基于邻域的超图核分解

Naheed Anjum Arafat 新加坡国立大学 新加坡 naheed_anjum@u.nus.edu

Arijit Khan 奥尔堡大学 丹麦 arijitk@cs.aau.dk

Arpit Kumar Rai 印度理工学院坎普尔分校 印度 arpitkr20@iitk.ac.in

Bishwamittra Ghosh 新加坡国立大学 新加坡 bghosh@u.nus.edu

arXiv:2301.06426v2 [cs.SI] 9 Apr 2023

摘要

我们提出基于邻域的核分解：一种将超图分解为分层邻域凝聚子超图的新方法。其他超图分解方法（例如，将其转换为团图或二部图）在某些应用中并无意义；后者还会导致分解效率低下；而现有的基于度的超图分解无法区分邻域大小不同的节点。我们的案例研究表明，在疾病干预以及提取具有可证明近似保证且在应用层面有意义的最稠密子超图方面，所提出的分解方法优于基于度和基于团图的分解。我们提出了三种算法：**Peel**、其高效变体**E-Peel**，以及一种新颖的局部算法**Local-核**，并实现了并行版本。通过采用多种优化，我们最高效的并行算法**Local-core(P)** 能够在内存中于 91 秒内分解包含 2700 万个节点和 1700 万条超边的超图。最后，我们通过同时考虑邻域和度约束，提出了一种新的超图核模型——（邻域，度）核，设计了其分解算法**Local-核+Peel**，并展示了该模型在扩散传播方面的优越性。

PVLDB 引用格式：
Naheed Anjum Arafat、Arijit Khan、Arpit Kumar Rai 和 Bishwamittra Ghosh。《基于邻域的超图核分解》。PVLDB, 14(1)：XXX-XXX，2023年。doi:XX.XX/XXX.XX

PVLDB 工件可用性：
源代码、数据和/或其他工件已发布于 https://github.com/toggled/vldbsubmission。

1 引言

将图分解为具有层次结构的凝聚子图，是解决许多图数据管理问题的重要工具，例如社区检测 [51],最稠密子图发现[16], 识别影响力节点 [50], 以及网络可视化[2, 9]。根据凝聚性的不同定义，目前已有

本作品采用知识共享 署名-非商业性使用-禁止演绎 4.0 国际协议许可许可证。访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看本许可证副本。对于本许可证未涵盖的任何用途，请发送电子邮件至 info@vldb.org 获取许可。版权归所有者/作者所有。出版权已授权给 VLDB基金会。VLDB基金会会议录，第 14 卷，第 1 期，ISSN 2150-8097。doi:XX.XX/XXX.XX

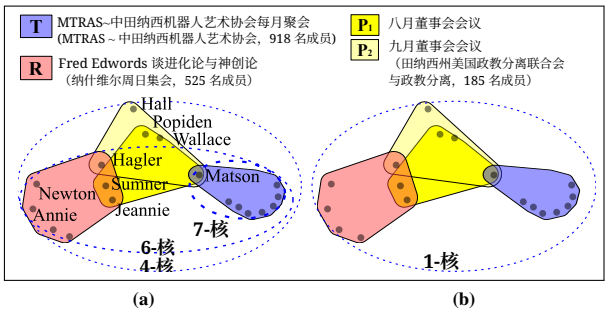


图 1：超图的 (a) 基于邻域和 (b) 基于度的核分解 H

多种分解方法：核分解 [10],桁架分解 [64], 核分解 [56], 等。在这项工作中，我们关注超图的分解。超图是图的一种推广，其中一条边可以连接两个以上的实体。

许多真实世界的关系由多元实体构成，例如，合著关系中的个体 [34], 议会投票中的立法者 [12], 网购购物车中的商品 [66], 蛋白质复合物中的蛋白质，以及代谢过程中的代谢物 [26, 30]。为方便起见，此类关系通常会被简化为团图或二部图（§2.2）。然而，出于两个原因，这种简化可能并不可取。首先，这种简化可能不具备实际意义，例如，某个蛋白质复合物中的一对蛋白质未必会发生两两相互作用，从而形成新的功能性蛋白质复合物。其次，将超图简化为团图或二部图会扩大问题规模 [37]：一个包含 2M 个节点和 15M 条超边的超图 [67]会被转换为一个包含 17M 个节点和 1B 条边的二部图。一个含有 m 条超边的 k均匀超图，其团图会有 $O(mk^2)$ 条边。二部图表示还需要进行距离 2 核分解[15, 47],，由于问题规模膨胀，这种分解的代价更高（§6）。

为此，我们提出了一种新颖的基于邻域凝聚性的超图核分解方法，将超图分解为嵌套的、强诱导极大子超图，使得每个子超图中的所有节点在该子超图中都至少拥有一定数量的邻居。所谓强诱导，是指当且仅当一条超边的所有组成节点都存在于某个子超图中时，该超边才存在于该子超图中。

示例1（基于邻域的核分解）。图 *1(a)* 中的超图 H 是根据纳什维尔聚会网络数据集 *[7]*中的四个事件（T、 R、 P_1、 P_2）构建的。每个

超边表示一个事件，超边中的节点是该事件的参与者。如果两个节点共同参与某个事件，则它们互为邻居。我们注意到 *Hall* 有4 个邻居：*Popiden*、*Hagler*、*Wallace* 和 *Matson*。同样，*Matson*、*Hagler*、*Jeannie*、*Sumner*、*Annie*、*Newton*、*Wallace* 和 *Popiden* 分别有13、10、9、9、6、6、6和6 个邻居。由于每个节点都有 ≥ 4 个邻居，基于邻域的4-核（记作 $H[V_4]$）就是超图 H本身。基于邻域的6-核是子超图 $H[V_6] = \{T, R\}$，因为 T（例如 *Newton*、*Annie*）的参与者和 R（例如 *Matson*）的参与者分别在 $H[V_6]$ 中有6 个和7 个邻居。最后，基于邻域的7-核是子超图$\{T\}$。

研究动机。现有文献中唯一的超图分解是基于度的分解（即与某个节点关联的超边数量）[53, 58]。在基于度的核分解中，k-核中的每个节点在该核中的度至少为 k。这种方法不考虑超边大小。因此，同一核中的节点可能具有差异极大的邻域大小。例如，*Hall* 和*Matson* 分别有 4 个和 13 个邻居，但在基于度的分解中，它们都属于同一个 1-核（图 1(b)）。在一些应用中，例如流行病学中的传染病传播、病毒式营销中的信息扩散，捕捉这类差异十分重要，因为已知在子超图中邻居数量相同的节点具有相似的扩散特征 [41]。事实上，从实践角度来看，如果传染病防控机构正在寻找会导致感染传播程度更高的关键事件（例如每位参与者与 6 个邻居会面），并因此需要对这些事件进行干预，那么这些事件就是{T、R}（邻居数6-核），因为它们会导致每位参与者至少与 6 个邻居会面。基于度的 H核分解无法区分不同事件和参与者之间的这类差异。此外，基于邻域的分解也符合逻辑：最内层的核包含 T，这是由一个相当受欢迎、拥有 918 名成员的团体组织的科技活动；其次是包含{R、T}的第二内层核。R 是由一个拥有 525 名成员、经常讨论宗教事务的世俗组织举办的活动。最外层的核包含由一个仅有 185 名成员的小众社会活动团体举办的两个活动。

挑战。一条超边可以关联两个以上的节点。此外，一对节点可能由多条彼此不同的超边关联。因此，将图的核分解算法直接应用于超图并非易事。例如，在图核分解的经典剥离算法中，移除一个节点时，其邻居的度会减 1：这使得一些重要优化成为可能，从而让剥离算法具有线性时间 复杂度且效率高 [10, 17, 54]。然而，在基于邻域的超图核分解中，删除一个节点可能使其邻居节点的邻域大小减少 1 以上。因此，要重新计算被删除节点的邻居节点的邻居数，必须构造删除操作后的残余超图，这使得分解具有多项式时间 复杂度，因而开销很大。此外，现有的、用于提高图核分解效率的下界 [15]并不适用于基于邻域的超图核分解，因此我们需要为此提出新的下界。下文将讨论将局部方法 [48], ——图核分解中最有效的方法之一——应用于超图时面临的挑战。

在局部方法中，针对图中的每个节点，核数估计会迭代更新 [40, 48] 或以分布式方式更新 [52]。节点的初始核数估计是其核数的上界。在后续轮次中，该估计会根据邻居节点的估计值迭代降低。[48] 使用 h-索引 [35] 进行此类更新。Khaouid 等人 [40] 也采用了类似方法迭代更新核数估计。他们证明，必须满足以下不变量：核数为 k 的每个节点，其 h-索引至少为 k，并且由 h-索引至少为 k 的节点诱导的子图中，每个节点都至少有 k 个邻居。前一条件成立，但后一条件在超图中可能不成立，因为由 h-索引$\geq k$ 诱导的子超图可能不包含那些部分包含其他节点的超边（§3.3）。由于这些“缺失”的超边，该子超图中某些节点的邻居数可能低于 k，从而违反核度条件。因此，在图中用于计算k-核 [40, 48, 52] 或 (k, h)-核 [47] 的局部方法，会得出不正确的 基于邻域的超图核（§6.1）。

我们的贡献总结如下。

新问题及其刻画（§2）。我们定义并研究了超图中基于邻域的核分解这一新问题。我们证明，基于邻域的 k-核是唯一的，且 k-核包含（$k+1$）-核。

精确算法（§3）。我们提出了三种精确算法，并分析了它们的正确性和时间复杂度。其中两种算法——**Peel**及其改进版**E-Peel**——采用经典剥离 [10], ，会引发超图的全局变化。对于**E-Peel**，我们推导出新的核数下界，以消除许多冗余的邻域重新计算。第三种算法**Local-core** 仅进行节点级局部计算。尽管现有的局部方法 [48,52]无法正确求出超图中基于邻域的核数，我们的算法**Local-core** 采用新的**核心校正**过程，确保核数计算正确。

优化与并行化策略（§4）。我们提出了四种优化策略，以提高**Local-core**的效率。针对超图的压缩表示（优化一）以及用于高效**核心校正**的一系列优化策略（优化二），都是核分解文献中的新方法。其他优化策略虽然受到图领域文献的启发，但此前尚未应用于超图相关工作中。我们还提出了适用于共享内存编程范式的**Local-core** 并行化方案。

（邻域，度）核（§5）。我们通过同时考虑邻域和度约束，定义了一种更通用的超图核模型——（邻域，度）核，并提出其分解算法**Local-core+Peel**，同时展示了该算法在扩散传播方面的优势。

实证评估（§6）表明，所提出的算法在真实和合成超图上均有效、高效且实用。我们的 OpenMP 并行实现**Local-core(P)** 可在 91 秒内分解包含 2700 万个节点和 1700 万条超边的超图。

应用（§7）。我们证明，与其他分解相比，我们的分解在扰乱扩散方面更有效。我们提出的用于体积最稠密子超图恢复的贪心算法可实现 $(d_{pair}(d_{card}-2)+2)$-近似，其中超边基数（该超边中的节点数）和节点对共现（包含该节点对的超边数）分别至多为 d_{card} 和 d_{pair}，

如果超图是图（$d_{card} = 2$），我们的结果推广了 Charikar 针对发现最稠密子图所提出的 2-近似保证 [16]。与基于度和团图分解的最稠密子超图相比，我们的体积最稠密子超图以不同方式捕捉重要的聚会事件。

2 我们的问题与刻画

超图。超图 $H = (V, E)$ 由节点集 V 和超边集 $E \subseteq P(V) \setminus \phi$ 组成，其中 $P(V)$ 是 V 的幂集。超边被建模为一个无序节点集。

邻居。超图 $H = (V, E)$ 中节点 v 的邻居 $N(v)$ 是与 v 一同出现在某条超边 $e \in E$ 中的节点 $u \in V$ 的集合。也就是说，$N(v) = \{u \in V \mid u \neq v \wedge \exists e \in E$ 满足 $u, v \in e\}$。

强诱导子超图 [6, 18, 33]。超图 $H = (V, E)$ 的强诱导子超图 $H[S]$ 由节点集 $S \subseteq V$ 诱导，是一个节点集为 S、超边集为 $E[S] \subseteq E$ 的超图，其中包含所有属于 S 子集的超边。

$$H[S] = (S, E[S]), \text{ where } E[S] = \{e \mid e \in E \wedge e \subseteq S\} \quad (1)$$

换句话说，强诱导子超图中的每条超边都必须存在于其父超图中。

2.1 问题形式化

超图 $H = (V, E)$ 的邻居数-k-核 $H[V_k] = (V_k, E[V_k])$，是满足每个节点 $u \in V_k$ 在 $H[V_k]$ 中至少有 k 个邻居的极大（强）诱导子超图。为简便起见，我们将邻居数-k-核记作 $H[V_k]$ H_k。H 的最大核 是使 H_k 非空的最大 k。节点 $v \in V$ 的核数 $c(v)$ 是满足 $v \in V_k$ 和 $v \notin V_{k+1}$ 的最大 k。超图的核分解 为每个节点赋予其核数。给定一个超图，本文研究的问题是如何正确且高效地计算其基于邻域的核分解。

2.2 与其他核分解的区别

可以大致借鉴文献中的两类方法。

方法一。将超图转换为其他对象（例如图），在该对象上应用现有的分解方法 [10, 15]，然后将分解结果投影回超图。例如，将超边替换为团，将超图转换为团图，再应用经典图核分解 [10]。还可以将超图转换为二部图：将超边表示为第二个分区中的节点；如果第二个分区中的超边包含第一个分区中的某个节点，就在这两个分区的节点之间创建一条边。最后，距离 2 核分解 [

15, 47] 得到了应用。在距离-2 核分解中，k核中的节点在子图中至少有 k 2 跳邻居。**首先**，它们得到的分解可能与我们的分解不同：在团图和距离-2 二部图分解中，$c(Popiden) = 5$，而*Popiden* 在我们的分解中核数为 4。团图和二部图分解未能识别出关系 *Popiden-Hagler* 不应存在，除非事件 P_2 存在，因为 P_2 是他们唯一共同参与的事件。**其次**，得到的分解可能不合理：同一利益集团的低重要性事件 P_1 和 P_2 被划分到不同的核中，导致难以

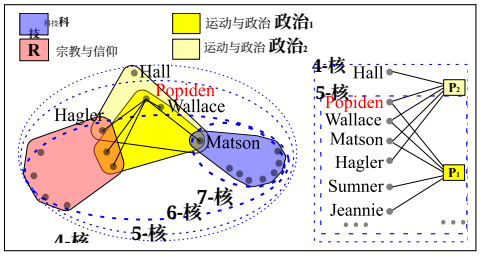


图 2：替代性分解有时并不可取：(a) H 的团图的核分解与 H 的二部图的距离 2 核分解产生相同的分解。相似事件（P_1 和 P_2）位于不同的核中。(b) H 的二部图表示。

