arXiv:2305.04682v1 [cs.DS] 8 May 2023

单侧纠缠图中的块交叉*

AlexanderDobler Martin Nöllenburg

算法与复杂性研究组，维也纳工业大学，奥地利维也纳
{adobler,noellenburg}@ac.tuwien.ac.at

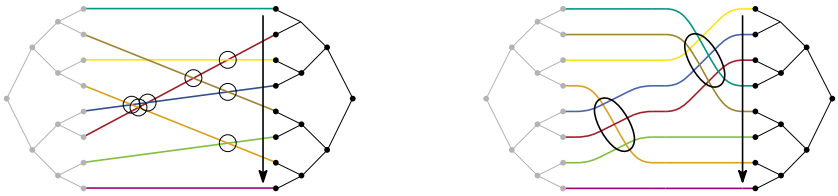
摘要。 纠缠图是两棵有根二叉系统发育树及其叶集之间的一种匹配的图示。两棵树无交叉地绘制在两侧，叶集彼此相对，分别位于两条垂直线上。我们不考虑将边的两两交叉数最小化，而是考虑将块交叉数最小化的问题，即两束线在局部相互交叉。

固定其中一棵樹後，可以根據第二棵樹的樹結構，對其葉節點進行排列。我們透過證明其 NP 完全性、提出常數因子近似演算法，以及設計固定參數演算法，完整呈現最小化單側糾纏圖中塊交叉問題的算法複雜度。我們也提出了關於非二叉樹的初步研究結果。

1 引言

纠缠图 [24] 是由两棵有根 n叶树以及它们的叶集之间以直线边绘制的一个匹配组成的图。树的绘制方式是：一棵树位于左侧，另一棵位于右侧，其叶集彼此相对，分别位于两条竖直线上（见图 1a）。纠缠图的一个重要应用是比较具有相同叶集的两棵系统发育树 [25,27]„ 这可用于研究共同成种，或比较由不同算法计算出的假设系统发育树。其他应用包括比较层次聚类中的树状图或软件层次结构 [20]。纠缠图的可读性很大程度上取决于两组叶集在竖直线上的排列顺序，因为这决定了匹配边之间交叉的数量。可能的排列顺序取决于两棵树的树结构，因此，寻找能够最小化两两交叉数量的适当排列顺序是一个非平凡问题，被称为纠缠图布局问题（TLP） [7, 12, 15]。在本文中，我们关注最小化块交叉，而不是两两边交叉；示例见图 1。这是通过放宽匹配边必须绘制为直线这一限制来实现的，而是将它们绘制为 x-单调曲线，从而能够更加灵活地移动和分组交叉点。一个 块交叉 [11] 随后被定义为受限区域 R内两个不相交边集之间的交叉，其中每个边集都形成一束局部平行曲线R；不得有其他边与其相交 R。块交叉提供了能够…

* 本研究得到维也纳科学与技术基金（WWTF）的支持[10。47379/ICT19035]。



(a) 最优解含 7 个两两交叉。

(b) 含 2 个块交叉的最优解（但有 9 次两两交叉）。

图 1：相同输入对应的两个纠缠图。左侧树的叶序固定，右侧树可以重新排列，以最小化两两交叉 (a) 或块交叉 (b)。

将多个交叉集中在一起，而不是让它们散布在整幅图中。此外，这通常能减少块交叉的数量，使其少于所需的两两交叉数量，从而降低视觉杂乱程度 [16]。我们开启了对纠缠图中块交叉的研究，并聚焦于二叉树的纠缠图，其中一棵树的叶序是固定的。

相关工作。TLP 主要针对二叉树进行研究，因为系统发育树大多是二叉树：如果两个叶集都可以排序，那么该问题已知是 NP 完全的 [15], ，即使树是完全的 [7]。不过，已有近似结果 [7], 、固定参数算法 [5,7,15], 、整数线性规划公式 [4], 和启发式算法 [23]。如果一棵树的叶序固定，那么该问题可以在多项式时间内求解 [15]; 然而，如果树不是二叉树，那么即使是这个问题也是 NP 完全的 [9]。

边捆绑是一种网络可视化技术，它将多条边组合在一起，以减少视觉杂乱。如果两个边束在局部相交，这称为捆绑交叉 [1,16]。关于一般图中捆绑交叉的相关研究成果，请参见 [22]。本质上，块交叉与捆绑交叉相同。不过，“块交叉”这一术语主要用于这样的情境：捆绑交叉完全通过排列以组合方式确定，无需拓扑结构。我们的工作与此类似，因此采用这一术语，并概述此前关于块交叉的研究：Fink 等人 [17] 研究了在预先指定的地铁网络中，如何尽可能减少地铁线路之间的块交叉，其中多条地铁线路可能沿同一条边铺设。他们考察了一般地铁网络，但主要关注路径、树和上向树等特殊情形。他们的成果主要包括近似算法和固定参数算法，因为即使是该问题限制最严格的变体也是 NP 完全的。Van Dijk 等人 [11] 在故事线的背景下研究了块交叉。他们证明了该问题的 NP 完全性、固定参数可处理性，并提出了一种近似算法。上述两项关于块交叉的研究都指出了块交叉与计算生物学中的一个问题之间的联系，该问题称为通过转置排序 [2],：通过交换排列中相邻的元素块，将一个排列变换为恒等排列；这种操作称为转置。

我们的贡献。我们研究纠缠图背景下的块交叉。具体而言，给定两棵有根 n 叶树 T_1 和 T_2，以及它们的叶集之间的一个匹配。T_1 中叶节点的顺序固定。我们要找到 T_2 中叶节点的一个排列，使其符合树结构，并且能够以最少的块交叉将匹配边从T_2 连接到 T_1。形式化的问题定义请参见第 3 节。我们主要关注二叉树 T_2：第 4 节证明，即使对于完全二叉树，该问题也是 NP 完全的。第 5 节给出 2.25-近似算法：第一个适用于一般二叉树，第二个速度更快，适用于完全二叉树。第 6 节证明，该问题关于块交叉数量这一参数是固定参数可处理的（FPT）。第 7 节证明，第 5 节中用于求得多项式时间近似算法的技术无法推广到非二叉树。接下来，我们将在下一节介绍一些预备知识。

