FREIGHT：快速流式超图划分

KamalEyubov 海德堡大学，
德国**Marcelo Fonseca**
Faraj 海德堡大学，德国
ChristianSchulz 海德堡大
学，德国

arXiv:2302.06259v1 [cs.DS] 13 Feb 2023

摘要

将（超）图的顶点划分为 k 大小大致均衡的块，使块间的（超）边尽可能少，这是大规模分布式处理中的一项关键问题。当前，使用较少计算资源划分大型（超）图的一种趋势是采用流式算法。在这项工作中，我们提出了 FREIGHT：一种快速流式超图划分算法，它是对广为人知的基于图的算法 Fennel 的改编。通过使用高效的数据结构，我们使 FREIGHT 的总体运行时间与超图的引脚数呈线性关系，并使内存消耗与网和块的数量呈线性关系。我们的大量实验结果展示了 FREIGHT 作为一种高效且有效的流式超图划分解决方案所展现出的良好性能。我们的算法在运行时间方面与哈希算法具有竞争力：在四分之三的实例中，观察到的最大差异为四倍。尤为重要的是，我们的研究结果表明，在割网和连通性度量方面，FREIGHT 优于所有现有的（带缓冲区的）流式算法，甚至优于内存算法 HYPE。这表明，我们提出的算法是一种很有前景的超图划分工具，可应对大规模动态数据处理所带来的挑战。

2012年ACM学科分类 计算理论 → 流式、次线性和近线性时间算法；计算理论 → 图算法分析

关键词和短语 超图划分、图划分、边划分、流式处理

1 引言

图在自然界中无处不在，可用于表示各种各样的现象，例如道路网络、数据库中的依赖关系、分布式算法中的通信、社交网络中的交互等。然而，当实体之间的交互不一定是成对关系时，使用超图建模更为恰当，因为超图能够捕捉高阶交互 [25]。随着数据的海量增长，在分布式系统和数据库上处理大规模（超）图已成为众多应用的必要需求。在并行处理（超）图时，k 处理器各自处理（超）图中不同的部分，并通过消息传递相互通信。为了提高并行处理的效率，一项重要的预处理步骤是将（超）图的顶点划分为 k 大小大致均衡的块，使得块之间只有少量（超）边相连。（超）图划分是NP难问题 [17]，且对于一般的（超）图，不存在具有常数近似比的近似算法 [9]。因此，实践中通常采用启发式方法。当前，快速划分大型（超）图并降低计算资源消耗的一种趋势是采用流式算法 [41, 5, 22, 14, 15, 28, 20, 3, 40]。

文献中最常见的流式方法是单遍模型 [1]，：顶点逐个到达，并携带其（超）边，随后必须被永久分配到各个块中。在图领域，大多数算法要么速度非常快，但

完全不考虑解质量（例如 Hashing [39]），要么虽然仍然很快，但速度慢得多，而且比随机分配能算出显著更好的解（例如 Fennel [41]）。最近，流式多分区算法 [15] 弥合了这两类算法之间的差距：其速度最高可比 Fennel 快两个数量级，而割边数量平均只多 5%。在超图领域，也存在类似的差距，至今尚未弥合。具体而言，一方面有同样简单的、基于 Hashing的算法，另一方面则有更复杂、开销更大的算法 [3, 40]。

在本文中，我们提出了 FREIGHT：一种快速流式超图划分算法，可针对割网和连通性指标进行优化。通过采用高效的数据结构，我们使 FREIGHT 的总体运行时间与超图的引脚数呈线性关系，并使内存消耗与网和块的数量呈线性关系。我们提出的算法展现出显著的效率，其运行时间与 Hashing 算法相当，并且在四分之三的实例中，最大差异仅为四。重要的是，我们的研究确立了 FREIGHT 相较于当前所有（带缓冲的）流式算法，甚至内存算法 HYPE的优越性，无论是在割网度量还是连通性度量方面均是如此。这表明，在大规模且持续变化的数据处理环境中，我们的算法有望成为划分超图的宝贵工具。

2 预备知识

2.1 基本概念

超图与图。设 $H = (V = \{0, \ldots, n-1\}\ E)$ 是一个没有重边或自环的无向超图，其中有 $n = |V|$ 个顶点和 $m = |E|$ 条超边（或网）。网定义为 V 的一个子集。组成网的顶点称为引脚。若 $v \in e$，则顶点 $v \in V$ 与网 $e \in E$ 关联。设 c: $V \to \mathbb{R}_{\geq 0}$ 为顶点权重函数，$\omega : E \to \mathbb{R}_{>0}$ 为网权重函数。我们将 c 和 ω 函数推广到集合，使得 $c(V') = \sum_{v \in V'} c(v)$ 且 $\omega(E') = \sum_{e \in E'} \omega(e)$。设 $I(v)$ 为与 v 关联的网集合，设 $d(v) := |I(v)|$ 为 v 的度，设 $d_w(v) := w(I(v))$ 为 v 的加权度，设 Δ 为 H 的最大度。我们将记号 $d(.)$ 和 $d_w(.)$ 推广到集合，使得 $d(V') = \sum_{v \in V'} d(v)$ 且 $d_w(V') = \sum_{v \in V'} d_w(v)$。如果两个顶点都与同一个网关联，则它们相邻。设网 e 中引脚 $|e|$ 的数量为 e 的大小，设 $\xi = \max_{e \in E} \{|e|\}$ 为 H 中网的最大大小。

设 $G = (V = \{0, \ldots, n-1\}\ E)$ 是一个没有重边或自环的无向图，且满足 $n = |V|$，$m = |E|$。设 $c : V \to \mathbb{R}_{\geq 0}$ 是顶点权重函数，设$\omega : E \to \mathbb{R}_{>0}$ 是边权重函数。我们将 c 和 ω 函数推广到集合，使得 $c(V') = \sum_{v \in V'} c(v)$ 且 $\omega(E') = \sum_{e \in E'} \omega(e)$。令 $N(v) = \{u : \{v, u\} \in E\}$ 表示 v 的邻居。若 $V' \subseteq V$ 且 $E' \subseteq E \cap (V' \times V')$，则称图 $S = (V', E')$ 是图 $G = (V, E)$ 的子图。当 $E' = E \cap (V' \times V')$ 时，S 是一个诱导子图。令 $d(v)$ 表示顶点 v 的度，令 Δ 表示 G 的最大度。

划分。（超）图划分 问题是将（超）图的每个顶点恰好分配到 k 个不同的块 中，并满足平衡约束，以最小化块之间（超）边的权重，即边割（分别为割网）。更准确地说，它将 V划分为 k 个块 V_1、$\ldots$、V_k（即 $V_1 \cup \cdots \cup V_k = V$ 和 $V_i \cap V_j = \emptyset$，其中 $i \neq j$），这称为该（超）图的 k-划分 。 k划分的边割（分别为割网）是割边 （分别为割网）的总权重，

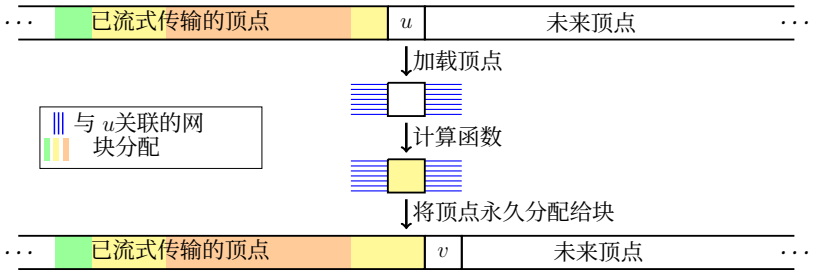


图 1 超图划分流式算法的典型布局。

也就是说，是跨越不同块的边（分别为网）。更正式地说，令边割（分别为割网）为 $\sum_{i<j}\omega(E')$，其中 $E' := \{e \in E, \exists \{u,v\} \subseteq e\colon\ u \in V_i, v \in V_j, i \neq j\}$ 是割集（即所有割网的集合）。平衡约束要求每个块中的顶点权重之和不超过与某个允许的不平衡度 ϵ 相关的阈值。更具体地说，$\forall i \in \{1,\ldots,k\}\colon\ c(V_i) \leq L_{\max} := \left\lceil (1+\epsilon)\frac{c(V)}{k} \right\rceil$。对于超图中的每个网 e，$\Lambda(e) := \{V_i \mid V_i \cap e \neq \emptyset\}$ 表示 e 的连通集。网 e 的连通性 $\lambda(e)$ 是其连通集的基数，即 $\lambda(e) := |\Lambda(e)|$。所谓的连通性指标（$\lambda$-1）计算为 $\sum_{e\in E'}(\lambda(e)-1)\ \omega(e)$，其中 E' 是割集。

