MPC 模型中的树上快速动态规划

Chetan Gupta · chetan.gupta@aalto.fi · 芬兰阿尔托大学**Rustam Latypov** · rustam.latypov@aalto.fi · 芬兰阿尔托大学**Yannic Maus** · yannic.maus@ist.tugraz.at · 奥地利格拉茨技术大学**Shreyas Pai** · shreyas.pai@aalto.fi · 芬兰阿尔托大学**Simo Särkkä** · simo.sarkka@aalto.fi · 芬兰阿尔托大学**Jan Studený** · jan.studeny@aalto.fi · 芬兰阿尔托大学**Jukka Suomela** · jukka.suomela@aalto.fi · 芬兰阿尔托大学**Jara Uitto** · jara.uitto@aalto.fi · 芬兰阿尔托大学**Hossein Vahidi** · hossein.vahidi@aalto.fi · 芬兰阿尔托大学

arXiv:2305.03693v1 [cs.DC] 5 May 2023

摘要。 我们提出了一种确定性算法，用于在大规模并行计算模型（MPC）中，在 $O(\log D)$ 轮次内求解树上的一大类动态规划问题；对于任意给定的常数 $0 < \delta < 1$，每台机器配备 $O(n^\delta)$ 个本地内存字。这里，D 是树的直径，n 是节点数——我们特别强调，我们的运行时间与 n 无关。

我们的算法可以解决许多经典的图优化问题，例如最大权独立集、最大权匹配、最小权支配集和最小权顶点覆盖。它还可以用于解决许多累积任务，即将某些汇总信息沿树向上或向下传播——例如，计算每个子树中输入标签的总和、最小值或最大值，以及许多通常用置信传播解决的推断任务。我们的算法还可以解决树上的任何局部可检验标记问题（LCL）。我们的算法适用于任何合理的输入树表示方式；例如，可以将树表示为边的列表，或表示为带有嵌套括号或标签的字符串。在广受认可的 2-环猜想成立的假设下，$O(\log D)$ 轮次的运行时间也已知是必要的。

我们的算法严格优于此前的两种算法：

1. Bateni、Behnezhad、Derakhshan、Hajiaghayi 和 Mirrokni [ICALP'18] 能在 $O(\log n)$ 轮次内解决这类问题，而我们的算法在低直径树上快得多。此外，他们的算法也使用随机性，而我们的算法是确定性的。 2. Balliu、Latypov、Maus、Olivetti 和 Uitto [SODA'23] 只能在 $O(\log D)$ 轮次内解决局部可检查的标签问题，而我们的算法可应用于范围广泛得多的问题。

1 引言

在本研究中，我们提出了一种通用且统一的算法框架，用于在大规模并行环境中解决与树结构数据 相关的各种计算问题。我们的算法能够解决的任务示例包括：

- 在树中求解传统的图优化问题（例如，求最大权独立集或最小权支配集）。
- 求解树上的约束满足问题（例如，求解任意局部可检验标记问题 [23], 以及该主题的许多推广问题）。
- 分析含有树结构数据的大型文本文档（例如，处理大型 XML [10]文档）。
- 在树中聚合信息（例如，计算每棵子树[15]中输入值的总和——这是将经典前缀和运算 [22] 从有向路径推广到有根树）。
- 在树结构图模型中执行统计推断（例如，在经典的顺序计算场景中通常使用置信传播完成的计算 [21]）。

1.1 设置：大规模并行计算模型

我们采用通常的大规模并行计算模型（MPC） [20]。输入的大小为 n 个字——这里的 n 远大于单台计算机本地内存所能容纳的大小，因此输入会分布在多台计算机之间。每台计算机的本地内存为 $\Theta(n^{\delta})$ 个字，其中 $0 < \delta < 1$ 为某个常数。共有 $\Theta(n^{1-\delta})$ 台计算机参与计算，因此总计拥有 $\Theta(n)$ 个字的分布式内存。

我们假设关键瓶颈在于计算机之间的通信，因此时间复杂度用通信轮次的数量来衡量。我们假设在每个轮次中，每台计算机最多可以向其他计算机发送 $\Theta(n^{\delta})$ 个字，并从其他计算机接收最多$\Theta(n^{\delta})$ 个字。本质上，你可以将本地内存中的所有内容发送给其他计算机，也可以接收任何能够放入本地内存的内容。当本文提到运行时间时，始终指通信轮次的数量（不过我们在此先指出，在我们的算法中，局部计算同样会很轻量）。

1.2 先前工作：快速求解 LCL问题

在最近的一项工作中，Balliu、Latypov、Maus、Olivetti 和 Uitto [4] 提出了高效的大规模并行计算算法，用于求解森林中的连通分量、树定根以及所谓的局部可检验标记问题（LCL）。由于我们直接建立在他们的工作之上，因此首先简要讨论他们的贡献。

LCL问题最早由 [23]形式化。这类图问题可以通过列出一组有限的可行局部邻域来描述。例如，“对最大度为 4 的图进行 5着色”就是一个 LCL问题；我们可以列出最大度为 4 的图中可能出现的所有合法 5着色邻域。通常，约束满足问题都是 LCL（只要度有界且标签集有限），而最大权独立集等全局优化问题则不是 LCL。

[4] 中的算法运行 $O(\log D)$轮次，其中 D 是输入图的直径，且没有渐近的全局内存开销。查找连通分量和根化是

它们的主要贡献，但这里我们主要关注解决LCL问题的部分。

用于求解 LCL问题 的算法由多个压缩输入图的阶段组成；共有 $O(1)$ 个阶段，每个阶段需要 $O(\log D)$ 轮次。在阶段 i 之后，他们在压缩图上定义一个新的 LCL 问题，使得其解可以扩展为针对阶段 $i-1$ 的图上所定义的 LCL 问题的解。经过 $O(1)$ 个阶段后，图被压缩为单个节点（树的根节点），对于该节点，任何 LCL 问题都可以轻松求解。随后，算法通过 $O(1)$ 个逆向阶段收尾，在解压所有压缩部分的同时，将正确的 LCL 解传播到图中解压后的部分。

1.3 主要新贡献：动态规划问题的统一框架

我们构建了搭建完成。ld 在 [4] 上，并提出一种新的算法框架，具有以下主要特点：

1. 我们能够在 $O(\log D)$ 时间内解决范围广得多的 一类问题——我们不再局限于解决 LCL 问题，而是能够解决一类更为一般的问题，即所谓的动态规划问题 （见定义1）。我们在表 1 中列出了一些示例，以展示我们的框架相较于 [4] 的适用范围。

2. 先前的算法 [4] 将压缩树和构造 LCL 解这两项任务交织在一起。我们将说明，这两项工作可以分开处理，具体方法将在第1.4节概述。尤其是，我们可以先用 $O(\log D)$ 轮次构造图的层次聚类，然后借助该聚类，在 $O(1)$ 轮次内解决任意动态规划问题。

大规模并行计算模型中，求解动态规划问题的最快先前算法是 Bateni、Behnezhad、Derakhshan、Hajiaghayi 和 Mirrokni 的算法 [5, 6], ，但该算法的运行时间为 $O(\log n)$，在低直径树上可能远差于 $O(\log\ D)$；此外，他们的算法是随机化的，而我们的算法是确定性的。

1.4 简单三步法

我们的算法框架分为三个步骤：

1. 将输入转换为**标准表示**；此阶段的运行时间为$O(\log D)$ 轮次。我们处理的是树结构数据，但这类数据可以用不同形式表示：例如，无根树可以表示为很长的无向边列表；有根树也可以表示为很长的字符串（例如，包含嵌套括号或嵌套开始和结束标签的字符串）。我们会将这些表示转换为更方便的标准形式：以有向边列表表示的有根树 。对于多种常用的树结构数据表示，我们证明可以在 $O(\log D)$ 轮次内完成转换。这是唯一一个取决于具体输入表示的步骤。详细内容见第3节。

2. 我们对这棵树进行**层次聚类**；这一阶段的运行时间为 $O(\log D)$ 轮次。我们将在第1.5节介绍层次聚类的性质。我们将证明，这样的聚类可以在 $O(\log D)$ 轮次内完成。这一步完全通用——既不依赖于输入表示，也不依赖于我们正在解决的问题。我们将在第4节中给出详细内容。

问题	先前工作 [4]	本文工作
顶点着色	✓	✓
边着色	✓	✓
极大独立集	✓	✓
最大权重独立集	—	✓
最大权重匹配	—	✓
最小权重支配集	—	✓
最小权重顶点覆盖	—	✓
加权最大可满足性问题	—	✓
最长路径问题	—	✓
和着色问题	—	✓
模 k计数匹配数	—	✓
树中位数问题	—	✓
贝叶斯图模型中的推断	—	✓
计算算术表达式	—	✓
验证类 XML 文档的结构	—	✓
计算每棵子树中输入标签的总和、最小值或最大值 每个子树中的输入标签	—	✓

表 1：使用我们的框架和先前工作 [4] 解决的问题示例。

3. 我们**解决所关注的问题**；这一阶段的运行时间为 $O(1)$ 轮次。我们将证明，在给定层次聚类的情况下，可以在$O(1)$轮次内解决各种与树结构数据相关的问题。我们将在第 5 节中给出详细内容。

总的来说，这种方法使我们能够在树上用 $O(\log D)$ 轮次解决各种计算问题。此外，由此得到的算法是条件最优的：假设（广为相信的）双环猜想 [1, 24, 2, 14] 成立，许多可以用该框架解决的问题都需要 $\Omega(\log D)$ 轮次。该猜想指出，即使有多项式数量的机器可用，判定输入图由一个长度为 n 的环还是两个长度为$n/2$的环组成，也需要 $\Omega(\log n)$ 个大规模并行计算轮次。已知该猜想意味着，求连通分量需要 $\Omega(\log D)$ 轮次 [7, 11], ，而这一结论又可用于证明，在树上求解一类动态规划问题需要 $\Omega(\log D)$ 轮次 [4]。

