) ∩ S = ∅。

5.1.2 使用 Vertex Deposit 的 Neighbor Sweep

顶点沉积。强边顶点策略在很大程度上依赖于强边顶点的数量。接下来,我们研究一种称为顶点沉积的新策略,以根据邻居信息进一步扫描顶点。我们首先给出以下引理：

引理17. 给定图 G 中的源顶点 u,对于任何顶点 v,如果存在 k 个顶点w1, v ∈ V (G),我们有 u ≡ k使得 ^{k} w2, ...。 . . , wi和wi ∈ N(v) 对于任何 1 ⩽ i 周 u ≡ ⩽ k。

证明。我们用反证法证明。假设你 ≡ ^{k} 在。在 u 和 v 之间存在一个具有 k − 1 个或更少顶点的顶点切割 S。对于任何wi(1 ⩽ i ⩽ k),我们有wi ≡ v 因为wi ∈ N(v) (引理 5) 并且我们也有wi ≡ v， wi不能同时满足wi因为你 除非wi^{k}∈ S。因此,我们得到一个至少有 k 个顶点w1你和我 的割 S。^{k} . .,周。这与 |S| 相矛盾< k。

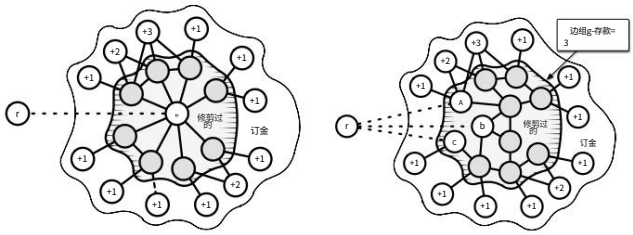
□

根据引理 17,给定一个源顶点 u,一旦我们找到一个至少有 k 个邻居wi且 u ≡ wi 的顶点 v,我们可以在不测试 (u, v) 的局部连通性的情况下获得 v。为了有效地检 u ≡ ^{k} 测这样的顶点 v,我们定义顶点 v 的沉积如下。

定义11.（顶点沉积)给定一个源顶点 u,每个顶点 v 的沉积,用 deposit(v) 表示,是 v 的邻居 w 的数量,使得 w 和 u 的局部连通性已经用 w ≡ 计算出来

^{k} 在。

根据定义 11,假设 u 是源顶点,对于每个顶点 v,deposit(v) 是一个动态值,取决于已处理的顶点对的数量。为了维持顶点存款,我们将存款初始化为 0,一旦我们知道某个顶点 w 的 w ≡ u,我们可以将每个顶点 v ∈ N(w) 的存款 deposit(v) 增加 1。我们可以得到遵循根据引理 17 的定理。



(a) 2-hop 存款 (b) 群扫存

图 6:增加与邻居和组扫描的存款

定理9. 给定源顶点 u,对于任何顶点 v,我们k有 u ≡ v 如果存款 (v) ⩾ k。

基于定理 9,我们可以推导出邻居扫描的第二条规则如下。

（邻居扫描规则 2)给定一个选定的源顶点 u,如果 deposit(v) ⩾ k,我们可以跳过对 (u, v) 的本地连通性测试。

我们在下面展示一个例子。

示例7. 图 5 (b) 给出了我们的顶点沉积策略的示例。给定图 G 和参数 k = 3,设顶点 r 为选定的源顶点。我们假设v0、 v1、 v2和v3是被测试的顶点。所有这些顶点都是与顶点 r 局部 k 连通的,即 r ≡ vi, i ∈ 0, 1, 2, 3,因为 v0、 v1、 v2和v3是 r 的邻居。我们为每个测试顶点的邻居存入一次。图中给出了所有受影响顶点的存款值。我们用深灰色标记存款不少于 3 的顶点。可以跳过 r 和这样一个顶点之间的本地连通性测试。

k

为了增加顶点 v 的存款,我们只需要 v 的任何邻居与源顶点 u 是局部 k-连接的。在处理强边顶点时,我们也可以使用顶点存放策略。

给定源顶点 u 和强边顶点 v,如果 u ≡ v 根据边顶点策略,我们将扫描所有 w ∈ N(v)。接下来我们增加每个^{k}非扫描顶点 w ∈ N(w) 的存款。

换句话说,对于一个强边顶点,我们可以通过结合两种邻居扫描策略来扫描它的 2 跳邻居。

下面给出了一个例子。

示例8. 图 6 (a) 显示了强边顶点 s 的过程。给定源顶点 r,假设 s 是强边顶点k 且 r ≡ s。 s 的所有邻居都被清除,s 的所有 2 跳邻居相应地增加他们的存款。每个顶点的存款增加值取决于扫过的连接顶点的数量。

5.2 群扫

邻居扫描策略只能通过使用邻居信息在 GLOBAL-CUT 的第一阶段修剪不必要的本地连接测试。在本小节中,我们介绍了一种新的修剪策略,即组扫描,它可以批量修剪不需要的本地连接测试。在组扫描中,我们不将跳过的顶点限制为某些顶点的邻居。更具体地说,我们的目标是将顶点划分为顶点组,并在满足特定条件时扫描整个组。此外,我们的组扫描策略也可用于减少 GLOBAL-CUT 两个阶段中不必要的本地连接测试。

首先,我们定义一个关于顶点 u 和一组顶点C如下。

u ≡ k C:对于所有顶点 v ∈ C,u ≡ k 在。

给定源顶点 u 和边顶点 v,我们假设 u ≡ v。根据引理 11 中的传递关系,我们可 k 以跳过对所有 w 的顶点 u 和 w 的测试,其中 w ≡ v。在我们的邻居扫描中策略,我们 k 选择 v 的所有邻居作为顶点 w,即 u ≡

k N(v)。为了每次扫描更多的顶点,我们定义了侧群。

DEFINITION 12. (SIDE-GROUP)给定一个图 G 和一个 in
teger k, G 中的一个顶点集 CC 是一个边群如果∀u, v ∈ C, u ≡ k 在。

请注意,边组可能包含某个顶点切割 S 中的顶点,其中 |S| < k。接下来,我们介绍如何构造图 G 中的边组,然后讨论我们的组清除规则。

边组构建。 4.2 节介绍了稀疏证书来限制图的大小。令Fi和Gi为定理 5 中定义的符号。假设 G 不是 k-连通的并且存在顶点切割 S 使得 |S| < k。根据[8],我们有以下引理。