流式处理。 流式算法通常遵循迭代式的加载-计算-存储逻辑。我们关注且最常用的流式模型是单遍 模型。在此模型中，（超）图的顶点连同其邻接表逐个加载，随后应用某种逻辑将其永久分配到各个块中，如图 1 所示。类似的操作序列也用于即时划分图的边流。在这种情况下，图的边连同其端点逐个加载，随后应用某种逻辑将其永久分配到各个块中。这种逻辑可以简单到只是一个`Hashing` 函数，也可以复杂到根据某个目标对所有块进行评分，再将顶点分配到得分最高的块。还有其他更复杂的流式模型，例如滑动窗口 [31] 和缓冲流式处理 [22, 14], ，但它们超出了本文的讨论范围。

2.2 相关工作

关于（超）图划分的研究成果极为丰富。用于对内存中的（超）图进行划分的代表性工具包括 `PaToH` [11], `Metis` [23], `hMetis` [24], `Scotch` [32],`HYPE` [27], `KaHIP` [35], `KaMinPar` [19], `KaHyPar` [36], `Mt-KaHyPar` [18] 和 `mt-KaHIP` [2]。关于这方面的详细资料和参考文献，请参阅 [12, 10, 38] 。本文将重点关注与本文范围直接相关的研究成果。具体而言，我们将基于单遍流式模型，详细回顾以下问题：超图划分和图顶点划分。

流式超图划分。 Alistarh 等人 [3] 提出了 `Min-Max`，这是一种单遍流式算法，用于将超图的顶点分配到各个块中。对于每个块，该算法都会跟踪其中包含引脚的网。这意味着内存消耗为 $O(mk)$。加载一个顶点时， `Min-Max` 会在满足严格负载均衡约束的前提下，将其分配到与该顶点的网交集最大的块中。作者从理论上证明，该算法能够以高概率恢复隐藏的协同聚类，

其中，协同聚类定义为同时对顶点和超边进行聚类。在实验评估中， Min-Max 在负载不平衡度方面优于五种直观的流式方法，但其生成的解的不平衡度最高可达内存算法（如 hMetis）的五倍。

Taşyaran 等人 [40] 提出了该算法 Min-Max [3]的改进版本。作者介绍了 Min-Max-N2P，这是对 Min-Max 的修改版本：它存储包含各网引脚的块，而不是像 Min-Max 那样按块存储网。在他们的实验中，Min-Max-N2P 的速度比 Min-Max 快三个数量级，同时保持了相同的割网数。作者还介绍了三种相比 Min-Max 内存消耗更低的算法： Min-Max-Lℓ，这是对 Min-Max-N2P 的修改版本，采用上界 ℓ 来限制每个网的内存消耗； Min-Max-BF 使用布隆过滤器进行成员查询； Min-Max-MH 则使用哈希函数替代块与网之间的连通性信息。在实验中，这三种算法相比 Min-Max 缩短了运行时间，尤其是 Min-Max-Lℓ 和 Min-Max-MH，速度最多快了四个数量级。另一方面，这三种算法生成的解的割网数都比 Min-Max 更高，尤其是 Min-Max-MH，其割网数最多增加了一个数量级。此外，作者提出了一种改进流式环境下划分决策的技术：加入一个缓冲区，用于存储部分顶点及其网集合。这种方法的运作方式与 Min-Max-N2P 类似，但还能够重新访问缓冲区中的顶点，并依据连通性指标调整其划分分配。作者基于这种缓冲方法提出了三种算法： REF 缓冲每个入射顶点，但只重新分配可能改善连通性的顶点； REF_RLX 缓冲所有顶点，并重新分配缓冲区中的所有顶点； REF_RLX_SV 只缓冲网集合较小的顶点，并重新分配缓冲区中的所有顶点。实验结果表明，与非缓冲方法相比，采用缓冲方法可使划分质量提高 5–20%，但代价是运行时间增加。

流式图顶点划分。 Stanton和Kliot [39] 在流式模型中引入了图划分，并提出了一些启发式方法来解决这一问题。他们最有代表性的启发式方法包括单遍方法 Hashing 和线性确定性贪心 (LDG)。在他们的实验中， LDG 取得了整体上最好的边割。在该算法中，顶点分配优先选择包含更多邻居的块，并使用惩罚乘数来控制不平衡度。具体而言，顶点 v 会被分配到使 $|V_i \cap N(v)| * \lambda(i)$最大化的块 V_i，其中 $\lambda(i)$是定义为$(1 - \frac{|V_i|}{L_{\max}})$的乘法惩罚项。其基本思路是，惩罚项可以避免已经非常繁重的块继续超载。当目标函数相同时， LDG会将该顶点移至顶点数较少的块。总体而言， LDG 以 $O(m+nk)$的运行时间划分图。另一方面， Hashing 的运行时间为 $O(n)$，但产生的边割较差。

Tsourakakis 等人 [41] 提出了 Fennel，这是一种基于广为人知的聚类目标函数 模块度 [8]的单遍划分启发式。 Fennel 将顶点 v 分配到一个块V_i中，同时遵循平衡阈值，以最大化形如 $|V_i \cap N(v)| - f(|V_i|)$ 的表达式，也就是带有加性惩罚的表达式。该表达式综合了两种性质：倾向于加入邻居众多的块，以及排斥非邻居众多的块。当 $f(|V_i|)$ 为常数时，该表达式与第一种性质一致。如果 $f(|V_i|) = |V_i|$，该表达式则与第二种性质一致。具体而言，作者定义的 Fennel 目标函数满足 $f(|V_i|) = \alpha * \gamma * |V_i|^{\gamma-1}$，其中 γ是一个自由参数，且 $\alpha = m\frac{k^{\gamma-1}}{n^{\gamma}}$。经过作者进行的参数调优后， Fennel 采用了 $\gamma = \frac{3}{2}$，由此实现了 $\alpha = \sqrt{k}\frac{m}{n^{3/2}}$由于 LDG， Fennel 能够在 $O(m + nk)$ 时间内划分一张图。

Faraj 和 Schulz [15]提出了一种用于顶点划分的共享内存流式算法

该算法能够在线执行递归多重划分。作为预处理阶段，他们的算法将一个 k-路划分问题分解为一个层次结构，其中包含 $\lceil \log_b k \rceil$ 层 b-路划分子问题。该层次结构既可以反映高性能系统的拓扑结构，以解决进程映射问题 [16, 34] ，也可以针对任意 k 进行计算，以解决常规顶点划分问题。随后，算法采用 Fennel 的一个改编版本来解决各个子问题，从而在对图进行单次遍历的过程中在线计算出整个 k-划分。尽管其边割比 Fennel 低约 5%，该算法的理论复杂度为 $O((m + nb) \log_b k)$，实验速度最高可比 Fennel 快两个数量级。

除了单遍模型之外，其他流式模型也被用于解决顶点划分问题。Nishimura 和 Ugander [30]提出了再流式图划分。在该模型中，允许多次遍历整个输入，从而能够进行迭代改进。作者提出了易于实现的 LDG和 Fennel的再流式版本，分别为 ReLDG 和 ReFennel。Awadelkarim 和 Ugander [5] 研究了顶点排序对流式图划分的影响。作者引入了优先级流式处理的概念，其中已（重新）流式传输的顶点会根据某种优先级进行静态或动态重排。作者提出了 ReLDG的优先级版本。Patwary 等人 [31] 提出了 WStream，这是一种保留滑动流窗口的贪心流式算法。Jafari 等人 [22] 提出了一种基于缓冲流式模型的共享内存多层算法。他们的算法将单遍算法 LDG 用于其多层方案中的粗化、初始划分和局部搜索步骤。Faraj 和 Schulz [14]提出了 HeiStream，这是一种同样基于缓冲流式模型的多层算法。他们的算法加载一个顶点块，构建一个模型，然后使用与 Fennel 目标函数扩展版本相结合的传统多层算法对该模型进行划分。