我们工作的核心概念是：

> 存在一种简洁、便捷的通用表示，可作为设计面向树结构数据的高效大规模并行算法的起点。

我们强调，对于给定的输入拓扑，只需计算一次层次聚类，之后即可将其复用于任何动态规划问题和任意输入值。

1.5 层次聚类

我们的层次聚类如图1所示。为方便起见，我们假设树中的所有节点出度均为 1；为确保这一点，我们在根节点处添加一条指向树外的额外虚拟边——求解所关注的问题时将忽略这条边。

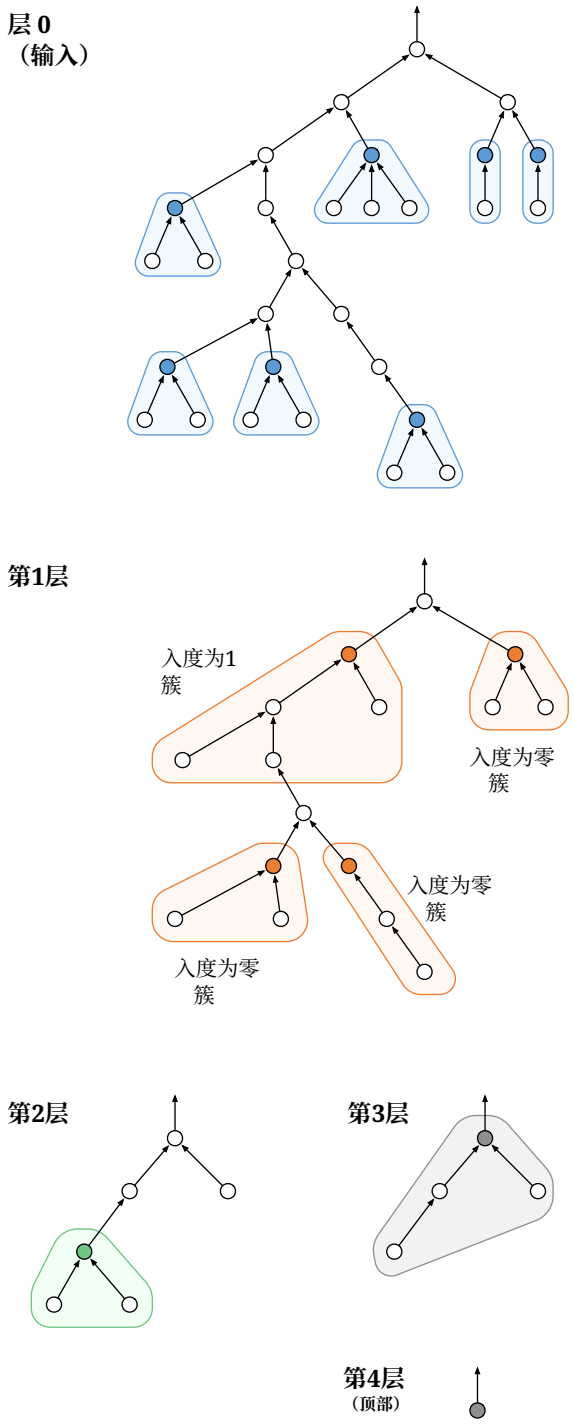


图 1：我们的层次聚类由常数个层组成。第 0 层是输入树。在每一层，我们压缩一些互不相交的簇，最终只剩下一个节点。每个簇至多包含 n^{δ} 个节点，每个簇恰好有一条出边，并且有零条或一条入边。

为了构造层次聚类，我们从原始树开始（这是第 0 层）。要得到第 $i+1$ 层，我们将一个节点簇收缩为一个节点。我们确保以下关键性质：

- 每个簇只包含 $O(n^{\delta})$ 个节点。
- 每个簇的出度为 1。
- 每个簇的入度为 0 或 1。
- 只有 $O(1)$ 层，且最顶层仅包含一个簇。

我们在第4节中正式定义层次聚类，并进一步证明它不仅存在，而且可以在 MPC 模型中用 $O(\log D)$ 轮次计算出来。

1.6 动态规划问题

我们主要关注的问题称为动态规划问题；正如我们将在第 1.6.1 节中看到的，许多典型的优化问题都可以轻松改造为这一框架：

定义1。 动态规划问题（DP 问题）是树上的一种计算问题，具有以下性质：

1. 任务是为每条边计算一个标签 。 2. 我们可以用一个动态规划表 $f(C)$来概括每个簇 C ，该表可用 $O(1)$ 个字表示。 3. 给定构成簇 C的所有节点的此类摘要，我们可以仅使用 $O(|C|)$ 个字的额外空间，计算动态规划表 $f(C)$ ；见图 2。 4. 给定 $f(C)$，我们可以计算顶层簇 C 的出边标签。 5. 假设我们知道簇 C 的入边和出边的标签，以及 C 各个组成部分的动态规划表，那么我们还可以仅使用 $O(|C|)$ 个字的额外空间，计算簇 C所有内部边的标签；见图 3。

这里，边的标签是对我们正在解决的具体任务的抽象，而动态规划表则是算法过程中所需的辅助数据结构。

1.6.1 示例：最大权独立集

我们将使用最大权独立集问题（最大独立集）作为贯穿全文的示例：在输入中，每个节点都有一个非负权重，任务是找到节点的最大权重子集 $X \subseteq V$，使得不存在边 $(u,\ v) \in E$ 满足 $u \in X$ 和 $v \in X$。

现在 th 最大独立集问题是一个动态规划问题，具体解释如下。 n:

- 边（u, v）的标签表示 $u \in X$。
- 设 C为入度为0的簇，其中（$u,\ v$）是出边。则 f（C）是一个包含两个元素的表：(1) 满足 $u \in X$ 的 C 中最大权重独立集的权重，以及 (2) 满足 $u \notin X$ 的 C 中最大权重独立集的权重。

- 设 C 为一个入度为1的簇，其中（u, v）是出边，（s, t）是入边。现在，f（C）是一个包含四个元素的表：对于 C 中 $u \in X$ 与 $u \notin X$ 以及 $t \in X$ 与 $t \notin X$的所有组合，分别记录最大独立集的权重。

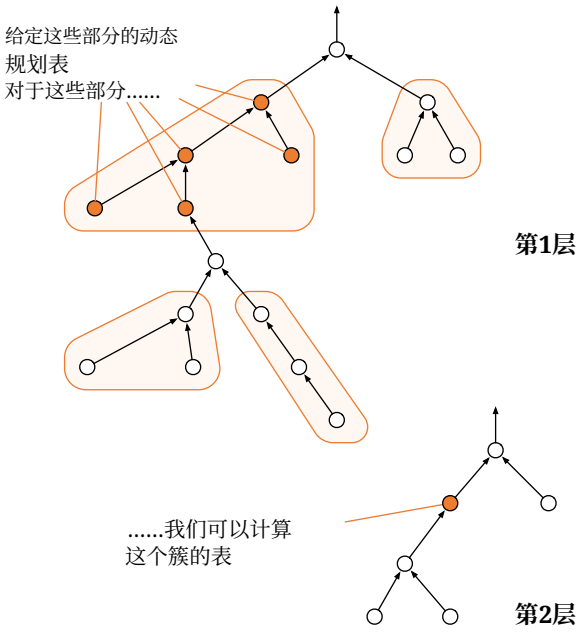


图 2：从底部到顶部：给定簇内的摘要后，我们假设可以计算出整个簇的摘要。

现在很容易推导出自底向上和自顶向下阶段的细节。请注意，我们处理入度为0的簇的方式，本质上与解决树上最大独立集的经典集中式顺序算法完全相同（参见例如 [13, 第 6.7] 节）。我们处理入度为1的簇的方式，可以看作解决有界树宽图上最大独立集的集中式顺序算法的一种特殊情形 [9]：我们可以用数量恒定的接口将簇与图的其余部分进行概括，并且可以合并这样的簇。

1.6.2 超越动态规划

虽然我们在此使用动态规划 这一术语来概括我们感兴趣的问题类别，但我们想强调的是，有许多问题都适用于这一框架，即使人们通常不会认为它们与动态规划有任何关系（请回顾表 1）。

1.7 技术问题：度非常高

到目前为止，我们一直忽略了一个技术难题：如果输入树中存在度大于 n^{δ} 的节点，该怎么办？在这种情况下，不可能找到较小的簇，因为包含节点 v的簇也会包含它的所有子节点。

幸运的是，对于最大独立集等许多问题，我们可以轻松地对输入和问题稍作修改，将每个度大于 $n^{\delta/2}$ 的节点 v 替换为一个 $O(1)$-深度树 T_v。新边会附带额外的标签，以便我们在动态规划算法中正确处理它们，并确保 T_v 中的所有节点都做出相同且一致的选择。

我们将在第4.4节和第5.3节中对此进行更详细的讨论。总而言之，只要度最多为 $n^{\delta/2}$，或者我们能通过用低度树替换高度节点来按需降低度，就可以解决任何动态规划问题（定义1）。

