arXiv:2603.10639v1 [cs.DS] 11 Mar 2026

可扩展 MPC 中密度相关的图定向与着色

MohsenGhaffari ChristophGrunau麻
省理工学院 苏黎世联邦理工学院
ghaffari@mit.edu cgrunau@inf.ethz.ch

摘要

本文提出了在强次线性内存区间（即，可扩展 MPC）中的大规模并行计算（MPC）算法，用于根据图的子图密度对图进行定向和着色。我们的算法在 poly(log log n) 轮次内运行，并计算出最大出度为 $O(\alpha \log\log n)$ 的边定向，以及使用$O(\alpha \log\log n)$ 种颜色的顶点着色。这里，α 表示最稠密子图的密度。我们算法的~轮复杂度之所以值得关注，是因为它突破了 $\Theta(\sqrt{\log n})$ 障碍；该障碍适用于此前已知最优的密度相关定向算法[Ghaffari 、 Lattanzi 和 Mitrovic ICML'19]，并且也是许多其他可扩展 MPC 算法的共同特点。

目录

1 引言 **1**
1.1 模型与基本定义 .11.2 可扩展 MPC 与分布式算法中的最先进成果 .21.3 我们的结果 .31.4 技术概述 .41.5 其他相关工作 .5

2 预备知识 **6**
2.1（部分）层赋值 .72.2 有效映射 .9

3 边定向算法 **9**
3.1 剪枝 .103.2 指数化 + 剪枝 .123.3 部分层赋值 .17

4 着色算法 **24**

1 引言

我们提出了首个轮次数为 poly(log log n) 的可扩展大规模并行计算（MPC）算法，用于低出度定向和着色，其复杂度取决于子图密度（或等价地，森林度）。我们首先回顾相关背景和最先进成果，然后陈述我们的结果。

1.1 模型和基本定义

大规模并行计算模型。我们采用大规模并行计算（MPC）模型 [KSV10a，GSZ11，BKS17，ANOY14], 它已成为 MapReduce 等大规模分布式数据处理框架背后事实上的标准理论抽象 [DG 08],Hadoop [Whi12], Spark [ZCF+10] 以及 Dryad [IBY+07]。下面给出简要定义；更多内容请参见 Im 等人最近出版的著作 [IKL+23]或 Ghaffari 的讲义 [Gha] 。

分布式系统由 M 台机器组成，数据（以任意方式）分布在这些机器上。对于图问题，这些数据就是输入图 $G = (V, E)$，我们通常也使用记号 $m = |E|$ 和 $n = |V|$。假设每台机器的本地内存容量为 S 字。一个字就是 $O(\log n)$ 位，例如可以描述单个顶点或边。我们通常将所有内存容量相加，记作 MS，并称其为系统的全局内存。显然，我们需要满足 $MS = \Omega(m + n)$。计算以同步轮次的方式进行。每一轮中，每台机器都可以对其持有的数据进行计算，然后向其他机器发送一些信息。唯一的通信约束是：每台机器在一轮中发送或接收的数据总量不得超过其内存容量 S。我们主要关注的指标是算法的轮复杂度。

当 S 可能比 n 小得多——具体而言，对于常数 $\delta \in (0,1)$， $S \leq n^{\delta}$ ——这种情形被称为强次线性内存范围，它也是算法设计中最具挑战性（同时也最令人向往）的领域。此范围内的算法通常称为可扩展 *MPC* 算法，本文将聚焦于此。

LOCAL 模型。 许多用于图问题的最先进（可扩展）MPC 算法，都是基于分布式计算中较为经典的 LOCAL 模型所采用的方法设计的 [Lin87,Pel00]。下面简要回顾这一模型：图 $G = (V, E)$ 是对计算机网络的一种抽象，每个节点对应一台计算机，并且具有一个 $O(\log n$ 位标识符。每一轮中，每台计算机/节点都可以向其在 G 中的每个邻居发送一条消息。开始时，每个节点只知道自己的邻居。计算结束时，每个节点都应知道输出中属于自己的部分，例如在着色中自己的颜色。同样，主要衡量指标是轮复杂度。

最密子图密度与森林度。图 G 的最大子图密度（通常简称子图密度）定义为 $\alpha(G) = \max_{S \subseteq V}\{\frac{|E(S)|}{|S|}\}$，其中 $E(S)$ 表示两端点都在 S 中的边集。一个密切相关的度量是图的森林度 $\lambda(G)$，定义为 $\lambda(G) = \max_{S \subseteq V}\{\lceil \frac{|E(S)|}{|S|-1} \rceil\}$，它也等价于将边 E 划分为森林所需的最少数量 [NW61, NW64, Tut61]。注意，$\alpha(G) \leq \lambda(G) \leq \alpha(G) + 1$。

低出度定向。 我们关注最大出度较小的边定向。在任何定向中，最大出度都至少为子图密度$\alpha(G)$。我们希望出度上界接近这一数值。由于 $\alpha(G)$和 $\lambda(G)$相差至多一个+1，我们的结果可以用其中任一者来表述；下文采用森林度 $\lambda = \lambda(G)$（此外， $\lambda = 1$ 的特殊情形恰好对应森林）。

1.2 可扩展 MPC 和分布式算法领域的最先进成果 s

LOCAL 算法。Barenboim 和 Elkin [BE08] 提出了一种简单的 LOCAL 模型算法，对于任意常数 $\varepsilon > 0$，它能在 $O(\log n)$ 轮次内求得一个出度至多为 $(2+\varepsilon)\lambda$ 的边定向：每一轮同时移除剩余图中所有度至多为 $(2+\varepsilon)\lambda$ 的节点，并将它们关联的边标记为出边（如果一条边的两个端点都被移除，则任意确定该边的方向，例如指向标识符较大的端点）。这一 $O(\log n)$ 轮的复杂度是最优的：即使在 $\lambda = 1$ 的情况下，求得任意一个出度为 $O(1)$ 的定向也需要 $\Omega(\log n)$ 轮次，这可由 Linial [Lin87] 的下界推出。此外，还有算法 [GS17,SV20]，能在 $\text{poly}(\log n)$ 轮次内计算 $((1+\varepsilon)\lambda)$-出度定向。这些 LOCAL 算法很容易在可扩展 MPC 中运行，尤其是那个简单的 $O(\log n)$ 轮算法。然而，鉴于 MPC 模型的强大能力，以及在由 MPC 建模的大规模计算环境中每一轮的高昂代价，我们必须设法获得轮复杂度显著更低的算法。

可扩展 MPC 算法。Ghaffari、Lattanzi 和 Mitrovic [GLM19 在第 4] 节介绍了一种~~可扩展~~ MPC 算法，能够在 $O(\sqrt{\log n})$[1] 轮内计算一个最大出度至多为 $(2+\varepsilon)\lambda$ 的定向。[2] 该算法基于对上述 LOCAL 算法的快速模拟，并采用了 Ghaffari 和 Uitto [GU19] 工作中的稀疏化思路。稍后我们将进一步说明~~——~~~~这种方法及其~~ $\Theta(\sqrt{\log n})$ 瓶颈。据我们所知，这个 $O(\sqrt{\log n})$ 上界是可扩展 MPC 定向问题最先进的轮复杂度；事实上，即使允许更大的最大出度上界，例如 $\lambda \cdot \text{poly}(\log n)$，也是如此。

上述结果有一个显著的例外：在 $\lambda = 1$这一特殊情形下——也就是说，当 G 只是一个森林时——Grunau 等人近期提出的一种可扩展 MPC 算法 [GLM+23]能够在 $O(\log \log n)$ 轮内计算出度至多为 2 的定向。他们利用这一结果（以及该定向的其他性质），在 $O(\log \log n)$ 轮内为森林提出了一种 3-着色。这改进了 Ghaffari 等人此前提出的 4-着色。 [GGJ20]。不过，这些算法似乎本质上仅适用于森林，并且关键在于每个节点周围的局部邻域中不含环。目前尚不清楚如何将这些方法应用于 $\lambda = 2$这一情形。

更广泛的背景——LOCAL 与可扩展 MPC、图指数化和稀疏化。LOCAL 算法与可扩展 MPC 算法之间有着密切联系。LOCAL 算法为设计可扩展 MPC 算法提供了自然的起点，而且我们通常可以期待获得快得多的 MPC 算法，甚至是快出指数级的算法。下面具体说明。若暂时忽略局部内存约束，一个 T轮 LOCAL 算法将意味着一个$O(\log T)$ 轮 MPC 算法，这一方法通常称为图指数化 [LW08]：我们需要进行$I = O(\log T)$ 次迭代。在第 i 次迭代中，我们只需让每个节点将自己从上一轮迭代中已知的（2^{i-1}）跳邻域发送给其（2^{i-1}）跳邻域中的所有节点，就能让每个节点获知距离其不超过 2^i 的所有节点（及其边）。最后，当（持有）每个节点的机器都获知其 T 跳邻域后，就可以直接在本地模拟 LOCAL 算法。我们指出，除了一些[3], 技术细节之外，这种指数级加速已是我们所能期待的最佳结果 [GKU19]。不过，问题在于，只有当局部邻域足够小、能够装入一台机器的内存时，这种指数级加速才可行。

---

[1]本文及下文中，我们用 $f(n) = \widetilde{O}(g(n))$ 表示省略关于 n 的多对数因子；也就是说，如果存在常数 $k \geq 0$，使得 $f(n) = O\big(g(n)(\log n)^k\big)$，则有 $f(n) = \widetilde{O}(g(n))$。[2]我们说明一下，他们更一般地对 *coreness decomposition* 给出了这一结论，但只需针对每个 $k = (1+\varepsilon)^i$ *coreness/arboricity estimate* 并行运行该算法即可。简要概述见 [Gha, Exercise 2.3] 。[3]该结论仅适用于随机算法，并以（广泛认可的）1-vs-2环猜想成立为前提，且仅适用于分量稳定MPC算法。详情参见 [GKU19] 。

情况并非总是如此，例如，对于 $T = \Theta(\log n)$ 轮 LOCAL 定向算法，其邻域可能包含图中的所有节点。

Ghaffari 和 Uitto [GU19]针对这一问题提出了一种分布式算法的稀疏化技术（应用于极大独立集和匹配算法）。简而言之，他们的稀疏化技术缩小了模拟 LOCAL 算法（部分步骤）时所需考虑的邻域范围。Ghaffari 等人 [GLM19] 将该技术应用于低出度定向，并表明：通过在一个随机选取的子图上运行一个 $\Theta(T')$ 轮的 LOCAL 算法，可以“模拟”由 $T = O(\log n)$ 轮 LOCAL 模型算法构成的每个阶段（包含 $T' = \Theta(\sqrt{\log n})$ 轮）；该子图的最大度为 $2^{\Theta(T')} = 2^{\Theta(\sqrt{\log n})}$。因此，只要恰当选择常数，相关的 $\Theta(T')$ 跳邻域的大小就是 $2^{\Theta((T')^2)} \leq n^{\delta}$，从而能够放入本地内存，为只需 $O(\log(T'))$ 个 MPC 轮次的图指数算法铺平道路。反复执行这一过程以运行全部 T/T' 个阶段后，MPC 轮复杂度为 $T/T' \cdot \log(T') = O(\sqrt{\log n})$。这一 $O(\sqrt{\log n})$ 上界是一般图定向问题的最先进复杂度（即使出度上界高得多，例如 $\lambda \text{poly}(\log n)$）。突破这一轮复杂度是本文的首要目标。

我们还指出，这一复杂度也是一般图上极大独立集与匹配的可扩展 MPC 算法目前已知的最优结果 [GU19], ，而改进该结果——例如改进到 $\text{poly}(\log \log n)$——是 MPC 文献中的核心开放问题之一。

1.3 我们的结果

我们提出一种可扩展 MPC 算法，可在 $\text{poly}(\log \log n)$ 轮次内计算出出度为 $O(\lambda \log \log n)$ 的定向，如下文正式所述。

定理 1.1. 存在一种随机化可扩展 *MPC* 算法，给定任意无向图 $G = (V, E)$，且具有 $n = |V|$ 与 $m = |E|$，该算法运行 $\text{poly}(\log \log n)$ 轮次，并计算边定向，使得每个节点的出度至多为 $O(\lambda \log \log n)$，以高概率成立。此处，λ 表示该图的森林度。该算法每台机器使用 n^{δ} 内存，其中 $\delta \in (0,1)$ 为任意正常数，并使用 $O(m+n)$ 字的全局内存。若 λ 的上界为 $(\log n)^{O(\log \log n)}$，则该算法是确定性的。

上述算法中的确定性部分适用于 $\lambda \leq (\log n)^{O(\log \log n)}$ 的情形，实际上提供了更多结构，这些结构将用于我们的着色结果：它将顶点集合 V 划分为 $L = \Theta(\log n)$ 层 $H_1, H_2, \ldots, H_{\Theta(\log n)}$，使得每个节点 $v \in H_i$ 在 $H_i \cup \cdots \cup H_L$ 中至多有 $O(\lambda \log \log n)$ 个邻居。只需将边定向到较高层（若层相同则任意打破平局，例如依据标识符），即可得到该定向。此外，层大小呈指数衰减，并且我们有 $|H_i| \leq n \cdot exp(-\Theta(i))$。这就是通常所说的 H-划分 [BE08,GLM+23]。利用这些性质，作为一个具体应用，我们为一般的 λ 图实现了一个同样快速的可扩展 MPC 算法，可使用 $O(\lambda \log \log n)$ 种颜色进行着色：

定理 1.2。 存在一种随机化可扩展 *MPC* 算法，对于任意无向图 $G = (V, E)$，满足 $n = |V|$ 和 $m = |E|$，该算法可在 $\text{poly}(\log \log n)$ 轮次内，以高概率计算出一种顶点着色，使用 $O(\lambda \log \log n)$ 种颜色。此处，λ 表示图的森林度。该算法使用每台机器 n^{δ} 内存，其中 $\delta > 0$ 是任意正常数，并使用 $\tilde{}$ 和 $O(m+n)$ 字的全局内存。