3 FREIGHT：快速流式超图划分

本节将详细介绍我们的算法贡献。首先，我们定义名为 FREIGHT的算法。接下来，我们介绍使用两种不同格式对流式超图进行处理，并使用 FREIGHT对其进行划分的优缺点。此外，我们还将说明，通过为块排序实现高效的数据结构，我们如何消除了FREIGHT 的复杂度对 k 的依赖。

3.1 数学定义

本节将通过把 Fennel 的思想扩展到超图领域，为 FREIGHT 给出数学定义。回顾一下，假设图的顶点逐个流入， Fennel 算法会将入射顶点 v 分配到块 V_d，其中 d 的计算方式如下：

$$d = \underset{i,\ |V_i|<L_{\max}}{\operatorname{argmax}} \left\{ |V_i \cap N(v)| - \alpha * \gamma * |V_i|^{\gamma-1} \right\} \tag{1}$$

术语 $-\alpha * \gamma * |V_i|^{\gamma-1}$ 在 Fennel 中用于惩罚块不平衡度，在 FREIGHT 中直接沿用，含义不变。术语 $|V_i \cap N(v)|$ 用于最小化 Fennel 中的边割，因此在 FREIGHT 中需要调整，以最小化预期指标，即割网或连通性。在说明如何调整之前，请回顾一点：与图划分不同，在超图划分中，入射顶点 v 的关联网 $I(v)$ 可能包含已经被割开的网，即引脚已分配到多个块的网。为优化 连通性 而设计的 FREIGHT 版本会考虑已被割开的网，方法是跟踪每个网 Fennel 最近流入的引脚被分配到的块

。更正式地说， FREIGHT 的连通性版本会将超图的入射顶点 v 分配到块 V_d，其 d 由方程（2）给出，其中 $I^i_{obj}(v) = I^i_{con}(v) = \{e \in I(v) : d_e = i\}$。另一方面，为优化 割网 而设计的 FREIGHT 版本会忽略已经被割开的网，因为它们对超图 k-划分的总割网的贡献已固定，无法再改变。更正式地说， FREIGHT 的割网版本会将超图的入射顶点 v 分配到块 V_d，其 d 由方程（2）给出，其中 $I^i_{obj}(v) = I^i_{cut}(v) = I^i_{con}(v) \setminus E'$， E' 是已被割开的网集合。

$$d = \underset{i,\ |V_i|<L_{\max}}{\operatorname{argmax}} \left\{ |I^i_{obj}(v)| - \alpha * \gamma * |V_i|^{\gamma-1} \right\} \tag{2}$$

FREIGHT 的两种配置都兼顾两个目标：倾向于选择包含许多关联（未割）网的块，同时惩罚基数较大的块。简要来说， FREIGHT 可调整以适用于加权超图。具体而言，处理加权网时，将项 $|I^i_{obj}(v)|$ 替换为 $\omega(I^i_{obj}(v))$。同样，处理加权顶点时，将项 $-\alpha * \gamma * |V_i|^{\gamma-1}$ 替换为 $-c(v) * \alpha * \gamma * c(V_i)^{\gamma-1}$，其中将 v 的权重 $c(v)$用作惩罚项中的乘数因子。

3.2 流式超图

本节介绍并讨论 FREIGHT采用的流式模型。回顾一下，在图的流式模型中，顶点及其邻接表会逐个加载。因此，仅流式处理图（不进行额外计算）就意味着时间开销为 $O(m+n)$。在我们的模型中，超图的顶点及其关联的网会逐个加载，如图 1 所示。我们的流式模型意味着流式处理超图的开销为$O(\sum_{e\in E}|e|+n)$，其中 $O(\sum_{e\in E}|e|)$是每个网恰好流式处理 $|e|$ 次的开销。 FREIGHT 使用 $O(m+k)$内存，其中 $O(m)$用于跟踪每个网 e的割/未割状态，以及该网最近流入的引脚被分配到的块 d_e。这种网跟踪信息取代了跟踪顶点分配的需求，是执行 FREIGHT所必需的。尽管 FREIGHT 比基于图的流式算法所需的内存更多（后者通常使用 $O(m+k)$内存），但它仍远优于流式超图划分领域最先进算法所需的 $O(mk)$最坏情况内存；[3, 40], 这些算法也都基于一种计算模型，而该模型仅流式处理超图就意味着开销为 $O(\sum_{e\in E}|e|+n)$。

3.3 高效实现

本节将介绍 FREIGHT 的一种高效实现。回顾一下，对于每个被加载的顶点 v， FREIGHT 都会使用方程（2），从最多 k 个选项中找出得分最高的块。一种简单的实现方法是显式计算每个块的得分，并找出得分最高的块。这样总共需要进行 $O(nk)$ 次计算，整体复杂度为 $O(\sum_{e\in E}|e|+nk)$。我们提出了一种比这种方法高效得多的实现。

对于每个已加载顶点 v，我们的实现会将满足条件 $|V_i| < L_{\max}$的块 V_i 划分为两个不相交集合： S_1 和 S_2。具体而言，集合 S_1 包含满足$|I^i_{obj}(v)| > 0$的块 V_i ，而集合 S_2 则包含其余块，即满足$|I^i_{obj}(v)| = 0$的块 V_i。利用这两个集合，我们将方程（2）分解为方程（3）和方程（4），并分别求解。然后根据所得解的

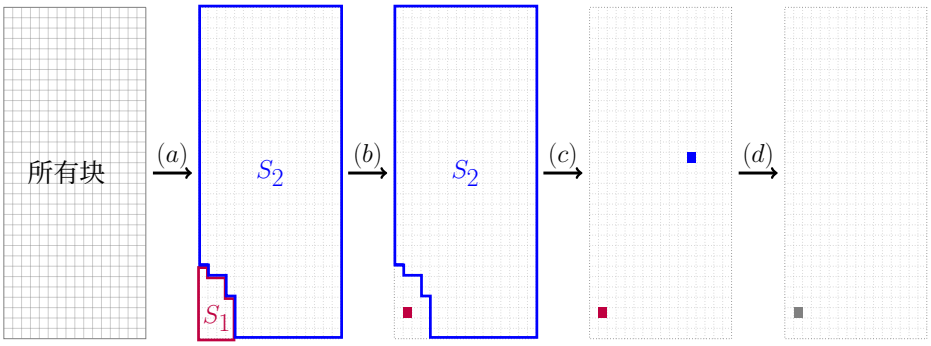


图 2 展示了如何为具有 $k = 512$ 个块的入射顶点 u 求解方程（2）。（a）将 k 个块分解为 S_1 和 S_2，其复杂度为 $|S_1| = O(|I(u)|)$。（b）以 $O(|I(u)|)$ 的代价显式求解方程（3）。（c）以 $O(1)$ 的代价隐式求解方程（4）。（d）随后使用各解的 FREIGHT 得分进行评估，以确定方程（2）的最终解。

得分 FREIGHT 进行比较，最终求得方程（2）的解。整体流程如图 2 所示。

$$d = \underset{i \in S_1}{\operatorname{argmax}} \left\{ |I^i_{obj}(v)| - \alpha * \gamma * |V_i|^{\gamma - 1} \right\} \tag{3}$$

$$d = \underset{i \in S_2}{\operatorname{argmax}} \left\{ |I^i_{obj}(v)| - \alpha * \gamma * |V_i|^{\gamma - 1} \right\} = \underset{i \in S_2}{\operatorname{argmin}} |V_i| \tag{4}$$

现在我们解释如何求解方程（3）和方程（4）。为了求解方程（3），我们采用定理 1 中概述的理论复杂度，并对其进行显式求解。相比之下，方程（4）通过确定基数最小的块进行隐式求解。在整个执行过程中，我们使用高效的数据结构按基数对所有块进行排序，从而能够在常数时间内求解方程（4）。