2 预备知识

设 $\delta_{i,j}$ 为克罗内克 δ 函数，当 $i = j$ 时取值为 1，否则取值为 0。令$[n] = \{1, \ldots, n\}$。

排列。排列 $\pi : [k] \to X$ 是从 $[k]$ 到集合 X 的双射，特别地，$\pi : [k] \to [k]$ 是 $[k]$的一个排列，而 inv(π) 是其逆排列。我们将 π_i 记作 $\pi(i)$，并在需要区分多个排列时使用上标。有时我们将排列 π 写成元素序列 $(\pi_1, \ldots, \pi_k)$。我们用 Π_n 表示从 $[n]$ 到 $[n]$的所有排列构成的集合。对于排列 $\pi \in \Pi_n$ 和 $i \in [n]$，令 $\pi \ominus \pi_i$ 为 $[n-1]$ 的一个排列，具体做法是先从 π 中移除 π_i，然后将所有大于 π_i 的 π 减一。对于两个由互不相同元素构成的排列 $\pi = (x_1, \ldots, x_n)$ 和 $\pi' = (y_1, \ldots, y_m)$，我们将它们的连接记作 $\pi \star \pi'$，即 $(x_1, \ldots x_n, y_1, \ldots, y_m)$。对于两个由排列构成的集合 Π 和 Π'，我们定义 $\Pi \star \Pi' = \{\pi \star \pi' \mid \pi \in \Pi, \pi' \in \Pi'\}$。

转置。一个转置 $\tau = \tau(i, j, k) \in \Pi_n$ 且 $1 \le i < j < k \le n + 1$是排列

$$(1, \ldots, i-1, j, \ldots, k-1, i, i+1, \ldots, j-2, j-1, k, \ldots, n).$$

这个定义不同于离散数学中的经典转置，因为它源自计算生物学 [2]。假设 $\pi \in \Pi_n$。排列$\pi \circ \tau(i,\ j,\ k)$的作用是交换连续子序列 π_i、.....、π_{j-1}和 π_j、.....、π_{k-1}。排列 $\pi \in \Pi_n$中的一个块 是一个极大的连续子序列 π_i，π_{i+1}、.....、π_j，它同时也是恒等排列 id_n的连续子序列。π 中的块数记作 blocks(π)。若索引$i \in [n] \cup \{0\}$ 满足以下条件，则称其为断点

- $i = 0$ and $\pi_1 \neq 1$,
- $1 \le i \le n-1$ and $\pi_i + 1 \neq \pi_{i+1}$, or

– $i = n$ 和 $\pi_n \neq n$。

从本质上说，π 中的一个断点对应于扩展排列 $\pi^e = (0) \star \pi \star (n+1)$ 中一对相邻元素 (x, y)，满足 $x + 1 \neq y$。π 中的每个断点 i 都有一个对应的断点元素 π^e_{i+1}（其中可以包括 0）。反过来，我们称 π^e_{i+1} 对应于断点 i。令 $\mathsf{bp}(\pi)$ 表示 π中的断点数。π 的换位距离 $d_t(\pi)$ 是满足以下条件的最小数目 $k \in \mathbb{N}_0$：存在转置 $\tau^1, \ldots, \tau^k$，使得 $\pi \circ \tau^1 \circ \cdots \circ \tau^k = \mathrm{id}_n$。此时，我们称 $\tau^1, \ldots, \tau^k$为 π 的恒等换位序列。注意，恒等换位序列总是存在，因为任何排列都可以通过相邻交换变换为恒等排列（参见冒泡排序）。设 π 是一个含有 $r > 0$ 个断点的排列。那么，$\mathsf{gl}(\pi) \in \Pi_{r-1}$ 是通过将每个块“粘合”成单个元素而形成的。此外，如果 π 以 1 开头，则移除第一个块；如果 π 以 n 结尾，则移除末尾的块（见 [10]）。例如，若 $\pi = (3, 1, 2, 8, 9, 4, 5, 6, 7, 10)$，则 $\mathsf{gl}(\pi) = (2, 1, 4, 3)$。下文将贯穿全文使用的两个重要引理如下所示。

引理 1 ([2]) 对于 $\pi \in \Pi_n$，我们有 $d_t(\pi) \geq \lceil \frac{\mathsf{blocks}(\pi)-1}{3} \rceil$，并且 $d_t(\pi) \geq \lceil \frac{\pi}{\mathsf{bp}(3)} \rceil$。

引理 2 ([10]) 对于 $\pi \in \Pi_n$，$d_t(\pi) = d_t(\mathsf{gl}(\pi))$。

基因组重排中一个经过深入研究的问题是转置排序。该问题询问：给定一个排列 π 和一个整数 k，是否满足 $d_t(\pi) \leq k$。已知该问题是 NP 完全的 [8], ，作者还证明了以下结果，我们将在本文中使用。

引理 3 ([8]) 对于 $\pi \in \Pi_n$，判定 $d_t(\pi) = \mathsf{bp}(\pi)\ 3$ 是否成立是 NP-困难的。

不过，Mahajan 等人提出了一种简单的固定参数算法 [21]：首先，如果 $\mathsf{bp}(\pi) > 3k$，根据引理 1，我们可以立即报告 (π, k) 是一个否实例。否则，我们采用简单的搜索树方法，为 $\mathsf{gl}(\pi)$ 搜索长度至多为 k 的恒等换位序列（见引理 2）。由于 $\mathsf{gl}(\pi)$ 至多有 $3k$ 个元素，因此只有 $\mathcal{O}((3k)^3)$ 种可能的转置。搜索树的深度至多为 k，因为我们最多可以执行 k 次转置。因此，如果存在 $\mathsf{gl}(\pi)$ 的恒等换位序列，我们可以在 $\mathcal{O}(n(3k)^{3k})$ 时间内确定并在肯定情况下报告该序列。通过对每次转置中对应于 $\mathsf{gl}(\pi)$ 元素的 π 块进行转置，可以轻松地将该转置序列转换为 π 的恒等换位序列。