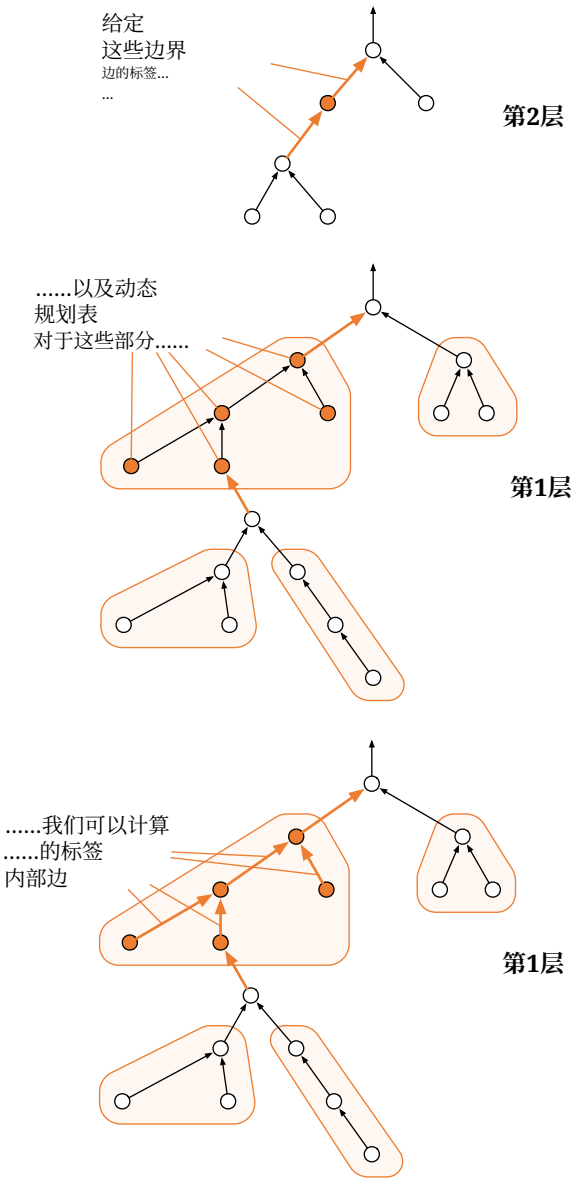


图3：从顶部到底部：给定边界边上的解，我们假设也可以计算内部边的解。

1.8 关于相关工作的进一步讨论

Bateni、Behnezhad、Derakhshan、Hajiaghayi 和 Mirrokni。先前的工作 [5, 6]提出了一种大规模并行计算算法，用于在大规模并行计算模型中对树进行动态规划，所需轮次为 $O(\log n)$。尽管他们处理的问题的具体类别表述略有不同，但其核心思想是一致的——他们也能够解决类似于最大独立集问题的问题。

我们的工作在两个方面严格优于他们的工作：我们的运行时间为 $O(\log D)$，这是条件最优的；而他们的运行时间为 $O(\log n)$；此外，我们的算法是确定性的，而他们的算法使用随机性。

在第 6.1 节中，我们还展示了如何使用我们的框架解决一个称为树中位数问题的问题。这个问题经过精心设计，不满足 [5, 6]中使用的一项技术要求，即二叉适配性质。非正式地说，对于二叉适配问题，可以用二叉树替换高度节点，因此只需解决有界度树上的动态规划问题；然而，树中位数问题无法通过如此直接的方式降低度。我们希望这个问题能够展示我们的框架具有广泛的适用性，也适用于先前工作所考虑范围之外的问题。

Balliu、Latypov、Maus、Olivetti 和 Uitto。先前工作 [4] 提出了一种 MPC 算法，可在 MPC 模型中通过 $O(\log D)$ 轮次解决树上的局部可检验标记问题（LCL）。我们的运行时间相同，但我们解决的问题涵盖了范围广泛得多的问题族（回顾表 1）。

我们使用了 [4] 中 वकिसति 的许多子程序和思想。例如，我们采用了他们用于将树进行根化的算法，而层次聚类的思想及其关键性质也归功于他们。

从概念层面来看，关键区别在于，他们的工作提出了一种单一的（可以说相当复杂的）算法，将树的聚类任务与 LCL 的解的构造任务混合在一起。层次聚类的过程相当隐式，而且它具有一些特性，使其无法直接应用于解决种类广泛的问题：例如，任意长的路径会被压缩成一个簇，而这个簇将无法再装入一台计算机的内存中；此外，叶节点会被大幅度地消除，这与并非所有动态规划问题都兼容。在我们的算法中，层次聚类首先被显式地构建出来，并且我们的聚类具有一些便于使用的特性，使我们能够在单台计算机内部对每个簇进行局部计算，同时也使我们能够处理范围广泛的问题。

其他相关工作。 虽然我们的技术对于一类 动态规划问题是有条件最优的，但在某些情况下，许多问题允许使用更快的算法。例如，Balliu、Brandt、Fischer、Latypov、Maus、Olivetti 和 Uitto [3] 研究了具有局部性质的 LCL问题类别，例如 MIS 问题。对于许多自然问题类别，他们给出的 大规模并行计算算法在直径较大的图上比 $\Theta(\log D)$ 高效得多。

Im、Moseley 和 Sun [19] 研究了 大规模并行计算 模型中的动态规划，针对的是与树结构输入没有直接关系的问题。

还有一种相关但更强大的模型，称为 AMPC。在该模型中，除了执行常规的大规模并行计算操作外，机器每轮还可以对分布式哈希表进行次线性的（自适应）查询。在 AMPC 模型中，计算子树大小的问题可在 $O(1)$轮次内解决 [8]。

在经典的 PRAM 模型中，类似的问题早在 20 世纪 90 年代就已得到研究——例如，Gibbons、Cai 和 Skillicorn [15] 提出了一种用于向上和

树中的向下累积，可在 $O(\log n)$ 时间内完成。我们强调，尽管对于 PRAM 模型中的所有此类问题，$\Omega(\log n)$ 是一个自然的下界，但我们仍能在 MPC 模型中实现 $O(\log D)$ 的运行时间。

2 预备知识

我们将使用以下基本操作：对含有 n 个元素的数组进行排序，以及对含有 n 个元素的数组计算前缀和。这两种操作都可以在大规模并行计算模型中通过确定性算法在 $O(1)$轮内完成，参见 [16, 17, 12]。

3 输入表示

第 4 节和第 5 节中的算法假设输入树以某个节点为根，并表示为一组有向边，其中每条边都从子节点指向其父节点。不过，除了这种标准表示之外，还可以用数组以多种方式表示树。本节将定义其他常用的表示方式，并说明如何在 $O(1)$轮内将这些表示方式中的任意一种输入转换为有向边数组。在第 6.3 节中，我们将说明算法框架如何使我们能够将标准表示转换回这些表示方式中的任意一种。

3.1 定义

我们考虑以下列形式之一表示的树结构数据；我们以图 4 所示的树 T为例：

- **边列表：** 这是我们的算法所使用的表示形式。输入数组中的每个元素都包含一对整数，表示树中从子节点指向其父节点的一条有向边。使用图 4 中给出的节点标签，树 T可表示为数组 [(1，4)，(2，3)，(5，4)，(4，3)],。

- **括号字符串：** 在这种表示中，树以一个括号正确嵌套的数组表示，等价地说，也可以用开始和结束标签表示。树中的每个节点都由一对括号“(”和“)”表示。我们可以自底向上地将该数组解释为一棵有根树，其中叶节点表示为空括号对“()”。最外层的括号对表示根节点。例如， T 可表示为数组 [(，(，(，)，(，)，)，(，)，)]。

- **BFS 遍历：** 该数组表示树的 BFS 遍历：数组的索引按 BFS 顺序对应树中的节点，而数组元素则包含父节点的索引。树 T 可表示为 $[-, 1, 1, 2, 2]$。

- **DFS 遍历：** 与上述内容类似，树以表示其 DFS 遍历的数组形式给出。树 T 可以表示为 $[-, 1, 2, 2, 1]$。

- **父节点指针：**与上文类似，但节点的顺序是任意的。若按图 4 中的标签对节点排序，则树 T可表示为 $[4, 3, -, 3, 4]$。

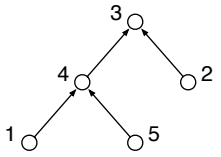


图4：第3.1节中用作示例的树 T 。

3.2 表示规范化

如果树最初以无向边列表的形式给出，我们可以先以任意节点为根，并使用 [4]中的算法，在 $O(\log D)$轮次内确定边的方向。

BFS 遍历、DFS 遍历和父节点指针已经以不同方式将输入表示为有向边集合，因此很容易将它们转换为边列表表示。非平凡之处在于证明我们可以在大规模并行计算模型中用 O 轮次，从**括号字符串转换为边列表表示**。

我们将首先说明如何对 $\delta = 1/2$ 执行此转换，也就是说，假设有 $m = \sqrt{n}$ 台计算机，每台计算机有 $O(\sqrt{n})$ 内存；然后我们将描述如何针对任意 δ 将同一策略推广。

设 A 为包含正确嵌套括号的数组。我们假设 A 中的每个左括号“(”都表示树中的一个节点。现在，对于每个左括号，我们需要找到它对应的父左括号。

最初， A 被均匀分布在 $\sqrt{n}$ 台计算机 $N_0, \ldots N_{m-1}$ 上，使得 N_i 包含元素 $A[i\sqrt{n}], \ldots, A[(i+1)\sqrt{n}-1]$。设 $A[i]$ 和 $A[j]$ 为两个左括号，满足$i < j$。我们知道，如果从 $A[i+1]$ 到 $A[j-1]$ 的所有括号都正确嵌套，那么 $A[i]$ 就是 $A[j]$ 的父节点。如果 $A[i]$ 是 $A[j]$ 的父节点，且它们都存储在同一台计算机中，那么该计算机可以轻松地将 $A[i]$ 识别为 $A[j]$的父节点。难点在于，当 $A[i]$ 和 $A[j]$ 存储在不同的计算机中时，如何识别父节点。