将较不重要的事件与较重要的事件区分开来。**第**三，此类转换会扩大问题规模（§1）。二部图表示会导致低效的分解（§6.2）。

方法 2. 超图中节点 v 的度 $d(v)$ 是与 v [13] 关联的超边数量，即 $d(v) = |\{e \in E \mid v \in e\}|$。Sun 等人 [58] 将超图 H 的*deg-k-*核 H_k^{deg} 定义为 H 的极大（强）诱导子超图，其中每个节点 u 在 H_k^{deg} 中的度至少为 k。这种方法不考虑超边大小（§1）。因此，它不一定会产生与我们的方法相同的分解（图 1）。

2.3 邻居数-k-核：性质

定理 1. 对于任意 $k > 0$，邻居数-k-核 H_k 都是唯一的。

证明。假设（如果可能）超图 $H = (V, E)$ 存在两个不同的邻居数-k-核：$H_{k_1} = (V_{k_1}, E[k_1])$ 和 $H_{k_2} = (V_{k_2}, E[k_2])$。根据定义，$H_{k_1}$ 和 H_{k_2} 都是 H $H_k = V_{k_1} \cup$ 的极大强 V_{k_2} $E[V_{k_1} \cup V_{k_2}]$ $u \in V_{k_1} \cup V_{k_2}$ u V_{k_1} 诱导子超图。构造并超图 (,)。对于任意，必须属于 或 V_{k_2}。因此，u 在 H_{k_1} 或 H_{k_2} 中必须至少有 k 个邻居。由于 $E[V_{k_1}] \cup E[V_{k_2}] \subseteq E[V_{k_1} \cup V_{k_2}]$，$u$在 H_k中也必须至少有 k个邻居。由于 H_k是 H_{k_1}和 H_{k_2}的超图，H_{k_1}和H_{k_2}都不是极大的，从而导致矛盾。□

定理 2. $(k+1)$-核包含于 k-核中，$\forall k > 0$。

证明。假设存在某个节点 $u \in V_{k+1}$，使得 $u \notin V_k$。构造 $S = V_k \cup V_{k+1}$。由于 $u \notin V_k$，但 $u \in V_{k+1} \subset S$，因此得到$|S| \geq |V_k| + 1$。容易验证，每个节点 $v \in S$在 $H[S]$和 $|S| > |V_k|$中至少有 k个邻居。于是，V_k不是极大的，因此不是 nbr-k-核，这就产生了矛盾。定理得证。□

3 算法

我们提出了三种算法：算法 **Peel** 及其高效变体 **E-Peel** 受到类似图核计算的剥离方法的启发 [10, 15]。算法 **Local-core** 则受到图核计算局部方法的启发 [48, 52]。在图核分解的经典剥离算法中，移除一个节点时，其邻居的度会减少 1：这使得高效优化成为可能 [10, 17, 54], 例如，[10] 提出了分箱排序来

对节点排序，并开发了一种线性时间剥离算法。然而，在基于邻域的超图核分解中，删除一个节点可能会使其相邻节点的邻域大小减少超过 1。因此，要重新计算被删除节点的邻居的邻居数量，必须构造残余超图，这使得分解需要多项式时间且代价高昂。尽管算法 **E-Peel** 和 **Local-core**受到现有图算法系列的启发，但它们绝非简单的改编。对于**E-Peel**，我们为核数设计了一种新的局部下界，因为图核的下界 [15] 不足以满足我们的需求。对于**Local-core**，我们说明了直接改编局部算法 [40, 48, 52] 会导致核计算错误（§3.3 和 §6.1）。因此，我们设计了超图 h索引和局部核度约束，并利用它们正确计算超图核。

3.1 剥离算法

根据定理 2，可以通过从k-核中“剥离”所有邻域大小小于 $k+1$ 的节点，计算 $(k+1)$-核。算法1描述了我们的剥离算法：**Peel**，该算法按邻域大小递增的顺序处理节点（第 4-10 行）。 B 是一个列表向量：每个单元 $B[i]$ 都是一个列表，用于存储所有邻域大小为 i 的节点（第 3 行）。在第 k 次迭代中处理节点 v 时，其核数被赋值为 $c(v) = k$ （第 7 行），随后该节点会从“剩余”节点集合 V 中删除（第 10 行）。随后重新计算 v邻域内节点的邻域大小（每个邻域大小可能减少不止 1，因为删除 v 时，所有涉及 v 的超边也会被删除），并将这些节点移至 B 中的相应单元（第 8-9 行）。当超图中的所有节点都处理完毕且各自的核数都计算完成后，算法结束。

正确性证明。 初始时， $B[i]$ 包含所有邻域大小为 i的节点。删除节点 u的某个邻居时，会重新计算 u 的邻域大小，并将 u 重新分配到与其缩减后的邻域大小对应的新单元中，直到发现删除 u 的某个邻居 v 会使 u的邻域大小降至当前迭代编号 k 以下（第 9 行）。此时，我们正确地将 u的核数 c（u）赋为 $= k$。**(1)** 考虑第（$k-1$）次迭代结束时由剩余节点和超边构成的剩余子超图。显然， u 属于 k-核，因为在这个剩余子超图中， u 至少有 k 个邻居，且其中所有节点的邻域大小均为 $\geq k$。**(2)** 移除 v 会使 u的邻域大小降至当前迭代编号 k以下；因此，当当前迭代编号增加到 $k+1$时， u 将没有足够的邻居来留在（$k+1$）-核中。

时间复杂度。在算法1中，每个节点 v 都从 B中恰好处理一次；处理该节点并将其从 V中删除后，会重新计算 v的邻域中各节点的邻域大小。假设一个节点的最大邻居数和超边数分别为 d_{nbr}和 d_{hpe}。因此，算法1的时间复杂度为 $O\left(|V| \cdot d_{nbr} \cdot (d_{nbr} + d_{hpe})\right)$。

3.2 带界的高效剥离

算法1 的一个低效之处在于，它会更新每个邻居节点的单元索引 u，这些节点是已删除节点 v的邻居。为此，它必须计算新构建的子超图中 u 的邻居数量

算法1 剥离算法：**Peel**

输入： 超图 $H = (V, E)$
输出： 核数 $c(u)$ 每个节点 $u \in V$
1: **对于所有** $u \in V$ **执行**
2: 计算 $N_V(u)$ ▷ 在 $H = (V, E)$中 u的邻居集合
3: $B[|N_V(u)|] \leftarrow B[|N_V(u)|] \cup \{u\}$
4: **对于所有** $k = 1, 2, \ldots\ldots, |V|$ **执行**
5: **当** $B[k] \neq \phi$ **时执行**
6: 从 $B[k]$中移除节点 v
7: $c(v) \leftarrow k$
8: **对所有** $u \in N_V(v)$ **执行**
9: 将 u移至 B[最大值 $\left(|N_{V\setminus\{v\}}(u)|, k\right)$]
10: $V \leftarrow V \setminus \{v\}$
11: 返回 c

算法 2 带界的高效剥离算法：**E-Peel**

输入： 超图 $H = (V, E)$
输出： 每个节点的核数 $c(u)$ $u \in V$
1: **对于所有** $u \in V$ **执行**
2: 计算 $\mathbf{LB}(u)$
3: $B[\mathbf{LB}(u)] \leftarrow B[\mathbf{LB}(u)] \cup \{u\}$
4: $setLB(u) \leftarrow True$
5: **对所有** $k = 1, 2, \ldots, |V|$ **执行**
6: **当** $B[k] \neq \phi$ **时执行**
7: 从 $B[k]$中移除节点 v
8: **如果** $setLB(v)$ **则**
9: $B[|N_V(v)|] \leftarrow B$ [最大值 $(|N_V(v)|, k)] \cup \{v\}$
10: $setLB(v) \leftarrow False$
11: **else**
12: $c(v) \leftarrow k$
13: **对所有** $u \in N_V(v)$ **执行**
14: **如果** $\neg setLB(u)$, **则**
15: 将 u移动到 $B\left[\max\left(|N_{V\setminus\{v\}}(u)|, k\right)\right]$
16: $V \leftarrow V \setminus \{v\}$
17: 返回 c

。为延迟这项重新计算，我们通过引理 1 推导出 $c(u)$ 的局部下界，并利用该下界消除许多冗余的邻域重新计算和单元更新（算法 2）。其基本思路是，在某次迭代 k <中，节点 u 不会被删除，因为 $c(u)$的值不低于该下界，因此，在 k 的值达到 $c(u)$的下界之前，我们无需计算 u的邻域大小。我们的下界是局部的，因为它针对每个节点单独确定。

LEMMA 1 (LOCAL LOWER-BOUND). 令 $e_m(v) = \arg\max\{|e| : e \in E \wedge v \in e\}$ 为与 $v \in V$ 关联的基数最大的超边。对于所有 $v \in V$，

$$c(v) \geq \max\left(|e_m(v)| - 1, \min_{u \in V} |N(u)|\right) = \mathbf{LB}(v) \tag{2}$$

证明。注意， $c(v) \geq \min_{u \in V}|N(u)|$，因为输入超图中的所有节点都必须属于 $(\min_{u \in V}|N(u)|)$-核。接下来，我们通过反证法证明 $c(v) \geq |e_m(v)| - 1$。假设 $c(v) < |e_m(v)| - 1$（如果可能的话）。这意味着它不属于 $(|e_m| - 1)$-核，记作 $H[V_{|e_m|-1}]$。考虑 $V' = V_{|e_m|-1} \cup \{u : u \in e_m\}$。显然，$|V'| \geq |V_{|e_m|-1}| + 1$，因为 $v \notin V_{|e_m|-1}$，但 $v \in e_m$，所以 $v \in V'$。接下来我们证明， $H[V_{|e_m|-1}]$ 不是每个节点至少有 $|e_m| - 1$ 个邻居的极大子超图，这就构成了矛盾。

要证明 $H[V_{|e_m|-1}]$ 不是极大的，只需证明对于任意 $u \in V \setminus V_{|e_m|-1}$ $N_{V'}(u) \geq |e_m| - 1$。如果 $u \in V_{|e_m|-1} \subset V$ $|N_V(u)| \geq |e_m| - 1$，则 $|N_{V'}(u)| \geq |e_m| - 1$。如果 $u \in e_m$ $N_{V'}(u) \geq N_{e_m}(u) = |e_m| - 1$，则。

由于我们的前提 $|e_m| - 1 > c(v)$ 与 $H[V_{|e_m|-1}]$ 是 $(|e_m| - 1)$-核这一事实相矛盾，因此 $|e_m| - 1 \leq c(v)$。□

算法。 我们高效的剥离方法如算法 2 所示：**E-Peel**。在第 14 行，对于那些满足 为 True 的邻居节点，我们不会重新计算邻域并更新

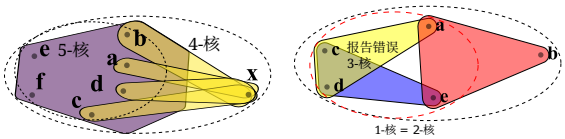


图 3：(a) 在计算 x 的核数期间，算法 2 不会对 x 的邻居 $\{a, b, c, d\}$ 执行邻域重新计算和单元更新；因此节省了四次冗余的邻域重新计算和单元更新。(b) 对于任意 $n > 1$，节点 a 的 h-索引（定义 2）永远不会从 $h_a^{(1)} = \mathcal{H}(2, 3, 3, 4) = 3$ 降至其核数 2：$\lim_{n\to\infty} h_a^{(n)} = 3 \neq$ 的核数。因为 a 始终至少有 3 个邻居（c、d 和 e），其 h-索引均不小于 3。因此，朴素方法会错误地报告一个3-核。

单元，从而提高效率。$setLB$ 对于那些已知 LB() 的节点，该条件 $N_V()$ 在当前迭代中未知。

例*2*。图 *3(a)* 展示了算法 *2* 在邻域重新计算和单元更新方面所作的改进。由于 $LB(x) = 1$，且每个邻居 $u \in \{a, b, c, d\}$ 都有 $LB(u) = 5$，算法 *2* 会先计算 $c(x)$，再计算 $c(u)$。得益于第 *1–4* 行中的基于局部下界的初始化以及 k 的升序迭代顺序，x 会在 u 之前被弹出。x 第一次从 B 中被弹出时，由于第 *9* 行，x 会变为 $B[4]$，并且在第 *10* 行中将 $setLB(x)$ 设置为 *False*。下一次弹出 x 时（同样是在迭代 $k = 4$ 中），算法会在第 *12* 行计算 $c(x)$。对于 u，$setLB(u)$ 仍为 *True*（第 *13–15* 行），因为 $setLB(u)$ 的默认初始化值为 *True*（第 *4* 行）。因此，对于 u，第 *15* 行中的计算均不会执行。直观地说，由于 *5*-核不包含 x，删除 x 和黄色超边不应影响正确计算 $c(u)$。$c(u)$ 会在其被弹出后的下一次迭代（$k = 5$）中计算，并被重新赋值为 $B[5]$，而 $setLB(u)$ 变为 *False*。

正确性证明。正确性证明遵循算法1的证明。当节点 v 在第 k 次迭代时从 $B[k]$ 中取出，我们检查 $setLB(v)$。(1) 引理 1 保证，如果我们从 $B[k]$ 中取出节点 v，且 $setLB(v)$ 为 True，那么 $c(v) \geq k$。在这种情况下，我们计算 $N_V()$ 的当前值，其中 V 表示剩余节点的集合，并将 v 插入单元：$B[\max(|N_V(v)|, k)]$。我们还设置 $setLB(v)$ = False，这意味着已知当前迭代中 $N_V(v)$ 的值。(2) 相反，如果我们从 $B[k]$ 中取出节点 v，且 $setLB(v)$ 为 False，则表明 $c(v) = k$，这与算法1中的论证相同。在这种情况下，我们将 v 的核数正确赋值为 k，并将 v 从 V 中移除。此外，对于 v 的那些邻居 u，若 $setLB(u)$ 为 True，意味着 $c(u) \geq k$，我们会适当延迟重新计算它们的邻域大小。

时间复杂度。按照算法1中的类似分析，**E-Peel**的时间复杂度为 $O(\alpha \cdot |V| \cdot d_{nbr} \cdot (d_{nbr} + d_{hpe}))$，其中$\alpha \leq 1$是算法2中的邻域重新计算次数与算法1中的邻域重新计算次数之比。根据我们在 §6 中的实验结果，**E-Peel** 的速度最高可比**Peel**快 17 倍。

