arXiv:2601.09139v1 [cs.DS] 14 Jan 2026

动态 Hi 分层 j-树分解及其应用

Gramoz Goranci* Monika Henzinger† Peter Kiss* Ali Momeni‡ Gernot Zöcklein†

摘要。 我们为在经历边插入和删除的带容量无向图上设计基于割的优化问题近似算法，开发了一种新的算法框架。具体而言，对于任意小的常数 $\varepsilon \in (0, 1)$，我们的框架动态维护 [Madry FOCS'10], 的分层 j-树分解变体，以实现对图割结构的多对数级近似因子，并支持边更新，摊销更新时间为$O(n^{\varepsilon})$。

因此，我们在全动态设置下，为基本的基于割的优化问题（包括全点对最小割、最稀疏割、多重割和多重割）获得了近似与更新时间/查询时间之间的新权衡。对于后三个问题，这些权衡首次给出了每次操作均以次线性时间实现多对数级近似的全动态算法。

我们动态层次结构背后的主要技术要素，是一种能够以低重配置开销处理顶点拆分的动态割稀疏化算法。为此，我们对 Abraham 等人基于森林打包的动态割稀疏器构造[FOCS'16], 进行了白盒化分析，并提出了关于顶点拆分时维护这些森林的新结构性见解。鉴于割稀疏化在静态和动态图算法领域都具有广泛用途，我们认为这一构造本身也值得关注。

1 引言 基于割的图问题是组合优化和理论计算机科学的核心问题。多年来，人们从理论和实践角度对这些问题进行了广泛研究，其应用领域包括超大规模集成电路设计、社区检测、聚类和图划分 [57]。典型问题包括 s-t 最小割（及其对偶问题 s-t 最大流）、全局最小割和最稀疏割问题。为求解这些问题而开展的研究催生了若干算法瑰宝，例如基于树打包的 Karger 的全局最小割算法 [38], 、Arora、Rao 和 Vazirani 关于扩展流的开创性工作 [14]（用于近似求解最稀疏割），以及用于求解图上流问题所产生的结构化线性规划的内点法 [23, 47, 62, 24, 63, 61]。

设计既快速又能提供有竞争力的近似保证的基于割的问题算法，是算法设计中反复出现的主题。使大多数这类快速算法得以实现的关键，是用更简单的图来近似图这一算法框架。在这一背景下，最有力的概念之一是树流稀疏器 [51]; 它是一棵树 T，能够在因子 q范围内近似无向图 G 所有割的值，这一因子称为稀疏化器的质量。一个密切相关的概念是j-树分解 [46], ，它是一族 j-树，共同近似 G 的割结构（形式化定义见定理 4.1）。Räcke [51] 的开创性工作证明了无知路由背景下树流稀疏器的存在。此后，一系列研究 [17, 35, 52, 46, 53, 33, 45] 提出了树流稀疏器和 j-树分解的快速构造方法，其质量为多对数级，运行时间接近线性。[1] 这些结构为许多基础问题带来了多项式时间近似算法，包括最小二分割、 k-多割和最小最大划分。它们还在近期无向 [55, 39, 49, 56]和有向 [22] s-t最大流问题的快速算法突破中发挥了关键作用。

树流稀疏器和 j-树分解具有广泛的适用性，这推动了近期人们对设计动态算法的浓厚兴趣，旨在支持这些结构在边插入和删除时的维护。尽管在静态场景中构造此类结构已颇具挑战，Goranci、Räcke、

*奥地利维也纳大学计算机科学学院。

†奥地利科学技术研究院 (ISTA)，克洛斯特新堡，奥地利。

‡维也纳大学计算机科学学院、维也纳大学计算机科学博士生院 DoCS，奥地利维也纳。具体而言， [53, 45] 构造一个多对数级质量的树流稀疏器，耗时 $\widetilde{O}(m)$，并 [46] 给出一个质量为 $\widetilde{O}(\log n)^L$ 的 j-树分解算法，耗时 $O(m^{1+2/L})$，适用于任意整数 $L \geq 1$。

Saranurak 和 Tan [33]成功为单位容量图构造了动态树流稀疏器，其质量达到 $n^{o(1)}$，并能以 $n^{o(1)}$ 的更新时间处理边更新。随后，van den Brand、Chen、Kyng、Liu、Meierhans、Gutenberg 和 Sachdeva [59], 将这一结果扩展到加权图，并实现了相同的保证。与此同时，Chen、Goranci、Henzinger、Peng 和 Saranurak [19] 设计了一种全动态算法，用于维护质量为 $\widetilde{O}$（$\log n$）、更新时间为 $\widetilde{O}$（$n^{2/3}$）的j-树分解。遗憾的是，如果我们希望以多对数级质量近似割，目前这些方法都无法突破 $n^{2/3}$ 的更新时间屏障。这引出了以下自然的问题：

是否存在一种全动态算法，可为基于割的优化问题提供多对数近似，同时仅需极短的多项式更新时间？

我们通过设计一种算法，动态维护 Madry [46] 的层次化 j-树分解的一个变体，对此问题给出了肯定回答。因此，对于全动态设置下的基本基于割的优化问题，包括所有点对最小割、最稀疏割、多路割和多重割，我们获得了近似与更新时间之间的新权衡。

我们的结果：动态层次化 j-树分解 为了陈述我们的主要结果，我们首先需要回顾一些记号。我们从 Madry [46] 提出的 j-树概念开始。对于一个整数 $j \geq 1$，如果无向图 H 是由以下两部分的并集构成的，则称其为 j-树：(1) 一个有根森林 $F \subseteq H$，其根集为 R，且满足$|R| \leq j$；(2) 一个核心 $C(H) = H[R]$，即根集 R 上的诱导子图。这个概念推广了树的概念，因为每棵 1-树都是一棵树；并且正如我们将要展示的，它在分解结构的复杂度与维护该结构所需时间之间提供了一种自然的权衡。

接下来，我们将形式化定义由一组 j-树近似保留图的割结构的含义。给定一个带有边容量函数 u 的无向图 $G = (V, E, u)$，以及一组顶点集为 V 的 j-树 $\mathcal{H}$，如果满足以下条件，我们称 $\mathcal{H}$ α-保留 G 的割，概率为 p：

(1) 对于每个割 $(S, V \setminus S)$，都有 $U_G(S) \leq \min_{H \in \mathcal{H}} U_H(S)$，[2] 并且

(2) 对于任意割 S，$\min_{H \in \mathcal{H}} U_H(S) \leq \alpha \cdot U_G(S)$ 的概率至少为 p。

我们指出，$\mathcal{H}$ 并不能保留 G 的指数多个割。相反，它只能保证每个固定的 割以概率 p 得到保留。由于在我们的构造中，$\mathcal{H}$ 将与 $p = 1 - 1/\text{poly}(n)$ 一起使用，因此我们总能通过对这些割应用并集界，使用 $\mathcal{H}$ 保留任意多项式数量的割。

我们的主要贡献是提出了首个全动态算法，用于维护一组分层 j-树，以近似图的割结构，具体如下文定理所述。

定理 1.1. 设 $c \geq 1$ 为常数。给定一个具有 n个顶点的带容量无向图 $G = (V, E, u)$，其容量比为[3] $U = \text{poly}(n)$，并经历边插入和删除；再给定整数 $j \geq 1$ 和 $1 \leq L \leq o\left(\sqrt{\log(n/j)/\log\log n}\right)$，则存在一种数据结构，维护 $\mathcal{H}$ 个由 $O(j)$棵树组成的集合，这些树以 $\widetilde{O}(\log n)^L$-近似保留 G 的割，且成功概率至少为 $1 - 1/n^c$。

该数据结构以 $(n/j)^{2/L} \log^{O(L)} n$ 摊销更新时间显式维护 $\mathcal{H}$，保证$|\mathcal{H}| = \log^{O(L)} n$，并且每棵 $O(j)$-树的容量比 $H \in \mathcal{H}$ 为 $n^{2L+1} U \log^{O(L)} n$。

对于任意常数 $\varepsilon = O(1/L) \in (0,1)$，我们的结果在割近似~方面实现了多对数级质量，且具有 $O(n^{\varepsilon})$ 摊销更新时间。相比之下，当前最先进的动态树流稀疏化构造 [33, 59]基于动态维护一个扩张器层次结构。这是一种自底向上的聚类方法，通过反复执行以下步骤推进：(1) 找到扩张器分解的一种变体，[4] (2) 收缩已识别出的扩张器，(3) 在所得的收缩图上递归处理。然而，与其他层次化自底向上聚类技术 [8, 26, 18],类似，这些构造只能保证次多项式 近似质量。

在静态场景中，Li、Rao 和 Wang [45] 最近的一项成果通过提出一种不同的自底向上分层构造，克服了这一局限，并实现了多对数级近似。然而，他们的方法似乎本质上是顺序式的，难以扩展到动态场景。

---

[2]对于无向带容量图 $G = (V, E, u)$，以及任意顶点子集 $\emptyset \neq S \subset V$，我们用 $U_G(S)$ 表示跨越割 $(S, V \setminus S)$ 的边的总容量。回想一下，图的容量比 是最大边容量与最小边容量之比。[3]

[4]所谓的边界关联 扩张器分解 [33]。

最后，我们强调，我们的算法不使用扩张子图分解，也不进行其动态维护。相反，它仅依赖于快速的静态算法来计算平均伸展较低的生成树 [7], ，并将其用作黑盒子程序。

应用。 我们利用定理 1.1，为全点对最小割、最稀疏割、多路割和多重割等基本的基于割的问题，设计了新的动态算法。概括来说，这些算法都利用了以下事实： j-树的核心相较于底层图的规模小得多，而且在动态场景下，森林中的这些优化问题 j-树有更简单的解法。这些应用的具体保证如下述定理所示。

定理 1.2. 设 $c \geq 1$ 为一个常数。给定一个经历边插入和删除、含 n个顶点的带容量无向图 $G = (V, E, u)$，其容量比为 $U = \mathrm{poly}(n)$ ，以及任意常数 $\varepsilon \in (0,1)$，存在一个$\widetilde{\ }$数据结构，能够以 $O(n^{\varepsilon})$ 摊销时间支持更新，并以至少 $1 - 1/n^c$的概率，为以下查询返回一个多对数级 近似解：

(1) 对任意 $(s, t) \in V^2$： s-t 最小割值（因此也是 s-t 最大流值），即全点对最小割；以及

(2) 最稀疏割值；以及

(3) 多路割、多重割值。

$\widetilde{\ }$ $\widetilde{\ }$前两种查询的摊还时间为 $O(n^{\varepsilon})$。对于多路割，最后一种查询的摊还时间为 $O(kn^{\varepsilon})$，其中 k 是多路割中的终端点数量；对于多重割，摊还时间为 $\widetilde{O}(kn^{\varepsilon} + k^2)$，其中 k 是多重割中指定集合的数量。[5]

$\widetilde{\ }$此前针对全点对最小割的全动态算法，要么实现：（1） $\widetilde{O}(n^{2/3})$的摊销更新时间和查询时间，近似比为 $O(\log n)$ [19], ；要么实现（2） $n^{o(1)}$ 的最坏情况下摊销更新时间和 $\widetilde{O}(1)$ 查询时间，近似比为 $n^{o(1)}$ [33, 59]。我们的结果首次突破了 $n^{2/3}$ 更新时间屏障，同时仍保证多对数级近似。

对于最稀疏割、多路割和多重割，所有先前的全动态算法只能达到 $n^{o(1)}$近似 [33, 59]。因此，我们的算法首次为这些问题在全动态场景下提供了多对数级近似保证。

2 技术概述

基于割的图分解。 许多基于割的问题在树上比在一般图上更容易求解。基于这一洞见，Räcke [52] 证明，对于任意 m边无向图 G，都存在一个多项式$\widetilde{\ }$时间算法，可以计算出一个由 $k := O(m)$ 棵树 $T_1, \ldots, T_k$组成的集合，以及$\{\lambda_i\}_i$ 这些树上的一个概率分布，使得对于任意割 S ，以及从该分布 T 中采样得到的树，均有$U_G(S) \leq U_T(S) \leq O(\log n) \cdot U_G(S)$，且该结论以常数概率成立。这一结果为一大类基于割的问题奠定了基础，使我们能够采用以下自然的算法方法，设计简单高效的近似算法：(a) 从该分布中采样少量树；(b) 在这些树上高效地求解问题。

在其开创性论文中，Madry [46] 引入了 j-树的概念，并指出结构复杂度与计算复杂度之间存在自然的权衡。具体而言，Madry 展示了如何计算一个集合，其中仅包含 $O(m/j)$ 个$\widetilde{\ }$ j-树，并为它们赋予概率分布，使得割能以至多$\widetilde{\ }O(\log n)$ 倍的因子得到保留；这一保证略逊于基于树的情形。重要的是，在 j-树上求解割类问题，通常可通过运行时间上的线性开销，转化为在所谓的核心 C上求解；这些核心是仅由 $j < n$ 个顶点诱导的子图，而非完整的顶点集。

$O(m/j)$ 个 j-树，采用 [64] 的迭代方案；该方案本身是乘法权重更新（MWU）方法 [12]的一种变体。每次迭代中，使用 [5, 7]的算法，根据为 G的边维护的某个长度函数，计算一棵低伸展生成树 T_i 。随后，Madry 运用相当复杂的分桶论证，找出$\widetilde{\ }$一个包含 $O(j \log U)$ 条高拥塞边的集合 E_i，使得移除这些边后，所得图的容量函数

与 (m/j) 类似，该算法在 [46] 中计算集合 $H_1, \ldots, H_k$，$k =$

[5]我们还可以报告相应的近似 s-t 最小割和最稀疏割，所需时间与较小割的规模成正比。

~森林 F_i 具有理想的性质，可将该迭代方案的迭代次数降至 $O(m/j)$。利用得到的森林序列 $\{F_i\}$， [46] 中的算法对每棵森林F_i执行一个三步构造过程：首先构建两个不同的中间图，以此作为最终构造 j-树 $\{H_i\}$的基础。

本文提出了 Madry 构造的一种模块化程度更高的实现。具体而言，我们证明，只需对 Madry 的算法进行细致修改，[6] 就足以采用一种更标准的 MWU 框架应用。我们的方法还通过避免 [46]复杂的分桶方案，简化了整体设计；~取而代之的是，我们直接移除 $O(j)$ 中拥塞最高的边。最后，受[20], 启发，我们证明可以完全绕过 Madry 的三步构造： j-树 H_i 可以利用所谓的无分支的 根集，直接从森林 F_i 中计算得到。

动态割分解。 Chen 等人 [19] 利用顶点稀疏化这一强大范式，动态维护 Madry 的图分解，将图分解为 j-树。他们的算法支持的一项关键操作，是将顶点 v 移入 $O(j)$ 树的核心——也就是说，扩大 $O(j)$-树的核心 C，使其也包含顶点 v ——同时不损害割近似保证。仅凭这一操作就足以支持动态边更新；每当一条边更新时，只需将其端点移入核心即可。

为了支持将顶点移入核心的操作， [19] 中的算法依赖两个主要组成部分：(i) 重新路由所维护的 $O(j)$ 树森林的容量，以及 (ii) 通过添加额外顶点来扩展核心（这可以视为在森林上计算一个击中集），从而使森林中的叶到根路径 $\{F_i\}$ 较短。我们工作的另一项贡献是消除了对 (i) 和 (ii) 的需求；我们完全避免重新路由，并证明随着森林中连通分量数量的增加，所得 j-树的嵌入质量不会降低。此外，我们采用更精细的摊还分析论证，避免了通过添加额外顶点来扩展核心的需要。在第 4 节中，我们证明以下定理。

定理2.1. 设 j 是一个参数， G 是一个初始包含 m 条边、且最多经历 $O(j)$ 次边插入和删除的动态图。存在一种算法，可初始化并维护~一个包含 $i = 1$、2、. . .、 $k = O(m/j)$ 个 $O(j)$-树的集合，其核心为 C_i，森林为 F_i。在任何时刻，均满足：

1. 对所有 i 以及所有割 S： $U_G(S) \le U_{H_i}(S)$； *2.* 对所有割 S： $\frac{1}{k}\sum_{i=1}^{k} U_{H_i}(S) \le \widetilde{O}(\log n) \cdot U_G(S)$； 以及 *3.* 一个

在 G 上执行了 t 次边更新后，对于所有情况，我们有 $|C_i^{(t)}| \le |C_i^{(0)}| + O(t)$ $i.$

算法的总运行时间为 $\widetilde{O}(m^2/j)$。

两层方案。 基于定理 2.1 的保证，考虑一个自然的两层方案：令$\mathcal{H}$ 为一个集合，其中包含从定理 2.1 中均匀随机抽取的 $O(\log n)$ 个 j-树。于是，定理第 2 项结合马尔可夫不等式可推出，集合 $\mathcal{H}$ $O(\log n)$-保留了 G 的割，且其概率为 $1 - 1/\text{poly}(n)$。通过运行来自 [6] 或 [16], 的动态（边）割稀疏化算法，我们~还可以确保图 $H \in \mathcal{H}$ 的核心 C 仅包含 $O(j)$ 条边。

这种随机子采样与稀疏化的结合能够大幅缩减规模。例如，在 G 中进行一次 s–t 最小割查询，可转化为在 $\mathcal{H}$ 中 $O(j)$ 棵树的稀疏核心内求解 s–t 最小割。每个这样的~核心都是一个包含 $O(j)$ 个顶点和 $O(j)$ 条边的图，并且总共只有 $O(\log n)$ 个。因此，查询复杂度降低为在规模为 $\widetilde{O}(j)$ 的图上求解少量子问题。令 $j = n^{2/3}$，即可得到 [19], 的动态算法，该算法实现了 $\widetilde{O}(\log n)$ 近似，且摊销更新和查询时间为 $\widetilde{O}(n^{2/3})$。

定理 2.2. 给定参数 $j \ge 1$，我们可以动态维护一个集合 $\mathcal{H}$，使得

(1) ~$\mathcal{H}$ $O(\log n)$-以概率 $1 - 1/\text{poly}(n)$ 保持 G 的割，且

(2) $\mathcal{H}$ 由 $O(\log n)$ 个 $O(j)$-树构成。

此外， $\mathcal{H}$ 可以在 $\widetilde{O}(m^2/j^2)$ 摊销时间内维护。

[6]我们直接对乘法权重更新预言机调用的宽度 进行界定。

分层方案的初步尝试。改进我们算法更新时间的一种自然尝试是递归地对 $O(j)$树的核心应用定理 2.2，如下所述。假设 G 有 $O(n)$条边，并令 $k = (n/j)^{1/2}$ 为规模缩减参数。首先，我们使用定理 2.2 抽样并维护集合 $\mathcal{H}_1$，其中包含 $O(\log n)$个 $n/k =: j_1$-树。集合 $\mathcal{H}$ $O(\log n)$-以高概率保留 G 的割。接下来，我们对图 $H_1 \in \mathcal{H}_1$ 的核心 C_1 应用动态割稀疏化算法，以维护稀疏化后的 核心 C_1，每个核心仅包含 $O(j_1)$条边。

然后，对于每个 $H_1 \in \mathcal{H}_1$，我们递归地在（稀疏化后的）核心上应用定理 2.2 中的算法 C_1 ofH_1，得到一个集合 $\mathcal{H}(H_1)$，其中包含 O 个（log n） $n/k^2 := j_2$-树。集合 $\mathcal{H}(H_1)$ $\widetilde{O}$（log n）近似保留了稀疏化核心 的割。我们将这些图称为 H_1 的子图，并令 $\mathcal{H}_2 := \bigcup_{H_1 \in \mathcal{H}_1} \mathcal{H}(H_1)$。

现在考虑 $H_1 = C_1 \cup F_1 \in \mathcal{H}_1$ 和 $H_2 = C_2 \cup F_2 \in \mathcal{H}(H_1)$，其中 C_2 是 H_2 的稀疏化核心。图

$$H^{H_1,H_2} := \widetilde{C}_2 \cup F_2 \cup F_1$$

因此是一个 $O(j_2)$-树，其 se 核心为 C_2，森林为 $F_1 \cup F_2$。如果我们将所有这类图汇集到 se t

$$\mathcal{H} := \{H^{H_1,H_2} : H_1 \in \mathcal{H}_1, H_2 \in \mathcal{H}(H_1)\},$$

可以证明， $\mathcal{H}$ $\widetilde{O}(\log^2 n)$ 以 $1 - 1/\text{poly}(n)$ 的概率保留 G 的割。

$\mathcal{H}$ 的初始化耗时 类似地，初始化集合 $\widetilde{O}(n^2/j^2)$ 需要 $O(\log n) \cdot \widetilde{O}(j_1^2/j_2) = \widetilde{O}(n \cdot (n/j)^{1/2})$ 的时间。因此，初始化 $\mathcal{H}$ 的总时间为 $\widetilde{O}(n \cdot (n/j)^{1/2})$，这已经构成了相较于定理 2.2 所实现的 $O(n^2/j)$ 初始化时间的显著改进。

一般来说，通过改为设置 $k = (n/j)^{1/L}$ 并重复此过程 L 次，我们可以构造一个集合 $\mathcal{H}$，使其以 $1 - 1/\text{poly}(n)$ 的概率 $\widetilde{O}(\log n)^L$- 保留 G 的割，并且 $\mathcal{H}$ 由 $O(\log^{O(L)})$ 个 $O(j)$- 树组成；该算法的初始化时间为 $O(n \cdot (n/j)^{2/L} \cdot \log^{O(L)} n)$。

遗憾的是，一旦转向必须处理对 G 的边插入和删除的动态场景，该方法就会失效。再次考虑上面的情形 $L = 2$。对 G 的一次更新可能导致定理 2.1 中 O（j_1）树的核心 C_1 发生多达 $\Omega(j_1)$ 次边插入和删除。即使 我们将 [6] 或 [16] 中的割稀疏器作为黑盒来处理这些更新，稀疏化核心 C_1 仍可能发生 $\Omega(j_1)$ 次边插入和删除。随后，必须将这 $\Omega(j_1)$ 次更新传播到引理 6.10 中运行于 C_1 上的数据结构，以维护集合 $\mathcal{H}_2$。这会产生

$$\Omega(j_1 \cdot j_1^2/j_2^2) = \Omega(n \cdot (n/j)^{1/2})$$

摊还时间，比定理 2.2 中原始的 $\widetilde{O}$（n^2/j^2）更新时间更差。事实上，无论选择何种 j，这一简单的递归方案都会导致 $\Omega(n)$ 的更新时间，从而违背递归的目的。

顶点分裂下的动态割稀疏化来解围。我们已经看到，在 j-树的核心 C 上运行割稀疏化算法至关重要。虽然这些核心仅包含 $O(j)$个顶点，但其中仍可能包含 $\Theta(m)$条边；除非采用边稀疏化过程，否则我们无法进行高效递归。然而，正如我们先前的计算所揭示的，仅动态维护一个稀疏割稀疏器还不够。为了使动态场景下的分层构造高效，我们还迫切需要开发一种能够稀疏化重调整次数的割稀疏化算法。具体而言，虽然对 G 的一次更新可能导致稠密核心 C发生$\Omega(j)$次变化，但我们展示了如何高效维护一个稀疏化核心$\sim C$，使其在处理 C中的更新后仅经历 $O(\text{polylog}(n))$次边更新。因此， C 的重调整次数显著小于底层稠密图 C中移动的边数。这一点至关重要，因为它使我们能够避免在层次结构的更高层级中遭受重调整次数的大幅膨胀。

为实现这一点，我们利用核心的更新结构。尽管 $C(H)$ 每次更新可能经历 $\Omega(j)$ 次边插入和删除，但这些变化具有高度结构化的形式。具体而言，它们对应于顶点分裂，这是一种结构化更新，在若干分层图数据结构 [26, 18, 33, 20]中也发挥了关键作用。

定义 2.3（顶点拆分）。给定一个 $G = (V, E)$ 以及一个顶点集合 $U \subseteq N_G(v)$，满足 $|U| \leq |N_G(v)|/2$。那么，如果顶点集合从 G 变为 $V \cup \{v'\}$，我们就称顶点 v 在 G 中被拆分，使得 $N_G(v) = N_G(v) \setminus U$，并且 $N_G(v') = U$。

换言之，顶点拆分会将 v 的邻居中的一个子集 U 移至新创建的顶点 v'。随后我们确立一个关键的结构性质：对图 G' 执行顶点拆分后得到的最小生成森林 F'，与原始最小生成森林 F 至多相差一条额外的边。

这一关键观察构成了我们方法的主要构件。借助这一观察，我们可以修改 [6] 的动态割稀疏化数据结构。算法维护所谓的束 B，其中包含打包到图 G 中的生成森林。随后，算法交替进行以下操作：对不属于所维护生成森林的边进行子采样，并再次为剩余边维护一个打包。

通过利用生成森林在顶点拆分下的低重配置开销性质，我们证明了可以以极低的重调整次数维护这样的打包。这进而带来了具有类似低重调整次数保证的动态割稀疏化算法。下面给出主定理。鉴于动态割稀疏化算法的多功能性，我们认为这一结果本身就具有独立的研究价值。详细解释请参见第 5 节。

定理2.4. 设 G 是一个初始包含 n 个顶点和 m 条边的全动态图。对于最多包含 m 次边插入、n 次顶点拆分和 D 次边删除的操作序列，存在一种随机化算法，能够显式地在 $O(n)$ 条边上维护一个 G 的 $1\pm\varepsilon$-割稀疏器 G。

该算法在更新过程中经历 $\widetilde{O}(n+D)$ 总时间，并维护 G 的稀疏化器 $\widetilde{O}(n+D)$ 总重配置量（即边删除和边插入的次数）。该算法以高概率正确。

我们再次强调，尽管一次顶点拆分会导致多达 $\Omega(|N(v)|)$ 条边发生移动，稀疏化器中的（摊销）重调整次数仍限制在 $O(\text{多对数}(n))$，且与 $|N(v)|$ 无关。这是一项关键改进：直接使用 [6], 只能保证重调整次数为 $O(|N(v)|)$，正如前文所述，这对我们的目的而言大得无法接受。

应用于定理 2.2 中的图时，我们能够证明以下结果。

引理2.5. 设 H 是一个由定理 2.2 维护的 $O(j)$-树，且 $C = C(H)$ 是其核心。那么，我们可以初始化并维护一个稀疏化核心 $C(H)$，使其在对 G 的完整 $O(j)$ 次更新序列中，总耗时为 $O(m)$，且总重配置量（即在 $C(H)$ 中进行的边插入/删除次数）为 $O(j)$。

现在我们可以重新审视之前的情形 $L = 2$。在 G 中的一次更新首先会触发对定理 2.2 中数据结构的更新，该数据结构维护集合 $\mathcal{H}_1$。处理这次更新需要摊销时间 $O(n^2/j_1^2) = O(n/j)$。随后，图 H_1 的核心 C_1 会发生 $O(1)$ 次顶点拆分。接着，利用引理 2.5 处理这些拆分，以维护稀疏化核心 C_1，摊销时间为 $O(n/j_1)$。处理完更新后，C_1 将发生 $O(\text{polylog}(n))$ 次边插入和删除。最后，这些更新会被转发至定理 2.2 中的数据结构，以维护 $\mathcal{H}_2$ 中的 $O(j_2)$-树。此过程需要摊销时间 $\widetilde{O}(j_1^2/j_2^2) = \widetilde{O}(n/j)$。总之，整个层次结构的摊销更新时间可界定为 $\widetilde{O}(n/j)$——这已经显著改善了两层方案中 $O(n^2/j^2)$的摊销更新时间。

稍加注意，上述分析可扩展到由 $L \geq 3$ 个层级组成的层次结构。与前面类似，关键在于：采用我们在定理 2.4 中提出的低重配置稀疏化器，而非朴素的割稀疏化方法所导致的 $\Omega(j_L j_{L-1} \dots j_1)$ 重调整次数，因此每一层的额外重调整次数仅为 $O(\log^{O(L)} n)$。此外，我们会定期重建层次结构的部分内容，以确保核心保持在所需的大小范围内。我们将在第 6 节中对分层构造进行完整分析。

应用。要回答 s-t-最小割、多路割和多重割查询，只需将相关顶点移入所有 $O(j)$-树 $H \in \mathcal{H}$ 的核心 $C(H)$ 中，然后应用静态算法在核心内解决这些问题。相比之下，要维护对近似最稀疏割值的查询访问，技术上更具挑战性。难点在于，最稀疏割是整个 j-树 H 的全局性质，因此无法仅通过将计算限制在核心内来解决。

我们通过观察到，H 中的任意最稀疏割必定满足以下条件之一，从而应对这一挑战：(a) 仅切割核心边，或 (b) 仅切割森林边。基于这一观察，我们分别维护仅核心边最稀疏割和仅森林边最稀疏割。收到查询时，我们可以使用标准的最稀疏割近似算法，在核心中计算一个近似最稀疏割。同时，我们还必须跟踪动态变化的森林 F中的最稀疏割值，该森林覆盖 $\Omega(n)$ 个顶点。为此，我们以审慎的方式扩展定理 2.2 中的两层方案，从而高效维护该值。最后，收到查询时，我们返回这两个候选值中的较小者。详情请参见第 7 节、第 D 节和第 E 节。

2.1 相关工作

相关技术工作。 我们的动态算法与 [20]中的全动态近似最小比率环数据结构非常相似。从概念上讲，可以将我们的方法视为他们的动态ℓ_∞ 近似问题动态 ℓ_1 优化框架的对应方案，不过我们避免了许多技术难题。他们的数据结构需要一种算法来在顶点拆分时动态维护低重构扩展子图，而我们需要在顶点拆分时维护低重构割稀疏化器。由于他们还需要显式维护短路径嵌入，他们的稀疏化过程比我们的复杂得多。我们的算法在结构上也类似于[41], 中的全动态全点对最短路径数据结构，不过同样地，我们避免了大多数难题，因为我们的割稀疏化器不要求输入图具有低度数。

部分动态最大流。 对于增量图，已知可以在 $O(m^{1+o(1)}\varepsilon^{-1})$ 总更新时间内，为无向 p-范数流 [60] 以及有向最小费用流 [21] 求得 $(1+\varepsilon)$-近似解。Goranci 和 Henzinger [29] 先前的工作已针对有向无权最大流，在 $O(m^{3/2+o(1)}\varepsilon^{-1/2})$ 总更新时间内获得了 $(1+\varepsilon)$-近似解。他们的算法的一种变体也可以扩展为在 $O(n^{5/2+o(1)})$ 总时间内维护精确解，不过仅适用于无向图。同样，对于精确情形，~Gupta 和 Khan [34] 展示了如何在 $O(mF^*)$ 总时间内维护有向最大流，其中 F^*表示插入序列结束时的最大流值。最近，Goranci、Henzinger、Räcke 和 Sricharan [31] 提出了一种算法，可在 $O(m+nF^*\varepsilon^{-1})$ 总时间内求得无向无容量图上的 $(1+\varepsilon)$-近似 s-t 最大流；这是首个针对稠密图、具有多对数摊销更新时间的问题算法。对于递减图，van den Brand、Chen、Kyng、Liu、Meierhans、Gutenberg 和 Sachdeva [59] 提出了一种有向最小费用流算法，可获得 $(1+\varepsilon)$-近似解，且总更新时间为 $O(m^{1+o(1)}\varepsilon^{-1})$。

全局最小割的动态算法。对于增量图，Goranci、Henzinger 和 Thorup [32]~提出了一种动态算法，能够以 $O(1)$ 摊销更新时间精确维护全局最小割的大小。已知的全动态场景结果则只能提供较弱的保证。Goranci、Henzinger、Nanongkai、~Saranurak、Thorup 和 Wulff-Nilsen [30]针对该问题提出了一种精确算法，分别在采用和不采用随机化时，实现了 $O(n)$ 最坏情况更新时间和 $\widetilde{O}(m^{30/31})$ 摊销更新时间。在近似情形下，当前最佳结果由 El-Hayek、Henzinger 和 Li [25] 提出，该算法以 $O(n^{o(1)})$ 摊销更新时间实现了$(1+o(1))$-近似。

基于割的问题的快速静态算法。最稀疏割问题在静态场景中受到了广泛关注 [9, 10, 42, 11, 40, 13, 48]。Arora、Rao 和 Vazirani [14] 的经典工作提出了首个解决该问题的算法，能够在多项式时间内实现 $O(\sqrt{\log n})$-近似。Sherman [54] 指出，要实现该问题的 $O(\sqrt{\varepsilon^{-1}\log n})$-近似，只需~求解 $O(n^{\varepsilon})$个最大流实例。Gomory-Hu 树 [28], 是一种与基于割的问题相关的重要稀疏化器，它是在图的节点上构建的加权树，并保留任意最小 $s-t$ 割的值。Abboud、Krauthgamer 和 Trabelsi [2]证明，它是用于回答 $s-t$ 割查询的近乎最优的数据结构。50 多年来，寻找这类树一直没有比 Gomory 和 Hu 的开创性论文更快的算法；他们通过 $n-1$ 最大流计算构造了该结构。近期的一系列工作将其构造时间缩短至 $O(n^{2+o(1)})$ [3, 1, 44, 65], ，并且还证明全点对最小割问题也能在相同时间内求解。最后，Abboud、Panigrahi 和 Saranurak [4] 利用 [20]近线性时间最大流算法，将 Gomory-Hu 树的构造时间降至 $m^{1+o(1)}$ 。