注意，如果 $A[p]$ 和 $A[q]$ 是一对表示同一节点的左括号和右括号，并且二者都存储在某台计算机 N_i 中，那么 $A[p]$ 不可能是 $A[k]$的父节点，如果 $A[k]$ 存储在另一台计算机中。因此，让我们消去 存储在同一台计算机中的正确嵌套括号对。现在，每台计算机N_i中的剩余括号将只是一串（可能为空的）右括号，后面接着一串（可能为空的）左括号，例如“)))))(((”。令 S_i 为 N_i中剩余括号的数组。

计算机 N_i 计算一对数值 (c_i, o_i)，其中 c_i 和 o_i 分别表示 S_i中的右括号和左括号数量，并将其广播给所有其他计算机。利用这些信息，我们可以为每个节点确定包含其父节点的数组 S_i ，以及该父节点在 S_i 中的索引，具体如下。对于存储在 N_i处的每个左括号 $A[j]$ ， N_i 在本地计算 l_j 和r_j，分别表示在 S_i中 $A[j]$左侧和右侧的右括号和左括号数量。然后，当 $j < k$且 $a < b$时，存储在 N_a中的 $S[j]$ 是存储在 N_b中的 $A[k]$ 的父节点，其中 a 是满足以下条件的最大整数。

$$r_j + \sum_{x=a+1}^{b} (o_x - c_x) - l_k = 0,$$

可由 N_b 在 $O(1)$ 轮次内计算。

要确定 A 中某个节点的父节点索引，我们还需要进行一些额外的计算。对于每个节点 v ，我们会生成两个元组：

- 类型 1： $[i, j, 1, v]$ 表示节点 v 存储在 j的第 S_i个索引处——持有节点 v的计算机可以直接获得此信息。
- 类型 2： $[i, j, 2, v]$ 表示节点 v 的父节点 存储在 S_i的第 j个索引处——持有节点 v的计算机可以按照上述方法计算出这一信息。

这样，系统中总共会有 n 个元组，我们可以在 $O(1)$ 轮次内对它们进行排序。排序后，数组中始终会有一个类型 1 的元组，表示一个节点 v，后面跟着零个或多个类型 2 的元组，表示其子节点 v。这样，我们可以在 $O(1)$ 轮次内识别所有父子边。

3.2.1 低内存版本

上文中，我们假设已经得到 $\delta = 1/2$。现在来看，对于任意 k，该策略如何推广到$\delta = 1/k$ 。在这种情况下，我们将采用 k 层策略。

在第 $\ell = 1, \ldots, k$ 层，我们在概念上将输入划分为长度为 $n^{\ell/k}$ 的数据块。设 i 为节点 j 的父节点。若边 (i, j) 的两个端点 i 和 j 位于同一个数据块中，则称该边为 局部 边；否则称其为全局边。在第 ℓ 层之后，我们维持以下不变式：

- 我们已经发现了所有局部的边（i， j）。
- 对于每个数据块，我们都计算了一个摘要（c,o），用于表示该数据块内部在消去其中正确嵌套的括号后，剩余的右括号和左括号的数量。

假设 C 是一个处于第 $\ell + 1$ 层的块，由处于第 ℓ 层的子块 C_1、.....、C_c 组成；根据定义，$c = n^{1/k}$。现在，所有持有块 C 部分内容的计算机都可以获知每个子块 C_1、.....、C_c 的摘要（c， o），因为这些信息能够存储在它们的本地内存中。通过采用我们在情形 $\delta = 1/2$ 中使用的相同策略，我们现在可以计算块 C 内的所有局部边，同时计算整个块 C 的摘要（c， o）。因此，给定第 ℓ 层的不变量，我们可以在 $O(1)$ 时间内满足第 $\ell + 1$ 层的不变量。

如果 δ 不是一个合适的有理数 $1/k$，我们可以将其向下取整，并让一台拥有 $O(n^{\delta})$ 内存的计算机扮演多台各自拥有 $O(n^{1/k})$ 内存的计算机的角色，这样上述方案仍然适用。

4 层次聚类

本节将介绍一种 $O(\log D)$ 轮算法，用于计算动态规划算法所需的层次聚类（见第 5 节）。请注意，该聚类与我们之后要解决的问题无关。

4.1 定义

现在，我们将第1.5节中介绍的层次聚类思想形式化；图1给出了一个示例。

定义 2 (簇)。 簇 C 是一个集合，其中每个元素要么是一个节点 u_i ，要么是另一个簇 C_i。我们递归地定义参与其中的节点集合 C 如下：

$$V(C) = \bigcup_{C_i \in C} V(C_i) \cup \{u_i \mid u_i \in C\}.$$

我们要求簇 C 至多包含 n^{δ} 个元素，并且割边集合 $(V(C)\ V \backslash V(C)) \subseteq E$ 恰好有一条出边，且至多有一条入边。

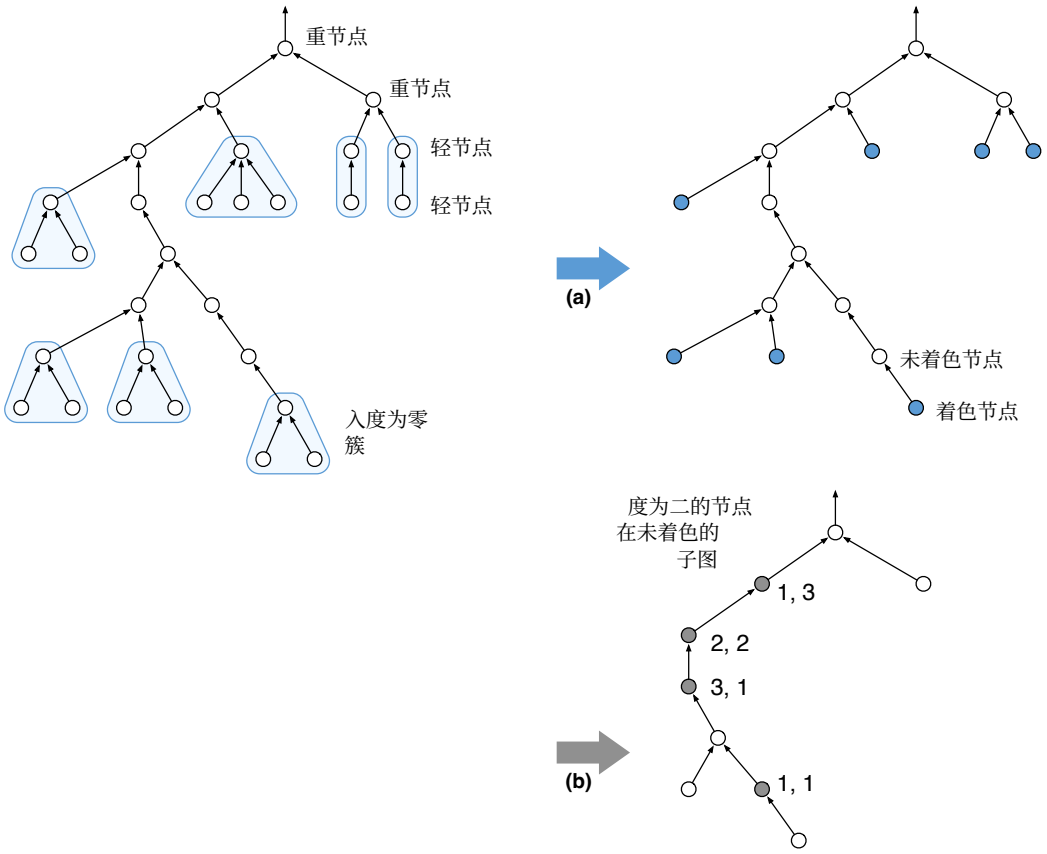


图 5：(a) 创建入度为零的簇。(b) 创建入度为一的簇：我们识别由未着色节点在其诱导子图中形成的路径，并计算这些节点在路径中向上和向下的位置。

根据入边的数量，我们将簇分为两种类型：入度为零和入度为一。

定义 3 （层次聚类）。 有根树 $T = (V, E)$ 的层次聚类 是一组集合 $S_0, S_1, ..., S_L$，称为层，并满足 $L = O(1)$ 以及以下条件

1. 每个 S_i 由节点或簇组成，

2. $S_0 = V$, 3. 对于 $i \geq 1$，(i) S_i 中的节点也是 S_{i-1} 中的节点，并且 (ii) S_i 的簇构成了 S_{i-1}其余元素的一个划分，4. S_L 包含一个作为簇的元素。

为了便于从算法角度使用聚类，虽然将其视为一个独立的图论概念 最容易理解，但我们需要为簇分配 ID，并存储簇与其节点/簇之间的某些指针等。更形式化地说，我们为每个簇 $C \in S_i$ 分配一个唯一的簇 ID，并建立指向和来自 S_{i-1} 中所包含的 C的簇和节点的指针。由于一个簇恰好有一条出边，且至多有一条入边，因此我们可以将 S_i 中的每个簇收缩为一个节点，使得所得的图构成一棵树 T_i ，其中每条边都对应于原始树中的一条边。

4.2 构造聚类

如第3节所述，不失一般性，我们可以假设输入是一棵有根树 $T = (V, E)$，包含 n 个节点，并以边列表的形式表示。我们还将假设最大度为 $n^{\delta/2}$，但在第4.4节中会介绍如何克服这一限制。通过对边进行排序，我们还可以假设每个节点及其相连的边都存储在同一台机器上。我们的目标是构造一个符合定义3的层次聚类。