3.3 局部算法

尽管**Peel** 及其高效变体**E-Peel** 能够正确计算核数，但它们必须在每次迭代中通过剥离节点和超边来修改剩余超图。剥离会对超图数据结构产生全局影响，因此必须按顺序执行。因此，通过并行化来进一步提升**Peel** 和**E-Peel** 的效率的空间很小。此外，它们不适用于时间受限的场景，而在这类场景中，高质量的部分解已足够。我们提出了一种新颖的局部算法，能够提供部分解，易于进行优化，并且可并行化。

在超图中直接采用局部算法：一个负面结果。Eugene 等人 [48] 采用 Hirsch 指数 [35]，（通常称为 h-指数，定义 1），提出了一种用于图核计算的局部算法。该算法依赖于一个递推关系，用于定义更高阶的 h-指数（定义 2）。图核计算的局部算法首先计算图中每个节点的 0 阶 h-指数。在每次迭代 $n > 0$ 中，算法使用上一轮迭代计算出的 $(n-1)$ 阶 h-指数，计算 n 阶 h-指数。众所周知，更高阶的 h-指数会单调收敛到图中该节点的核数。

定义 1（$\mathcal{H}$-算子 [35, 48]）。给定一个正整数有限集合 $\{x_1, x_2, \ldots, x_t\}$，$\mathrm{H}(x_1, x_2, \ldots, x_t) = y > 0$，其中 y 是满足以下条件的最大整数：在 $\{x_1, x_2, \ldots, x_t\}$ 中，至少有 y 个元素，且每个元素都不小于 y。

EXAMPLE 3 (H-OPERATOR). $\mathcal{H}(1,1,1,1) = 1, \mathcal{H}(1,1,1,2) = 1, \mathcal{H}(1,1,2,2) = 2, \mathcal{H}(1,2,2,2) = 2, \mathcal{H}(1,2,3,3) = 2, \mathcal{H}(1,3,3,3) = 3$

定义 2（h-阶数 n [48]-索引）。设 $\{u_1, u_2, \ldots, u_t\}$ 为图 $G = (V, E)$ 中节点 $v \in V$ 的邻居集合。节点 $v \in V$ 的 n阶 h索引，记为 $h(n)_v$，对任意$n \in \mathbb{N}$由以下递推关系定义：

$$h_v^{(n)} = \begin{cases} |N(v)| & n = 0 \\ \mathcal{H}\left(h_{u_1}^{(n-1)}, h_{u_2}^{(n-1)}, \ldots, h_{u_t}^{(n-1)}\right) & n \in \mathbb{N} \setminus \{0\} \end{cases} \tag{3}$$

对于通过局部算法进行的基于邻域的超图核分解，我们将 $h(0)_v$ 定义为超图 $H = (V, E)$中节点 v 的邻居数（而非图 G中的邻居数）。对于 $n > 0$，$h(n)_v$的定义保持不变。然而，直接采用局部算法计算超图核并不可行。尽管可以证明，为超图采用的序列（$h(n)_v$）存在极限，但该极限值不一定对每个 $v \in V$都是核数 $c(v)$。对于某些节点，其 h索引的极限值严格大于该节点的核数。原因如下。H 算子既是计算图核的必要条件，也是充分条件。已有研究表明，由 $G[S]$诱导的子图（其中 S 包含节点 v的所有邻居 u，且这些邻居满足 $h(\infty)_u \geq c(v)$）满足 $h(\infty)_v = c(v)$ [48, p.5 定理 1]。然而，对于超图，定义 2 不足以证明 $h(\infty)_v = c(v)$。因为无法保证 v 在子超图 $H[\{u: h_u^\infty \geq h_v^\infty\}]$中至少有 h_v^∞个邻居，而该子超图被报告为 $c(v)$-核。因此，所报告的 $c(v)$-核可能不正确。

例 *4*. 对于图 *3(b)* 中的超图，h-索引的极限值为 $h(\infty)_a = h(\infty)_c = h(\infty)_d = h(\infty)_e = 3$ 和$h(\infty)_b = 2$。无论 n 取值多大，公式 *(3)* 都无法帮助 $h(n)_a$ 达到 a的正确核数（=2）。在图 *3(b)* 中，节点 a 的三个邻居，即 c、d 和 e，其 h^∞值至少为 $3 = h_v^\infty$。但是，a在子超图 $H[\{a, c, d, e\}] = H[\{u: h_u^\infty \geq h_v^\infty\}]$中并没有至少 *3* 个邻居。因此，基于朴素 h-索引的局部方法报告的 *3*-核 $H[\{a, c, d, e\}]$并不正确。原因是，由于缺少 b，a和 e在 $H[\{a, c, d, e\}]$ 中不再互为邻居。

算法 3 满足局部核度约束的局部算法：**Local-core**

输入： 超图 $H = (V, E)$
输出： 每个节点 $v \in V$ 的核数 $c(v)$

对所有 $v \in V$ 执行
  $\hat{h}_v^{(0)} = h(0)_v \leftarrow |N(v)|$
对于所有 $n = 1, 2, \ldots, \infty$ **do**
  对于所有 $v \in V$ 执行
    $h_v^{(n)} \leftarrow$ 最小 $\left(\mathrm{H}(\{\hat{h}(n-1)u : u \in N(v)\}), \hat{h}(n-1)_v\right)$
  对于所有 $v \in V$ 执行
    $c(v) \leftarrow \hat{h}(n)_v \leftarrow$ **核心校正** $(v, h(n)v, H)$
  如果 $\forall v, \hat{h}_v^{(n)} == h_v^{(n)}$ **then**
    结束循环
返回 c

算法 4 核心校正过程

输入： 节点 v 的超图 h 索引 $h(n)v$，超图 H

当 $h_v^{(n)} > 0$执行
  计算 $E^+(v) \leftarrow \{e \in Incident(v) : h(n)u \geq h(n)v, \forall u \in e\}$
  计算 $N^+(v) = \{u : u \in e, \forall e \in E^+(v)\} \setminus \{v\}$
  如果 $|N^+(v)| \geq h(n)v$ **则**
    返回 $h(n)v$
  else
    $h(n)v \leftarrow h(n)v - 1$

具有局部核度约束的局部算法。受上述观察启发，我们将约束定义为一个充分条件：满足该条件后，可以保证对于每个节点 v，1）其 h-索引序列收敛；2）极限值 $h(\infty)v$ 满足以下条件：v 在由 $H[u : h(\infty)u \geq h(\infty)v]$ 诱导的子超图中至少有 $h(\infty)v$ 个邻居。第一项条件对于算法终止至关重要。正如例 4 中所讨论的，第二项条件对于正确计算核数至关重要。

定义 3（局部核度约束（Lcc））。给定一个正整数 k，对于任意节点 $v \in V$，令 $H^+(v) = (N^+(v)\ E^+(v))$为 H 的子超图，使得对于任意 $n > 0$

$$E^+(v) = \{e \in Incident(v) : h_u^{(n)} \geq k, \forall u \in e\}$$
$$N^+(v) = \{u : u \in e, \forall e \in E^+(v)\} \setminus \{v\} \quad (4)$$

节点 v 的局部核度约束在 k 处满足，记作 $LCCSAT(k)$，当且仅当 $\exists H^+(v)$ 至少包含 k 个节点，即 $|N^+(v)| \geq k$。此处，$Incident(v)$ 是与 v 关联的超边集合。

我们基于 $LCCSAT$ 的概念以及为 $h(n)v$ 重新定义的递推关系，定义超图 h-索引。

定义4（超图 h-阶超图索引 n）。节点 v 的 n 阶超图 h-索引，记作 $\hat{h}(n)v$，对任意自然数 $n \in \mathbb{N}$，定义如下递推关系：

$$\hat{h}_v^{(n)} = \begin{cases} |N(v)| & n = 0 \\ h_v^{(n)} & n > 0 \wedge LCCSAT(h_v^{(n)}) \\ \max\{k \mid k < h_v^{(n)} \wedge LCCSAT(k)\} & n > 0 \wedge \neg LCCSAT(h_v^{(n)}) \end{cases} \quad (5)$$

h（n）v 是超图上新定义的递推关系：

$$h_v^{(n)} = \begin{cases} |N(v)| & n = 0 \\ \min\left(\mathcal{H}\left(\hat{h}_{u_1}^{(n-1)}, \hat{h}_{u_2}^{(n-1)}, \ldots, \hat{h}_{u_t}^{(n-1)}\right), \hat{h}_v^{(n-1)}\right) & n \in \mathbb{N} \setminus \{0\} \end{cases} \quad (6)$$

方程（5）和（6）中的递推关系是耦合的：$\hat{h}(n)v$ 取决于 $h(n)v$ 的求值，而后者又取决于 $\hat{h}(n-1)v$ 的求值。正如正确性分析所证明的，这种相互依赖使两个序列都能收敛。

Local-core（算法 3）根据公式 (6) 和公式（5），分别将每个节点的 $h(0)v$ 和 $\hat{h}(0)v$ 初始化为 $|N(v)|$（第 1–2 行）。在每次迭代 $n > 0$中，算法 3 首先根据公式 (6) 计算每个节点的$h(n)v$（第 4–5 行）。为了判断算法是否应在该次迭代终止 n（第 8–9 行），算法使用算法 4 计算 $\hat{h}(n)v$。算法 4 会检查每个节点 $v \in V$，判断 $LCCSAT(h(n)v)$ 是 True 还是 False。根据公式（5），如果 $LCCSAT(h(n)v)$ 为 True，则返回 $h(n)v$；如果 $LCCSAT(h(n)v)$ 为 False，则返回一个小于 $h(n)v$ 的合适值。返回值 $\hat{h}(n)v$ 被视为该次迭代中核数 $c(v)$ 的估计值（第 7 行）。为计算$LCCSAT(h(n)v)$，算法 4 会在第 4 行检查第 2–3 行构造的子超图$H^+(v) = (N^+(v)\ E^+(v))$是否至少包含$h(n)v$ 个 v的邻居。如果 v 在子超图 $H^+(v)$中至少有 $h(n)v$ 个邻居，根据公式（5），无需对 $h(n)v$ 进行修正。在这种情况下，算法 4 会在第 5 行返回 $h(n)v$。如果 v 在子超图 $H^+(v)$中没有至少 $h(n)v$ 个邻居（第 4 行），根据定义3，$LCCSAT(h(n)v)$ 为 False。根据公式（5），需要修正 $h(n)v$。为寻找合适的、低于 $h(n)v$ 的修正值以及合适的子超图 $H^+(v)$，第 7 行不断将 $h(n)v$ 减 1。将 $h(n)v$ 减小会使 $|N^+(v)|$ 增加，同时使 $h(n)v$ 减小，直到满足第 4 行中的条件。最终必定能找到合适的子超图。

定理 4 证明，算法 3 在该时刻返回的数值确实与真实核数一致。终止条件 $\hat{h}(n)v = h(n)v$ 必定会在某个时刻满足，因为定理 3 证明了 $\lim_{n\to\infty} \hat{h}(n)_v = \lim_{n\to\infty} h(n)_v \quad \forall v \in V$。

示例*5*。考虑算法 *3* 的第 $n = 1$ 次迭代，输入为图 *3(b)* 中的超图时的情形。该算法在第 *7* 行将核估计 $h(1)_a = 3$ 校正为 $\hat{h}(1)_a = 2$。这是因为在核心校正的第 *4* 行，算法发现，对于$h(1)_a = 3$，a在 $H^+(v) = H[\{a, c, d, e\}]$中只有 $|N^+(v)| = 2$ 个邻居，因此违反了条件 $|N^+(v)| > h(1)_a$。因此，需要校正 $h(1)_a$，以满足 $LCCSAT(h(1)_a)$。在核心校正的第 *7* 行，算法将 $h(1)_a$ 减一。随后，对于 $h(1)_a = 2$，子超图$H^+(a) = H$确实满足 $LCCSAT(h(1)_a)$。这就是算法 *3* 如何校正例 *4* 中讨论的核数错误情形。

3.4 局部 Local-core 的理论分析

正确性证明。算法 3 会在有限次迭代后终止，因为根据定理 3，对于任意 $v \in V$，序列 $(h(n)v)$ 和 $(\hat{h}(n)v)$ 都是有限的，且具有相同的极限。在极限处，$\forall v \in V$，$\lim_{n\to\infty} h(n)_v = \lim_{n\to\infty} \hat{h}(n)v$ 成立；由定理 4 可得 $\forall v \in V$，$\lim_{n\to\infty} h(n)_v = \lim_{n\to\infty} \hat{h}(n)_v = c(v)$。因此，算法终止时，会为每个节点 $v \in V$ 返回正确的核数。

定理 3. 对于超图 $H=(V,E)$ 中的任意节点 $v\in V$，由公式 (6) 定义的两个序列 $(h(n)_v)$ 和由公式 (5) 定义的 $(\hat{}h(n)_v)$ 具有相同的极限：$\lim_{n\to\infty} h(n)_v = \lim_{n\to\infty} \hat{}h(n)_v$。

证明。通过交错（$h(n)_v$）和（$\hat{}h(n)_v$）中的分量，构造一个新序列（$h\hat{}h_v$），如下所示：

$$(h\hat{h}_v) = (h_v^{(0)}, \hat{h}_v^{(0)}, h_v^{(1)}, \hat{h}_v^{(1)}, h_v^{(2)}, \hat{h}_v^{(2)} \ldots)$$

我们首先证明，这个记为（hh_v）的交错序列 [63,Defn 680],$\hat{}$ 单调不增，且下界为 0。任意选取（$h\hat{}h_v$）中的一对相邻分量。情况 1（$\hat{}h(n-1)_v, h(n)_v$）：对于 $n\geq 1$，若 $\mathrm{H}\left(\hat{}h(n-1)_{u_1}, \hat{}h(n-1)_{u_2}, \ldots\ldots, \hat{}h(n-1)_{u_t}\right) >$

$\hat{}h(n-1)_v$，根据递推关系（公式 (6)）$h(n)_v = \hat{}h(n-1)_v$。否则，$\hat{}h(n-1)_v \geq \mathrm{H}\left(\hat{}h(n-1)_{u_1}, \hat{}h(n-1)_{u_2}, \ldots\ldots, \hat{}h(n-1)_{u_t}\right) = h(n)_v$。这里，最后一个表达式中的等式同样由公式 (6) 中的递推关系得出。因此，对于任意 $n\geq 1$，$\hat{}h(n-1)_v \geq h(n)_v$。