引理18. Fk不包含一个简单的树路径Pk ,其
两个端点在 G − S 的不同连通分量中。

根据引理18,我们可以得到如下定理。

定理10. 设 CC 表示Fk中任意连通分量的顶点集。 CC是一个侧基。

证明。假设你 ≡ k v 在 CC 中。从 u 到 v 的所有简单路径都将穿过顶点切割 S。这与引理 18 相矛盾。 □

示例9. 查看图 4 中稀疏证书的构造。给定 k = 3,在F3中获得两个具有多个顶点的连通分量。两个连通分量的顶点数分别为6和9。它们中的每一个都是一个边组,并且同一连通分量中的任何两个顶点都是局部 3-连通的。请注意,具有 6 个顶点的连通分量在顶点切割中包含两个顶点,以灰色标记。

我们将所有侧基表示为 CS = {CC1, CC2, . . . , CCt}。
根据定理 10,CS 可以很容易地计算为稀疏证书的副产品。有了CS,根据引理11中的传递关系,我们可以很容易地得到如下剪枝规则。

(Group Sweep Rule 1)令 u 为算法 GLOBAL-CUT 中的源顶点,给定边组 CC,如果存在强边顶点 v ∈ CC 使得 u ≡ v,我们可以跳过局部连通性测试所有 w ∈ CC − {v} 的顶点对 (u, w)。 k

上述组扫描规则依赖于在某个侧组中成功检测到强侧顶点。在下文中,我们进一步介绍了一种基于存款的方案来处理某个边组中不存在强边顶点的情况。

团体存款。与顶点存放策略类似,组存放策略旨在存放组级别的值。为了展示我们的团体存款方案,我们首先介绍以下引理。

引理19. 给定源顶点 u、整数 k 和边 v}| ⩾ k。
CC组,我们有你 ≡ k CC 如果 |{v|v ∈ C, u ≡ k

证明。我们用反证法证明。假设存在w。顶点切割 S 存在于
k CC 中的一个顶点 w 使得 u ≡
|S| < k。令v0, v1, ..., vk−1为 CC 中满足vi, 0 ⩽ i ⩽ k − 1 的k 个顶点。根据边组的 u ≡ k 定义,我们有 w ≡ vi 。每个vi必须属于 S,k因为 u ≡ w。因此,S 的大小至少为 k。这与 |S| 相矛盾< k。 k

□

基于引理 19,给定一个源顶点 u,一旦我们找到一个具有至少 k 个顶点 v 且 u ≡ v 的边组 CC,我们就可以得到 CC,而无需测试从 u 到 CC 中其他顶点的局部连通性。 u ≡ k 为了有效地检测这种侧基 CC,我们定义侧基 CC 的组沉积如下。

定义13.（组存款)每个侧组 CC 的组存款,由 g-deposit(CC) 表示,是顶点的数量 v ∈ CC 使得 v 和 u 的局部连通性已经用 v ≡ 计算

k 在。

根据定义 13,假设 u 是源顶点,对于每个边组 CC ∈ CS,g-deposit(CC) 是一个动态值,取决于已处理的顶点对。为了维持每个侧群 CC 的组存款,我们将 CC 的组存款初始化为 0。一旦某个顶点 v ∈ CC 的 v ≡ u,我们可以将 g-deposit(CC) 增加 1。我们得到遵循引理 19 的定理。
k

定理11. 给定源顶点 u,对于任何边组 CC ∈ CC 如果 g-deposit(CC) ⩾ k。
CS,我们有你 ≡ k

基于定理 11,我们可以推导出我们的第二个组扫描规则如下。

（组扫描规则 2)给定一个选定的源顶点 u,如果 g-deposit(CC) ⩾ k,我们可以跳过 u 和 CC 中的顶点之间的局部连接测试。

请注意,组扫描操作可以进一步触发邻居扫描操作,反之亦然,因为这两种操作都会产生新的局部 k-连通顶点对。我们在下面展示一个例子。

例10. 图 6 (b) 给出了一个组扫描的例子。
假设 k = 3,灰色区域是检测到的侧群。给定一个源顶点r,假设a、b、c分别是k b 和r ≡ c 的被测顶点。根据定理 11,我们可以安全地扫过同一个边群中的所有顶点。此外,我 r ≡ k a, r ≡ k 们对边组之外的邻居应用顶点存放策略。增加的存款价值显示在每个顶点上。

接下来,我们将展示侧组也可用于修剪 GLOBAL-CUT 第二阶段的本地连接测试。

回想一下,在 GLOBAL-CUT 的第二阶段,给定源顶点 u,我们需要测试u 的邻居的每对 (va, vb)的局部连通性。有了边群,我们可以很容易地得到下面的群扫规则。

(Group Sweep Rule 3)设u为源顶点， va和vb为u的两个邻居。如果va和vb属于同一个侧群,我们有va ≡ vb ,因此我们不需要在 GLOBAL-CUT 的第二阶段测试 (va, vb) 的局部连通性。

邻居扫描和组扫描技术的详细实现在下一节中给出。

5.3 总体算法

在本节中,我们结合我们的剪枝策略并给出优化算法 GLOBAL-CUT* 的实现。伪代码在算法 3 中给出。我们可以用 KVCC-ENUM 中的 GLOBAL-CUT* 替换 GLOBAL-CUT,以获得我们计算所有 k-VCC 的最终算法。

GLOBAL-CUT*算法仍然遵循 GLOBAL-CUT 的相似思想,即考虑源顶点 u,然后根据 u 是否属于顶点切割 S 分两阶段计算顶点切割。给定源顶点 u,阶段 1 （第 8-15 行)考虑 u /∈ S 的情况。第 2 阶段（第 16-21 行)考虑以下情况

u ∈ S。如果在两个阶段都没有找到顶点切割 S,则不存在这样的切割,我们只需在第 22 行返回 ∅。

我们在计算稀疏证书时计算边组 CS（第 1 行）。请注意,这里我们只考虑大小大于 k 的边组,因为根据定理 11,只有在组中至少有 k 个顶点被扫过时,该组才能被扫过。