3 预备知识

图论。 本文中，我们研究的所有图 $G=(V,E_G)$ 均为无向多重图。我们用 $n=|V|$ 表示顶点数，用 $m=|E_G|$ 表示边数。大多数情况下，这些图都带有容量，即在其边上定义了一个函数 $u_G\in\mathbb{R}^E_{>0}$。对于带容量图，我们通常用 U 表示 G的容量比，其定义为 $U=\max_{e,e'\in E_G}u_G(e)/u_G(e')$。在主要结果中，我们假设$U$ 是多项式有界的，即 $U=O(\text{poly}(n))$。对于顶点 $v\in V$，我们用 $N_G(v)$ 表示 v 在 G中的邻域，并将其视为多重集。除非另有说明，我们用于访问和修改图的基本数据结构是图的邻接表表示。

对于图 G的一个割 $(S,V\setminus S)$，我们将 $E_G(S,V\setminus S)$ 定义为 G 中恰有一个端点位于集合 S中的边的集合，并将 $U_G(S):=\sum_{e\in E_G(S,V\setminus S)}u_G(e)$ 定义为跨越该割的总容量。

给定两个图 $G_1=(V,E_1,u_1)$ $G_2=(V,E_2,u_2)$在同一顶点集上，标量 $\alpha,\beta>0$，我们定义

线性组合图 $G := \alpha G_1 + \beta G_2$ 为顶点集 V 上、边集为 $E_G = E_1 \cup E_2$ 且容量为$u_G(e) = \alpha u_1(e) + \beta u_2(e)$的图。

设 $G_1 = (V_1, E_1, u_1)$ $G_2 = (V_2, E_2, u_2)$ 是两个图，且 $V_2 \subseteq V_1$，也就是说，G_2 定义在 G_1 的顶点子集上。然后我们将并集 $G := G_1 \cup G_2$ 定义为顶点集 V_1、边集 $E_1 \cup E_2$ 上的一个（多重）图，其中多重边的容量分别对应于它们在 G_1 或 G_2 中的容量。

对手模型。我们的所有结果都假设存在一个非自适应对手，也就是说，完整的更新序列独立于算法的输出以及算法内部使用的内部随机性。

动态图。我们将按时间 t 编号的一系列图 $G^{(t)}$ 称为动态图。我们关注运行于动态输入图 $G^{(t)}$ 上的算法，其中对所有时刻 t，$G^{(t+1)}$ 与 $G^{(t)}$ 的差异仅为一次边插入或边删除。在本文中，每当我们使用上标 t 时，它始终表示所有数据结构处理完第 t 次对 G 的更新之后的时刻。值得注意的是，尽管输入图 G 每次仅发生一次边插入或删除，但我们也会遇到这样的辅助图：$H^{(t+1)}$ 与 $H^{(t)}$ 之间存在多条边的差异。我们将把将 $H^{(t)}$ 转换为 $H^{(t+1)}$ 所需执行的边插入和边删除总次数称为 H 的重调整次数 。

定义3.1（生成森林）。设 G 是一个图。若子图 F 是 G 的生成森林，则称 F 为 G 的生成森林，当且仅当 F 是一个森林，且 G 的每个连通分量也是 F 中的一个连通分量。若森林 F 仅由一个连通分量构成，则称其为生成树。

定义 3.2（有根森林）。若图 $F = (V, E_F, u_F)$ 的每个连通分量 T 都是满足 $|V(T) \cap R| = 1$ 的树，则称其为根集为 $R \subseteq V$ 的有根森林。G

给定一个顶点 $v \in V$ 在一个连通 d 分量 $T^{v,}$ 我们 l.....根$_F(v)$ d..... h唯一根 $r_v \in R \cap V(T_v)$. 对于有根森林 $F = (V, E_F, u_F)$ 和顶点 $v \in V$，我们将 $v^{\downarrow} \subseteq V$定义为在 F 中属于 v 后代的顶点集合（约定 v 是其自身的后代）。

定义 3.3（递减有根森林）。我们称有根森林序列 $\left(F^{(t)} = (V, E_F^{(t)}, u_F^{(t)})\right)_{t\in\mathbb{N}}$具有根集 $R^{(t)}$ 为递减式，若对所有 $t \geq 1$，都有 $E_F^{(t+1)} \subseteq E_F^{(t)}$，$R^{(t+1)} \supseteq R^{(t)}$ 且 $u_F^{(t+1)} \leq u_F^{(t)}$ 。请注意，两个连续的图之间可能相差不止 1 条边。

图嵌入。我们先给出多商品流的定义。

定义 3.4. 给定一个图 $G = (V, E_G, u_G)$，我们称 $\boldsymbol{f} = \{f_i\}_{1\leq i\leq k}$ 个 s_i-t_i 流在 G 中构成一个多商品流。对于一条边 $e \in E$，我们定义 $|\boldsymbol{f}(e)| := \sum_{i=1}^{k} |f_i(e)|$ 为穿过边 e的流量总量。若对于每个 $e \in E_G$，都有 $|\boldsymbol{f}(e)| \leq \alpha \cdot u_G(e)$，则称多商品流 $\boldsymbol{f}$ 是 α-可行的。根据这个定义

，我们现在可以引入可嵌入性这一概 p可嵌入性t y 两个 graphs.

定义 3.5. 设 $G = (V, E_G, u_G)$和 $H = (V, E_H, u_H)$是具有相同顶点集的两个图，且$\boldsymbol{f} = \{f_e\}_{e\in E_H}$ 是 G中的一个多商品流。那么，如果对于每个$e = (s, t) \in E_H$，流 f_e在 s和 t之间路由 $u_G(e)$ 的流量，我们就称 $\boldsymbol{f}$ 是从 H 到 G的一个嵌入。如果

$\boldsymbol{f}$ 在 G 中是 α-可行的，那么我们称 H 可嵌入到 G 中（记作 $H \preceq_\alpha G$ ）. 每当 H 可 1-嵌入到 G 中时，我们就简称 H 可嵌入到 G 中，并记作 $H \preceq G$。这种嵌入概念使我们能够用一个图的割结构来近似另一个（希望更简单的）图的割结构。

引理 3.6. 设 G, H 是具有相同顶点集的两个图，并假设 $H \preceq_\alpha G$。那么对于任意割$(S, V \setminus S)$，都有 $U_H(S) \leq \alpha \cdot U_G(S)$。

证明。设 $\boldsymbol{f}$是从 H到 G的嵌入。穿过割 $(S, V \setminus S)$ 的流量总量至少等于 H中该割的容量，即 $U_H(S)$。此外，根据假设，我们知道对于每条边 $e \in E_G$，都有 $|\boldsymbol{f}(e)| \leq \alpha u_G(e)$。因此

$$\alpha U_G(S) = \sum_{e\in E_G(S,V\setminus S)} \alpha u_G(e) \geq \sum_{e\in E_G(S,V\setminus S)} |\boldsymbol{f}(e)| \geq \sum_{e\in E_H(S,V\setminus S)} u_H(e) = U_H(S).$$

□

α-割保持集合。

定义 3.7. 设 G 是一个图，$\mathcal{H}$ 是一个与 G 具有相同顶点集的图集合。若集合 α 以概率 p 保持 G 的割，则称其为 G 的割保持集合，当

1. 对于每个割 $(S, V \setminus S)$，都有 $U_G(S) \leq$最小值$_{H\in\mathcal{H}} U_H(S)$，并且

2. 对于任意割 S，最小值$_{H\in\mathcal{H}} U_H$（S）$\leq \alpha \cdot U_G$（S）的概率至少为 p。

注意，如果 $\mathcal{H}$ 以概率 $1-1/n^c$保留了 G 的割，这并不意味着 G 的所有指数级多个割都得以保留。它只能保证对于任意固定的割，该割以概率 $1-1/n^c$得以保留。然而，如果考虑至多包含 n^{c-1} 个割的任意集合，那么其中所有割同时得以保留的概率至少为 $1-1/n$，我们将其称为高概率（*whp*）。特别地，如果 $c \geq 3$，那么 $s-t$ 个最小割的集合将以高概率同时得以保留。

割稀疏化。我们定义图 $H \subseteq G$ 是 G的割稀疏器的含义。

定义 3.8。设 $G=$（V、E_G、u_G）是一个图，$H=$（V、E_H、u_H）是 G的一个子图。设 $\varepsilon > 0$。那么，我们称 H 为 G 的（$1\pm\varepsilon$）-割稀疏器，如果对于 G的每个割（S、$V \setminus S$），都有

$$(1-\varepsilon)\cdot U_G(S) \leq U_H(S) \leq (1+\varepsilon)\cdot U_G(S).$$

请注意，该定义要求上界和下界对所有割都成立，而不仅仅是以高概率对任意一个割成立。

标准标量 *Chernoff* 界。为了证明第 5 节中动态割稀疏化算法的保证，我们需要以下标准 Chernoff 界。

定理 3.9. 设 $X_1, \ldots, X_n$ 是取值于 $\{0,1\}$ 的独立随机变量，且 $X=\sum_{i=1}^n X_i$ 是它们的求和。设 $\mu \geq 0$ 满足 $\mathbb{E}[X] \leq \mu$。那么对于任意 $\delta \geq 0$,

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\delta^2\mu/(2+\delta)\right)$$

动态树数据结构。本文需要高效地维护动态变化的树的顶点和边上的量。以下数据结构将使我们能够做到这一点。

引理 3.10（Link-Cut 树 [58]）。给定一个容量为 u_F 的有根森林 F，存在一种确定性数据结构 $\mathcal{D}_1$，它支持以下操作，摊还时间为 $O(\log n)$:

1. FindRoot（u）：返回 F 中 u 的根$_F$（u）。

2. MakeRoot（u）：将 u 设为包含 u 的树在 F 中的根。

3. PathMin（u）：返回 F 中 u-根$_F$（u）路径上容量最小的边 $e^F_{\min}$（u）。

4. Cut（e）：从 F 中移除边 e。

引理 3.11（欧拉游览树 [36]）。给定一个有根森林 $F=$（V、E）和一个向量 $z \in \mathbb{Z}^V$，存在一种确定性数据结构 $\mathcal{D}_2$，支持以下操作，最坏情况时间为 O（log n）：

1. FindRoot（u）：返回 F 中 u 的根$_F$（u）。

2. MakeRoot(u)：将 u 设为包含 u 的 F 中的根。

3. 割(e)：从 F 中移除边 e。

4. 求和(v)：返回 $\sum_{u\in v^\downarrow} z(u)$。

5. 加法(v, Δ)：更新 $z(v) \leftarrow z(v) + \Delta$。

顶点分裂。构造我们的层次结构需要通过顶点分裂来改变图，定义如下。

定义 3.12（顶点拆分）。给定一个多重图 $G=$（V, E）以及一个满足条件 $|U| \leq |N_G(v)|/2$ 的顶点（多重）集合 $U \subseteq N_G(v)$。如果 G 的顶点集变为 $V \setminus v \cup \{v', v''\}$，我们就称顶点 v 在 G 中被拆分，使得 $N_G(v') = N_G(v) \setminus U$，并且 $N_G(v'') = U$。

下面的引理表明，由于顶点拆分由集合 U 编码，而该集合可能远小于完整邻域 $N_G(v)$，因此可以用 $O(m\log n)$ 次边插入和删除来模拟一系列 n 次顶点拆分。也就是说，我们通过将顶点 v 重命名为 v'、从 v' 中移除邻居 U、插入一个新顶点 v''，并将 U 中的邻居添加为 v'' 的邻居，来模拟一次顶点拆分。

引理 3.13。 *[参见 [21] 中的引理 7.9 和附录 AA] 设 $G=(V,E)$ 是一个动态图，初始包含 n个顶点和 m 条边，随时间进行 m次边更新和 n次顶点分裂。那么，这一系列顶点分裂可以通过 $O(m$对数 $n)$ 次边插入和删除来模拟。*

4 维护割保持 j-树 本节旨在介绍 j-树，说明其用途，然后展示如何构造和维护它们。正如 Chen、Goranci、Henzinger、Peng 和 Saranurak 的工作所开创的那样， [19], 一个 j-树将作为"顶点稀疏化器"。也就是说， j-树允许我们将初始图 G 上某些割问题的求解，从包含 n 个顶点的图，缩减到仅包含j 个顶点的较小图上。

4.1 基本定义 引理 3.6 告诉我们，如果 $G \preceq H \preceq_\alpha G$，那么对于任意割 S 也有$U_G(S) \le U_H(S) \le \alpha \cdot U_G(S)$。因此，如果我们能找到一个结构简单图 H，其顶点集与G相同，并满足 $G \preceq H \preceq_\alpha G$，那么在基于割的计算中，我们可以用 H 作为 G 的代理，同时损失的近似因子至多为 α。在 [46], 中，介绍了如何找到一个由 $i=1$、.....、k 个结构更简单的图 H_i 组成的集合，使得其平均值满足 $G \preceq k^{-1}\sum_i H_i \preceq_\alpha G$。我们将大量借鉴他的方法。

给定一个图 G 和一个根集为 R 的有根森林 $F \subseteq G$[7] ，我们将 cross_F^G 记作边集 $e=(u,v) \in E(G)$，其中端点不在同一棵树中，即 $\text{root}_F(u) \neq \text{root}_F(v)$。给定一条跨越边 $e=(u,v) \in \text{cross}_F^G$，我们定义其投影边 $\text{proj}_F^G(e) := (\text{root}_F(u), \text{root}_F(v))$。此外，我们将 $E_G \setminus \text{cross}_F^G$ 中的边称为内部边。这些边的两个端点都位于 F 的同一个连通分量中。最后，给定 u、v 这两个位于 F 的同一连通分量中的顶点，我们将 $F[u,v]$ 记作 F 中从 u 到 v 的唯一 u 路径。

现在，我们可以介绍本节的核心定义，该定义最初在 [46] 中提出。

定义 4.1（j-树）。如果一个图 $H=(V,E_H,u_H)$ 是一个 j-树，当且仅当它是一个根集为 R、满足 $|R| \le j$ 的有根森林 $F \subseteq H$ 与核心 $C(H) = H[R]$ 的并集，其中核心是根集 R 上的诱导子图。

我们通常用 $H=C(H)$ 和 $\cup F(H)$ 分别表示 H 的核心和森林。

给定图 G 和根集为 R 的有根森林 F，现在定义收缩图 G/F。它是一个顶点集为 R 而非 V 的图，其中每个森林连通分量都收缩为一个顶点。

定义 4.2（收缩图）。给定图 $G=(V,E_G,u_G)$ 和根集为 R 的有根森林 $F \subseteq G$，定义收缩图 G/F 为一个多重图，其顶点集为 $V(G/F) = R$，边集为 $E(G/F) = \{\text{proj}_F^G(e) : e \in \text{cross}_F^G\}$，容量为 $u_{G/F}(\text{proj}_F^G(e)) = u_G(e)$。

诱导的 j-树。现在讨论如何构造 j-树。给定图 G 和容量为 u_F、根集为 R 的森林 $F \subseteq G$，有一种自然的方法来构造由 F和 R 诱导的 $|R|$-树 $H=H(G,F)$，如下所示。我们将 F 的根集 R 用作核心的顶点集。然后，对于每条边 $e \in \text{cross}_F^G$，向核心中添加一条容量为 $u_G(e)$ 的多重边 $\text{proj}_F^G(e)$。此外，对于每条森林边 $e \in E_F$，我们在 H 中保留森林容量 $u_F(e)$。简洁地说，可以写作 $H(G,F) = G/F \cup F$。

对于一个诱导的 j-树 H，我们总能关联一个将 G嵌入到 H中的规范嵌入 $\boldsymbol{f}$。对于每条内部边$e=(u,v) \notin \text{cross}_F^G$，我们通过令 f_e沿树路径 $T[u,v]$发送 $u_G(e)$ 单位的流，将其嵌入到 H中，其中 T是包含 u和 v的 F 的连通分量。对于每条跨越边 $e=(u,v) \in \text{cross}_F^G$，我们令 f_e为沿路径 $F[u,\text{root}_F(u)] \oplus \text{proj}_F^G(e) \oplus F[\text{root}_F(v)$ $v]$发送 $u_G(e)$ 单位流的流，从而将其嵌入，其中$\oplus$ 表示路径拼接。

请记住，根据定义 3.5，只有当对每条边 $e \in E_H$ 都有 $|\boldsymbol{f}(e)| \le u_H(e)$ 时，这种嵌入才有效。要使上述嵌入满足这一条件，显然只需确保对森林边 $e \in E_F$有 $|\boldsymbol{f}(e)| \le u_H(e)$；也就是说，我们需要选择森林容量 $u_F(e)$，使得 $|\boldsymbol{f}(e)| \le u_F(e)$。

为了从另一个方向验证可嵌入性，即验证诱导的 j-树 H 嵌入 G，只需考虑森林边上的容量。这可由以下断言看出。

[7]我们强调，此处的 F 通常不是 G 的生成森林。

断言 4.3。设 $G=(V,E_G,u_G)$是一个图，$F=(V,E_F,u_F)\subseteq G$是一个根集为R的有根森林。设 $H=H(G,F)$是诱导的 $|R|$-树。假设 $G\preceq H$通过规范嵌入，并令$\alpha:=\max_{e\in E_F}u_F(e)/u_G(e)$。则 $H\preceq_{2\alpha}G$。此外，该嵌入在边 $e\notin F$上仅路由 $u_G(e)$流，而在其他地方至多路由 $2u_F(e)$流。

证明。我们指定一个从 H 到 G 的嵌入 $\{f_e\}_{e\in E_H}$。首先，对于每个 $e\in E_F$，我们也有 $e\in E_G$，因此只需令 f_e 为流，使 $u_F(e)$ 单位的流经由 G 中的边 e 路由。其次，对于每条容量为 $u_H(e)$ 的（多重）边 $e=(u,v)\in C(H)$，存在唯一一条容量为 $u_G(e')=u_H(e)$ 的边 $e'=(u',v')\in E_G$，使得 $e=\mathrm{proj}_F^G(e')$，且 $u=\mathrm{root}_F(u')$ $v=\mathrm{root}_F(v')$。因此，令 f_e 沿着 G 中的路径 $F[u,u']\oplus e'\oplus F[v,v']$ 发送 $u_G(e')$ 单位的流。

至此，嵌入的定义完成。显然，任意边 $e\in E_G\setminus E_F$ 上通过的流量至多为 $u_G(e)$ 单位。因此，只需考虑森林边 $e\in E_F$ 的拥塞。但请注意，按照构造，通过任意边 $e\in E_F$ 的流量总量为 $u_F(e)+|\boldsymbol{f}(e)|\le 2u_F(e)\le 2\alpha u_G(e)$，其中 $\boldsymbol{f}$ 是从 G 到 H 的规范嵌入，并且我们利用了假设 $|\boldsymbol{f}(e)|\le u_F(e)$。□

动态诱导 $|R|$-树与顶点拆分 给定一个动态图 G和一个以根集 R为根集的递减有根森林 F，我们希望维护诱导的 $|R|$-树 $H(G,F)=G/F\cup F$。随着森林的根集 R（因此连通分量的数量）不断增长，收缩图 G/F的顶点集和边集会发生显著变化。假设森林 F' 与森林 F 的区别在于，从 F 中以 r 为根的连通分量 T 移除一条边 e，且根集按 $R'=R\cup\{u\}$增长。那么，在 F'中，连通分量 T会变为两个连通分量 T_r、T_u，其中 T_u的根为 u，而 T_r的根为 r。因此，更新后的收缩图 G/F'现在具有顶点集 $V(G/F')=R'=R\cup\{u\}$。其边集变化如下。一方面，所有满足 $x\in T_u$和 $y\in T_r$的边 $e=(x,y)$ 现在都是跨边，因此每条这样的边都对应一条插入到 G/F'中的投影边 $\mathrm{proj}_F^G(e)$。

另一方面，一条具有 $x\in T_u$ 和 $y\notin T_u\cup T_r$ 的边 $e=(x,y)$ 之前会导出一条投影边 $\mathrm{proj}_F^G(e)=(r,\text{根}_F(y))$。然而，在新的森林 F' 中，我们有根$_{F'}(x)=u$，因此该投影边变为 $\mathrm{proj}_{F'}^G(e)=(u,\text{根}_F(y))$。因此，我们需要通过将所有此类投影边的端点从 r 移至 u 来更新边集。这正是定义 3.12 中顶点拆分的定义所描述的情形，并且将在后续章节中发挥关键作用。总而言之，当一个有根森林发生边删除时，相应的 $|R|$-树可以通过一次边删除、一次顶点拆分以及可能的多次边插入来更新。

4.2 维护 j-树理想情况下，我们希望找到一个具有容量 u_F 和根集 R 的有根森林 F，使得 $|R|\approx j$，其中 $j\ll n$ 和 $G\preceq H(G,F)\preceq_\alpha G$，对于某个较小的 α，例如 $\alpha=\widetilde{O}(\log n)$[8]。根据断言 4.3，为了实现这一点，我们需要每条森林边 $e\in E_F$ 的容量至多在一个不超过 $\sim O(\log n)$ 的因子范围内。另一方面，如果我们将 G 嵌入 H，G 的许多边可能会沿森林边嵌入，这迫使我们选择较大的森林容量 u_F。本质上，将 G 嵌入 H 与将 H 嵌入 G 之间存在张力。因此，对于单个 j-树 H，满足 $G\preceq H\preceq_\alpha G$ 这一目标可能过于苛刻。

为解决这一问题，Madry [46] 证明，在静态图中，如果不试图寻找单一的 j-树，而是对多个这样的树取平均，就能获得更好的嵌入质量。也就是说，他展示了如何找到一系列根集 R_i 相对较小的有根森林 F_i，使得由此导出的 $|R_i|$-树的平均嵌入能以较小的 α嵌入到 G 中。类似地，我们将证明以下定理。在附录 B 中，我们给出一个众所周知的论证，证明——至少在考虑诱导 j-树时——为保证 α-可嵌入性所需的 j-树数量，与下文定理中的数量几乎是紧的。

定理4.4. 给定参数 j 和动态图 $G=(V,E_G,u_G)$，该图初始有 m 条边，容量比为 U，并经历至多 $O(j)$ 次边插入和删除。则存在一种算法，能够初始化并$\widetilde{\ }$维护一个由 $i=1$、2、.....、$k=O(m/j)$ 个递减有根森林组成的集合 $F_i=(V,E_{F_i},u_{F_i})$，这些森林的根集为 R_i，并维护相应的诱导 $|R_i|$-树 $H_i=H(G,F_i)$。

初始时，对于所有 i，我们有 $|R_i|\le O(j)$。此外，在任何时刻，均有：

1. $G\preceq H_i$ *for all* $i=1,2,\dots,k$,

[8]在本文中，如果存在某个常数 $C>0$ 使得 $g(n)=O(f(n)\log^C f(n))$，我们就记作 $g(n)=\widetilde{O}(f(n))$；如果存在一个常数 $C>0$ 使得 $g(n)=O(\log^C f(n))$，则记作$g(n)=\mathrm{polylog}(f(n))$。

2. 对于某个 $\alpha = \widetilde{O}(\log n)$，有 $\frac{1}{k}\sum_{i=1}^{k} H_i \preceq_\alpha G$，

3. 每个 H_i 的容量比为 $O(mU)$，

4. 在对 G执行了 t 次边更新后，对于所有 i，都有 $|R_i^{(t)}| \le |R_i^{(0)}| + O(t)$.

算法的总运行时间为 $\widetilde{O}(m^2/j)$。此外，每个收缩图 G/F_i （以及核心 $C(H_i)$）的总变化包括 $O(j)$ 次顶点分裂、$O(j)$ 次边删除，以及 $O(m)$ 次边插入。

请注意，上述定理告诉我们，在初始化一组（诱导的） $O(j)$-树之后，我们可以在一系列 $O(j)$次更新中维护它，同时略微扩展根集 R ——从而扩展核心 $C(H_i)$。然后，如前一节所述，扩展根集 R_i 对应于边插入，尤其对应于收缩图 G/F_i中的顶点分裂。我们将在下一节给出该定理的完整证明。

目前请注意，虽然图 $C(H_i) = G/F_i$ 只有 $|R_i| = O(j)$ 个顶点，但可能包含 $\Omega(m)$ 条（多重）边。因此，此时边数并未显著减少。为此，我们还需要对核心运行边稀疏化过程。我们将稀疏化核心定义如下。

定义 4.5（稀疏化核心）。给定一个具有核心 $C(H)$ 且容量比为 U 的 j-树 H，若图$\sim C(H) \subseteq C(H)$ 的质量为 β，则称其为一个稀疏化核心，当且仅当

~ ~1. $C(H)$ 是一个边数至多为 $O(j \log U)$ 的图，

~2. $C(H)$ 的容量比为 $\mathrm{poly}(n)U$，并且

3. 对于 $C(H)$ 中的所有割 $(S, V \setminus S)$，均有

$$U_{C(H)}(S) \le U_{\widetilde{C}(H)}(S) \le \beta \cdot U_{C(H)}(S).$$

~我们通常也称 $C(H)$ 为 H 的 β-近似稀疏化核心。

~命题 4.6. 设 $H = C \cup F$ 是一棵 j-树，其核心为 $C = C(H)$，森林为 $F = F(H)$，且设 C 是一个质量为 β 的稀疏化 ~ ~ 核心。那么 $H := C \cup F$ 是一棵 j- 树，对每个割 $(S, V \setminus S)$，均满足

$$U_H(S) \le U_{\widetilde{H}}(S) \le \beta \cdot U_H(S)$$

~证明。 H 是一棵 j-树，这一点显而易见。现在考虑一个割 S，首先注意到，由于森林容量 ~ ~在 H 和 H中相同，因此割中森林边的总容量在 H 和 H中也相同。因此，我们只需考虑核心边；但根据 C 作为质量为 β的稀疏化核心所提供的保证，还有$U_C(S \cap V(C)) \le U_{\widetilde{C}}($ $S \cap V(C)) \le \beta \cdot U_C(S \cap V(C))$。 □

通过运行定理 5.1 中的割稀疏器，我们可以以低重配置开销维护稀疏化核心。请注意，图 G/F_i 会随着时间推移通过顶点分裂而不断增大。此外，尽管数据结构在初始图 G上仅处理 $O(j)$ 次更新，核心 $C_i(H)$ 最多仍可能发生 $\Omega(m)$ 次额外的边插入。因此，(a) 稀疏化器能够以低重配置开销处理顶点分裂，以及 (b) 其重配置次数不依赖于边插入次数，这两点至关重要。

推论 *4.7.* 设 $H = H(G, F)$ 是由定理 *4.4* 维护的 $O(j)$-树，且 $C = C(H)$ 是其核心。~那么，通过运行定理 *5.1*，我们可以初始化并维护质量为 *2* 的稀疏化核心 $C(H)$，覆盖对 G 进行的完整 ~ 更新序列 $O(j)$，总耗时为 $O(m \log U)$，总重配置量（即 ~ ~$C(H)$ 中的边插入/删除次数）为 $O(j \log U)$。对于算法开始前指定的任意常数 $c > 0$，该算法以概率 $1 - 1/n^c$ 正确。

4.3 定理 4.4 的证明 Madry 在 [46] 中展示了如何从头初始化满足定理 4.4 性质的 j-树。然而，将他的初始化过程完全作为黑箱使用，不便于分析动态算法。因此，我们只使用他的定理来初始化 G 的某个生成树集合 T_i，该集合包含一个规模较小的特殊高拥塞边子集 $S_i \subseteq E_{T_i}$。利用这些树，可以直接构造相应的 j- 树，而且它们的维护也很简单。我们首先介绍一种简单的方法，为 G 的生成树 T 选择容量 u^T，使得 $G \preceq T$。

给定一个图 $G=(V,E_G,u_G)$、 G 的一棵生成树 T，以及一条边 $e=(u,v)\in E_G$，在 T 中将 $u_G(e)$ 单位的流从 u 路由到 v 有唯一的方法。我们只需沿唯一的树路径$T[u,v]$路由流。这构成了从 G 到 T 的规范嵌入。对 G 的每条边都这样做，会在 T 中产生一个多商品流，记作 $\boldsymbol{f}_T$。接着，我们定义边 $e\in E_T$ 的诱导树容量 u^T[9] 为$u^T(e):=|\boldsymbol{f}_T(e)|=\sum_{e'=(u,v)\in E_G:e\in T[u,v]}u_G(e')$。请注意，根据构造， $G\preceq T$，并且 T 的容量比为 $O(mU)$，其中 G 的容量比为 U。接下来，我们给出一个度量，用于衡量 T 嵌入回 G 的效果。

定义 4.8。 给定 G 的一棵生成树 T，我们将边 $e\in E_G$ 的拥塞定义如下 :

$$\mathsf{cong}_G^T(e)=\begin{cases}u^T(e)/u_G(e) & \text{if } e\in E_T\\ 1 & \text{otherwise.}\end{cases}$$

现在我们可以陈述 Madry 构造的核心部分。我们在 C 节给出一个自成一体的证明。

定理 4.9. 设 $G=(V,E_G,u_G)$ 是一个图， j 是一个参数。那么，在时间 $\widetilde{O}(m^2/j)$ 内，可以~构造一列生成树 $i=1,2,\ldots,k=O(m/j)$，记为 $T_1,\ldots,T_k$，它们属于 G，并配以边集 $S_i\subseteq E_{T_i}$，满足 $|S_i|\le j$，使得对于每条边 $e\in E_G$

$$\frac{1}{k}\sum_{i:e\notin S_i}\mathsf{cong}_G^{T_i}(e)\le\widetilde{O}(\log n).$$

当然，根据构造可知，对于所有 i，都有 $G\preceq T_i$。上述定理实质上表明，如果允许从每棵树 T_i中移除一个小边集 S_i，那么这些树的平均也能通过恒等嵌入良好地嵌入回 G。与 [46], 中的构造类似，这个边集 S_i 包含拥塞 $\mathsf{cong}_G^{T_i}(e)$ 较大的边 $e\in T_i$，不过我们以一种略有不同、且可以说更简单的方式构造了它。

自然地，移除边 S_i 后，图 G 将无法嵌入由此得到的森林 F_i 中；关键在于，经过一些额外修改后，可以证明它仍能嵌入由森林 F_i 所诱导的 $O(j)$-树中。接下来我们就要这样做，也就是说，现在我们将展示如何利用这些树 T_i 和集合 S_i，快速构造满足定理 4.4 所述性质的 $O(j)$-树集合。

我们首先介绍 [22] 中使用的无分支集的定义。它们有助于使算法具有动态性。

定义 4.10. 给定一棵有根树 T 和一个顶点集合 $R\subseteq V(T)$，如果对于任意 u、$v\in R$， u 和 v 在 T 中的最近公共祖先也属于 R，则称集合 R 为无分支集合。

注 4.11. 对于任意非无分支的集合 $R\subseteq V(T)$，可以在 $O(n)$ 时间内构造一个满足 $|R'|\le 2|R|-1$ 的新无分支集合 $R'\supseteq R$。

现在，我们可以概述如何根据定理 4.9 给出的树 T_i 构造 $O(j)$-树。首先，我们将描述构造过程，暂不论证其效率，随后立即对此进行说明。我们先以参数 j运行~定理 4.9，得到一个包含 Om/j个树 T_i 及其对应边集 S_i的序列，并满足 $|S_i|\le j$。然后按如下步骤进行。

指定根集：我们任意选取每棵树 T_i 的根，并如下构造森林 F_i 及其根集 R_i。对于每条边 $e\in S_i$，我们将其两个端点 u， v 都加入 R_i。根据定理 4.9 的保证，此时我们有 $|R_i|=O(j)$。然后，我们再向 R_i 中加入至多 $O(j)$ 个额外顶点，使 R_i 成为相对于 T_i的无分支集合。由此确定初始根集 R_i。