情形 2（$h(n)_v, \hat{}h(n)_v$）：对于 $n=0$，根据公式（5），$h(0)_v = \hat{}h(0)_v$。对于 $n>0\wedge LCCSAT(h(n)_v)$，同样根据公式（5），$h(n)_v = \hat{}h(n)_v$。对于 $n>0\wedge\neg LCCSAT(h(n)_v)$，$\hat{}h(n)_v = \max\{k: k<h(n)_v \wedge LCCSAT(k)\} < h(n)_v$。因此，对于任意 $n\geq 0$，无论 $LCCSAT(h(n)_v)$ 如何，$h(n)_v \geq \hat{}h(n)_v$。

由于对于任意选取的一对相邻分量，左侧分量都不小于右侧分量，因此序列 $(h\hat{}h)$ 是单调非增的：$h(0)_v \geq \hat{}h(0)_v \geq h(1)_v \geq \hat{}h(1)_v \geq h(2)_v \geq \hat{}h(2)_v \geq \ldots$

接下来，我们通过归纳法证明，对于所有 $n\geq 0$，$h(n)_v \geq 0$ 且 $\hat{}h(n)_v \geq 0$。对于 $n=0$，根据公式 (6) 和公式（5），$h(0)_v = \hat{}h(0)_v = N(v) \geq 0$。假设对于 $n=m$，$h(m)_v \geq 0$ 且 $\hat{}h(m)_v \geq 0$（归纳假设）。对于 $n=m+1$，$h(m+1)_v$ 的递推关系可能是 $|N(v)|\geq 0$，也可能是 H 算子的输出（根据定义 1，该值严格大于 0），或者是 $\hat{}h(m)_v \geq 0$（根据归纳假设）。考虑这三种情况：$h(m+1)_v \geq 0$。对于公式（5）中定义的任意 $n=m+1$，$\hat{}h(m+1)_v$ 可能是 $|N(v)|\geq 0$，也可能是 $h(m+1)_v \geq 0$（刚刚已证明），或者是值 $k^* = \max\{k: k<h(m+1)_v \wedge LCCSAT(k)\}$。集合 $\{k: k<h(m+1)_v \wedge LCCSAT(k)\}$ 必须包含 0。因为对于任意 v，$h(m+1)_v \geq 0$（已证明），且 $LCCSAT(0)$ 显然为真。由于集合 $\{k: k<h(m+1)_v \wedge LCCSAT(k)\}$ 包含 0，该集合的最大值 k^* 必定至少为 0。考虑这三种情况：$\hat{}h(m+1)_v \geq 0$，由此完成证明：交错序列有下界 0。

因此，$(h\hat{}h)$ 单调非增，且下界为 0。根据单调收敛定理，[8], $(h\hat{}h_v)$ 存在极限。交错序列有极限，当且仅当其组成的序列对均收敛且极限相同 [63]。由于（$h\hat{}h_v$）存在极限，序列（$h(n)_v$）和（$\hat{}h(n)_v$）收敛到相同的极限：$\lim_{n\to\infty} h(n)_v = \lim_{n\to\infty} \hat{}h(n)_v$。□

定理 4. 如果在终止迭代时所有节点 $v\in V$ 均满足局部核约束，则终止迭代时修正后的 h-索引 $\hat{}h(\infty)_v$ 满足：$\hat{}h(\infty)_v = c(v)$。

证明。给定任意 $v\in V$，我们首先证明 $c(v)\geq \hat{}h_v^\infty$，接着证明 $\hat{}h_v^\infty \geq c(v)$。

构造 $N'\subseteq V$，其中每个节点 $w\in N'$ 都具有其收敛值 $\hat{}h_w^\infty \geq \hat{}h_v^\infty$。由于在终止迭代时，所有这类 $\hat{}$ 节点 w 都在各自的 h_w^∞ 处满足 LCC，显然 w 在任意整数 $k<\hat{}h_w^\infty$ 处也满足 LCC，包括 $k=\hat{}h_v^\infty$。根据 $\hat{}LCCSAT(h_v^\infty)$ 的定义（对于 w），存在一个对应的子超图 $H^+(w)=(N^+(w)\ E^+(w))$，使得任意 $u\in N^+(w)$ 满足 $\hat{}h_u^\infty \geq \hat{}h_v^\infty$，且 $|N^+(w)|\geq \hat{}h_v^\infty$。构造一个新的超图 $H^+[N']=(\cup_w N^+(w)\ \cup_w E^+(w))$。根据 $H^+[N']$ 的构造，任何节点 $u\in H^+[N']$ 都满足 $\hat{}h_u^\infty \geq \hat{}h_v^\infty$。因此，对于这样的 u，以下关系成立：$u\in N'$（根据 N' 的构造），u 对应的 $N^+(u)\subseteq \cup_w N^+(w)$（根据 $H^+[N']$ 的构造），以及 $|N^+(u)|\geq \hat{}h_v^\infty$（根据 LCC 的定义）。因此，$H^+[N']$ 中的每个节点 u 在 $H^+[N']$ 中至少有 $\hat{}h_v^\infty$ 个邻居。

由于 $\hat{}h_v^\infty$-核是一个极大子超图，其中每个节点在该子超图中至少有 $\hat{}h_v^\infty$ 个邻居，$H^+[N']$ 是 $\hat{}h_v^\infty$-核的子超图：$H^+[N']\subseteq \hat{}h_v^\infty$-核。注意，$v\in H^+[N']$。根据核数的定义，$c(v)$ 是使 $v\in c(v)$-核成立的最大整数。因此，$c(v)\geq \hat{}h_v^\infty$。

接下来，我们证明 $c(v)$-核记为 H'。定义 $\mathrm{LBH}' = \min_{u\in H'}|N_{H'}(u)|$。由于 $c(v)$-核中的每个节点 $u\in H'$，$|N_{H'}(u)|\geq c(v)$，在该核中至少有 $c(v)$ 个邻居：$v\in H'$，对任意 v，均成立。因此，$\mathrm{LBH}' \geq c(v)$。由于 $v\in H'$，且 $\mathrm{LCCSAT}(\hat{}h_v^\infty)$ 为真，根据引理 2：$\hat{}h_v^\infty \geq \mathrm{LBH}' \geq c(v)$。□

引理 2。设 LBH' 为子超图 $H'=(V'E')$ 中任意节点的最少邻居数，其中该子超图属于 $H=(V,E)$。对于任意 $v\in V'$ 和整数 $n\geq 0$，$LCCSAT(\hat{}h(n)_v)$ 为真，则 $\Longrightarrow \hat{}h(n)_v \geq LBH'$。

证明。我们通过归纳法进行证明。对于 $n=0$ and $v\in V'$，$\hat{}h(0)_v = h(0)_v = |N(v)| \geq \min_{u\in V'}|N_{H'}(u)| = \mathrm{LBH}'$。假设命题对 $n=m$ 成立：$LCCSAT(\hat{}h(m)_v)$ 为真 $\Longrightarrow \hat{}h(m)_v \geq \mathrm{LBH}'$，对所有 $v\in V'$ 均成立。我们证明该命题对 $n=m+1$ 也成立。

根据 $\hat{}h(n)_v$ 的定义，由于 $LCCSAT(\hat{}h(m+1)_v)$ 为真且 $m+1>0$，因此可得 $\hat{}h(m+1)_v = h(m+1)_v$。但是 $h(m+1)_v$ 对于 $m+1>0$ 只能取两个值：要么是 $h(m+1)_v = \hat{}h(m)_v$，要么是 $h(m+1)_v = \mathrm{H}\left(\hat{}h(m)_{u_1}、\hat{}h(m)_{u_2}、\ldots\ldots、\hat{}h(m)_{u_t}\right)$。这里 u_1、……、u_t 是 H中的 v的邻居。对于前一种情况，$h(m+1)_v = \hat{}h(m)_v \geq \mathrm{LBH}'$（根据归纳假设）。对于后一种情况，$\mathrm{H}\left(\hat{}h(m)_{u_1}、\hat{}h(m)_{u_2}、\ldots\ldots、\hat{}h(m)_{u_t}\right) \geq \mathrm{LBH}'$，因为至少存在 LBH' 个操作数，且每个操作数的值都至少为 LBH'。这是因为（1）$v\in V' \Longrightarrow \mathrm{LBH}' \leq |N_{V'}(v)|$（根据 LBH' 的定义）。因此，使 $u_i\in V'$ 成立的 u_i 的数量至少为 LBH'。（2）对操作数 $\{u_i: u_i\in V'\}$ 应用归纳假设，u_i 在 $\hat{}h(m)_{u_i}$ 处满足 LCC；因此 $\hat{}h(m)_{u_i} \geq \mathrm{LBH}'$。综合所有情况，$\hat{}h(m+1)_v \geq \mathrm{LBH}'$。□

时间复杂度。假设算法 3 在第 3 行 for 循环的第 τ 次迭代时终止。每次迭代的时间复杂度为 $O(\sum_v |N(v)|(d(v)+|N(v)|)+\sum_v |N(v)|)$，第一项对应第 6–7 行，第二项对应第 4–5 行。计算 H- 算子需要获取超图中 v 的邻居的 h-索引，耗时为线性：$O(|N(v)|)$。核心校正 v 至多需要进行 $|N(v)|$ 次 while 循环迭代（算法 4，第 1 行）；每次迭代中，构造 $E^+(v)$ 和 $N^+(v)$ 各自的开销为 $O(d(v)+|N(v)|)$。因此，Local-core 的时间复杂度为 $O(\tau * \sum_v (d(v)|N(v)|+|N(v)|^2))$。

迭代次数的上界。为了推导 **Local-core** 所需迭代次数的上界 τ，我们定义超图中节点的 邻域层次结构 这一概念，并证明每个节点都会在第 3 行（算法 3）中的 for 循环迭代器达到其邻域层次时收敛。

定义 5（邻域层次结构）。给定一个超图 H，第 i 阶邻域层次结构记为 N_i，它是 $H[V_i]$，中邻居数量最少的节点集合，其中 $V_i = V \setminus \cup_{0 \le j < i} N_j$。形式化地，

$$N_i = \{v : \arg\min_{v \in V'} |N_{H[V']}(v)|\} \tag{7}$$

考虑图 1(a) 中的超图 H。由于 *Hall* 有 4 个邻居，$N_0 = \{Hall\}$。删除 *Hall* 后，剩余子超图中邻居数量最少（4 个）的节点是 *Wallace* 和 *Popiden*。因此，$N_1 = \{Wallace, Popiden\}$。删除它们后，$R$ 中的节点在剩余子超图中邻居数量最少（6 个）。因此，$N_2 = R$。最后，$N_3 = T$。

引理 3. 给定两个顶点 $v_i \in N_i$、$v_j \in N_j$，满足 $i \le j$，则有 $c(v_i) \le c(v_j)$。

引理 3 源于剥离算法的正确性。

定理 5（单节点收敛）。给定超图 H 中的一个节点 $v \in N_i$，有 $\forall n \ge i$，$\hat{h}(n)_v = c(v)$。

证明。我们对 i 使用归纳法。对于 $i = 0$，N_0 是超图中邻居数量最少的节点集合。因此，$\forall v \in N_0$，$c(v) = |N(v)|$，根据公式（5）（对于 $n = 0$），我们有：$h(0)_v = |N(v)| = c(v)$。假设该定理对不超过$i = m$的情况成立，即 $\forall u \in N_j \mid j \le m$ 有 $\forall n \ge j$，$\hat{h}(n)_u = c(u)$。令$v \in N_{m+1}$。我们需要证明 $\hat{h}(m+1)_v = c(v)$。

根据定理 4 和定理 3，$\hat{h}(m+1)_v \ge c(v)$，因为 $\hat{h}$ 单调非增，且以 $c(v)$ 为极限。

现在我们证明 $\hat{h}(m+1)_v \le c(v)$。根据公式（5），有两种情况。情况 1：（$\hat{h}(m+1)_v = h(m+1)_v$）：在这种情况下，根据公式 (6)，$\hat{h}(m+1)_v = h(m+1)_v \le \mathrm{H}(\hat{h}^m_{u_1}, \hat{h}^m_{u_2}, \ldots, \hat{h}^m_{u_t})$，因为两个整数的最小值不大于其中任意一个。情况 2：（$\hat{h}(m+1)_v < h(m+1)_v$）：在这种情况下，出于与情况 1 相同的原因，$\hat{h}(m+1)_v < h(m+1)_v \le \mathrm{H}(\hat{h}^m_{u_1}, \hat{h}^m_{u_2}, \ldots, \hat{h}^m_{u_t})$。因此，只需证明 $\mathrm{H}(\hat{h}^m_{u_1}, \hat{h}^m_{u_2}, \ldots, \hat{h}^m_{u_t}) \le c(v)$。

为此，我们将 v 的 t 邻居划分为两个集合 S' 和 S''，使得 $S' = \{u' : u' \in N(v) \wedge u' \in N_j$ 对于某个 $j < m+1\}$，且 $S'' = \{u'' : u'' \in N(v) \wedge u'' \in N_j$对于某个 $j \ge m+1\}$。将 H 中的项按如下顺序排列：$\mathrm{H}(\hat{h}^m_{u'_1}, \ldots, \hat{h}^m_{u'_x}, \hat{h}^m_{u''_1}, \ldots, \hat{h}^m_{u''_y})$，其中 $u'_x \in S'$ 且 $u''_y \in S''$。(1)：$\forall u'_x \in S'$，根据归纳假设和引理 3，$h^m_{u'_x} = c(u'_x) \le c(v)$，因为 $v \in N_{m+1}$ 且 $u'_x \in N_j$，使得 $j < m+1$，根据 S' 的定义。(2)：$y = |S''|$，且 S'' 包含 $H' = H[\cup_{j \ge m+1} N_j]$ 中 v 的所有邻居。H' 是一个子超图，其中每个节点（包括 $v \in N_{m+1}$）的邻居数至少为$|S''|$。因此，H' 中每个节点的核数至少为 $|S''|$。于是，$c(v) \ge |S''| = y$。结合结果 (1) 和 (2)，H () 中前 x 项均为 $< c(v)$；即使接下来的 y 项可能是 $\ge c(v)$，其数量也少于 $c(v)$。因此，$\mathrm{H}(\hat{h}^m_{u_1}, \hat{h}^m_{u_2}, \ldots, \hat{h}^m_{u_t}) \le c(v)$，证明完毕。□