树 我们只考虑有序有根树 T。令 $\mathsf{root}(T)$ 表示 T 的根。令 $\mathsf{leaf}(T)$ 表示 T 的叶节点集合。对于 $v \in V(T)$，令 $\mathsf{depth}(v)$ 表示 $\mathsf{root}(T)$ 与 v 在 T 中之间最短路径的长度。对于 $v \in V(T)$，令 $T(v)$ 表示以 v 为根的 T 的子树。对于内部节点 $v \in V(T)$，令 $\mathsf{ch}(v)$ 表示 v 的子节点集合，并令 $\mathsf{par}(v)$ 表示 v 的父节点。此外，令 $\mathsf{anc}_T(v)$ 表示 v 在 T 中的祖先集合。对于两个不同的顶点 $v, w \in V(T)$，令 $\mathsf{lca}(v, w)$ 表示 v 和 w 的最近公共祖先。若两个顶点 $v, w \in V(T)$ 是兄弟节点，当且仅当

它们是同一个顶点的子节点。如果 T 是二叉树，则将两个子节点记作 lc(v) 和 rc(v)。

有根树 T编码了其叶节点的一组排列 $\Pi(T)$，这些排列可以通过置换内部节点的子节点得到：具体来说，设 $\Pi(T(v)) = \{(v)\}$if $v \in$leaf(T)。如果 $v \notin$leaf(T)，令 ch(v) $= \{w_1, \ldots, w_k\}$，并定义

$$\Pi(T(v)) = \bigcup_{\psi \in \Pi_k} \Pi(T(w_{\psi_1})) \star \Pi(T(w_{\psi_2})) \star \cdots \star \Pi(T(w_{\psi_k})).$$

如果 $\pi \in \Pi(T)$，我们称 π与 T一致。

3 纠缠图中的块交叉

本节将对我们所研究的问题作出准确的定义。重申一下，给定两棵树 T_1 和 T_2，它们各有 n 个叶节点；两棵树的叶集之间存在一个匹配，并且 T_1的叶序固定。注意，不失一般性，我们可以假设两棵树的叶集都标记为 $[n]$，且匹配边连接标记有相同整数的叶节点。此外，通过重新标记，我们可以假设 T_1 的叶序为恒等排列 id$_n$。现在，我们要寻找：(1) T_2 叶节点的一个排列 π，该排列符合 T_2 的结构；(2) 一系列块交叉，将匹配边从 T_1 移动到 T_2。(1) 意味着我们要寻找一个排列 $\pi \in \Pi(T_2)$。由于一次块交叉只会改变两块匹配边的垂直顺序，(2) 要求找到一个边块相邻块之间的交换序列——这是一种纯粹的组合过程。此外，注意到每次块交叉都可以用边垂直顺序上的一个转置 τ 来建模。因此，我们不必寻找块交叉序列，而可以等价地寻找一个转置序列，将 π 变为恒等排列。这引出了我们问题的以下判定版本，其中 $T := T_2$。

单棵树块交叉最小化（OTBCM）实例：一棵有根树 T，具有叶节点(T) $= [n]$ 和一个正整数 k。问题：是否存在一个排列 $\pi \in \Pi(T)$，使得 $d_t(\pi) \leq k$?

如果是一个是实例，我们的算法将生成一个见证，即一个排列 $\pi \in \Pi(T)$ 和一个恒等换位序列 $\tau^1, \ldots, \tau^\ell$，其对应于 π，并满足 $\ell \leq k$。在接下来的章节中，我们将研究这一问题的算法复杂度。我们首先介绍假设 T 是一棵二叉树时所得到的结果。

4 NP 困难性

OTBCM 隐含地包含一个子问题：通过较少的转置序列对一个排列进行排序。由于通过转置排序是 NP 完全的[8], ，这表明 OTBCM 也是 NP 完全的。我们证明了这一点适用于输入树为完全二叉树的受限情形。不过，证明

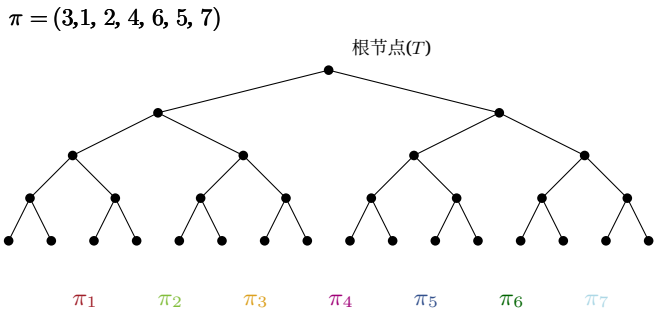


图2：定理1的证明中关于 T 的一个示例，展示了排列$\pi = (3,1,2,4,6,5,7)$。

并不像 OTBCM 与通过转置排序之间的关系所暗示的那样直接。主要思路是针对一个输入排列 π 构造一棵树 T，使得存在 $\pi' \in \Pi(T)$，满足 $\mathrm{gl}(\pi) = \mathrm{gl}(\pi')$，并且$\pi'$ 是 $\Pi(T)$ 中唯一能使断点数最小化的排列。由于判定 $d_t(\pi)$ 是否等于下界 $\mathrm{bp}(\pi)/3$（引理 3）本身已是 NP 困难的，这表明 OTBCM 是 NP 困难的。

定理1。 OTBCM 对于完全二叉树是NP-完全的。

令 π 为由 $2^p - 1$ 个元素组成的任意排列，其中 $p \in \mathbb{N}$。我们定义一个带标号的完全二叉树 T，它有 2^{p+1} 个叶节点。我们对 T 的叶节点进行标记，使最左侧的叶节点标记为 1，最右侧的叶节点标记为2^{p+1}。其余叶节点从左到右依次标记为

$$2\pi_1, 2\pi_1 + 1, 2\pi_2, 2\pi_2 + 1, \ldots, 2\pi_{2^p-2}, 2\pi_{2^p-2} + 1, 2\pi_{2^p-1}, 2\pi_{2^p-1} + 1.$$

T 的叶节点定义了排列 $\pi^I = \pi^I(T)$，具体如下。我们有 $\pi^I(T) = \pi^I(\mathrm{root}(T))$，其中

$$\pi^I(v) = \begin{cases} (v), & \text{if } v \text{ is a leaf,} \\ \pi^I(\mathsf{lc}(v)) \star \pi^I(\mathsf{rc}(v)), & \text{otherwise.} \end{cases}$$