$\tilde{}$讨论。 上述两项结果的主要意义在于打破了之前讨论的 $\Theta(\sqrt{\log n})$ 复杂度障碍。不过，我们也注意到，我们的结果有一个缺点：出度上界略有增加，从 $O(\lambda)$ 增至 $O(\lambda \log \log n)$。在某些情况下，这一点可以

接受。例如，当定向产生的着色用于调度时，这个 $\log\log n$ 因子对时间复杂度的影响很小。尽管如此，如何在 poly(log log n) 个可扩展 MPC 轮次内达到 $O(\lambda)$ 的出度上界，仍然是一个值得研究的开放问题。

1.4 技术概述

定向 对于随机定向算法，只需设计一个适用于森林度为 $\lambda \leq O(\log n)$ 的图的算法；如果我们愿意接受出度损失一个 $(1+\varepsilon)$ 倍的因子，其中任意固定常数 $\varepsilon > 0$都可以，这对我们来说显然成立。为了将森林度较高的图归约为这些森林度较低的图，我们将图的边随机划分为 $\lambda/\Theta(\log n)$ 个部分。详情见 引理 2.1 。因此，从现在起，在本概述中，我们将重点讨论森林度为 $\lambda = O(\log n)$ 的图。

我们的定向方法的基本思路是“近似模拟”自然的 $\Theta(\log n$ 轮 LOCAL 过程 [BE08],，但只需 poly(log log n) 个 MPC 轮次。回顾一下，该 LOCAL 算法仅由 $L = \Theta(\log n)$ 次迭代组成；在每次迭代中， i 我们从剩余图中移除所有度为 $d = O(\lambda)$ 的节点，并将它们放入一层 H_i。这就得到一个划分$V = H_1 \sqcup H_2 \ldots \sqcup H_L$，在本概述的后续部分中，我们将以此为参照。通过将边定向至更高层（度相同时按标识符打破平局），即可保证出度至多为 d 。

如前所述，一个朴素的设想是，利用上文介绍的图指数化，将这个需要 $\Theta(\log n)$ 轮的 LOCAL 算法加速到 MPC 中的 $\Theta(\log\log n)$ 轮：只需进行 $\Theta(\log\log n)$ 次迭代，从而获知 $\Theta(\log n)$ 跳邻域。然而，这并非总是可行，因为邻域大小可能很大，无法装入本地内存。

从宏观上看，暂且忽略一些重要方面，我们的算法使用局部邻域的某些剪枝视图，执行图指数化的一种变体。这样一来，我们便能控制几乎所有感兴趣节点的邻域视图的增长。不过，要实现这一点，在每次指数化迭代中，我们都会从每个节点剪枝掉一些 $O(\lambda)$ 条边，并允许这些边的定向任意。这使得整体出度达到 $O(\lambda \log\log n)$。下面我们来仔细看看。

我们尝试部分“模拟” LOCAL 算法的前 $L' = O(\delta \log n/\log\log n)$ 轮。在上文固定的划分 $V = H_1 \sqcup H_2 \ldots \sqcup H_L$ 中（当然，我们的算法并不知道这个划分；这里只是为了帮助理解算法和进行分析），我们删去每一层内部的所有边，即边 (u, v)，其中 $u, v \in H_i$，对某个 $i \in [1, L]$ 成立。每个节点因此至多删去 $O(\lambda)$ 条边。这样，剩下的每条有向路径都会单调地经过各层。对于每个节点 v，从 v 出发、长度至多为 L' 的有向路径总数至多为 $d^{L'} \leq n^{\delta/3}$. 。因此，除至多 $n/n^{\delta/3}$ 个节点外，每个剩余节点 u 的终点为 u、长度至多为 L' 的有向路径不超过 $n^{2\delta/3}$ 条。对于这样的节点 u，其距离至多为 L' 的入邻域似乎可以装入一台机器的本地内存，因此我们有理由认为，应该能够“执行” LOCAL 模型中与该节点 u 相关的部分。当然，问题在于，这里讨论的是在固定定向中只能沿入边到达的邻域，而算法并不知道这个定向。如果不作这一区分，允许路径混合经过出边和入边，那么从一个节点可到达的路径数可能会非常大。

理想情况下，我们希望忽略沿着 $O(\lambda)$出边 学习邻域和指数化，并让这些边任意定向。但我们不知道哪些边是出边。我们尝试做一些效果类似的事情：直观来说，在这样一个假设情形中， v 的邻域大小可以看作沿着通向不同邻居的边对邻域大小求和（例如，如果所考虑的局部邻域只是树 [4]），我们

[4]从某种意义上说，可以认为这是 Grunau 等人的算法的一部分， [GLM+23] 用于森林中的定向和着色；在该算法中，他们能够区分并“忽略”指向最重子树的边/方向。但这在很大程度上依赖于他们处理森林时的树局部视图，而这与我们的一般图设定相去甚远。我们的

会删除最多 $O(\lambda)$ 条连接到最大邻域的边。然而，主要挑战在于我们处理的是一般图：由于环的存在（且可能数量很多），不存在这种求和性质。事实上，可以想象，由于存在一个很大的区域，通向许多邻居的连接都可能呈现出较大的邻域。

从某种意义上说，我们的算法迫使树状的朴素直觉得以实现。不过，算法及其分析大不相同，也复杂得多。在指数化过程中，我们让每个节点都将其邻域视图维护为一棵有根树：允许每个节点沿着树的不同分支在邻域中出现多次，本质上，每条到达该节点的不同路径都对应一次出现。这样就得到了一个树状视图，呈现连接到该节点的各条路径。现在，我们需要格外谨慎地处理这些树状视图（以及关于它们的保证）与图中实际邻域之间的关系。由于其中涉及的技术细节较多，我们会在此尝试概述这些方面。我们证明，在每次指数化过程中，都可以对这些树状局部视图进行剪枝：从叶节点开始向根迭代，每次为每个节点剪去 $O(\lambda)$个最重子树。这样可以确保未剪枝的视图能够放入 MPC 的本地内存中，并使我们得以进行指数化。经过 $O(\log\log n)$ 次指数化迭代后，这会转化为 $O(\lambda\log\log\ n)$ 的出度。不过，我们还需要额外的性质和论证，才能证明：如果基于这些剪枝后的树状视图模拟 LOCAL 算法，就能够将其保证转化到原图上。

着色。 对于着色，假设我们已经计算出了 $L=\Theta(\log n)$-层划分$V=H_1\sqcup H_2\ldots\sqcup H_L$，其出度至多为 $d=O(\lambda\log\log\ n)$，如上所述。此外，我们再次只关注具有 $\lambda=O(\log n)$ 的图；一般情况可归约到此情形（这次通过随机顶点划分，参见 引理2.2）。

从 LOCAL 模型的角度来看，我们可以在 $\Theta(\log n)$ 个阶段内为该图着色，在每个阶段中，我们对由 H_i 诱导的图进行一次 $degree+1$ 列表着色。每个阶段需要$\widetilde{O}(\log^{5/3}\log\ n)$ 个 LOCAL 轮次，这得益于最先进成果 [HKNT22、GG24b]。因此，整个算法需要 $O(\log\ n)\cdot\widetilde{O}(\log^{5/3}\log\ n)$ 轮次。

我们可以在 MPC 中以快得多的速度模拟上述过程：假设每一层内部的边都复制为双向边，而跨层的边则指向层级较高的端点。这样可以看出，在上述算法中，每个节点 $v\in H_i$ 的颜色只受从 v 出发、沿距离至多为 $(L-i+1)\cdot\widetilde{\Theta}(\log^{5/3}\log n)$ 的有向路径可达的节点影响。这使我们能够沿出边进行指数化，所得规模足够小，可以放入本地内存。例如，只需一次操作，我们就能让最后 $\Theta(\log\ n/\widetilde{\Theta}(\log^{8/3}\log\ n))$ 层中的所有节点获知从它们各自可达的所有其他节点，且只需 $O(\log\log n)$ 个 MPC 轮次。为此，我们沿出边进行有向图指数化 。随后便可利用这一结果，在本地模拟这 $\widetilde{\Theta}(\log\ n/(\log^{8/3}\log n))$ 层的着色。将此过程重复 $\mathrm{poly}(\log\log\ n)$ 次迭代，即可完成整个着色。

1.5 其他相关工作

Δ相关着色。 着色算法一直是分布式计算 LOCAL 模型中受到深入研究的主题，我们建议参阅 Barenboim 和 Elkin 于 2013 年撰写的 [BE13]以获取一份（略显过时的）综述。分布式着色中最基本的参数化基于图中的最大度 Δ。例如，参见 [CLP18]，其中介绍了 $\Delta+1$着色领域的最先进成果，该算法在 LOCAL 模型中仅需 $\mathrm{poly}(\log\log n)$ 轮次，以及 [HKNT22、GG24b] 中的推广和改进。

该算法的部分灵感源于尝试让这一想法在一般图中奏效。

Chang 等人 [CFG+19] 给出了一种用于 $\Delta+1$ 着色的可扩展 MPC 算法，该算法部分基于这一 LOCAL 算法，实现了 $O(\log\log\log n)$ 的轮复杂度——这需要将改进后的网络分解界 [RG20]代入其算法中。这是可扩展 MPC $\Delta+1$ 着色算法已知的最佳轮复杂度。我们还指出，在 MPC 的$\tilde{}$准线性（及更高）内存区间中，当本地内存为 $\Omega(n)$ 时，Assadi、Chen 和 Khanna 的一种算法 [ACK19] 可以在 $\Delta+1$ 着色中实现 $O(1)$ 轮次。然而，从技术角度来看，这与可扩展 MPC 区间有着极大的不同。

依赖密度的着色。上文提到的依赖于 Δ 的着色，对于许多我们感兴趣的图来说可能过于宽松。本文研究的森林度（或密度）指标，为广泛类型的图提供了更精确的着色界限。举个简单的例子，在星形图中，$\Delta=\Theta(n)$ 和 $\lambda=1$，当然，我们希望颜色数更接近后者。

如果我们处于 MPC 中的存区间，且每台机器拥有 $O(\lambda)$，可以通过简单的随机划分，将整个相关子图收集到一台机器中，$\tilde{}$在 $2+\varepsilon)\lambda$ 定向和 $1+\varepsilon)\lambda$ 内计算 O（1）轮次。在可扩展 MPC 中，然而，问题大不相同。在 LOCAL 模型中，关于森林度的着色最佳已知算法$\tilde{}$由 Ghaffari 和 Lymouri 提出 [GL17]，并在 O（λ）种颜色下，在 O（$\log n$）轮次内计算出一种着色。关于界的更精确表述，请参见他们的论文。另请参见 [BE08] 中的确定性算法。

其他问题与模型。还有大量切向相关的工作，全面综述这些工作超出了我们的范围。我们仍将提及一些关于可扩展 MPC、其他图问题，以及其他模型中依赖密度的着色的工作。极大独立集和匹配已在低森林度图中得到研究 [BBD+19、GGJ20、FGG23]。对于一般图（即非恒定 λ），忽略 $O(\log\log n)$ $\tilde{}$ 因子后，其复杂度停留在 $O(\sqrt{\log\lambda})$，这与前文提到的 [GU19] 的相同瓶颈；其本质原因是先将最大度 Δ 降至 poly(λ)，然后依赖 $\tilde{}$ 最先进水平的 Δ-依赖性 $O(\sqrt{\log\lambda})$。此外，已有大量关于可扩展 MPC $\tilde{}$ 算法的工作，用于在 $O(\log(diameter))$ 轮次 [ASS+18、BDE+19] 中解决图连通性问题，以及其去随机化版本 [CC22、FGG22、BLM+23]。上述讨论没有强调随机化 MPC 算法与确定性 MPC 算法之间的区别。关于 MPC 中的去随机化，近期研究的示例可参见 [CDP21a、CDP21b、CC22、FGG23]，及其中的引文。另请参见 [LMPU24] 和 [GG24a、CNR23]，分别了解其他模型中的依赖密度的着色算法，即自适应 MPC 和动态算法。

2 预备知识

以下两个引理是众所周知的。它们使我们能够通过对边/顶点进行简单随机划分，将有效森林度降至 $O(\log n)$。

引理 2.1 （边划分）。设 G 为一个至多有 n 个顶点的图，并设 $k\in\mathbb{N}$ 满足$k\geq\lambda(G)$。设 $L:=\lceil k/\log(n)\rceil$，且 G_1、G_2、……、G_L 为将边均匀随机划分为 L 份后得到的 L 个图。则，以高概率$\max_{i\in[L]}\lambda(G_i)=O(\log n)$。

证明。考虑对 G 中的边进行定向，使其出度为 $O(\lambda(G))$。现在，考虑某个固定顶点 v。期望情况下，v 在 L 个划分中的每个划分内的出边数量为 $O(\lambda(G))/L=O(\log n)$，而切尔诺夫界表明，以高概率该界成立。因此，根据并集界，以高概率每个节点在每个部分中都有 $O(\log n)$ 条出边，从而以高概率 $\max_{i\in[L]}\lambda(G_i)=O(\log n)$。 □

引理2.2 (顶点划分)。 设 G 是一个至多包含 n 个顶点的图，并设 $k \in \mathbb{N}$ 满足$k \geq \lambda(G)$。令 $L := \lceil k/\log(n) \rceil$，且令 $G_1, G_2, \ldots, G_L$ 为将顶点均匀随机地划分为 L 个部分后得到的 L 个顶点导出子图。那么，以高概率$\max_{i\in[L]} \lambda(G_i) = O(\log n)$。