指定森林：由 T_i 和 R_i，我们按如下方式构造森林 F_i。若两个顶点 $u,v\in R_i$之间的树路径 $T_i[u,v]$ 上不包含 R_i中的其他顶点，则称它们相邻。令 $\mathcal{P}_i$ 为相邻顶点 $u,v\in R_i$之间的路径$T_i[u,v]$ 集合。注意， $|\mathcal{P}_i|=O(|R_i|)=O(j)$[10]。现在，给定一条路径 $p\in\mathcal{P}_i$，令 e^p*min* 为路径 p 上容量 $u^{T_i}(e)$ 最小的边 $e\in p$ （若有并列，则以任意但一致的方式打破并列）。然后，通过移除所有边 $e^{p,T_i}{}_{\min}$，从 T_i 得到 F_i，即令 $E_{F_i}=E_{T_i}\setminus\{e^{p,T_i}{}_{\min}:p\in\mathcal{P}_i\}$。注意，因为

[9]我们沿用 u^T 中的记号 [46], 以强调容量由树 T[10]所诱导。这一点可以通过观察得出：在 T_i 中的每条叶到根路径上， $\mathcal{P}_i$ 中的路径数量与给定叶到根路径上的顶点 $r\in R_i$ 数量相同。

对于任意边 $e=(u,v)\in S_i$，我们有两个 $u,v\in R_i$，特别地有 $E_{F_i}\subseteq E_{T_i}\setminus S_i$。最后，我们将森林容量设为 $u_{F_i}(e)=2u^{T_i}(e)$，并将 $H_i=H(G,F_i)=G/F_i\cup F_i$。需要强调的是，整个算法过程中森林容量保持不变。至此，图 H_i的初始化完成。

维护：现在我们描述如何在针对 G 的边更新过程中维护森林 F_i 和根集 R_i。假设一条边 $e=(u,v)$ 被插入到 G 中，或从中删除。然后，我们通过将两个端点 u、v 加入 R_i，并可能再添加 2 个额外顶点，使 R_i 相对于 T_i 保持为无分支集合，从而更新根集R_i。这将引入 $O(1)$ 条新路径 $p\in\mathcal{P}_i$。对于这些新路径 p 中的每一条，我们再次找出边 e^{p,T_i}min，并将其从 F_i 中移除。然后，我们相应地更新诱导的 $|R_i|$-树$H_i=H(G,F_i)$。请记住，$H_i=G/F_i\cup F_i$，因此，为了更新 H_i，我们需要更新收缩图 G/F_i。注意，完成此操作后，该边的两个端点 u,v 都位于核心 $C_i=C(H_i)$ 内。因此，我们只需将边 (u,v) 插入更新后的核心 $C(H_i)$，或从中删除。我们稍后将证明，在核心内直接处理此类边更新不会降低嵌入质量。

效率：虽然我们现在已经介绍了如何维护森林 F和图 $H_i=H(G,\ F_i)$，但仍需讨论如何高效地实现所有操作。首先，为了找到新边 $e^{p,\ T_i}{}_{\min}$，我们将使用引理 3.10 中的动态树数据结构 $\mathcal{D}_1$。它能让我们在 $O(\log n)$ 摊还时间[11]内识别每条这样的边。

接下来，我们需要证明能够高效地维护图 $H_i=H(G,F_i)$，这归结为高效地维护收缩图 G/F_i。我们在算法 1 的 AddTerminal(w) 操作中按如下方式实现这一点。如第 4.1 小节末尾所述，如果森林 F' 与森林 F 的区别在于移除了一条边 e，且根集增长为 $R'=R\cup\{w\}$，那么 F_i 中以 r 为根的连通分量 T 现在由 F_i'中的两个连通分量 T_r,T_w组成，其中 T_w以 w为根，T_r以 r为根。因此，图 G/F_i'会发生以下变化：(a) 对于满足 $x\in T_u$和 T_r的 $e=(x,y)$，插入边 $\mathrm{proj}_F^G(e)$；(b) 如果边 (r,y) 对应于投影边 $\mathrm{proj}_F^G((x,y))$，且 $\mathrm{root}_F(x)=r$，但现在 $\mathrm{root}_{F'}^G(x)=w$，则将边 (r,y) 的端点移至 (w,y)。以这种方式移动边恰好对应于顶点拆分。图 4.1 还以图示方式展示了森林和核心可能发生的变化。

为了高效地执行这些顶点拆分，我们希望相应地模拟这些操作，然后调用定理 3.13。为此，对于 F_i 的每个连通分量 T 和顶点 $v\in T$，我们维护与以 v为根的子树 $v^{\downarrow}$ 相邻的跨越边 $e\in\mathrm{cross}_F^G$ 的数量，即满足$e\cap v^{\downarrow}\neq\emptyset$的跨越边 e 的数量。我们可以使用引理 3.11 中的 O数据结构，在 n对数最坏情况时间内完成这一操作。随后，若一次边删除将树 T 分成两棵树 T_r 和 T_w，通过调用 $\mathcal{D}_2$.Sum(w)和 $\mathcal{D}_2$.Sum(r)，我们可以找到 $\min\{|E_G(T_r,\ V\setminus(T_r\cup T_w))|,\ |E_G(T_w,\ V\setminus(T_r\cup T_w))|\}$，并且只需移动较小一半中的边。这对于调用定理 3.13 至关重要；由此可知，为维护由 F_i 诱导的$O(j)$-树而需要移动的边，可以在总计 $O(m$对数 $n)$时间内完成。伪代码见算法 1。

在开始定理 4.4 的形式化证明之前，我们先给出两个关键引理。第一个引理表明，在初始化之后，$G\preceq H_i$，并且扩展根集 R_i（从而扩展核心 G/F_i）能够保持可嵌入性。

引理 4.12. 设 G 是一个图，T_i 和 S_i 是定理 4.9 中的树及其对应边集。设 R_i 是一个无分支根集，满足 $R_i\supseteq\{v:$v 是边的端点 $e\in S_i\}$，并设 F_i 是对应的森林，其边集为 $E_{F_i}=E_{T_i}\setminus\{e^{p,T_i}{}_{\text{最小}}:\ p\in\mathcal{P}_i\}$。设 $H_i=H(G,F_i)$ 是诱导的 $|R_i|$-树，其森林容量集设为 $u_{F_i}(e)=2u^{T_i}(e)$。那么 $G\preceq H_i$ 可由规范嵌入得到。

备注 4.13. 注意，在我们的动态算法中，根集 R_i 和森林 F_i 始终满足该定理的条件。若令 $G^{(0)}$ 表示初始图，并令 $F_i^{(t)}$ 表示数据结构处理完 t 次更新后得到的森林 F_i，则由引理 4.12 可知 $G^{(0)}\preceq H(G^{(0)},F_i^{(t)})$。这是使该构造适用于动态场景的关键性质，因为据此我们还可以推得 $G^{(t)}\preceq H(G^{(t)},F_i^{(t)})$。

证明。首先回忆一下，规范嵌入的定义如下。对于每个连通分量$T\in F_i$和内部边 $e=(u,\ v)$，满足 $u,v\in T$，令 f_e为沿树路径 $T[u,v]=T_i[u,v]$传送 $u_G(e)$ 单位流的流。对于所有其他边 $e=(u,\ v)\in\mathrm{cross}_{F_i}^G$，令 f_e为沿路径 $F_i[u,$根$_{F_i}(u)]\oplus\mathrm{proj}_{F_i}^G(e)\oplus F_i[$根$_{F_i}(v)\ v]$传送 $u_G(e)$ 单位流的流。

[11]因此，我们的算法不会显式维护集合 $\mathcal{P}_i$。我们只需维护正确的边集 $E_{F_i}=E_{T_i}\setminus\{e^{p,T_i}{}_{\text{最小}}:\ p\in\mathcal{P}_i\}$。

现在注意，任何边 $e=(x,\ y)\in E_G$，若 $x,\ y$ 位于同一个连通分量 T 中，则 F_i 仍然会沿树路径 $T_i[x,y]$ 进行路由。因此，嵌入路径在 H_i 中不同于树路径的唯一边，是边 $e\in \mathsf{cross}_{F_i}^{G}$，即满足存在路径 $p\in\mathcal{P}_i$ 且 $e^{p}{}_{\text{最小}}\in T_i[x,y]$ 的边 $e=(x,\ y)$。

现在令 $p\in\mathcal{P}_i$ 为一条与 T 的连通分量相交的路径，其中 T 是 F_i 的连通分量，即 $p\cap T\neq\emptyset$。根据定义，我们知道 $\sum_{e=(x,y):e\in T_i[x,y]} u_G(e)=u^{T_i}(e^{p,T_i}\min)$，也就是说，在树 T_i 中，经过 $e^{p,T_i}\min$ 的流量总量恰好等于边 $e^{p,T_i}\min$ 的容量。现在注意，对于任意边 $e'\in p\cap T$，满足 $e'\in f_e$ 但不满足 $e'\notin T_i[x,y]$ 的潜在边 $e=(x,y)$，恰好就是满足 $e^{p,T_i}\min\in T_i[x,y]$ 的边。但这意味着嵌入 $\boldsymbol{f}$ 经过 e' 发送的流量至多为 $u^{T_i}(e')+u^{T_i}(e^{p,T_i}\min)\le 2u^{T_i}(e')$ 个单位。在最后一个不等式中，我们使用了如下事实：根据 $e^{p,T_i}\min$ 的定义，有 $u^{T_i}(e^{p,T_i}\min)\le u^{T_i}(e')$。

第二个观察是，在两个端点 u,v 都已位于 j-树的核心中时，插入/删除边 $(u,\ v)$ 不会使规范嵌入失效。

引理 4.14. 设 G 是一个图，$F\subseteq G$ 是一个以 R 为根集的有根森林，并设 $H=H(G,F)$ 是由 F 和 R 诱导的 $|R|$-树。进一步假设 $G\preceq H$ 通过规范嵌入。则若 G' 是一个图，它与 G 的差异仅在于插入/删除边 $e=(u,v)$，其中 $u,v\in R$，那么由 F 和 G' 诱导的 $|R|$-树 $H'=H(F,G')$ 仍满足 $G'\preceq H'$ 通过规范嵌入。

证明。若 $\{f_e\}_{e\in E_G}$ 是将 G 嵌入 H 的规范嵌入，且 $e=(u,v)$ 是一条边，其中 $u,v\in R$，那么只有流 f_e 会经过这条边，其他流均不会使用这条边。因此，从 G 和 H 中同时删除该边后，剩余的流仍构成有效的嵌入。若将边 $e=(u,v)$（其中 $u,v\in R$）插入 G，并且 $C(H)$，我们可以通过添加流 f_e 来扩展该嵌入，该流将 $e\in G$ 映射到同一条边 $e\in C(H)$，其容量为 $u_G(e)=u_H(e)$。 □

现在可以证明主定理，伪代码见算法 1。

定理 4.4 的证明。正确性：由于树 T_i 的容量比至多为 mU，因此递减森林 $F_i\subseteq T_i$ 在容量为 $2u^{T_i}(e)$ 时的容量比也至多为 $2mU$。因此，$H_i=G/F_i\cup F_i$ 的容量比至多为 $2mU$。

我们也已经看到，根集 R_i 最初满足 $|R_i|\le O(j)$。每次更新后，根集至多增长 $O(1)$，也正如所承诺的那样。

接下来，我们证明对于所有 i，规范嵌入始终满足 $G\preceq H_i$。假设 G 已接收 t 次边更新。我们的目标是证明 $G^{(t)}\preceq H_i^{(t)}=H(G^{(t)},\ F_i^{(t)})$。首先注意，根据引理 4.12，有 $G^{(0)}\preceq H(G^{(0)},\ F_i^{(t)})$。由根集 $R_i^{(t)}$ 的构造可知，若 $G^{(t)}$ 由 $G^{(0)}$ 通过插入/删除边 $e_1=(u_1,\ v_1)\ldots,\ e_t=(u_t,\ v_t)$ 得到，则对所有 $l=1,\ \ldots,\ t$，均有 $u_l,\ v_l\in R_i^{(t)}$。于是，反复应用引理 4.14 可得 $G^{(t)}\preceq H(G^{(t)},\ F_i^{(t)})$，正如所需。

还需证明，对于某个 $\alpha=\widetilde{O}(\log n)$，我们在所有时刻都有 $k^{-1}\sum_{i=1}^{k}H_i\preceq_\alpha G$。考虑断言 4.3 中指定的从 H_i 到 G_i 的嵌入 $\boldsymbol{f}_i$。该断言表明，对于每条边 $e\in E_G$，若 $e\notin F_i$，则有 $|\boldsymbol{f}_i(e)|=u_G(e)$；否则有 $|\boldsymbol{f}_i(e)|\le 2u_{F_i}(e)$。令 $k^{-1}\sum_{i=1}^{k}\boldsymbol{f}_i$ 为从 $k^{-1}\sum_{i=1}^{k}H_i$ 到 G 的嵌入，并注意到，根据断言 4.3，对于每个 $e\in E_G$

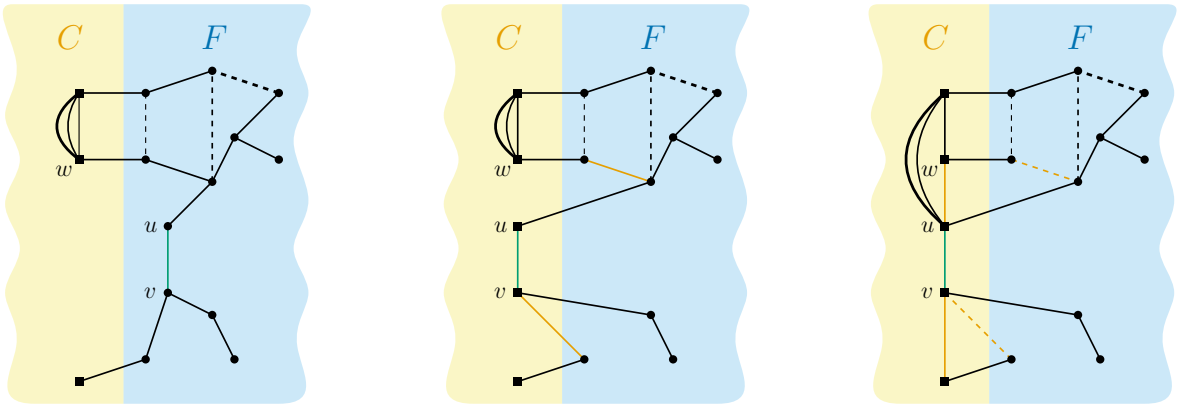
$$\begin{aligned}\frac{|\boldsymbol{f}(e)|}{u_G(e)}&=\frac{1}{k}\sum_{i=1}^{k}\frac{|\boldsymbol{f}_i(e)|}{u_G(e)}\le 1+\frac{1}{k}\sum_{F_i\ni e}\frac{|\boldsymbol{f}_i(e)|}{u_G(e)}\le 1+\frac{1}{k}\sum_{F_i\ni e}2\frac{u_{F_i}(e)}{u_G(e)}=1+\frac{1}{k}\sum_{F_i\ni e}4\frac{u^{T_i}(e)}{u_G(e)}\\&=1+\frac{1}{k}\sum_{F_i\ni e}4\mathsf{cong}_G^{T_i}(e)\le 1+\frac{1}{k}\sum_{i:e\notin S_i}4\mathsf{cong}_G^{T_i}(e)\le\widetilde{O}(\log n),\end{aligned}$$

其中，倒数第二个不等式是因为 $\{F_i:\ e\in F_i\}\subseteq\{i:\ e\notin S_i\}$，而最后一个不等式则源于定理 4.9 的初始化保证。

这样，剩下要证明，在一系列 $O(j)$ 次更新中，更新所有图 H_i 和森林 F_i 的总时间为 $\widetilde{O}(m^2/j)$。我们证明，维护单个图 H_i 及其森林 F_i 的总时间为 $O(m)$，由此即可得出结论，因为这样的图共有 $O(m/j)$ 个。

函数 AddTerminal(w) 首先更新森林 F_i，将顶点 w 纳入其根集：在 F_i 中沿 w 至根 $r_{F_i}(w)$ 的路径 p 上找到并删除最小容量边 $e^{p,\ T_i}{}_{\text{最小}}$。然后，它相应地更新收缩的

2026/4/1 内容由豆包 AI 生成，请仔细甄别



(a) (b) (c)

图 4.1：示意核心 C 和森林 F 在更新后可能发生的变化。虚线边表示属于 G 但不属于 H的边。绿色边表示新插入到 G 中的边，数据结构必须对其进行处理。橙色边表示边 $e^F_{\min}(u)$ 和 $e^F_{\min}(v)$。在 (b) 中，顶点 u、v被添加为集合 R中的根。在 (c) 中，最小容量边已从森林中移除，并作为投影边添加到核心中。此外，核心中的两条投影边的端点从 w 移动到了 u。

图 G/F_i，使得所诱导的 $|R_i|$-树 $H(G, F_i) = G/F_i \cup F_i$ 仍然有效。我们证明，对于给定的森林 F_i，执行所有 AddTerminal(·) 调用所需的总运行时间为 $O(m)$。我们还证明，在对 G进行的完整 $O(j)$ 次更新序列中，收缩图 G/F_i（或等价地，核心 C_i）会发生 $O(j)$ 次顶点拆分、$O(j)$ 次边删除和 $O(m)$ 次边插入。

我们首先注意到，AddTerminal(·) 调用的总次数上界为 $O(j)$。在每次调用中，我们使用引理 3.10 中的数据结构 $\mathcal{D}_1$，在 $O(\log n)$ 时间内找到边 $e^{p、T_i}_{\min}$，因此更新森林 F_i的总时间为 $O(j \log n)$。我们还注意到，在整个算法过程中，我们使用引理 3.11 中的数据结构 $\mathcal{D}_2$ 维护向量$z(v) = |\{e \in \text{cross}^G_{F_i} : v \in e\}|$。我们将在下文证明这总共需要 $\widetilde{O}(m)$ 时间。现在请注意，由此可知，对于任意时刻具有根 r的连通分量 $T_r \in F_i$，都有$\mathcal{D}_2$.求和$(r) = |E_G(T_r、V \setminus T_r)|$。

接下来，考虑算法 1 第 18 行中的第二个 for 循环。只有当在此前所有时刻，边 $e = (x, y)$ 的两个端点 x, y 都属于 F_i 的同一个连通分量，但删除 $e^F_{\min}(u)$ 后，我们现在有 $x \in T_u$ 和 $y \in T_r$ 时，才会考虑该边。注意，由于森林 F_i 是递减式的，x 和 y 将再也不会属于同一个连通分量，因此在算法 1 后续执行的任何循环中都不会再考虑它们。因此，每条边至多被加入核心一次，所以插入 H_i 的总次数为 $O(m)$，执行这些插入操作以及在第一个 for 循环中调用数据结构 $\mathcal{D}_2$ 来增加计数所花费的时间为 $O(m\log\ n)$，由引理 3.11 可知。

接下来需要证明，算法 1 第 15 行中执行的非树边移动操作，总耗时为 $O(m\log\ n)$。注意，原树 T同时包含 u和 r，并会被拆分成两棵子树 T_r 和 T_u。注意，在切断边 $e_{\min}$ 之后（且在插入跨越边之前），我们有 $\mathcal{D}_2$.求和$(r) = |E(T_r, V \setminus (T_r \cup T_w))|$ 且 $\mathcal{D}_2$.求和$(w) = |E(T_w, V \setminus (T_r \cup T_w)|$。因此，通过调用求和$(r)$和求和$(w)$，我们可以在 $O(\log\ n)$ 时间内判断 $|E(T_w, V \setminus (T_r \cup T_w))| \le |E(T_r, V \setminus (T_r \cup T_w))|$ 是否成立。不失一般性，假设 $|E(T_w, V \setminus (T_r \cup T_w))| \le |E(T_r, V \setminus (T_r \cup T_w))|$；否则，我们只需重新命名顶点即可。那么，$G/F_i$ 中的边插入/删除恰好对应于一次顶点拆分。因此，除了进行 $O(m)$ 次边插入和 $O(j)$ 次边删除外，只需进行 $O(j)$ 次顶点拆分即可更新 G/F_i。根据定理 3.13，模拟所有这些顶点拆分时需要移动的边总数有上界 $O(m\log\ n)$。

□

5 支持顶点拆分的全动态割稀疏器本节将展示如何调整 [6] 中的动态割稀疏化器，使其也能处理顶点拆分。我们的目标是证明，在

算法 1： Maintain-j-Tree()

过程 初始化(G, F_i)
  构造 $H_i = H(G, F_i)$.
  $C_i \leftarrow$ H_i的核心。
  令 $z(v) = |\{e \in$ 穿过$F_i^G : v \in e\}|$
  从引理 3.10 中初始化数据结构 $\mathcal{D}_1$，作用于 F_i。
  从引理 3.11 中初始化数据结构 $\mathcal{D}_2$，作用于 F_i，并使用向量 z。

过程AddTerminal(w)
  $r \leftarrow \mathcal{D}_1$.FindRoot($w$)
  $e_{\min} \leftarrow \mathcal{D}_1$.PathMin($w$)
  将顶点 w 插入 C_i
  $\mathcal{D}_1$.割($e_{\min}$); $\mathcal{D}_2$.割($e_{\min}$)
  $\mathcal{D}_1$.MakeRoot(w); $\mathcal{D}_2$.MakeRoot(w)
  令 T_w、T_r 分别为 F_i 的两个连通分量，其根分别为 w、r。
  不失一般性，假设 $\mathcal{D}_2$.求和(w) $\leq \mathcal{D}_2$.求和(r)
  对每个 $e = (x, y) \in E_G \setminus F_i$ 满足 $x \in T_w$ 且 $y \in V \setminus (T_w \cup T_r)$，**执行**
    从 $C(H)$ 中移除对应的多重边 (r, y)
    将边 (w, y) 插入到 $C(H)$ 中，容量为 $u_G(e)$
  对每个 $e = (x, y) \in E_G$ 其中 $x \in T_w$，$y \in T_r$ **执行**
    将多重边 (w, r) 插入到 $C(H)$ 中，容量为 $u_G(e)$
    $\mathcal{D}_2$.加法(x, 1); $\mathcal{D}_2$.加法(y, 1)

过程 插入边$(e = (u, v))$
  $G \leftarrow G \cup \{e\}$
  令 R' 为更新后的无分支根节点集合，其中包含 u和 v
  对于每个 $w \in R' \setminus R$ **执行**
    AddTerminal(w)
  将边 (u, v) 插入到 C_i 中
  $\mathcal{D}_2$.加法(u, 1); $\mathcal{D}_2$.加法(v,1)

过程 删除边$(e = (u, v))$
  $G \leftarrow G \setminus \{e\}$
  令 R' 为更新后的无分支根节点集合，其中包含 u 和 v
  对 $w \in R' \setminus R$ **执行**
    AddTerminal(w)
  从 C_i 中移除边 (u, v)
  $\mathcal{D}_2$.加法(u, -1); $\mathcal{D}_2$.加法(v, -1)

底层图 G 上进行一次顶点拆分，O只会导致我们维护的割稀疏化器 H 产生 (polylog(n)) 级别的重调整次数。这对于处理重调整次数沿着我们将构建的 j-树层次结构传播至关重要。此外，我们还注意到，该稀疏化器具有一项性质：它能够以极低的重调整次数处理大量边插入。也就是说，如果边插入的数量远大于边删除和顶点拆分的数量，那么重调整次数将是总更新次数的次线性函数；总更新包括边插入、边删除和顶点拆分。这对我们的应用至关重要。我们证明以下定理。

定理 5.1. 考虑参数 $\varepsilon > 0$ 以及一个全动态图 $G = (V, E, u_G)$，其具有 $|V(G^{(0)})| = n, |E(G^{(0)})| = m \leq O(n^2)$ 和容量比 U。那么，对于最多包含 m 次边插入、n 次顶点分裂和 D 次边删除的序列，存在一种随机化算法，可显式维护 G的$(1 \pm \varepsilon)$割稀疏器 H，使得在任何时刻

1. $|E(H)| \leq O($多重对数(n) $\cdot \varepsilon^{-2} \cdot \log U \cdot n)$,

2. H 的容量比为 $O(nU)$。

该算法的总运行时间为 $O(\text{多对数}(n)\cdot\varepsilon^{-4}\cdot\log U\cdot m)$，而在所有更新过程中，稀疏化器 H 经历的总重配置量为 $O(\text{多对数}(n)\cdot\varepsilon^{-2}\cdot\log U)\cdot(n+D)$（即边删除和边插入的次数）。对于算法启动前指定的任意常数 $c>0$，该算法以概率 $1-1/n^c$ 正确。

我们强调，如果 m 远大于 n 和 D，那么这实际上会使重调整次数相对于更新总数呈次线性。在 [6], 中证明了一个类似的定理。不过，他们的数据结构无法处理顶点分裂，而且没有明确给出一个与边插入次数无关的总重配置量上界。为便于比较，我们在此重述该定理。

定理 5.2（定理 1.2 见 [6]）。考虑 $\varepsilon>0$ 和一个动态图 $G=(V,E)$，该图有 n 个顶点，并经历边插入和删除。则存在一种随机化算法，能够以高概率显式维护一个 $(1\pm\varepsilon)$ 割稀疏器 H，其为 G，大小为 $\widetilde{O}(n\cdot\varepsilon^{-2})$，每次更新的摊销时间和重调整次数均为 $O(\text{多对数}(n)\cdot\varepsilon^{-2})$。

将定理 5.2 中的割稀疏化器调整为支持顶点分裂的一种朴素方法，是将顶点分裂模拟为一系列边插入和删除。若顶点 v 被拆分为顶点 v' 和 v''，使得$N(v')=N(v)\setminus U$且 $N(v'')=U$，其中 $|U|\le|N_G(v)|/2$。那么，我们可以先插入一个新顶点 v''，然后从 v 中移除 U 里的所有邻居，并将它们添加为 v''的邻居，从而模拟该操作。根据定理 3.13，按此方式模拟顶点分裂后，我们总共只需移动 $O(m\log n)$ 条边，因此将这些~次更新输入定理 5.2 的数据结构，可得总更新时间为 $O(m)$，符合预期。

然而，事先看来，按这种方式模拟一次顶点拆分可能会导致 $\Omega(|N(v)|)$ 次重调整；而经过一系列 n 次顶点拆分后，总重配置量可能达到 $\Omega(m\log n)$，这对于我们的目的而言大得无法接受。因此，我们的主要任务是证明，通过在内部修改算法，可以保证摊销重配置代价为 $\sim O(\text{polylog}(n))$，从而在进行 n 次顶点拆分后，总重配置量关键性地仅为 $O(n)$。实现这一目标的主要工具，是维护经历顶点拆分的图的低重置次数生成森林。

为了证明定理 5.1，我们首先回顾图的 l-SF 束 B 的核心定义，该定义见 [6]。

定义 5.3（l-SF 束）。设 F_1 是 G 的生成森林，F_2 是 $G\setminus F_1$ 的生成森林，并且归纳地，F_k 是 $G\setminus\bigcup_{i=1}^{k-1}F_i$ 的生成森林。则称 $B=\bigcup_{i=1}^{l}F_i$ 为 G 的 l-SF 束。

定义 5.4（边连通度）。给定一个（多重）图 $G=(V,E)$ 以及一条边 $e=(u,v)\in E$，其边连通度 $\lambda_G(e)$ 定义为 e 中 u与v 之间的最大流值。

注意，如果 G 是一个图，而 B 是 G的一个 l-SF 束，那么 $G\setminus B$ 中的任意边的边连通度都至少为 l，因为我们可以沿着每条边不相交树路径各发送一单位流。因此，束 B 可作为边连通度的证明，这对[15] 和 [27]的基于子采样的割稀疏化过程至关重要。我们马上会看到，只需对无容量图进行稀疏化，因此，我们的图的每个生成森林也都是最小生成森林。

将问题标准化约简为无容量图 在这里，我们将简要说明如何运用简单的技巧，将定理 5.1 的证明约简到无容量图的情形。我们首先证明以下结构性质。

引理 5.5. 设 $G=(V,E_G,u_G)$ 是一个图，$\{G_i\}_{i\in\mathcal{I}}$是一个图的集合，且 $c_i\ge 0$ 满足$G=\bigcup_{i\in\mathcal{I}}c_i\cdot G_i$。假设对于所有 $i\in\mathcal{I}$，都有 H_i是 G_i的$(1\pm\varepsilon)$-割稀疏器。那么 $H:=\bigcup_{i\in\mathcal{I}}c_i\cdot H_i$是 G的$(1\pm\varepsilon)$-割稀疏器。

证明。 设 $(S,V\setminus S)$ 是一个割。那么对于每个 i，我们有 $U_{H_i}(S)\in(1\pm\varepsilon)\cdot U_{G_i}(S)$，因此

$$U_H(S)=\sum_{i\in\mathcal{I}}c_iU_{H_i}(S)\le\sum_{i\in\mathcal{I}}c_i(1+\varepsilon)U_{G_i}(S)=(1+\varepsilon)U_G(S),$$

下界可类似地得出。 □

现在设 G 是一个容量取自集合 $\{0,1,\dots,U-1\}$ 的整数容量图。那么通过二进制展开，可以将其表示为 $G=\bigcup_{i=0}^{\log_2 U}2^iG_i$ 个无容量图 G_i，其中若边 $e\in E_G$ 的二进制展开中第 $(i+1)$ 位为 1，则 G_i 包含该边。另请注意，经过顶点拆分移动的边不会改变其容量，因此在 G 中进行一次顶点拆分，会恰好在每个图 G_i 中各进行一次顶点拆分。因此，假设定理 5.1 的某个版本

适用于无容量图，我们可以分别在图 G_i 上运行该版本，得到稀疏化器 H_i，然后调用前述引理，将 $H = \bigcup_{i=0}^{\log_2 U} 2^i H_i$ 作为最终的稀疏化器。在接下来的讨论中，我们将始终假设图 G 是无容量图。

5.1 全动态 l-SF 束 证明定理 5.1 的第一步是说明，我们也能在顶点拆分下维护 l-SF 束。我们证明如下结论。

定理 5.6. 设 $G = (V, E)$ 是一个全动态的初始空图，满足 $|V(G^{(0)})| = n$。那么，对于最多包含 m 次边插入/删除和 n 次顶点拆分的操作序列，存在一种确定性算法，能够维护一个 l-SF 束 B，其属于 G，并具有以下重调整次数性质：

1. 如果 Gc 通过边插入发生变化时，那么 B 的变化至多是插入同一条边。 $ge.$

2. 如果 G 因边删除而发生变化，则 B 至多通过删除同一条边以及可能再插入一条边而发生变化。

3. 如果 G 因顶点拆分而发生变化，则 B 通过执行顶点拆分以及至多再插入 l 条边而发生变化。

算法总耗时为 $\widetilde{O}(l \cdot m + l^2 \cdot n)$。

为此，我们需要使用 [37] 的数据结构，该结构能够在仅接收边更新的情况下，动态维护图 G 的生成森林 F。在 [6] 中，算法所提供的重调整性质对于控制图的 l-SF 束中的重调整次数至关重要，这些图仅接收边更新。这表明，在边删除和边插入时，数据结构维护的生成森林变化非常缓慢。

定理 5.7 ([37]). 对于一个经历边插入/删除、具有 n 个顶点的动态图 G，存在一种确定性算法，可以维护其最小生成森林 F。此外，以下重配置性质成立：

1. 如果 G 因一次边插入而发生变化，那么 F 的变化至多是插入同一条边 $e.$

2. 如果 G 因一次边删除而发生变化，那么 F 的变化至多是删除同一条边，并可能额外插入一条边。

该算法的摊销更新时间为 $O(\log^4 n)$。

在下一个引理中，我们将证明，即使允许进行顶点分裂，也总是存在一个新的生成森林，使其与之前的生成森林仅有细微差异。

引理 5.8. 设 F 是 G 的一个最小生成森林。假设将 G 中的顶点 $v \in V$ 拆分为顶点 v'、v''，以接收 G'。那么，存在 G' 的一个最小生成森林 F'，满足 $E(F') \supseteq E(F)$。特别地，可以通过在 F 中对 v 进行顶点拆分，并至多额外插入一条边，从 F 得到 F'。

证明。对于每条边 $e = (x, y) \in F$，在 G 中存在一个 x-y 割 $(S, V \setminus S)$，使得 e 是该割中容量最小的边。不失一般性，假设 $v \in S$。令 $S' := (S \setminus \{v\}) \cup \{v', v''\}$。若 $x \neq v$，显然，$e = (x, y)$ 仍是 G' 中割 $(S', V' \setminus S')$ 上容量最小的边，因此它仍是 F' 中的一条边。若 $x = v$，则我们知道，$e' = (v', y)$ 或 $e'' = (v'', y)$ 是 G' 中一条容量为 $u(e)$ 的边，因此它是割 $(S', V' \setminus S')$ 上容量最小的边。这证明了 $E(F') \supseteq E(F)$。

现在令 T 为包含顶点 v 的 F 树。执行顶点拆分后，树 T 将被划分为互不相交的树 T'、T''，它们分别包含顶点 v'、v''，且满足 $E(T) = E(T') \cup E(T'')$。如果 v' 和 v'' 在 G' 中属于同一个连通分量，那么我们可以通过添加连接 T' 和 T'' 的最小容量边，从 F 得到 F'。如果 v' 和 v'' 不属于同一个连通分量，则无需插入新边。 □

通过在运行定理 5.7 的算法时仅在内部为 G 指定唯一（且人为设定的）容量，我们可以使生成森林唯一，因此前一个引理表明，该算法能够以低重配置次数维护生成森林。现在，将 G 的一次顶点拆分模拟为依次执行一次顶点插入、边删除和边插入，以下推论便可由引理 5.8 和定理 3.13 直接得出。

推论 5.9. 给定一个初始为空的全动态图 $G=(V,E)$，其中 $|V(G^{(0)})|=n$ 经历至多 m 次边插入/删除和 n 次顶点拆分，则存在一种确定性算法，能够为 G 维护生成森林 F。以下重配置性质成立：

1. 如果 G_c 通过一次边插入发生变化，那么 F 的变化至多是插入同一条边 e.