注意，$\mathrm{bp}(\pi^I) = \mathrm{bp}(\pi)$。 T的示例请参见图2。

接下来我们证明，在 $\Pi(T)$ 的所有排列中，π^I 具有最少的断点。

引理 4 对于所有满足 $\pi' \neq \pi^I$ 的 $\pi' \in \Pi(T)$，我们有 $\mathrm{bp}(\pi') > \mathrm{bp}(\pi^I)$。

证明。 任取满足 $\pi' \neq \pi^I$ 的 $\pi' \in \Pi(T)$。令 $X = \{x_1, \ldots, x_\ell\}$ 为 π^I 中未出现在 π' 中的断点元素集合。此外，令 $Y = \{y_1, \ldots, y_m\}$ 为出现在 π' 中但未出现在 π^I 中的断点元素集合。我们将通过证明 X 中的每个断点元素都会导致 Y 中存在某些断点元素，来证明 $m > \ell$。

首先注意，所有元素 $x \in X$ 都是奇数，且是其父节点的左子节点。由于 x 不是 π' 中的断点元素，因此它必须在 π' 中出现在 $x+1$ 之前，其中 $x+1$ 是偶数，且是其父节点的右子节点。因此，在 π' 中，x 和 $x+1$ 的父节点的子节点顺序相反。这意味着 $x-1$ 和 $x+1$ 是 Y 中的断点元素（对于 $x=1$ 的情况，请注意 0 也可以是断点元素）。示例见图2：如果 $9 \in X$，那么它必须在 π' 中出现在 10 之前。因此，9 和 10 的父节点必须翻转，导致 8 和 10 成为 π' 中的断点元素。

现在，对于任意一个所有元素均为奇数的集合 X，我们有 $Y \supseteq \{x-1, x+1 \mid x \in X\}$。由此可得 $|X| < |Y|$，因为如果 $|X| = 1$，那么 $|Y| = 2$；否则，根据集合 $|X|$ 的大小归纳可得 $|Y| \geq |X| + 1$。这正是我们想要证明的。

□

定理*1*的证明。 由于这是一个带有多项式规模见证的纯组合问题，因此 NP 成员资格显而易见。对于一个排列 π，判定 $d_t(\pi) = \mathsf{bp}(\pi)_3$ 引理 3。我们将证明，给定任意排列 π，我们可以构造一个 OT- BCM 实例 (T, k)，使得 $d_t(\pi) = \mathsf{bp}(\pi)_3$ 当且仅当 (T, k) 是一个是实例。这足以证明 OTBCM 是 NP 困难的。

令 π 为任意排列。根据引理 2，我们可以假设 π 的长度为 $2^p - 1$，其中 $p \in \mathbb{N}$: 如果 π 的长度不是 $2^p - 1$，例如大小为 $2^p - 1 - \ell$，我们只需将 π 的最后一个元素 e 替换为块 e、$e+1$、$\ldots$、$e+\ell$，并将所有其他 $e' > e$ 替换为 $e' + \ell$。我们有，新的排列 π^a 具有相同数量的断点，并且 $d_t(\pi) = d_t(\pi^a)$，正如 $\mathsf{gl}(\pi) = \mathsf{gl}(\pi^a)$ 一样。因此，判定 $d_t(\pi) = \mathsf{bp}(\pi)_3$ 等价于判定 $d_t(\pi^a) = \mathsf{bp}(\pi)_3$。通过适当选择 p，π^a 的大小至多为 π 的两倍。今后，我们假设 π 的大小为 $2^p - 1$。

令 T 为上述所述的完全二叉树，并令 $k = \mathsf{bp}(\pi)_3 = \mathsf{bp}(\pi^I(T))_3$。现在我们证明所述的等价关系。若 $d_t(\pi) = \mathsf{bp}(\pi)_3$，则 (T, k) 是一个 YES 实例，因为 $d_t(\pi^I(T)) = d_t(\pi) = \mathsf{bp}(\pi)_3 = k$。等式 $d_t(\pi^I(T)) = d_t(\pi)$ 成立，因为 $\mathsf{gl}(\pi^I(T)) = \mathsf{gl}(\pi)$。

反过来，如果 $d_t(\pi) \neq \mathsf{bp}(\pi)_3$，那么 $d_t(\pi^I(T)) > k$（根据引理 1，距离不可能更小）。对于所有其他满足 $\pi' \neq \pi^I(T)$ 的 $\pi' \in \Pi(T)$，根据引理 4，我们有 $\mathsf{bp}(\pi') > \mathsf{bp}(\pi^I(T))$。根据引理 1，$d_t(\pi') \geq \lceil \mathsf{bp}(\pi')_3 \rceil > \mathsf{bp}(\pi^I(T))_3 = k$。总而言之，我们有 $\forall \pi \in \Pi(T) : d_t(\pi) > k$，且 (T, k) 是一个否实例。

□

5 近似

我们针对二叉树上的 OTBCM 提出两种近似算法。这些算法基于 Walter 和 Dias [28] 提出的多项式算法，该算法可在 $\mathcal{O}(n^2)$ 时间内，通过至多 $3 \cdot 4\mathsf{bp}(\pi)$ 次转置对长度为 n 的排列π 进行排序。结合引理1 的下界，这意味着 Sort- 的近似比为 2.25 的近似算法。

通过转置[1]。因此，我们的算法需要找到使断点数量最少的排列$\pi \in \Pi(T)$。将 Walter 和 Dias 的算法应用于该排列，可为 OTBCM 提供 2.25-近似。我们提出两种用于最小化断点数量的算法。对于二叉树，我们提出一种 $\mathcal{O}(n^3)$ 算法；对于完全二叉树，我们可以将其改进为 $\mathcal{O}(n^2)$。在这两种算法中，我们首先讨论如何最小化块的数量，而不是断点数量，然后说明如何将算法调整为针对断点。请注意，对于任何排列，块数与断点数最多相差一，因此，最小化块数也能得到类似的近似结果。