证明。 考虑 G 中出度为 $O(\lambda(G))$的边定向。现在，考虑某个固定的顶点 v。期望情况下，与 v 处于同一部分的出邻居数量为 $O(\lambda(G))/L = O(\log n)$，而切尔诺夫界表明，以高概率该界成立。因此，联合界表明，以高概率每个节点在每个部分中都有 $O(\log n)$条出边，因此 $\max_{i\in[L]} \lambda(G_i) = O(\log n)$ 以高概率成立。

□

2.1 （部分）层赋值

我们使用以下符号表示层赋值。需要注意的是，这在概念上类似于引言中讨论的划分 $H_1 \sqcup \ldots \sqcup H_L$，但我们在此使用函数记法，以便能够通过相关的层分配函数 $\ell_G(v)$，明确引用每个给定顶点 v的层编号。

定义 2.1 (部分层赋值)。 设 G 为一个图，且 L 和 d 为正整数。 G 的一个部分层分配，若其有 L 层且出度为 d，则是一个函数

$$\ell_G \colon V(G) \to [L] \cup \{\infty\}$$

使得对于每个满足 $\ell_G(v) \neq \infty$ 的顶点 $v \in V(G)$，以下结论成立：

$$\big|\{\, u \in N_G(v) : \ell_G(u) \geq \ell_G(v)\}\big| \leq d.$$

在我们的算法中，每个节点仅使用其局部视图计算一个部分层分配。因此，一个节点可能从不同的部分层分配中得到不同的层号。我们只需为每个节点指定它从任一部分层分配中得到的最小层号，即可合并这些不同的层赋值。以下命题表明，取两个部分层分配的最小值，会得到另一个有效的部分层分配，且层数相同、出度有界。

命题 2.3 （两个部分层分配的最小值）。设 G 是一个图， L 和 d 是正整数。假设

$$\ell_G^{(1)} \colon V(G) \to [L] \cup \{\infty\} \quad \textit{and} \quad \ell_G^{(2)} \colon V(G) \to [L] \cup \{\infty\}$$

是 G 的两个部分层分配，具有 L 层和出度 d。定义 ℓ_G 如下：

$$\ell_G(v) \;=\; \min\big\{\ell_G^{(1)}(v), \ell_G^{(2)}(v)\big\} \quad \textit{for all } v \in V(G).$$

那么 ℓ_G 也是 G 的一个部分层分配，具有 L 层和出度 d。

证明。函数 ℓ_G 显然映射到 $[L] \cup \{\infty\}$ 中。对于任意满足 $\ell_G(v) \neq \infty$ 的 v，设 $\ell_G(v) = k$，我们必须证明

$$|\{\, u \in N_G(v) : \ell_G(u) \geq k\}| \leq d.$$

由于 $k = \min(\ell_G^{(1)}(v), \ell_G^{(2)}(v))$，因此要么成立 $k = \ell_G^{(1)}(v)$，要么成立 $k = \ell_G^{(2)}(v)$。如果 $\ell_G^{(1)}(v) = k$，则

$$\{\, u \in N_G(v) : \ell_G(u) \geq k\} \subseteq \{\, u \in N_G(v) : \ell_G^{(1)}(u) \geq \ell_G^{(1)}(v)\},$$

其大小至多为 d，因为 $\ell_G^{(1)}$ 是出度为 d的部分层分配。若 $\ell_G^{(2)}(v) = k$，则对称论证同样适用。因此 ℓ_G 也满足出度条件，从而完成证明。

□

定义 2.2（严格递增路径与路径计数）。设 G 为一个图，L 为一个正整数，并设 $\ell_G\colon\ V(G) \to [L] \cup \{\infty\}$ 为 G的部分层分配。

路径 $P = (v_1,\ v_2,\ \ldots\ldots,\ v_k)$ 在 G 中称为严格递增（相对于 ℓ_G），如果

$$\ell_G(v_1) \ <\ \ell_G(v_2) \ <\ \cdots \ <\ \ell_G(v_k) < \infty.$$

对于每个顶点 $v \in V(G)$，定义 $\mathrm{NumPathsIn}_{G,\,\ell_G}(v)$ 为 G 中以 v 结尾的不同严格递增路径的数量，并定义$\mathrm{NumPathsOut}_{G,\,\ell_G}(v)$ 为 G 中以 v 开始的不同严格递增路径的数量。

我们使用以下引理来证明，对于大多数节点 v，$\mathrm{NumPathsIn}_{G,\,\ell_G}(v)$ 都很小。然后我们证明，$\mathrm{NumPathsIn}_{G,\,\ell_G}(v)$ 较小的节点将由我们的算法赋予一层。

引理 2.4。设 G 是一个图，L 和 d 是正整数。令 $\ell_G\colon\ V(G) \mapsto [L]$ 为一个出度为 $d \geq 2$ 的完整分层赋值。则有

$$\begin{aligned}
\sum_{v\in V(G)} \mathrm{NumPathsIn}_{G,\ell_G}(v) &= \sum_{v\in V(G)} \mathrm{NumPathsOut}_{G,\ell_G}(v) \\
&\leq |V(G)| \cdot \max_{v\in V(G)} NumPathsOut_{G,\ell_G}(v) \\
&\leq |V(G)| \cdot \left(\sum_{j=0}^{L-1} d^j\right). \\
&\leq |V(G)| \cdot d^L.
\end{aligned}$$

证明。第一个等式由一个简单的双重计数论证得出。因此，只需证明对于每个 $v \in V(G)$，$|NumPathsOut_{G,\ \ell_G}(v)| \leq \sum_{j=0}^{L-1} d^j$。我们通过对层 $i = \ell_G(v)$ 归纳，证明一个稍强的陈述：

$$\mathrm{NumPathsOut}_{G,\ell_G}(v) \ \leq\ \sum_{j=0}^{L-i} d^j \quad \text{for all vertices } v \text{ with } \ell_G(v) = i.$$

基础情形（$\ell_G(v) = L$）：不存在层级高于 L 的顶点，因此从 v 出发的每条严格递增路径都必定是单顶点路径 (v)。因此 $\mathrm{NumPathsOut}_{G,\,\ell_G}(v) \ =\ 1 \ =\ \sum_{j=0}^{0} d^j$，符合要求。

归纳步骤：假设该命题对层 $> i$ 中的所有顶点都成立。设 v 满足 $\ell_G(v) = i < L$。从 v 出发的任意严格递增路径，要么仅为 (v)，要么延伸至满足 $\ell_G(u) > i$ 的邻居 u。因此，

$$\mathrm{NumPathsOut}_{G,\ell_G}(v) \ =\ 1 + \sum_{\substack{u\in N_G(v)\\ \ell_G(u)>i}} \mathrm{NumPathsOut}_{G,\ell_G}(u).$$

由于 v 在更高层中至多有 d 个邻居，并且根据归纳假设 $\mathrm{NumPathsOut}_{G,\,\ell_G}(u) \ \leq \sum_{j=0}^{L-(i+1)} d^j$，我们得到

$$\mathrm{NumPathsOut}_{G,\ell_G}(v) \ \leq\ 1 \ +\ d \cdot \sum_{j=0}^{L-i-1} d^j \ =\ \sum_{j=0}^{L-i} d^j.$$

至此，归纳法完成，证明也就结束了。 □

2.2 有效映射

在图幂运算过程中，每个节点都维护一棵有根树，用于刻画其局部邻域的一部分。树中的每个树节点都对应原图中的一个节点，并且树中的不同节点可能对应同一个节点。树中的每条边都映射到图中的一条边；同时，我们还保持以下性质：给定树节点的子节点分别映射到原图中的不同节点。

定义 2.3 (**带有效映射的树**)。设 G 是一个图， T 是一棵有根树，且 $map\colon V(T) \mapsto V(G)$。若满足以下条件，则称 map 是一个有效映射：

1。对于树 T 中的每条边 (x, y)， $\{map(x)\ map(y)\}$ 是 G 中的一条边；并且

2. 对于每个节点 $x \in V(T)$，以及 x 的任意两个不同子节点 c_1 和 c_2，均有 $map(c_1) \neq map(c_2)$。

定义 *2.4*（树剪枝与有效映射）。设 G 是一个图， T 是一棵带有有效映射 $map\colon V(T) \mapsto V(G)$ 的有根树。设 T' 是一棵有根树，它与 T 具有相同的根，并由移除 T 中的一些节点得到。我们将对应的有效映射 $map'\colon V(T') \to V(G)$ 定义为将 map 限制在 $V(T')$ 上。换言之，对于每个节点 $v \in V(T')$，令 $map'(v) := map(v)$。

树附加过程将用于每一个 $O(\log\log n)$图幂运算步骤。

定义 *2.5*（树附加，含有效映射）。设 G为一个图， T 为一棵具有有效映射 $map\colon V(T) \mapsto V(G)$ 的有根树，并设 x_1、x_2、……、x_η 为 T中互不相同的叶节点。此外，设 T_1、T_2、……、T_η 为有根树，且 map_1、map_2、……、map_η 使得 $map_i\colon V(T_i) \mapsto V(G)$为有效映射，并且 T_i 的根 r_i 满足 $map(x_i) = map_i(r_i)$。我们将附加 T' 定义为由 T将每个 x_i 替换为（ T_i的一个新副本）而得到的有根树，其中新映射 map' 对于曾位于 $V(T) \setminus (\cup_{i=1}^{\eta}\{x_i\})$中的 T' 的节点定义为与 map 相同，对于现位于 T'中的 T_i 的节点定义为与 map_i 相同。

定义 2.6 (**缺失邻居**)**.** 设 G 是一个图， T 是一棵有根树，且 $map\colon V(T) \mapsto V(G)$ 是一个有效映射。对于节点 $x \in V(T)$，我们定义

$$Missing_{G,T,map}(x) := N_G(map(x)) \setminus \{map(c)\colon c \in children_T(x)\}.$$

定义 2.7 (**严格单调可达**)**.** 设 G 是一个图，并设 $\ell_G\colon V(G) \to \mathbb{N}\cup\{\infty\}$ 是一个部分层分配。此外，设 T 是一棵根为 r 的有根树，并设映射 $: V(T) \to V(G)$ 是一个有效映射。若顶点 $x \in V(T)$ 相对于 ℓ_G 满足以下条件，则称其为严格单调可达： 当我们记

$$x = x_1, x_2, \ldots, x_k = r$$

从 x 到 T 中根 r 的唯一路径时，有

$$\ell_G\big(map(x_1)\big) < \ell_G\big(map(x_2)\big) < \cdots < \ell_G\big(map(x_k)\big).$$

3 边定向算法

现在，我们可以解释边定向算法的不同部分了。

3.1 剪枝

LocalPrune 过程会递归地移除权重最大的 k 子树。我们将使用它来 $k = O(\lambda(G))$. 该算法将在单台机器上本地执行，无需任何通信。

算法 1： LocalPrune

输入： 一棵根为 r的有根树 T，剪枝参数 k
输出： 由 T导出的有根子树 T_{pruned}
如果 r 至多有 k 个子节点 **则**
  返回仅由 r组成的单节点有根树
end
对于 c **的每个子节点** r，**执行**
  令 T_c 为以 c 为根的子树
  T_c，LocalPrune(T_c，k)
end
// 找出并移除子节点中最大的 k 个剪枝子树
令 $\mathcal{C} \leftarrow \{T_{c,\text{pruned}}：\ c$ 是 $r\}$ 的子节点
按大小对 $\mathcal{C}$ 中的子树进行降序排序（并列时任意打破）
从 $\mathcal{C}$ 中移除前 k 棵子树（即最大的 k 棵）
// 通过将剩余的子节点连接到 r来构成剪枝后的树
创建一棵以 r为根的新树 T_{pruned}
对每个$T_{c,\text{pruned}} \in \mathcal{C}$ **执行**
  在 T_{pruned}中将$T_{c,\text{pruned}}$ 作为 r 的子树进行连接
end
返回 T_{pruned}

命题 3.1. 设 G 是一张图，T 是一棵有根树，且 map: $V(T) \mapsto V(G)$ 是一个有效映射。固定任意整数 $k \geq 1$，并考虑带有诱导映射的 $T_{\text{pruned}} \leftarrow LocalPrune(T, k)$

$$map_{\text{pruned}}(x) \ := \ map(x) \quad for\ every\ x \in V\big(T_{\text{pruned}}\big).$$

那么，对每个 $x \in V\big(T_{\text{pruned}}\big)$，都有

$$Missing_{G,T_{\text{pruned}},map_{\text{pruned}}}(x) \ \leq \ Missing_{G,T,map}(x) \ + \ k.$$

证明。我们对 T 的高度进行归纳证明：

基础情形。如果 T 的根 r 至多有 k 个子节点，那么剪枝算法返回仅由 r 组成的单节点树。由于 r 至多失去 k 个子节点，因此显然有

$$Missing_{G,T_{\text{pruned}},map_{\text{pruned}}}(r) \ \leq \ Missing_{G,T,map}(r) \ + \ k.$$

没有其他节点留下，因此该命题显然成立。

归纳步骤。假设 r 有超过 k 个子节点 c_1，.....，c_m。对于每个 c_j，考虑以 c_j 为根的子树 T_{c_j}，并令 $T_{c_j,\ pruned}$ 为其剪枝后的版本，$map_{c_j,\ pruned}$：$V(T_{c_j,\ pruned}) \mapsto V(G)$ 为其映射。根据归纳假设，$T_{c_j,\ pruned}$ 中的每个节点 x 都满足

$$\text{Missing}_{G,T_{c_j},pruned,\,\text{map}_{c_j},pruned}(x) \ \leq \ \text{Missing}_{G,T_{c_j},\,\text{map}}(x) \ + \ k.$$