2. 如果 G 因边删除而发生变化，那么 F 的变化至多是删除同一条边，并可能额外插入一条边。

3. 如果 G 因顶点拆分而发生变化，那么 F 通过执行该顶点拆分以及至多额外插入一条边而发生变化。

~算法总共耗时 $O(m)$。

现在，我们可以展示如何以低重配置开销高效维护全动态图 G的一个 l-SF 束 B。我们只需对每个 $k=1,2,\ldots,l$ 维护森林 F_k，方法是在全动态图 $H_k := G\setminus\bigcup_{i=1}^{k-1}F_i$上调用推论 5.9 中的算法。关键在于，尽管顶点拆分会在我们的束中导致最多 l 次额外更新，但这些更新仅表现为边插入，而边插入本身不会导致额外的重配置开销。

定理 5.6 的证明。根据推论 5.9 的正确性，B 在任何时刻都构成 G的一个有效 l-SF 束，这一点是显然的。下面只需证明关于重调整次数和运行时间的断言。我们先证明前者。

先考虑定理 5.6 的性质 1。我们希望证明，当 G 收到一次边插入时，B 至多因同一次边插入而发生变化。我们对 l进行归纳法证明。如果 $l=1$，这正是推论 5.9 所保证的。现在假设关于 $l-1$的断言已经得到证明。那么，当 G 收到一次边插入时，$\cup_{i=1}^{l-1}F_i$ 要么因插入边 e 而发生变化，要么完全不变。在前一种情况下，我们知道$H_l = G\setminus\cup_{i=1}^{l-1}F_i$ 完全不变，因此 F_l 也完全不变。在后一种情况下，H_l 收到边 e的插入，此时 F_l 可能因插入同一条边而发生变化。总之，$\cup_{i=1}^{l}F_i$ 至多因插入边 e而发生变化。

现在考虑定理 5.6 的性质 2。我们现在要证明，当 G因删除一条边而发生变化时，B 的变化至多包括同一次删除以及额外插入一条边。我们再次通过对 l进行归纳法来证明这一结论。当 $l=1$时，由推论 5.9 可知结论成立。现在假设已对 $l-1$证明了该结论，并且 G 因删除边 e而发生变化。如果边 e 不在 B中，则无需证明。如果边e 在 F_l中，则由推论 5.9 也可得出结论。如果边 e 在 $\cup_{i=1}^{l-1}F_i$中，根据归纳假设，我们知道 $\cup_{i=1}^{l-1}F_i$ 因删除边 e而发生变化，并且可能还插入了边 e'。另一方面，这意味着 H_l 至多因删除边 e'而发生变化。如果 e' 不在 F_l中，则无需证明。但如果 e' 在 F_l中，那么根据推论 5.9，我们知道 F_l 将因删除 e' 而发生变化，并且可能还会额外插入边 e''。然而总的来说，这意味着 $\cup_{i=1}^{l}F_i$ 恰好因删除边 e 和插入边 e''而发生变化，正如所要证明的。

现在我们处理定理 5.6 的性质 3。我们再次通过对 l 进行归纳法证明该断言。如果 $l=1$，则由推论 5.9 可得。假设该断言已对 $l-1$ 得到证明。那么我们知道，执行顶点拆分和至多 $l-1$ 次额外边插入后，$\bigcup_{i=1}^{l-1}F_i$ 发生了变化。插入到$\bigcup_{i=1}^{l-1}F_i$ 中的每条边都可能构成 F_l 中的一次边删除。我们先将顶点拆分输入定理 5.7，再处理边删除，从而完成此次更新。顶点拆分使 F_l 至多发生一次边插入。对 H_l 的每次边删除都会使 F_l 至多发生给定的删除操作和一次额外插入。根据与前面相同的论证，如果从 F_l 中删除了一条边 e，并因此将一条新边 e'' 插入到 F_l 中，那么并集 $\bigcup_{i=1}^{l}F_i$ 只会因插入 e'' 而发生变化。总之，这意味着 $\bigcup_{i=1}^{l}F_i$ 至多因 l 次额外边插入而发生变化。

我们先考虑一个固定的 i。根据定理 3.13，模拟 ~所有顶点拆分所需的边移动总时间为 $O(m)$（在我们用于存储图的邻接表数据结构中，移动一条边需要 $O(1)$ 时间）。每次顶点拆分都会使维护 F_i 的数据结构产生至多 l 次额外更新。经过一系列 n 次顶点拆分后，这会导致总共 $n\cdot l$ 次额外边更新。因此，维护所有边更新的总时间可以界定为每个数据结构 $O(m+nl)$，从而总运行时间为 $O(ml+nl^2)$。

□

5.2 初始化割稀疏器本节将介绍割稀疏器的初始化过程，该过程基于 [6] 中给出的算法。接下来，在下一节中，我们将展示如何借助定理 5.6 中的低重配置动态 l-SF 束，使该算法支持动态更新。

在开始描述算法之前，我们需要陈述以下引理，该引理源自 [6], , 而 [6], 又改编自 [27]。它表明，根据图 G 中各边的边连通度对子图中的边进行子采样，就能以高概率得到一个割稀疏化器。

引理5.10（参见 5.6 中的引理 [6]）。设 G 是一个无容量图，令 H 为从 G 中通过以概率 $p_e \geq \min\{1, \rho/\lambda_G$（$e$）$\}$ 对每条边 e 进行抽样而得到的子图，其中 $\rho = C_\xi c \log^2 n/\varepsilon^2$，且 C_ξ 是一个已知的显式常数。若边 e 被选中，则在 H 中为其赋予容量 $1/p_e$。那么， H 是 G 的一个 $(1\pm\varepsilon)$ 割稀疏化器，且该结论成立的概率至少为 $1-1/n^c$。

G 的 l-SF 束 B 的作用如今显而易见：对于每条边 $e \in G \setminus B$，都有 λ_G（e）$\geq l$，也就是说，对于不包含在 B 中的边，该束 B 为其边连通度提供了一个下界。其推论归纳于以下引理中。

引理 5.11. 设 G 是一个图，且 $l > 0$。假设 B 是 G的一个 l-*SF* 束。令 G' 为对每条边 $e \in G \setminus B$ 以概率 $p \geq C_\xi c$对数$^2\, n/(l\varepsilon^2)$ 进行子采样后得到的图，并将被采样到的边的容量设为 $1/p$。那么， $G' \cup B$ 是 G 的一个$(1\pm\varepsilon)$-割稀疏器，且其成立的概率至少为 $1-1/n^c$。

证明。 这由引理 5.10 得出。每条边 $e \in B$ 以概率 $p_e = 1$被包含在 $G' \cup B$ 中，而每条边 $e \in G \setminus B$的 λ_G(*e*) 为 $\geq l$，因此以概率 $p_e \geq C_\xi c \log^2 n/(\varepsilon^2 \lambda_G(e))$ 被包含在 $G' \cup B$ 中。 □

Th这又推出以下推论，该推论在算法中被大量运用。 .

推论 5.12. 设 G 是一个图，且 $l \geq 2C_\xi c$对数$^2\, n/\varepsilon^2$。假设 B 是 G的一个 l-*SF* 束。设 G'是通过以概率 $1/2$ 对每条边 $e \in G \setminus B$ 进行子采样，并在该边被采样时为其赋予容量 2 而得到的图。那么，以至少 $1-1/n^c$的概率， $G' \cup B$ 是 G 的一个$(1\pm\varepsilon)$-割稀疏器。

注5.13. 设 G 是一个带容量图，所有 $e \in E_G$ 的均匀容量均为 u_G（e）$= U$；设 G' 如推论 5.12 中所定义。那么，根据引理 5.5 和上述结果，给每条边 $e \in B$ 赋予容量 u_H（e）$= U$，并给每条边 $e \in G'$赋予容量 u_H（e）$= 2U$，则以至少 $1-1/n^c$的概率，图 $H = G' \cup B$是G 的一个（$1\pm\varepsilon$）-割稀疏器。

算法利用上述推论，我们现在可以描述该算法。令 $G_0 = G$，并将 B_1 计算为 $l_s := \lceil 2 \cdot C_\xi \cdot 40 \cdot \log^2(2n) \cdot \varepsilon^{-2} \rceil$-SF 束 G_0。然后，通过以概率 $1/2$ 对每条边 $e \in G_0 \setminus B_1$ 进行子采样，并在抽中时为其分配容量 2，得到 G_1。接着，我们迭代地将 B_i 计算为 G_{i-1} 的 l_s-SF 束（不考虑容量）。也就是说，我们对 G_{i-1} 运行定理 5.6 中的数据结构，但不考虑其容量（它们均为 2^{i-1}）。随后，为每条边 $e \in B_i$ 分配容量 $u_{B_i}(e) = u_{G_{i-1}}(e) = 2^{i-1}$。

然后，我们通过以概率 $1/2$ 对 $G_{i-1} \setminus B_i$ 中的每条边独立进行子采样，并在选中时为其分配容量 $2u_{G_{i-1}}(e) = 2^i$，来构造 G_i。我们重复此过程 $\rho = \lceil \log(2n) \rceil$ 次，并将 $H = G_\rho \cup \bigcup_{i=1}^{\rho} B_i$ 作为最终的稀疏化器返回。

定理 5.14（另见第 5.3 节 [6]）。设 G 是一个最多有 $2n$ 个顶点、 $m \leq O(n^2)$ 条边的（多重）图，且 $\varepsilon > 0$ 是一个参数。那么，*InitCutSparsifier*(G, ε) 的输出 H 是 G 的一个$(1\pm\varepsilon)$ 割稀疏器，并以概率 $1-1/n^c$ 满足 $|E(H)| = \widetilde{O}(n \cdot \varepsilon^{-2})$，其中任意常数 $c > 0$ 均在算法开始前指定。

证明。我们先证明 $|E(H)| = \widetilde{O}(n \cdot \varepsilon^{-2})$。首先注意到， $\bigcup_{i=1}^{\rho} B_i$ 中的边数可由 $\rho \cdot l_s \cdot n = \widetilde{O}(n\varepsilon^{-2})$ 界定，因此我们只需界定 G_ρ 中的边数。

为此，考虑指示随机变量 $\mathbf{1}\{e \in G_\rho\}$。我们有 $\mathbb{E}[\mathbf{1}\{e \in G_\rho\}] = \Pr[e \in G_\rho] \leq 2^{-\rho} \leq n^{-1}$，因此 $\mathbb{E}[|G_\rho|] = \sum_{e \in E} \mathbb{E}[\mathbf{1}\{e \in G_\rho\}] \leq m/n = O(n)$。现在我们希望应用定理 3.9 中的切尔诺夫界，从而证明以至少 $1-n^{-40}$ 的概率也有 $|G_\rho| \leq \widetilde{O}(n)$。然而请注意，一般而言，对于两条边 e， $e' \in E_G$， $\mathbf{1}\{e \in G_\rho\}$ 与 $\mathbf{1}\{e' \in G_\rho\}$ 并不独立，因为边 e' 的状态可能会影响 e 是否被纳入束 B_i。

为了解决这个问题，我们考虑如下耦合：对于每条边 $e \in G$，令 $b_{e,i} \sim$Ber$(1/2)$ 为另一个指示随机变量，表示决定在 $e \in G_{i-1} \setminus B_i$ 的情况下边 e 是否也会被采样到 G_i 的独立抛硬币过程（即算法 2 第9行使用的抛硬币过程）。由于对于一条边 $e \in G_\rho$

它必须被采样到每个 G_i 中，因此几乎必然有 $\mathbf{1}\{e \in G_\rho\} \le \prod_{i=1}^{\rho} b_{e,i} =: b_e$。特别地，几乎必然有

$|E\ G_\rho\ | \le \sum_{e\in E} b_e$　　$X = \sum_{e\in E} b_e$　　$\mathbb{E}[X] = m2^{-\rho} = (\ )$。令，并 $O\ n$　　$3\ 9,$　　$[X \ge\ \ n\ \ n\ ] \le\ \ -\ \ n\ \ n\ \ \le n^{-40}$ 注意到 $(\ )$。因此，由定理 . 可得 $\Pr\ \Omega(\ \log\ )\ \exp\ (\ \Omega(\ \log\ ))$。由此可知，$|E(G_\rho)| \le O(n\log\ n)$ 的概率至少为 $1 - n^{-40}$。

现在证明 H 是 G 的 $(1\pm\varepsilon)$-割稀疏器。首先注意，根据推论 5.12 及其后续说明，无论 G_{i-1} 中的随机性如何，$G_i \cup B_i$ 都以至少 $1 - n^{-40}$ 的概率是 G_{i-1} 的 $(1\pm\varepsilon)$-割稀疏器。现在令 $\mathcal{E}_i$ 表示事件：$G_i \cup B_i$ 是 G_{i-1} 的 $(1\pm\varepsilon)$-割稀疏器，并令 $\mathcal{E} = \bigcap_{i=1}^{\rho} \mathcal{E}_i$。接下来证明，在事件 $\mathcal{E}$ 的条件下，H 是 G 的 $(1\pm\varepsilon)$-割稀疏器。为此，令 H_j : $= G_j \cup \bigcup_{i=1}^{j} B_j$。我们对 j 使用归纳法证明，在事件 $\mathcal{E}$ 上，H_j 是 G 的 $(1\pm\varepsilon)$-割稀疏器。根据推论 5.12 可知，$G_1 \cup B_1$ 是 G 的 $(1\pm\varepsilon)$-割稀疏器。现在假设 $G_j \cup \bigcup_{i=1}^{j} B_j$ 是 G 的 $(1\pm\varepsilon)$-割稀疏器。由于在事件 $\mathcal{E}$ 的条件下，$G_{j+1} \cup B_{j+1}$ 是 G_j 的 $(1\pm\varepsilon)$-割稀疏器，因此根据引理 5.5，$G_{j+1} \cup \bigcup_{i=1}^{j+1} B_j$ 是 G 的 $(1\pm\varepsilon)$-割稀疏器，证毕。

利用简单的并集界，现在只需注意，根据推论 5.12 给出的成功概率保证，$\Pr[\neg\mathcal{E}] \le \sum_{i=1}^{\rho}\Pr[\neg\mathcal{E}_i] \le \rho n^{-40} \le n^{-30}$。最后，我们指出，失败概率 n^{-30} 是任意的；对于任意 $c>0$，只需改为设置 $l_s = \lceil 2C_\xi c\log^2(2n)\varepsilon^{-2}\rceil$，即可直接将其提升至 $1/n^c$。 □

算法 2： InitCutSparsifier($G,\ \varepsilon$)

```
G_0 ← G = (V, E)
n ← |V|
ρ ← ⌈对数(2n)⌉
l_s ← ⌈2 · C_ξ · 40 · 对数²(2n) · ε^{-2}⌉        \\ 令 C_ξ 为引理 5.10 中的常数
对于 i = 1 到 ρ，执行
    使用定理 5.6 将 B_i 初始化为一个由 G_{i-1} 个元素构成的动态 l_s-SF 束
    G_i ← (V, ∅)
    对于 e ∈ E (G_{i-1}) \ E (B_i)，执行
        以 1/2 的概率，将 e 加入 G_i，其容量为 u_{G_i}(e) ← 2^i
H ← G_ρ ∪ ⋃_{i=1}^{ρ} B_i
```

5.3 全动态割稀疏器 现在，我们将展示如何利用全动态 l-SF 束证明定理 5.1。在 [6] 中，静态算法 2 通过采用定理 5.7 给出的动态最小生成树数据结构，实现了动态化以处理边更新。我们将展示，采用定理 5.6 中能够处理顶点拆分的动态 l-SF 束数据结构，也可以允许将顶点拆分作为割稀疏器的一种更新操作。这种方法的正确性可直接由定理 5.14 得出；主要任务是通过仔细分析算法 2 的全动态图层次结构如何处理由顶点拆分引起的更新，来证明其重调整次数和运行时间界。

动态算法 首先请注意，为了使算法 2 具有动态性，我们只需确保在任何时刻，B_i 都是 l_s-SF 束 G_{i-1}（其中 $l_s = \lceil 2\cdot C_\xi\cdot 40\cdot\log^2(2n)\cdot\varepsilon^{-2}\rceil$），并且 G_i 是从 $G_{i-1}\setminus B_i$ 中按照算法 2 第9行所规定的样本，即边 $e \in G_{i-1}\setminus B_i$ 满足 $\Pr[e\in G_i] = 1/2$。如果满足这一条件，那么根据定理 5.14，以高概率，$G_\rho \cup \bigcup_{i=1}^{\rho} B_i$ 是 G 的 $(1\pm\varepsilon)$-割稀疏器。由于我们假设对手是非自适应对手，因此可以对所有更新应用并集界，从而得到最终的高概率结论。因此，剩下的问题是说明如何以低重配置开销高效地维护这样的 B_i 和 G_i。下文中，我们用 $G_i^{(t)}$，$B_i^{(t)}$ 分别表示在数据结构处理完对 G 的第 t 次更新后，图 G_i，B_i。

我们可以直接维护 B_i，如算法 3 所示：设 $U_{i-1}^{(t)}$ 是在 $G_{i-1}^{(t)}$ 上执行、以得到 $G_{i-1}^{(t+1)}$ 的批量更新。然后，将更新批次 $U_{i-1}^{(t)}$ 输入定理 5.6 的数据结构，即可从 $B_i^{(t)}$ 得到 $B_i^{(t+1)}$。接着，通过以下更新批次 $U_i^{(t)}$，从 $G_i^{(t)}$ 得到 $G_i^{(t+1)}$：

- 如果输入图 G 中的顶点 v 被拆分，使得 $N_G(v') = N_G(v)\setminus U$ 且 $N_G(v'') = U$，则在 G_i 中相应地拆分顶点 v，设置为 $N_{G_i}(v') = N_{G_i}(v)\cap N_G(v)\setminus U$ 且 $N_{G_i}(v'') = N_{G_i}(v)\cap U$。

•对于添加到 $G^{(t+1)}_{i-1}$ 中、但未纳入其 l_s-SF 束 $B^{(t+1)}_i$ 的每条边 e，即对于 $e \in (E(G^{(t+1)}_{i-1}) \setminus E(B^{(t+1)}_i)) \setminus (E(G^{(t)}_{i-1}) \setminus E(B^{(t)}_i))$，以概率 $1/2$ 将其加入 $G^{(t+1)}_i$，并在抽样选中时为其赋予容量 2^i。

•对于从 $E(G^{(t+1)}_{i-1}) \setminus E(B^{(t+1)}_i)$ 中删除的每条边 e，也就是对于 $e \in (E(G^{(t)}_{i-1}) \setminus E(B^{(t)}_i)) \setminus (E(G^{(t+1)}_{i-1}) \setminus E(B^{(t+1)}_i))$，也从 $G^{(t)}_i$ 中删除该边（如果它曾存在于图中）。

算法分析 3. 我们首先证明，$G_\rho \cup \bigcup_{i=1}^{\rho} B_i$ 的分布始终保持正确，这表明我们确实以高概率维护了一个割稀疏化器。

断言 5.15. 在任何时候，$B^{(t)}_i$ 都是 $G^{(t)}_{i-1}$ 的一个 l_s-SF 束，并且对于每条边 $e \in G^{(t)}_{i-1} \setminus B^{(t)}_i$，都有 $\Pr[e \in G^{(t)}_i] = 1/2$。特别地，$H^{(t)}$ 满足 $|E(H^{(t)})| \le \widetilde{O}(n \cdot \varepsilon^{-2})$，并且以概率 $1 - 1/n^c$ 是 $G^{(t)}$ 的一个 $(1 \pm \varepsilon)$-割稀疏化器。

证明。我们通过对时间 t 进行归纳来证明该结论。情形 $t = 0$ 可由初始化推出。假设该结论在时间 t 成立，我们证明它在时间 $t+1$ 仍然成立。

首先，由定理 5.6 的正确性可立即得出，$B^{(t+1)}_i$ 是 $G^{(t+1)}_{i-1}$ 的 l_s-SF 束。剩下要证明的是关于 $G^{(t+1)}_i$ 的部分。令 $e \in G^{(t+1)}_{i-1} \setminus B^{(t+1)}_i$。我们要验证 $\Pr[e \in G^{(t+1)}_i] = 1/2$。分两种情况。若 e 已在 $G^{(t)}_i \setminus B^{(t)}_i$ 中，则根据归纳假设，$\Pr[e \in G^{(t+1)}_i] = \Pr[e \in G^{(t)}_i] = 1/2$。否则，根据更新批次 $U^{(t)}_i$ 的构造，边 e 按照算法 3 第9行的规定进行采样，从而直接满足 $\Pr[e \in G^{(t+1)}_i] = 1/2$。由于 $G^{(t+1)}$ 的顶点数仍有 $2n$ 的上界（因为 G 总共至多经历 n 次顶点拆分），定理 5.14 的保证也表明，$H^{(t+1)}$ 满足 $|E(H^{(t+1)})| \le \widetilde{O}(n \cdot \varepsilon^{-2})$，并且以概率 $1 - 1/n^c$ 是 $G^{(t+1)}$ 的 $(1 \pm \varepsilon)$- 割稀疏器。

□

现在我们证明关于重调整次数的以下结论。

结论 5.16. 设 H 是由算法 3维护的 G 的割稀疏化器，且 $l_s = \lceil 2 \cdot C_\xi \cdot 40 \cdot \log^2(2n) \cdot \varepsilon^{-2} \rceil$。则以下重配置性质成立：

1. 若 G 因插入一条边而发生变化，则 H 至多通过插入同一条边而发生变化。
2. 若 G 因删除一条边而发生变化，则 H 至多通过删除同一条边以及可能再插入一条额外的边而发生变化。
3. 若 G 因顶点拆分而发生变化，则 H 至多通过额外插入 $\rho \cdot l_s$ 条边而发生变化 s.

证明。首先考虑性质 1。我们要证明，当 G 收到一条边插入时，H 至多通过插入同一条边（尽管其容量可能不同）发生变化。由定理 5.6 可知，B_i只能因插入给定的边而发生变化，因此这一点显然成立。再注意到，算法 3 第9行的额外子采样可能导致 B_i完全不发生变化，而这更为有利。

现在假设 G 因删除边 e而发生变化。如果 $e \notin H$，稀疏化器完全不会改变，结论便得证。因此假设 $e \in H$。若 $e \in G_\rho$，则稀疏化器 H 仅因删除边e而发生变化。因此，只需证明从某个 B_j中删除边 e 的情形。我们用归纳法证明，在这种情况下，图 $\bigcup_{i=j}^{\rho} B_i$ 至多因额外插入一条边而发生变化。首先考虑 $e \in B_\rho$的情形。由定理 5.6 可知，B_ρ 至多因额外插入一条边而发生变化，正如所需。假设该命题对所有 $\rho \ge i \ge j$均已证明，并且从 B_{j-1} 中删除了一条边。由定理 5.6 可知，B_{j-1} 至多因插入一条额外边 e''而发生变化。如果 e'' 此前不在 H 中，则不会再进行其他更改，结论便得证。否则，这对应于从 G_ρ 或某个满足 $i > j-1$的 B_i 中删除一条边。对于前一种情形，无需再证明任何内容。对于后一种情形，由归纳假设可知，$\bigcup_{i=j}^{\rho} B_j$ 至多因插入一条额外边而发生变化，因此 $\bigcup_{i=j-1}^{\rho} B_i$ 至多因插入一条额外边而发生变化。

接下来还需分析顶点分裂。我们通过对 j 使用归纳法，证明图 $\bigcup_{i=1}^{j} B_i$ 至多变化 $j \cdot l_s$ 次额外的边插入。对于 $j = 1$ 的情况，这由定理 5.6 得出。因此，假设该陈述已对 $j-1$ 证明。接着，为了更新 B_j，我们首先执行顶点分裂；根据定理 5.6，这会导致向 B_j 额外插入 l_s 条边。根据归纳假设，$\bigcup_{i=1}^{j-1} B_i$ 至多变化了 $(j-1)$ 次额外的 $\cdot l_s$

边插入。其中每次插入都可能对应于 B_j 中的一次边删除。我们在 B_j 中逐一处理这些边删除；根据定理 5.6，我们知道，对于每次此类删除， B_j 可能会因插入一条额外的边而发生变化。总的来说， $\bigcup_{i=1}^{j} B_i$ 至多变化了 $j \cdot l_s$ 次边插入，这正是我们要证明的结论。 □

现在我们可以通过证明完整定理来结束。

定理5.1的证明。如开篇所述，不失一般性，我们可以假设 G 没有容量限制。根据定理 5.14、命题5.15，并对所有 $O(m)$ 个故障事件应用并集界，再注意到 H 中的容量有 $2^{\rho} = O(n)$ 的界，即可得到正确性。

为证明关于重调整次数的陈述，首先注意，根据命题 5.16，在 G 上执行的边插入和顶点分裂分别至多导致 $\rho l_s = O(\text{polylog}(n))$ 次额外的边插入，这些插入发生在 H 中；至关重要的是，它们不会导致 H 发生边删除。因此，由于在任何时刻 $|E(H)| \leq \widetilde{O}(n \cdot \varepsilon^{-2})$，且只有当一条边从 G中删除时，它才会离开 H，所以 H 发生变化所涉及的边删除和边插入总数必须不超过 $O(\text{polylog}(n) \cdot \varepsilon^{-2} \cdot (n + D))$，其中 D 是 G 接收的边删除总数。不过请注意， H 还会因 n顶点分裂而发生变化。幸运的是，定理 3.13 表明，通过边删除和边插入来模拟顶点分裂，至多会额外导致 $O(\text{polylog}(n) \cdot \varepsilon^{-2} \cdot (n + D))$ 次更新。

为证明运行时间保证，首先注意到每个束 B_i 接收的顶点分裂不超过 n 次，因此根据定理 5.6，可以在总计 $\widetilde{O}(\rho l_s \cdot m + \rho \cdot l_s^2 \cdot n) \leq \widetilde{O}(m\rho l_s^2) = \widetilde{O}(m \cdot \varepsilon^{-4})$ 的时间内处理这些分裂。此外，根据命题 5.16，每个维护 B_i 的数据结构需要处理的边更新总量可界定为 $O(m + \rho l_s \cdot n) = O(m + \rho l_s n) = \widetilde{O}(m \cdot \varepsilon^{-2})$。由于每次边更新都可以在摊还时间 $O(\text{polylog}(n))$ 内处理，因此维护 B_i 所花费的总时间为 $\widetilde{O}(\rho \cdot m \cdot \varepsilon^{-2}) = \widetilde{O}(m \cdot \varepsilon^{-2})$。 □

算法 3： UpdateCutSparsifier(u)

1\\ 此处 u 要么编码边插入/删除，要么编码顶点拆分。2 $U_0^{(t)} \leftarrow \{u\}$3 对于 $i = 1$ 到 ρ 执行 4 将更新批次 $U_{i-1}^{(t)}$ 转发至定理 5.6 的数据结构。5 使用按以下方式构造的更新批次 $U_i^{(t)}$ 更新 $G_i^{(t)}$ ： 6 如果 $U_0^{(t)}$ 包含顶点拆分，则 7 将所有顶点拆分添加到 $U_i^{(t)}$ 8 对于 $e \in (E(G_{i-1}^{(t+1)}) \setminus E(B_i^{(t+1)})) \setminus (E(G_{i-1}^{(t)}) \setminus E(B_i^{(t)}))$，执行 9 以 $1/2$的概率，将容量为 2^i 的边 e 的插入添加到 $U_i^{(t)}$ 10 对于 $e \in (E(G_{i-1}^{(t)}) \setminus E(B_i^{(t)})) \setminus (E(G_{i-1}^{(t+1)}) \setminus E(B_i^{(t+1)}))$，执行 11 将边 e 的移除添加到 $U_i^{(t)}$

6 层次结构 $O(j)$**-树**本节中，我们将在 (L, j)-层次结构中运用第 4 节和第 5 节开发的工具来证明定理 1.1；该层次结构是本节的核心概念。我们首先重述定理 1.1，并讨论一些相关说明。

定理1.1. 设 $c \geq 1$ 为一个常数。给定一个具有 n 个顶点的带容量无向图 $G = (V, E, u)$，其容量比为 [12] $U = \text{poly}(n)$，并且经历边插入和删除操作；再给定整数 $j \geq 1$ 和 $1 \leq L \leq o\left(\sqrt{\log(n/j)/\log\log n}\right)$，则存在一个数据结构，维护由 $O(j)$ 个 G 的树组成的集合 $\mathcal{H}$，以 $\widetilde{O}(\log n)^L$-保留 G 的割，且成功概率至少为 $1 - 1/n^c$。

该数据结构显式维护 $\mathcal{H}$ *in* $(n/j)^{2/L} \log^{O(L)} n$ 的摊销更新时间，保证 $|\mathcal{H}| = \log^{O(L)} n$，并且每棵 $O(j)$-树 $H \in \mathcal{H}$ 的容量比为 $n^{2L+1} U \log^{O(L)} n$。

[12] Re称图中最大边容量与最小边容量之比为该图的容量比。

备注 6.1. 我们指出，定理 1.1 中 L 的选择存在权衡。若将 L 设为常数，则割质量为 $\mathrm{poly}(\log n)$，摊销更新时间为 $\widetilde{O}((n/j)^{2/L})$。若为 L 选择更大的常数，则更新时间可以变得任意小，但仍是多项式级别。另一方面，设定 $L = \Theta(\sqrt{\log n/\log\log n})$，可将摊销更新时间改善至 $n^{o(1)}$，代价是割质量变差，为 $n^{o(1)}$。

注记 6.2. 定理 1.1 中的上界 $L \le o\left(\sqrt{\log(n/j)/\log\log n}\right)$ 旨在确保对于较大的 L，项 $\log^{O(L)} n$ 不会主导更新时间。这是因为

$$\begin{aligned}
\log^{O(L)} n &= \exp\left(O(L\log\log n)\right) \\
&= \exp\left(\frac{1}{L}O\left(L^2\right)\log\log n\right) \\
&= \exp\left(\frac{1}{L}o\left(\frac{\log(n/j)}{\log\log n}\right)\log\log n\right) && \text{由于 } L \le o\left(\sqrt{\log(n/j)/\log\log n}\right) \\
&= \exp\left(o\left(\frac{1}{L}\right)\log(n/j)\right) \\
&= \left(\frac{n}{j}\right)^{o(1/L)}.
\end{aligned}$$

本节组织如下。我们首先在第 6.1 小节中定义 (L, j)-层次结构，并证明一些基本性质。值得注意的是，我们证明该层次结构自然会产生一个由 $\mathcal{H}$ 个 $O(j)$-树组成的集合。在第 6.2 小节中，我们介绍如何初始化和维护层次结构的单个层，这将作为完整构造中的一个关键子程序。接着在第 6.3 小节中，我们说明如何初始化和维护定理 1.1 的数据结构。最后，我们在第 6.4 小节中证明定理 1.1。

6.1 层次结构的定义与基本性质 如前所述，为了维护 $\mathcal{H}$ 个 $O(j)$-树的集合，我们使用一种称为 G 的 (L, j)-层次结构的分层构造，这是本节的核心概念。在正式定义层次结构之前，我们先回顾第 4 节中的一个定义。

定义 6.3（稀疏化核心）。给定一棵 j-树 H，其核心为 $C(H)$，容量比为 U，若一个图 $\widetilde{C}(H)$ (H) 满足以下条件，则称其为质量为 β 的稀疏化核心