二叉树。 我们先介绍一种适用于二叉树的算法。从本质上说，寻找具有最少块数的 $\pi \in \Pi(T)$，与以下已研究过的问题密切相关：给定一棵有 n 个叶节点的完全二叉树 T，以及任意距离函数 $d\colon \mathsf{leaf}(T) \times \mathsf{leaf}(T) \to \mathbb{R}$，求使 $\sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1})$ 最小的 $\pi \in \Pi(T)$。对 $i, j \in \mathsf{leaf}(T)$，定义 $d(i,j) = (1 - \delta_{i+1,j})$，便能准确刻画我们的问题（注意，$d$ 并不对称）。Bar-Joseph 等人 [3] 针对 T 为二叉树且距离函数 d 对称的情形，提出了一种算法。我们无法直接应用他们的算法，因此将展示如何调整该算法以解决我们的问题；其中很大一部分与原算法相同。其核心思想是采用自底向上的动态规划，在固定最左和最右叶节点的情况下，计算子树叶节点的最优排序。

定理 2。 设 T 是一棵具有 叶节点$(T) = [n]$ 的有根二叉树。那么，可以在 $\mathcal{O}(n^3)$ 时间和 $\mathcal{O}(n^2)$ 空间内计算出一个使块数（断点数）最小化的置换 $\pi \in \Pi(T)$ 。

证明。我们展示如何找到使块的数量最小的排列，并在证明末尾讨论如何使断点的数量最小。我们按照 [3] 的方法，通过自底向上动态规划来求解。对于每个顶点 $v \in V(T)$ 以及每对叶节点 $i, j \in \mathsf{leaf}(T)$，使得 $v = \mathsf{lca}(i, j)$，我们计算值 $B(v, i, j)$，它表示满足 i 是该排列的第一个元素且 j 是最后一个元素的 $\mathsf{leaf}(T(v))$ 排列所含块数的最小值。此过程通过自底向上动态规划完成。显然，对于 $v \in \mathsf{leaf}(T)$，$B(v, v, v) = 1$。然后，对于所有其他满足 $w = \mathsf{lc}(v)$、$x = \mathsf{rc}(v)$、$i \in \mathsf{leaf}(T(w))$ 且 $j \in \mathsf{leaf}(T(x))$ 的 $v \in V(T)$，

$$B(v,i,j) = \min\{B(w,i,h) + B(x,\ell,j) + (1-\delta_{h+1,\ell}) \mid h \in \mathsf{leaf}(T(w)), \ell \in \mathsf{leaf}(T(x))\}.$$

交换 w 和 x 的角色，可以等价地计算出值 $B(v, j, i)$。不难看出，

$$\min_{\pi' \in \Pi(T)} \mathsf{blocks}(\pi') = \min\{B(\mathsf{root}(T), i, j) \mid i, j \in \mathsf{leaf}(T), \mathsf{lca}(i,j) = \mathsf{root}(T)\},$$

[1] 存在一种基于断点、可实现 2-近似的更优近似算法，但作者没有说明其运行时间 [14]。

并且，实现该块数量的一个排列也可以通过在 v 处保留使 $B(v,i,j)$ 最小化的 π 叶节点$(T(w))$ 和 $\ell \in$叶节点$(T(x))$ 来获得。

直接计算 $B(v,\ i,\ j)$ 会得到一个 $\mathcal{O}(n^4)$ 算法，但正如 Bar-Joseph 等人 [3] 所述，还有一种更快的方法。为了计算 $B(v,\ i,\ j)$，我们保存中间值 $\mathrm{Temp}(i,\ \ell)$，其计算方式如下

$$\mathrm{Temp}(i,\ell) = \min_{h \in T(w)} B(w,i,h) + (1 - \delta_{h+1,\ell}).$$

然后，我们可以如下计算 $B(v,\ i,\ j)$

$$B(v,i,j) = \min_{\ell \in T(x)} \mathrm{Temp}(i,\ell) + B(x,\ell,j).$$

两项计算均需 $\mathcal{O}(n)$ 时间，$B(v,\ j,\ i)$ 也可以用类似方法计算。对于每一对 $i,\ j \in$叶节点(T)，$B(v,\ i,\ j)$ 和 $B(v,\ j,\ i)$ 的值只会针对一个 $v =$ 最近公共祖先$(i,\ j)$ 计算一次，计算耗时为 $\mathcal{O}(n)$。因此，总运行时间为 $\mathcal{O}(n^3)$，总空间复杂度为 $\mathcal{O}(n^2)$。

上述算法可以调整为最小化断点数，而非块数：算法完成后，我们可以查看所有值 $B($根节点$(T)\ i,\ j)$。如果 i 不为 1，就将该值加一。如果 j 不为n，就将该值加一。根据这些新的 $B($根节点$(T)\ i,\ j)$ 值，我们可以求出最小断点数。达到该断点数的排列 $\pi \in \Pi(T)$ 可以用与上述等效的方法得到。□

完全二叉树。如果处理的是完全二叉树，我们可以给出一种更快的算法来最小化块（断点）。该算法基于 Brandes 的一种算法 [6]，用于解决上述问题：寻找使 $\sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1})$ 最小的 $\pi \in \Pi(T)$。Brandes 的算法可在 $\mathcal{O}(n^2 \log n)$ 时间和 $\mathcal{O}(n)$ 空间内解决完全二叉树上的问题。这已经为我们的完全二叉树问题给出了一种算法 T：令$d(i,j) = (1-\delta_{i+1,j})$，其中 $i,j \in$叶节点(T)。但由于我们处理的并非任意距离函数，因此可以找到一种更快的算法。

定理 3。设 T 为一棵具有 叶节点$(T) = [n]$ 且满足 $n = 2^k$的完全有根二叉树。那么，可以在 $\mathcal{O}(n^2)$ 时间和 $\mathcal{O}(n\log\ n)$ 空间内，计算出一个使块（断点）数量最小化的排列 $\pi \in \Pi(T)$。