显然，τ 至多为节点的最大邻域层次结构。

算法 5 优化的局部算法：**Local-core(OPT)**

输入： 超图 $H = (V, E)$
输出： 核数 $c(v)$ 针对每个节点 $v \in V$

```
1: 构造 CSR 表示/*Opt-I*/
2: 对所有 v ∈ V 执行
3:     计算局部下界(v) /* Opt-IV 的局部下界 */
4:     c(v) ← ĥ(0)v ← h(0)v ← |N(v)| /* Opt-III 的核估计 c(v) 已初始化*/
5: 对所有 n = 1,2,...,∞ 执行
6:     对所有 v ∈ V 执行
7:         如果 h(n)v > LB(v) 则 /* Opt-IV*/
8:             c(v) ← h(n)v ← 最小化 (H({c(u): u ∈ N(v)}) c(v)) /* Opt-III*/
9:     对所有 v ∈ V 执行
10:        如果 h(n)v > LB(v) 则 /* Opt-IV */
11:            c(v) ← ĥ(n)v ← 核心校正 (v, h(n)v , H) /* Opt-II & Opt-III*/
12:    如果 ∀v, ĥ(n)v == h(n)v 则
13:        终止循环
14: 返回 c
```

4 局部核算法的优化与并行化

我们提出四种优化，以提高**Local- core** 的效率（§3.3）。算法 5给出了优化后算法**Local-core(OPT)**的伪代码，其中标明了全部四种优化。优化一采用稀疏表示，以高效执行超图中的邻域查询；优化二则包含三项针对具体实现的改进，以高效执行**核心校正**。此前针对图和超图的核分解研究均未采用优化一和优化二。我们的优化三和优化四受到 [47],的启发；该文提出了类似的优化，分别用于提高图的(k, h)-核分解的收敛速度和消除冗余计算。然而，此类优化尚未应用于此前与超图相关的研究中。

优化一（压缩超图表示）。**Local-core** 在超图结构上执行两种基本的邻域查询：邻居枚举（用于 h-索引计算，算法 3 第 2 行和第 5 行）以及关联超边枚举（在**核修正**期间，算法 4 第 2 行）。朴素实现会为邻居计数查询维护一个 $|V| \times |V|$ 矩阵，并为关联超边查询维护一个 $|V| \times |E|$ 矩阵。然而，在内存中存储这类矩阵代价高昂，而且对于大型超图而言并无必要，因为实际应用中的这些矩阵通常是稀疏的。因此，必须采用压缩稀疏表示，例如使用压缩稀疏行（CSR）来表示这些矩阵。例如，在 CSR 表示中，我们使用两个数组 F 和 N 来存储邻居。对于所有节点 $v \in V$，N [v] 存储 F 中包含节点 v的邻居的起始索引。为便于执行邻域枚举查询，节点 v的邻居存储在 F 中索引连续的位置 N [v]、N [v] + 1、…、N [v + 1] − 1 处。任何通过超边与 v相连的邻居 u，都有其自身的邻居集合 $N(u)$，存储在 F 中索引连续的位置 N [u]、N [u] + 1、…、N [u + 1] − 1 处。因此，对于任意节点 $v \in V$，N [v + 1] –N [v]可在 O(1) 时间内给出邻居 $|N(v)|$ 的数量，而邻域枚举需要 O($|N(v)|$) 时间；相比之下，矩阵表示执行邻居枚举和关联超边枚举查询均需要 O($|V|$) 时间。由于内存位置连续，CSR 表示还可以在 **Local-core** 算法中利用空间内存访问模式。

优化二（高效的核心校正与 LCCSAT）。我们设计了三种优化方法，以更高效地执行**核心校正**。

用于高效 E^+ 计算的超边索引：在核心校正（算法 4）的第 2 行中，对于每个满足 $e \in Incident(v)$ 的节点 $u \neq v$，我们检查 $h(n)_u \geq h(n)_v$。在核心校正的每次 while 循环（第 行）中，这一计算都会产生 $O(d(v)\,(v))$ 的开销。最坏情况下，while 循环的迭代次数为 $|N(v)|$，因此，低效的 E^+ 计算会使核心校正的总开销增加 $O(|N(v)|(d(v) + |N(v)|))$。通过为超边维护索引 E_e，我们将这一开销降至 $O(|N(v)|d(v))$。$E(n)_e$ 记录每条超边 $e \in E$ 的组成节点的 - 索引的最小值（$\min_{u \in e} h(n)_u$）。我们只遍历满足 $E(n)_e \geq h(n)_v$ 的关联超边，以计算 $E^+(v)$。存储所有超边的 E_e 需要 $O(|E|)$ 空间，构建这些索引需要 $O(\sum_{e \in E}|e|)$ 时间。不过，超边索引只在 Local-core 的每次迭代之前构建一次（算法 3，第 3 行）。此外，超边索引有助于高效计算 N^+，如下所示。

高效计算 N^+ 的动态规划：核心校正过程中的 while 循环计算 $k = h(n)_v, h(n)_v - 1, \ldots, \hat{}h(n)_v$；对于每个 $\hat{}h(n)_v \leq k \leq h(n)_v$，都会重新计算 $E^+(v)$和$N^+(v)$，直至返回 $\hat{}h(n)_v$ 作为输出。若不进行任何优化，在每次 while 循环迭代 k（第 1 行）中计算 $N^+(v)$的代价为 $O(d(v) + |N(v)|)$。由于 while 循环的迭代次数为（$h(n)_v - \hat{}h(n)_v$）$\leq |N(v)|$，低效的 N^+ 计算会使核心校正的总开销增加 $O(|N(v)|(d(v) + |N(v)|))$。

我们通过构建一个索引B，将此开销降至 $O(d(v) + |N(v)|)$，使得 $B[h(n)_v]$ 记录其超边索引为 $E(n)_e \geq h(n)_v$的关联超边集合；对于每个 $k < h(n)_v$，$B[k]$ 记录其超边索引为 $E(n)_e = k$的关联超边。

将 k处的 $E^+(v)$ 和 $N^+(v)$ 分别记为 $E^+(v,\ k)$ 和 $N^+(v,\ k)$。利用索引结构 B，我们可以采用以下动态规划范式，高效计算 $N^+(v,\ k)$。

$$N^+(v,k) = \begin{cases} \cup_e B[k] & k = h_v^{(n)} \\ N^+(v,k+1) \cup (\cup_e B[k]) & \hat{h}_v^{(n)} \leq k < h_v^{(n)} \end{cases} \quad (8)$$

我们不再于每个 $\hat{}h(n)_v \leq k \leq h(n)_v$ 处遍历所有关联超边，而只遍历 $B[k]$ 处的超边，以计算 $N^+(v,k)$。由于索引 $B[k]$ 彼此互斥，在整个核心校正过程中，每条关联超边至多遍历一次。B 的存储成本为 $O(h(n)_v + d(v))$，因为 B 中至多有 $h(n)_v$ 个键，且 B 中恰好存储了 $d(v)$ 条超边。得益于高效的 E^+ 和 N^+ 计算，核心校正的开销为 $O(|N(v)|d(v) + d(v) + |N(v)|)$，而非 $O(|N(v)|d(v) + |N(v)|^2)$。

高效计算 $LCCSAT$：找到第一条超边 $e \in Incident(v)$ 后，我们返回 真；将其添加到 $E^+(v)$ 会导致$|N^+(v)| \geq h(n)_v$。继续向 $E^+(v)$ 添加关联超边会进一步增大 $|N^+(v)|$，而不影响 $LCCSAT(k)$=真。

优化 III（加快收敛）。在第 8 行（算法 5）中，我们不再按照公式 (6) 计算 $h(n)_v$，而是通过将 H 算子中的一些操作数 $\hat{}h(n-1)_{u_i}$ 替换为 $h(n)_{u_i} \leq \hat{}h(n-1)_{u_i}$，来计算一个更小的值 $g(n)_v$（依据定理 3）。这能加快收敛，因为 $g(n)_v \leq h(n)_v$，降低少数几个参数可能会使 H() 的输出进一步减小，例如，H(1, 1, 2, 2) = 2，而 H(1, 1, 1, 2) = 1。计算 g_v不会影响正确性，因为新的交错序列（$h_v, g_v, \hat{}g_v$）同样

算法6 Local-core+Peel 用于（k,d）-核分解

输入： 超图 $H = (V, E)$
输出： H的所有非空(k,d)-核；$1 \leq k \leq k_{max}, 1 \leq d \leq d_{max}$

```
1: 初始化 ℂ = {C(k,d) ← ϕ  ∀k_min ≤ k ≤ k_max, d_min ≤ d ≤ d_max}
2: 计算基于邻域的核数 c(v)  ∀v ∈ V /* Local-core(OPT)*/
3: 对于所有 基于邻域的核数 k = 1, 2, ....., k_max = 最大值_{v∈V} c(v) 执行
4: 构造邻居数 k-核 H[V_k]: V_k = {v: c(v) ≥ k}
5:     对于所有 u ∈ V_k 执行
6:         初始化 B[deg_{V_k}(u)] ← B[deg_{V_k}(u)] ∪ {u}
7:     对于所有 d = 1, 2, ....., 最大值^{u∈V_k}(deg_{V_k}(u)) 执行
8:         当 B[d] ≠ ϕ 时执行
9:             从 B[d]中移除节点 v。
10:            C(k, d) = C(k, d) ∪{v}
11:            对于所有 u ∈ N_{V_k}(v) 执行
12:                如果 |N_{V_k\{v}}(u)| ≥ k 则
13:                    移动 u到 B[最大化 (deg_{V_k\{v}}(u) d)]
14:                else
15:                    将 u 移动到 B[d]
16:            V_k ← V_k \ {v}
17: 返回 ℂ
```

单调非增且下界为 0。这是因为 $h(n)_v \geq g(n)_v$，并且核心校正不会增大 $g(n)_v$，因此$g(n)_v \geq \hat{g}(n)_v$。因此，$(h(n)_v)$、$(g(n)_v)$ 和 $(\hat{g}(n)_v)$ 的极限与定理 3 中的相同：$\hat{}h(\infty)$。最后，由定理 4 可知，尽管进行了这项优化，仍有 $\hat{}h(\infty) = c(v)$ 成立。

优化 IV（减少冗余的 H 计算和 LCC- SAT 检查）。我们使用核数的局部下界 $LB(v)$（引理 1）来减少 h-索引计算和核心修正的次数。在某次迭代 n 中，如果 $\hat{}h(n)_v$ =LB(v)，则可确保 $c(v)$=LB(v)。因此，h-索引在后续迭代中不再降低；否则，$c(v) = \hat{}h(\infty)_v < \hat{}h(n)_v$ = LB(v)，这就产生了矛盾。因此，必有 $c(v) = \hat{}h(\infty) = \hat{}h(n)$。后续迭代中，我们无需计算 v的 h-索引和核心修正。

请注意，Liu 等人根据邻居 $u \in N(v)$ 的 $h(n)_v$ 和 $h(n)_u$ 是否收敛，来判断节点 v 的 H 计算是否冗余。这种方法不适用于我们的问题，因为即使节点 v 及其邻居的 h-索引已经收敛，节点 v 仍可能需要进行核心修正（例如，例 4）。

Local-core(Opt) 与 Peel 的效率比较。在 **Peel**（§3.1）的时间复杂度中，d_{nbr}和 d_{hpe}分别是 $|N(v)|$和 $d(v)$ 的上界，$\forall v \in V$。因此，**Peel**的复杂度可与 $O(\sum_{v \in V}|N(v)|d(v) + N(v)^2)$ 相比较，这与 **Local-core**（§3.3）执行一轮的复杂度相同。因此，**Local- core** 比 **Peel** 更慢。然而，我们的优化显著降低了其复杂度。例如，通过高效计算 E^+ 和 N^+ $\tau \sum_v d\ v\ |N\ v\ |+$，优化二将复杂度降低至 $O(\ (\ )\ (\ )\ (\ )\ +|N(\ )|)$ $O(\tau \sum_v d(\ )|N(\ )|)$ $O(\tau|V|d_{hpe}d_{nbr})$。因此，由于优化二，**Local-core(Opt)** 至少比 **Peel** 快 $\frac{d_{nbr}+d_{hpe}}{\tau d_{hpe}} = (\frac{d_{nbr}}{\tau d_{hpe}} + \frac{1}{\tau})$ 倍。在大规模真实世界超图中，一个节点通常比其度拥有更多邻居（例如表1中的 *aminer* 和 *dblp*），从而导致 $d_{nbr} > d_{hpe}$。优化 III 和 IV 进一步降低 τ，使比率达到 > 1，因此在我们的实验中，**Local-core(Opt)** 比 **Peel** 快得多。

Local-core 的并行化。我们遵循共享内存、数据并行编程范式，提出并行算法 **Local-core(P)**，进一步提高效率。该算法将节点划分为 T个分区，其中 T为线程数。每个

表1：数据集： $|V|$ **个节点，** $|E|$ **条超边，节点的** d（v）**度（平均值），超边的** $|e|$ **基数（平均值），** $|N(v)|$ **每个节点的邻居数（平均值）**

	超图	$\|V\|$	$\|E\|$	$d(v)$	$\|e\|$	$\|N(v)\|$
Syn.	*bin4U*	500	12424	99.4±8.5	4±0	225.3±15.5
	bin3U	500	16590	99.5±8	3±0	164.1±11.6
	pref3U	125329	250000	5.9±915.9	3±0	4.5±412.4
Real	*Enron*	4423	5734	6.8±32	5.2±5	25.3±44
	接触	242	12704	127±55.2	2.4±0.5	68.7±26.6
	国会	1718	83105	426.2±475.8	8.8±6.8	494.7±248.6
	dblp	1836596	2170260	4±11.6	3.4±1.8	9±21.4
	aminer	27850748	17120546	2.3±5	3.7±2.6	8.4±24.1

线程负责计算其所在分区中节点的核数。为改善负载均衡，我们采用最长处理时间优先调度方法 [32]，使不同线程所负责节点的邻居总数大致相同。与顺序算法 **Local-core(OPT)**（算法5）相比，**Local-core(P)** 主要有三处不同。

首先，在第 0 次迭代（第 3 行），每个线程都会并行初始化分配给自己的节点的超图 h-索引。对分配给不同线程的节点并发计算 $|N(v)|$ 和 $|N(u)|$，需要并发读取 CSR 表示。由于 CSR 表示在各次查询之间不会改变，这两种查询都会产生与其顺序执行版本相同的结果。**其次**，在后续迭代（$n > 0$）中，每个线程都会并行计算分配给自己的节点的 $h_{(n)v}$ 和校正后的值 $\hat{}h_{(n)v}$（第 6–11 行）。有趣的是，我们的算法仍会以正确的输出终止，因为我们此前的理论结果均不依赖于同一迭代中节点的特定计算顺序。计算顺序只会影响收敛所需的迭代次数。**最后**，所有线程都会各自维护一个核校正计数器，用于统计某一轮中已完成核心校正的分配节点数量。当所有计数器均报告 0 时，**Local-core(P)** 终止。