然后我们根据定理 8（第 3 行)计算所有强边顶点 SV。这里,强侧顶点是根据第 5.1.1 节中讨论的方法计算的。如果 SV 不为空,我们可以选择一个内部顶点作为源顶点 u 并且不需要考虑阶段 2,因为 u 不能在任何带有 |S| 的切割 S 中。 < k 在这种情况下。否则,我们仍然选择度数最小的源顶点 u（第 4-7 行）。

在阶段 1（第 8-15 行)中,我们将每个侧组的组存款初始化为 0（第 8 行）,这是侧组中扫描顶点的数量。此外,我们将每个顶点的本地存款初始化为 0,并将每个顶点的 pru 初始化为 false（第 9 行）。这里用pru来标记一个顶点是否可以扫过。由于源顶点 u 与自身是局部 k 连通的,我们首先通过调用 SWEEP 过程（第 10 行)在源顶点上应用扫描规则。直观上,靠近源顶点 u 的顶点往往与 u 处于相同的 k-VCC 中。换句话说,一个远离u的顶点v往往会被顶点切割S与u分开。因此,我们按照dist(u, v, G)的非升序处理G中的顶点v (第 11 行）。我们的目标是通过处理尽可能少的顶点来找到顶点切割。对于第 1 阶段要处理的每个顶点 v,如果 pru(v) 为真,我们将跳过它（第 12 行）。否则,我们使用 LOC-CUT（第 13 行)测试 u 和 v 的本地连通性。如果存在尺寸小于 k 的切割 S,我们只需返回 S（第 15 行）。

否则,我们调用 SWEEP 过程使用第 5 节中介绍的扫描规则来扫描顶点。我们将在后面详细介绍 SWEEP 过程。

在阶段 2（第 16-21 行)中,我们首先检查源顶点 u 是否是强边顶点。如果是这样,我们可以跳过阶段 2,因为强边顶点不包含在任何大小小于 k 的顶点切割中。否则,我们对 N(u) 中的所有顶点执行成对局部连通性测试。在这里,我们应用组扫描规则 3 并跳过测试同一侧组中的那些顶点对（第 19 行）。

程序扫描。 SWEEP 过程如算法 4 所示。要扫描顶点 v,我们将 pru(v) 设置为真。此操作可能会导致其他顶点的邻居扫描和组扫描,如下所示。

· (Neighbor Sweep) 在第1-5 行,我们考虑了neighbor sweep。对于v的所有没有被扫过的邻居w,我们首先根据定义11将deposit(w)增加1。然后我们考虑两种情况。第一种情况是 v 是强边顶点。根据 5.1.1 节中的邻居清除规则 1,w 可以被清除,因为 w 是 v 的邻居。第二种情况是 deposit(w) > k。根据5.1.2节邻居扫描规则2,可以对w进行扫描。在这两种情况下,我们都调用 SWEEP 来递归地扫描 w。 （第 4-5 行） · （组扫描)在第 6-11 行中,如果 v 包含在侧组CCi 中,我们考虑组扫描。我们首先根据定义 13 将 g-deposit(CCi) 增加 1。然后我们考虑两种情况。第一种情况是 v 是一个强边顶点。根据 5.2 节中的组扫描规则 1,我们可以扫描CCi 中的所有顶点。

第二种情况是 g-deposit ⩾ k。根据 5.2 节中的组扫描规则 2,我们可以扫描CCi 中的所有顶点。
在这两种情况下,我们递归调用 SWEEP 来扫描 CCi 中每个未扫描的顶点（第 8-11 行）。

算法 3 GLOBAL-CUT* (G, k)

输入:一个图G和一个整数k;
输出:尺寸小于k的顶点切割;

1:计算 G 的稀疏认证 SC 并将所有边组收集为 CS = {CC1, ..., CCt};
2:构造SC的有向流图SC;
3: SV←计算SC中的所有强边顶点; 4:如果SV = ∅那么5:选择一个度数最小的顶点 u; 6: else 7:从SV中随机选择一个顶点u;

8:对于CS中的所有CCi :g-deposit(CCi) ←0; 9: 对于 V 中的所有v:deposit(v) ← 0,pru(v) ← false;
10: SWEEP(u, pru, deposit, g-deposit, CS); 11:对所有v ∈ V 在 dist(u, v, G) 的非升序中做12: if pru(v) = true then continue;
13: S ← LOC-CUT(u, v, SC, SC); 14:如果S = ∅则返回S;
15: SWEEP(v, pru, deposit, g-deposit, CS);

16:如果u 不是强边顶点,则17:对于所有va ∈ N(u)做18:19:20:21:
对于所有vb ∈ N(u)如果va和vb在同一个 CCi 中则继续;
S ← LOC-CUT(u, v, SC, SC);如果S = ∅则返回S;

22:返回∅;

算法 4 SWEEP(v, pru, deposit, g-deposit, CS) 1: pru(v) ← true;

2:对于所有w ∈ N(v) st pru(w) = false do 3: deposit(w)++; 4:如果v是强边顶点或deposit(w) ⩾ k ,则5: SWEEP(w, pru, deposit, g-deposit, CS); 6:如果 v 包含在 CCi 中并且 CCi尚未被处理,则7: g-deposit(CCi) ++; 8:如果v 是强边顶点或g-deposit(CCi) ⩾ k ,则9:将 CCi 标记为已处理; 10:对于所有w ∈ CCi st pru(w) = false做11:

SWEEP(w, pru, deposit, g-deposit, CS)

6. 实验

在本节中,我们通过实验评估我们提出的算法的性能。

所有算法都是在 C++ 中使用 gcc 编译器在 -O3 优化级别实现的。所有实验都是在 Linux 操作系统下进行的,该操作系统运行在配备 Intel Xeon 3.4GHz CPU、32GB 1866MHz DDR3-RAM 的机器上。算法的时间成本是用程序执行期间流逝的挂钟时间量来衡量的。