我们首先说明块的算法。主要思路是，我们不必自底向上应用动态规划，而可以从左到右构造排列，同时应用动态规划。

Brandes [6] 指出，将 T 的一个叶节点固定在排列 $\pi \in \Pi(T)$ 的特定位置，就确定了一个将前序叶节点和后序叶节点划分开来的划分。对于 $p \in [n]$，令 $b^p = (b_k^p \ldots b_1^p)$ 为与数字 $p-1$ 对应的 k 位字符串。对于 $i \in$叶节点(T)，我们递归地定义i的第 j 个父节点 $\mathrm{par}(i,j)$：$\mathrm{par}(i,0) = i$，且 $\mathrm{par}(i,j) = \mathrm{par}(\mathrm{par}(i,j-1))$. 此外，对于 $i \in$叶节点(T) 和 $1 \le j \le k$，我们将前序函数 $\mathrm{prec}(i,j)$ 定义为

$$\mathsf{prec}(i,j) = \begin{cases} \mathsf{lc}(\mathsf{par}(i,j)), & \text{if } i \in \mathsf{leaf}(T(\mathsf{rc}(\mathsf{par}(i,j)))) \\ \mathsf{rc}(\mathsf{par}(i,j)), & \text{otherwise.} \end{cases}$$

par(i,j) 和 prec(i,j) 的值可以在 $\mathcal{O}(n\log n)$ 时间内预先计算，并使用 $\mathcal{O}(n\log n)$ 空间存储。值 rob$(p) = \min\{i\colon 1 \le i \le k, b_i^p = 1\}$ 是对应于 $p-1$ 的二进制数中为 1 的最右侧位。

Brandes 给出的关键见解表述为以下引理。

引理 5 ([6]) 如果叶节点 $i \in [n]$ 固定在位置 $p \in [n]\setminus\{1\}$，那么恰好叶节点$(T(\mathsf{prec}(i, \mathsf{rob}(p))))$ 可以排在 i 之前。

本质上，如果叶节点 i 固定在 $\pi \in \Pi(T)$ 的特定位置 p，我们就能准确知道哪些叶节点 $j \in \mathsf{leaf}(T)$ 可以在 π 中排在它之前。此外，这些叶节点 j 恰好对应于 leaf$(T(\mathsf{prec}(i, \mathsf{rob}(p))))$. 由此得到定理 3 的算法。

定理 3 的证明。与 Brandes 的算法一样 [6]，我们计算 opt(i,p) 的值，它对应于以叶节点 i 结尾、长度为 p 的前缀所需的最小块数。计算 opt(i,p) 时，只需查看所有 opt$(j,p-1)$ 的值，其中叶节点 j 在 i 固定于位置 p 时可以排在 i 之前。但在我们的情形中，除 $j = i-1$ 外，对于所有 j， $d(j,i)$ 均为 1。因此，我们可以将 $\min_j \mathsf{opt}(j,p-1)$ 存储在与这些叶节点 j 的最低公共祖先对应的内部节点处。此外，还必须检查 $i-1$ 可以排在 i 之前的情形。形式化地，我们计算如下。对于内部节点 v，令 opt$(v,p) = \min(\mathsf{opt}(\mathsf{lc}(v)\ p), \mathsf{opt}(\mathsf{rc}(v)\ p))$。显然，对于所有 $i \in [n]$, opt$(i,1) = 1$。对于 $i \in \mathsf{leaf}(T)$ 和 $p > 1$，我们可以如下计算 opt(i,p)

$$\mathsf{opt}(i,p) = \begin{cases} \mathsf{opt}(\mathsf{prec}(i,\mathsf{rob}(p)), p-1) + 1, & \text{if } i-1 \notin \mathsf{leaf}(T(\mathsf{prec}(i,\mathsf{rob}(p)))) \\ \min(\mathsf{opt}(\mathsf{prec}(i,\mathsf{rob}(p)), p-1) + 1, \mathsf{opt}(i-1,p-1)) & \text{otherwise.} \end{cases}$$

情况 $i-1 \in$ 叶节点$(T(\mathsf{prec}(i,\ \mathsf{rob}(p))))$ 恰好对应于 $i-1$ 位于 i 之前的可能性。可以通过预先计算所有 $2 \le i \le n$ 的 lca$(i-1,\ i)$ 来检查此条件。显然，所有值都可以在 $\mathcal{O}(n^2)$ 时间内计算出来，并且

$$\min_{\pi' \in \Pi(T)} \mathsf{blocks}(\pi') = \min_{i \in [n]} \mathsf{opt}(i,n).$$

利用 opt 数组中的值，可以立即计算排列 $\pi \in \Pi(T)$（含有 blocks$(\pi) = \min_{i\in[n]} \mathsf{opt}(i,n)$），其空间和时间复杂度均为 $\mathcal{O}(n^2)$。不过，如 Brandes [6] 所示，空间复杂度甚至可以降至 $\mathcal{O}(n\log n)$。这里我们只介绍高层思路，完整描述见 [6]。首先，在计算值 $\min_{\pi'\in\Pi(T)}$ blocks(π') 时，无须存储所有 $p \in [n]$ 对应的 opt(i,p) 和 opt(v,p) 值。我们只需让 p 从 1 迭代到 n，并且只保留 p 和 $p-1$ 对应的 opt 值。接下来的核心思路是，只存储最优排列中间位置的元素，并在此边界条件下递归地确定前半部分和后半部分的最优排列。这有些类似于单枢轴快速排序。时间复杂度为 $T(n) = 2T(n/2) + n^2$，可化简为 $T(n) = \mathcal{O}(n^2)$。这种方法也适用于我们的改编版本，因为递归过程会将搜索空间分成两个相等的部分，且两部分仍然都是完全二叉树。因此，opt 值仍可存储在内部节点中。空间复杂度主要由存储 par(i,j) 和 prec(i,j) 所需的 $\mathcal{O}(n\log n)$ 决定。

最后，只需在算法开始时设置 opt(1,1) $= 0$，并在算法结束时设置 opt(n, n) $=$opt(n, n) $- 1$，即可将算法改编为针对断点而非块的情况。□