然后，算法从 r 的子节点中移除最大的 k 个剪枝子树。因此，由于每个$T_{c_j,\ pruned}$ 至少包含一个节点，r 恰好失去 k 个子节点，导致其缺失集至多增加 k。

因此，对于最终剪枝后的树 T_{pruned} 中的每个 x，$\text{Missing}_{G,\ T_{\text{pruned}},\ 映射pruned}(x)$ 至多为 $\text{Missing}_{G,\ T,\ \text{map}}(x) + k$。 □

引理 3.2。 设 G 是一个图， T 是一棵以 r 为根的有根树。假设我们有一个有效的映射

$$\mathrm{map}\colon V(T) \to V(G).$$

设 L 和 d 为正整数，并设 ℓ_G: $V(G) \to [L] \cup \{\infty\}$ 是一个具有出度 d 的部分层分配，满足 ℓ_G（map（r））$\neq \infty$。此外，设 $k \geq d$ 为正整数，并通过调用 LocalPrune（T、k） 得到 T_{pruned}。于是，

$$\left|V\left(T_{\text{pruned}}\right)\right| \leq \mathrm{NumPathsIn}_{\mathrm{G},\ell_{\mathrm{G}}}\left(\mathrm{map}(r)\right).$$

证明。 我们通过对 ℓ_G（map(r)）进行归纳法来证明:

基础情形： ℓ_G（map(r)) $= 1$。 由于 ℓ_G 的出度为 d，第 1 层中的任意顶点在 G 中至多有 d 个邻居。由于 map 有效，这意味着 r 至多有 $d \leq k$ 个子节点位于 T 中。因此，算法 LocalPrune(T, k) 将 T 剪枝为单节点树 $\{r\}$。于是， $\left|V(T_{\text{pruned}})\right| \leq 1$。另一方面，至少存在一条以 map($r$) 结尾的严格递增路径（平凡路径），因此，$\mathrm{NumPathsIn}_{\mathrm{G},\ell_{\mathrm{G}}}\left(\mathrm{map}(r)\right) \geq 1$。因此，

$$\left|V\left(T_{\text{pruned}}\right)\right| \leq 1 \leq \mathrm{NumPathsIn}_{\mathrm{G},\ell_{\mathrm{G}}}\left(\mathrm{map}(r)\right),$$

这就解决了基础情形。

归纳步骤： ℓ_G(map(r)) $= j > 1$。 假设对于所有根满足 ℓ_G(map(r')) $< j$ 的有根树 T'，该引理都成立。我们证明它对当前根 r 也成立，其中该根满足 ℓ_G(map(r)) $= j$。

1. **按层划分 r 的子节点**。 将 r 的子节点划分为两个集合 :

$$C_{<j} = \left\{ c \in \mathrm{children}_T(r) : \ell_G(\mathrm{map}(c)) < j \right\} \quad \text{and} \quad C_{\geq j} = \mathrm{children}_T(r) \setminus C_{<j}.$$

根据部分层分配的出度条件（并且由于 map 有效且 ℓ_G(map(r)) $\neq\infty$），顶点 map(r) 在同一层或更高 层中的邻居至多有 d 个；特别是，

$$\left|C_{\geq j}\right| \leq d \leq k.$$

2. **递归剪枝每个子节点**。 对于每个子节点 c，令 T_c 为以 c 为根的子树。算法调用 LocalPrune(T_c, k)，生成 $T_{c,\ \text{pruned}}$。

- 若 $c \in C_{<j}$，则 ℓ_G(map(c)) $< j$。根据归纳假设，

$$\left|V\left(T_{c,\text{pruned}}\right)\right| \leq \mathrm{NumPathsIn}_{\mathrm{G},\ell_{\mathrm{G}}}\left(\mathrm{map}(c)\right).$$

3. **移除最大的 k 剪枝子树**。 对每个子节点的子树进行剪枝后， LocalPrune(T, k)按大小（降序）对这些剪枝子树进行排序，并移除前 k 个，不再予以考虑。记 $\mathcal{R}$ 为其子树得以保留的索引（子节点）集合。因此，

$$\left|V\left(T_{\text{pruned}}\right)\right| = 1 + \sum_{c\in\mathcal{R}} \left|V\left(T_{c,\text{pruned}}\right)\right|.$$

4. **界定 T_{pruned} 的大小**。 由于移除了 $|C_{\geq j}| \leq k$ 和最大的 k 剪枝子树，我们得到

$$\left|V\left(T_{\text{pruned}}\right)\right| = 1 + \sum_{c\in\mathcal{R}} \left|V\left(T_{c,\text{pruned}}\right)\right| \leq 1 + \sum_{c\in C_{<j}} |V(T_{c,pruned})|.$$

因此，

$$\begin{aligned}
\left|V\left(T_{\text{pruned}}\right)\right| &\leq 1+\sum_{c \in C_{<j}}\left|V\left(T_{c, pruned}\right)\right| \\
&\leq 1+\sum_{c \in C_{<j}} \text{NumPathsIn}_{\mathrm{G}, \ell_{\mathrm{G}}}\left(\text{map}(c)\right) \\
&\leq 1+\sum_{u \in N_G(map(r)):\, \ell_G(u)<\ell_G(map(r))} \text{NumPathsIn}_{\mathrm{G}, \ell_{\mathrm{G}}}(u) \\
&= \text{NumPathsIn}_{\mathrm{G}, \ell_{\mathrm{G}}}\left(map(r)\right).
\end{aligned}$$

□

3.2 指数化 +剪枝

我们现在描述我们的图指数化算法，它在 $s = O(\log \log\ n)$步内运行。在该算法中，每个节点 v 维护一棵有根树，以及一个将该树的根分配给 v的有效映射。关键在于，每棵有根树的大小都保持在预算 B之内。

这 s 个步骤中的每一步都分两个阶段进行：

1. **局部剪枝：**每个节点对其有根树进行剪枝，所使用的过程详述于算法 1。此步骤完全是局部的（每棵树都存储在单台机器上），因此不需要通信。

2. **图指数化：**利用剪枝后的树，该算法随后通过附加过程执行图指数化步骤，该过程定义于定义 2.5。

算法 2： ExponentiateAndLocalPrune

输入： 一个图 G、一个预算 B、一个剪枝参数 k和一个步数 s

输出： 对于每个 $v \in V(G)$，输出一个具有有效映射 map_v 的有根子树 T_v： $V(T_v) \to V(G)$。

初始化：

对每个 $v \in V(G)$ 具有 $|N_G(v)| < B$，**执行**

  构造一棵有根树 $T_v^{(0)}$ 具有 $|N_G(v)|$ 个子节点。

  定义 $map_v^{(0)}$： $V(T_v^{(0)}) \mapsto V(G)$，使得 $T_v^{(0)}$ 的根映射到 v，每个子节点映射到 $N_G(v)$ 中一个不同的邻居。

  将 v 标记为 活跃。

end

对于每个 $v \in V(G)$ 满足 $|N_G(v)| \geq B$，**执行**

  构造一棵有根树 $T_v^{(0)}$ 其中包含一个映射到 v的单节点。

  将 v 标记为非活动。

end

对于 $i = 1$ **至** s **执行**// 主循环

  局部剪枝步骤：

  对每个 $v \in V(G)$ 执行

    $T_{v,\text{已剪枝}}^{(i-1)} \leftarrow \text{LocalPrune}(T_v^{(i-1)}, k)$。

    限制 $map_v^{(i-1)}$ 以得到 $map_{v,\text{已剪枝}}^{(i-1)}$ 于 $T_{v,\text{已剪枝}}^{(i-1)}$。

    如果 $\left|V\left(T_v^{(i-1)}{}_{,\text{已剪枝}}\right)\right| > \sqrt{B}$ **则**// 检查大小是否超出预算 B

      标记 v 为非活动。

    end

  end

  指数化 / 附加步骤：

  对每个 $v \in V(G)$ **执行**

    如果 v 为 非活动**则**

      $T_v^{(i)} \leftarrow T_{v,\text{已剪枝}}^{(i-1)}$ // 不再进一步扩展

    else

      令 x_1、.....、x_η 为距离根恰好为 2 的叶节点 $T_{v,\text{剪枝}}^{(i-1)}$ 的叶节点，它们映射到一个活跃顶点。

      对每个x_j **执行**

        令 $u_j \leftarrow map_v^{(i-1)}{}_{,\text{剪枝}}(x_j)$，并取回子树 $T_{u_j}^{(i-1)}{}_{,\text{剪枝}}$ 及其有效映射 $map_{u_j,\text{剪枝}}^{(i-1)}$。

      end

      附加这些剪枝子树 $\left(T_{u_j}^{(i-1)}{}_{,\text{剪枝}}, map_{u_j}^{(i-1)}{}_{,\text{剪枝}}\right)$ 到 x_j 的叶节点 $T_{v,\text{剪枝后的}}^{(i-1)}$（参见定义 2.5）。

      获取 $T_v^{(i)}$ 及其有效映射 $map_v^{(i)}$.

    end

  end

end

返回 $T_v^{(s)}$和 $map_v^{(s)}$，对于每个 $v \in V(G)$。

命题 3.3。 对于每个 $i \in \{0, 1, \ldots, s\}$ 和 $v \in V(G)$，都有 $map_v^{(i)}$: $V(T_v^{(i)}) \mapsto V(G)$ 是一个有效映射。

证明。 回顾一下，映射 map: $V(T) \to V(G)$ 是有效的，当且仅当：

1. 对于 T 中的每条边 (x, y)，配对 $\{map(x)\ map(y)\}$ 是 G 中的一条边。

2. 对于每个节点 $x \in V(T)$，其任意两个不同的子节点 $c_1 \neq c_2$ 都满足 $map(c_1) \neq map(c_2)$。

我们对 i 使用归纳法证明该命题。

基础情形 ($i = 0$)。 如果 $|N_G(v)| < B$，那么构造 $T_v^{(0)}$，使其根映射到 v，且该根的子节点映射到 G 中 v 的不同邻居。因此，每条树边（根,子节点）都映射到 G 中的一条边，并且根的任意两个子节点的像都不同。因此，对于所有 $v \in V(G)$， $map_v^{(0)}$ 都是有效的。如果 $|N_G(v)| \geq B$，那么 $T_v^{(0)}$ 仅由一个根组成，因此显然是有效的。

归纳步骤（$i - 1 \to i$）。假设对于某个 $i - 1 \geq 0$，映射 $map_u^{(i-1)}$对每个 $u \in V(G)$ 都有效。我们证明，对于每个 $v \in V(G)$， $map_v^{(i)}$ 都有效：

- **剪枝**。我们通过从 $T_{(i-1)v}$ 中移除一些节点，得到经过剪枝的 $T_{(i-1)\,v,}$ 。由于 $(i-1)v$ 在 $T_{(i-1)}$ v 上有效，将其限制到经过剪枝的子树 $T_{(i-1)\,v,}$ 可保持像的邻接关系以及兄弟节点的互异性条件。
- **附加**。根据定义 2.5，经过剪枝的 $T_{(i-1)}$ v 中的每个叶节点，（映射到经过剪枝的 $(i-1)$ v 中的 剪枝(x_j)），都将被整个经过剪枝的子树 $T_{(i-1)}$ u_j所替换。具体而言，

  1. 从 $T_v^{(i-1)}$，经过剪枝的 中移除 x_j 。2. 插入经过剪枝的 $T_{u_j}^{(i-1)}$，，其根也映射到 u_j，因此新树中的父边连接 x_j 的父节点与插入子树的根。

  归纳假设保证每个插入的子树都已有一个有效映射。由于 x_j 的父节点被映射到了 G中的某个节点 p，而新的根也被映射到了u_j，因此邻接关系得以保留（因为当 x_j 存在时， $\{p, u_j\}$ 是 G 中的一条边）。此外，不同插入子树中的子节点，或每个有效子树内部的兄弟节点，仍然映射到不同的顶点。

因此， $T_v^{(i)}$被有效地映射到 G中，从而完成归纳证明。

结论。 根据关于 i的归纳法，对于所有 $i \in \{0, \ldots, s\}$ 和 $v \in V$（G）， $map_v^{(i)}$都是有效的。 □

命题 3.4。 对于每个 $v \in V$（G）以及每个整数 $i \in \{0, 1, \ldots, s\}$，树 $T_v^{(i)}$至多有 B个节点。

证明（对 i进行归纳）。**基础情形**（$i = 0$）。如果 $|N_G(v)| \geq B$，那么 $T_v^{(0)}$仅由一个单节点组成。如果 $|N_G(v)| < B$，那么 $T_v^{(0)}$是一棵有 $1 + |N_G(v)| \leq B$ 个节点的星。

归纳步骤（$i \to i+1$）。假设对于某个 $i \in \{0, 1, \ldots, s-1\}$，每个顶点 $u \in V(G)$ 都有 $\left|T_{\,(i)}\right.$ $\left._u\right| \leq B$。我们必须证明，对于每个 v，都有 $|T_v^{(i+1)}| \leq B$。

情况 *1*： v 为非活动状态。那么 $T_v^{(i+1)} = T_v^{(i)})$，剪枝后 $\subseteq T_{(i)}\, v$，因此

$$\left|T_v^{(i+1)}\right| \;\le\; \left|T_v^{(i)}\right| \;\le\; B \quad \text{(by the induction hypothesis).}$$

情况 *2*： v 处于活跃状态。此处 $|T_v^{(i)}$，被剪枝$| \le \sqrt{B}$，且每棵附着子树 $T_{(i)}\, u_j$，也被剪枝，其大小为$\le \sqrt{B}$。由于这类附着子树最多有 $\sqrt{B}$ 棵，因此我们得到

$$\left|T_v^{(i+1)}\right| \;\le\; \left|T_{v,\text{pruned}}^{(i)}\right| \cdot \max(1, \max_j \left|T_{u_j,\text{pruned}}^{(i)}\right|) \le \sqrt{B} \cdot \sqrt{B} = B.$$