▶**定理 1**。 *方程（3）可以在时间 $O(|I(v)|)$内求解。*

证明。 方程（3）中的各项 $|I^i_{obj}(v)|$ 可以通过遍历 v 的网来计算，代价为 $O(|I(v)|)$，并确定它们的状态是割、未分配，还是已分配到块。因子 $-\alpha * \gamma * |V_i|^{\gamma-1}$ 在方程（3）中的计算可以在时间$O(|S_1|) = O(|I(v)|)$内完成，证明完毕。◀

现在，我们来介绍一种数据结构，以便在整个算法执行过程中始终按基数对块排序。该数据结构由两个数组 A 和 B（二者均包含 k 个元素）以及一个列表 L实现。数组 A 按升序存储所有 k 块。数组 B 将块 V_i 的索引 i 映射到其在 A中的位置。列表 L中的每个元素都代表一个桶。每个桶对应唯一的块基数，并包含当前具有该基数的块在 A 中的范围，其最左侧和最右侧位置分别为 ℓ 和 r。反过来，A 中的每个块都有一个指针，指向 L中与其基数对应的唯一桶。算法开始时，L 被初始化为一个基数为 0 的单一桶，该桶覆盖 A中的 k 个位置；也就是说，其参数 ℓ 和 r 分别为 1 和 k。初始时，A 中的块可以按任意顺序排列；不过，由于每个块的初始基数均为 0，因此它们已按基数排序。

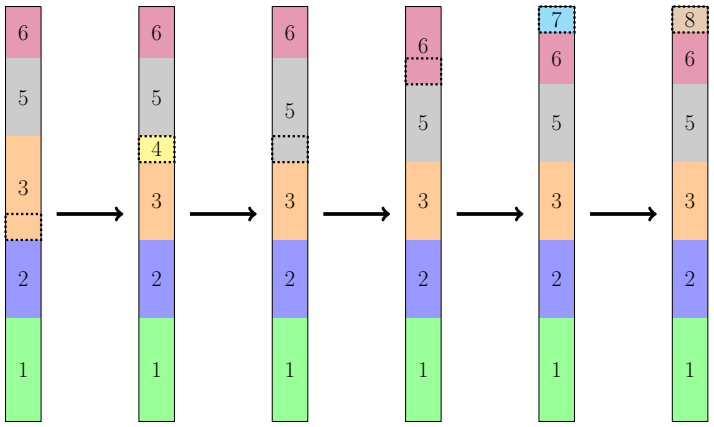


图3 用于在整个 FREIGHT 执行过程中保持块按基数排序的数据结构示意图。数组 A 以竖直矩形表示。A 的每个区域都由一个唯一的桶覆盖；在 A 中，对应区域以唯一的填充颜色表示。每个桶对应的基数写在其覆盖的 A 区域中央。这里展示了连续五次将顶点分配给虚线矩形所围住的块时，该数据结构的工作情况。

当一个顶点被分配到块 V_d时，我们按照算法1中的详细说明更新数据结构，图3给出了示例。下面详细介绍算法1。第 1 行中，我们找到 V_d 在 A 中的位置 p，并找到与其关联的桶 C。第 2 行中，我们交换 A中两个位置的内容：V_d 所在的位置，以及 C中由变量 r 标识的位置；该变量标记了 A 中由 C覆盖的最右侧块。随后在第 3 行中将变量 r减 1，因为 V_d 现在已不再由桶 C覆盖。在第 4 行和第 5 行中，我们检查 V_d 增加后的新基数是否与其紧邻右侧块在 A中的基数相同。如果相同，则将 V_d 关联到同一个桶，并在第 6 行中将该桶最左侧的位置 ℓ 减 1；否则，在第 8 行和第 9 行中，我们向 L中加入一个新桶，并将其恰当地与 V_d 匹配。最后，在第 10 行中，如果 C 的范围 $[\ell, r]$ 为空，则将其删除。图3展示了连续执行算法1五次期间的数据结构变化。定理2证明了该数据结构的正确性。定理3表明，使用我们提出的数据结构，只需 $O(1)$ 时间即可求解方程（4），或者证明方程（3）的解也能求解方程（2）。需要指出的是，使用桶队列会得到相同的复杂度，但需要 $O(k + L_{\max})$ 内存，而我们的数据结构只需 $O(k)$。由定理 1 和定理 3 直接推出的 FREIGHT 总体复杂度见推论 4。

▶**定理2**。 我们提出的数据结构使数组 A 中的块始终按基数升序排列。

证明。 我们通过归纳同时证明两个断言：(a) 每个桶中包含的变量 ℓ 和 r 分别存储从 L 中当前具有该基数的唯一块范围的最左侧位置和最右侧位置；(b) 数组 A 中的块按基数升序排列。初始时，这两个断言显然成立，因为所有块的基数均为 0，且 L 初始化为仅含一个桶的数组，其中$\ell = 1$ 和 $r = k$。现在假设在某个时刻 (a) 和 (b) 均成立，我们来证明执行算法1后它们仍然成立。注意，在整个算法执行过程中，第 2 行是 A 中唯一进行位置交换的地方。由于假设 (a) 成立，因此 V_d

算法1 在所提出的数据结构中增加块 V_d 的基数

1: $p \leftarrow B_d$; $C \leftarrow A_p.bucket$;
2: $q \leftarrow C.r$; $c \leftarrow A_q.id$; $Swap(A_p, A_q)$; $Swap(B_c, B_d)$;
3: $C.r \leftarrow C.r - 1$;
4: $C' \leftarrow A_{q+1}.bucket$;
5: **如果** $C.cardinality + 1 = C'.cardinality$ **那么**
6: $A_q.bucket \leftarrow C'$; $C'.\ell \leftarrow C'.\ell - 1$;
7: **否则**
8: $C'' \leftarrow NewBucket()$; $A_q.bucket \leftarrow C''$; $L \leftarrow L \cup \{C''\}$;
9: $C''.cardinality \leftarrow C.cardinality + 1$; $C''.\ell \leftarrow q$; $C''.r \leftarrow q$;
10: **如果** $C.r = C.\ell$ **那么** $L \leftarrow L \setminus \{C\}$;

会与 A 中具有与 V_d 相同基数的最右侧块交换位置。由于 V_d 的基数将增加 1，且所有块的基数均为整数，因此 (b) 得证。为证明 (a) 仍然成立，注意到 L 中仅有以下桶会被修改：C （第 3 行）、C' （第 6 行）和 C'' （第 9 行）。对于 C，断言 (a) 仍然成立，因为基数将增加的 V_d 是从其所属范围中移除的唯一块。对于 C'，断言 (a) 仍然成立，因为只有当 V_d 的新基数等于 C' 的基数时才会执行第 6 行，而 C' 当前范围的起始位置紧接在 A 中 V_d 的新位置之后。只有当 V_d 的新基数分别大于和小于 C 和 C' 的基数时，才会创建桶 C''。由于 (b) 成立，只有在 A 中不存在与 V_d 的新基数相同基数的块时，才会出现这种情况。因此，对于新创建的 C''，断言 (a) 仍然成立；该桶仅覆盖 A 中 V_d 所在的位置。
◀

▶**定理 3.** *利用我们提出的数据结构，可以在 $O(1)$ 时间内求解方程（4），或者证明方程（3）的任意解也是方程（2）的解。*

证明。算法1不包含循环，且其中每条命令的复杂度均为 O(1)，因此该算法的总成本为 O(1)。我们的数据结构会针对每个已分配的顶点执行一次算法1，因此每个顶点的成本为 O(1)。假设我们正在评估一个入射顶点 v。根据定理2，基数最小的块 V_d 存储在数组 A的第一个位置，因此可以在 O(1) 时间内访问。如果 $V_d \in S_2$，那么 d就是方程（4）的一个解。另一方面，如果 V_d 位于 S_1中，那么 V_d 的得分 FREIGHT 至少比方程（4）解的得分 FREIGHT 大 $|I_d(v)| > 0$。在这种情况下，可以得出方程（3）的任意解都能解方程（2）。
◀