5 扩展至（邻域，度）核分解

我们通过同时考虑邻域和度约束，提出了一种新的超图核模型：**（邻域，度）核**，简称为（k，d）核。（k，d）核解决了这样的问题：如果某个节点属于一条规模很大的超边，其邻居数-k核可能会非常大。请注意，对于这样的节点，（k，d）核的 d 值会较小，因此可以将其与度和邻居数都更高的其他节点区分开来。我们展示了（k，d）核在扩散传播方面的优越性（§7.1）。

（k，d）核。给定整数 k、$d > 0$，（k，d）核记作 $H[V(k,d)] = (V(k,d)\ E[V(k,d)])$，是满足以下条件的极大子超图：在强诱导子超图 $H[V(k,d)]$ 中，每个节点 $u \in V(k,d)$ 至少有 k 个邻居，且度为 $\geq d$。

(k,d)-核分解很困难：**(1)** 不同于基于邻域的核分解会在不同核之间定义全序层级关系，(k,d)-核分解形成的是一个核格，只定义部分包含关系：$(k+i, d+j)$-核包含于 (k,d)-核 $\forall k, d > 0$; $i, j \geq 0$。**(2)** 设 k_{max} 和 d_{max} 分别表示通过基于邻域和基于度的核分解得到的最大核数。不同的 (k,d)-核最多可达 $O(k_{max} \cdot d_{max})$ 个。鉴于这些挑战，我们仍将设计用于 (k,d)-核分解的整体局部算法这一问题悬而未决。借鉴多层核分解中的类似概念 [28], ，我们提出一种局部与剥离混合方法。

用于 (k,d)-核分解的 Local-core+Peel 算法（算法6）。首先，我们将 (k,d)-核初始化为空（第 1 行），并使用 Local-core(OPT) 计算所有节点的基于邻域的核分解（第 2 行）。在每次外层for循环迭代 k 中（第 3 行），我们构造邻居数 k-核 $(H[V_k])$（第 4 行），并剥离 V_k 中的节点，以找出具有相同 k 但不同 d 的各种 (k,d)-核。为此，我们使用列表向量 B，根据节点在单元 $H[V_k]$ 中的度数，定义 V_k 的初始剥离顺序（第 5–6 行），这与 Peel 算法类似。我们按值 d 的升序遍历 B（第 7 行），并从当前单元 $B[d]$ 中剥离节点，直到 $B[d]$ 为空（第 8 行）。每当从 B 中移除一个节点（第 9 行），便将其分配到 (k,d)-核 $C(k,d)$（第 10 行），并将其在 $H[V_k]$ 中的邻居移至 B 中的相应单元（第 11–15 行）。将 V_k 中的所有节点分配到不同 d 对应的各种 (k,d)-核后，我们会在外层for循环的下一次迭代（第 2 行）中，对 V_{k+1} 执行相同步骤。

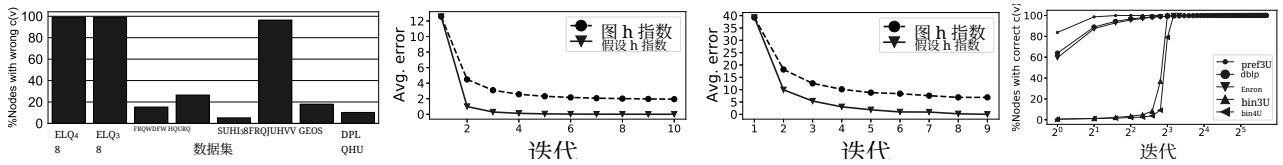
6 实证评估

我们在四个合成数据集和五个真实世界数据集（表1）上对算法的性能进行了实证评估。我们使用 GNU C++11 和 OpenMP API 3.1版实现算法。所有实验均在一台配备 128 个 AMD EPYC 32 核处理器和 256GB RAM 的服务器上进行。**我们的代码和数据集位于 [4]**。

数据集。在合成超图中，*bin4U* 和 *bin3U* 分别是使用先进的超图配置模型 [3]生成的 4-均匀和 3-均匀超图。*pref3U* 是使用超图优先连接模型 [5] 生成的 3-均匀超图，其参数为 $p = 0.5$，其中 p 表示新节点优先连接到超图中现有节点的概率。*pref3U* 中的节点度近似服从指数为 $= 2.2$的幂律分布。在真实世界超图中，*enron* 是一个电子邮件超图，其中每封电子邮件往来对应一条超边，用户为节点 [12]。我们从一个图中构建*contact*（学校）数据集，其中每个极大团视为一条超边，个体视为节点 [12]。在*congress* 数据集中，节点为国会议员，每条超边由一项法案的提案人和共同提案人（支持者）组成 [12]。在 *dblp*中，节点为作者，超边由 DBLP 中记录的一篇出版物的作者组成 [12]。类似地，*aminer* 由 Aminer 中记录的作者和出版物构成 [60]。

6.1 Local-core算法的有效性

实验 1：超图的创新性与重要性 h**索引**。我们通过展示一种不进行任何核心校正、直接改编图 h-索引（定义 2）的方法，即从 *[48, 52]* 运行局部算法可能会产生错误的超图核数，来证明所提出的超图 h-索引（定义 4）的创新性。图 4(a) 显示，对于**核心校正**（§ 3.3），若仅考虑图h-索引而不采用我们提出的方法，局部算法会在*bin4U*、*bin3U*和国会中为至少 90% 的节点生成错误的核数。在接触、*Enron*、*pref3U*、*DBLP*和*aminer*中，分别有至少 15%、26%、5%、17% 和 10% 的节点核数错误。由于*bin4U*、*bin3U*和国会中的节点具有相对较高的平均值（$|N(v)|$）（表1），这些数据集中存在更多相关邻居。相关邻居的错误 h-索引会引发多米诺效应：少数节点的 h值一旦出错，若不加以校正，就可能导致其所有邻居的 h值也出错，

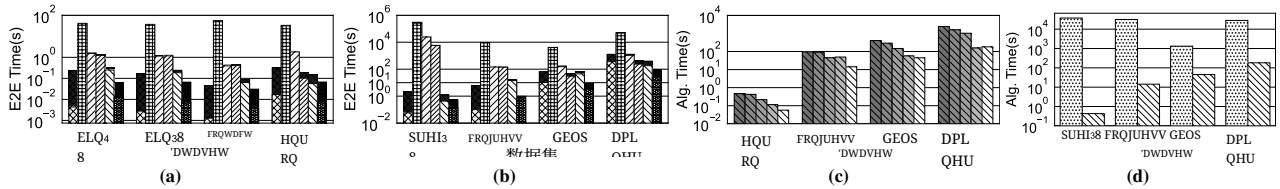


(a) 图索引的无效性 **(b) 超图和图的平均误差 *h*-Enron 上的索引** **(c) 超图和图的平均误差 *h*-bin4U 上的索引** **(d) Local-core 收敛性**

图 4：超图 *h*-索引和 Local-core 的有效性评估

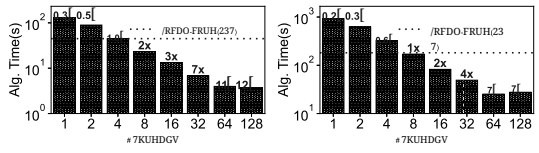


图 5：图 6 和图 7 的图例



(a) **(b)** **(c)** **(d)**

图 6：（a）-（b）我们的算法 Peel、E-Peel、Local-core(OPT) 和 Local-core(P) 使用 64 个线程时的端到端（E2E）运行时间，与基线方法 Clique-Graph- Local 和 Distance-2 Bipartite-Graph-Local 的运行时间对比。端到端（E2E）运行时间 =数据结构初始化时间（以每个柱形顶部的深黑色部分表示）+ 算法执行时间。（c）四种优化对 Local-core 算法执行时间的影响。此处，Local-core(OPT) = Local-core + 优化-I + II +III + IV。（d）Local-core+Peel 算法在（邻域，度）核分解中的执行时间，与 Local-core(OPT) 在邻域核分解中的执行时间对比。



（a）*dblp* （b）*aminer*

图 7：Local-core(P) 算法在更大规模数据集上的执行时间。与顺序 Local-core(OPT) 相比，Local-core(P) 最多可实现 12 倍加速。

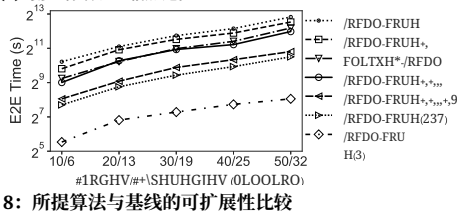


图 8：所提算法与基线的可扩展性比较

进而可能导致邻居的邻居的*h*值出错，如此等等。除非校正所有这类相关节点，否则最终几乎所有节点的核数都会出错。

我们还比较了图 *h*-索引和我们的超图 *h*-索引在每次迭代时核数估计的平均误差。这里，第 n 次迭代时的平均误差为 $= \sum_{u \in V}(h(n)(u) - core(u))/|V|$。图 4(b)-(c) 展示了在 *Enron* 和 *bin4U* 上每次迭代结束时产生的平均误差。初始化时，两个索引在同一数据集上的误差相同。对于这两个索引，某次迭代的平均误差均小于或等于前一次迭代的平均误差。然而，在迭代次数较多时，超图 *h*-索引产生的平均误差低于图 *h*-索引。终止时，尽管超图*h*-索引得到了正确的核数，图 *h*-索引仍存在非零平均误差。这些结果表明，在中间迭代阶段，超图 *h*-索引对核数的估计比图 *h*-索引更准确。我们在其他数据集上也观察到了类似趋势；不过，在相同迭代次数下，图 *h*-索引在 *bin4U* 上的平均误差高于在 *Enron*上的误差。这是因为，如前所述，*bin4U* 中的相关邻居数量多于 *Enron* 。

实验 2：Local-core 收敛性。图 4(d) 表明，随着我们的 **Local-core** 算法（§ 3.3）迭代次数增加，核数正确收敛的节点比例也随之提高。收敛所需的迭代次数取决于超图结构以及节点的计算顺序。由于 *pref3U*、*DBLP*和 *Enron* 中拥有较少相关邻居的节点更多，因此更多节点在第一次迭代后便得到正确的核数。在 *pref3U* 中，到第 2 次迭代时已有 98% 的节点收敛。这一观察结果还表明，可以提前终止 **Local-core** 算法，但代价是 *pref3U*上会有一部分结果不正确。

6.2 效率评估

我们评估所提算法的效率：**Peel**、**E-Peel**、**Local-core(OPT)**和 **local-core(P)** ，并与基线方法：**Clique-Graph- Local** 和 **Bipartite-Graph-Local**进行比较。基线方法 **Clique-Graph-Local** 和**Bipartite-Graph-Local** 分别采用超图的团图和二部图表示（§2.2），然后应用局部算法 [47, 48] 进行图核分解。从报告端到端运行时间的图 6(a)-(b) 中可以看出：(1) 在所有数据集上，**E-Peel** 都比**Peel** 更高效。**(2) 在所有数据集上，Local-core(OPT)** 都比**Peel** 更高效。**(3) 在所有数据集上，Local-core(P) 都是效率最高的算法**。**(4)** 基线方法 **Bipartite-Graph-Local** 在所有算法中效率最低。这是因为将超图转换为二部图会增加图节点和边的数量（§1），而在这个规模膨胀后的图上进行距离 2 核分解 [47] 会显著增加运行时间（§2.2）。另一方面，**Clique-Graph-Local**的端到端运行时间与**Local-core(OPT)** 相当（例如，在*contact* 和*con- gress*上略高效，而在*enron* 和*aminer*上效率较低）。这是因为转换为团图同样会扩大问题规模（§1），但在团图上进行基于图的 *h*索引计算 [48] 无需进行任何核心校正。不过，团图分解

表2：所有算法在*aminer*上的数据结构初始化时间

局部 -核(P)	局部 -核（OPT）	E-Peel	Peel	团 -G-局部	二分 -G-局部
66 s	510 秒	195 秒	198 秒	806 秒	793 秒

得到的分解与我们的分解不同；在某些应用中（如 §2.2、§7 所示），我们的分解比团图分解更理想。

各种算法需要不同的数据结构，其中包括超图的不同表示方式，例如，查找所有节点的邻居和关联超边（所有算法），创建二部图（**Bipartite-Graph-Local**）、团图（**Clique-Graph-Local**）和压缩稀疏行（**Bipartite-Graph-Local**、**Clique-Graph- Local**、**Local-core(OPT)**、**Local-core(P)**）。在端到端运行时间中（图 6(a)-(b)），我们将数据结构的初始化时间计入其中（以每个柱形顶部的深黑色阴影部分表示）。我们还在表2中单独报告了最大数据集*aminer* 的初始化时间。我们使用 OpenMP 和动态调度并行构建压缩稀疏行，并查找节点的邻居和关联超边 [25]。在*aminer*上，数据结构的并行初始化时间和**Local-core(P)** 的执行时间分别为 66 秒和 25 秒，因此端到端时间为 91 秒。

实验 3：E-Peel 的效率。如 §3.2 所述，**E-Peel** 相较于**Peel** 的加速比与 α 有关，该值是**E-Peel** 执行的 $\#|N(u)|$ 次查询与**Peel**执行的查询次数之比。在图 6(a)-(b) 中，**E-Peel**在*enron* 数据集上取得了最高的加速比（相较于**Peel**为 17 倍），因为该数据集上的 $\alpha = 0.35$ 最小。

在其余实验中（图 6(c)-(d)、图 7），我们比较在同一数据集上具有相同初始化时间的相似算法，因此比较这些算法的执行时间（ y轴上的“算法时间”）。

实验 4：Local-core 的效率及优化的影响。接下来，我们分析所提出的优化（§4）相对于未经任何优化的 Local-core 的效率。Local-core+I 在 Local-core 中加入了优化一；Local-core+I+III 在 Local-core+I 的基础上加入了优化三；Local-core+I+III+IV 在 Local-core+I+III 的基础上加入了优化二。最后，Local-core(OPT) 纳入了全部四种优化。图 6(c) 展示了 Local-core、Local-core+I、Local-core+I+III、Local-core+I+III+IV 和 Local-core(OPT) 在四个代表性数据集上的执行时间。在所有超图上，我们观察到，添加每项优化通常都会缩短执行时间。