□

声明 3.5。 算法 *2* 可以在 $O(s)$ 个 *MPC* 轮次内实现，使用 $O(n^\delta + B)$ 字的本地内存和 $O(nB + m)$ 字的全局内存，其中 n 是顶点数的上界，m 是图 G的边数上界。

证明。 该断言直接由 声明 3.4 以及使用先前工作中提出的标准 MPC 原语进行的直接实现得出，例如 [[ASS+18]，E 节] 及其中的参考文献。

□

断言 3.6。 在第 i 次迭代中，对于每个映射到活跃节点的节点 $x \in T_{(i)}\, v$，若其满足 $dist_{T_v^{(i)}}(r,\ x) < 2^i$，则有

$$|Missing_{G,T_v^{(i)},map_v^{(i)}}(x)| \le ik.$$

证明。 **基础情形（$i = 0$）：**

当 $i = 0$ 时，根是距离根严格小于 $2^0 = 1$ 的唯一节点。如果根映射到一个活跃节点，那么由于剪枝尚未发生，根节点的所有邻居都已映射到初始树 $T_v^{(0)}$ 中。因此，根节点缺失邻居的数量为

$$|\text{Missing}_{G,T_v^{(0)},map_v^{(0)}}(r)| = 0 \le 0 \cdot k.$$

因此，基础情形成立。

归纳步骤：

考虑树 $T_{(i+1)v}$ 中一个映射到活跃节点的节点 x，它与根 r 的距离为 $\text{dist}_{T_v^{(i+1)}}(r,\ x) < 2^{i+1}$。我们需要证明

$$|Missing_{G,T_v^{(i+1)},map_v^{(i+1)}}(x)| \le (i+1)k.$$

根据经过第 $i+1$ 次迭代后，x 到树中根 r 的距离，有两种可能：

情况 1： $\text{dist}_{T_v^{(i+1)}}(r, x) < 2^i$

由于距离$_{T_v^{(i+1)}}(r,\ x) < 2^i$，根据归纳假设，节点 x 在 $i \cdot k$ 中至多有 $T_v^{(i)}$个缺失邻居。在第 $(i+1)$ 次迭代中，LocalPrune 步骤可能从每个节点移除至多 k 棵子树，从而可能使 x 的缺失邻居数量最多增加 k（见命题 3.1）。因此，在 $T_v^{(i+1)}$中，节点 x 满足：

$$|Missing_{G,T_v^{(i+1)},map_v^{(i+1)}}(x)| = |Missing_{G,T_{v,pruned}^{(i)},map_{v,pruned}^{(i)}}| \le |Missing_{G,T_v^{(i)},map_v^{(i)}}(x)| + k \le (i+1) \cdot k.$$

情况 2： $2^i \le$ 距离$T_v^{(i+1)}(r,\ x) < 2^{i+1}$ 节点 x 位于一棵子树中；在第 2^i 层，根节点到该子树所附着的叶节点的距离恰好为 2^i，而该子树是在第（$i+1$）次迭代期间附着上去的。令 r' 表示该子树的根，令 $u' \leftarrow map_v^{(i+1)}(r')$ 表示映射到的节点 r'。则有

$$dist_{T_{u'}^{(i)}}(r', x) = dist_{T_v^{(i+1)}}(r, x) - dist_{T_v^{(i+1)}}(r, r') = dist_{T_v^{(i+1)}}(r, x) - 2^i < 2^{i+1} - 2^i = 2^i.$$

由于 $dist_{T_{u'}^{(i)}}(r',\ x) < 2^i$ 且 x 映射到一个活跃节点，我们可以应用归纳假设，从而得出

$$|Missing_{G,T_{u'}^{(i)},map_{u'}^{(i)}}(x)| \le i \cdot k.$$

因此，我们可以使用 命题 3.1 得出

$$\begin{aligned} |Missing_{G,T_v^{(i+1)},map_v^{(i+1)}}(x)| &= |Missing_{G,T_{u',pruned}^{(i)},map_{u',pruned}^{(i)}}(x)| \\ &\le |Missing_{G,T_{u'}^{(i)},map_{u'}^{(i)}}(x)| + k \\ &\le (i+1)k. \end{aligned}$$

□

引理 3.7。 设 G 为一个图，d、$L \in \mathbb{N}$ 和 $\ell_G\colon\ V(G) \mapsto [L] \cup \{\infty\}$，使得对于每个 $v \in V(G)$ 满足 $\ell_G(v) \ne \infty$、$|\{u \in N_G(v)\colon\ \ell_G(u) \ge \ell_G(v)\}| \le d$。

*Furthe*此外，设 B、k、s 为满足 $k \ge d$ 且 $s >$ 对数2（L 的三个非负整数 ）.

设 $v \in V(G)$ 满足 $\ell_G(v) \ne \infty$ 且 $NumPathsIn_{G,\ \ell_G}(v) \le \sqrt{B}$.

设 $T_v^{(s)}$和 $map_v^{(s)}$为通过调用所得到的有根树和映射

$ExponentiateAndLocalPrune(G, B, k, s)$.

那么，对于每个 $x \in V(T_v^{(s)})$，若其相对于 ℓ_G（参见定义 2.7）严格单调可达，则有

$$|Missing_{G,T_v^{(s)},map_v^{(s)}}(x)| \le s \cdot k.$$

证明。考虑任意相对于 ℓ_G 严格单调可达的节点 $x \in T_v^{(s)}$。由于 $\ell_G(v) \le L$，且 x 严格单调可达，因此 x 到根的距离至多为 $L - 1 < 2^s$。此外，x 单调可达也意味着

$$NumPathsIn_{G,\ell_G}(map(x)) \le NumPathsIn_{G,\ell_G}(v) \le \sqrt{B}.$$

因此，由于 $k \ge d$，引理 3.2 推出 $V(T_{(i)\,map(x)\,pruned}) \le NumPathsIn_{G,\ \ell_G}(map(x)) \le \sqrt{B}$对一每个 $i \in \{0, 1, \ldots, s-1\}$ 均成立，因此 $map(x)$ 一直处于活跃状态，直至结束。因此，x 到根的距离小于 2^s，且它映射到一个活跃节点。因此，我们可以利用 断言 3.6得出结论：

$$|Missing_{G,T_v^{(s)},map_v^{(s)}}(x)| \le s \cdot k.$$

□

3.3 部分层分配

执行图指数化算法（算法 2）后，对于每个节点 v，我们都有一棵有根树。对于每棵树，我们使用 算法 3 通过一个简单的剥离过程，为树的部分顶点分配层，之后我们令 $a = O(\lambda(G)\log\log n)$。算法 3 将在单台机器上执行，无需任何通信（假设每个树节点都知道 $|Missing(x)|$）。

算法 3： PartialLayerAssignmentTree

输入： 图 G，有根树 T，有效映射 $map\colon\ V(T) \mapsto V(G)$，$a \in \mathbb{N}, L \in \mathbb{N}$

输出： 层赋值 $\ell_T\colon\ V(T) \mapsto [L] \cup \{\infty\}$

对于每个 $x \in V(T)$，令 $Missing(x) := N_G(map(x)) \setminus \{map(c)\colon\ c \in children_T(x)\}$

$V_{\geq 1} \leftarrow V(T)$

对 $j = 1, 2, \ldots, L$ **执行**

  $V_j \leftarrow \{x \in V_{\geq j}\colon\ |children_T(x) \cap V_{\geq j}| + |Missing(x)| \leq a\}$

  对于每个 $x \in V_j$，$\ell_T(x) \leftarrow j$

  $V_{\geq j+1} \leftarrow V_{\geq j} \setminus V_j$

end

对于每个 $x \in V_{\geq L+1}$，$\ell_T(x) \leftarrow \infty$

返回 ℓ_T

引理 *3.8.* 设 G 是一个图，d、$L \in \mathbb{N}$ 和 $\ell_G\colon\ V(G) \mapsto [L] \cup \infty$，满足对于每个 $v \in V(G)$，若 $\ell_G(v) \neq \infty$，则 $|\{u \in N_G(v)\colon\ \ell_G(u) \geq \ell_G(v)\}| \leq d$。此外，设 T 是一棵有根树，$map\colon\ V(T) \mapsto V(G)$ 是一个有效映射，且 $missing \in \mathbb{N}_0$ 是一个非负整数，满足以下条件：对于每个相对于 ℓ_G（见定义 *2.7*）严格单调可达的 $x \in V(T)$，都有 $|Missing(x)| \leq missing$（其中 $Missing(x)\colon = N_G(map(x)) \setminus \{map(c)\colon\ c \in children_T(x)\}$）。考虑具有 $a \geq d + missing$ 的 $a \in \mathbb{N}$，并令 $\ell_T\colon\ V(T) \mapsto [L] \cup \{\infty\}$ 为调用 $PartialLayerAssignmentTree(G, T, map, a, L)$ 所得到的层赋值。那么，对于每个相对于 ℓ_G 严格单调可达的 $x \in V(T)$，都有 $\ell_T(x) \leq \ell_G(map(x))$。

证明。我们通过对 $\ell_G(\mathrm{map}(x))$ 进行归纳法来证明以下命题：

对于每个 $x \in V(T)$，相对于 ℓ_G，$\ell_T(x) \leq \ell_G\big(\mathrm{map}(x)\big)$ 严格单调可达 .

基础情形： $\ell_G(\mathrm{map}(x)) = 1$. 设 $x \in V(T)$ 相对于ℓ_G 严格单调可达，且 $\ell_G(\mathrm{map}(x)) = 1$。我们必须证明 $\ell_T(x) = 1$。根据 V_1 在 PartialLayerAs-signment 中的定义，只需证明

$$\big|\mathrm{children}_T(x)\big| + \big|\mathtt{Missing}(x)\big| \leq a,$$

其中

$$\mathtt{Missing}(x) := N_G\big(\mathrm{map}(x)\big) \setminus \{\mathrm{map}(c) : c \in \mathrm{children}_T(x)\}.$$

由于 x 严格单调可达，引理的假设意味着 $\big|\mathrm{Missing}(x)\big| \leq$ 缺失。因为 map 是有效映射，所以

$$|children_T(x)| \leq |N_G(map(x))|.$$

此外，根据 $\ell_G(\mathrm{map}(x)) = 1$ 以及 ℓ_G 的性质，我们有：

$$\big|N_G(map(x))\big| = \big|\{\, u \in N_G(\mathrm{map}(x)) : \ell_G(u) \geq 1\}\big| \leq d.$$

因此， $\big|$子节点$_T(x)\big| \le \big|\{\, u \in N_G(\mathsf{map}(x)) :\ \ell_G(u) \ge 1\}\big| \le d.$

结合这两个界，并利用 $a \ge$ 出度 $+$ 缺失，我们得到

$$\big|\mathrm{children}_T(x)\big| \ + \ \big|\mathtt{Missing}(x)\big| \ \le \ d + missing \ \le \ a.$$

因此 $x \in V_1$，这意味着 $\ell_T(x) = 1$，如所需。

归纳步骤。令 $j \in [L-1]$。假设对于所有严格单调可达的 $y \in V(T)$，且 $\ell_G(\mathsf{map}(y)) \le j$，我们已经知道 $\ell_T(y) \le \ell_G(\mathsf{map}(y))$。

现在考虑一个节点 $x \in V(T)$，它

$$\text{strictly monotonically reachable and and} \quad \ell_G\big(\mathrm{map}(x)\big) \ = \ j+1.$$

We 要证明 $\ell_T(x) \le j+1$。根据 V_{j+1} 的定义，只需验证 :

$$\big|\mathrm{children}_T(x) \cap V_{\ge j+1}\big| \ + \ \big|\mathtt{Missing}(x)\big| \ \le \ a.$$

首先， x 严格单调可达意味着 $\big|$Missing$(x)\big| \le$ 缺失。因此，根据 $a \ge d +$缺失，只需证明

$$\big|\mathrm{children}_T(x) \cap V_{\ge j+1}\big| \ \le \ d.$$

回忆 $V_{\ge j+1} = V(T)\setminus\big(V_1 \cup \cdots \cup V_j\big)$，因此若 $c \in$子节点$_T(x) \cap V_{\ge j+1}$，则 $\ell_T(c) \ge j+1$。根据映射的有效性，$\mathsf{map}(c) \in N_G(\mathsf{map}(x))$。此外，若 $\ell_T(c) \ge j+1$，但 $\ell_G(\mathsf{map}(c))$ 是 $< j+1$，那么 c 将是严格单调可达的，而归纳假设将推出 $\ell_T(c) \le j$，这就产生了矛盾。因此，我们必有 $\ell_G(\mathsf{map}(c)) \ge j+1$。由此，每个 $c \in$子节点$_T(x) \cap V_{\ge j+1}$都对应于 G 中 $\mathsf{map}(x)$ 的一个邻居，其 ℓ_G-值至少为 $j+1$。

因为 $\ell_G(\mathsf{map}(x)) = j+1$，并且对于所有满足 $\ell_G(v) \ne \infty$ 的 $v \in V(G)$，

$$\big|\{\, u \in N_G(\mathrm{map}(x)) : \ell_G(u) \ge \ell_G(\mathrm{map}(x))\}\big| \ \le \ d,$$

由此立即可得

$$\big|\mathrm{children}_T(x) \cap V_{\ge j+1}\big| \ \le \ d.$$

Thus

$$\big|\mathrm{children}_T(x) \cap V_{\ge j+1}\big| \ + \ \big|\mathtt{Missing}(x)\big| \ \le \ d + missing \ \le \ a.$$

因此，若 $x \in V_{\ge j+1}$，则算法会将 x 放置在 V_{j+1} 中，从而 $\ell_T(x) = j+1$。归纳步骤至此完成。

□

引理 3.9. 设 G 是一个图， d、 $L \in \mathbb{N}$ 和 $\ell_G\colon V(G) \mapsto [L] \cup \{\infty\}$，且对于每个 $v \in V(G)$，若 $\ell_G(v) \ne \infty$，则 $|\{u \in N_G(v)\colon \ell_G(u) \ge \ell_G(v)\}| \le d$。

此外，设 B、 k、 $s \in \mathbb{N}_0$ 为满足 $k \ge d$ 和 $s > \log_2(L)$ 的参数。

设 $v \in V(G)$ 满足 $\ell_G(v) \ne \infty$ 且 $NumPathsIn_{G,\ell_G}(v) \le \sqrt{B}$.