1. $\widetilde{C}(H)$ 是一个至多含有 $O(j\log U)$ 条边的图，

2. $\widetilde{C}(H)$ 的容量比为 $\mathrm{poly}(n)U$，并且

3. 对于 $C(H)$ 中的所有割 $(S, V\setminus S)$，均有

$$U_{C(H)}(S) \le U_{\widetilde{C}(H)}(S) \le \beta \cdot U_{C(H)}(S).$$

现在我们继续定义 (L, j)-层次结构。

定义 6.4 ((L, j)-层次结构 G)。给定一个有 n 个顶点的图 $G = (V, E, u)$ 以及正整数 $j, L \le n$，令 $n = j_0 > j_1 > \cdots > j_{L-1} > j_L = j$ 为一个严格递减的整数序列。那么，G 的一个 (L, j)-层次结构是由集合 $\mathcal{H}_0, \ldots, \mathcal{H}_L$ 构成的序列，满足

1. 集合 $\mathcal{H}_0$ 是一个包含 G 的单元素集合，即 $\mathcal{H}_0 = \{G\}$。图 G 被视为 G 和 $C(G)$ 的平凡 n-树[13]，而它表示 G 的一个2-近似割稀疏化器（定理 4.5）。

2. 对于每个 $1 \le i \le L$ 和 $H_i \in \mathcal{H}_i$，都存在唯一的父图 $H_{i-1} \in \mathcal{H}_{i-1}$，使得 H_i 是定义在 $C(H_{i-1})$ 的顶点集上的一棵 $O(j_i)$-树，也是 H_{i-1} 的一个2-近似稀疏化核心。此外，每个图 $\widetilde{H_i}$ 都有一个与之关联的2-近似稀疏化核心 $C(H_i)$。

对于每个 $H_i \in \mathcal{H}_i$，我们用 $\mathcal{H}(H_i)$ 表示以 H_i 为父节点的图的集合 $H_{i+1} \in \mathcal{H}_{i+1}$。

[13] 平凡的 n-树由定义在整个顶点集上的核心和空森林组成。

在 (L, j)-层次结构中，集合 $\mathcal{H}_L$ 定义为由图 $\mathcal{H}_{L-1}$ 的稀疏化核心的顶点集上的 $O(j)$-树构成的集合；而定理 1.1 中的 $O(j)$-树集合 $\mathcal{H}$ 则定义在输入图 G 的完整顶点集上。现在，我们将说明如何使用 (L, j)-层次结构构建集合 $\mathcal{H}$ 。首先，我们定义 (L, j)-层次结构中链的概念。

定义 6.5（$L\,j$）-层次结构中的链）。考虑一个（$L\,j$）-层次结构 $\mathcal{H}_0, \ldots, \mathcal{H}_L$。如果对于每个 $1 \le i \le L$，图 H_{i-1} 都是层次结构中 H_i 的父节点，即 $H_i \in \mathcal{H}(H_{i-1})$，则图的序列$H_0, \ldots, H_L$ 是该层次结构中的一条链。

现在，给定层次结构中的一条链 H_0，H_1，……，H_L，令 $F_i = F(H_i)$ 和 $C_i = C(H_i)$ 分别表示对应的森林和 $O(j_i)$-树 H_i的稀疏化核心。于是，我们将图与该链关联起来：

$$H^{H_0,\ldots,H_L} := \widetilde{C}_L \cup F_L \cup \cdots \cup F_1. \tag{6.1}$$

现在我们证明，任意链的关联图都是输入图 G的顶点集 V 上的一个 $O(j)$-树。

引理 6.6。对于层次结构 $\mathcal{H}_0, \ldots, \mathcal{H}_L$中的任意链 $H_0, \ldots, H_L$，图 $H^{H_0,\ldots,H_L}$ 是顶点集 V（G）上的一个 O（j）-树。

证明。通过对 $1 \le i \le L$进行归纳法，我们证明图 $C_i \cup F_i \cup \cdots \cup F_1$ 是一个 $O(j_i)$-树。由 $j_L = j$这一事实即可得出结论。

- 基本情况：根据定义，H_1 是一个以 C_1 为核心、以 F_1为森林的 $O(j_1)$-树。因此，根据引理 4.6，$C_1 \cup F_1$ 也是一棵 $O(j_1)$-树。
- 归纳步骤：假设对于某个层 i，森林 $F_i \cup \cdots \cup F_1$ 与稀疏化核心 C_i 共同构成一棵 $O(j_i)$-树。根据定义，$H_{i+1} = C_{i+1} \cup F_{i+1}$ 是顶点集为 $V(C_i)$的 $O(j_{i+1})$-树，这意味着$C_{i+1} \cup F_{i+1}$也是同一顶点集上的一棵 $O(j_{i+1})$-树。因此，$F_i \cup \cdots \cup F_1$中每棵树的根都恰好属于 F_{i+1}中的一棵树，所以 $F_{i+1} \cup F_i \cup \ldots F_1$ 确实是一个根集为 $V(C_{i+1})$的有根森林。

□

因此，对于每个 (L, j)-层次结构，我们都可以关联一个相应的 $\mathcal{H}$ 个 $O(j)$-树集合，如下所述。

定义 6.7（与 $\mathcal{H}$ 对应的集合，属于一个 (L, j)-层次结构）。给定一个 (L, j)-层次结构 $\mathcal{H}_0, \ldots, \mathcal{H}_L$，我们将关联的 $\mathcal{H}$ 个 $O(j)$-树集合定义为

$$\mathcal{H} := \left\{ H^{H_0,\ldots,H_L} : H_0, \ldots, H_L \text{ is a chain in the hierarchy} \right\}.$$

除了 $\mathcal{H}$ 由 $O(j)$-树组成之外，我们还需要保证每个割都能以高概率被 $\mathcal{H}$ 近似保留（定义 3.7）。如定理 1.1 所述，我们构造并维护的层次结构将满足这一额外条件。

在本节余下部分，我们将简要概述如何初始化并维护这样的 (L, j)-层次结构。随后，在第 6.2 和 6.3 小节中，我们将给出正式描述；在第 6.4 小节中，我们将证明所维护的关联 $\mathcal{H}$ 个 $O(j)$-树集合满足定理 1.1 的条件。

首先注意，对于任意 $1 \le i \le L$，我们都可以写成 $\mathcal{H}_i = \bigcup_{H_{i-1} \in \mathcal{H}_{i-1}} \mathcal{H}(H_{i-1})$。这表明，要初始化并维护一个 (L, j)-层次结构 $\mathcal{H}_0, \ldots, \mathcal{H}_L$，我们可以专注于初始化并维护每个集合 $\mathcal{H}(H_{i-1})$，以及稀疏化核心 C_{i-1}，其顶点集是 $\mathcal{H}(H_{i-1})$ 中图的定义域。为此，我们将采用前几节中的两种数据结构：

- 为了初始化并维护集合 $\mathcal{H}(H_{i-1})$，我们利用定理 4.4 获得一组 $O(j_i)$-树，其顶点集为稀疏化核心 C_{i-1}。由于根据定理 4.4 得到的图集合对于我们的目的而言过于庞大，我们通过均匀随机抽样 $O(\log n)$ 个元素，大幅缩减其规模，具体细节见第 6.2 小节。
- 通过定理 5.1 的割稀疏化器来维护（稠密）核心 C_i 的稀疏化核心 C_i 。该割稀疏化器能够处理顶点分裂，以及在 C_i中大量的边插入。我们在下面的推论中重新陈述该定理的保证。

推论 *6.8.* 设 $H = H(G, F)$ 是由定理 *4.4* 维护的 $O(j)$-树，且 $C = C(H)$ 是其核心。~那么，通过运行定理 *5.1*，我们可以初始化并维护一个质量为 *2* 的稀疏化核心 $C(H)$，覆盖 ~上的完整更新序列 $O(j)$，总耗时为 $O(m \log U)$，总重调整次数（即~ ~$C(H)$中的边插入/删除）为 $O(j \log U)$。对于算法开始前指定的任意常数 $c > 0$，该算法以概率 $1 - 1/n^c$ 正确运行。

为了处理层次结构中的更新（从而维护 $\mathcal{H}$），数据结构必须能够高效处理每个 $1 \le i \le L$ 中 $\mathcal{H}_{i+1}$ 与 $\mathcal{H}_i$ 之间的交互。我们将在第 6.1 小节中对此进行详细讨论。下面简要介绍维护层次结构的要点。

处理重调整次数传播。给定层次结构中的一条链 H_0，H_1，……，H_L，对 H_0 的一次更新可能导致对 H_1 的多次更新，而每次更新又会导致对 H_2 的多次更新，以此类推。根据定理 4.4，对 H_{i-1} 的一次更新后，H_{i-1} 的核心 C_{i-1} 会经历 $O(1)$ 次顶点拆分。因此，将每次顶点拆分模拟为一系列边插入和删除，会导致每次更新产生 $\Omega(j_{i-1})$ 的重调整次数；又由于每棵 $O(j_i)$-树 $H_i \in \mathcal{H}_i$ 的核心大小为 $\Theta(j_i)$，对 G 的一次更新可能导致图 $H_L \in \mathcal{H}_L$ 中出现 $\Omega(j_1 j_2 \dots j_L)$ 次更新，这是我们无法承受的。因此，直接对 C_{i-1} 使用定理 4.4 来初始化并维护集合 $\mathcal{H}(H_{i-1})$，在我们的用途下计算上不可行。

~为了解决这个问题，我们通过在稀疏化核心 C_{i-1} 上应用定理 4.4，来初始化集合 $\mathcal{H}(H_{i-1})$；该核心使用定理 5.1 进行维护。也就是说，若 γ_{rec} 表示任意一个 H_i 在响应其父节点 H_{i-1} 的单次更新时所经历的更新次数的（摊销）上界，那么对初始图 G 的一次更新，最多可能导致每个图 $H_i \in \mathcal{H}_i$ 经历 γ^i_{rec} 次更新。根据定理 4.7，我们可以确保 $\gamma_{\text{rec}} = \text{poly}(\log n) \log U$。假设 $U = \text{poly}(n)$，那么结果为 $\gamma^i_{\text{rec}} = \log^{O(i)} n$。[14]

层级重建。维护层次结构时出现的另一个问题是，任何 $O(j_i)$-树 $H_i \in \mathcal{H}_i$ 最多只能处理 $O(j_i)$ 次更新：我们已在定理 4.4 中看到，核心 C_i 的大小会随底层图接收的更新次数成比例增长。因此，为了保证 H_i 始终是 $O(j_i)$-树，我们无法处理超过 $O(j_i)$ 次更新。

为了解决这个问题，正如我们将在第 6.3 小节讨论的那样，一旦某个图 $H_i \in \mathcal{H}_i$ 的核心自初始化以来大小翻倍，我们就重新计算集合 $\mathcal{H}_i$、$\mathcal{H}_{i+1}$、……、$\mathcal{H}_L$。

保持 G 中的割。为了证明 $\mathcal{H}$ 能近似保持 G 中的割，我们采用归纳论证：根据引理 6.9，对于任意 $1 \le i \le L$，$\mathcal{H}(H_i) \subseteq \mathcal{H}_{i+1}$ 个 $O(j_{i+1})$-树组成的集合，能够近似保持其更稠密对应对象 C_i 的~ ~割。根据定理 5.1，任意稀疏化核心 C_{i-1} 本身也能近似保持它们在更稠密对应对象 C_{i-1} 中的割。

为了实现这些保证，我们的数据结构借鉴了 [19], 的数据结构中的一些元素；该数据结构维护一个 $O(j)$-树的单层层次结构（即，当 $L = 1$ 时）。在下一节中，我们将介绍单层情形，它将作为我们层次结构中的一个子程序。

6.2 预备：单层方案 本节总结如何运用定理 4.4，在单层层次结构中维护一组 $\mathcal{H}$ 个 $O(\log n)$ 个 $O(j)$-树（即证明情形$L = 1$下的定理 1.1）。该单层方案稍后将作为我们层次结构中的一个子程序使用，具体内容详见第 6.1 小节。为清晰起见，我们首先重新陈述定理 4.4。

定理 4.4. 给定参数 j 和动态图 $G = (V, E_G, u_G)$，该图初始具有 m 条边和容量比 U，并经历最多 $O(j)$ 次边插入和删除。则存在一种算法，能够初始化并~维护一组 $i = 1$、2、……、$k = O(m/j)$ 个递减有根森林 $F_i = (V, E_{F_i}, u_{F_i})$，其根集为 R_i，以及相应的诱导 $|R_i|$-树 $H_i = H(G, F_i)$。

初始时，对于所有 i，我们有 $|R_i| \le O(j)$。此外，在任何时刻，都有：

1. 对所有 $i = 1, k$，*2.* 对某些 $\alpha = \widetilde{O}\, n$，

3. 每个 H_i 的容量比为 OmU，

[14]由于定理 4.7 仅保证整个更新序列上的摊销重调整次数为 $\gamma_{\text{rec}} = \text{polylog}(n)$，因此第 6.4 小节中的最终分析更加复杂。

4. 在对 G 执行了 t 次边更新后，对于所有 i，我们有 $|R_i^{(t)}| \le |R_i^{(0)}| + Ot$。

该算法的总运行时间为 $\widetilde{O}(m^2/j)$。此外，每个收缩图 G/F_i （因此核心 $C(H_i)$）的总变化包括 $O(j)$ 次顶点分裂、 $O(j)$ 次边删除，以及 $O(m)$ 次边插入。

现在，令 S 为 G中的任意一个割，并令 $\mathcal{H}' = \{H_i\}_{i=1}^{k}$ 为由定理 4.4 得到的 O个 (j)-树的集合。于是，根据引理 3.6，我们有

$$U_G(S) \le k^{-1} \sum_{i=1}^{k} U_{H_i}(S) \le \alpha \cdot U_G(S). \tag{6.2}$$

特别地，这意味着集合中必定存在一个图 H_i ，使得 $U_{H_i}(S) \le \alpha U_G(S)$。~因此，集合 $\mathcal{H}'$ α确定性地保留了 G 所有割，其中 $\alpha = O(\log n)$。

遗憾的是，集合 $\mathcal{H}'$ 的大小为 $|\mathcal{H}'| = \Omega(m/j)$，因此对于我们在第 6 节中的递归应用来说太大了。为了解决这个问题，我们从 $\mathcal{H}'$ 中均匀随机抽取 $O(\log n)$ 个图，以初始化集合 $\mathcal{H}$。这大幅缩小了集合的大小，但代价是只能以高概率保留这些割。

引理 6.9. 设 $H_1, \ldots, H_k$ 是定理 4.4 中的 $\mathcal{H}'$ 个 $O(j)$-树，并设 $(S, V \setminus S)$ 是 G 中的一个割。那么，若我们均匀随机抽取 $l = O(\log n)$ 个 $O(j)$-树 $H_{i_1}, \ldots, H_{i_l}$，

- $U_G(S) \le$ 最小值$_{1 \le j \le l} U_{H_{i_j}}(S)$，并且
- 以至少 $1 - 1/n^c$ 的概率，最小值$_{1 \le j \le l} U_{H_{i_j}}(S) \le 2\alpha \cdot U_G(S)$。

证明。根据 usi利用马尔可夫不等式和公式 (6.2)，对于任意随机抽样得到的 $O(j)$-树 H_{ij}，

$$\Pr\left[U_{H_{i_j}}(S) \ge 2\alpha U_G(S)\right] \le 1/2.$$

因此，均匀随机抽样 $l = O(c\log\ n)$ 个 $O(j)$-树即可。 □

因此，鉴于我们只要求蒙特卡罗算法能够应对非自适应对手，只需先抽样得到 $O(\log n)$ 个 $O(j)$-树，之后的所有计算便可在少量 $O(j)$-树上进行，而无需使用完整序列。下面的引理总结了我们的单层算法所提供的保证，随后将介绍该算法并给出定理的证明。

引理 6.10. 给定一个整数 $j \ge 1$，以及一个有 n 个顶点的图 $G = (V, E, u)$，其初始有 m 条边，则存在一个~数据结构，用于维护一个由 $O(j)$-树组成的集合 $\mathcal{H}$ ，以至少 $1 - 1/n^c$ 的概率对 G 的割提供 $O(\log n)$ 保真度；其中，任意常数 $c > 0$ 均须在算法开始前指定。

该数据结构保证在任何时刻 $|\mathcal{H}| = O(\log n)$，并且每棵 $O(j)$-树$H \in \mathcal{H}$ 的容量比为 $O(mU)$，其中 $U = \mathrm{poly}(n)$ 是 G 容量比的上界。

算法。 我们采用定理 4.4 中参数为 j 的算法，在 ~$O(j)$ 次更新期间初始化并维护一个由 $O(j)$ 棵树组成的集合 $\mathcal{H}'$ ，该集合以 $O(\log n)$ 的因子保留 G 中的割。集合 $\mathcal{H}$ 通过从 $\mathcal{H}'$ 中均匀随机地子采样$O(\log n)$ 棵 $O(j)$ 树得到。在 G 中进行 $O(j)$ 次更新后，我们从头重新启动算法，以确保集合 $\mathcal{H}'$ 始终由 $O(j)$ 棵树组成（即，其核心包含$O(j)$ 个顶点）。

引理的证明 *6.10* 引理。 我们证明更新时间界以及高概率成立的结论 .

更新时间：这直接由定理 4.4 得出，因为在 $O(j)$ 次更新中维护图 H_i 的总耗时为$\widetilde{O}(m^2/j)$，因此摊销时间为 $\widetilde{O}(m^2/j^2)$。

高概率断言：对于任意固定的割 $(S, V \setminus S)$，在任意时刻，根据引理 6.9，我们有 e

$$\min_{H \in \mathcal{H}} U_H(S) \le 2\alpha \cdot U_G(S)$$

~的概率至少为 $1 - 1/n^c$，且为 $\alpha = O(\log n)$。这意味着集合 $\mathcal{H}$ 是一个由 $O(j)$-树

~ ~组成的集合，它以至少 $1 - 1/n^c$ 的概率对 G 的割实现 $O(\log\ n)$-保持。此外，根据定理 4.7，图 C_i以至少 $1 - 1/n^c$ 的概率仍是 C_i 的 2-近似稀疏化核心。□

6.3 定理 1.1 的数据结构 定理 1.1 的数据结构背后的主要动机，是以更大的近似为代价，改善引理 6.10 的更新时间。我们通过使用定义 6.4 中定义的 $(L,\ j)$-层次结构 $\mathcal{H}_0,\ \ldots,\ \mathcal{H}_L$，维护一个由 $O(j)$-树组成的集合 $\mathcal{H}$ 来实现这一点。注意，如果我们在定理 1.1 中设置 $L=1$，就能获得与引理 6.10 相同的保证。

该数据结构总结于算法 4 中。我们将在第 6.3.1 和 6.3.2 小节分别介绍初始化和维护。

6.3.1 初始化 如定义 6.4 所述，数据结构的每一层 i 都会初始化集合 $\mathcal{H}_i$，其中每个图 $H_i \in \mathcal{H}_i$ 都有一个大小为 $O(j_i)$ 的核心 C_i。序列 $j_0 > \cdots > j_L$ 的递减性质确保每一层的核心大小都小于前一层中的核心。我们首先解释每个 j_i的选择。

对于每一层 i，我们选择整数 j_i，使得 $\mathcal{H}_i$ 中的核心包含来自 $\mathcal{H}_{i-1}$中核心的顶点的 $1/k = (j/n)^{1/L}$。也就是说，从第 0 层的 $j_0 = n$ 个顶点开始，$\mathcal{H}_1$ 中的核心包含 $O(n/k)$ 个顶点，$\mathcal{H}_2$ 中的核心包含 $O\left(n/k^2\right)$个顶点，依此类推，直到 $\mathcal{H}_L$ 中的核心包含 $n/k^L = n(j/n)^{L/L} = j$个顶点。因此，我们设

$$j_i = n/k^i = n\,(j/n)^{i/L} \tag{6.3}$$

对每个 $0 \le i \le L$。

数据结构的初始化过程如下。首先从 $\mathcal{H}_0 = \{H_0\}$ 开始，其中 H_0 是 G 的平凡 n-树，其核心为 $C_0 = G$，空森林为 $F_0 = (V,\ \emptyset,\ 0)$。随后，它在 C_0 上初始化定理 5.1~ 的数据结构，以维护稀疏化图 C_0。

为了初始化 $\mathcal{H}_1$，该数据结构在其核心 C_0 上应用引理 6.10 的 $O(j_1)$-树。由于 $\mathcal{H}_0$~ 仅由 H_0 组成，因此 $\mathcal{H}_1$ 的 $O(j_1)$-树定义为$\mathcal{H}_1 = \mathcal{H}(H_0)$。

为了获得下一层的图，该数据结构会重复此过程，直到达到层 L。例如，层 2 在 $\mathcal{H}_1$ 中 $O(j_1)$-树的核心上初始化：每个 $H_1 \in \mathcal{H}_1$ 都由一个核心 C_1 构成，数据结构在其上~初始化定理 5.1，以维护稀疏化核心 C_1。随后，它在 ~ ~C_1 上应用引理 6.10，以维护集合 $\mathcal{H}(H_1)$，其中包含 C_1 的顶点集上的 $O(j_1)$-树。接着，$\mathcal{H}_2$ 的 $O(j_2)$-树定义为$\mathcal{H}_2 = \bigcup_{H_1 \in \mathcal{H}_1} \mathcal{H}(H_1)$。

图 6.1 总结了层级 i 和 $i+1$的初始化，其中 $1 \le i \le L-1$。

6.3.2 处理更新 该数据结构可以处理一批更新 U，其中每次更新$u \in U$ 都由向 G中插入或删除一条边组成。为简单起见，我们假设 U 是一个单元素集合，因为这可以自然地扩展到一般情况。

为了动态维护层次结构 $\mathcal{H}_0$、.....、$\mathcal{H}_L$，我们采用批处理技术（如下所述），以保证 $\mathcal{H}_i$ 中的每个核心在任意时刻的大小都保持为 $O(j_i)$。简而言之，更新过程遵循如下自顶向下方法。对于层次结构中的任意链 H_0、.....、H_L 以及每个 $0 \le i \le L-1$，数据结构都会获取一个集合 $U(H_i)$，并将其传递给 H_i。集合 $U(H_i)$由因 G中的更新 U 而必须传递给 H_i 的更新组成。当 H_i 正在处理 $U(H_i)$中的更新时，数据结构会获取一个集合 $U(H_{i+1})$。随后将集合 $U(H_{i+1})$传递给 H_{i+1}，依此类推。

使用这一术语，对于层次结构中的任意链 $H_0, \ldots, H_L$，更新从 H_0（即 G）开始，方法是设置 $U(H_0) = U$。更新 $U(H_0)$ 被传递给定理 5.1 的数据结构，该结构维护稀疏化 ~ ~ ~核心 C_0。由于每个图 $H_1 \in \mathcal{H}_1$ 都定义在 C_0上，算法将 $U(H_1)$ 设为此次更新导致的 C_0 变化。对于每个 $H_1 \in \mathcal{H}_1$，数据结构都会将 $U(H_1)$ 传递给引理 6.10 的数据结构，该结构维护H_1[15]。这会导致核心 ~ C_1 ~ H_1 5 1 中出现一些变化，并将这些变化传递给定理 的数据结构。该结构维护稀疏化核心 C_1。随后，对于每个 $H_2 \in \mathcal{H}(H_1)$，将 C_1 中的变化设为 $U(H_2)$。这一过程会持续进行，直至更新 H_L，或某一层级 i的核心大小自初始化以来翻了一倍（即对于某个固定且足够大的常数 c，更新后有 $|V(C_i)| > cj_i$）。在后一种情况下，数据结构会停止向更高层级传递更新，转而从层 i到层 L重新初始化层次结构中的集合 $\mathcal{H}_i, \ldots, \mathcal{H}_L$。

[15]随着核心 $\widetilde{C}_i$ 不断增长，引理 6.10 中的数据结构也需要能够处理孤立顶点插入。不过，我们可以通过在初始化时纳入一个包含 O（j_{i+1}）个虚拟顶点的集合，并按需为其重新命名，轻松地处理这些插入。

算法 4： Dynamic-J-Tree(G, L, j)

输入： 一个具有 n 个顶点的图 $G=(V, E, u)$，层级数 L，以及正整数 j

维护： 一个由 $O(j)$-树构成的集合 $\mathcal{H}$，这些树属于 G，且具有 $|\mathcal{H}| = O\left(\log^L n\right)$

过程初始化
  $C_0 \leftarrow G$ 和 $F_0 \leftarrow (V, \emptyset, \mathbf{0})$
  $\mathcal{H}_0 \leftarrow \{C_0 \cup F_0\}$
  对每个 $0 \le i \le L$ **执行**
    $j_i \leftarrow n(j/n)^{i/L}$ /* 注意 $j_0 = n$ 和 $j_L = j$ */
  初始化 $\widetilde{C}_0$ 在 $C_0 = G$ 上使用定理 5.1 的割稀疏器
  Compute-Hierarchy(L) /* 初始化 $H_L, \ldots, H_1$ */
过程 更新(U)
  /* $u \in U$ 是边插入或删除 */
  将 U 传递给用于维护的数据结构 $\widetilde{C}_0$
  对于每个图 $H_1 \in \mathcal{H}_1$ **执行**
    $U(H_1) \leftarrow$ 中的更改 $\widetilde{C}_0$
  对于 $i = 1$ 到 L **执行**
    对每个具有核心 C_i 的图 $H_i \in \mathcal{H}_i$ **执行**
      将 $U(H_i)$ 传递给维护 H_i 的数据结构
        /* 使用引理 6.10 更新 $O(j_i)$-树 $H_i = C_i \cup F_i$ */
      将 C_i 中的更改传递给维护以下内容的数据结构 $\widetilde{C}_i$
        /* 更新底层割稀疏器 $\widetilde{C}_i$，使用定理 5.1 */
      对每个 图 $H_{i+1} \in \mathcal{H}_{i+1}(\widetilde{C}_i)$ **执行**
        $U(H_{i+1}) \leftarrow$ 中的变化 $\widetilde{C}_i$
    如果 存在 $H_i \in \mathcal{H}_i$，其核心大小为 $|V(C_i)| > cj_i$ **则**
      /* c 是一个足够大的常数 */
      Compute-Hierarchy(i)
      换行
过程Compute-Hierarchy(i) /* 计算 $\mathcal{H}_i$、…、$\mathcal{H}_L$ */
  对于 $k = i$ 到 L，**执行**
    $\mathcal{H}_k \leftarrow \emptyset$
    对于 $H_{k-1} \in \mathcal{H}_{k-1}$ 及其稀疏化核心 $\widetilde{C}_{k-1}$ **执行**
      计算 $\mathcal{H}_k(\widetilde{C}_{k-1})$ 作为 $O(j_i)$-树 $\widetilde{C}_{k-1}$ 使用引理 6.10
      将 $\mathcal{H}_k(\widetilde{C}_{k-1})$ 加入 $\mathcal{H}_k$
    对每个 $H_k \in \mathcal{H}_k$ 含有核心 C_k **执行**
      （重新）初始化 $\widetilde{C}_k$ 使用定理 5.1 的割稀疏器
      $U(H_k) \leftarrow \emptyset$

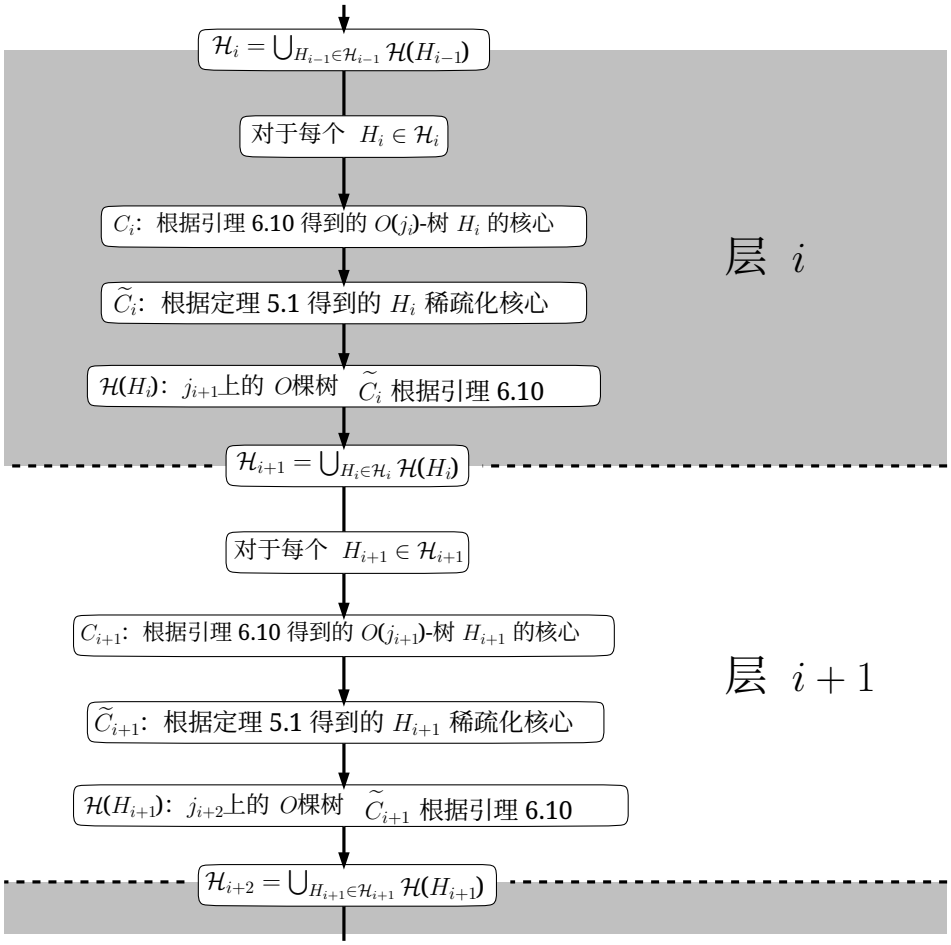


图 6.1：初始化概要。

图 6.2 总结了层级 i 和 $i+1$ 的更新，其中 $1 \le i \le L-1$。

6.4 定理 1.1 的证明 在本节中，我们完成定理 1.1 的证明。我们首先在第 6.4.1 小节证明一些辅助断言，这些断言将在第 6.4.2 小节的更新时间分析以及第 6.4.3 小节定理 1.1 的证明中使用。

6.4.1 辅助断言 给定算法 4 得到的 (L, j)-层次结构 $\mathcal{H}_0, \ldots, \mathcal{H}_L$，我们首先对每个 $\mathcal{H}_i$ 的大小给出一个上界。接着，我们将对每棵 $O(j_i)$-树 $H_i \in \mathcal{H}_i$ 中的总重配置量进行界定，并以此求得 $\mathcal{H}_i$ 的总重配置量。在此过程中，我们将看到定理 1.1 中的假设 $U = \text{poly}(n)$ 如何在分析中发挥作用。

CLAIM 6.11. *For every* $0 \le i \le L$, $|\mathcal{H}_i| = \log^{O(i)} n$.