实验 5：并行化的影响。我们通过改变线程数来测试 **Local-core(P)** 的并行性能（图 7）。增加 64–128 个线程，可使较大的*DBLP* 和*aminer* 数据集上的总执行时间最多缩短 7–12 倍。我们测量了不采用负载均衡（§4）的**Local-core(P)** 的执行时间，并发现负载均衡可使**Local-core(P)**在*aminer* 和 *DBLP*上的速度最多提升 1.8 倍。

实验 6：可扩展性。我们在具有幂律度分布（指数为 2.1）的 5-均匀超图上测试所提算法和基线算法（**Clique-Graph-Local**）的可扩展性，这些超图包含 1000 万至 5000 万个节点。图 8 显示，我们最高效的算法 **Local-core(P)**，以及顺序算法 **Local-core-I+ III+IV** 和 **Local-core(OPT)**，都优于基线算法 **Clique-Graph-Local**。我们的 **Local-core(P)** 也具有良好的可扩展性，在包含 5000 万个节点和 3200 万条超边的 5-均匀超图上，端到端运行时间仅为 248 秒。

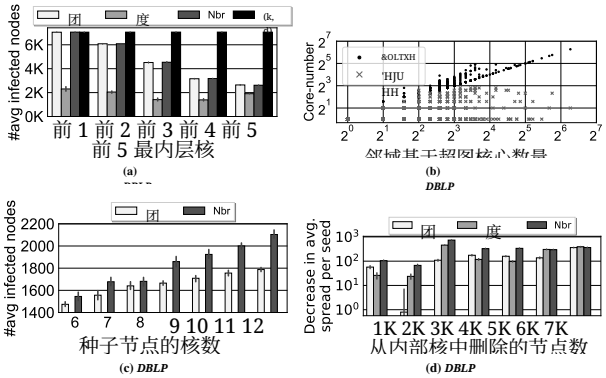


图 9：(a) 种子节点从最内层核选到外层核时，平均感染节点数会减少。(b) 邻域分解的核数与度，以及节点在团图分解中的核数。(c) 从外层核中选择种子节点时，基于邻域的分解优于基于团的方法。(d) 通过基于邻域的分解删除排名前 k 的节点时，每个种子节点对应的平均预期感染节点数降幅最大。

实验 7：Local-core 的效率+用于（邻域，度）-核分解的 Peel。图 6(d) 显示，在更大规模的数据集上，用于（k，d）-核分解的 **Local-core+Peel**算法比 **Local- core(OPT)** 耗时更长。这是意料之中的，因为可能的（k，d）-核数量为 $O(k_{max} \cdot d_{max})$，而基于邻域的核数量仅为 k_{max}(§5)。随着输出规模增大，算法 **Local-core+ Peel**除了运行**Local-core(OPT)**外，还会显式构造子超图 $H[V_k]$，并针对所有邻居数核数 k，根据度维度进行剥离 $H[V_k]$。

7 应用与案例研究

我们研究基于邻域的分解所识别的重要节点，相较于其他分解方法识别的重要节点的优势。

7.1 影响力传播与干预

我们考虑*SIR* 扩散过程 [41]：初始时，除一个称为种子节点的节点外，所有节点都处于易感状态。种子节点初始时处于感染状态。在每个时间步，每个受感染节点都会以 β的概率感染其易感邻居，然后变为免疫状态。节点一旦免疫，就不会再次感染。

内核包含有影响力的传播者（图9(a-c)）。对于每种分解方法，我们选取最内层的前5个核（沿 x 轴），并对每个核中的100个均匀随机选取的种子节点分别运行 SIR模型（β=0.3）。对于（k，d）-核，我们选取 5 k 值最高的内核，并针对每个 k，选取 d 值最大的种子节点。图9(a)展示了每个核中各个种子节点所对应的感染节点数的平均值。我们发现**(1)** 从内层核向外层核移动时，平均感染节点数通常会减少，这表明内核包含质量更高的种子节点。**(2)** 根据（k，d）-核分解选出的种子节点优于通过其他分解方法选出的种子节点，这说明我们的（k，d）-核分解能够产生质量最高的种子节点，从而最大化扩散。**(3)** 通过基于度的分解选出的种子节点，其传播范围最小。**(4)** 基于邻域和团图分解得到的种子节点质量相近，原因在于这两种分解的内核在组成节点方面相似度较高，而它们的外层核在 *DBLP* 中则大不相同。更具体地说，

图9(b)通过比较不同方法得出的1000个随机选取节点的核数，突显了这种差异。我们注意到，对于内核中的节点，基于邻域和团图分解得到的核数呈线性相关；但随着我们考察外层核中的节点，这种线性相关性逐渐减弱。我们没有观察到基于邻域的方法与基于度的方法所得到的核数之间存在相关性。**(5)** 图9(c)表明，当从相对外层的核中选取种子节点时，基于邻域的方法优于团图方法。

删除内核，以最大限度地干预传播（图 9(d)）。我们提出了一种干预策略来阻断扩散，这对于遏制流行病学中传染病的传播、限制错误信息的传播，或阻止营销中的竞争性营销活动至关重要。根据特定分解方法（邻域、度或团）划分出的内核中的节点，被视为比外层核中的节点更重要。我们按照特定分解方法，选取排名最高的k个重要节点（位于x轴上），然后删除这些节点及其关联超边。为公平比较不同的分解方法，我们对所有方法采用相同的种子节点集合——这些种子节点是根据三种分解方法，从已删除节点集的并集之外选取的。为了确定这种删除策略的效果，我们在y轴上报告每个种子节点的平均传播范围降幅（即感染节点的预期数量降幅）。我们观察到，删除的节点不超过 6K 时，通过基于邻域的分解删除重要节点，相较于通过其他分解方法（度和团）进行相同操作，总能使传播范围降幅最大。超过 6K 个节点后，每种分解方法都会根据自身的分解结果删除大量重要节点，而且种子节点的质量（这些种子节点来自已删除区域之外）也会下降；因此，所有方法导致的传播范围降幅相近。这一结果表明，我们的基于邻域的分解方法能够为删除数量有限的节点提供最优的重要节点排序，同时最大限度地干预传播。

7.2 最稠密子超图发现

度最稠密子图是给定图中所有子图里节点平均度最大的子图 [16, 23, 31],，它可能对应社交网络中的社群 [20] 和回音室 [42]，也可能对应对刺激作出反应的脑区 [45]。遵循同一原则，我们根据超图中节点的邻居数量，定义一种新的最稠密子超图概念，称为体积最稠密子超图。超图 $H=$ 中节点子集 $S \subseteq V$ 的**体积密度** $\rho^N[S]$ （V, E）定义为诱导子超图 $H[S]$ 中所有节点 $u \in S$ 的邻域大小之和与 $H[S]$中节点数量的比值。

$$\rho^N[S]=\frac{\sum_{u\in S}|N_S(u)|}{|S|} \tag{9}$$

体积最稠密子超图 是所有子超图中体积密度最大的子超图。

近似算法。受 Charikar [16], 的启发，我们的方法遵循剥离范式：每一轮中，我们移除当前子超图中邻居数最少的节点。具体而言，我们按照基于邻域的核数升序排列节点，该核数由任意一种基于邻域的核分解算法得到（例如 .g.，**Peel**，**E-Peel**，**Local-core（OPT** **),** 或 **Local-core(P)**）。我们按此顺序剥除节点。在核数相同的节点中，当前子超图中邻居数量最少的节点会先被剥除。最后，我们返回体积密度最大的子超图。

定理 6. 我们的体积最稠密子超图发现算法返回($d_{pair}($ $d_{card}-2)+2$)-近似最稠密子超图，前提是输入超图中超边的最大基数为 d_{card}，且任意一对节点之间的超边数最大为 d_{pair}。

证明。设一个体积最稠密子超图为 $H^*=(S^*, E[S^*])$。对于所有 $v \in H^*$，根据 H^*的最优性：

$$\rho^N[H^*]=\frac{\sum_{u\in S^*}|N_{S^*}(u)|}{|S^*|} \geq \frac{\sum_{u\in S^*\setminus\{v\}}|N_{S^*\setminus\{v\}}(u)|}{|S^*|-1} \tag{10}$$

接下来，我们验证：

$$\sum_{u\in S^*\setminus\{v\}}|N_{S^*\setminus\{v\}}(u)| \geq \sum_{u\in S^*}|N_{S^*}(u)|-|N_{S^*}(v)|-\left(d_{pair}(d_{card}-2)+1\right)|N_{S^*}(v)| \tag{11}$$

其中，d_{card}是输入超图中任意超边的最大基数。右侧减去 $|N_{S^*}(v)|$，因为 v已从S^*中移除。我们还减去 $\left(d_{pair}(d_{card}-2)+1\right)|N_{S^*}(v)|$，因为删除 v后，所有涉及 v的超边都会被移除。因此，$|N_{S^*}(v)|$中每个顶点的邻域大小最多可减少 $d_{pair}(d_{card}-2)+1$。结合不等式 10 和 11，我们可以对所有 $v \in H^*$推导出：

$$|N_{S^*}(v)| \geq \frac{\rho^N[H^*]}{d_{pair}(d_{card}-2)+2} \tag{12}$$

我们证明，当节点 $u \in S^*$ 被算法移除时，当前子超图的体积密度必定至少为最优值的 $\frac{1}{d_{pair}(d_{card}-2)+2}$。考虑满足 $S^* \subseteq S_i$但不满足 $S^* \nsubseteq S_{i+1}$的那次迭代。根据算法的贪心性，对所有 $w \in S_i$，以下关系成立：$|N_{S_i}(w)| \geq |N_{S_i}(u)| \geq |N_{S^*}(u)|$ 第二个不等式成立是因为 $S^* \subseteq S_i$。因此，

$$\sum_{w\in S_i}|N_{S_i}(w)| \geq |S_i||N_{S^*}(u)| \implies \frac{\sum_{w\in S_i}|N_{S_i}(w)|}{|S_i|} \geq |N_{S^*}(u)| \tag{13}$$

代入不等式 12，得到：

$$\rho^N[S_i] \geq \frac{\rho^N[H^*]}{d_{pair}(d_{card}-2)+2} \tag{14}$$

□

请注意，$d_{pair}=2$ 如果超图是图，那么我们的结果为最稠密子图发现提供了 2-近似保证 [16]。

案例研究：Meetup 数据集。我们从纳什维尔 Meetup 数据集中提取所有包含 <100 名参与者的事件 [7]。提取出的超图包含 24115 个节点（参与者）和 11027 条超边（事件），这些事件由各个兴趣小组组织。我们计算并分析了团图的体积最稠密子超图、度最稠密子超图 [36],以及度最稠密子图 [16]（表 3）。**(1)** 度最稠密子超图包含一些随意、频繁的小型聚会，参与者较少，且均来自同一个社交小组（*Bellevue Meetup:* 结识新朋友）。**(2)** 然而，团图的度最稠密子图和 **),**体积最稠密子超图包含涉及多个技术小组、围绕多样技术主题组织聚会的事件。

表 3：体积最稠密和度最稠密子超图、团图的度最稠密图：Meetup 数据集 [7]

	体积最稠密子超图	团图的度最稠密子图	度最稠密子超图
事件数	**27**	17	26
体积密度	**116.9**	0.4（投影到子超图）	9.3
示例事件	身份与访问 控制全景 在 .NET 中；网页抓取 使用 Python；监管环境。 围绕区块链	与.....一起实地考察 家谱学会； 每月会议：公民 科技；纳什维尔 (Nv) PHP 用户组	问答之夜ⓐ Plantation 酒吧； 问答之夜ⓐ Three Stones 酒吧； 晚餐 a◯道尔顿
组织群组	Nv .NET 用户组； Nv. 数据科学； Nv. 区块链用户组	州和地方政府开发 网络；开发者启动平台； Nv. PHP 用户组	贝尔维尤聚会： 结识新朋友

(3) 尽管能找到技术主题相关的事件，体积最稠密子超图找到的事件集合与团图的度最稠密子图找到的不同。这是因为，从团图中提取这些事件所用的度密度准则不同于体积密度。团图的度最稠密子图虽然具有较高的度密度（100.7），但投影为子超图（将团投影为超边）后，体积密度较低（0.4）。我们还发现，这些技术类事件的受欢迎程度较低，平均有 5 位参与者，而体积最稠密子超图找到的事件平均有 78 位参与者。

8 相关工作

近年来，人们对超图数据管理的兴趣日益浓厚 [24, 39, 43, 57, 59, 65]。由于我们专注于核分解，因此请参阅近期的综述 [11, 21, 44, 61]，了解相关的总体介绍。

超图中的核分解。关于超图核分解的研究相对较少。Ramadan 等人 [53] 提出了一种剥离算法，用于寻找超图中基于最大度的 k-核。 [38, 57] 讨论了基于剥离方法的基于度的超图核计算并行实现。Sun 等人 [58] 提出了一种全动态近似算法，用于维护无权超图的近似基于度的核数。Gabert 等人 [27] 研究了动态超图中的基于度的核维护，并提出了一种并行（共享内存）局部算法。这些工作均未探讨我们提出的基于邻域的超图核分解，它不同于基于度的超图核计算（§1）。现有的基于度的核分解方法难以直接改造为基于邻域的超图核计算方法（§3）。

图中的核分解。图核分解的线性时间剥离算法由 Batagelj 和 Za- veršnik 提出。图核分解也已在基于磁盘、分布式、并行和流式场景中，以及各种类型的图（例如加权图、有向图、时序图、不确定图和多层网络）上得到研究。图中的高阶核（例如（k，h）核、三角形核、 h团核、模式核）以及属性网络中更复杂的核（例如基于元路径的核和（k，r）核）也已被提出。在 §2.2 中，我们指出，现有的图核分解方法难以直接适用于基于邻域的超图核分解。我们还说明，局部方法——一种高效的图核分解方法——用于基于邻域的超图核分解时，会产生错误的核数（§3.3，§6）。

9 结论

我们提出了超图的邻域凝聚核分解，该方法具有唯一性和核包含性等理想性质。随后，我们提出了三种超图核分解算法：Peel、E-Peel，以及新颖的 Local-core。对合成超图和真实世界超图进行的实证评估表明，经过优化并采用并行实现的新颖 Local-core，在所有提出的算法和基线算法中效率最高。与基于度和团图的分解相比，我们提出的分解在干预扩散方面更有效。案例研究表明，与基于度和团图分解得到的最稠密子超图相比，我们新颖的体积最稠密子超图捕捉到的聚会事件各不相同。最后，我们通过同时考虑邻域和度约束，提出了一种新的超图核模型——（邻域，度）核；设计了相应的分解算法 Local-core+Peel；并展示了它在扩散传播方面的优势。未来，我们将为（邻域，度）核分解设计高效算法。