6 固定参数可处理算法

本节将介绍一种针对任意二叉树的 OTBCM 固定参数可处理算法，其参数为转置的数量。如果存在 OTBCM 的见证，该算法还能在相同时间内给出见证。

首先，我们陈述两个用于归约规则的引理。第一个引理表明，从排列中删除一个元素不会增加换位距离。

引理 6 设 $\pi \in \Pi_n$ 和 $i \in [n]$。那么 $d_t(\pi \ominus \pi_i) \leq d_t(\pi)$。

证明。设 $\tau^1, \ldots, \tau^k$是 π的一个恒等换位序列。对于 $\ell \in [k]$，令 $\upsilon^\ell = \tau^\ell \ominus$ inv(($\pi \circ \tau^1 \circ \cdots \circ \tau^{\ell-1}$))($\pi_i$)。容易看出，所有 υ^ℓ都是转置；附录 A 中的穷举分类表明

$$(\pi \circ \tau^1 \circ \cdots \circ \tau^\ell) \ominus \pi_i = (\pi \ominus \pi_i) \circ \upsilon^1 \circ \cdots \circ \upsilon^\ell. \tag{1}$$

因此，($\pi \ominus \pi_i$) $\circ \upsilon^1 \circ \ldots \upsilon^k =$恒等$_{n-1}$。本质上，恒等换位序列$\tau^1$、.....、$\tau^k$对于 π是在 $\pi \ominus \pi_i$上通过换位序列 υ^1、.....、υ^k进行模拟的。因此， d_t($\pi \ominus \pi_i$) $\leq d_t$(π)。

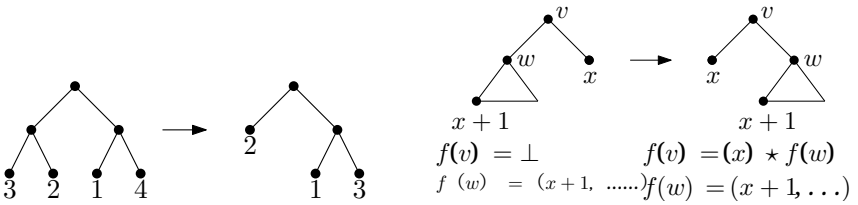
这可以用来说明，就换位距离而言，在某种意义上，将元素 $x \in \pi$ 移到排列中的 $x-1$ 之后或 $x+1$ 之前总是“安全的”。下面给出形式化定义和引理。

定义 1 设 $\pi \in \Pi_n$、 $i \in [n]$和 $\pi_i = x$。将 前置邻接(π、 x) 定义为从 π 得到的排列：当 $x = 1$时，将 x 移到开头；否则，将其移到 $x - 1$之后的位置。形式化地，

$$\text{adjbef}(\pi, x) = \begin{cases} (x, \pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n) & \text{if } x = 1 \\ (\pi_1, \ldots, \pi_{j-1}, x-1, x, \pi_{j+1}, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n) & \\ \qquad \text{if } x \neq 1 \text{ and } i > j = \text{inv}(\pi)(x-1) & \\ (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots \pi_{j-1}, x-1, x, \pi_{j+1}, \ldots, \pi_n) & \\ \qquad \text{if } x \neq 1 \text{ and } i < j = \text{inv}(\pi)(x-1). & \end{cases}$$

类似地，将 后置邻接(π, x) 定义为由 π 经过排列得到的结果：如果满足 $x = n$，则将 x 移至末尾；否则，将其移至 $x + 1$ 之前的位置。形式化地，

$$\text{adjaft}(\pi, x) = \begin{cases} (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n, x) & \text{if } x = n \\ (\pi_1, \ldots, \pi_{j-1}, x, x+1, \pi_{j+1}, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n) & \\ \qquad \text{if } x \neq n \text{ and } i > j = \text{inv}(\pi)(x+1) & \\ (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots \pi_{j-1}, x, x+1, \pi_{j+1}, \ldots, \pi_n) & \\ \qquad \text{if } x \neq n \text{ and } i < j = \text{inv}(\pi)(x+1). & \end{cases}$$



(a) 归约规则 1 的示例。 (b) 归约规则 2 的示意图。

图 3：FPT 算法的归约规则。

引理 7 令 $\pi \in \Pi_n$、 $i \in [n]$ 和 $\pi_i = x$。则 d_t(前置邻接$(\pi, x)) \leq d_t(\pi)$ 和d_t(后置邻接$(\pi, x)) \leq d_t(\pi)$。

证明。我们展示 d_t(前置邻接$(\pi, x)) \leq d_t(\pi)$、 d_t(后置邻接$(\pi, x)) \leq d_t(\pi)$ 可以用一种等价的方式展示。如果前置邻接$(\pi, x) = \pi$，则结论显然成立。因此，假设前置邻接$(\pi, x) \neq \pi$。在这种情况下，我们有 gl(前置邻接$(\pi, x))$ =gl$(\pi \ominus x)$。此外，根据引理 2， d_t(前置邻接$(\pi, x))$ $= d_t$(gl(前置邻接$(\pi, x)))$ $= d_t$(gl$(\pi \ominus x))$，且 $d_t(\pi)$ =d_t(gl$(\pi))$。因此，由引理 6 可得结论。□

直观地说，本节末尾定理 4 中的算法首先穷尽地应用基于引理 7 的归约规则，并移除那些我们已知其在最优解的树中的排列方式的叶节点。然后，它自底向上应用一种分支方案，对内部节点子节点的两种排列方式分别进行分支，从而能够界定所得搜索树的深度。在这一分支过程中，还会应用另一条与第一条类似的归约规则；而在搜索树的叶节点处，则应用用于通过转置排序的 FPT 算法。

我们首先介绍第一条归约规则（示例见图 3a）。该规则基于引理 7，允许我们从输入树 T 中删除一对满足以下条件的子节点：它们互为兄弟节点，且其差值为 1。

归约规则 1 如果有两个叶节点 $x, y \in$叶节点(T) 是兄弟节点，满足 $x + 1 = y$，则 *(1)* 从 T 中删除对应于 x 和 y 的叶节点，*(2)* 将它们的父节点（现在是一个叶节点）替换为 x，以及 *(3)* 对于每个满足 $z > y$ 的叶节点$z \in$叶节点(T)，设置 $z = z - 1$。