证明。 我们对 i进行归纳证明。

- 基本情况：对于 $i = 0$，这显然成立，因为根据定义 6.4，$\mathcal{H}_0 = \{G\}$。
- 归纳步骤：假设 $|\mathcal{H}_i| = \log^{O(i)} n$。引理 6.10 保证，对于每个 $H_i \in \mathcal{H}_i$，都有$|\mathcal{H}(H_i)| = O(\log n)$。因此，我们有

$$|\mathcal{H}_{i+1}| \le O(\log n)|\mathcal{H}_i| = \log^{O(i+1)} n.$$

□

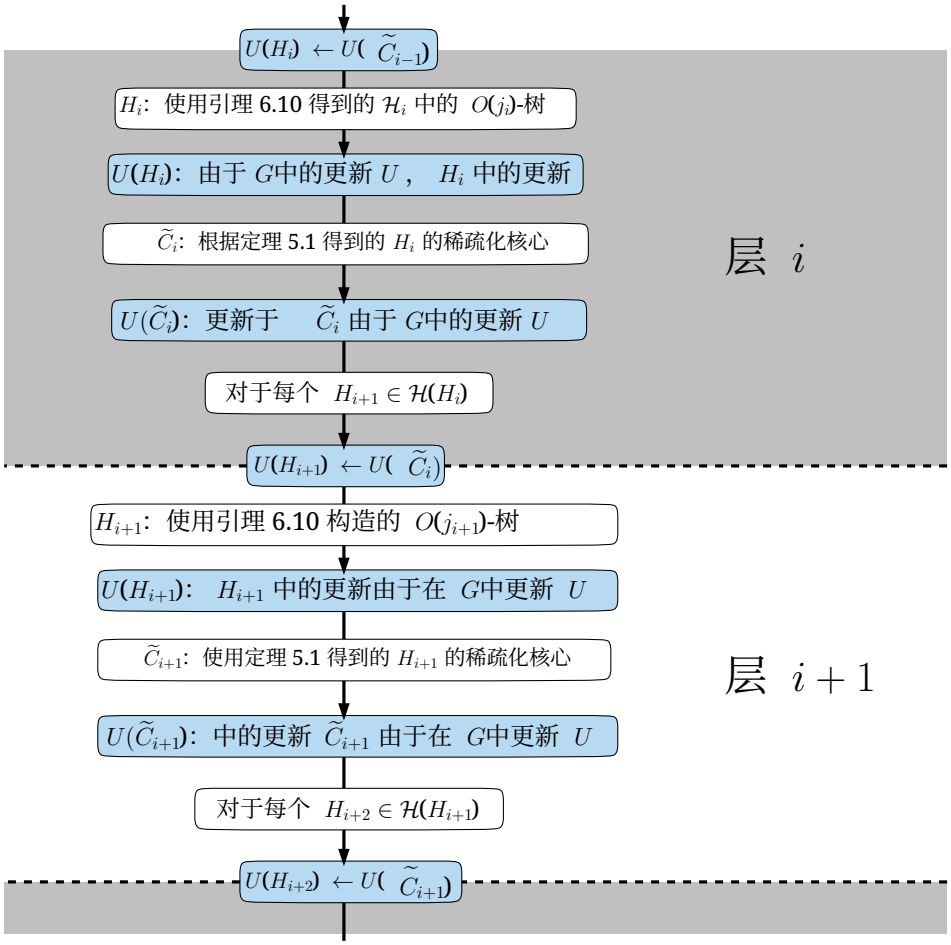


图 6.2：更新总结。

断言 6.12. 设 $H_0, \ldots, H_L$ 是层次结构中的任意一条链，且设 γ_{rec} 是 H_i中稀疏化核心 C_i 在单次更新后的~摊还重构代价的上界。那么，在 G中经过 $l = \Omega(j_1)$ 次总更新后，若 H_i 尚未被算法 4重新初始化，则在 G 中的更新会导致 $\widetilde{C}_i$ 中的 $O\left(l\gamma_{rec}^{i+1}\right)$ 总重配置量。

证明。 回想一下，在 G 中进行更新后，数据结构会将更新从一层传递到下一层，直到 C_k 中 $O(j_k)$-树 $H_k \in \mathcal{H}_k$ 的核心大小超过值 cj_k，其中 c 是算法在过程开始前指定的常数。因此，如果更新后没有重新初始化 $H_i \in \mathcal{H}_i$ ，那么 H_0、.....、H_{i-1} 也不会重新初始化。

我们对 i 使用归纳法进行证明。

- 基本情况： 根据定义 6.4，由于 $\mathcal{H}_0 = \{G\}$ ，在 $C_0 = G$ 中进行 l 次更新会导致 C_0 中的总重配置量为 l ，因此根据定理 5.1， $O(l\gamma_{\text{rec}})$ 是 C_0 中总重配置量的上界。
- 归纳法步骤：假设 $\widetilde{C}_i$ 中的总重调整次数为 $O\left(l\gamma^{i+1}{}_{\text{重配置量}}\right)$。由于 H_{i+1} 定义在 C_i的顶点~集合上，因此传递给 H_{i+1} 的更新总数不能超过 C_i中的总重调整次数。因此，$O\left(l\gamma^{i}{}_{\text{重配置量}}\gamma_{\text{重配置量}}\right) = O\left(l\gamma^{i+1}{}_{\text{重配置量}}\right)$ 可作为 $\widetilde{C}_{i+1}$中总重调整次数的上界。

命题 6.13. 给定一个含有 n 个顶点的图 $G = (V, E, u)$，其容量比上界为 $U = \text{poly}(n)$。那么，在 G 中经历 $l = \Omega(j_1)$ 次总更新后，如果 $\mathcal{H}_i$ 尚未被算法 4 重新初始化，则 G 中的更新会导致 $\mathcal{H}_i$ 的稀疏化核心产生 $l\log^{O(i)} n$ 次总重调整。

证明。 根据命题 6.12， $H_i \in \mathcal{H}_i$ 的每个稀疏化核心 $\widetilde{C}_i$ 都会经历 $O\left(l\gamma_{\mathrm{rec}}^{i+1}\right)$ 次重调整。根据 Claim 6.11，$|\mathcal{H}_i| = \log^{O(i)} n$。因此， $\mathcal{H}_i$ 中稀疏化核心的总重调整次数为

$$O\left(\sum_{H_i \in \mathcal{H}_i} l\gamma_{\mathrm{rec}}^{i+1}\right) = l\gamma_{\mathrm{rec}}^{i+1} \log^{O(i)} n = l \log^{O(i)} n,$$

其中，最后一个等式由以下事实和假设得出：根据定理 4.7， $\gamma_{\mathrm{rec}} = \mathrm{poly}(\log n)\log U = \log^{O(1)} n$，且 $U = \mathrm{poly}(n)$。 □

断言 6.14. 给定一个具有 n 个顶点的图 $G = (V, E, u)$，其容量比的上界为 $U = \mathrm{poly}(n)$。那么，在 G 中经过 $l = \Omega(j_1)$ 次总更新后，集合 $\mathcal{H}_i$ 被算法 4重新初始化的次数为 $O\left(l\log^{O(i)} n/j_i\right)$。

证明。 回顾第 4 节，每次对一棵 j-树进行更新，最多会有 $O(1)$ 个顶点移入其核心。因此，算法 4 中的一棵 $O(j_i)$-树会在以下任一条件成立时重新初始化：(1) 其核心大小超过 cj_i；或 (2) 在 $\mathcal{H}_k, k > i$ 中存在一棵 $O(j_k)$-树，其核心大小超过 cj_k。

我们对 i 进行归纳法证明。令 N_i 表示 $\mathcal{H}_i$ 被重新初始化的次数。

- 基本情况：由于根据定义 6.4，集合 $\mathcal{H}_0$ 在 G 中经过 $O(j_0)$ 次更新后会重新初始化，这意味着 $N_0 = O(l/j_0)$。
- 归纳步骤：假设 $\mathcal{H}_i$ 已被重新初始化 $O\left(l\log^{O(i)} n/j_i\right)$ 次。根据上文第 (2) 项，每次这样的重新初始化也会触发对 $\mathcal{H}_{i+1}$ 的重新初始化。因此，我们只需界定由第 (1) 项导致的重新初始化次数，即求出一个上界 k，用于限制在 $\mathcal{H}_i$ 尚未重新初始化时 $\mathcal{H}_{i+1}$ 被重新初始化的次数。

  为此，令 l_i 表示 G 中导致 $\mathcal{H}_i$ 首次重新初始化的更新次数。根据断言 6.13，当 $l_i \log^{O(i)} n = cj_i$ 时会发生这种情况，因为 $\widetilde{C}_i$ 中的重调整次数上界了因这些更新而移至 C_i 的顶点数。通过直接计算可得 $k = O(l_i/l_{i+1}) = j_i \log^{O(1)} n/j_{i+1}$。因此，我们有

$$N_{i+1} = O(N_i + kN_i) = \frac{j_i}{j_{i+1}} \log^{O(1)} n \left(\frac{l}{j_i} \log^{O(i)} n\right) = \frac{l}{j_{i+1}} \log^{O(i+1)} n.$$ □

6.4.2 总更新时间 给定算法 4 中的一个 (L, j)-层次结构 $\mathcal{H}_0, \ldots\ldots, \mathcal{H}_L$，我们先分析一个 $O(j_i)$-树 $H_i \in \mathcal{H}_i$ 的总更新时间，然后以此求得 $\mathcal{H}_i$ 的总更新时间。在此过程中，我们将看到，对 j_i 的取值进行这样的选择，如何得到所需的更新时间。具体而言，我们将用到公式 (6.3) 所给出的关键事实： $j_{i-1}/j_i = (n/j)^{1/L}$。

引理 6.15. 给定一个有 n 个顶点的图 $G = (V、E、u)$，其容量比的上界为 $U = \mathrm{poly}(n)$。那么，在 G 中进行 $l = \Omega(j_1)$ 次总更新后，维护一个 $O(j_i)$-树 $H_i \in \mathcal{H}_i$ 及其稀疏化核心 C_i 的总更新时间为

$$l\left(\frac{j_{i-1}^2}{j_i^2} + \frac{j_{i-1}}{j_i}\right) \log^{O(i)} n$$

对每个 $1 \le i \le L$ 均成立。

证明。We ~分别讨论 $O(j_i)$-树 $H_i \in \mathcal{H}_i$ 及其稀疏化核心 C_i 的总时间 .

- 对于 H_i：根据引理 6.10，在 H_i 重新初始化之前，初始化并维护 H_i 所花费的总时间为$\widetilde{O}\left(l_i^2/j_i\right)$，其中 l_i 是 $\widetilde{C}_{i-1}$（H_{i-1}）的初始边数，而 $\widetilde{C}_{i-1}$是其父节点 $H_{i-1} \in \mathcal{H}_{i-1}$的稀疏化核心。

~ ~ ~由于 C_{i-1}仅包含 O（$j_{i-1}\log\ U$）条边，因此 $l_i = O$（$j_{i-1}\log\ U$），所以总时间为$\widetilde{O}\left(j_{i-1}^2/j_i \log^2 U\right)$。

- 对于 $C_i(H_i)$：根据定理 4.7，在 $\widetilde{O}(j_i)$次更新期间初始化并维护稀疏化核心 $C_i(H_i)$ 所花费的总时间为 $O(l_i \log U)$，其中 l_i 是 H_i中的初始边数。由于 H_i 构建于 C_{i-1}之上，$l_i \le |E(C_{i-1})| = O(j_{i-1} \log U)$。因此，在重新初始化之前维护 C_i所花费的总时间为$\widetilde{O}(j_{i-1} \log^2 U)$。

因此，在重新初始化之前维护 C_i 和 H_i 所需的总时间为

$$\widetilde{O}\left(\frac{j_{i-1}^2}{j_i} \log^2 U + j_{i-1} \log^2 U\right).$$

根据命题 6.14，H_i 会被重新初始化 $O\left(l \log^{O(i)} n / j_i\right)$ 次。因此，维护 H_i 所需的总时间为

$$\frac{l \log^{O(i)} n}{j_i}\left(\frac{j_{i-1}^2}{j_i} + j_{i-1}\right)\log^2 U = l\left(\frac{j_{i-1}^2}{j_i^2} + \frac{j_{i-1}}{j_i}\right)\log^{O(i)} n. \qquad \square$$

引理 6.16. 设 n 顶点图 $G=(V,E,u)$ 的容量比上界为 $U=\mathrm{poly}(n)$。那么，在 G 中进行 $l=\Omega(j_1)$ 次总更新后，维护 $\mathcal{H}_i$ 的总更新时间为

$$l\left(\frac{n}{j}\right)^{2/L}\log^{O(i)} n$$

对每个 $1 \le i \le L$ 均成立。

证明。结合引理 6.15 中每个 $H_i \in \mathcal{H}_i$ 的总更新时间与断言 6.11 中 $\mathcal{H}_i$ 的大小，我们得出 $\mathcal{H}_i$ 的总时间为

$$\begin{aligned}\left(l\left(\frac{j_{i-1}^2}{j_i^2} + \frac{j_{i-1}}{j_i}\right)\log^{O(i)} n\right)\log^{O(i)} n &= l\left(\frac{j_{i-1}^2}{j_i^2}\right)\log^{O(i)} n \\ &= l\left(\frac{n}{j}\right)^{2/L}\log^{O(i)} n,\end{aligned}$$

因为根据公式 (6.3)，$j_{i-1}/j_i = (n/j)^{1/L}$。 $\square$

在本节余下部分，我们证明定理 1.1。

6.4.3 定理 1.1 的证明。我们以证明定理 1.1 结束本节。

该算法以高概率正确维护一个 (L,j)-层次结构，这一事实由引理 6.10 的正确性推出：(1) 对定理 5.1 中所有数据结构的失败概率应用并集界；(2) 注意到，任意时刻 $\mathcal{H}_L$ 中的图都构成 $O(j_i)$-树，因为一旦存在一个核心大小为 $|V(C_i)| > cj_i$ 的 $O(j_i)$-树 $H_i \in \mathcal{H}_i$，算法 4 就会重新初始化$\mathcal{H}_i, \ldots, \mathcal{H}_L$，其中 c 是算法在过程开始前指定的常数。$\mathcal{H}$ 的正确性则由引理 6.6 得出。

还剩证明 (1) 以高概率保持 $\widetilde{O}(\log)^L$ 条中的割；(2) 关于$|\mathcal{H}|$的界，(3) 显式维护 $\mathcal{H}$，(4) 容量比，以及 (5) 更新时间。

O以高概率保持 G 中的割（log）^{L}：根据定义，我们需要证明，在任意时刻，对于 G 中的任意割 $(S, V\setminus S)$，以高概率存在一个图 $H \in \mathcal{H}$，使得 $U_H(S) \le \widetilde{O}(\log n)^L U_G(S)$。由于每个 $H \in \mathcal{H}$ 都唯一对应一条链 $H_0, \ldots, H_L$（即 $H = H^{H_0,\ldots,H_L}$），只需对 i 使用归纳法，证明以高概率存在一条链，使得 $U_{H^{H_0,\ldots,H_i}}(S) \le \widetilde{O}(\log n)^i U_G(S)$。

- 基本情况：根据引理 6.10，以高概率存在一个图 $H_1 \in \mathcal{H}_1$，使得 $U_G(S) \le U_{H_1}(S) \le \widetilde{O}(\log n) \cdot U_G(S)$。基本步骤由 $H^{H_0,H_1} = \widetilde{C}_1 \cup F_1$ 得出，这是根据公式 (6.1)，并且利用引理 6.6 的证明可知，$\widetilde{C}_1 \cup F_1$ 是一个 $O(j_1)$-树 H_1。

- 归纳步骤：假设以高概率存在 $H_0, H_1, \ldots, H_i$，使得 $U_{H^{H_0,\ldots,H_i}}(S) \le \widetilde{O}(\log n)^i U_G(S)$。根据引理 6.10，以高概率存在一个图 $H_{i+1} \in \mathcal{H}(H_i)$，其核心为 C_{i+1}

~ ~ ~和森林 F_{i+1}，它对 $V(C_i)$ 中的割 S 具有 $O(\log n)$-保留性质。由于 C_{i+1} 是C_{i+1}的 2-近似割稀疏器，因此，以高概率，图 $\widetilde{C}_{i+1} \cup F_{i+1} \cup \cdots \cup F_1$ $\widetilde{O}(\log\ n)^{i+1}$能够保留 G中的割S 。

$\mathcal{H}$的大小：由于 $\mathcal{H}_L$ 中的图与层次结构中的链一一对应，因此如 Claim 6.11 所述，$|\mathcal{H}| = |\mathcal{H}_L|$ 。该结论可由 Claim 6.11 立即推出。

显式维护 $\mathcal{H}$：对于层次结构中的每一条链 H_0、.....、H_L，数据结构都会显式维护森林 F_i 和稀疏化核心 $\widetilde{C}_i$ 。因此，我们可以通过在定理 1.1 的维护过程中更新集合，显式维护 H^{H_0}、.....、^{H_L} ：

~ =

$C_L \cup F_L \cup \cdots \cup F_1$。如果数据结构重新初始化了某些集合，则可以在 $L\log^{O(L)} n$ 最坏情况更新时间内更新集合 $\mathcal{H}$，这不会增加数据结构的更新时间开销。这是因为 $|H| = \log^{O(L)} n$，且每条链都是 L个集合的并集。

容量比：由于 $\widetilde{C}_0$是由 $C_0 = G$得到的，根据定理 5.1，$\widetilde{C}_0$的容量比为 $O(mU) = O\left(n^2U\right)$。其余部分可直接由归纳法得出，且依据以下事实：在每一层 $1 \le i \le L$，数据结构都会使容量比增加 $O\left(n^2\right)$ 开销；此外，由于对核心 C_L进行稀疏化，还会产生 $O(n)$的额外开销：(1) 根据假设，每个 C_i 包含 $\widetilde{O}(n) = n\log^{O(1)} n$ 条边；(2) 根据定理 5.1，稀疏化图 C_i 的容量比是核心图 C_i的 $O(n)$倍；(3) 根据引理 6.10，$\widetilde{C}_i$ 的 $O(j_{i+1})$-树 $H_{i+1} = C_{i+1} \cup F_{i+1}$ 的容量比是 $\widetilde{C}_i$的 $O\left(n^2\right)$ 倍。

更新时间：利用引理 6.16 中 $\mathcal{H}_i$ 的总更新时间，在 G 中经过 m 次更新后维护 $(L,\ j)$-层次结构的总更新时间为

$$\sum_{i=0}^{L} m\left(\frac{n}{j}\right)^{2/L} \log^{O(i)} n = Lm\left(\frac{n}{j}\right)^{2/L} \log^{O(L)} n = m\left(\frac{n}{j}\right)^{2/L} \log^{O(L)} n,$$

其中，最后一个等式由定理 1.1 中的平凡上界 $L = O(\log n)$ 得出。注意，定理 1.1 中关于 L 的上界确保了项 $\log^{O(L)} n$ 不会在更新时间中超过 $(n/j)^{2/L}$，如定理 6.2 中所述。该数据结构的摊销更新时间可直接得出，因为所计算的时间是在 m 次更新后的总更新时间。

7 应用本节将说明如何利用定理 1.1 的数据结构，加速动态场景中的一些核心问题。具体而言，我们讨论以下问题。

- 动态全点对最小割和动态全点对最大流，详见第 7.1 小节 .
- 动态多路割和多重割，详见第 7.2 小节。
- 动态最稀疏割见第 7.3 小节。

常见的方法是利用 $O(j)$-树的结构性质，并结合定理 1.1 的保证。这是因为，与一般图相比，这些问题在树上更容易求解。由于我们可以利用定理 1.1 维护一棵 $O(j)$-树，因此我们将说明如何高效维护该~森林中 j-树上的解，并在查询时，仅在具有 $O(j)$ 个顶点和边的稀疏化核心上求解问题。每个问题都利用了 j-树的特定结构性质，因此我们将分别进行讨论。

7.1 动态近似全点对最小割和全点对最大流 给定一个图$G = (V,\ E,\ u)$、一个汇点 $s \in V$和一个源点 $t \in V$，最小 s-t 割是指一个割 $(S,\ V \setminus S)$，使得在所有分隔 s 和 t的此类割中，$s \in S$、$t \in V \setminus S$ 和 $U_G(S)$ 最小。

在全动态全点对最小割问题中，输入图 G 会经历边插入和删除，目标是维护查询接口，以便查询任意两个顶点 s、$t \in V$之间的 s-t 最小割值。本节将展示如何利用定理 1.1 中的数据结构，以高概率维护对 $\widetilde{O}(\log\ n)^L$ 倍近似的 s-t 最小割值的查询接口。

定理 7.1. 给定一个无向的 n 顶点图 $G = (V, E, u)$，其容量比为多项式有界，且经历多项式次边插入和删除，参数为 $j \ge 1$ 和 $1 \le L \le o\left(\sqrt{\log(n/j)/\log\log n}\right)$，则存在一种数据结构，能够在非自适应对手的情况下支持以下操作：

- 插入边(u,v,c)：将一条边 (u,v) 的容量从 c 增至 G，摊销时间为 $O((n/j)^{2/L}\log^{O(L)} n)$。
- 删除边(u,v)：在摊销时间 $G(u,v)$ 内，从 O 中移除边 $G((n/j)^{2/L}\log^{O(L)} n)$。
- 最小割(s,t)：在时间 $O(j\log^{O(L)} n)$ 内，返回 $\widetilde{O}(\log n)^L$-近似的 s-t 最小割值。

该算法以高概率正确。

通过设置 $j=n^{2/(2+L)}$，使查询时间和更新时间达到平衡，并通过在更新过程中维护一个解，将查询时间的开销转移到更新时间中，我们尤其得到以下结果。

推论 7.2. 对于任意固定整数 $L\geq 1$，存在一种动态数据结构，在面对非自适应对手时以高概率返回 $\widetilde{O}(\log n)^L$-近似的 s-t-最小割值，其摊销更新时间为 $O(n^{\frac{2}{2+L}}\log^{O(L)} n)$，查询时间为 $O(1)$。

推论 7.3. 存在一种动态数据结构，在非自适应对手下以高概率返回 $O(\mathrm{polylog}(n))$-近似的 s-t-最小割值，摊销更新时间为 $O(n^{0.001})$，查询时间为 $O(1)$。

令 L 随 n 增长，并设置 $L=\Theta(\sqrt{\log n}/\log\log n)$，还可得到一个更新时间为次多项式的数据结构。

推论7.4. 存在一种动态数据结构，在非自适应对手下以高概率返回$e^{O(\log^{1/2} n\log\log n)}$-近似的 s-t-最小割值，摊销更新时间为 $e^{O(\log^{1/2} n\log\log n)}$，查询时间为 $O(1)$。

这里我们指出，[59] 中定理 4.1 的数据结构也具有次多项式近似以及查询/更新时间，但它还适用于自适应对手。另一方面，他们的次多项式因子 $e^{O(\log^{3/4}\log\log n)}$ 比我们的数据结构更差。

在证明这一定理之前，我们先陈述以下简单但强大的结构性观察。

观察 7.1。设 $H=C\cup F$ 是一个带有森林 F 和核心 C 的 j-树。设 s、$t\in V$ 满足 $s\notin V(C)$、$t\in V(C)$。令 $e^F_{\min}(s)$ 为沿着 s-根$_F(s)$ 路径的最小容量边，并令 ν 为根$_F(s)$ 与 t 之间的最大流值。那么，s-t 最大流值为 $\min(u_H(e^F_{\min}(s))\ \nu)$。

定理 7.1的证明。我们使用定理 1.1 中的数据结构，该数据结构保证，在对 G 进行任意更新后，对于任意割 S，都有 $U_G(S)\leq U_H(S)$，对所有 $H\in\mathcal{H}$ 均成立。此外，以高概率存在一个 $H\in\mathcal{H}$，使得 $U_H(S)\leq\widetilde{O}(\log n)^L U_G(S)$。我们可以对针对 G 执行的所有更新中所有 $O(n^2)$ s-t 最小割的失败概率应用并集界，因此可以假设在任何时刻，对于任意 s-t 最小割S，都存在一个 H，使得 $U_H(S)\leq\widetilde{O}(\log n)^L U_G(S)$。因此，如果我们能够为所有图 $H\in\mathcal{H}$ 计算 2-近似的 s-t 最小割值，那么返回所找到的最小值将得到所需的近似。

接下来需要说明如何高效计算 $\mathcal{H}$ 中各图的 s-t 割（等价地，s-t最大流）值的 2-近似。为此，记住图 $H\in\mathcal{H}$ 唯一对应于一条链 H_0，H_1，……，H_L，且 $H=C_L\cup F_L\cup F_{L-1}\cup\ldots F_1$。如果 s，$t\in V(C_L)$，那么 H 中的 s-t 最大流值等于 C_L 中的 s-t 最大流值。为了计算该值，我们可以使用 [49] 中近似线性的 2-近似最大流算法，该算法的运行时间为 $O(|E(C_L)|)=O(j)$，其中我们利用了 $|E(C_L)|=O(j)$。

现在假设 s 或 t 或二者均不在核心中。令 $u_H(e^F_{\min}(s))$ $u_H(e^F_{\min}(t))$ 表示 F 中沿 s-根$_F(s)$ 路径以及 t-根$_F(t)$ 路径上的最小容量边的容量，另令 ν 表示 C_L 中根$_F(s)$-根$_F(t)$ 最大流值。

根据观察 7.1，H 的 s-t 最大流值为 $\min(\nu,\ u_H(e^F_{\min}(s))\ u_H(e^F_{\min}(t)))$。现在只需注意到，通过在 F_i 的每个森林上维护引理 3.10 中的链接切割树数据结构（事实上，算法已经维护了该数据结构，正如算法 1 所示），我们可以在 $O(L\log n)$ 时间内找到最小容量边 $e^F_{\min}(s)$ $e^F_{\min}(t)$。此外，我们可以通过 [49] 计算根$_F(s)$-根$_F(t)$ 最大流值的 2-近似。

因此，我们可以在 $O(j\log^{O(L)} n)$ 时间内确定图 H 的 s-t 最小割值的 2-近似值，因为该集合满足 $|\mathcal{H}|=O(\log^L n)$。□

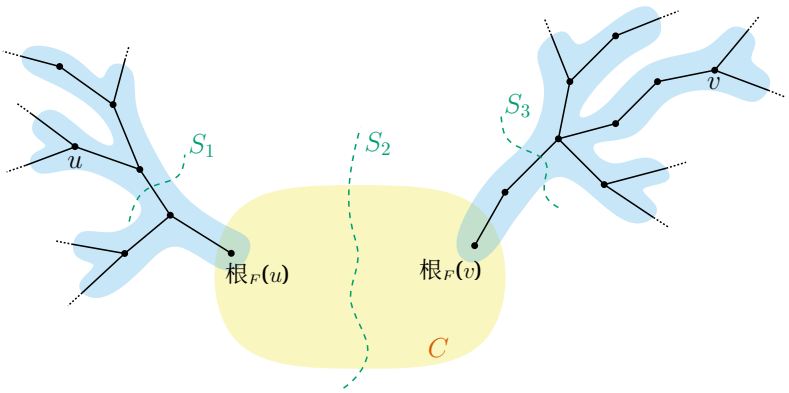


图 7.1：观察 7.1 的图示，以及我们的算法如何在一棵 j-树中找到一个（近似的） u-v-最小割。 S_1, S_3 分别对应于将 u, v 与其在 F 中的根分隔开的最小割。然后， S_2表示核心 C 中根$_F(u)$与根$_F(v)$之间的最小割。最小的 u-v 割对应于这三个候选方案中容量最小的割。

7.2 动态多路割与多重割

定义 7.5. 设 $G = (V, E, u_G)$ 为一个图。若一个子集的集合 $\{S_i\}_{i=1}^k$ 是 V 的一个划分，且 $V = \bigcup_{i=1}^k S_i$，并且对所有 $i \neq j$，都有 $S_i \cap S_j = \emptyset$，则称其为 V 的一个划分。给定一个划分 $\{S_i\}$，跨越两个不同连通分量的边的总容量定义为 $U_G(\{S_i\}) := \frac{1}{2}\sum_{i=1}^k U_G(S_i)$。

给定一个图 $G = (V, E, u)$ 和终端点集合 $T \subseteq V$，相对于 T， G 的多路割是 V 的一个划分 $\{S_i\}$，其中 T 的每个节点都属于不同的划分部分。最小代价多路割问题的目标是找到 G 相对于 T 的一个多路割 $\{S_i\}$，使 $U_G(\{S_i\})$ 最小。

给定一个终端点对集合 $T = \{s_1, t_1 \dots\}$，其中 $t_i, s_i \in V$，相对于 T， G 的多重割是 V 的一个划分 $\{S_i\}$，将 T 中的每个终端点对彼此分隔开。最小代价多重割问题的目标是找到 G 相对于终端点对 T 的一个多重割 $\{S_i\}$，使 $U_G(\{S_i\})$ 最小。

下面的定理说明，如何使用定理 1.1 的数据结构，在经历边插入和删除的图上回答近似多路割和多重割查询。其证明推迟到第 D 节。

定理7.6. 对于一个具有 n 个顶点、容量比多项式有界的图 $G = (V, E, u)$，在经历多项式有界的边插入和删除时，给定参数 $j \geq 1$ 和 $1 \leq L \leq o\left(\sqrt{\log(n/j)/\log\log n}\right)$，存在一个数据结构，能够在非自适应对手面前支持以下操作：

- 删除边(u, v, c)：以 $O((n/j)^{2/L}\log^{O(L)} n)$ 摊销时间，将容量为 c 的边 (u, v) 插入到 G 中。
- 删除边(u, v)：以 $O((n/j)^{2/L}\log^{O(L)} n)$ 摊销时间，从 G 中移除边 (u, v)。
- 多路割（$T \subseteq V$）：在 G 关于终端集合 T 的 $\widetilde{O}\left(k \cdot (n/j)^{2/L}\log^{O(L)} n + j\right)$ 摊还时间内，返回最小成本多路割值的一个 $\widetilde{O}(\log n)^{L+1}$-近似。
- 多重割（$T = \{(s_i, t_i) \mid i \in [k]\}$）：在 G 关于终端点对集合 T 的 O中，返回最小成本多重割值的一个 $\widetilde{O}(\log n)^{L+2}$-近似 $\left(k \cdot (n/j)^{2/L}\log^{O(L)} n + (k+j)^2\right)$ 摊还时间。

该算法以高概率保证正确。

具体而言，定理 7.6 可用于得到以下推论：

推论 7.7. 对于任意固定整数 $L \geq 1$，存在一种动态数据结构，能够以高概率在摊销更新时间 $O(\log^{O(L)} n^{1+\frac{2}{L}})$ 的条件下，对抗非自适应对手，并以 $\widetilde{O}(\log n)^{L+2}$ 近似因子返回相对于大小为 k 的终端集合的最小多路割和多割值，其摊还查询时间分别为 $\widetilde{O}(k\log^{O(L)} n^{1+\frac{2}{L}})$ 和 $\widetilde{O}(k\log^{O(L)}(k + n^{1+\frac{2}{L}}))$。

推论7.8. 存在一种动态数据结构，其摊销更新时间为 $O(n^{0.001})$ ，能够对抗非自适应对手，并以 $O(\mathrm{polylog}(n))$ 近似因子返回相对于大小为 k 的终端集合的最小多路割和多割值，其摊还查询时间分别为 $\widetilde{O}(kn^{0.001})$ 和 $\widetilde{O}(kn^{0.001}+k^2)$ ，且该结论以高概率成立。

与前一节类似，通过令 L 随 n 增长，并设置 $L=\Theta(\sqrt{\log n}/\log\log n)$，我们同样可以得到一个具有亚多项式更新时间的数据结构，并能以几乎 $\mathrm{poly}(k)$ 的时间回答查询。

推论 7.9. 存在一种动态数据结构，其摊销更新时间为 $O(e^{O(\log^{1/2} n\log\log n)})$，在非自适应对手下，能够以摊还查询时间 $O(k\cdot e^{O(\log^{1/2} n\log\log n)})$ 和 $\widetilde{O}(ke^{O(\log^{1/2} n\log\log n)}+k^2)$，分别以高概率返回关于大小为 k 的终端集合的最小多路割和多割值的 $e^{O(\log^{1/2} n\log\log n)}$ 近似。

7.3 动态最稀疏割 令 $G=(V,E,u)$ 为一个图。割 $(S,V\setminus S)$ 的稀疏度 定义为

$$\psi_G(S)=\frac{U_G(S)}{\min\{|S|,|V\setminus S|\}},$$

且 G的最稀疏割值定义为

$$\psi(G):=\min_{S\subseteq V}\psi_G(S).$$

在全动态最稀疏割问题中，输入图 G 会经历边插入和删除，目标是维护对 G中最稀疏割值的查询能力。在本节中，我们将说明，定理 1.1 中的数据结构可用于维护对最稀疏割的 $\widetilde{O}(\log n)^{L+1/2}$-近似的查询能力。证明推迟至第 E 节。

定理 7.10. 给定一个具有 n个顶点的图 $G=(V,E,u)$，其容量比为多项式有界，且经历多项式次边插入和删除，参数为 $j\geq 1$ 和 $1\leq L\leq o\Big(\sqrt{\log(n/j)}/\log\log n\Big)$，则存在一种数据结构，可在非自适应对手的情况下支持以下操作：

- 插入边(u,v,c) ：以摊还时间 $O((n/j)^{2/L}\log^{O(L)} n)$ 将容量为 c 的边 (u,v) 插入到 G。
- 删除 边(u,v)：以摊还时间 $O((n/j)^{2/L}\log^{O(L)} n)$ 从 G 中移除边(u,v)。
- 最稀疏割()：在时间 $O(j^2\log^{O(L)} n)$ 内返回 G 的最稀疏割值的 $\widetilde{O}(\log n)^{L+1}$-近似。

该算法以高概率正确。

通过设置 $j=n^{1/(1+L)}$ 以平衡查询时间和更新时间，并通过在更新期间维护一个解，将查询时间的开销转移到更新时间中，我们特别得到以下结果。

推论7.11. 对于任意固定整数 $L\geq 1$，存在一种动态数据结构，能够以高概率在面对非自适应对手时，返回最稀疏割值的一个 $\widetilde{O}(\log n)^{L+1}$ 近似，其摊销更新时间为 $O(\log^{O(L)} n\, n^{\frac{2}{1+L}})$，查询时间为 $O(1)$。

推论 7.12. 存在一种动态数据结构，能够以高概率在面对非自适应对手时，返回最稀疏割值的一个 $O(\text{多对数}(n))$ 近似，其摊销更新时间为 $O(n^{0.001})$ ，查询时间为 $O(1)$ 。