设 $T_v^{(s)}$ 和 $map_v^{(s)}$ 分别为通过调用所得到的有根树和映射

$ExponentiateAndLocalPrune(G, B, k, s)$，并令 $\ell_v\colon V(T(s)_v) \mapsto [L] \cup \{\infty\}$ 为调用 $PartialLayerAssignmentTree(G, T_v^{(s)}, map_v^{(s)}, a := k \cdot (s+1)\ L)$ 后得到的层分配。于是，若 r_v 表示 $T_v^{(s)}$ 的根，则有 $\ell_v(r_v) \le \ell_G(v)$。

证明。引理 3.7 表明，每个 $x \in V(T_v^{(s)}$相对于 ℓ_G 严格单调可达的都满足

$|Missing_{G,T_v^{(s)},map_v^{(s)}}(x)| \leq s \cdot k$。因此，我们可以使用引理 3.8，结合 $missing := s \cdot k$和$a := (s+1) \cdot k$，得出对于每个相对于 ℓ_G 严格单调可达的 $x \in V(T_v^{(s)})$，都有 $\ell_v(x) \leq \ell_G(map(x))$。由于 $T_v^{(s)}$ 的根 r_v 相对于 ℓ_G 严格单调可达，这意味着 $\ell_v(r_v) \leq \ell_G(v)$。

□

引理 3.10。设 G 是一个图，T 是一棵有根树，$map\colon\ V(T) \mapsto V(G)$ 是一个有效的 映射，并且a, $L \in \mathbb{N}$。设 $\ell_T\colon\ V(T) \mapsto [L] \cup \{\infty\}$ 是通过以下方式计算出的层赋值

$PartialLayerAssignmentTree(G, T, map, a, L)$，并将 $\ell\colon V(G) \mapsto [L] \cup \{\infty\}$ 定义为 $\ell(v) = \min\{\ell_T(x)\colon x \in V(T) \text{ and } map(x) = v\}$；若没有节点映射到 v，则采用 $\ell(v) = \infty$ 的特殊情况定义。于是，对于每个满足 $\ell(v) \neq \infty$ 的 $v \in V(G)$，有：

$$|\{u \in N_G(v)\colon \ell(u) \geq \ell(v)\}| \leq a.$$

证明。任取 $v \in V(G)$ 和 $j \in \mathbb{N}$，使得 $\ell(v) = j$。根据 $\ell(v)$ 的定义，存在某个节点 $x \in V(T)$，使得 $map(x) = v$ 且 $\ell_T(x) = j$。特别地，有 $x \in V_j$，因此

$$|children_T(x) \cap V_{\geq j}| + |Missing(x)| \leq a.$$

令 $u \in N_G(v)$ 满足 $\ell(u) \geq j$。首先，考虑存在某个子节点 $c \in children_T(x)$ 满足 $map(c) = u$ 的情形。此时，$\ell_T(c) \geq \ell(u) \geq j$，因此 $c \in children_T(x) \cap V_{\geq j}$。否则，如果不存在满足 $map(c) = u$ 的子节点 $c \in children_T(x)$，那么 $u \in Missing(x)$。因此，

$$|\{u \in N_G(v)\colon \ell(u) \geq j\}| \leq |children_T(x) \cap V_{\geq j}| + |Missing(x)| \leq a.$$

□

算法 4： PartialLayerAssignment

输入： 图 G，非负整数 B，k，L，$s \in \mathbb{N}$

输出： 一个层赋值 $\ell\colon\ V(G) \mapsto [L] \cup \{\infty\}$

$(T_v^{(s)},\ map_v^{(s)})_{v \in V(G)} \leftarrow ExponentiateAndLocalPrune(G,\ B,\ k,\ s)$

对每个 $v \in V(G)$ **执行**

  $\ell_v\colon\ V(T_{(s)v}) \mapsto [L] \cup \{\infty\} \leftarrow PartialLayerAssignmentTree(G,\ T_{(s)v},\ map_v^{(s)},\ (s+1) \cdot k,\ L)$

end

$\forall u \in V(G),\ \ell(u) \leftarrow$ 最小值 $_{v \in V(G)\ x \in V(T_v^{(s)})\ map_v^{(s)}(x) = u} \ell_v(x)$

返回 ℓ

命题 3.11。算法 *4* 可以在 $O(s)$ *MPC* 轮次中实现，使用 $O(n^\delta + B)$ 字的本地内存和 $O(nB + m)$ 字的全局内存，其中 n 是顶点数的上界，m 是图 G 的边数上界。

证明。该断言可直接由命题3.5 以及使用先前工作中提出的标准 MPC 原语进行的直接实现得出，例如 [[ASS+18]、E 节] 及其中的参考文献。

□

断言 3.12。 设 G 为一个图， B、k、L、$s \in \mathbb{N}$ 和 $\ell\colon\ V(G) \to [L] \cup \{\infty\}$ 为通过以下方式计算得到的部分层分配

$PartialLayerAssignment(G, B, k, L, s)$。那么，对于每个 $v \in V(G)$ 满足 $\ell(v) \neq \infty$ 的对象，都有

$$|\{u \in N_G(v)\colon \ell(u) \geq \ell(v)\}| \leq (s+1) \cdot k.$$

证明。 任取满足 $\ell(v) \neq \infty$ 的 $v \in V(G)$。根据 ℓ 的定义，存在某个顶点$v' \in V(G)$ 和一个节点 $x \in V(T_{v'}^{(s)})$，使得 $map_{v'}^{(s)}(x) = v$ 且 $\ell_{v'}(x) = \ell(v)$。令

$$\widetilde{\ell}_{v'}\colon V(G) \to [L] \cup \{\infty\}$$

定义如下：对每个 $u \in V(G)$，

$$\widetilde{\ell}_{v'}(u) \;=\; \begin{cases} \min\{\,\ell_{v'}(y)\colon y \in V(T_{v'}^{(s)}) \text{ with } map_{v'}^{(s)}(y) = u\,\}, & \text{if such } y \text{ exists,} \\ \infty, & \text{otherwise.} \end{cases}$$

根据整体赋值 ℓ 的定义，我们有

$$\ell(u) = \min\{\,\ell_w(y)\colon w \in V(G),\ y \in V(T_w^{(s)}),\ map_w^{(s)}(y) = u\,\} \;\leq\; \widetilde{\ell}_{v'}(u)$$

~对于每个 $u \in V(G)$。因此，对于任意满足 $\ell(u) \geq \ell(v) = \ell_{v'}(x)$ 的 $u \in N_G(v)$，我们得到 $\ell_{v'}(u) \geq \ell_{v'}(x)$。因此，

$$\{\,u \in N_G(v) : \ell(u) \geq \ell(v)\,\} \;\subseteq\; \{\,u \in N_G(v) : \widetilde{\ell}_{v'}(u) \geq \ell_{v'}(x)\,\}.$$

现在，根据调用 PartialLayerAssignmentTree 中的假设以及引理3.10（应用于树 $T_{v'}^{(s)}$ 及其有效映射 $map_{v'}^{(s)}$），我们知道

$$\left|\{\,u \in N_G(v) : \widetilde{\ell}_{v'}(u) \geq \ell_{v'}(x)\,\}\right| \;\leq\; (s+1) \cdot k.$$

由此可得

$$\left|\{\,u \in N_G(v) : \ell(u) \geq \ell(v)\,\}\right| \;\leq\; (s+1) \cdot k.$$

这就完成了该结论的证明。 □

引理 3.13。 设 G 是一个图，且 B、$k \in \mathbb{N}$ 满足 $k^{100} \leq B \leq n^{\delta/100}$ 和 $k \geq 100 \cdot \lambda(G)$，其中 n 和 m 分别是 G的顶点数和边数的上界。那么，存在一个 ***MPC*** 算法，可在 $O(\log\log n)$ 轮次内运行，使用 $O(n^{\delta})$ 本地内存和$O(nB+m)$ 字的全局内存，并计算出一个部分层分配 $\ell\colon\ V(G) \to \{1, 2, \ldots, \lceil 0.1\log_k(B)\rceil\} \cup \{\infty\}$，并满足以下性质：

1. ℓ 的出度至多为 $\lceil 100\log\log(n)\rceil \cdot k$。
2. *For every* $j \in \{1, 2, \ldots, \lceil 0.1\log_k(B)\rceil\}$, $\left|\{v \in V(G) : \ell(v) \geq j\}\right| \leq 0.5^{j-1} \cdot |V(G)|$.

证明。 我们首先执行算法

$$\textsc{PartialLayerAssignment}(B, k, L, s)$$

参数为 $L := \lceil 0.1\log_k(B)\rceil$，$s := \lceil 10\log\log(n)\rceil$. 令 ℓ 表示由此得到的部分层分配。于是，根据 断言 3.12， ℓ的出度至多为

$$(s+1) \cdot k = \big(\lceil 10\log\log(n)\rceil + 1\big)k \leq \lceil 100\log\log(n)\rceil\, k,$$

这就证明了性质 (1)。

剩下要证明的是，对每个 $j \in \{1, 2, \ldots, L\}$，我们都有

$$\left|\{v \in V(G) : \ell(v) \geq j\}\right| \leq 0.5^{j-1} \cdot |V(G)|.$$

为此，定义如下辅助层分配 ℓ_G: $V(G) \to \{1, 2, \ldots, L\} \cup \{\infty\}$。在每次 L 迭代中，从当前图中移除所有度至多为 k 的顶点，并将这些顶点被移除时所在迭代的编号赋给每个被移除的顶点。直接计算可知，利用假设 $k \geq 100 \cdot \lambda(G)$，对每个 $j \in \{1, \ldots, L\}$，我们都有

$$\left|\{v \in V(G) : \ell_G(v) \geq j\}\right| \leq 0.1^{j-1} \cdot |V(G)|.$$

现在，任取 $j \in \{2, 3, \ldots, L\}$。由 引理 3.9 结合 引理 2.4 可得

$$\begin{aligned}
\left|\{v \in V(G) : \ell(v) \geq j\}\right| &\leq \left|\{v \in V(G) : \ell_G(v) \geq j\}\right| \left|\left\{v \in V(G) : \mathrm{NumPathsIn}_{G,\ell_G}(v) \geq \sqrt{B}\right\}\right| \\
&\leq 0.1^{j-1} \cdot |V(G)| + \frac{k^L}{\sqrt{B}} \cdot |V(G)| \\
&\leq 0.5^{j-1} \cdot |V(G)|.
\end{aligned}$$

□

引理 3.14。设 G 是一个图，且 B、$k \in \mathbb{N}$ 满足 $k^{100} \leq B \leq n^{\delta/100}$ 和 $k \geq 100 \cdot \lambda(G)$，其中 n 和 m 分别是 G 的顶点数和边数的上界。则存在一个 *MPC* 算法，在 $O(\log(k) \log\log(n))$ 轮次内运行，使用 $O(n^\delta)$ 本地内存和$O(nB + m)$ 字全局内存，并计算一个部分层分配 ℓ: $V(G) \to \{1, 2, \ldots, \lceil 100 \log(B) \rceil\} \cup \{\infty\}$，且满足以下性质：

1. ℓ 的出度至多为 $\lceil 100 \log\log(n) \rceil \cdot k$。
2. *For every* $j \in \{1, 2, \ldots, \lceil 100 \log(B) \rceil$, $\left|\{v \in V(G) : \ell(v) \geq j\}\right| \leq 0.5^{j-1} \cdot |V(G)|$.

证明。主要思路是在未分配的顶点上迭代应用引理 3.13 中的算法。现在我们详细描述这一过程。

设置。令

$$T = \left\lceil \frac{\lceil 100 \log(B) \rceil}{\lceil 0.1 \log_k(B) \rceil} \right\rceil = O(\log k)$$

为我们将执行的迭代次数。我们按轮次进行，轮次编号为 $i = 1, \ldots, T$。在第 i 轮中，我们考虑由仍未分配（即当前分配为 ∞）的顶点所诱导的子图。将这些顶点记为

$$U_i = \{v \in V(G) : \ell(v) = \infty\},$$

约定 $U_1 = V(G)$。

迭代。在第 i 轮中，我们在子图 $G[U_i]$ 上运行引理 3.13 中的 MPC 算法 $G[U_i]$，参数为 B 和 k。这将产生一个部分赋值

$$\ell_i \colon U_i \to \{1, 2, \ldots, \lceil 0.1 \log_k(B) \rceil\} \cup \{\infty\}$$

满足：

ℓ_i 的出度至多为 $\lceil 100\log\log(n)\rceil \cdot k$。

2. 对于每个 $j \in \{1, 2, \ldots, \lceil 0.1\log_k(B)\rceil\}$, $\left|\{v \in U_i\colon \ell_i(v) \geq j\}\right| \leq 0.5^{j-1}|U_i|$.