▶ **推论 4.** *FREIGHT 的整体复杂度为 $O\big(\sum_{e \in E} |e| + n\big)$。*

4 实验评估

设置。 我们使用 C++ 实现了这些程序，并在开启完整优化（-O3 标志）的情况下，使用 gcc 11.2 对其进行编译。除非另有说明，所有实验均在一台运行 Ubuntu 20.04.1 的机器上使用单核进行。该机器配备一颗 16 核 Intel Xeon Silver 4216 处理器，主频为 2.1 GHz，拥有 100 GB 主内存、16 MB L2 缓存和 22 MB L3 缓存，并支持超线程，可处理 32 个线程。除非另有说明，我们均直接从内存中流式处理（超）图，以便清晰地比较运行时间。不过请注意， FREIGHT 以及大多数其他使用的算法也可以从硬盘流式处理超图。

10 FREIGHT：快速流式超图划分

基线。 我们将 FREIGHT 与各种最先进算法进行比较。本节将列出这些算法，并说明我们选择算法的标准。由于 Hashing 是一种简单的算法，我们使用 C++ 实现了它。该算法基本上是将传入顶点的 ID 哈希到 $\{1, \ldots, k\}$ 中。其余算法均取自官方代码库或由作者私下提供，只有 Min-Max 例外，因为它没有可用的官方实现。在此，我们使用 Taşyaran 等人 [40] 提供的 Min-Max 实现。所有算法均使用 gcc 11.2 编译。

我们运行 Hashing、 Min-Max [3] 及 Taşyaran 等人提出的所有改进版本 [40]: Min-Max-BF、 Min-Max-N2P、 Min-Max-Lℓ、 Min-Max-MH、 REF、 REF_RLX 和 REF_RLX_SV。（不同 Min-Max 版本的详情见第 2.2 节）、 HYPE [27] 以及 PaToH v3.3 [11]。Hashing 值得关注，因为它是最简单、最快的流式算法，可为划分时间提供下界。 Min-Max 是当前在割网和连通性方面表现最先进的流式超图划分算法。 [40] 中提出的 Min-Max 的改进版本和缓冲版本值得关注，因为其中一些版本的速度快了几个数量级，而另一些版本生成的划分则优于 Min-Max 。HYPE 和 PaToH 是用于超图划分的内存算法，因此不适用于流式环境。不过，我们仍将它们作为比较对象，因为 HYPE 属于最快的内存算法之一，而 PaToH 速度很快，并且也能计算出割网和连通性都非常出色的划分。请注意，KaHyPar [36]是解质量方面的领先工具，但它也比 PaToH慢得多。

实例。我们从多个来源选取了超图来测试我们的算法。所考虑的超图曾被用于以往超图划分研究中的基准测试。每次实验之前，我们都将所有超图转换为各算法所需的相应流式处理格式。我们移除了平行超边、空超边和自环，并为所有顶点和超边赋予单位权重。在所有使用流式算法的实验中，我们均按照顶点给定的自然顺序对超图进行流式处理。除非另有说明，我们使用的块数为 $k \in \{$512、1024、1536、2048、2560$\}$。所有实验（以及所有算法）均采用固定的 3% 不平衡度，这是划分文献中常用的取值。除 HYPE 外，所有算法生成的始终都是平衡划分； HYPE 在约 5% 的实验中生成了高度不平衡的划分。

我们使用与 [36] 相同的基准测试。该基准测试包含来自三个基准测试集的 310 个超图: 18 个来自 ISPD98 电路基准测试套件的超图 [4], 192 ，基于佛罗里达大学稀疏矩阵集合的超图 [13], ，以及来自国际 SAT 竞赛的 100 个实例 2014 [7]。SAT 实例通过将每个布尔变量及其补变量映射为一个顶点、将每个子句映射为一条网，转换为超图。从稀疏矩阵集合中，我们为每个应用领域选取了一个列数介于 10 000 和 10 000 000 之间的矩阵。我们使用行网模型将这些矩阵转换为超图，其中每一行视为一条网，每一列视为一个顶点。

方法论。 根据实验的侧重点，我们测量运行时间、割网和/或连通性。对于每种算法和实例，我们使用随机种子对非确定性算法进行 5 次重复实验，并计算每个实例的目标函数值和运行时间的算术平均值。对多个实例进一步求平均时，我们使用几何平均值，以确保每个实例对最终得分的影响相同。

给定算法 A 的结果，我们将其值 σ_A （可以是目标函数值或运行时间）表示为相对于算法 B 的改进幅度 ，计算方式为 $\left(\frac{\sigma_B}{\sigma_A} - 1\right) * 100\%$ ；我们还使用性能剖面 来表示结果。它们将一组算法的运行时间（质量）与逐实例最快（最佳）算法进行比较，

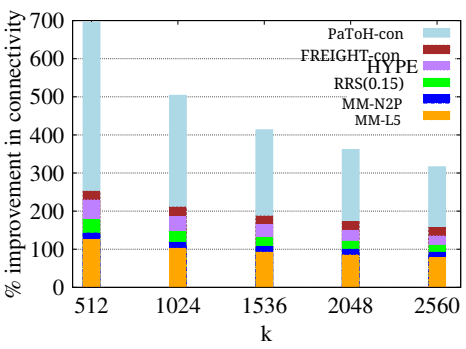
（而非按 k 分组）。x 轴表示因子 τ，y 轴则表示 A 的运行时间（质量）不超过最快（最佳）算法的 τ 倍的实例所占百分比。我们还使用条形图和箱线图来呈现研究结果。我们使用条形图展示目标函数的平均值与 k 的关系，其中每种算法均用一种指定颜色的竖条表示，竖条起点位于 x 轴上。对于 k 的每个取值，相应的条形具有共同的起点，并按照高度排列，使所有条形的高度都清晰可见。我们使用箱线图清晰展示数据集分布，包括最小值、最大值、中位数、第一四分位数和第三四分位数，同时忽略异常值。

4.1 结果

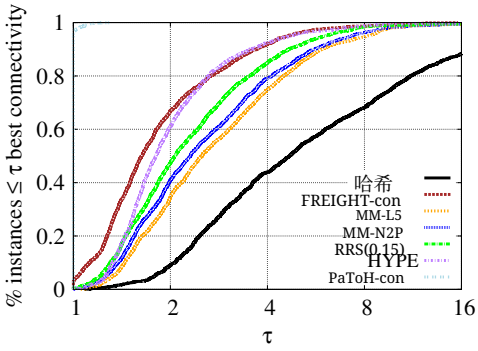
在本节中，我们展示了一组实验，将 FREIGHT 与当前最先进的流式超图划分方法进行比较。如前所述，我们还使用两种内存算法 [27, 11] 作为更通用的基线进行比较。实验评估主要关注解质量和运行时间的比较。请注意， PaToH 和 FREIGHT 各自有不同版本，分别针对不同的质量指标（即连通性和割网）进行优化。为确保比较有意义，在考察每项质量指标时，我们只考虑相应的版本；不过，在比较运行时间时，仍会考虑两个版本。为区分不同版本，我们在名称后添加后缀 -con 和 -cut ，分别表示针对连通性优化的版本和割网版本。为清晰起见，我们不讨论被其他算法支配的最先进流式算法。我们将支配算法定义为：与另一算法相比运行时间更差，且未能以更优的解质量作为补偿，或反之。具体而言，我们不纳入 Min-Max 和 Min-Max-BF ，因为它们被 Min-Max-N2P支配，后者在下文中称为 MM-N2P 。同样，我们省略 Min-Max-MH ，因为它被 Hashing支配。我们按照 [40]中列出的最佳结果，为缓冲算法 REF、 REF_RLX和 REF_RLX_SV采用 15% 的缓冲区大小进行测试。由于前两者被后者支配，我们将其省略，并从此将后者称为 RRS(0.15) 。由于 Min-Max-Lℓ 不被任何其他算法支配，我们展示其与 $\ell = 5$结合的结果，如 [40], 中的最佳结果所示，并从此将其称为 MM-L5 。