与前一节类似，让 L 随 n 增长，并设置 $L=\Theta(\sqrt{\log n}/\log\log n)$，我们也能得到一个具有亚多项式更新时间的数据结构。

推论7.13. 存在一种动态数据结构，能够以高概率在非自适应对手面前，以摊销更新时间为$e^{O(\log^{1/2} n\log\log n)}$、查询时间为$O(1)$的代价，返回最稀疏割值的 $O(e^{O(\log^{1/2} n\log\log n)})$ 近似。

8 致谢 Monika Henzinger：由欧盟资助。本文所表达的观点和意见仅代表作者本人，并不一定反映欧盟或欧洲研究理事会执行机构的观点。欧盟及其资助机构均不对这些观点和意见承担责任。

本项目获得了欧洲研究委员会（ERC）在欧盟地平线2020研究与创新计划（MoDynStruct，编号：101019564）下提供的资助，以及奥地利科学基金（FWF）拨款（DOI：10.55776/I5982）的资助。为实现开放获取，作者已对因本次投稿产生的任何作者已接受稿版本申请了知识共享署名公共版权许可。



Peter Kiss：本研究全部或部分由奥地利科学基金（FWF）资助，项目编号 10.55776/ESP6088024。

辅助引理的证明我们先证明定理 3.13。

引理 A.1. /参见引理 7。9，见 /21/ 和第 A/ 节。设 $G=(V,E)$ 是一个初始时具有 n 个顶点和 m 条边的动态图，随着时间推移进行 m 次边更新和 n次顶点分裂。那么，这一系列顶点分裂可以通过 $O(m\log n)$ 次边插入和删除来模拟。

证明。令 $G^{(t)}$ 为时刻 t 的图，并考虑以下势函数

$$\Phi^{(t)} := \sum_{v\in V(G^{(t)})} \deg_{G^{(t)}}(v)\log \deg_{G^{(t)}}(v).$$

显然，在初始化时 $\Phi^{(0)} \le 2m\log n$；若 $G^{(t+1)}$ 与 $G^{(t)}$ 相差一次边插入/删除，则有 $\Phi^{(t+1)} \le \Phi^{(t)} + 2\log n$。因此，所有边更新中势函数的总增量至多为 $2m\log n$。现在考虑将 v 拆分为 u 和 w 的顶点拆分。此时势函数按如下方式递减:

$$\begin{aligned}
&\Phi^{(t)} - \Phi^{(t+1)} = \deg_{G^{(t)}}(v)\log\deg_{G^{(t)}}(v) - \deg_{G^{(t+1)}}(u)\log\deg_{G^{(t+1)}}(u) - \deg_{G^{(t+1)}}(w)\log\deg_{G^{(t+1)}}(w)\\
&= \deg_{G^{(t+1)}}(u)\cdot(\log\deg_{G^{(t)}}(v) - \log\deg_{G^{(t+1)}}(u)) + \deg_{G^{(t+1)}}(w)\cdot(\log\deg_{G^{(t)}}(v) - \log\deg_{G^{(t+1)}}(w))\\
&\ge \deg_{G^{(t+1)}}(u)\cdot(\log\deg_{G^{(t)}}(v) - \log\deg_{G^{(t+1)}}(u))\\
&\ge \deg_{G^{(t+1)}}(u)\cdot(\log 2\deg_{G^{(t+1)}}(u) - \log\deg_{G^{(t+1)}}(u))\\
&= \Omega(\deg_{G^{(t+1)}}(u))
\end{aligned}$$

这里我们用到了 $\deg_{G^{(t)}}(v) = \deg_{G^{(t+1)}}(u) + \deg_{G^{(t+1)}}(w)$，并且不失一般性，可以假设 $\deg_{G^{(t+1)}}(u) \le \deg_{G^{(t+1)}}(v)/2$。

现在注意，我们可以通过先删除 v 中将与 u 相连的边，再将这些边插入并连接到一个孤立顶点，来模拟对 v 的顶点拆分。这需要 $O(\deg_{G^{(t+1)}}(u)) = O(\Phi^{(t)} - \Phi^{(t+1)})$ 次边更新。由于势函数始终满足 $\Phi \ge 0$，初始时满足 $\Phi^{(0)} \le 2m\log n$，且随时间推移，通过边插入最多增加 $2m\log n$，所有顶点拆分都可以通过 $O(m\log n)$ 次边更新来模拟。 □

B 树可嵌入性的反例 这里，我们给出一个广为人知的论证，说明存在稀疏图 G，需要 $\Omega(n/\alpha)$ 棵生成树 $T_i \subseteq G$，才能满足其平均值的可嵌入性 $k^{-1}\sum_{i=1}^{k} T_i \preceq_\alpha G$。这也表明，定理 4.4 和定理 4.9 中的结果几乎是紧的。

该构造基于常数度扩展图。提醒读者，若一个图 d-正则图$G=(V,\ E)$有 n 个顶点，则称其为 δ-扩展图；如果对任意满足 $|S|\le n/2$ 的集合 $S\subseteq V$，都有 $|E(S,\ V\setminus S)| \ge \delta\cdot d\cdot|S|$。

定理B.1 ([50])。 存在一个固定的 $\delta > 0$，使得对于任意 $d \ge 3$ 和偶整数 n，都存在一个d-正则 δ-扩展图 G，其有 n 个顶点。

现在证明如下。

定理 B.2. 设 $G=(V,E)$ 是一个 d-正则 δ-扩展子，且 T_1、.....、$T_k \subseteq G$ 是 G的生成树。设$\alpha \ge 1$ 并假设

$$\frac{1}{k}\sum_{i=1}^{k} T_i \preceq_\alpha G.$$

则 $k \ge \delta n/(2\alpha)$。

证明。 设 $T\subseteq V$ 是 G的一棵生成树，并设 $v\in V$ 是 T的一个顶点分隔点，即一个顶点 v，移除该顶点后，所得的所有子树大小均至多为 $n/2$。由于 G 的最大度为 d（因此 $T\subseteq G$的最大度也如此），必然存在一个所得子树 T'，使得 $|T'|\ge n/(2d)$。但是，由于 G 是一个 δ-扩展子，我们知道 $E(T',\ V\setminus T') \ge \delta d\cdot n/(2d)$。因此，$G$ 到 T 的任何嵌入 f都必须沿连接 T'与 v的边传输 $\delta n/2$ 单位的流。因此，存在一条边 $e\in T$，使得 $u^T(e) \ge \delta n/2$。

特别是，如果 $H := \frac{1}{k}\sum_{i=1}^{k} T_i$，则对于给定的边 e, $u_H(e) \ge \delta n/(2k)$。因此，为了满足 $H \preceq_\alpha G$，我们需要满足 $\delta n/(2k) \le \alpha$，即 $k \ge \delta n/(2\alpha)$。 □

注记 B.3. 类似的论证表明，若 G 是一个 d-正则 δ-扩张图，且 H 是 G的一个诱导 j-树，则必定存在一条边满足 $u_H(e) \ge \Omega(\delta n/j)$，因此需要 $k=\Omega(\delta n/(j\alpha))$个诱导 j-树，才能使它们的平均值 α-嵌入回 G。

C 定理 4.9 的证明 这里，我们要给出定理 4.9 的完整证明。我们认为，本证明比 Madry 在 [46]中给出的证明略为简单。我们证明中的主要区别在于，可以通过直接界定实例的宽度，更标准地应用 MWU 框架。这需要在 Madry 基于低伸展生成树的方法中引入一个新的正则化项。

提醒读者，对于 G 的生成树 T 和边 $e \in E_G$，我们定义了 $\mathsf{cong}_G^T(e) = u^T(e)/u_G(e)$，当 $e \in E_T$ 时成立；否则 $\mathsf{cong}_G^T(e) = 1$。为方便读者，我们现在重述将要证明的定理。

定理 4.9. 设 $G = (V, E_G, u_G)$ 是一个图，j 是一个参数。那么，在 $\widetilde{O}(m^2/j)$ 时间内，可以构造一个由 $i = 1$、2、... ...、$k = O(m/j)$ 棵生成树组成的序列 T_1、.....、T_k，它们都是 G 的生成树，并配有边集 $S_i \subseteq E_{T_i}$，满足 $|S_i| \leq j$，使得对于每条边 $e \in E_G$

$$\frac{1}{k} \sum_{i: e \notin S_i} \mathsf{cong}_G^{T_i}(e) \leq \widetilde{O}(\log n).$$

C.1 乘法权重更新的应用 在本节中，我们将在以下结果成立的前提下证明定理 4.9；该结果将在下一节证明。

引理 C.1. 设 G 是一个图，j 是一个参数，$w \in \mathbb{R}^E_{>0}$ 是任意权重向量。那么，在时间 $\widetilde{O}(m)$ 内，可以找到 G 的一棵生成树 T 和一组边 $S \subseteq E_T$，满足 $|S| \leq j$，使得

1. $\sum_{e \in E_G} w(e) \cdot \mathsf{cong}_G^T(e) \leq \widetilde{O}(\log n) \cdot \|w\|_1$，*and*
2. $\max_{e \notin S} \mathsf{cong}_G^T(e) \leq \widetilde{O}(\log n) \cdot m/j$。

定理 4.9 的证明。我们紧密遵循 [22]中引理 6.6 的证明，并通过标准的乘法权重更新 [12]，按顺序构造这些树。

令 $\gamma \in \widetilde{O}(\log n)$ 满足定理 C.1 中的树满足 $\sum_{e \in E_T} w(e) u^T(e) u_G(e) \leq \gamma \cdot \|w\|_1$，且 $\max_{e \in E_T \setminus S} u^T(e) u_G(e) \leq \gamma \cdot m/j$。令 $k = 10\gamma m/j = \widetilde{O}(\log n) m/j$。我们首先将 $w_0 \in \mathbb{R}^E$ 设为 $w_0 = \mathbf{1}^E$，并针对这个权重函数，使用定理 C.1 计算一棵树 T_0 和集合 S_0。

之后，经过 $t \in [k]$ 次迭代，我们为所有 $e \in E_G$ 定义 $w_t(e) = \exp(\frac{1}{k} \mathbf{1}\{e \notin S_{t-1}\} \mathsf{cong}_G^{T_{t-1}}(e))$，并使用定理 C.1，利用权重 w_t 得到树 T_t 和边集 S_t。

注意，对于所有 $t \in [k]$:

$$\max_{e \in E_G} \frac{1}{k} \sum_{i=0}^{t-1} \mathbf{1}\{e \notin S_i\} \mathsf{cong}_G^{T_i}(e) \leq \log \left(\sum_{e \in E_G} \exp \left(\frac{1}{k} \sum_{i=0}^{t-1} \mathbf{1}\{e \notin S_i\} \mathsf{cong}_G^{T_i}(e) \right) \right) = \log \|w_t\|_1$$

根据我们对 γ 的选择，对于所有 $t \in [k]$:

$$\frac{1}{k} \mathbf{1}\{e \notin S_t\} \mathsf{cong}_G^{T_t}(e) \leq \gamma m/(jk) \leq 1/10$$

因此，我们可以得出结论，对于所有 $t \in [k]$：

$$\begin{aligned}
\|w_t\|_1 &= \sum_{e \in E_G} w_{t-1}(e) \exp \left(\frac{1}{k} \mathbf{1}\{e \notin S_{t-1}\} \mathsf{cong}_G^{T_{t-1}}(e) \right) \\
&\leq \sum_{e \in E_G} w_{t-1}(e) \left(1 + \frac{2}{k} \mathbf{1}\{e \notin S_{t-1}\} \mathsf{cong}_G^{T_{t-1}}(e) \right) \\
&= \|w_{t-1}\|_1 + \frac{2}{k} \sum_{e \in E_G} \mathbf{1}\{e \notin S_{t-1}\} \mathsf{cong}_G^{T_{t-1}}(e) \\
&\leq (1 + 2\gamma/k) \|w_{t-1}\|_1
\end{aligned}$$

这里使用了 $\exp(x) \leq 1 + 2x$ 对于 $x \leq 1/10$ 以及定理 C 的第二项。 1.

最后，我们可以得出：

$$\exp\left(\max_{e\in E_G}\frac{1}{k}\sum_{i=1}^{k}\mathbf{1}\{e\notin S_i\}\mathsf{cong}_G^{T_i}(e)\right)\le\|w_k\|_1\le(1+2\gamma/k)^k\|w_1\|_1\le\exp(2\gamma)m \qquad \square$$

对两边取对数即可完成该引理的证明。

C.2 定理 C.1 的证明 我们的算法所需的唯一构件是一项关于低伸展生成树的结果，现介绍如下。

低伸展生成树 设 $G=(V,E)$ 是一张图，其边上带有长度向量 $l\in\mathbb{R}^E_{>0}$。设 T 是 G 的一棵生成树， d^l_G,d^l_T 分别表示由 l 在 G 和 T 上诱导的最短路径度量。当然，对于所有 u,v，均有 $d^l_G(u,v)\le d^l_T(u,v)$。定义边 $e=(u,v)\in E_G$ 的伸展 $\mathrm{str}^l_T(e)$ 为

$$\mathrm{str}^l_T(e):=\frac{d^l_T(u,v)}{d^l_G(u,v)}.$$

那么，下述定理表明，可以高效地计算出一棵生成树 T，使得其伸展在 $\widetilde{}$平均 意义下仅为 $O(\log n)$ 的一个相当小的倍数。

定理 C.2 ([7]). 给定一张（多重）图 $G=(V,E)$ 以及一个长度向量 $l\in\mathbb{R}^E_{>0}$，存在一种算法$\widetilde{}$，可在 $O(m)$ 时间内计算出 G 的一棵生成树 T，使得

$$\frac{1}{m}\sum_{e\in E}\mathrm{str}^l_T(e)\le\widetilde{O}(\log n).$$

根据该定理，如何证明以下结论是众所周知的。

引理 C.3. 给定一个（多重）图 $G=(V,E)$、一个长度向量 $l\in\mathbb{R}^E_{>0}$ 和一个权重向量 $w\in\mathbb{R}^E_{>0}$，存在一个算法，可在 $O(m)$时间内计算出 G 的一棵生成树 T，使得，

$$\sum_{e\in E}w(e)\mathrm{str}^l_T(e)\le\widetilde{O}(\log n)\cdot\|w\|_1.$$

证明。 在顶点集 V上构造一个辅助图 G_w。对于每条边 $e\in G$，我们添加 $\lceil mw(e)/\|w\|_1\rceil$条多重边。于是，$G_w$ 的任意一棵生成树 T 也是 G的一棵生成树，并且， G_w 中的边数可界定为

$$\sum_{e\in E_G}\lceil mw(e)/\|w\|_1\rceil\le\sum_{e\in E_G}(1+mw(e)/\|w\|_1)=2m.$$

$\widetilde{}$现在我们调用定理 C.2，在 G_w 上计算一棵生成树 T，所需时间为 $O(m)$，其长度函数为 l （这里， G_w 的每条多重边都继承其在 G中的原始权重所对应的长度。）显然，对于边 $e\in G$的每个副本 $e'\in G_w$，其相对于 T的伸展在 G_w 中与边 e 在 G中的伸展相同。因此，根据 G_w中生成树 T 的保证，我们有

$$\sum_{e\in E_G}\mathrm{str}^l_T(e)\cdot\lceil mw(e)/\|w\|_1\rceil=\sum_{e\in G_w}\mathrm{str}^l_T(e)\le\widetilde{O}(\log n)\cdot m.$$

现在我们可以通过计算来完成证明

$$\begin{aligned}\sum_{e\in E_G}w(e)\mathrm{str}^l_T(e)&=\frac{\|w\|_1}{m}\sum_{e\in E_G}\frac{w(e)m}{\|w\|_1}\mathrm{str}^l_T(e)\le\frac{\|w\|_1}{m}\sum_{e\in E_G}\left\lceil\frac{w(e)m}{\|w\|_1}\right\rceil\mathrm{str}^l_T(e)\\&\le\frac{\|w\|_1}{m}\cdot\widetilde{O}(\log n)\cdot m=\widetilde{O}(\log n)\|w\|_1.\end{aligned} \qquad \square$$

利用这一定理，我们现在证明如下结论。

引理 C.4（见 5.4 中的 [46]）。给定一个（多重）图 $G=$（V、E_G）和一个长度向量 $l\in\mathbb{R}^E_{>0}$，存在一个算法，可以在 O（m）时间内计算出一棵生成树 T，使得 G

$$\sum_{e\in E_G} l(e)u^T(e)\le \widetilde{O}(\log n)\cdot\sum_{e\in E_G} l(e)u_G(e)$$

证明。 应用引理 C.3 于 G，其中长度向量为 l，权重向量为 $w(e)=l(e)u_G(e)$。注意，我们有

$$\begin{aligned}\sum_{f\in E_T} l(f)u^T(f) &= \sum_{f\in E_T} l(f)\sum_{e=(u,v)\in E:f\in T[u,v]} u_G(e)\\ &= \sum_{e=(u,v)\in E} d^l_T(u,v)u_G(e)\\ &\le \sum_{e=(u,v)\in E}\frac{d^l_T(u,v)}{d^l_G(u,v)}u_G(e)l(e)\\ &= \sum_{e=(u,v)\in E}\mathrm{str}^l_T(e)w(e)\\ &\le \widetilde{O}(\log n)\sum_{e\in E_G} w(e)\\ &= \widetilde{O}(\log n)\sum_{e\in E_G} l(e)u_G(e).\end{aligned}$$

□

由此可得

定理 C.1 的证明。 应用前述引理，其中长度函数为 $l(e)=w_{(e)u_G\ e}^{+\|w\|_1/m}()$。于是我们得到一棵生成树 T，满足

$$\begin{aligned}\sum_{e\in E_G} l(e)u^T(e) &= \sum_{e\in E_G}\frac{u^T(e)}{u_G(e)}\left(w(e)+\frac{\|w\|_1}{m}\right)\le \widetilde{O}(\log n)\cdot\sum_{e\in E_G} l(e)u_G(e)\\ &= \widetilde{O}(\log n)\cdot\sum_{e\in E_G}\left(w(e)+\frac{\|w\|_1}{m}\right)=\widetilde{O}(\log n)\cdot 2\|w\|_1:=\gamma\cdot\|w\|_1,\end{aligned}$$

这当然特别意味着 $\sum_{e\in E_G} w(e)u^T(e)u_G(e)\le\gamma\cdot\|w\|_1$，其中 $\gamma\in\widetilde{O}$ 为某个（log n），从而证明了该定理的第一个性质。现在考虑集合 $S=\{e\in E_T:\ u^T(e)/u_G(e)\ge\overline{\gamma m/j}\}$，并注意到

$$\begin{aligned}\gamma\cdot\|w\|_1 &\ge \sum_{e\in E_G} l(e)u^T(e)\ge\sum_{e\in S} l(e)u^T(e)=\sum_{e\in S}\frac{u^T(e)}{u_G(e)}\left(w(e)+\frac{\|w\|_1}{m}\right)\\ &\ge\sum_{e\in S}\frac{u^T(e)}{u_G(e)}\frac{\|w\|}{m}\ge|S|\cdot\frac{\|w\|}{m}\cdot\gamma m/j.\end{aligned}$$

因此，$|S|\le j$，根据构造它满足所需条件。 □

D 定理 7.6 的证明 为了使用定理 1.1 的割稀疏化器来处理多重割和多路割，我们首先将陈述一些关于 j-树的割保持性质（以多重割的形式表述）的关键主张和引理。

引理 D.1. 设 G、H 是具有相同顶点集的两个图，并假设 $H\preceq_\alpha G$。那么，对于任意划分 $\{S_i\}$，我们有 $U_H(\{S_i\})\le\alpha\cdot U_G(\{S_i\})$。

证明。 根据引理 3.6，对于所有 $i=1,2,\ldots,k$，我们有 $u_H(S_i)\le\alpha\cdot U_G(S_i)$。特别地，$U_H(\{S_i\})=\frac{1}{2}\sum_{i=1}^k U_H(S_i)\le\frac{1}{2}\sum_{i=1}^k\alpha\cdot U_G(S_i)=\alpha\cdot U_G(\{S_i\})$。 □

以下引理可视为引理 6.9 在多路割上的推广。

引理 D.2. 设 $H_1, \ldots, H_k$ 是定理 4.4 中的 $O(j)$-树，且满足 $G \preceq k^{-1}\sum_{i=1}^{k} H_i \preceq_\alpha G$。设 $\{S_i\}$ 是 V 的一个划分。那么，如果我们均匀随机抽取 $l = O(\log n)$ $O(j)$-树 $H_{i_1}, \ldots, H_{i_l}$，则有

$$U_G(\{S_i\}) \le \min_{1\le j\le l} U_{H_{i_j}}(\{S_i\}) \le 2\alpha \cdot U_G(\{S_i\})$$

以高概率成立。

证明。 根据前一个引理，我们有

$$U_G(\{S_i\}) \le k^{-1}\sum_{i=1}^{k} U_{H_i}(\{S_i\}) \le \alpha \cdot U_G(\{S_i\}).$$

因此，根据马尔可夫不等式，对于任意随机抽样得到的 $O(j)$-树 H_{i_j}，都有 $\Pr[U_{H_{i_j}}(\{S_i\}) \ge 2\alpha U_G(\{S_i\})] \le 1/2$。因此，均匀随机抽样 $l = O(\log(n))$ 棵 $O(j)$-树就足够了。 □

回顾一下，我们已经从宏观层面证明，由定理 1.1 维护的层次结构 $\mathcal{H}$ 将通过归纳论证近似保留底层图的割：根据定理 5.1，对于任意 $i \in [L]$，这些~稀疏化核心 $C(H_{i-1})$ 能近似保留其更稠密对应图 $C(H_{i-1})$ 中的割；而根据引理 6.9~，$O(j_{i+1})$ 棵 $\mathcal{H}(H_i) \subseteq \mathcal{H}_{i+1}$ 树的集合能近似保留其在 $C(H_i)$ 中更大对应图的割。可以验证，在整个归纳论证中将引理 6.9 替换为引理 D.2 后，还能证明 $\mathcal{H}$ 同样可以在一个 $\widetilde{O}(\log n)^L$ 因子范围内保留多路割的大小。

当收到包含 k 个终端点或终端点对的查询时，数据结构首先将所有 $O(k)$-终端点移入每棵 $O(j)$-树 $H \in \mathcal{H}$ 的核心中；当核心增长到超过 $\Theta(j)$ 的规模时，无需重建层次结构的最低层。实现方法是先在所有终端顶点与核心之间插入一条任意边，然后将其删除。根据定理 1.1，这需要 $O(k \cdot (n/j)^{2/L}\,\mathrm{poly}\left(\log^L n\right)\log U)$ 时间。

将终端点移入核心后，任何分离这些终端点或终端点对的最小规模割都不会切断森林中与核心相连的边。因此，我们可以只关注 $H \in \mathcal{H}$ 中每棵 $O(j)$-树的稀疏化核心。由于终端点被移入核心，每个核心都有 $O(k+j)$ 个顶点和 $O((j+k))$ 条边（回想一下，$O(j)$-树的核心是通过定理 5.1 进行稀疏化的）。查询得到解答后，我们将数据结构恢复到查询前的状态。

最后，我们再次重新插入所有 $O(k)$ 条边，然后从图中删除它们；这一次，正常处理这些边，并在核心规模超过 $\Theta(j)$ 时重建核心。这个看似多余的步骤是一种分析技巧，使我们能够摊销将终端顶点推入核心所需的 $O(k)$ 次边更新的成本。回想一下，定理 1.1 仅表明边更新的摊销更新时间有上界 $O(k \cdot (n/j)^{2/L}\,\mathrm{poly}\left(\log^L n\right)\log U)$。因此，将终端顶点拉入核心所需的 $O(k)$ 次边更新可能代价高昂。

如果我们只是反转这些更新的影响，就可能出现它们总是代价高昂的情况。不过请注意，第二组更新是正常的边更新，不会被撤销，因此其成本可摊销为 $O(k \cdot (n/j)^{2/L}\,\mathrm{poly}\left(\log^L n\right)\log U)$。此外，第一组更新的成本由第二组更新的成本上界限制。因此，我们可以在 $O(k \cdot (n/j)^{2/L}\,\mathrm{poly}\left(\log^L n\right)\log U)$ 更新时间内将终端点移入核心。

在恢复第一组更新之前，我们还需要说明如何在核心中以 $\mathrm{polylog}(n)$ 因子近似多路割和多重割的值。等价地，我们需要说明：对于一个具有 n 个顶点和 $O(n)$ 条边的~图 $G = (V,\ E,\ u)$，针对大小为 $k \le n$ 的终端或终端点对集合，如何在 $O(\mathrm{polylog}(n))$ 因子范围内近似最小多路割和多重割的大小。文献中已经存在多种用于这些目的的算法，其运行时间为 $\mathrm{poly}(n)$。为了优化我们定理对 k 的依赖关系，我们将介绍一些使用标准技术的简单方法；我们不声称这些方法具有创新性，它们对于多路割和多重割的运行时间分别为 $O(n)$ 和 $O(n^2)$。

~引理 D.3. 设 $G = (V, E, u)$ 是一个有 n 个顶点、$O(n)$ 条边的图，且 $T \subseteq V$ 是一组终端点。存在一种算法，可在 $\sim O(n)$ 时间内返回相对于 T 的 G 的 $O(\log n)$-近似多路割的大小。

证明。 将终端点 T 划分为两个集合，重复 $t = O(\log n)$ 次，使得对于 T 中任意一对顶点，至少在一个划分中，它们会落入不同的集合。例如，通过随机采样，以高概率即可实现这一点。令划分集合为 $T_{i_1}, T_{i_2} : i \in [t]$。

针对每一种划分，构造一个图 G_i，每个图都是在 G 的基础上扩展而来，并添加一对汇点和源点顶点s，t，每个顶点都通过一条容量极高的边连接到划分的一侧。在每个 G_i 中，计算一个 α 近似 $s-t$ 最小割，耗时为某个 $\alpha = 1$ 在 $O(n)$ 时间内（例如可以通过[43]实现）。算法返回这些割的并集的大小。

注意，这些割的并集是 G 关于 T 的一个多路割，因为任意一对顶点都必须被某个 $s-t$ 割分开。此外，对于任意 G_i， G 关于 T 的最小规模多路割会将 T_{i_1} 和 T_{i_2} 分开，因此，计算出的所有 $s-t$ 割的代价至多为 G 的最小规模多路割的 α 倍。因此，这些 $s-t$ 割的并集相对于 G 关于 T 的最小代价多路割，是 $O(\log n)$ 近似的。

引理 D.4. 设 $G = (V, E, u)$ 是一个具有 n 个顶点和 $O(n)$ 条边的图，且 $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ 是 V 中至多包含 n 个终端点对的集合。存在一种算法，可在 $\widetilde{O}(n^2)$ 时间内，返回相对于 T 的 $\widetilde{O}(\log^2 n)$-近似多重割的大小。

证明。 利用定理 4.9，我们可以在 $\widetilde{O}(n^2)$ 时间内得到一个由 n 棵树组成的集合（ $j = 1$ 的 j-树）。根据引理 D.1和引理 D.2，这些树中抽取的一个大小为 $O(n)$ 的样本，能够以 $O(\log n)$ 的因子保留相对于 T 的 G 的多割值。因此，只需在 $\widetilde{O}(n^2)$ 时间内，求出每棵抽样树上的多割值的 $O(\log n)$-近似。

我们可以通过以下方式将多重割问题转化为加权集合覆盖问题的一个实例：定义集合与边相对应，并将边的权重赋给相应集合；定义元素与终端点对相对应；如果某条边切断了相应的终端点对，就称相应的集合包含该元素。我们可以通过枚举树中任意终端点对之间的唯一简单路径，在 $O(n^2)$时间内初始化加权集合覆盖问题。

最后请注意，贪心算法求解加权集合覆盖问题可获得 $O(\log n)$-近似，这是一个众所周知的事实；其运行时间为 $O(n^2)$（集合数乘以元素数）。

E 定理 7.10 的证明 本节将给出定理 7.10 的证明，即说明如何使用定理 1.1 中的动态 $O(j)$-树数据结构，动态维护对近似最稀疏割值的查询访问。

我们的证明分为两部分。首先，我们陈述并证明一个简单的结构引理：对于任意 j-树 H，存在一个最稀疏割，它要么只切断端点位于核心 $C(H)$ 中的边，要么只切断一条森林边 $e \in F(H)$。请记住，对于 j-树 $H = C(H) \cup F(H)$，图 $F = F(H)$ 是一片有根森林，而核心 $C = C(H)$ 是根集 R of F 上的诱导子图（特别地， $V(C) = R$）。

接下来，我们将说明如何高效地维护树边中的最稀疏割，即 $\min_{e \in E_F} \psi(S(e))$。为此，我们会对定理 4.4 中的数据结构稍作修改，并采用来自 [19] 的思路，确保森林 F_i 中的路径足够短。

结合这两部分，我们便可证明，在收到查询时，只需运行(行文)最稀疏割算法，处理所有 $\widetilde{H} \in \mathcal{H}$对应的（稀疏化）核心 $C(H)$。由于这些核心 $C(H)$仅包含 $\widetilde{O}(j)$条边，最稀疏割算法只需 $\widetilde{O}(j^2)$时间。

结构引理 设 $H \in \mathcal{H}$ 是一个带有森林 F 和（稀疏化）核心 C的 $O(j)$-树。对于一条边$e = (u, v) \in E_F$，若 u 是 v的子节点，则定义 $S(e) := u^{\downarrow} \subseteq V$为由考虑 u 的后代集合（其中包括 u）所诱导的割。记住，每个 $v \in V(C)$都是 F中唯一一棵树的根，我们也可以为集合 $S \subseteq V(C)$定义 $S^{\downarrow} = \bigcup_{v \in S} v^{\downarrow}$。主要引理表明，要在一棵$j$-树上找到最稀疏割，只需分别考虑核心和森林内部的割。也就是说，它证明总存在一个最稀疏割 S，它只割核心中的边或者 森林中的边，但不会同时割两者。

我们先证明一个简单的辅助引理。

引理 E.1. 引理 那么，对于图 H，总存在一个最稀疏割 S，使得 $H[S]$ 是连通的 .