4.2.1 高层思路

我们主要沿用 [4]算法中的思路。不过，以下内容将重点介绍两项关键差异，并且我们还需要证明层数仍然有常数上界。

如果一棵子树包含一条中心路径，且所有其他节点到该路径的距离都不超过1，我们就称它为毛毛虫树。我们将交替执行两个步骤，共进行 $O(1)$轮：

1. 创建入度为零的簇：我们找出节点 v ，使得可以将以 v 为根节点的整棵子树 $T(v)$ 替换为一个簇。2. 创建入度为一的簇：我们找出一组互不相交的毛毛虫树，并可以将它们替换为簇。

在 [4], 中，他们完全移除了我们所说的入度为零的簇，随后只需收缩较长的路径。此外，他们收缩的路径长度可以任意长，而我们的簇不能太大。尽管如此，我们可以证明进展足够快，并且经过 $O(1)$对这样的步骤后即可完成。

在我们的算法中，我们将为对应入度为零的簇的节点着色，而不是将它们移除。这样，对于树中未着色的部分，我们基本上可以沿用 [4] 的流程和分析。由于着色节点始终是叶节点，且每个节点最多有 $n^{\delta/2}$ 个邻居，如果我们在每个簇中放入至多 $n^{\delta/2}$ 个未着色节点及其着色邻居，那么簇的大小将不超过 n^{δ}，满足所需条件。

4.2.2 创建入度为零的簇

按照 [4], 的定义，若节点 v 的子树中未着色节点超过 $n^{\delta/2}$ 个 T（v），则称其为重节点；其余节点称为轻节点。

我们将以下来自 [4] 的引理 6.13 的结果应用于未着色子图（即由未着色节点诱导的子图）：存在一个确定性的最优空间 O（log D）时间大规模并行计算算法（CountSubtreeSizes），其中每个节点 v 都能获知 T（v）的确切大小，或者获知 $|T$（v）$| > n^{\delta/2}$。

利用这些信息，我们可以识别出每个节点 u，使得 u 是轻节点，而其父节点 v 是重节点。我们应用 [4]中的引理 6.14：存在一个确定性的最优空间复杂度 O、运行时间为(log D)的 MPC 算法（GatherSubtrees），对于每个这样的节点 u，将 T(u)收集到托管 u 的机器中。然后，我们用一个入度为零的簇替换 T(u)，并将其表示为一个着色节点——参见图 5。整体运行时间为 O(log D)。该簇的大小将被限制在 n^{δ}以内，因为只有 $n^{\delta/2}$ 个未着色节点，且每个节点最多附着 $n^{\delta/2}$ 个着色叶节点。

4.2.3 创建入度为一的簇

现在我们可以描述第二步：创建入度为一的簇。其思路是在未着色子图中识别长路径。若同时考虑着色节点，未着色子图中的一条长路径就对应一只毛毛虫。

我们将引理 6.17 应用于未着色子图：存在一个确定性的 O(log D) 时间大规模并行计算算法（CountDistances），使每个度为二的节点都知道自己到由度为二的节点构成的路径两端点的距离——见图 5。

利用这些距离，我们将把未着色子图中由度为二的节点构成的每条路径 P 划分为长度至多为 $n^{\delta/2}$ 的子路径（即距离值为 1，.....，$n^{\delta/2}$ 的节点构成第一条子路径，依此类推）。我们称这些子路径为路径片段 P'。我们将每个片段集中到一台机器上，并通过将所有与 P' 相连的着色节点也纳入其中来形成一个簇 C。这样将得到一只毛毛虫 C；由于图的最大度为 $n^{\delta/2}$，簇的大小至多为 n^{δ}，符合要求。这一步的总运行时间为 O(log D)。

4.3 层数

根据构造，所有簇都足够小。我们仍需证明层数有一个常数上界：

引理 4。 我们创建的层次聚类中的层数为 $O(1)$.

为证明引理 4，首先考虑一个替代过程 Π_1：在该过程中，我们删除入度为零的簇，而不是将其标记为有色；并且将任意长的路径替换为一条边，这与 [4]类似。我们可以证明：

引理 5。 过程 Π_1 的每次迭代都会使树的规模缩小$\Omega(n^{\delta/2})$倍。

证明。 假设我们从一棵含有 T_0 个 n_0 节点的树开始。删除入度为零的簇，并将所有路径替换为单个入度为一的簇后，所得的树中有 n_1 个节点 T_1。这意味着所有路径的长度至多为 1。考虑一棵树 T_1'，它与原树相同，只是将所有路径替换为一条边。注意，$|T_1'| \geq n_1/2$，并且 T_1 的叶节点数量与 T_1' 相同。现在，在 T_1' 中不存在度为 2 的节点。由于任何树的叶节点数至少与度为 3 或更大的节点数相等，因此 T_1' 至少有 $|T_1'|/2$ 个叶节点，这意味着 T_1 中至少有 $n_1/4$ 个叶节点。

考虑一个叶节点 v。由于 v 未被删除，它必定是重的，因此以 v 为根的子树大小为 $> n^{\delta/2}$。因此，在我们开始进行聚类处理之前，节点数为 $n_0 \geq (n_1/4) \cdot n^{\delta/2}$。所以，每个聚类步骤中的节点数都会缩小为原来的 $n^{\delta/2}$。

□

接下来稍微修改这个过程；令 Π_2 为这样一个过程：我们仍然删除入度为零的簇，而不是将它们标记为已着色，但会将任意长的路径替换为一个节点和两条边。

引理 6. 过程 Π_2 的每次迭代都会使树的规模缩小$\Omega(n^{\delta/2})$倍。

证明。 本质上， Π_2 的行为相当于先执行一次 Π_1 ，然后再细分一些边。细分只会使节点数量增加一倍。 □

最后，令 Π_3 为这样一个过程：我们仍然删除入度为零的簇，而不是将它们标记为已着色，但会将长路径替换为一串簇，每个簇最多包含 $n^{\delta/2}$，这与我们的实际过程类似。我们可以证明：

引理 7. 过程 Π_3 经过 O次迭代 后，会使树的规模缩小$\Omega(n^{\delta/2})$倍。

证明。 如果将 Π_3 迭代超过 $2/\delta$ 次，每条路径都会被收缩成只包含一个节点的路径。因此，Π_3 迭代 $2/\delta$ 次所取得的进展至少相当于Π_2迭代一次。

□

引理 4 现在可以通过观察得出： Π_3 准确描述了我们的实际过程中未着色子图中发生的情况：

引理 *4* 的证明。对未着色子图迭代应用引理 7 O(1) 次后，我们可以看到未着色部分收缩为一个节点，此时整个图将容纳在一个入度为零的簇中。

□

4.4 处理高度节点

到目前为止，我们一直假设给定的输入树的度最多为 $n^{\delta/2}$。克服这一限制的通用解是将高度节点替换为深度为 O(1)的子树。

现在简要介绍如何在 MPC 模型中用 O(1) 轮实现它。我们可以按照父节点标识符对原始边列表进行排序。现在，每当一台机器持有的具有相同父节点 u 的边超过 $n^{\delta/2}$ 条时，它就会引入新的节点，这些节点的父节点为 u ，而这些新节点将成为 u 的 $n^{\delta/2}$ 个子节点的新父节点。我们重复此过程 O(1) 步，直到所有节点的度都足够低。在整个过程中，我们会跟踪边的类型：它是原始边，还是在拆分高度节点时创建的辅助边——在之后求解动态规划问题时（参见第 5.3 节），需要用到这一信息。

这一过程只会使节点数量和直径增加一个常数倍。因此，如果现在应用聚类算法，运行时间仍然是 O（log D）轮次，其中 D 是原始树的直径。

5 解决动态规划问题

现在我们将展示如何利用第4节中计算出的层次聚类来求解动态规划问题（回顾定义1）。

5.1 从底部到顶部

设 $L = O(1)$ 为层次聚类中的层数。我们通过维护以下不变式，在 L 次迭代中填充动态规划表：

定义 8（自底向上不变量）。 在迭代 $i = 0$，1、.....、L之后，第 i 层的每个簇 C都标记有其动态规划表 f（C），而所有其他节点都标记有其原始输入。

开始时，这一不变量很容易满足，因为 layer0 是我们的输入树，并且目前还没有簇。

现在假设在迭代 $i > 0$之前我们满足该不变量。此时，每个仍参与计算的节点都知道其在 layer i 中的簇标识符，以及其输入或动态规划表。此外，根据假设，这些信息可以放入 $O(1)$ 个词中。现在，我们可以对簇标识符数组和节点标签进行排序，从而确保与同一簇相关的数据连续存储。这样，一个簇最多跨越两台机器；再经过一步额外的路由，我们就能确保每个簇都完全包含在一台机器内。

现在，我们可以通过应用假设存在的顺序算法，在本地汇总每个簇 C。最后，我们为每个簇得到一个汇总结果 f（C）。随后，我们可以再次应用排序，将汇总结果 f（C）移回用于存储簇 C信息的数组位置。从本质上说，这使我们能够并行地为每个簇执行图 2 中所示的操作。

最终，我们已经计算出了所有层中所有簇的动态规划表 rs.