数据集。我们使用 7 个公开可用的真实世界网络来评估算法。网络统计如表1所示。

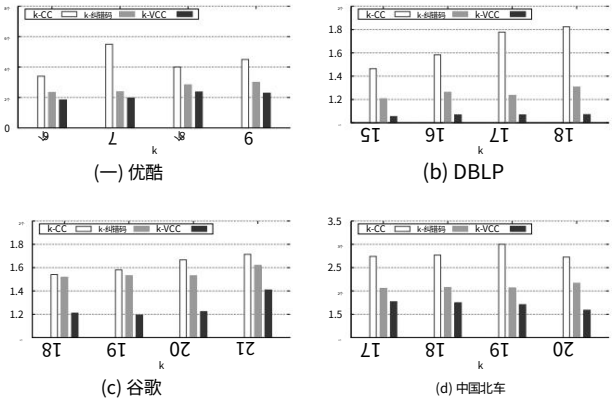
Stanford 是一个网络图,其中顶点代表斯坦福大学(stanford.edu)的页面,边代表它们之间的超链接。 DBLP 是 DBLP 的合着网络。

Cnr 是意大利 CNR 域的小型爬网。 ND 是一个网络图,其中顶点代表圣母大学(nd.edu) 的页面,边代表它们之间的超链接。 Google 是来自 Google Programming Contest 的网络图表。 Youtube 是来自视频共享网站 Youtube 的社交网络。 Cit 是由国家经济研究局维护的引文网络。所有数据集都可以从 SNAP1 下载。

" http://snap.stanford.edu/index.html

表 1:网络统计

数据集	|在|	|和|	密度最大度数 38,625 343	
斯坦福大学	281,903 2,312,497 317,080		8.20	18,236
DBLP	1,049,866 325,557 3,216,152		3.31	
心率	325,729 1,497,134 875,713		9.88	10,721
无损检测	5,105,039 3,774,768		4.60	6,332
谷歌	16,518,948		5.83	793
城市			4.38	



(一) 优酷 (b) DBLP

(c) 谷歌 (d) 中国北车

图 7:平均直径

6.1 有效性评价

我们采用以下三个质量措施进行有效性评估:

· 直径diam。直径定义如方程式所示。 1. · 边缘密度ρe。边密度是图中的边数与具有相同顶点集的完全图的边数之比。形式方程如下:

$$\rho e(g) = \frac{2|E(g)|}{|V(克)| \cdot (|V(g)| - 1)} \quad (4)$$

· 聚类系数 C。顶点 u 的局部聚类系数 c(u) 是包含 u 的三角形数量与以 u 为中心的三元组数量的比率,定义为:

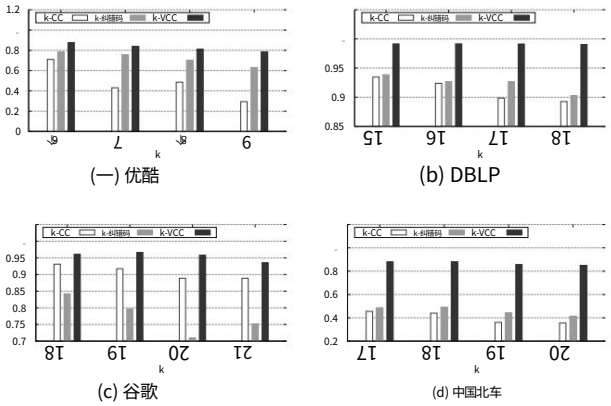
$$|\{(v, w) \in E | v \in N(u), w \in N(u)\}| \; c(u) = \frac{}{|N(u)| \cdot (|N(u)| - 1)/2} \quad (5)$$

图的聚类系数是所有顶点的平均局部聚类系数:

$$C(G) = \frac{}{|在|} \; c(u) \quad (6)$$

在我们的有效性测试中,给定一个图 G 和一个参数 k,我们分别计算 G 的每个 k-VCC 的直径、边密度和聚类系数。我们显示每个参数 k 的所有 k-VCC 的平均值。我们为 G 的所有 k-ECC 和 k-cores 计算相同的统计数据作为比较。

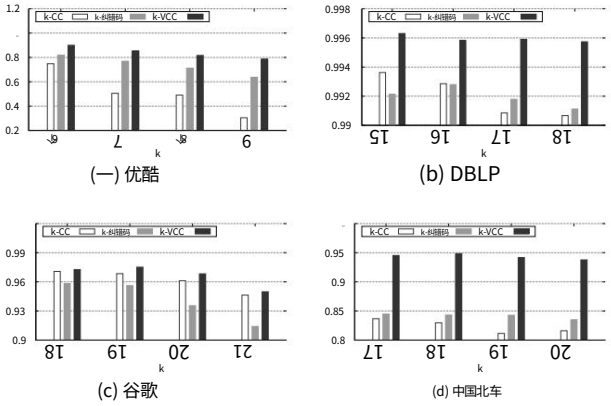
图 7 显示了真实数据集中不同参数 k 下所有 k 核、k-ECC 和 k-VCC 的平均直径。同样,图8和图9分别给出了边缘密度和聚类系数的统计。我们选择 Youtube、DBLP、Google 和 Cnr 四个数据集作为本次实验的代表。在实验结果中,我们可以看到对于相同的参数k值,k个VCC具有最小的平均直径,最大的平均边缘



(一) 优酷 (b) DBLP

(c) 谷歌 (d) 中国北车

图 8:平均边缘密度



(一) 优酷 (b) DBLP

(c) 谷歌 (d) 中国北车

图 9:平均聚类系数

密度和所有三个测试指标中最大的聚类系数。结果表明,我们的 k-VCC 比 k-ECC 和 k-cores 更具凝聚力。

值得一提的是,在图 7 中,当 k 增加时,获得的 k-cores、k-VCCs 和 k-ECCs 的直径可以增加或减少。例如,在 Youtube 数据集中,当 k 从 7 增加到 8 时,k 个 VCC 的平均直径略有下降。这是因为当 k 增加时,得到的 k-VCCs 更内聚,因此包含某个顶点的 k-VCCs 的直径变小。这种原因也导致这些数据集中某些 k 值的边缘密度和聚类系数增加。再举一个例子,在谷歌数据集中,当 k 从 20 增加到 21 时,k-VCC 的平均直径略有增加。造成这种现象的原因是存在一些小的20-VCC,其中没有顶点属于任何21-VCC。这里小的k-VCC意味着存在少量