随后，我们通过如下方式为每个在第 i 轮获得有限层的顶点 $v \in U_i$ 指定一个最终层

$$\ell(v) \;=\; (i-1)\cdot\lceil 0.1\log_k(B)\rceil \;+\; \ell_i(v).$$

（假设 $(i-1)\cdot\lceil 0.1\log_k(B)\rceil + \ell_i(v) \leq \lceil 100\log(B)\rceil$）。对于满足 $\ell_i(v) = \infty$ 的顶点，不为其分配层，并将其留待下一轮处理（即它们属于 U_{i+1}）。

出度。 每一轮都会产生一个分配方案 ℓ_i，其出度至多为 $\lceil 100\log\log(n)\rceil\cdot k$，因此最终分配方案的出度也至多为 $\lceil 100\log\log(n)\rceil \cdot k$。

衰减性质。 固定一个整数 j，使得 $1 \leq j \leq \lceil 100\log(B)\rceil$。记

$$j \;=\; (i-1)\cdot\lceil 0.1\log_k(B)\rceil + j',$$

其中 $1 \leq j' \leq \lceil 0.1\log_k(B)\rceil$，且 $i \in \{1, 2, \ldots, T\}$。在第 i 轮，根据 ℓ_i 的衰减性质，U_i 中至多有比例为 $0.5^{j'-1}$ 的顶点获得至少为 j' 的值。此外，将未分配的顶点从第 $i-1$ 轮带到第 i 轮的过程保证

$$|U_i| \;\leq\; 0.5^{(i-1)\cdot\lceil 0.1\log_k(B)\rceil}\,|V(G)|.$$

因此，最终层为 v 的顶点数量

$$\ell(v) \geq j \text{ (i.e., those with either } v \in U_i \text{ or with } \ell_i(v) \geq j' \text{ in round } i),$$

至多为

$$0.5^{j'-1}\cdot 0.5^{(i-1)\cdot\lceil 0.1\log_k(B)\rceil}\cdot|V(G)| \;\leq\; 0.5^{j-1}|V(G)|.$$

轮复杂度与内存。 由于 引理 3.13中的算法每次调用耗时 $O(\log\log n)$ 轮，且我们依次执行 $T = O(\log k)$ 轮，因此整体轮复杂度为

$$O\big(\log(k)\log\log n\big).$$

本地内存和全局内存的界仍分别为 $O(n^\delta)$ 和 $\widetilde{O}(nB+m)$，与 引理 3.13 中相同。

因此，最终的部分层分配

$$\ell\colon V(G) \to \{1, 2, \ldots, \lceil 100\log(B)\rceil\} \cup \{\infty\}$$

满足所需的出度和衰减性质，并且 MPC 算法在 $O(\log(k)\log\log n)$ 轮次内运行，且满足给定的内存界。 □

引理 3.15。 设 G 是一个图，并设 $k \in \mathbb{N}$ 满足 $k \geq 100\cdot\lambda(G)$，其中 n 和 m 分别是 G的顶点数和边数的上界。则存在一种 *MPC* 算法，可在 $O(\log(k)\log^2\log(n))$ 轮次内运行，使用 $O(n^\delta)$ 本地内存和 $O(n+m)$ 字全局内存，并计算一个完整层赋值 $\ell\colon V(G) \to \mathbb{N}$，具有以下性质：

1. ℓ 的出度至多为 $\lceil 100\log\log(n)\rceil \cdot k$。

2. 对于每个 $j \in \mathbb{N}$, $\left|\{v \in V(G) : \ell(v) \geq j\}\right| \leq 0.5^{j-1} \cdot |V(G)|$.

证明。 完整分层赋值首先通过"剥离"图来减少顶点数量，然后迭代应用引理 3.14 中的部分赋值算法，并辅以增强过程。我们将这一过程分为三个阶段进行描述。

阶段 1：初始剥离。 为了最终能够以较大的每顶点预算运行部分赋值算法，我们首先对G 执行 $\lceil 100\log(k)\rceil$ 轮简单的剥离过程，具体如下。在每一轮剥离中，从当前图中移除所有度至多为k 的顶点。标准论证表明，每轮至少会移除一半的顶点。因此，经过 $\lceil 100\log(k)\rceil$ 轮后，剩余顶点集合 U 满足

$$|U| \leq \frac{n}{k^{100}}.$$

因此，在子图 $G[U]$ 中，有效的每顶点预算至少为 $B_0 = k^{100}$。

阶段 2：通过预算增强进行部分赋值。 现在我们在子图$G[U]$ (其中 $U \subseteq V(G)$)上操作，并迭代地为其中的顶点分配层。在每个阶段 i 中，我们维护一个当前集合 U_i，其中包含尚未分配有限层的顶点，以及一个预算 B_i （初始时，$U_0 = U$ 和 $B_0 = k^{100}$）。在阶段 i中，我们在 $G[U_i]$ 上运行引理 3.14 中的 MPC 算法，参数为 B_i 和 k。这将生成一个部分层分配

$$\ell_i \colon U_i \to \{1, 2, \ldots, \lceil 100\log(B_i)\rceil\} \cup \{\infty\},$$

其满足：

1. 出度至多为 $\lceil 100\log\log(n)\rceil \cdot k$。
2. 对每个 $j \in \{1, \ldots, \lceil 100\log(B_i)\rceil\}$,

$$\left|\{v \in U_i : \ell_i(v) \geq j\}\right| \leq 0.5^{j-1}|U_i|.$$

特别地，令 $j = \lceil 100\log(B_i)\rceil$，我们可以推得在阶段 i之后仍未分配的顶点数为

$$|U_{i+1}| \leq \frac{|U_i|}{B_i^{100}}.$$

对于每个顶点 $v \in U_i$，若其被分配到有限层（即 $\ell_i(v) \neq \infty$），则定义其最终层为

$$\ell(v) = L_i + \ell_i(v),$$

其中， L_i 是选取的偏移量，用于确保在不同阶段分配的层互不重叠。例如，可以设为

$$L_i = \lceil 100\log(k)\rceil + \sum_{t=0}^{i-1} \lceil 100\log(B_t)\rceil.$$

在阶段 i之后，我们通过提高 预算，令

$$B_{i+1} = \min(B_i^{100}, n^{\delta}).$$

经过 $O(\log\log n)$ 个阶段后，我们得到 $U_T = \emptyset$，因此 U 中（进而 $V(G)$ 中）的每个顶点都已被赋予一个有限层。

阶段 3。复杂度与验证。在阶段 1 中，剥离过程需要 $O(\log k)$ 轮次。在阶段 2 中，每个阶段都会运行引理 3.14 中的部分赋值算法，耗时 引理 3.14 在$O(\log(k)\log\log(n))$ 轮次。由于共有 $T = O(\log\log n)$ 个阶段，总轮次复杂度为

$$O\Big(\log k + \log\log(n)\cdot\log(k)\log\log(n)\Big) = O\big(\log(k)\log^2\log(n)\big).$$

此外，每个阶段所需的本地内存为 $O(n^\delta)$，使用的全局内存为 $O(|U_i|B_i + m) = O(n+m)$ 字。最后，最终的完整分层赋值 $\ell\colon V(G) \to \mathbb{N}$继承了部分赋值的以下性质：

1. 出度至多为 $\lceil 100\log\log(n)\rceil \cdot k$。
2. 对于每个 $j \in \mathbb{N}$，通过对各阶段作简单归纳可得

$$\Big|\{v \in V(G) : \ell(v) \geq j\}\Big| \leq 0.5^{j-1}\cdot|V(G)|.$$

至此，证明完成。 □

定理 1.1. 存在一种随机化可扩展 *MPC* 算法，对于任意无向图$G = (V, E)$，若其具有 $n = |V|$ 和 $m = |E|$，该算法可在$\mathrm{poly}(\log\log n)$ 轮次内运行，并计算出一种边定向，使得每个节点的出度至多为 $O(\lambda\log\log n)$，且以高概率成立。这里，λ 表示该图的森林度。该算法使用每台机器 n^δ 内存，其中 $\delta \in (0,1)$ ~为任意正数常量，全局内存为 $O(m+n)$ 字。若 λ 有上界$(\log n)^{O(\log\log n)}$，则该算法为确定性算法。

证明。利用全局内存中额外的 $O(\log(n))$ 因子，我们可以假设给定 k，且满足 $k \in [100\lambda(G)\ 200\lambda(G)]$。首先，考虑 $k = \log(n)^{O(\log\log n)}$ 的情形。在该情形下，我们可以使用引理 3.14，在 $\mathrm{poly}(\log\log n)$ 轮次内计算出一个完整分层 ℓ，其出度为 $O(k\log\log n)$。然后，通过将边从较低层定向到较高层，并任意打破平局，我们可以得到所需的定向，其出度为 $O(\lambda(G)\log\log n)$。若 $k \geq \log(n)^{O(\log\log n)}$，我们首先计算一个随机边划分，将其划分为 $P = \Theta(k/log(n))$ 个部分 $E_1, E_2, \ldots, E_P$。引理 2.1 表明，图 $G_i = (V(G)\ E_i)$ 的森林度为 $O(\log n)$。因此，我们可以使用引理 3.14，在 $\mathrm{poly}(\log n)$ 轮次内将 E_i 中的边定向为出度 $O(\log n\log\log n)$。然后，将这 P 个边定向组合起来，即可得到 G 的一个边定向，其出度为 $O(\lambda(G)\log\log n)$。 □

4 着色算法

定理 1.2. 存在一种随机化可扩展 *MPC* 算法，对于任意无向图$G = (V, E)$，满足 $n = |V|$ 和 $m = |E|$，可在 $\mathrm{poly}(\log\log n)$ 轮次内以高概率计算出一个包含 $O(\lambda\log\log n)$ 种颜色的顶点着色。这里，λ 表示图的森林度。该算法使用每台机器 n^δ 内存，其中 $\delta > 0$ 是任意正数常量，~并且使用 $O(m+n)$ 字的全局内存。

证明. 我们分几步给出证明。通过在全局内存中额外使用一个 $O(\log(n))$ 因子，我们可以假设给定的 k 满足 $k \in [100\lambda(G)\ 200\lambda(G)]$。

如有需要，进行随机划分。首先，我们将问题转化为对森林度至多为 $O(\log n)$ 的图进行着色：如果 $k \leq O(\log n)$，则无需进行任何处理。否则，我们采用随机顶点划分，将顶点划分为 $P = \Theta(k/\log n)$ 个部分 V_1，V_2、……、V_P。根据

引理2.2，我们知道，以高概率，每个部分所诱导的子图的森林度$O(\log n)$。随后，我们将分别解决每个部分的着色问题，并为每个部分使用大小为 $O((k/P)\log\log n$ 的不同调色板，由此得到整个图的 $O(\lambda \log\log n$ 着色。基于此，在本证明的其余部分，我们将专注于对森林度至多为$O(\log n \log\log n)$ 的图进行着色。

分层与定向。 我们应用 引理 3.15 来计算 $L = \Theta(\log n)$-分层划分$V = H_1 \sqcup H_2 \ldots \sqcup H_L$，其出度至多为 $d = O(\lambda \log\log n)$，并满足以下性质：$\cup_{i=j}^{L}|H_i| \leq 2^{-j+1} \cdot n$ 对于每个 $j \in [L]$。这至多需要 $O(\log^3 \log n)$ 轮次的 MPC。我们使用 $3d$ 种颜色对该图进行着色。

LOCAL 模型着色。给定已计算出的分层，从 LOCAL 模型的角度来看，我们可以在 $\Theta(\log n)$ 个阶段内为该图着色，其中在阶段 i，任务是为 H_{L-i+1} 中的顶点计算一种着色，使每个节点 v 都避开其位于更高层的邻居所使用的颜色。这本质上是图 H_i 所诱导的一个 $degree + 1$ 列表着色问题（事实上，可用颜色的数量至少为 $2d$，其中 $d = \Theta(\lambda \log\log n)$）。该方案从为最后一层 H_L 着色开始，逐层进行，最后为 H_1 中的节点着色。得益于最先进的 LOCAL 模型算法 [HKNT22,GG24b]，每个阶段需要 $O(\log^{1.67}\log n)$ 轮 LOCAL。于是，整个算法将需要 $O(\log n \cdot \log^{1.67}\log n)$ 轮次。

在 MPC 中进行更快的模拟。我们使用有向图指数化，在 MPC 中更快地模拟上述 LOCAL 算法：假设每一层内的边都复制为双向边，而跨层的边则指向较高层的端点。此外，假设我们已经确定了 $H_j \sqcup H_{j+1} \sqcup \ldots \sqcup H_L$ 中每个节点的颜色。现在，我们希望确定 $H_{j'} \sqcup H_{j'+1} \sqcup \ldots \sqcup H_{j-1}$ 中每个节点的颜色，其中 j' 为某个值。为此，只需让每个节点 $v \in H_{j'} \sqcup H_{j'+1} \sqcup \ldots \sqcup H_{j-1}$ 学会所有从 v 出发、沿有向路径且距离至多为 $O\left((j-j')\log^{1.67}\log(n)\right)$ 的可达节点，以及这些节点在属于 $H_j \sqcup H_{j+1} \sqcup \ldots \sqcup H_L$ 层时的颜色。利用 $d = O(\lambda \log\log n)$ 和 $\lambda = O(\log n)$，每个节点 $v \in H_{j'} \sqcup H_{j'+1} \sqcup \ldots \sqcup H_{j-1}$ 至多需要学会 $d^{O((j-j')\log^{1.67}\log(n))} = 2^{O((j-j')\log^{2.67}\log(n))}$ 个节点。特别是，只要 $j - j' = \Theta(\delta \log(n)/\log^{2.67}\log(n))$，且常数选得足够小，每个节点 v 就只需学会 $O(n^{\delta})$ 个节点，因此满足局部内存约束。为了满足全局内存约束，只需确保

$$\frac{n}{2^{j'}} \cdot 2^{O\left((j-j')\log^{2.67}\log(n)\right)} = O(n),$$

作为 $\cup_{i=j'}^{L}|H_i| \leq 2^{-j'+1} \cdot n$。只要满足以下条件，这一约束就成立

$$j - j' = O\left(\frac{\delta \cdot j}{\log^{2.67}\log(n)}\right).$$

因此，通过沿出边进行有向图指数化（参见 [LU21，定义 3。3]关于更低层次的细节，或参见 引理 4.1 中的概述），我们可以在 $O(\log\log n)$ 轮内，计算尚未求出颜色的层中 $\Theta\left(\frac{1}{(\log^{2.67}\log n)}\right)$比例的层的颜色，前提是$j \geq \Theta(\log^{2.67}\log n)$。由于总共有 $O(\log n)$ 层，我们需要重复此过程 $O(\log^{3.67}\log n)$ 次，这需要 $O(\log^{4.67}\log n)$ 个 MPC 轮次。一旦 $j \leq \Theta(\log^{2.67}\log n)$，我们只需逐阶段运行 LOCAL 模型算法，不进行加速。这还需要额外的 $\Theta(\log^{2.67}\log n)$. $\Theta\left(n^{1.67}\right) = \Theta\left(n^{4.34}\right)$ (log log) (log log) 轮。因此，完整的轮复杂度为 $O(\log^{4.67}\log n)$。