5.2 从顶部到底部

现在我们继续求解该问题，也就是为边填入标签。现在我们按相反的顺序遍历各层，并维持以下不变式：

定义 9（自顶向下不变式）。 经过第 $i = L$ 次迭代后，$L - 1$，.....，0，我们已经计算出树中所有边（u，v）的标签，该树对应于第 i层；这些信息与节点 u 一起存储。

对于 $i = L$，这个不变式可以成立：树中只有一条边，即最顶部簇 C 的出边；根据假设，给定 $f(C)$ 后，我们可以为这条边赋予标签。

现在假设在迭代 $i < L$ 之前我们满足这个不变量。现在，如果 C 是出现在层 i中的一个簇，我们可以通过排序确保 C 知道其出边的标签和入边的标签（如果有）。然后，我们再次重新组织数据，使构成层 i 上的簇 C 的层 $i - 1$ 中的节点存储在同一台计算机上。现在我们可以应用顺序算法，为 C 的所有内部边标记。本质上，这使我们能够并行地为每个簇执行图 3 中所示的操作。

最终，我们计算出了第 0 层所有边的标签，也就解决了原始问题。

5.3 处理高度节点

在第4.4节中，我们用 $O(1)$-深度子树替换了高度节点；因此，树中将同时包含原始边和辅助边。一般而言，这将产生一个新的动态规划问题，不同的边可能适用不同的规则。对于我们一直使用的示例——最大独立集，其规则可以指定如下：

- 原始边（u，v）：如果有 $u \in X$，就必须有 $v \notin X$，反之亦然。

- 辅助边（u, v）：如果有 $u \in X$，就必须有 $v \in X$，反之亦然。

本质上，这确保代表同一个原始节点的所有新节点都作出一致的选择。类似的策略适用于广泛的图问题。

6 进一步应用

在本节中，我们将讨论该框架的进一步应用。首先，我们将探讨如何在复杂程度高于最大独立集问题的问题中处理高度节点这一挑战。

6.1 树中位数问题

在树中位数问题中，输入是一棵有根树，其中每个叶节点都关联一个数字。该任务以递归方式定义：节点的标签必须是其子节点标签的中位数。

对于子节点数量为偶数的节点，我们要求输出两个中位数中较小的一个。这使我们可以不失一般性地假设所有节点的子节点数量均为奇数，因为子节点数量为偶数的节点可以添加一个值为 $-\infty$的虚拟叶子子节点。

这个问题有一种简单的顺序策略，即从叶节点开始为节点标记。然而，如前所述，树中位数问题不属于 [5, 6] 先前工作所考虑的问题类别。具体而言，树中位数问题并非二叉适配，因为我们无法用一棵二叉树替换高度数节点（除非使用大得多的动态规划表，从而占用更多内存）。尽管如此，在本节中，我们将证明，我们的算法可以在 $O(\log D)$ 轮次内解决该问题（且空间复杂度最优）。

6.1.1 处理高度节点

我们按照第4.4节所述，用一棵直径为 $O(1)$ 的树替换每个高 度节点 u 的子节点。请回想一下，u 原有的子节点在这棵树中都是叶节点，且每个内部节点的度至多为 $n^{\delta/2}$。由于我们不关心这些新创建的节点所计算出的输出，因此将它们称为 无关 节点。无关节点会保存一些中间值，以帮助 u 计算出正确的中位数。在整个过程中，我们还会记住每个节点原来的父节点。

6.1.2 入度为零的簇

层 i 中入度为零的簇位于单台机器上。因此，可以根据层数不超过 $i-1$ 的簇所计算出的中位数，在本地计算该簇中所有节点的中位数。如果该簇包含高程度节点，我们可以通过按原始父节点标识符和中位数值对所有节点进行排序，并选取这些高程度节点的子节点的中位数，从而并行计算所有此类节点的中位数。

6.1.3 入度为一的簇

层 i 上的入度为一的簇由一条唯一的有向路径 P 构成，该路径包含 ℓ 个节点 p_j （$0 \le j < \ell$）；簇的入边指向 $p_{\ell-1}$，簇的出边从 p_0 发出。对于 $j > 0$，每个节点 $p_j \in P$ 都可以有任意数量来自其他节点和较低层簇的入边，以及一个子节点 $p_{j-1} \in P$。p_j 的输出是变量 x_j，初始时未知。

引理 10。 对于每个节点 p_{j-1}，只需存储两个值 a_j, b_j ，使得 x_{j-1} =中位数(x_j, a_j, b_j) 和 $a_j \le b_j$。

证明。 如果 p_{j-1} 有超过两个叶节点子节点，可以删除除中间两个之外的所有子节点，而中位数仍会保持不变。通过按父节点标识符和值排序，可以并行地对树中的所有节点执行此操作。□

我们将说明如何用一条边替换长度为 2 的子路径。所有节点都可以并行重复此过程，在 O轮次内压缩路径 P P(1)。

情况如下：我们有一条路径 $p_2 \to p_1 \to p_0$，其中有 $x_0 = \text{median}(x_1, a_1, b_1)$ 和$x_1 = \text{median}(x_2, a_2, b_2)$。我们希望写出 $x_0 = \text{median}(x_2, a, b)$。我们可以按照以下规则进行：

$$(a,b) = \begin{cases} (a_1, a_1) & \text{if } b_2 \le a_1 \\ (b_1, b_1) & \text{if } b_1 \le a_2 \\ (\max\{a_1, a_2\}, \min\{b_1, b_2\}) & \text{otherwise} \end{cases}$$

引理 11。 值 x_0 是我们希望计算的正确中位数。

证明。 注意， $x_0 = \text{median}(x_1, a_1, b_1) \in [a_1, b_1]$，并且 $x_1 = \text{median}(x_2, a_2, b_2) \in [a_2, b_2]$。现在我们进行分类讨论：

如果 $b_2 \le a_1$，这意味着 $x_1 \le a_1$，因此 $x_0 = a_1 = \text{median}(x_2, a_1, a_1)$。

类似地，如果 $b_1 \le a_2$，这意味着 $x_1 \ge b_1$，因此 $x_0 = b_1 = \text{median}(x_2, b_1, b_1)$。

最后一种情况是， $[a_1, b_1]$ 和 $[a_2, b_2]$ 的交集为区间 $[a, b]$，其中

$$a = \max\{a_1, a_2\}$$
$$b = \min\{b_1, b_2\}.$$

很容易看出， $x_0 = x_2$ 当且仅当 $x_2 \in [a, b]$。并且如果 $x_2 \notin [a, b]$，那么 $x_0 = a$ 当 $x_2 < a$ 时成立，而 $x_0 = b$当 $x_2 > b$时成立。因此 $x_0 = \text{median}(x_2, a, b)$。□

如果 P 不包含任何（原始的）高度数节点，则通过将所有节点 $p_j \in P$ 的值 x_j 写为 $\text{median}(x_\ell, a'_j, b'_j)$，即可在本地处理整个簇；其中 a'_j 和 b'_j 通过反复应用针对长度为 2 的路径所述的过程计算得出。

现在令 $p_{j-1} \in P$ 为一个高 degree 节点。我们可以像引理 10 中那样，按照原始父节点标识符排序，写出 $p_{j-1} = \text{median}(x_j, a_j, b_j)$。这里，$x_j$ 是无关子节点$p_j \in P$的值，但我们希望 x_j 是 p_{j-1}的原始子节点中属于 P的那个节点的值。为了传播正确的值，我们对 P 中的无关节点采用不同的规则：它们只需复制其子节点的值。因此， P 中的每个高 degree 节点也会计算出正确的中位数值。

6.2 贝叶斯树推断与置信传播

概率图模型或贝叶斯图模型在机器学习和统计学中无处不在[21]。概率图模型是一种图，其中的节点（例如， $x_i \in \mathbb{R}^{d_x}$）表示具有由图的顶点定义的条件分布结构的隐随机变量。我们还会获得图的观测值（例如， $y_i \in \mathbb{R}^{d_y}$），而图模型推断中的一个重要问题是计算节点的后验分布，也就是 $p(x_k \mid y_1, \ldots, y_n)$，其中 $k = 1, \ldots, n$ 是选定的节点。

在本节中，我们考虑贝叶斯图模型中的一个重要特殊情形，其中图是一棵树，观测是获得的条件独立观测。

根据给定的条件分布模型 $p(y_i \mid x_i)$ 生成每个节点。随后，节点的条件分布形式为 $p(x_i \mid x_{\gamma_i})$，其中 γ_i 是节点 x_i 的子节点索引集合（叶节点 j 没有 $\gamma_j = \emptyset$）。现在看来，本文提出的算法框架能够在 $O(\log D)$ 个大规模并行计算轮次内计算 $p(x_k \mid y_1, \ldots, n)$，至少在我们此处考虑的高斯特殊情形中是如此。不失一般性，我们假设 x_k 对应于树的根节点。

我们 将节点 i 的团索引记为 $\alpha_i = \{i\} \cup \gamma_i$，并定义团势函数 a s

$$\psi_i(x_i, x_{\gamma_i}) = \psi_i(x_{\alpha_i}) = p(y_i \mid x_i)\, p(x_i \mid x_{\gamma_i}).$$

计算后验概率密度 $p(x_k \mid y_1, \ldots, n)$，对应于计算团势函数乘积的边缘分布：

$$p(x_k \mid y_1, \ldots, n) \propto \int \cdots \int \prod_{i=1}^{n} \psi_i(x_{\alpha_i})\, d(x_{1:n \setminus k}).$$

用于在树上解决此类问题的高效算法称为置信传播 [21]。对于路径图（即每个 α_i 都是一对索引），推断问题的解由贝叶斯滤波器和平滑器 [25], 给出，而置信传播则对应于所谓的双滤波器平滑器。近期已有研究开发出贝叶斯滤波与平滑问题（即概率路径图上的推断）的并行算法 [26, 18]，但尚未涉及 MPC 模型。不过，这些算法中使用的结合式表述为路径压缩提供了实用手段，而这正是贝叶斯树中同样需要的。