里面的顶点。这些小型 20-VCC 的直径很小,这使得平均直径很小。这种原因也导致这些数据集中某些 k 值的边缘密度和聚类系数降低。

案例研究显示在第 6.4 小节中以进一步证明 k-VCC 的有效性。

6.2 效率评估为了测试我们提出的技术的效率,我们比较了以下四种计算 k-VCC 的算法。对于每个

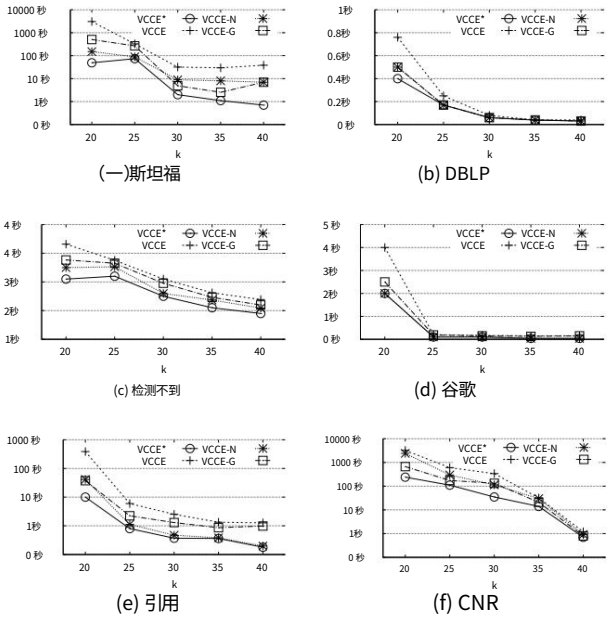


图 10:处理时间

表 2:不同规则的比例

规则	斯坦福	DBLP	ND	谷歌	Cit	Cnr
NS 1	14%	68%	32%	4%	1%	9%
国服2	40%	9%				
GS	13%					
非保诚	33%					

规则	斯坦福	DBLP	ND	谷歌	Cit	Cnr
NS 1	67%	1%	29%	12%	11%	21%
国服2	12%	48%	8%	56%	26%	8%
GS						
非保诚						

数据集,我们展示了不同参数 k 从 20 到 40 变化的算法的统计数据。

- VCCE:我们在第 4 节中介绍的基本算法。
- VCCE-N:具有邻居扫描策略的基本算法
  在第 5.1 节中介绍。
- VCCE-G:带组扫描策略的基本算法
  在第 5.2 节中介绍。
- VCCE* :具有邻居扫描和组扫描策略的算法。

测试时间成本。从图 10 可以看出,随着参数 k 的增加,各算法的时间成本总体呈下降趋势。这是因为参数 k 的值越高,k-VCC 的数量越少。直观地说,当 k 增加时,该算法将在处理过程中测试较少的本地连接性。这里的一个特例是,在斯坦福数据集中,算法 VCCE* 在 k = 25 下比在 k = 20 下花费的时间多一点。

这种现象的发生是由于斯坦福图的结构,其中 k = 25 导致比 k = 20 更多的分区。

我们还发现算法 VCCE-N 和 VCCE-G 在所有测试用例中都比基本算法更有效。考虑到不同数据集的具体结构,我们发现组扫描策略在图 Cnr 上更有效,而邻居扫描策略在其他数据集上更有效。我们的 VCCE*算法在所有测试用例中都优于所有其他算法。

测试扫描规则的有效性。为了进一步调查我们的扫描规则的有效性,我们还跟踪每个处理过的

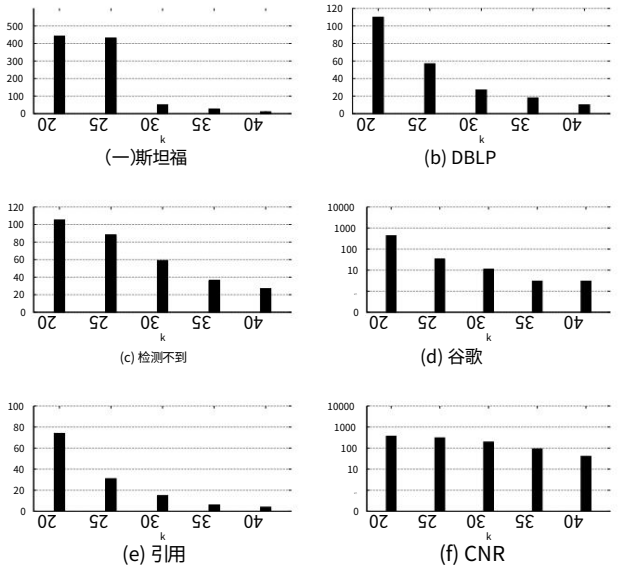


图 11:k-VCC 的数量

VCCE* 执行期间的顶点,并记录每个策略修剪的顶点数。具体来说,在执行扫描过程时,我们分别标记由邻居扫描规则 1(强边顶点)、邻居扫描规则 2(邻居存款)和组扫描修剪的顶点。在这里,我们将 neighbor sweep 分为两个详细的子规则,因为它们都表现良好并且这两种策略的有效性在不同的数据集中不是很一致。对于 GLOBAL-CUT 的第 11 行中的每个顶点 v

,如果 v 被修剪(第 13 行),我们会增加相应策略的计数。我们还记录了未修剪和真正测试的顶点数(第 12 行)。对于每个数据集,我们在 20 到 40 的不同 k 下记录这些数据并获得平均值。结果如表2所示。NS 1和NS 2分别代表邻居扫描规则1和邻居扫描规则2。 GS是group sweep,Non-Pru是指未剪枝顶点的比例。请注意,有大量顶点被 k-core 技术预先修剪。

结果表明我们的修剪策略是有效的。超过 90% 的顶点在 DBLP、Cit 和 Cnr 中被修剪。 ND中完全剪枝的顶点比例最小,约为45%。在这些剪枝策略中,neighbor sweep rule 1 和 group sweep 的有效性取决于数据集的具体结构。邻居扫描规则 1 在 DBLP 中比组扫描执行得更好。由于这种策略修剪的顶点占总数的 67%(包括真正测试的顶点)。组扫描比 Cnr 中的邻居扫描规则 1 更有效。 group sweep 的百分比约为 48%,而 neighbor sweep rule 1 的百分比仅为 11%。这两种策略在其他数据集中的效果大致相同。作为比较,邻居扫描规则 2 密切依赖于现有的已处理顶点。随着不断测试或修剪顶点,它变得越来越有效。我们的结果表明它非常强大和稳定。这种策略的百分比在 Cit 中达到 68%,在所有其他数据集中超过 20%。