参考文献

[1] Mohammed Ali Al-garadi, Kasturi Dewi Varathan 和 Sri Devi Ravana。2017。使用交互加权K核分解方法识别在线社交网络中有影响力的传播者。*Physica A: Statistical Mechanics and its Applications* 468（2017），278–288。[2] J. Ignacio Alvarez-Hamelin、Luca Dall'Asta、Alain Barrat 和 Alessandro Vespignani。2005。使用k核分解对大规模网络进行指纹识别与可视化。载于神经信息处理系统进展（*NeurIPS*）。MIT出版社，41–50。[3] Naheed Anjum Arafat、Debabrota Basu、Laurent Decreusefond 和 Stéphane Bressan。2020。具有指定度序列和维数序列的超图的构造与随机生成。载于数据库与专家系统应用（*DEXA*）（计算机科学讲义），第12392卷。施普林格，130–145。[4] Naheed Anjum Arafat、Arijit Khan、Arpit Kumar Rai 和 Bishwamittra Ghosh。2022。我们的代码和数据集。https://github.com/toggled/vldbsubmission/。[5] Chen Avin、Zvi Lotker、Yinon Nahum 和 David Peleg。2019。随机优先连接超图。载于*IEEE/ACM*社会网络分析与挖掘国际会议（*ASONAM*）。398–405。[6] Mohammad A. Bahmanian 和 Mateja Sajna。2015。超图中的连通与分离。图论与图的应用2，2（2015），5。[7] Stephen Bailey。2017。纳什维尔聚会网络。https://www.kaggle.com/datasets/stkbailey/nashville-meetup。[8] Robert G. Bartle。1976。实分析要素。威利父子公司。[9] Vladimir Batagelj、Andrej Mrvar 和 Matjaž Zaveršnik。1999。用于大型图可视化的划分方法。载于图绘制（计算机科学讲义），第1731卷。[10] Vladimir Batagelj 和 Matjaž Zaveršnik。2011。用于确定社交网络中（广义）核心群组的快速算法。数据分析与分类进展5，2（2011），129–145。[11] Federico Battiston、Giulia Cencetti、Iacopo Iacopini、Vito Latora、Maxime Lucas、Alice Patania、Jean-Gabriel Young 和 Giovanni Petri。2020。超越成对交互的网络：结构与动力学。物理学报告874（2020），1–92。[12] Austin R. Benson、Rediet Abebe、Michael T. Schaub、Ali Jadbabaie 和 Jon Kleinberg。2018。单纯复形闭包与高阶链接预测。美国国家科学院院刊115，48（2018），E11221–E11230。[13] Claude Berge。1989。超图——有限集组合学。北荷兰数学丛书，第45卷。北荷兰出版社。[14] Francesco Bonchi、Francesco Gullo、Andreas Kaltenbrunner 和 Yana Volkovich。2014。不确定图核分解。载于*ACM SIGKDD*知识发现与数据挖掘国际会议。1316–1325。[15] Francesco Bonchi、Arijit Khan 和 Lorenzo Severini。2019。距离广义核分解。载于国际数据管理会议（*SIGMOD*）。ACM，1006–1023。[16] Moses Charikar。2000。用于在图中查找稠密组件的贪心近似算法。载于组合优化近似算法：第三届国际研讨会（计算机科学讲义），第1913卷。施普林格，84–95。[17] James Cheng、Yiping Ke、Shumo Chu 和 M. Tamer Özsu。2011。在大规模网络中进行高效核分解。载于*IEEE* 国际数据工程会议（*ICDE*）。51–62。

[18] Megan Dewar、Kirill Ternovsky、Benjamin Reiniger、John Proos、Pawel Pralat、Xavier Pérez-Giménez 和 John Healy。2018。非均匀随机超图中的子超图。Internet Math. 2018（2018）。[19] Laxman Dhulipala、Guy Blelloch 和 Julian Shun。2017。Julienne：一种使用工作高效分桶的并行图算法框架。载于ACM算法与体系结构并行研讨会。293–304。[20] Yon Dourisboure、Filippo Geraci 和 Marco Pellegrini。2009。网络图中稠密隐式社区的提取与分类。ACM网络汇刊3，2（2009），1–36。[21] Tina Eliassi-Rad、Vito Latora、Martin Rosvall 和 Ingo Scholtes。2021。高阶图模型：从理论基础到机器学习（Dagstuhl研讨会21352）。Dagstuhl报告11，7（2021），139–178。[22] Yixiang Fang、Yixing Yang、Wenjie Zhang、Xuemin Lin 和 Xin Cao。2020。在大型异构信息网络中进行有效且高效的社区搜索。Proc. VLDB Endow. 13，6（2020），854–867。[23] Yixiang Fang、Kaiqiang Yu、Reynold Cheng、Laks V. S. Lakshmanan 和 Xuemin Lin。2019。用于最稠密子图发现的高效算法。Proc. VLDB Endow. 12，11（2019），1719–1732。[24] Pit Fender 和 Guido Moerkotte。2013。Counter strike：面向超图的通用自顶向下连接枚举。Proc. VLDB Endow. 6，14（2013），1822–1833。[25] Andrew Finlayson。 [无日期。]OpenMP调度。http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/OpenMP_动态_Scheduling.pdf。[26] Christoph Flamm、Bärbel M.R. Stadler 和 Peter F. Stadler。2015。广义拓扑：超图、化学反应与生物进化。载于《数学化学与应用进展》。Bentham Science，300–328。[27] Kasimir Gabert、Ali Pinar 和 Ümit V. Çatalyürek。2021。动态图与超图上的共享内存可扩展k核维护。载于IEEE国际并行与分布式处理研讨会。998–1007。[28] Edoardo Galimberti、Francesco Bonchi、Francesco Gullo 和 Tommaso Lanciano。2020。多层网络中的核分解：理论、算法与应用。ACM Trans. Knowl. Discov. Data 14，1（2020），11:1–11:40。[29] Edoardo Galimberti、Martino Ciaperoni、Alain Barrat、Francesco Bonchi、Ciro Cattuto 和 Francesco Gullo。2021。时序网络的跨度核分解：算法与应用。ACM Trans. Knowl. Discov. Data 15，1（2021），2:1–2:44。[30] Thomas Gaudelet、Noël Malod-Dognin 和 Natasa Przulj。2018。高阶分子组织是生物功能的来源。生物信息学34，17（2018），i944–i953。[31] Andrew V. Goldberg。1984。寻找最大密度子图。加州大学伯克利分校。[32] Ronald L. Graham。1969。多处理时序异常的界限。SIAM应用数学杂志17，2（1969），416–429。[33] Ronald L. Graham、Martin Grötschel 和 László Lovász（编）。1996。《组合学手册》（第2卷）。MIT出版社。[34] Yi Han、Bin Zhou、Jian Pei 和 Yan Jia。2009。理解合著网络中协作的重要性：一种支持度分析方法。载于SIAM国际数据挖掘会议（SDM）。1112–1123。[35] Jorge E. Hirsch。2005。一种量化个人科研产出的索引。《美国国家科学院院刊》102，46（2005），16569–16572。[36] Shuguang Hu、Xiaowei Wu 和 T.-H. Hubert Chan。2017。在演化超图中高效维护最密子集。载于ACM国际信息与知识管理会议（CIKM）。929–938。[37] Jin Huang、Rui Zhang 和 Jeffrey Xu Yu。2015。可扩展的超图学习与处理。载于IEEE国际数据挖掘会议（ICDM）。775–780。[38] Jiayang Jiang、Michael Mitzenmacher 和 Justin Thaler。2017。并行剥离算法。ACM并行计算汇刊3，1（2017），1–27。[39] Igor Kabiljo、Brian Karrer、Mayank Pundir、Sergey Pupyrev、Alon Shalita、Yaroslav Akhremtsev 和 Alessandro Presta。2017。社交哈希划分器：一种可扩展的分布式超图划分器。Proc. VLDB Endow. 10，11（2017），1418–1429。[40] Wissam Khaouid、Marina Barsky、Venkatesh Srinivasan 和 Alex Thomo。2015。单台个人计算机上的大型网络k核分解。Proc. VLDB Endow. 9，1（2015），13–23。[41] Maksim Kitsak、Lazaros K. Gallos、Shlomo Havlin、Fredrik Liljeros、Lev Muchnik、H. Eugene Stanley 和 Hernán A. Makse。2010。识别复杂网络中有影响力的传播者。《自然·物理》6，11（2010），888–893。[42] Laks V.S. Lakshmanan。2022。探索应对过滤气泡与错误信息之道。载于国际数据管理会议（SIGMOD）。ACM，2。[43] Geon Lee、Jihoon Ko 和 Kijung Shin。2020。超图模体：概念、算法与发现。Proc. VLDB Endow. 13，11（2020），2256–2269。[44] Geon Lee、Jaemin Yoo 和 Kijung Shin。2022。真实世界超图挖掘：模式、工具与生成器。载于第31届ACM国际信息与知识管理会议（CIKM）论文集。5144–5147。

[45] Robert Legenstein、Wolfgang Maass、Christos H. Papadimitriou 和 Santosh S. Vempala。2018。长期记忆与最稠密 k 子图问题。载于《理论计算机科学创新会议》（ITCS）（莱布尼茨国际信息学会议录（LIPIcs）），第 94 卷。Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik，57:1–57:15。[46] Xuankun Liao、Qing Liu、Jiaxin Jiang、Xin Huang、Jianliang Xu 和 Byron Choi。2022。大型有向图上的分布式 d 核分解。Proc. VLDB Endow. 15, 8（2022），1546–1558。[47] Qing Liu、Xuliang Zhu、Xin Huang 和 Jianliang Xu。2021。大型动态图上的距离广义核分解局部算法。Proc. VLDB Endow. 14, 9（2021），1531–1543。[48] Linyuan Lü、Tao Zhou、Qian-Ming Zhang 和 H. Eugene Stanley。2016。网络节点的 h 指数及其与度和核度的关系。《自然·通讯》7, 1（2016），1–7。[49] Fragkiskos D. Malliaros、Christos Giatsidis、Apostolos N. Papadopoulos 和 Michalis Vazirgiannis。2020。网络的核分解：理论、算法与应用。《VLDB 杂志》29, 1（2020）。[50] Fragkiskos D. Malliaros、Maria-Evgenia G. Rossi 和 Michalis Vazirgiannis。2016。在复杂网络中定位影响力节点。《科学报告》6, 19307（2016）。[51] Irene Malvestio、Alessio Cardillo 和 Naoki Masuda。2020。复杂网络中 k 核与社群结构的相互作用。《科学报告》10, 14702（2020）。[52] Alberto Montresor、Francesco De Pellegrini 和 Daniele Miorandi。2012。分布式 k 核分解。《IEEE 并行与分布式系统汇刊》24, 2（2012），288–300。[53] Emad Ramadan、Arijit Tarafdar 和 Alex Pothen。2004。酵母蛋白质复合物网络的超图模型。载于《国际并行与分布式处理研讨会》。[54] Ahmet Erdem Saríyüce、Buˇgra Gedik、Gabriela Jacques-Silva、Kun-Lung Wu 和 Ümit V. Çatalyürek。2013。k 核分解的流式算法。Proc. VLDB Endow. 6, 6（2013），433–444。[55] Ahmet Erdem Sariyüce、C. Seshadhri 和 Ali Pinar。2018。层次化稠密子图发现的局部算法。Proc. VLDB Endow. 12, 1（2018），43–56。[56] Ahmet Erdem Sariyüce、C. Seshadhri、Ali Pinar 和 Ümit V. Çatalyürek。2015。使用核分解发现稠密子图的层次结构。载于《万维网国际会议》（WWW）。ACM。[57] Julian Shun。2020。实用并行超图算法。载于 ACM SIGPLAN 并行编程原理与实践研讨会。232–249。[58] Bintao Sun、T.-H. Hubert Chan 和 Mauro Sozio。2020。超图上的全动态近似 k 核分解。ACM Trans. Knowl. Discov. Data 14, 4 2020 39 1 39 21 ( ), : – : . [59] Justin Sybrandt、Ruslan Shaydulin 和 Ilya Safro。2022。基于嵌入的超图划分。《IEEE 知识与数据工程汇刊》34, 6（2022），2771–2782。[60] Jie Tang、Jing Zhang、Limin Yao、Juanzi Li、Li Zhang 和 Zhong Su。2008。ArnetMiner：学术社交网络的抽取与挖掘。载于 ACM SIGKDD 知识发现与数据挖掘国际会议。990–998。[61] Leo Torres、Ann Sizemore Blevins、Danielle S. Bassett 和 Tina Eliassi-Rad。2021。复杂系统表示的缘由、方式与时机。SIAM Rev. 63, 3（2021），435–485。[62] Charalampos E. Tsourakakis。2015。k 团最稠密子图问题。载于《万维网国际会议》（WWW）。ACM，1122–1132。[63] Thierry Vialar。2017。数学手册。BoD-Books on Demand。[64] Jia Wang 和 James Cheng。2012。大规模网络中的桁架分解。Proc. VLDB Endow. 5, 9（2012），812–823。[65] Joyce Jiyoung Whang、Rundong Du、Sangwon Jung、Geon Lee、Barry L. Drake、Qingqing Liu、Seonggoo Kang 和 Haesun Park。2020。MEGA：超图的多视图半监督聚类。Proc. VLDB Endow. 13, 5（2020），698–711。[66] Xin Xia、Hongzhi Yin、Junliang Yu、Qinyong Wang、Lizhen Cui 和 Xiangliang Zhang。2021。用于基于会话的推荐的自监督超图卷积网络。载于 AAAI 人工智能大会。4503–4511。[67] Jaewon Yang 和 Jure Leskovec。2015。基于真实标注定义和评估网络社群。《知识与信息系统》42, 1（2015），181–213。[68] Fan Zhang、Ying Zhang、Lu Qin、Wenjie Zhang 和 Xuemin Lin。2017。当参与度遇上相似性：社交网络上的高效 (k, r)-核计算。Proc. VLDB Endow. 10, 10（2017），998–1009。[69] Yang Zhang 和 Srinivasan Parthasarathy。2012。提取、分析并可视化网络中的三角形 k 核模体。载于 IEEE 国际数据工程会议（ICDE）。1049–1060。[70] Wei Zhou、Hong Huang、Qiang-Sheng Hua、Dongxiao Yu、Hai Jin 和 Xiaoming Fu。2021。加权图中的核分解与维护。《万维网》24, 2（2021），541–561。