如果我们现在认为当前的树实际上是当前簇内的子树，那么需要考虑以下两种可能的情况：

1. 入度为零的簇，我们要计算

$$\bar{\psi}_1(x_1) = \int \cdots \int \prod_{i=1}^{n} \psi_i(x_{\alpha_i})\, d(x_{2:n}),$$

其中 x_1 是根节点。势函数 $\psi_1(x_1)$ 对应于将入度为零的簇压缩为单个节点。

2. 入度为一的簇，我们要计算

$$\bar{\psi}_{j \to 1}(x_1, x_j) = \int \cdots \int \prod_{i=1}^{n} \psi_i(x_{\alpha_i})\, d(x_{2:n \setminus j})$$

对于某个索引 $j \in \{2, \ldots, n\}$。这里的 $\psi_{j \to 1}(x_1, x_j)$ 对应于将该簇压缩为一个节点 x_1，并保留一个开放的子节点位置 x_j。

具体来说，现在来看一个线性高斯图，其中我们有（对于$i = 1, \ldots, n$）：

$$p(x_i \mid x_{\gamma_i}) = \mathcal{N}(x_i; \sum_{j \in \gamma_i} F_j\, x_j + c_i, Q_i),$$
$$p(y_i \mid x_i) = \mathcal{N}(y_i; H_i\, x_i + d_i, R_i),$$

也就是说，

$$\psi_i(x_{\alpha_i}) = \mathcal{N}(y_i; H_i\, x_i + d_i, R_i)\, \mathcal{N}(x_i; \sum_{j \in \gamma_i} F_j\, x_j + c_i, Q_i),$$

其中，$\mathcal{N}(x;\,\mu,\Sigma)$表示均值向量为 μ、协方差矩阵为 Σ的多元高斯概率密度。因此，一个节点的表示由 $|\gamma_i|$ 矩阵$\{F_j:\ j\in\gamma_i\}$以及 c_i、 Q_i、 y_i、 H_i、 d_i和 R_i组成。

上文中的入度为零的簇操作（1）只需要一个基本操作即可实现：消去一个叶节点。我们可以重复此操作，直到整棵树缩减为单个节点。不过，我们需要确保每个节点只需使用常数大小的内存即可执行此操作。幸运的是，高斯情形正是如此。

现在考虑一棵树，其中添加了一个额外的节点 x_{n+1}，并将其连接到节点 j。这样一来，节点 j 就会多出一个子节点：

$$\psi_j(x_{\alpha_j}) \to \tilde{\psi}_j(x_{\alpha_j}, x_{n+1}),$$

我们还需要乘以叶节点的势函数 $\psi_{n+1}(x_{n+1})$。因此，联合势函数为$\psi(x_{1:n+1}) =$

$$\left[\prod_{i=1}^{j-1}\psi_i(x_{\alpha_i})\right]\,\tilde{\psi}_j(x_{\alpha_j}, x_{n+1})\,\psi_{n+1}(x_{n+1})\left[\prod_{k=j+1}^{n}\psi_k(x_{\alpha_k})\right],$$

在当前的入度为零的簇情形中，我们希望对除 x_1 以外的所有变量积分；而在下文的入度为一的簇情形中，则希望对除 $x_1,\ x_j$ 以外的所有变量积分。在这两种情形中，消去叶节点都对应于对 x_{n+1}积分。

在高斯情形中，可以对 x_{n+1} 进行闭式积分。实际操作中，这包括计算 $\psi_{n+1}(x_{n+1})$的后验协方差和均值参数，它们为

$$\begin{aligned}\tilde{Q}_{n+1} &= \left[Q_{n+1}^{-1} + H_{n+1}^{\top}R_{n+1}^{-1}H_{n+1}\right]^{-1},\\ \tilde{b}_{n+1} &= \tilde{Q}_{n+1}\left[H_{n+1}^{\top}R_{n+1}^{-1}(y_{n+1}-d_{n+1}) + Q_{n+1}^{-1}c_{n+1},\right],\end{aligned}$$

然后将它们融合到其父节点的均值和协方差参数中：

$$\begin{aligned}c_j &\leftarrow F_{n+1}\,\tilde{b}_{n+1} + c_j,\\ Q_j &\leftarrow Q_j + F_{n+1}\tilde{Q}_{n+1}F_{n+1}^{\top},\end{aligned}$$

这两项操作都可以在常数内存中完成。

为了实现入度为一的簇操作（2），我们可以先反复使用上述叶节点消除过程，将入度为一的簇缩减为一条单独的入度为一路径。接下来剩下的是一条如下形式的路径（中间节点已重新编号）

$$\psi_1(x_1,x_2)\,\psi_2(x_2,x_3)\,\psi_3(x_3,x_4)\ \times\cdots\times\ \psi_{j-1}(x_{j-1},x_j)$$

我们希望对其进行积分$x_{2:j-1}$。这可以通过对势函数进行两两组合来实现，并可递归地完成，如下所示

$$\bar{\psi}_{m+1\to 1}(x_1,x_{m+1}) = \int \bar{\psi}_{m\to 1}(x_1,x_m)\,\psi_m(x_m,x_{m+1})\,dx_m$$

¯初始条件为 $\psi_{2\to 1}(x_1,x_2) = \psi_1(x_1,x_2)$。这是 [26, 18] 中推导的卡尔曼滤波器结合规则的一种特殊情形（不过时间方向相反），因此可以用恒定的额外内存来存储临时变量。该算法给出参数 (A,b,C,η,J)，它们定义了如下形式的分解：

$$\begin{aligned}\bar{\psi}_{j\to 1}(x_1,x_j) &\propto \mathcal{N}(x_1;\,A\,x_j+b,C)\,\mathcal{N}_I(x_j;\eta,J)\\ &= \mathcal{N}(x_1;\,A\,x_j+b,C)\,\mathcal{N}(x_j;\,J^{-1}\,\eta,J^{-1}).\end{aligned}$$

术语 $\mathcal{N}(x_j;\ J^{-1}\,\eta, J^{-1})$ 现在可以融合到节点 j的测量模型中，方法是找到一个人工测量 z_j，以及 $\bar{H}_j$和 $\bar{R}_j$，使得 $\mathcal{N}(x_j;\ J^{-1}\,\eta, J^{-1})\ N(y_j \mid \quad H_j\,x_j + d_j, R_j) \propto N(z_j \mid H_j\,x_j, R_j)$。通过简单的矩阵和向量运算，即可在常数内存中完成此操作。总之，路径压缩只要求我们计算条件分布 $\mathcal{N}(x_1;\ A\,x_j + b, C)$ 的参数，并为节点 x_j构造人工测量模型 $N(z_j \mid H_j\,x_j, R_j)$。这将产生一个新的图，我们可以继续递归处理。

6.3 构造非标准表示

在第3节中，我们看到如何在 $O(\log D)$ 轮次内，将输入从各种其他表示转换为边列表形式。本节中，我们将利用已有的算法，展示如何在 $O(\log D)$ 轮次内将边列表转换为其他表示。令$A[(a_1,\ b_1),\ (a_2,\ b_2)\ldots(a_k,\ b_k)]$ 为一个数组，其中包含树的边列表表示，即 A 的每个索引都包含一对整数，分别表示树中的一个子节点及其父节点。

边列表 $\rightarrow$ 父指针： 只需按 a_i 对 A 排序，然后将 $(a_i,\ b_i)$ 替换为 b_i。

边列表 $\rightarrow$ BFS 遍历： 我们使用算法在 $O(\log\ D)$ 轮次内计算每个节点 a_i 的深度 d_i 。将数组 A 中的每个条目 $(a_i,\ b_i)$ 替换为 $(a_i,\ d_i)$，然后按 d_i 对数组 A 排序，即可得到 BFS 遍历。整个计算在 $O(\log D)$ 轮次内完成。

边列表 $\rightarrow$ DFS 遍历：首先，每个节点计算以该节点为根的子树大小，这可以使用我们的算法在 $O(\log\ D)$ 轮次内完成。设 v_1、v_2、$\ldots v_k$ 是节点 u 的子节点，t_i 是以 v_i 为根的子树大小。将边 (v_i, u) 标记为值 $\sum_{j<i} t_j$（注意，(v_1, u) 的标签为 0）。这是一个前缀和操作，可以在大规模并行计算中用 $O(1)$ 轮次完成。令 $l(u, v)$ 表示边 (u, v) 的标签。现在，我们可以计算每个节点 v 的 DFS 遍历时间戳，记为 $t(v)$，如下：设 $t(v) = t(\text{parent}(w)) + l(v, \text{parent}(w)) + 1$。这是一个动态规划问题，我们可以在 $O(\log D)$ 轮次内解决。按照时间戳对节点排序，即可得到 DFS 遍历。

DFS 遍历 $\rightarrow$ 括号字符串： 给定表示 DFS 遍历的数组 A，我们在 $O(\log D)$ 轮次内找出每个节点的深度。设 d_i 为树中节点 i 的深度。每台计算机从左到右扫描其负责的数组部分，并按如下方式计算其负责的字符串部分。对于从 0 开始的索引（或树中的节点）i，重复以下步骤。

- 如果 i 是树中唯一的节点，则在字符串中添加“()”并退出。
- 如果 i 是根节点，则在字符串中添加“(”。
- 如果 $d_{i+1} = d_i + 1$，则在字符串中添加“(”。
- 如果 $d_i + k = d_i$，则添加 $))\ldots)$（在字符串中添加“(”。$\overbrace{\quad}^{k次}$
- 如果 i 是数组中的最后一个索引，则在字符串中添加“))”，一个表示节点，另一个表示根节点。