证明。设 S 是一个 spa 与 $|S| \leq n/2$ 的最小割，并假设 $H[S]$ 由连通分量组成 ts

$S_1, \ldots, S_k$。那么

$$\psi_G(S) = \frac{U_H(S_1) + \ldots + U_H(S_k)}{|S_1| + \ldots + |S_k|} \geq \min_{1 \leq j \leq k} \frac{U_H(S_j)}{|S_j|} = \min_{1 \leq j \leq k} \psi_H(S_j),$$

因此，其中一个连通分量也必然构成一个最稀疏割。 □

现在我们可以陈述并证明主要的结构引理。

引理 E.2. 设 H 是一棵具有核心 $C(H)$ 和森林 $F(H)$ 的 j-树。则存在一个最稀疏割 S，使得以下两者之一成立：

1. $S = S(e)$ 对于某条森林边 $e \in E_F$ 成立；或者，
2. $S = X^{\downarrow}$ 对于某个集合 $X \subseteq V(C(H))$ 成立。

证明。假设不存在一条森林边 $e \in E_F$ 使得 $\psi(S(e)) = \psi_H(S)$，即假设 $U_H(S(e))/\min\{|S(e)|, n-|S(e)|\} > U_H(S)/\min\{|S|, n-|S|\}$，否则便没有什么需要证明的。此外，我们可以假设 $|S| \leq n/2$，并且根据前一个引理，假设 $H[S]$ 是连通的。

我们的目标现在是证明，对于某个集合 $X \subseteq V(C(H))$，$S = X^{\downarrow}$。这等价于证明 $E(S, V \setminus S)$ 不包含任何森林边 $e \in E_F$。我们将通过矛盾证明这一结论。假设存在一条森林边 $e \in E_F$，使得 $e \in E(S, V \setminus S)$。接下来，我们将证明割 $S' = S \cup S(e)$ 满足 $\psi_H(S') < \psi_H(S)$，由此得到矛盾。

注意，根据 $H[S]$ 连通这一假设，我们知道 $S \cap S(e) = \emptyset$。特别地，这意味着 $U_H(S') = U_H(S) - U_H(S(e))$。我们首先注意到

$$\begin{aligned}
&\psi_H(S') = \frac{U_H(S) - U_H(S(e))}{\min\{|S'|, n - |S'|\}} = \frac{U_H(S) - U_H(S(e))}{\min\{|S| + |S(e)|, n - |S| - |S(e)|\}} < \frac{U_H(S)}{|S|} = \psi_H(S) \\
\Leftrightarrow \quad &(U_H(S) - U_H(S(e)))\,|S| < U_H(S) \min\{|S| + |S(e)|, n - |S| - |S(e)|\} \\
\Leftrightarrow \quad &U_H(S)\,(|S| - \min\{|S| + |S(e)|, n - |S| - |S(e)|\}) < U_H(S(e))|S|.
\end{aligned}$$

利用假设 $\psi_H(S) < \psi_H(S(e))$，因此只需证明

$$U_H(S)\,(|S| - \min\{|S| + |S(e)|, n - |S| - |S(e)|\}) \leq U_H(S) \min\{|S(e)|, n - |S(e)|\},$$

也就是说，

$$|S| - \min\{|S| + |S(e)|, n - |S| - |S(e)|\} \leq \min\{|S(e)|, n - |S(e)|\}.$$

首先考虑 $\min\{|S| + |S(e)|, n - |S| - |S(e)|\} = |S| + |S(e)|$ 的情形。此时命题显然成立，因为 $|S| - (|S| + |S(e)|) = -|S(e)| \leq \min\{|S(e)|, n - |S(e)|\}$。现在考虑 $\min\{|S| + |S(e)|, n - |S| - |S(e)|\} = n - |S| - |S(e)|$ 且 $\min\{|S(e)|, n - |S(e)|\} = |S(e)|$ 的情形。此时我们需要证明 $2|S| - n + |S(e)| \leq |S(e)|$，也就是说，证明 $2|S| \leq n$；这成立，因为 $|S| \leq n/2$。最后，考虑 $\min\{|S| + |S(e)|, n - |S| - |S(e)|\} = n - |S| - |S(e)|$ 且 $\min\{|S(e)|, n - |S(e)|\} = n - |S(e)|$ 的情形。此时我们需要证明 $2|S| + |S(e)| - n \leq n - |S(e)|$，也就是说，证明 $2|S| + 2|S(e)| \leq 2n$。这成立，因为 $2|S| + 2|S(e)| = 2(|S| + |S(e)|) = 2|S \cup S(e)| \leq 2n$。 □

维护森林中的最稀疏割 现在我们要证明引理 6.10 的一个推广。在此之前，我们先推广最稀疏割的概念。假设图 H 还配有顶点权重 $w \in \mathbb{R}^n_{>0}$，并对一个子集 $S \subseteq V$ 定义 $w(S) = \sum_{v \in S} w(v)$。那么，割 S的稀疏度定义为 $\psi_H(S) = U_G(S)/\min\{w(S), w(V) - w(S)\}$，而最稀疏割值定义为 $\psi(H) := \min_{S \subseteq V} \psi_H(S)$。注意，令 $w = \mathbf{1}$即可恢复先前的定义。

引理 E.3. 设 $H_1, \ldots, H_k$ 为引理 6.10维护的 $k = O(\log n)$ 个 $O(j)$-树。此外，假设算法运行所依托的动态图 G配有顶点权重$w \in \mathbb{R}^n_{>0}$。

假设每当 引理 6.10 所运行于其上的底层图 G发生边插入或删除时，此外 $O(1)$ 权重函数 w 的条目会以任意方式更新。

然后，对于每个 H_i，我们都可以动态维护其森林边中的最小（加权）稀疏度，~也就是说，我们维护值 $\min_{e \in E(F(H_i))} U_{H_i}(S(e))/\min\{w(S), w(V) - w(S)\}$。每次更新的运行时间为 $O(n/j)$。特别是，引理 6.10的渐近运行时间不会改变。

证明。 我们采用了 [19] 中的路径长度缩减思路， F_i通过确保在初始化森林之后，~森林中的所有路径长度都至多为 $O(n/j)$。由于森林是递减式的，路径长度只会随时间推移而缩短。假设路径长度如此之短，首先注意到，定理 4.9 保证底层图在收到更新后，森林 F_i 至多发生 $O(1)$ 次边删除。如果森林 F_i 因删除边 $e = (u, v)$ 而发生变化，其中 v 是 u的父节点，那么森林中稀疏度可能发生变化的边，只有位于 v-根$_{F_i}(v)$路径上的边。通过在动态森林F_i上运行引理 3.11 中的数据结构，我们还可以在对 F_i 和 w进行更新时，以 $O(\log n)$ 的更新时间维护对 $w(u^{\downarrow})$ 的查询访问，即根为顶点 u的子树中各顶点权重之和。因此，我们可以在 $O(\log n)$ 时间内，重新计算 v-根$_{F_i}(v)$路径上每条边的稀疏度。由于该路径长度至多为 $O(n/j)$，这表明我们可以在 $O(n/j)$ 时间内维护 F 中每条边的稀疏度。

为完整起见，我们现在也说明如何初始化 F_i ，使其路径长度满足给定界限。为此，请回想定理 4.9 的证明：我们最初指定了一个大小为 $|R_i| = O(j)$ 的分支根集合 R_i ，并通过切断初始生成树 T_i 中顶点 u, v 之间的树边 $e^{T_i}_{\min}(u, v)$，由此得到森林 F_i 。下文将证明，我们可以找到一个大小为 $|S| = O(j)$ 的集合 S ，使得在设定 $R'_i = R_i \cup S$ 并利用集合 R'_i 作为定理 4.4 中的初始集合来使 R'_i 无分支后，对应的森林 F_i 中所有路径的长度都至多为 $O(n/j)$。

为找到集合 S，我们按如下方式进行。如果存在一条从叶节点 v 到其根 r、长度为 n/j 的路径，我们就在该路径上找到一条边 $e = (x, y)$，使得从 v 到 x 的路径长度至少为 $n/(2j)$。然后，我们将 x, y 的两个端点都加入集合 R'。由此可知，定理 4.4 会通过移除边 e 来更新森林 F_i 。随后我们重复这一过程，直到不再存在这样的路径。每次执行这一过程， F_i 的一个连通分量 T 的大小至少减少 $n/(2j)$。对于大小小于 $n/(2j)$ 的连通分量，所有路径的长度都至多为 n/j；因此，迭代总次数为 $O(j)$，如所需。每次迭代可在 $O(n/j)$ 时间内完成，因此找到该集合所需的总时间为 $O(n)$。 □

~综合起来 设 H 是集合 $\mathcal{H}$中的一棵 $O(j)$-树，其中 C_L 是其核心，且 $F = F_L \cup \ldots \cup F_1$ 对应的森林。设 $w \in \mathbb{R}^{V_{(C_L)>0}}$是满足 $w(v) = |v^{\downarrow}|$的顶点权重，即， F 中以 v为根的连通分量 T 的大小。那么，引理 E.2 告诉我们，可以通过以下方式在 H 中找到一个最稀疏割：

1. 计算 移除单条森林边所诱导的割的稀疏度，即计算 $\min_{e \in E_F} \psi_H(S(e))$.

2. 计算仅割断端点位于核心中的边的最稀疏割的值。这等价于计算带有顶点权重 $w(v) = |v^{\downarrow}|$的图 $\widetilde{C}_L$ 中的最稀疏割的值。

3. 最后，返回这两类割的最小值。

因此，我们的算法针对每个 $H \in \mathcal{H}$ 执行以下操作：(1) 维护使 $\psi_H(S(e))$ 最小的边 $e \in E_F$ ；(2) 查询时采用标准的非均匀最稀疏割算法。我们首先说明所使用的静态算法的保证；该算法是近似广义最稀疏割算法的一个特例。

~引理 E.4. 存在一种算法，对于带有顶点权重 w 的图 G ，能够在 $\widetilde{O}(m^2)$ 时间内，返回顶点加权最稀疏割的一个 $O(\log n)$近似值。

证明。 利用定理 4.9，我们可以在 $\widetilde{O}(m^2)$ 时间内得到包含 $\widetilde{O}(m)$ 棵树的集合（针对 $j = 1$ 的 j-树）。根据引理 5.10，从这些树中均匀随机抽样 $O(\log n)$ 棵，可以保证以概率 $1 - 1/n^c$,~ 保留顶点加权最稀疏割，误差至多为 $O(\log n)$ 倍：分别对每棵树计算该割，然后返回找到的最小值。然而，在一棵树上，只需分别检查每条边，即可在 $O(n)$ 时间内轻松完成这一操作。 □

定理 7.10的证明。 定理 1.1 中的数据结构保证，在对 G进行任意更新后，对于任意割 S，对于所有 $H \in \mathcal{H}$，都有 $U_G(S) \leq U_H(S)$。此外，以高概率存在一个$H \in \mathcal{H}$，使得 $U_H(S) \leq \widetilde{O}(\log\ n)^L U_G(S)$。特别地，以高概率存在一个 $H \in \mathcal{H}$，使得 S中的最稀疏割满足 $\psi_H(S) \leq \widetilde{O}(\log\ n)^L \psi_G(S)$。我们可以对针对 G执行的所有更新的失败概率应用并集界，因此可以假设在任何时刻都存在一个 H，使得 $\psi(H) \leq \widetilde{O}(\log n)^L \psi(G)$。~因此，如果我们能够对所有 $H \in \mathcal{H}$计算最稀疏割的 $O(\log n)$-近似，那么返回所找到的最小值将得到所需的近似。

$\widetilde{\ }$给定一个具有 $H = C_L \cup F$和 $F = F_L \cup \ldots \cup F_1$的图，根据引理 E.2，我们可以通过以下方式实现这一目标：(1) 维护$\widetilde{\ }$ $\min_{e \in E_F} \psi_H(S(e))$；以及 (2) 在查询时，计算所有$S$ 仅割核心内部边的割的最小稀疏度的 $O(\log\ n)$ 近似。我们先讨论前者。

F中的最稀疏割：为了计算 $\min_{e \in E_F} \psi_H(S(e))$，我们针对所有森林 F_i 计算 $\min_{e \in E_{F_i}} \psi_H(S(e))$，然后返回所有 $i = 1$、.....、 L 中的最小值。不过请记住，根据定理 4.4 维护的 F_i 是定义在顶点集 $V(C_{i-1})$ 上的 $\widetilde{\ }$ 森林，而不是定义在完整顶点集 $V(H)$ 上。这意味着，如果我们想用引理 E.3 维护稀疏度，就需要引入顶点权重 $w(v) = |v^{\downarrow}|$，其值为完整森林 F 中以 v 为根的子树的大小。不过，引理 E.3 中的数据结构恰好能让我们做到这一点。因此，我们可以始终维护 F 中最稀疏割的值。

$O(j)$ 条边，因此，我们可以使用引理 E.4 的算法，给核心内部边的割的最小稀疏度的 $\widetilde{O}(j^2)$ 时间内计算所有仅切割核心内部边的割的最小稀疏度的 $\widetilde{O}(\log n)$-近似值 S 。

现在，只需注意到 $|\mathcal{H}| = O(\log^L n)$，即可完成证明，因此每次查询所耗费的总时间为 $O(j^2 \log^{O(L)} n)$。

参考文献

[1]A. Abboud、R. Krauthgamer 和 O. Trabelsi，《$APMF < $ 近二次时间内无权图的 *apsp*？*Gomory-Hu* 树》，载于第 62 届 IEEE 计算机科学基础年会（FOCS），IEEE，2021年，第 1135–1146 页，https://doi.org/10.1109/FOCS52979.2021.00112，https://doi.org/10. 1109/FOCS52979.2021.00112。

[2] A. Abboud、R. Krauthgamer 和 O. Trabelsi，无向图中所有点对最大流的新算法与下界，Adv. Math. Commun.，17（2021年），第 1–27 页，https://doi.org/10.4086/TOC.2021. V017A005，https://doi.org/10.4086/toc.2021.v017a005。

[3] A. Abboud、R. Krauthgamer 和 O. Trabelsi，无权图中 *Gomory-Hu* 树的次立方算法，载于《第53届 ACM 计算理论年会》（STOC），S. Khuller 和 V. V. Williams 编，ACM，2021年，第 1725–1737 页，https://doi.org/10.1145/3406325.3451073，https://doi.org/10. 1145/3406325.3451073。

[4] A. Abboud, J. Li, D. Panigrahi 和 T. Saranurak，全点对最大流并不比单点对最大流更难：近线性时间构造 *Gomory-Hu* 树，载于第64届 IEEE 计算机科学基础年会（FOCS），IEEE，2023年，第 2204–2212 页，https://doi.org/10.1109/FOCS57990.2023.00137，https://doi.org/10.1109/FOCS57990.2023.00137。

[5] I. Abraham、Y. Bartal 和 O. Neiman，近乎紧致的低伸展生成树，载于第49届IEEE计算机科学基础年会（FOCS），IEEE 计算机学会，2008年，第 781–790 页，https://doi.org/10.1109/FOCS.2008.62。

[6] I. Abraham、D. Durfee、I. Koutis、S. Krinninger 和 R. Peng，关于全动态图稀疏化器，载于第57届 IEEE 计算机科学基础年会（FOCS），2016年，第 335–344 页，https://doi.org/10.1109/FOCS.2016.44，https://doi.org/10.1109/FOCS.2016.44。

[7] I. Abraham 和 O. Neiman，利用花瓣分解构建低伸展生成树，SIAMJ. Com- put.，48（2019年），第 227–248 页，https://doi.org/10.1137/17M1115575，https://doi.org/10.1137/17M1115575。

[8] N. Alon、R. M. Karp、D. Peleg 和 D. B. West，一种图论博弈及其在 *k* 服务器问题中的应用，SIAM J. Comput.，24（1995年），第 78–100 页，https://doi.org/10.1137/S0097539792224474，https://doi.org/10.1137/S0097539792224474。

[9] N. Alon 和 V. D. Milman，λ_1图的等周不等式与超集中器，《组合理论杂志 B 辑》，38（1985年），第 73–88 页，https://doi.org/10.1016/0095-8956(85)90092-9，https://doi.org/10.1016/ 0095-8956(85)90092-9。

[10]R. Andersen 和 Y. Peres，利用演化集合在局部寻找稀疏割，载于第41届 ACM计算理论研讨会（STOC），M. Mitzenmacher 编，ACM，2009年，第 235–244 页，https: //doi.org/10.1145/1536414.1536449, https://doi.org/10.1145/1536414.1536449。[11] S. Arora、E. Hazan 和 S. Kale，在 *õ(n^2)* 时间内对最稀疏割进行 *O(sqrt(log(n))* 近似，《SIAM 计算杂志》，39（2010年），第 1748–1771 页，https://doi.org/10.1137/080731049，https://doi.org/10.1137/ 080731049。[12] S. Arora、E. Hazan 和 S. Kale，乘法权重更新方法：一种元算法及其应用，《计算理论》，8（2012年），第 121–164 页，https://theoryofcomputing.org/articles/v008a006。[13] S. Arora 和 S. Kale，半定规划的组合式原始-对偶方法，《美国计算机学会学报》，63（2016年），第 12:1–12:35 页，https://doi.org/10.1145/2837020, https://doi.org/10.1145/2837020。[14] S. Arora、S. Rao 和 U. V. Vazirani，扩展流、几何嵌入与图划分，《美国计算机学会学报》，56（2009年），第 5:1–5:37 页，https://doi.org/10.1145/1502793.1502794, https://doi.org/10.1145/1502793. 1502794。[15] A. A. Benczúr 和 D. R. Karger，带容量图中割与流的随机化近似方案，《SIAM 计算杂志》，44（2015年），第 290–319 页，https://doi.org/10.1137/070705970, https: //doi.org/10.1137/070705970，https://arxiv.org/abs/https://doi.org/10.1137/070705970。[16]A. Bernstein、J. van den Brand、M. Probst Gutenberg、D. Nanongkai、T. Saranurak、A. Sidford 和 H. Sun，针对自适应对手的全动态图库稀疏化器，载于第49届国际自动机、语言与程序设计学术讨论会（ICALP），M. Bojańczyk、E. Merelli 和 D. P. Woodruff 编，Leibniz 国际信息学论文集（LIPIcs）第 229 卷，德国达格施图尔，2022年，Schloss Dagstuhl – Leibniz 信息学中心，第 20:1–20:20 页，https://doi.org/10.4230/LIPIcs.ICALP.2022.20，https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2022.20。[17] M. Bienkowski、M. Korzeniowski 和 H. Räcke，构造无知路由方案的实用算法，载于第15届 ACM 算法与体系结构并行性研讨会（SPAA），A. L. Rosenberg 和 F. M. auf der Heide 编，ACM，2003年，第 24–33 页，https://doi.org/10.1145/777412.777418，https://doi.org/10.1145/777412.777418。[18] S. Chechik 和 T. Zhang，亚多项式时间内的动态低伸展生成树，载于 ACM-SIAM 离散算法研讨会（SODA），S. Chawla 编，SIAM，2020年，第 463–475 页，https://doi.org/10. 1137/1.9781611975994.28, https://doi.org/10.1137/1.9781611975994.28。[19] L. Chen、G. Goranci、M. Henzinger、R. Peng 和 T. Saranurak，借助顶点稀疏化器快速处理动态图中的割、距离与有效电阻，载于第61届 IEEE计算机科学基础年会（FOCS），2020年，第 1135–1146 页，https://doi.org/10.1109/FOCS46700.2020.00109, https://doi.org/10. 1109/FOCS46700.2020.00109。[20]L. Chen、R. Kyng、Y. Liu、R. Peng、M. Probst Gutenberg 和 S. Sachdeva，近线性时间内的最大流与最小费用流，《美国计算机学会学报》，72（2025年），第 1–103 页，https: //doi.org/10.1145/3728631，http://dx.doi.org/10.1145/3728631。[21]L. Chen、R. Kyng、Y. P. Liu、S. Meierhans 和 M. Probst Gutenberg，增量图的近线性时间算法：环检测、强连通分量、 s-t 最短路径与最小费用流，载于第56届 ACM计算理论研讨会（STOC），2024年，第 1165–1173 页，https://doi.org/10.1145/ 3618260.3649745。[22]L. Chen、R. Kyng、Y. P. Liu、R. Peng、M. P. Gutenberg 和 S. Sachdeva，用于最大流与最小费用流的近线性时间算法，《ACM 通讯》，66（2023年），第 85–92 页，https: //doi.org/10.1145/3610940，https://doi.org/10.1145/3610940。[23] S. I. Daitch 和 D. A. Spielman，通过内点算法更快地近似求解有损广义流，载于第40届 ACM计算理论研讨会（STOC），2008年，第 451–460 页，https://doi.org/10. 1145/1374376.1374441，https://doi.org/10.1145/1374376.1374441。

[24] S. Dong、Y. Gao、G. Goranci、Y. T. Lee、R. Peng、S. Sachdeva 和 G. Ye，嵌套剖分与内点法相遇：近线性时间求解平面最小费用流，载于 ACM-SIAM 离散算法研讨会（SODA），J. S. Naor 和 N. Buchbinder 编，SIAM，2022年，第 124–153 页，https://doi.org/10.1137/1.9781611977073.7，https://doi.org/10.1137/1.9781611977073.7。[25] A. El-Hayek、M. Henzinger 和 J. Li，每次操作亚多项式时间的全动态近似最小割，载于 ACM-SIAM 离散算法研讨会（SODA），Y. Azar 和 D. Panigrahi 编，SIAM，2025年，第 750–784 页，https://doi.org/10.1137/1.9781611978322.22，https://doi.org/10.1137/1.9781611978322.22。[26] S. Forster 和 G. Goranci，通过动态低直径分解构造动态低伸展树，载于第51届 ACM 计算理论年会（STOC），M. Charikar 和 E. Cohen 编，ACM，2019年，第 377–388 页，https://doi.org/10.1145/3313276.3316381，https://doi.org/10.1145/3313276.3316381。[27]W. S. Fung、R. Hariharan、N. J. Harvey 和 D. Panigrahi，图稀疏化的一般框架，载于第43届 ACM 计算理论年会（STOC），STOC '11，纽约，NY，美国，2011年，计算机协会，第 71–80 页，https://doi.org/10.1145/1993636.1993647，https://doi.org/ 10.1145/777412.777419。[28] R. E. Gomory 和 T. C. Hu，多端网络流，《工业与应用数学学会杂志》，9（1961年），第 551–570 页，https://doi.org/https://doi.org/10.1137/0109047。[29]G. Goranci 和 M. Henzinger，用于增量式精确与近似最大流的高效数据结构，载于第50届自动机、语言与编程国际研讨会（ICALP），K. Etessami、U. Feige 和 G. Puppis 编，LIPIcs 第 261 卷，达格施图尔城堡—莱布尼茨信息中心，2023年，第 69:1–69:14 页，https://doi.org/10.4230/LIPICS.ICALP.2023.69，https://doi.org/10.4230/ LIPIcs.ICALP.2023.69。[30]G. Goranci、M. Henzinger、D. Nanongkai、T. Saranurak、M. Thorup 和 C. Wulff-Nilsen，亚线性时间的全动态精确边连通性，载于 ACM-SIAM 离散算法研讨会（SODA），N. Bansal 和 V. Nagarajan 编，SIAM，2023年，第 70–86 页，https://doi.org/10.1137/1. 9781611977554.CH3，https://doi.org/10.1137/1.9781611977554.ch3。[31] G. Goranci、M. Henzinger、H. Räcke 和 A. R. Sricharan，通过残量图稀疏化实现增量式近似最大流，载于第52届自动机、语言与编程国际研讨会（ICALP），K. Censor-Hillel、F. Grandoni、J. Ouaknine 和 G. Puppis 编，LIPIcs 第 334 卷，达格施图尔城堡—莱布尼茨信息中心，2025年，第 91:1–91:20 页，https://doi.org/10.4230/LIPICS. ICALP.2025.91，https://doi.org/10.4230/LIPIcs.ICALP.2025.91。[32] G. Goranci、M. Henzinger 和 M. Thorup，多对数摊销更新时间的增量式精确最小割，载于第24届欧洲算法研讨会（ESA），P. Sankowski 和 C. D. Zaroliagis 编，LIPIcs 第 57 卷，达格施图尔城堡—莱布尼茨信息中心，2016年，第 46:1–46:17 页，https: // doi.org/10.4230/LIPICS.ESA.2016.46，https://doi.org/10.4230/LIPICS.ESA.2016.46。[33] G. Goranci、H. Räcke、T. Saranurak 和 Z. Tan，扩张器层次结构及其在动态图算法中的应用，载于第32届 ACM-SIAM 离散算法研讨会（SODA），2021年，第 2212–2228 页，https://doi.org/10.1137/1.9781611976465.132，https://doi.org/10.1137/1.9781611976465.132。[34]M. Gupta 和 S. Khan，用于极大独立集、最大流和最大匹配的简单动态算法，载于第4届 SIAM 算法简洁性研讨会（SOSA），H. V. Le 和 V. King 编，SIAM，2021年，第 86–91 页，https://doi.org/10.1137/1.9781611976496.10，https://doi.org/ 1.9781611976496.10。[35] C. Harrelson、K. Hildrum 和 S. Rao，用于最小化拥塞的多项式时间树分解，载于第15届 ACM 算法与体系结构并行性研讨会（SPAA），A. L. Rosenberg 和 F. M. auf der Heide 编，ACM，2003年，第 34–43 页，https://doi.org/10.1145/777412.777419，https://doi.org/ 10.1145/777412.777419。

[36] M. R. Henzinger 和 V. King，每次操作多对数时间的随机全动态图算法，《美国计算机学会学报》，46（1999），第 502–516 页，https://doi.org/10.1145/320211.320215，https://doi.org/10.1145/320211.320215。[37] J. Holm、K. de Lichtenberg 和 M. Thorup，用于连通性、最小生成树、*2*-边连通性和双连通性的多对数时间确定性全动态算法，《美国计算机学会学报》，48（2001），第 723–760 页，https://doi.org/10.1145/502090.502095，https://doi.org/10.1145/502090.502095。[38] D. R. Karger，近线性时间求解最小割，《美国计算机学会学报》，47（2000），第 46–76 页，https://doi.org/10.1145/331605.331608，https://doi.org/10.1145/331605.331608。[39]J. A. Kelner、Y. T. Lee、L. Orecchia 和 A. Sidford，用于无向图近似最大流的近线性时间算法及其多商品推广，载于第25届 ACM-SIAM 离散算法研讨会（SODA），C. Chekuri 编，SIAM，2014年，第 217–226 页，https://doi.org/10. 1137/1.9781611973402.16，https://doi.org/10.1137/1.9781611973402.16。[40]R. Khandekar、S. Rao 和 U. V. Vazirani，利用单商品流进行图划分，《美国计算机学会学报》，56（2009），第 19:1–19:15 页，https://doi.org/10.1145/1538902.1538903，https://doi.org/10.1145/ 1538902.1538903。[41] R. Kyng、S. Meierhans 和 M. P. Gutenberg，动态最短路径工具箱：低拥塞顶点稀疏化器及其应用，载于第56届 ACM 计算理论年会（STOC），B. Mohar、I. Shinkar 和 R. O'Donnell 编，ACM，2024年，第 1174–1183 页，https://doi.org/10.1145/3618260. 3649767，https://doi.org/10.1145/3618260.3649767。[42]F. T. Leighton 和 S. Rao，多商品最大流最小割定理及其在设计近似算法中的应用，《美国计算机学会学报》，46 （ 1999 ），第 787–832 页，https://doi.org/10.1145/331524.331526，https: //doi.org/10.1145/331524.331526。[43]J. Li 、D. Nanongkai、D. Panigrahi 和 T. Saranurak，通过公平割实现割问题的近线性时间近似，载于 ACM-SIAM 离散算法研讨会（SODA），N. Bansal 和 V. Nagarajan 编，SIAM，2023年，第 240–275 页，https://doi.org/10.1137/1.9781611977554.CH10，https://doi. org/10.1137/1.9781611977554.ch10。[44] J. Li、D. Panigrahi 和 T. Saranurak，简单图中近乎最优的全对最小割算法，载于第 62 届 IEEE 计算机科学基础年会（FOCS），IEEE，2021年，第 1124–1134 页，https://doi.org/10.1109/FOCS52979.2021.00111，https://doi.org/10.1109/FOCS52979.2021.00111。[45] J. Li、S. Rao 和 D. Wang，自底向上构造拥塞近似器，载于 ACM-SIAM 离散算法研讨会（SODA），Y. Azar 和 D. Panigrahi 编，SIAM，2025年，第 2111–2131 页，https: //doi.org/10.1137/1.9781611978322.68，https://doi.org/10.1137/1.9781611978322.68。[46] A. Madry，无向图中基于割问题的快速近似算法，载于第51届 IEEE 计算机科学基础年会（FOCS），2010年，第 245–254 页，https://doi.org/10.1109/FOCS. 2010.30，https://doi.org/10.1109/FOCS.2010.30。[47] A. Madry，利用增广电流计算最大流，载于第57届 IEEE 计算机科学基础年会（FOCS），I. Dinur 编，IEEE 计算机学会，2016年，第 593–602 页，https://doi.org/10.1109/FOCS.2016.70，https://doi.org/10.1109/FOCS.2016.70。[48] L. Orecchia、L. J. Schulman、U. V. Vazirani 和 N. K. Vishnoi，利用单商品流进行图划分，载于第40届 ACM 计算理论年会（STOC），C. Dwork 编，ACM，2008年，第 461–470 页，https://doi.org/10.1145/1374376.1374442，https://doi.org/10.1145/1374376.1374442。[49] R. Peng，在 O*(*m*polylog(*n*))* 时间内求解近似无向最大流，载于第27届 ACM-SIAM 离散算法研讨会（SODA），R. Krauthgamer 编，SIAM，2016年，第 1862–1867 页，https://doi.org/10.1137/ 1.9781611974331.CH130，https://doi.org/10.1137/1.9781611974331.ch130。

[50] M. Thorup，近似k-流割，载于第17届欧洲算法年会（ESA），第1–4页，https://doi.org/10.1007/978-3-642-04128-0_1，https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=71c9fd11ff75889aaa903b027af3a06e750e8add.[51] H. Räcke，在一般网络中最小化拥塞，载于第43届 IEEE 计算机科学基础年会（FOCS），2002年，第43–52页，https://doi.org/10.1109/SFCS.2002.1181881，https://doi.org/ 10.1109/SFCS.2002.1181881.[52] H. Räcke，用于最小化网络拥塞的最优分层分解，载于第40届 ACM 计算理论年会（STOC），2008年，第255–264页，https://doi.org/10.1145/1374376. 1374415，https://doi.org/10.1145/1374376.1374415.[53] H. Räcke、C. Shah 和 H. Täubig，近线性时间计算基于割的分层分解，载于第25届 ACM-SIAM 离散算法研讨会（SODA），C. Chekuri 编，SIAM，2014年，第227–238页，https://doi.org/10.1137/1.9781611973402.17，https://doi.org/10.1137/1.9781611973402.17.[54] J. Sherman，突破多商品流障碍，实现对最稀疏割的 *o(vlog n)* 近似，载于第50届 ACM 计算理论年会（STOC），IEEE 计算机学会，2009年，第363–372页，https://doi.org/10.1109/FOCS.2009.66，https://doi.org/10.1109/FOCS.2009.66.[55] J. Sherman，近线性时间求近似最大流，载于第54届 IEEE 计算机科学基础年会（FOCS），IEEE 计算机学会，2013年，第263–269页，https://doi.org/10.1109/FOCS. 2013.36，https://doi.org/10.1109/FOCS.2013.36.[56] J. Sherman，广义预处理与无向最小费用流，载于第28届 ACM-SIAM 离散算法研讨会（SODA），P. N. Klein 编，SIAM，2017年，第772–780页，https://doi.org/10.1137/ 1.9781611974782.49，https://doi.org/10.1137/1.9781611974782.49.[57]D. B. Shmoys，割问题及其在分治中的应用，PWS出版公司，1996年，第192–235页，https://people.orie.cornell.edu/shmoys/pdf/multicut.pdf.[58] D. D. Sleator 和 R. E. Tarjan，动态树的数据结构，《计算机与系统科学杂志》，26（1983），第362–391页，https://doi.org/10.1016/0022-0000(83)90006-5，https://doi.org/10.1016/0022-0000(83)90006-5.[59]J. van den Brand、L. Chen、R. Kyng、Y. P. Liu、S. Meierhans、M. P. Gutenberg 和 S. Sachdeva，递减图的近线性时间算法：通过对偶性求解最小费用流及更多问题，载于第65届 IEEE 计算机科学基础年会（FOCS），IEEE，2024年，第2010–2032页，https://doi.org/10.1109/FOCS61266.2024.00120，https://doi.org/10.1109/FOCS61266.2024.00120.[60] J. van den Brand、L. Chen、R. Kyng、Y. P. Liu、R. Peng、M. P. Gutenberg、S. Sachdeva 和 A. Sidford，次多项式更新时间下无向图的增量式近似最大流，载于 ACM-SIAM 离散算法研讨会（SODA），D. P. Woodruff 编，SIAM，2024年，第2980–2998页，https://doi.org/10.1137/1.9781611977912.106，https://doi.org/10.1137/1.9781611977912.106.[61] J. van den Brand、L. Chen、R. Peng、R. Kyng、Y. P. Liu、M. P. Gutenberg、S. Sachdeva 和 A. Sidford，一种确定性近线性时间最小费用流算法，载于第64届 IEEE 计算机科学基础年会（FOCS），IEEE，2023年，第503–514页，https://doi.org/10. 1109/FOCS57990.2023.00037，https://doi.org/10.1109/FOCS57990.2023.00037.[62] J. van den Brand、Y. T. Lee、Y. P. Liu、T. Saranurak、A. Sidford、Z. Song 和 D. Wang，稠密实例的近线性时间最小费用流、马尔可夫决策过程和 ℓ_1回归，载于第53届 ACM 计算理论年会（STOC），S. Khuller 和 V. V. Williams 编，ACM，2021年，第859–869页，https://doi.org/10.1145/3406325.3451108，https://doi.org/10.1145/3406325.3451108.[63] J. van den Brand、Y. P. Liu 和 A. Sidford，通过动态内点法实现动态最大流，载于第55届 ACM 计算理论年会（STOC），B. Saha 和 R. A. Servedio 编，ACM，2023年，第1215–1228页，https://doi.org/10.1145/3564246.3585135，https://doi.org/10.1145/3564246.3585135。

[64] N. Young，混合装箱与覆盖的串行与并行算法，载于第42届 IEEE 计算机科学基础研讨会论文集，2001年，第538–546页，https://doi.org/10.1109/SFCS.2001. 959930。

[65] T. Zhang，《简单图中更快的割等价树》，载于《第49届国际自动机、语言与程序设计学术讨论会（ICALP）》，M. Bojanczyk、E. Merelli 和 D. P. Woodruff 编，第229卷，LIPIcs，Schloss Dagstuhl - 莱布尼茨信息学中心，2022年，第109:1–109:18页，https://doi.org/10.4230/ LIPICS.ICALP.2022.109，https://doi.org/10.4230/LIPIcs.ICALP.2022.109。