连通性。 我们首先考察连通性指标。在图4a中，我们绘制了每个 k取值下，相较于 Hashing 的平均连通性提升。 PaToH-con 的平均连通性最佳，相较于 Hashing平均提升了443%。这与（超）图划分领域以往的研究结果一致：流式算法通常计算出的解不如内存算法，因为后者可以访问整个图。就连通性而言， FREIGHT-con 是表现第二好的算法，优于内存算法 HYPE 和缓冲流式算法 RRS(0.15)。平均而言，这三种算法相较于 Hashing分别提升了194%、171%和136%。最后， MM-N2P 和 MM-L5 计算出的解平均分别比 Hashing 提升了111%和96%。直接比较可见， FREIGHT-con 的平均连通性分别比 HYPE、 RRS(0.15)、 MM-N2P和 MM-L5高出8%、24%、39%和50%。请注意，在所有 k取值下，各算法的平均连通性排名保持不变。

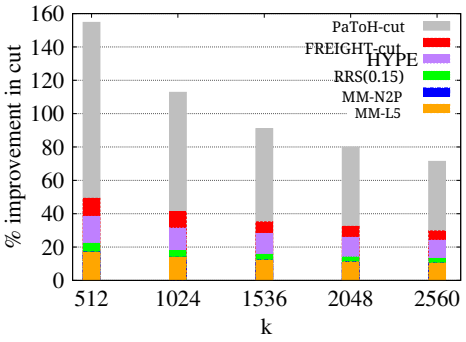
在图4b中，我们绘制了所有实验的连通性性能剖面。 PaToH-con在96.4%的实例中实现了整体最佳连通性，而 FREIGHT-con 在3.1%的实例中实现了最佳连通性；其他算法中，没有任何一种在超过0.35%的实例中实现最佳连通性。所实现的连通性由



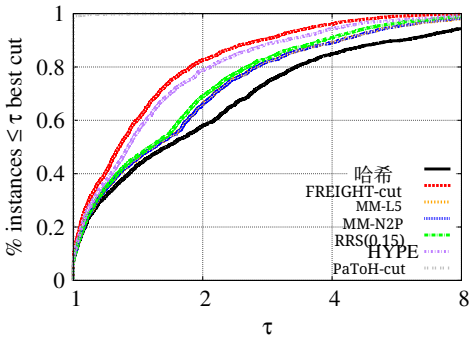
（a）连通性 ioH。



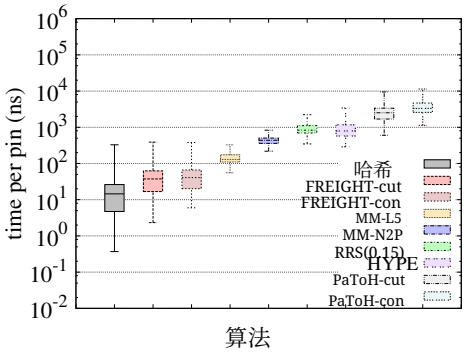
(b) 连通性性能剖面。



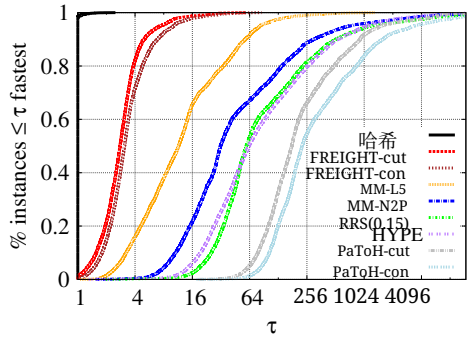
(c) 割网改进图 ioH。



(d) 割网性能剖面。



(e) 运行时间箱线图。



(f) 运行时间性能剖面。

图4 与超图划分最先进的流式算法进行比较。我们展示了性能剖面（pp）、相对于 `Hashing` 的改进图（ioH）以及箱线图（bp）。请注意，在图 4b、4d 和 4f 中，`PaToH-con`、`PaToH-cut` 和 `Hashing` 分别几乎与 y 轴完全重合。此外，在图 4d 和图 4c 中，`MM-N2P` 和 `MM-L5` 的曲线和柱形大致重叠。

FREIGHT-con、 HYPE、 RRS(0.15)、 MM-N2P、 MM-L5和 Hashing 在分别占实例总数 67%、61%、47%、41%、34% 和 9% 的情况下，其连通性结果与最佳结果相差不超过 2 倍。总而言之，在（缓冲）流式处理竞争算法中， FREIGHT-con 的连通性最佳，甚至优于内存算法 HYPE。

割网。 接下来，我们考察割网指标。在图 4c 中，我们绘制了相较于 Hashing的割网改进情况。 PaToH-cut 的整体割网表现最佳，相较于 Hashing平均提升 100%。 FREIGHT-cut 是割网方面的第二佳算法，优于内存算法 HYPE 和缓冲流式算法 RRS(0.15)。这三种算法相较于 Hashing的连通性分别提升了 37%、30% 和 17%。最后， MM-N2P 和 MM-L5 相较于 Hashing的平均连通性均提升了 13%。直接比较可见， FREIGHT-cut 相较于 HYPE、RRS(0.15)、 MM-N2P和 MM-L5的平均连通性分别提升了 6%、18%、22% 和 22%。在所有 k 取值下，各算法的平均割网相对排名均保持不变。

在图 4d 中，我们绘制了所有实验的割网性能剖面。图中， PaToH-cut 在 98.0% 的实例中实现了整体最佳连通性，而FREIGHT-cut 和 HYPE 分别在 6.8% 和 5.2% 的实例中获得最佳割网；其他所有流式算法（RRS(0.15)、 MM-N2P、 MM-L5 和 Hashing）则在 4.8% 的实例中获得最佳割网。 FREIGHT-cut、 HYPE、RRS(0.15)、 MM-N2P、 MM-L5 和 Hashing 得到的割网结果分别在 83%、79%、69%、66%、66% 和 58% 的实例中，与找到的最佳割网相差不超过 2 倍。这表明，FREIGHT-cut 在所有（带缓冲区的）流式处理竞争算法中产生的割网效果最佳，甚至优于内存算法 HYPE。

运行时间。 现在我们比较各算法的运行时间。图4e中的箱线图展示了所有实例中每个引脚的运行时间分布，单位为纳秒。Hashing、 FREIGHT-cut和 FREIGHT-con 是速度最快的三种算法，每个引脚的运行时间中位数分别为 15 纳秒、38 纳秒和 41 纳秒。 MM-L5、MM-N2P、 HYPE和 RRS(0.15)紧随其后，每个引脚的运行时间中位数分别为 130 纳秒、437 纳秒、792 纳秒和 833 纳秒。最后，每个引脚运行时间中位数最高的算法是 PaToH-cut 和 PaToH-con，分别为 2 516 纳秒和 3 333 纳秒。 HYPE 和 PaToH测得的每个引脚运行时间均与先前研究 [37]报告的数值一致。

图 4f 展示了运行时间性能曲线。对于 98.3% 的实例， Hashing 是最快的算法；对于 1.2% 的实例， FREIGHT-cut 最快，而其他算法成为最快算法的实例比例均不超过 0.4%。在 82% 和 72% 的实例中， FREIGHT-cut 和 FREIGHT-con 的运行时间分别不超过 Hashing 的 4 倍。相比之下，这种情况对MM-L5仅出现在 16% 的实例中，而对所有其他算法则出现在不到 0.4% 的实例中。考虑到 FREIGHT的解质量优于Hashing 、所有其他流式算法，甚至 HYPE， FREIGHT 与 Hashing 运行时间接近这一点令人惊讶。

进一步比较。对于图顶点划分， FREIGHT 和 Fennel 在数学上等价。然而，与 Fennel 的标准实现相比， FREIGHT 的计算复杂度更低，为 $O(m+n)$；而 Fennel的复杂度为 $O(m+nk)$，这是因为它需要为每个节点评估所有块。为了针对这一用例优化性能，我们实现了 FREIGHT 的优化版本，其内存消耗为$O(n+k)$，与 Fennel相同。在我们的实验中，我们采用了与 [15] 相同的图，并使用 $k \in \{512, 1024, 1536, 2048, 2560\}$进行了测试。平均而言， FREIGHT 比 Fennel的标准实现快109倍。此外，性能