7 结论

在这项工作中，我们展示了如何通过相对简单的三步法，在 MPC 模型中解决树上的一大类动态规划问题：在 $O(\log D)$ 轮次内将输入转换为标准表示，在 $O(\log D)$ 轮次内构建层次聚类，并在 $O(1)$ 轮次内解决所关注的问题。我们认为，层次聚类也将在动态规划问题范围之外得到应用。

一个重要的开放问题是，离开树之后会发生什么。自然的第一步是考虑有界树宽图。是否也能在有界树宽图中高效地找到类似的层次聚类？如果可以，在已知该层次聚类的情况下，我们是否仍能在常数时间内解决动态规划问题？

致谢

我们感谢 Alkida Balliu、Darya Melnyk 和 Dennis Olivetti 与我们进行了多次富有成效的讨论，也感谢匿名审稿人对本工作先前版本提出的宝贵意见。本研究部分由芬兰科学院资助，资助项目编号为 321901（Gupta 和 Vahidi）和 334238（Latypov 和 Pai）。

参考文献

[1] Alexandr Andoni、Zhao Song、Clifford Stein、Zhengyu Wang 和 Peilin Zhong。以对数直径轮次并行计算图连通性。载于 Mikkel Thorup 编，第*59*届 *IEEE*计算机科学基础年会（*FOCS 2018*），法国巴黎，*2018*年*10*月*7*日至*9*日，第 674–685 页。IEEE计算机学会，2018年。 doi:10.1109/FOCS.2018.00070。

[2] Sepehr Assadi、Xiaorui Sun 和 Omri Weinstein。用于在稀疏图中寻找良连通分量的大规模并行算法。载于 Peter Robinson 和 Faith Ellen 编，*2019*年美国计算机协会分布式计算原理研讨会论文集（*PODC 2019*），加拿大安大略省多伦多，*2019*年*7*月*29*日至*8*月*2*日，第461–470页。美国计算机协会，2019年。 doi:10.1145/3293611.3331596。

[3] Alkida Balliu、Sebastian Brandt、Manuela Fischer、Rustam Latypov、Yannic Maus、Dennis Olivetti 和 Jara Uitto。具有最优内存的MPC中相对于局部性的指数级加速。载于 Christian Scheideler 编，第*36*届国际分布式计算研讨会，*DISC 2022*，*2022*年*10*月*25*日至*27*日，美国佐治亚州奥古斯塔，第246卷 *LIPIcs*，第9:1–9:21页。达格斯图尔城堡—莱布尼茨信息学中心，2022年。 doi:10.4230/LIPIcs.DISC.2022.9。

[4] 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。载于 Nikhil Bansal 和 Viswanath Nagarajan（编），*2023*年*ACM-SIAM*离散算法研讨会论文集，*SODA 2023*，意大利佛罗伦萨，*2023*年*1*月*22*日至*25*日，第2589–2631页。SIAM，2023年。 doi:10.1137/1.9781611977554.ch99。

[5] MohammadHossein Bateni、Soheil Behnezhad、Mahsa Derakhshan、MohammadTaghi Haji-aghayi 和 Vahab S. Mirrokni。《简短公告：大规模树的 MapReduce 算法》。载于 Ioannis Chatzigiannakis、Christos Kaklamanis、Dániel Marx 和 Donald Sannella 编，第*45*届国际自动机、语言与程序设计研讨会（*ICALP 2018*），*2018*年*7*月*9*日至*13*日，捷克共和国布拉格，第107卷，*LIPIcs*，第162:1–162:4页。达格斯图尔城堡—莱布尼茨信息学中心，2018年。 doi:10.4230/LIPIcs.ICALP.2018.162。

[6] MohammadHossein Bateni、Soheil Behnezhad、Mahsa Derakhshan、MohammadTaghi Haji- aghayi 和 Vahab S. Mirrokni。树上的大规模并行动态规划。*CoRR*，abs/1809.03685，2018年。 arXiv:1809.03685.[7] Soheil Behnezhad、Laxman Dhulipala、Hossein Esfandiari、Jakub Lacki 和 Vahab S. Mirrokni。近乎最优的大规模并行图连通性。载于 David Zuckerman（编），第*60*届*IEEE*计算机科学基础年会，*FOCS 2019*，美国马里兰州巴尔的摩，*2019*年*11*月*9*日至*12*日，第1615–1636页。IEEE计算机学会，2019年。doi:10.1109/FOCS.2019.00095.[8] Soheil Behnezhad、Laxman Dhulipala、Hossein Esfandiari、Jakub Lacki、Vahab S. Mirrokni 和 Warren Schudy。通过远程内存访问实现的大规模并行计算。*ACM* 并行计算汇刊，8(3)：13:1–13:25，2021年。 doi:10.1145/3470631.[9] Hans L. Bodlaender。有界树宽图上的动态规划。载于 Timo Lepistö 和 Arto Salomaa（编），自动机、语言与程序设计，第*15*届国际研讨会，*ICALP88*，芬兰坦佩雷，*1988*年*7*月*11*日至*15*日，论文集，计算机科学讲义第317卷，第105–118页。施普林格，1988年。 doi:10.1007/3-540-19488-6_110.[10] Tim Bray、Jean Paoli、C. M. Sperberg-McQueen、Eve Maler 和 François Yergeau（编）。可扩展标记语言（*XML*）*1.0*（第五版）。2008年。网址： https://www.w3.org/TR/REC-xml/。[11] Sam Coy 和 Artur Czumaj。确定性大规模并行连通性。载于 Stefano Leonardi 和 Anupam Gupta（编），*STOC '22*：第*54*届*ACM SIGACT*计算理论年会，意大利罗马，*2022*年*6*月*20*日至*24*日，第162–175页。美国计算机协会，2022年。doi:10.1145/3519935.3520055.[12] Artur Czumaj、Peter Davies 和 Merav Parter。用于以低空间实现大规模并行计算去随机化的图稀疏化。*ACM* 算法汇刊，17(2)：16:1–16:27，2021年。 doi:10.1145/3451992.[13] Sanjoy Dasgupta、Christos H. Papadimitriou 和 Umesh V. Vazirani。算法。McGraw-Hill，2008年。[14] Mohsen Ghaffari、Fabian Kuhn 和 Jara Uitto。由分布式下界得到的大规模并行计算条件性困难性结果。载于 David Zuckerman（编），第*60*届*IEEE*计算机科学基础年会，*FOCS 2019*，美国马里兰州巴尔的摩，*2019*年*11*月*9*日至*12*日，第1650–1663页。IEEE计算机学会，2019年。doi:10.1109/FOCS.2019.00097.[15] Jeremy Gibbons、Wentong Cai 和 David B. Skillicorn。树累积的高效并行算法。科学计算与编程，23(1)：1–18，1994年。 doi:10.1016/0167-6423(94)00013-1。[16] Michael T. Goodrich。通信高效的并行排序。*SIAM*计算杂志，29(2)：416–432，1999年。 doi:10.1137/S0097539795294141.[17] Michael T. Goodrich、Nodari Sitchinava 和 Qin Zhang。MapReduce框架中的排序、搜索与模拟。载于 Takao Asano、Shin-Ichi Nakano、Yoshio

Okamoto 和 Osamu Watanabe（编），*算法与计算——第22届国际研讨会，ISAAC 2011，日本横滨，2011年12月5日至8日。论文集*，计算机科学讲义第7074卷，第374–383页。施普林格，2011年。doi:10.1007/978-3-642-25591-5_39.[18] Syeda Sakira Hassan、Simo Särkkä 和 Ángel F. García-Fernández。隐马尔可夫模型推断的时间并行化。*IEEE信号处理汇刊*，69：4875–4887，2021年。doi:10.1109/TSP.2021.3103338.[19] Sungjin Im、Benjamin Moseley 和 Xiaorui Sun。用于动态规划的高效大规模并行方法。载于 Hamed Hatami、Pierre McKenzie 和 Valerie King（编），*第49届美国计算机协会SIGACT计算理论年会论文集，STOC 2017，加拿大魁北克省蒙特利尔，2017年6月19日至23日*，第798–811页。美国计算机协会，2017年。 doi:10.1145/3055399.3055460.[20] Howard J. Karloff、Siddharth Suri 和 Sergei Vassilvitskii。MapReduce的计算模型。载于 Moses Charikar（编），*第21届ACM-SIAM离散算法年会论文集，SODA 2010，美国得克萨斯州奥斯汀，2010年1月17日至19日*，第938–948页。SIAM，2010年。 doi:10.1137/1.9781611973075.76.[21] Daphne Koller 和 Nir Friedman。概率图模型：原理与技术。麻省理工学院出版社，2009年。[22] Richard E. Ladner 和 Michael J. Fischer。并行前缀计算。美国计算机协会杂志，27(4)：831–838，1980年。 doi:10.1145/322217.322232.[23] Moni Naor 和 Larry J. Stockmeyer。局部可计算什么？*SIAM计算杂志*，24(6)：1259–1277，1995年。 doi:10.1137/S0097539793254571.[24] Tim Roughgarden、Sergei Vassilvitskii 和 Joshua R. Wang。洗牌操作与电路（关于现代并行计算的下界）。美国计算机协会杂志，65(6)：41:1–41:24，2018年。 doi:10.1145/3232536.[25] Simo Särkkä。贝叶斯滤波与平滑。剑桥大学出版社，2013年。[26] Simo Särkkä 和 Ángel F. García-Fernández。贝叶斯平滑器的时间并行化。*IEEE自动控制汇刊*，66(1)：299–306，2021年。 doi:10.1109/TAC.2020.2976316。