□

引理 4.1 (有向指数扩展)。 对于任意常数 $\delta > 0$，有如下结论：假设每个节点 v 都有一个包含 $b_v \leq n^{\delta/2}$ 个字的信息包 B_v，其他节点可能希望获取这些信息；并且每个节点 u 都希望接收列表 L_u 中各节点的信息包。再假设 *(A)* $|L_u| \leq n^{\delta/2}$ 且 *(B)* $\sum_{u \in V} \sum_{v \in L_u} |b_v| = O(m+n)$。那么，存在一种 *MPC* 算法，使用 n^{δ} 本地内存和 $O(m+n)$ 全局内存，可在 $O(1)$ 轮内完成此任务。

证明概要。首先，我们让每个节点 v 了解有多少节点 u 拥有 $v \in L_u$。为此，我们应用一个常数轮次排序算法： v 启动两个项目 $(v.ID, -\infty)$ 和 $(v.ID, +\infty)$，每个拥有 $v \in L_u$ 的节点 u 启动一个项目 $(v.ID, u)$。然后，我们使用任意标准的常数轮次排序算法 [KSV10b、IKL+23、Gha]，按字典序对所有这些项目进行排序。持有 v 的机器将知道项目 $(v.ID, -\infty)$ 和 $(v.ID, +\infty)$ 的排名差，由此得知有多少节点 u 拥有 $v \in L_u$。将这个数量记为 k_v。

其次，对于每个 v，我们生成 k_v 份信息包 B_v副本，编号从 1 到 k_v。我们在常数轮次内同时对所有 v执行此操作，为每个 B_v 使用一棵标准广播树，每轮将其副本数量从 1 增加到 k_v，增加倍数为 $n^{\delta/2}$。例如参见 [Gha，第 1.3 节。2]。

第三，我们对两类内容进行排序：(1) 每个节点 u 和每个节点 $v \in L_u$ 的条目列表 (u, v)，用于表明 u 希望接收 B_v 的副本。(2) 上文生成的 B_v 的 i^{th} 副本对应的条目列表 (v, i)。然后，我们只需进行一次匹配：在第一个列表中排名为 j 的节点 u 联系第二个列表中排名为 j 的机器，并从该机器接收相应的 B_v 副本。

□

参考文献

[ACK19] Sepehr Assadi、Yu Chen 和 Sanjeev Khanna。顶点着色的次线性算法。在第三十届 *ACM-SIAM* 离散算法年会论文集，第 767–786 页。SIAM，2019 年。[ANOY14] Alexandr Andoni、Aleksandar Nikolov、Krzysztof Onak 和 Grigory Yaroslavtsev。几何图问题的并行算法。在第 *46* 届 *ACM* 计算理论研讨会（*STOC*）论文集，第 574–583 页，2014 年。[ASS+18] Alexandr Andoni、Zhao Song、Clifford Stein、Zhengyu Wang 和 Peilin Zhong。对数直径轮次内的并行图连通性。在*2018* 年 *IEEE* 第 *59* 届计算机科学基础年会（*FOCS*），第 674–685 页。IEEE，2018 年。[BBD+19] Soheil Behnezhad、Sebastian Brandt、Mahsa Derakhshan、Manuela Fischer、Moham- madTaghi Hajiaghayi、Richard M. Karp 和 Jara Uitto。在稀疏图上大规模并行计算匹配和极大独立集。在第 *38* 届 *ACM* 分布式计算原理研讨会（*PODC*）论文集，第 481–490 页，2019 年。[BDE+19] Soheil Behnezhad、Laxman Dhulipala、Hossein Esfandiari、Jakub Lacki 和 Vahab Mirrokni。近乎最优的大规模并行图连通性。在*2019* 年 *IEEE* 第 *60* 届计算机科学基础年会（*FOCS*），第 1615–1636 页。IEEE，2019 年。[BE08] Leonid Barenboim 和 Michael Elkin。利用 Nash-Williams 分解求解稀疏图的次对数分布式极大独立集算法。在国际分布式计算原理研讨会（*PODC*）论文集，第 25–34 页，2008 年。

[BE13] L. Barenboim 和 M. Elkin。《分布式图着色：基础与最新进展》。Morgan & Claypool 出版社，2013 年。[BKS17] Paul Beame、Paraschos Koutris 和 Dan Suciu。并行查询处理的通信步骤。《ACM学报》，64(6):1–58，2017 年。[BLM+23] Alkida Balliu、Rustam Latypov、Yannic Maus、Dennis Olivetti 和 Jara Uitto。森林上的最优确定性大规模并行连通性。在 2023 年 ACM-SIAM 离散算法研讨会（SODA）论文集，第 2589–2631 页。SIAM，2023 年。[CC22] Sam Coy 和 Artur Czumaj。确定性大规模并行连通性。在第 54 届 ACM SIGACT 计算理论研讨会论文集，第 162–175 页，2022 年。[CDP21a] Artur Czumaj、Peter Davies 和 Merav Parter。用于以低空间去随机化大规模并行计算的图稀疏化。《ACM算法汇刊》（TALG），17(2):1–27，2021 年。[CDP21b] Artur Czumaj、Peter Davies 和 Merav Parter。低空间 MPC 中改进的确定性着色。在 2021 年 ACM 分布式计算原理研讨会论文集，第 469–479 页，2021 年。[CFG+19] Y.-J. Chang、M. Fischer、M. Ghaffari、J. Uitto 和 Y. Zheng。拥塞团、大规模并行计算和集中式局部计算中着色的复杂度。在 ACM 分布式计算原理研讨会（PODC），第 471–480 页，2019 年。[CLP18] Yi-Jun Chang、Wenzheng Li 和 Seth Pettie。一种最优的分布式着色算法？在 ACM计算理论研讨会（STOC），2018 年。[CNR23] Aleksander Bjørn Grodt Christiansen、Krzysztof Nowicki 和 Eva Rotenberg。稀疏图动态着色的改进。在第 55 届 ACM 计算理论研讨会论文集，第 1201–1214 页，2023 年。[DG08] Jeffrey Dean 和 Sanjay Ghemawat。MapReduce：简化大型集群上的数据处理。《ACM通讯》，51(1):107–113，2008 年。[FGG22] Manuela Fischer、Jeff Giliberti 和 Christoph Grunau。大规模并行计算中的改进确定性连通性。在第 36 届国际分布式计算研讨会，2022 年。[FGG23] Manuela Fischer、Jeff Giliberti 和 Christoph Grunau。稀疏图的确定性大规模并行对称性破缺。在第 35 届 ACM算法与体系结构并行性研讨会论文集，第 89–100 页，2023 年。[GG24a] Mohsen Ghaffari 和 Christoph Grunau。多对数最坏情况下时间内的动态 o（森林度）着色。在第 56 届 ACM 计算理论研讨会论文集，第 1184–1191 页，2024 年。[GG24b] Mohsen Ghaffari 和 Christoph Grunau。近乎最优的确定性网络分解与统治集，以及改进的极大独立集。在 2024 年 IEEE 第 65 届计算机科学基础年会（FOCS），第 2148–2179 页。IEEE，2024 年。

[GGJ20] Mohsen Ghaffari、Christoph Grunau 和 Ce Jin。《改进的 MPC 算法：适用于树及更广泛图类的极大独立集、匹配与着色》。发表于第34届国际分布式计算研讨会，2020年。[Gha] 大规模并行算法。https://people.csail.mit.edu/ghaffari/MPA19/ Notes/MPA.pdf。[GKU19] Mohsen Ghaffari、Fabian Kuhn 和 Jara Uitto。《基于分布式下界的大规模并行计算条件困难性结果》。发表于计算机科学基础研讨会（FOCS）论文集，2019年。[GL17] Mohsen Ghaffari 和 Christiana Lymouri。《面向稀疏图的简单且近乎最优的分布式着色》。发表于第31届国际分布式计算研讨会（DISC 2017）。达格斯图尔-莱布尼茨信息学中心，2017年。[GLM19] Mohsen Ghaffari、Silvio Lattanzi 和 Slobodan Mitrovi´c。《基于密度的网络聚类的改进并行算法》。发表于国际机器学习会议，第2201–2210页。PMLR，2019年。[GLM+23] Christoph Grunau、Rustam Latypov、Yannic Maus、Shreyas Pai 和 Jara Uitto。《森林的条件最优并行着色》。发表于国际分布式计算研讨会（DISC）论文集，第23页。达格斯图尔-莱布尼茨信息学中心，2023年。[GS17] Mohsen Ghaffari 和 Hsin-Hao Su。《分布式度数分裂、边着色与定向》。发表于第二十八届 ACM-SIAM 离散算法研讨会论文集，第2505–2523页。SIAM，2017年。[GSZ11] Michael T. Goodrich、Nodari Sitchinava 和 Qin Zhang。《MapReduce 框架中的排序、搜索与模拟》。发表于第22届国际算法与计算研讨会（ISAAC）论文集，第374–383页，2011年。[GU19] Mohsen Ghaffari 和 Jara Uitto。《稀疏化分布式算法及其在大规模并行计算和集中式局部计算中的影响》。发表于第30届 ACM-SIAM 离散算法研讨会（SODA）论文集，第1636–1653页，2019年。[HKNT22] Magn´us M Halld´orsson、Fabian Kuhn、Alexandre Nolin 和 Tigran Tonoyan。《近乎最优的分布式度着色》。发表于第54届 ACM SIGACT计算理论研讨会年会论文集，第450–463页，2022年。+ 1IBY[Michael Isard、Mihai Budiu、Yuan Yu、Andrew Birrell 和 Dennis Fetterly。《Dryad：由顺序构建模块组成的分布式数据并行程序》。发表于第2届 ACM SIGOPS/EuroSys欧洲计算机系统会议论文集，第59–72页，2007年。+07] IKL[Sungjin Im、Ravi Kumar、Silvio Lattanzi、Benjamin Moseley、Sergei Vassilvitskii 等。《大规模并行计算：算法与应用》。《优化领域基础与趋势》，5(4):340–417，2023年。+23] KSV10a[Howard Karloff、Siddharth Suri 和 Sergei Vassilvitskii。《MapReduce 的计算模型》。发表于第21届 ACM-SIAM 离散算法研讨会（SODA）论文集，第938–948页，2010年。

[KSV10b] Howard Karloff、Siddharth Suri 和 Sergei Vassilvitskii。《MapReduce 的计算模型》。发表于第二十一届 *ACM-SIAM* 离散算法研讨会论文集，第938–948页。SIAM，2010年。[Lin87] Nati Linial。《分布式图算法——利用局部数据求解全局问题》。发表于第*28*届 *IEEE* 计算机科学基础研讨会论文集（*FOCS*），第331–335页，1987年。[LMPU24] Rustam Latypov、Yannic Maus、Shreyas Pai 和 Jara Uitto。《稀疏图中的自适应大规模并行着色》。发表于第*43*届 *ACM*分布式计算原理研讨会论文集，第508–518页，2024年。[LU21] Rustam Latypov 和 Jara Uitto。《大规模并行计算中的树着色》。*arXiv* 预印本 *arXiv:2105.13980*，2021年。[LW08] C. Lenzen 和 R. Wattenhofer。《利用 Linial局部性极限》。发表于第*22*届分布式计算研讨会（*DISC*）论文集，第394–407页，2008年。[NW61] C St JA Nash-Williams。《有限图的边不相交生成树》。《伦敦数学学会杂志》，1(1):445–450，1961年。[NW64] C St JA Nash-Williams。《有限图的森林分解》。《伦敦数学学会杂志》，1(1):12–12，1964年。[Pel00] David Peleg。《分布式计算：一种局部性感知方法》。SIAM，2000年。[RG20] Vaclav Rozhoˇn 和 Mohsen Ghaffari。《多对数时间确定性网络分解与分布式去随机化》。发表于第*52*届 *ACM*计算理论研讨会（*STOC*）论文集，第350–363页，2020年。[SV20] Hsin-Hao Su 和 Hoa T Vu。《分布式稠密子图检测与低出度定向》。发表于第*34*届国际分布式计算研讨会，2020年。[Tut61] William Thomas Tutte。《将图分解为 n 个连通因子的问题》。《伦敦数学学会杂志》，1(1):221–230，1961年。[Whi12] Tom White。《*Hadoop*：权威指南》。“O’Reilly Media, Inc.”，2012年。[ZCF+10] Matei Zaharia、Mosharaf Chowdhury、Michael J. Franklin、Scott Shenker 和 Ion Sto- ica。《Spark：带工作集的集群计算》。发表于第*2*届 *USENIX*云计算热点主题会议论文集，2010年。