测试k-VCC的数量。图 11 给出了每个数据集在不同 k 值下的 k-VCC 数量。图 11 中,当 k 从 20 变为 40 时,所有测试数据集上的 k-VCC 数量都有下降趋势。原因是当增加k时,部分k-VCC不能满足要求,因此不会出现在结果列表中。 k-VCC数量的趋势ex-

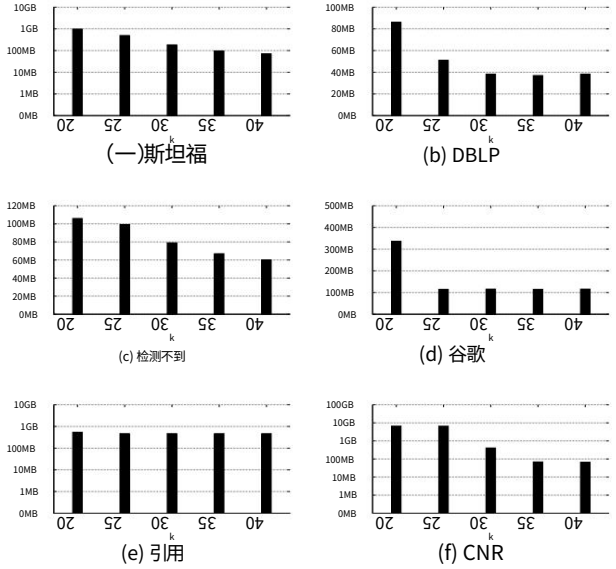


图 12:算法 VCCE* 的内存使用

解释了当图 10 中的 k 增加时我们算法的处理时间减少的原因。请注意,对于相同的 k 值,k-VCC 的数量在不同的数据集中可能会有很大差异。在同一数据集中,当 k 增加时,k-VCC 的数量可能会急剧下降。例如,当 k 从 20 增加到 25 时,Google 中的 k 个 VCC 数量减少了 10 倍。 k-VCC 的数量取决于每个特定图的图结构。

测试内存使用情况。图 12 显示了算法 VCCE* 在不同数据集上的内存使用情况,同时使用参数 k。请注意,四种算法 VCCE、VCCE-N、VCCE-G 和 VCCE* 的内存使用量非常接近,因为它们遵循相同的框架递归切割图,内存使用量主要取决于图的大小以及所有四种算法相同的分区图的数量。

因此,我们只显示 VCCE* 的内存使用情况。

从图中我们可以看出,随着 k 的增加,大多数数据集上的内存使用量都有下降的趋势。原因是双重的。首先,回想一下,在算法期间,首先为每个子图删除所有度数小于 k 的顶点。更高的 k 必须导致更多的顶点移除,因此,使图更小。其次,当 k 增加时,k-VCC 的数量减少,分区图的数量也减少,从而导致内存使用量减少。对于某些情况,当 k 增加时,内存 us age 增加,这是因为当 k 增加时,图的稀疏证书变得更密集,这需要更多的内存。总的来说,所有测试用例的内存使用都保持在一个合理的范围内。

6.3 可扩展性评估

在本节中,我们测试了我们提出的算法的可扩展性。

我们选择两个真实的图数据集 Google 和 Cit 作为代表。对于每个数据集,我们通过分别从 20% 到 100% 随机采样顶点和边来改变图的大小和图的密度。对顶点进行采样时,得到采样顶点的导出子图,对边进行采样时,得到边的入射顶点作为顶点集。在这里,我们只报告处理时间。内存使用量与顶点数成线性关系。

我们在实验中比较了四种算法。实验结果如图 13 所示。

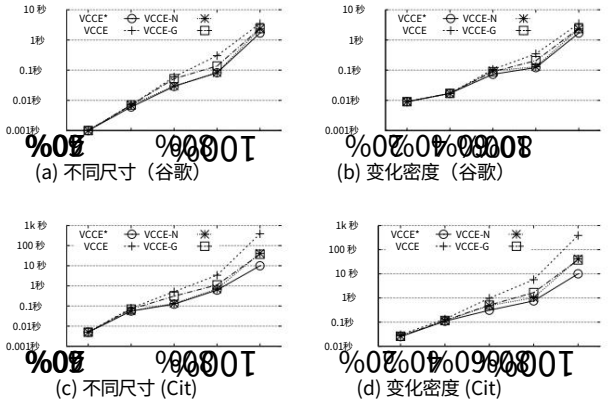


图 13:可扩展性评估

图 13 (a) 和 (c) 报告了我们提出的算法在改变 |V | 时的处理时间。分别在 Google 和 Cit 中。当 |V |增加,所有算法的处理时间都会增加。 VCCE*在所有情况下都表现最好,而 VCCE 是最差的。图 13（b)和（d)中的曲线分别报告了当 |E| 变化时我们的算法在 Google 和 Cit 中的处理时间。同样,VCCE*是所有测试案例中最快的算法。此外,当 |E| 时,VCCE* 和 VCCE 之间的差距增加增加。例如,在 Cit 中,当 |E| 时,VCCE* 的处理时间比 VCCE 快 20 倍达到 100%。结果表明我们的剪枝策略是有效的,我们的优化算法比基本算法更高效和可扩展。