差距会随着 k 的值增大而扩大；在某些实例中， FREIGHT 的速度最高可达 Fennel 的261倍。

5 结论

在这项工作中，我们介绍了 FREIGHT，这是一种高效且有效的超图划分流式算法。我们的算法采用了经过优化的数据结构，因此运行时间相对于引脚数呈线性增长，内存消耗则与网和块的数量呈线性关系。大量实验结果表明， FREIGHT 的运行时间可与 Hashing 算法相媲美；在四分之三的实例中，观察到的最大差距仅为四倍。值得注意的是，我们的研究结果表明，在割网和连通性度量方面， FREIGHT 始终优于所有现有的（带缓冲区的）流式算法，甚至优于内存算法 HYPE。这凸显了我们提出的算法作为一种高效且有效的解决方案，对于大规模动态数据处理中的超图划分具有重要意义。鉴于取得的良好结果，我们计划很快公开发布该算法。

参考文献

1 Zainab Abbas、Vasiliki Kalavri、Paris Carbone 和 Vladimir Vlassov。流式图划分：一项实验研究。Proc. VLDB Endow.，11(11):1590–1603，2018。URL: http://www.vldb.org/pvldb/vol11/p1590-abbas.pdf, doi:10.14778/3236187.3236208。2 Yaroslav Akhremtsev、Peter Sanders 和 Christian Schulz。高质量共享内存图划分。载于 Marco Aldinucci、Luca Padovani 和 Massimo Torquati（编），Euro-Par 2018：并行处理——第 24 届国际并行与分布式计算会议，意大利都灵，2018 年 8 月 27–31 日，会议论文集，《计算机科学讲义》第 11014 卷，第 659–671 页。Springer，2018。URL： https://doi.org/10.1007/978-3-319-96983-1_47, doi:10.1007/978-3-319-96983-1_47。3 Dan Alistarh、Jennifer Iglesias 和 Milan Vojnovic。流式 Min-Max 超图划分。载于《神经信息处理系统进展》，第 1900–1908 页，2015。doi:10.5555/2969442.2969452。4 Charles J. Alpert。ISPD98 电路基准测试套件。载于 Majid Sarrafzadeh（编），1998 年国际物理设计研讨会论文集，ISPD 1998，美国加利福尼亚州蒙特雷，1998 年 4 月 6–8 日，第 80–85 页。ACM，1998。URL: https://doi.org/10.1145/274535.274546, doi:10.1145/274535.274546。5 Amel Awadelkarim 和 Johan Ugander。面向平衡图划分的优先级重流算法。载于 Rajesh Gupta、Yan Liu、Jiliang Tang 和 B. Aditya Prakash（编），KDD '20：第 26 届 ACM SIGKDD知识发现与数据挖掘大会，线上活动，美国加利福尼亚州，2020 年 8 月 23–27 日，第 1877–1887 页。ACM，2020。URL： https://doi.org/10.1145/3394486.3403239, doi:10.1145/3394486.3403239。6 D. A. Bader、H. Meyerhenke、P. Sanders、C. Schulz、A. Kappes 和 D. Wagner。图聚类与划分的基准测试。载于《社会网络分析与挖掘百科全书》，第 73–82 页。Springer，2014。 doi:10.1007/978-1-4939-7131-2_23。7 A. Belov、D. Diepold、M. Heule 和 M. Järvisalo。2014 年 SAT 竞赛。 http://www.satcompetition.org/2014/，2014。8 Ulrik Brandes、Daniel Delling、Marco Gaertler、Robert Gorke、Martin Hoefer、Zoran Nikoloski 和 Dorothea Wagner。关于模块度聚类。IEEE知识与数据工程汇刊，20(2):172–188，2007。 doi:10.1109/TKDE.2007.190689。9 T. N. Bui 和 C. Jones。寻找良好的近似顶点与边划分是 NP难问题。IPL，42(3):153–159，1992。 doi:10.1016/0020-0190(92)90140-Q。

10 Aydın Buluç、Henning Meyerhenke、Ilya Safro、Peter Sanders 和 Christian Schulz。图划分的最新进展，第 117–158 页。Springer International Publishing，沙姆，2016。doi:10.1007/978-3-319-49487-6_4。11 Ümit V. Çatalyürek 和 Cevdet Aykanat。PaToH（超图划分工具）。载于 David A. Padua（编），《并行计算百科全书》，第 1479–1487 页。Springer，2011。URL: https://doi.org/10.1007/978-0-387-09766-4_93, doi:10.1007/978-0-387-09766-4_93。12 Ümit V. Çatalyürek、Karen D. Devine、Marcelo Fonseca Faraj、Lars Gottesbüren、Tobias Heuer、Henning Meyerhenke、Peter Sanders、Sebastian Schlag、Christian Schulz、Daniel Seemaier 和 Dorothea Wagner。（超）图划分的更多最新进展。ACM计算调查，2023。 doi:doi.org/10.1145/3571808。13 Timothy A. Davis 和 Yifan Hu。佛罗里达大学稀疏矩阵集合。ACM Trans. Math. Softw.，38(1):1:1–1:25，2011。URL: https://doi.org/10.1145/2049662.2049663,doi:10.1145/2049662.2049663。14 Marcelo Fonseca Faraj 和 Christian Schulz。缓冲式流图划分。ACM J. Exp. Algorithmics，27，2022 年 10 月。URL： https://doi.org/10.1145/3546911, doi:10.1145/3546911。15 Marcelo Fonseca Faraj 和 Christian Schulz。即时递归多路划分：用于层次图划分与进程映射的共享内存流式算法。载于 2022 年 IEEE国际集群计算会议（CLUSTER），第 473–483 页，2022。URL: https://ieeexplore.ieee.org/document/9912716, doi:10.1109/CLUSTER51413.2022.00057。16 Marcelo Fonseca Faraj、Alexander van der Grinten、Henning Meyerhenke、Jesper Larsson Träff 和 Christian Schulz。高质量层次进程映射。载于 Simone Faro 和 Domenico Cantone（编），第 18 届国际实验算法研讨会，SEA 2020，2020 年 6 月 16–18 日，意大利卡塔尼亚，第 160 卷 LIPIcs，第 4:1–4:15 页。Schloss Dagstuhl - Leibniz-Zentrum für Informatik，2020。URL： https://doi.org/10.4230/LIPIcs.SEA.2020.4,doi:10.4230/LIPIcs.SEA.2020.4。17 M. R. Garey、David S. Johnson 和 Larry J. Stockmeyer。一些简化的 NP完全问题。载于 Robert L. Constable、Robert W. Ritchie、Jack W. Carlyle 和 Michael A. Harrison（编），第 6 届 ACM计算理论研讨会年度会议论文集，1974 年 4 月 30 日至 5 月 2 日，美国华盛顿州西雅图，第 47–63 页。ACM，1974。URL： https://doi.org/10.1145/800119.803884, doi:10.1145/800119.803884。18 Lars Gottesbüren、Tobias Heuer、Peter Sanders 和 Sebastian Schlag。可扩展共享内存超图划分。载于算法工程与实验研讨会 ALENEX 论文集，第 16–30 页，2021。 doi:10.1137/1.9781611976472.2。19 Lars Gottesbüren、Tobias Heuer、Peter Sanders、Christian Schulz 和 Daniel Seemaier。深度多层图划分。载于 Petra Mutzel、Rasmus Pagh 和 Grzegorz Herman（编），第 29 届欧洲算法研讨会，ESA 2021，2021 年 9 月 6–8 日，葡萄牙里斯本（线上会议），第 204 卷 LIPIcs，第 48:1–48:17 页。Schloss Dagstuhl - Leibniz-Zentrum für Informatik，2021。URL: https://doi.org/10.4230/LIPIcs.ESA.2021.48, doi:10.4230/LIPIcs.ESA.2021.48。20 Loc Hoang、Roshan Dathathri、Gurbinder Gill 和 Keshav Pingali。Cusp：用于分布式图分析的可定制流式边划分器。载于 2019 年 IEEE国际并行与分布式处理研讨会（IPDPS），第 439–450 页。IEEE，2019。 doi:10.1109/IPDPS.2019.00054。21 Manuel Holtgrewe、Peter Sanders 和 Christian Schulz。构建可扩展的高质量图划分器。载于第 24 届 IEEE国际并行与分布式处理研讨会，IPDPS 2010，美国佐治亚州亚特兰大，2010 年 4 月 19–23 日——会议论文集，第 1–12 页。IEEE，2010。URL: https://doi.org/10.1109/IPDPS.2010.5470485,doi:10.1109/IPDPS.2010.5470485。