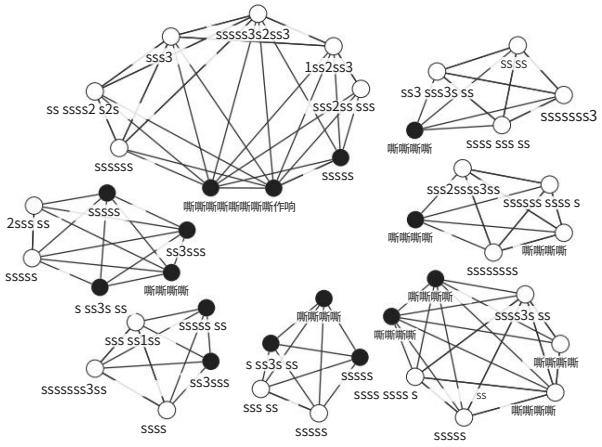
6.4 案例研究在本实验中,我

们进行案例研究以直观地揭示 k-VCC 的质量。我们从 DBLP (http://dblp.uni-trier.de/) 构建了一个协作图。图的每个顶点代表一个作者,如果两个作者有 3 个或更多的共同出版物,则他们之间存在一条边。由于原始图中的 k-VCC 太大而无法显示,我们选择了作者“Jiawei Han”和他的邻居。我们使用这些顶点的导出子图来进行这个案例研究。

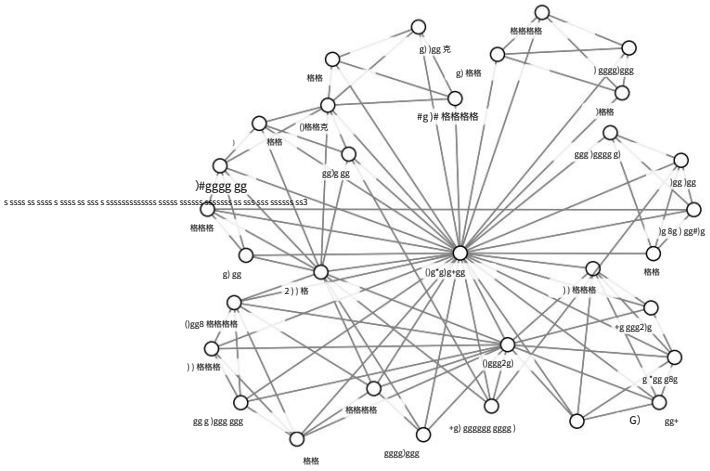
我们查询所有包含“Jiawei Han”的 4-VCC,结果如图 14（a)所示。我们获得了七个 4-VCC。他们每个人都很密集。如果一个顶点出现在多个 4-VCC 中,则它被标记为黑色。结果清楚地揭示了与 Jiawei Han 相关的不同研究组。一些核心作者出现在多个组中,例如“Philip S. Yu”和“Jian Pei”。作为对比,我们只得到一个 4-ECC,其中包含所有 4-VCC 中的作者。 4核的结果与本实验中的4-ECC相同。请注意,作者“Haixun Wang”出现在 4-ECC 和 4-cores 中,但没有出现在任何 4-VCC 中。这意味着他与“韩家伟”研究组的一些作者有合作,但这些作者来自不同的确定组,他不属于这些组中的任何一个。

7. 相关工作

内聚子图。基于指定度量的高效计算内聚子图最近引起了大量关注。 [5, 7] 提出了最大团问题的算法。然而,集团的定义过于严格。为了放松,提出了一些类似集团的指标。这些指标可以粗略地分为三类,1) 全局凝聚力,2) 局部度和三角测量,以及 3) 连通性凝聚力。



(a) 4-VCC



(b) 4-ECCs 和 4-core图

14:DBLP 案例研究 1. 全局凝聚力。

[17] 定义了一个 s-clique 模型,通过允许两个顶点之间的距离最多为 s 来放宽 clique 模型,即最短路径中最多有 s-1 个中间顶点。然而,它并不要求所有中间顶点都在 s-clique 本身中。为了处理这个问题,[19] 提出了一个 s-club 模型,要求所有中间顶点都在同一个 s-club 中。此外,k-plex 允许此类子图中的每个顶点最多可以错过 k 个邻居 [4, 23]。准团是一个有 n 个顶点和至少 γ* 条边的子图 [33]。这些类型的指标在全球范围内要求图表满足指定的密度或其他特定标准。他们没有仔细考虑每个顶点的情况,因此不能有效减少搭便车效应[31]。

格格 克

n

2+

2.局部度和三角剖分。 k-core 是最大子图,其中每个顶点的度数至少为 k [3]。它只要求图中每个顶点的最小邻居数不小于k。因此,每个顶点的非邻居数可能很大。当 k-core 的尺寸很大时,很难保持熟悉度。 k-truss 也在 [9, 27, 24] 中进行了研究。它要求 k-truss 中的每条边至少包含在 k-2 个三角形中。 k-truss 与 k-core 有类似的问题。很容易看出,如果两个内聚子图仅共享一条边,则它们可以简单地识别为一个 k-truss。另外,k-truss在一些流行的图如二分图中是无效的。该模型也被独立定义为 k-mutual-friend 子图并在 [36] 中进行了研究。基于三角形,参数为 k 的 DN-graph [28] 是一个 con

连通子图 G (V , E) 满足以下两个条件： 1) G 中的每一对连通顶点共享至少 λ 个公共邻居。 2) 对于任意 v ∈ V \V ,λ(V ∪ {v}) < λ;并且对于任何 ,λ(V \{v}) ⩽ λ。这样的度量标准似乎有点严格,并且 v ∈ V 会产生许多冗余结果。此外,检然很高　测所有 DN-graphs 是 NP-Complete。给出了近似解,时间复杂度仍 [28]。

3. 连接凝聚力。在这个类别中,大多数现有作品只考虑图的边连通性。图的边连通性是指删除后断开图的边的最小数量。 [32] 首先提出了一种算法,可以从一组数据图中有效地计算出频繁闭合的 k 边连通子图。但是,频繁闭子图可能不是导出子图。为了解决这个问题,[37] 给出了一种基于切割的方法来计算图中所有的 k 边连通分量。为了进一步提高效率,[6]提出了针对同一问题的分解框架,并在算法中实现了较高的加速比。

顶点连通性。 [14] 证明在未加权的有向图中计算最大流的时间复杂度达到 O(n 0.5m),而内部的每个顶点都有一条从它发出的边或一条进入它的边。该结果用于在 O(n 0.5m2) 时间内测试具有给定 k 的图的顶点连通性。 [13] 进一步将此类问题的时间复杂度降低到 O(k 3m+knm)。还有其他解决方案可以找到图的顶点连通性 [15, 12]。为了加快顶点连通性的计算,[8] 找到了一个稀疏的 k 顶点连通性证书,可以通过执行扫描优先搜索 k 次来获得。