22 Nazanin Jafari、Oguz Selvitopi 和 Cevdet Aykanat。快速共享内存流式多层图划分。《并行与分布式计算杂志》，147:140–151，2021。 doi:10.1016/j.jpdc.2020.09.004。23 George Karypis 和 Vipin Kumar。面向不规则图的并行多层k路划分方案。载于《1996年ACM/IEEE超算大会论文集》，1996年11月17—22日，美国宾夕法尼亚州匹兹堡，第35页。IEEE计算机学会，1996。网址： http://doi.ieeecomputersociety.org/10.1109/SC.1996.32, doi:10.1109/SC.1996.32。24 George Karypis 和 Vipin Kumar。多层k路超图划分。载于 Mary Jane Irwin 编，《第36届设计自动化大会论文集》，1999年6月21—25日，美国路易斯安那州新奥尔良，第343–348页。ACM出版社，1999。网址： https://doi.org/10.1145/309847.309954, doi:10.1145/309847.309954。25 Renaud Lambiotte、Martin Rosvall 和 Ingo Scholtes。从网络到复杂系统的最优高阶模型。《自然·物理》，15(4):313–320，2019。网址： https://doi.org/10.1038/s41567-019-0459-y, doi:10.1038/s41567-019-0459-y。26 J. Leskovec 和 A. Krevl。SNAP：斯坦福大型网络数据集。 http://snap.stanford.edu/data，2014年6月。27 Christian Mayer、Ruben Mayer、Sukanya Bhowmik、Lukas Epple 和 Kurt Rothermel。HYPE：通过邻域扩展进行大规模超图划分。载于 Naoki Abe、Huan Liu、Calton Pu、Xiaohua Hu、Nesreen K. Ahmed、Mu Qiao、Yang Song、Donald Kossmann、Bing Liu、Kisung Lee、Jiliang Tang、Jingrui He 和 Jeffrey S. Saltz 编，《IEEE国际大数据会议（IEEE BigData 2018）》，2018年12月10—13日，美国华盛顿州西雅图，第458–467页。IEEE，2018。网址： https://doi.org/10.1109/BigData.2018.8621968, doi:10.1109/BigData.2018.8621968。28 Christian Mayer、Ruben Mayer、Muhammad Adnan Tariq、Heiko Geppert、Larissa Laich、Lukas Rieger 和 Kurt Rothermel。Adwise：用于高速图处理的自适应窗口式流式边划分。载于《2018年IEEE第38届国际分布式计算系统会议》，第685–695页。IEEE，2018。 doi:10.1109/ICDCS.2018.00072。29 Ruben Mayer、Kamil Orujzade 和 Hans-Arno Jacobsen。2ps：采用两阶段流式处理的高质量边划分。CoRR，abs/2001.07086，2020。网址： https://arxiv.org/abs/2001.07086, arXiv:2001.07086。30 Joel Nishimura 和 Johan Ugander。再流式图划分：适用于高级负载均衡的简单通用算法。载于《第19届ACM SIGKDD国际知识发现与数据挖掘会议论文集》，第1106–1114页，2013。doi:10.1145/2487575.2487696。31 Md Anwarul Kaium Patwary、Saurabh Garg 和 Byeong Kang。基于窗口的流式图划分算法。载于《澳大拉西亚计算机科学周多会议论文集》，第1–10页，2019。 doi:10.1145/3290688.3290711。32 F. Pellegrini 和 J. Roman。静态映射双递归二分算法的实验分析。技术报告，TR 1038-96，LaBRI，1996。网址： https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=94b913363b57e019b8a32529b076a8d4181587ac。33 Fabio Petroni、Leonardo Querzoni、Khuzaima Daudjee、Shahin Kamali 和 Giorgio Iacoboni。Hdrf：幂律图的基于流的划分。载于《第24届ACM国际信息与知识管理会议论文集》，第243–252页，2015。 doi:10.1145/2806416.2806424。34 Maria Predari、Charilaos Tzovas、Christian Schulz 和 Henning Meyerhenke。一种将复杂网络映射到分层架构上的MPI算法。载于 Leonel Sousa、Nuno Roma 和 Pedro Tomás 编，《Euro-Par 2021：并行处理——第27届国际并行与分布式计算会议》，2021年9月1—3日，葡萄牙里斯本，会议录，《计算机科学讲义》第12820卷，第167–182页。Springer，2021。网址： https://doi.org/10.1007/978-3-030-85665-6_11, doi:10.1007/978-3-030-85665-6_11。

35 Peter Sanders 和 Christian Schulz。局部思考，整体行动：高度平衡的图划分。载于 Vincenzo Bonifaci、Camil Demetrescu 和 Alberto Marchetti-Spaccamela 编，《实验算法》，第12届国际研讨会 SEA 2013，2013年6月5—7日，意大利罗马，会议录，《计算机科学讲义》第7933卷，第164–175页。Springer，2013。网址：https://doi.org/10.1007/978-3-642-38527-8_16, doi:10.1007/978-3-642-38527-8_16。36 Sebastian Schlag、Vitali Henne、Tobias Heuer、Henning Meyerhenke、Peter Sanders 和 Christian Schulz。通过n层递归二分进行k路超图划分。载于 Michael T. Goodrich 和 Michael Mitzenmacher 编，《第十八届算法工程与实验研讨会论文集》，ALENEX 2016，2016年1月10日，美国弗吉尼亚州阿灵顿，第53–67页。SIAM，2016。网址：https://doi.org/10.1137/1.9781611974317.5,doi:10.1137/1.9781611974317.5。37 Sebastian Schlag、Tobias Heuer、Lars Gottesbüren、Yaroslav Akhremtsev、Christian Schulz 和 Peter Sanders。高质量超图划分。《ACM 实验算法期刊》（JEA），2022。网址：https://doi.org/10.1145/3529090, doi:10.1145/3529090。38 C. Schulz 和 D. Strash。图划分：形式化表述及其在大数据中的应用。载于《大数据技术百科全书》。2019。doi:10.1007/978-3-319-63962-8_312-2。39 Isabelle Stanton 和 Gabriel Kliot。面向大型分布式图的流式图划分。载于《第18届ACM SIGKDD国际知识发现与数据挖掘会议论文集》，第1222–1230页，2012。 doi:https://doi.org/10.1145/2339530.2339722。40 Fatih Taşyaran、Berkay Demireller、Kamer Kaya 和 Bora Uçar。有限内存环境下的流式超图划分算法。载于 HPCS 2020——国际高性能计算与仿真会议，第1–8页，在线虚拟会议，西班牙，2021年3月。IEEE。网址： https://hal.archives-ouvertes.fr/hal-03182122。41 Charalampos Tsourakakis、Christos Gkantsidis、Bozidar Radunovic 和 Milan Vojnovic。Fennel：面向超大规模图的流式图划分。载于《第7届ACM国际网络搜索与数据挖掘会议论文集》，第333–342页，2014。 doi:https://doi.org/10.1145/2556195.2556213。42 Cong Xie、Ling Yan、Wu-Jun Li 和 Zhihua Zhang。分布式幂律图计算：理论与实证分析。载于 Zoubin Ghahramani、Max Welling、Corinna Cortes、Neil D. Lawrence 和 Kilian Q. Weinberger 编，《神经信息处理系统进展》第27卷：神经信息处理系统2014年度会议，2014年12月8—13日，加拿大魁北克省蒙特利尔，第1673–1681页，2014。网址： https://proceedings.neurips.cc/paper/2014/hash/67d16d00201083a2b118dd5128dd6f59-Abstract.html。43 Chenzi Zhang、Fan Wei、Qin Liu、Zhihao Gavin Tang 和 Zhenguo Li。通过邻域启发式方法进行图边划分。载于《第23届ACM SIGKDD国际知识发现与数据挖掘会议论文集》，2017年8月13—17日，加拿大新斯科舍省哈利法克斯，第605–614页。ACM，2017。网址： https://doi.org/10.1145/3097983.3098033, doi:10.1145/3097983.3098033。