八、结论

内聚图检测是一个具有广泛应用的重要图问题。大多数现有模型都会导致将不相关的子图组合成一个子图的搭便车效应。在本文中,我们首先研究了检测给定图中所有 k 顶点连通分量的问题,其中顶点连通性已被证明是一种有用的正式定义和社会群体凝聚力的度量。该模型有效地降低了搭便车效应,同时保留了许多良好的结构特性,如有界直径、高内聚性、有界图重叠和有界子图数。我们提出了一种多项式时间算法,通过重叠图分区框架枚举所有 k-VCC。我们提出了几种优化策略来显着提高算法的效率。我们使用七个真实数据集进行了大量实验,以证明我们方法的有效性和效率。

9. 参考文献

[1] JI Alvarez-Hamelin、L. Dall　Asta、A. Barrat 和 A. Vespignani。 k 核分解:大规模网络可视化的工具。 CoRR,abs/cs/0504107,2005 年。

[2] G. Bader 和 C. Hogue。一种在大型蛋白质相互作用网络中寻找分子复合物的自动化方法。 BMC 生物信息学,4(1),2003。

[3] V. Batagelj 和 M. Zaversnik。用于网络核心分解的 o(m) 算法。 CoRR，cs.DS/0310049,2003 年。

[4] D. Berlowitz、S. Cohen 和 B. Kimelfeld。最大 k-plexes 的有效枚举。在过程中。SIGMOD　15,2015 年。

[5] L. Chang、JX Yu 和 L. Qin。中的快速最大派系枚举稀疏图。算法, 66, 2013.

[6] L. Chang、JX Yu、L. Qin、X. Lin、C. Liu 和 W. Liang。通过图分解有效地计算 k 边连通分量。在过程中。 SIGMOD　13,2013 年。

[7] J. Cheng, Y. Ke, AW-C. Fu、JX Yu 和 L. Zhu。在大规模网络中寻找最大派系。 ACM 跨。数据库系统, 36, 2011.

[8] J. Cheriyan、M. Kao 和 R. Thurimella。扫描优先搜索和稀疏证书：一种改进的 k 顶点连接并行算法。 SIAM J. Comput., 22(1):157–174, 1993。

[9] J.科恩。桁架：用于社交网络分析的内聚子图。国家安全局技术报告，第 16 页，2008 年。

[10] W. Cui、Y. Xiao、H. Wang 和 W. Wang。本地搜索大图中的社区。在过程中。 SIGMOD 14，2014 年。

[11] J. Edachery、A. Sen 和 F. Brandenburg。使用 distance-k cliques 的图形聚类。在过程中。 GD 12, 1999.

[12] AH Esfahanian 和 S. Louis Hakimi。在计算图和有向图的连通性。网络，14(2):355–366, 1984。

[13] S.甚至。一种用于确定图的连通性是否至少为 k 的算法。 SIAM J. Comput., 4, 1975.

[14] S. Even 和 RE Tarjan。网络流和测试图连通性。 SIAM J. Comput., 4, 1975.

[15] Z.加利尔。寻找图的顶点连通性。 SIAM J. Comput., 9, 1980.

[16] X. Huang, H. Cheng, L. Qin, W. Tian, and JX Yu.在大型动态图中查询 k-truss 社区。在过程中。 SIGMOD 14，2014 年。

[17] 路卢斯。社会计量群体结构中的连通性和广义派系。 Psychometrika, 15, 1950.

[18] K. 门格尔。 Zur allgemeinen kurventheorie数学基础，10(1):96–115, 1927。

[19] RJ 莫肯。派系、俱乐部和氏族。质量和数量，13，1979.

[20] J. Moody 和 DR White。结构凝聚力和嵌入性：社会群体的等级概念。美国社会学评论，68，2000。

[21] JW Moon 和 L. Moser。关于图表中的派系。以色列杂志数学，3(1):23–28, 1965。

[22] J. Pattillo、N. Youssef 和 S. Butenko。网络分析中的集团松弛模型。欧洲运筹学杂志，226，2013 年。

[23] SB 塞德曼和 BL 福斯特。集团概念的图论概括。数学社会学杂志，6，1978。

[24] Y. Shao、L. Chen 和 B. Cui。并行高效的内聚子图检测。在过程中。 SIGMOD 14，2014 年。

[25] M. Stoer 和 F. Wagner。一个简单的最小割算法。 J.ACM, 44(4), 1997.

[26] A. Verma 和 S. Butenko。通过集团松弛的网络聚类：基于社区的方法。图分区和图聚类，588，2012。

[27] J. Wang 和 J. Cheng。大规模网络中的桁架分解。过程。 VLDB 基金会，2012 年 5 月。

[28] N. Wang, J. Zhang, K.-L. Tan 和 AKH Tung。先生基于三角剖分的密集邻域图发现。过程。2010 年 10 月 4 日。

[29] DR White 和 F. Harary。社会块的凝聚力网络：节点连通性和条件密度。社会学方法论，31(1):305–359, 2001。

[30] H.惠特尼。全等图和图的连通性。美国数学杂志，54(1):150–168, 1932。

[31] Y. Wu, R. Jin, J. Li, and X. Zhang.强大的当地社区检测：关于搭便车效应及其消除。过程。 VLDB Endow., 8, 2015.

[32] X. Yan, XJ Zhou, and J. Han.挖掘具有连通性约束的封闭关系图。在过程中。 ICDE 05，2005 年。

[33] Z. Zeng、J. Wang、L. Zhou 和 G. Karypis。相干封闭来自大型密集图形数据库的准集团发现。在过程中。 KDD 06，2006 年。

[34] H. Zhang、H. Zhao、W. Cai、J. Liu 和 W. Zhou。使用 k 核心分解来分析大型软件系统的静态结构。超级计算杂志，53（2），2010。

[35] Y. Zhang 和 S. Parthasarathy。提取分析和可视化网络中的三角形 k 核心图案。在过程中。 ICDE 12，2012 年。

[36] F. Zhao 和 AK Tung。用于社交网络视觉分析的大规模内聚子图发现。在过程中。 VLDB 12，第 6 卷，2012 年。

[37] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li. Finding来自大图的最大 k 边连接子图。在过程中。 EDBT 12，2012 年。