如果叶节点$(T) = [n]$，那么在应用归约规则 1 后，对于新树 T'，我们有叶节点(T') $= [n - 1]$。归约规则 1 的安全性可直接由引理 7 和引理 2 得出。此外，如果我们得到一个见证排列 $\pi' \in \Pi(T')$，满足 $d_t(\pi') \leq k$，那么可以很容易地将该排列转换为排列 $\pi \in \Pi(T)$，使其满足 $d_t(\pi) \leq k$：将排列中所有大于 x 的元素加 1，并将元素 x 替换为数对 $(x, x + 1)$。随后，可以在 π 中模拟 π' 的恒等换位序列。

穷尽地应用归约规则 1 后，我们采用一种搜索树算法，使得在每个搜索树节点中，对应于内部节点 v 的集合 leaf$(T(v))$ 的排序有一部分已经确定。形式上，在每个搜索

树节点，我们被赋予一个函数 f，其定义域为 $V(T)$，满足以下条件之一：$f(v) = \perp$，或 $f(v) \in \Pi(T(v))$。此外，如果 $f(v) \in \Pi(T(v))$ 且 v 不是叶节点，那么对于 v 的子节点 w、x，有 $f(w) \neq \perp$ 且 $f(x) \neq \perp$，并且 $f(v)$ 与 $f(w)$ 和 $f(x)$ 一致，即 $f(v) = f(w) \star f(x)$ 或 $f(v) = f(x) \star f(w)$。排列 $\pi \in \Pi(T)$ 与 f 一致，当且仅当对于所有满足 $f(v) \neq \perp$ 的 $v \in V(T)$，都存在某个 ϕ、ψ，使得 $\pi = \phi \star f(v) \star \psi$，其中该排列也可以是空排列。我们将与搜索树节点相关联的问题 (T, f) 称为 OTBCM-Extension。问题是，是否存在一个与 f 一致的排列 $\pi \in \Pi(T)$，使得 $d_t(\pi) \leq k$。在我们的搜索树根节点以及与 OTBCM 等价的初始问题中，对于所有叶节点 $x \in \mathsf{leaf}(T)$，$f(x) = (x)$，而对于所有内部节点，$f(v) = \perp$。

一个关键观察是，f 已经能告诉我们，任何与 f 一致的排列 $\pi \in \Pi(T)$ 中断点的数量。于是，我们令 $\mathsf{bp}(f)$ 表示任意与 f 一致的排列 $\pi \in \Pi(T)$ 中会构成断点的 $x, y \in$ 叶节点对的数量。注意，$\mathsf{bp}(f)$ 可以在多项式时间内确定：断点对 (x, y) 满足 $x + 1 \neq y$ 对应于某些排列 $f(v) \neq \perp$ 中的相邻元素对。

现在介绍搜索树算法执行过程中应用的下一条归约规则（示意见图 3b），以及归约规则不适用时采用的分支规则。

归约规则 *2* 设 $v \in V(T)$ 是一个内部节点，其两个子节点为 x 和 w，满足 $f(v) = \perp$，x 是叶节点，$f(w) \neq \perp$，且 $f(w)(1) = x + 1$ 或 $f(w)(|T(w)|) = x - 1$。如果 $f(w)(1) = x + 1$，则令 $f(v) = (x) \star f(w)$。否则，令 $f(v) = f(w) \star (x)$。

归约规则 2 的安全性同样由引理 7 得出，因为它本质上是在任何与 f 一致的排列中，将 x 移到 $x+1$ 之前或 $x-1$ 之后。注意，如果 $f(w)(1) = x+1$ 且 $f(w)(|T(w)|) = x-1$，那么 v 的子节点如何排序并不重要，因为由此得到的、与 f 一致的排列 $\pi \in \Pi(T)$ 在胶合操作方面是等价的。穷尽应用归约规则 2 后，即可应用主要分支规则。

分支规则 1 设 $v \in V(T)$ 为一个内部节点，其两个子节点为 u 和 w，满足 $f(v) = \perp$、$f(u) \neq \perp$，且 $f(w) \neq \perp$。创建两个新分支 (T, f^α) 和 (T, f^β)，其中

$$f^\alpha(x) = \begin{cases} f(u) \star f(w) & \text{if } x = v, \\ f(x) & \text{otherwise,} \end{cases} \qquad f^\beta(x) = \begin{cases} f(w) \star f(u) & \text{如果 } x = v, \\ f(x) & \text{否则。} \end{cases}$$

分支规则 1 本质上是在由 f 确定了以两个子节点为根的子树的相应排序后，尝试以两种可能的方式对子节点进行排序。如果 u 和 w 都是叶节点，那么

$$\mathsf{bp}(f^\alpha) = \mathsf{bp}(f^\beta) = \mathsf{bp}(f) + 1, \tag{2}$$

因为配对 (u, w) 或 (w, u) 将产生一个断点。这是因为归约规则 1 已经被穷尽地应用。此外，如果 u 和 w 中只有一个是

算法 1： 用于 OTBCM-Extension 的递归固定参数可处理算法。

```
递归式 OTBCM 扩展(T, f):
    对 (T, f) 穷尽地应用归约规则 2;
    如果bp(f) > 3k 则
        返回 false
    如果 f（根节点（T））≠ ⊥ 则
        π ← f（根节点（T））;
        如果 d_t（π）≤ k 则 // O（n（3k）^{3k}）固定参数可 tractable 算法
            返回 π以及恒等换位序列（τ^1, ....., τ^ℓ），其中 ℓ ≤ k
        else
            返回 false
    对于 （T′, f′）∈分支规则 1（T）执行 // （T′, f′）遍历这两个
     通过对 T 应用分支规则 1 创建的分支
        ans ← 递归式 OTBCM 扩展(T′, f′);
        如果答案 ≠为 false 则
            <escape>return ans;</escape>
    返回 false;
```

叶节点，并且归约规则 2 已经被穷尽地应用，那么公式 (2) 也成立：例如，如果 u是叶节点，那么根据 u 和 w的选取顺序，$(u, f(w)(1))$ 或 $(f(w)(|\text{叶节点}(T(w))|), u)$ 将成为一个断点。这一发现将使我们能够限制搜索树的深度。

有了分支规则 1，我们现在可以给出描述该算法的定理。

定理 4。 OTBCM 可在 $\mathcal{O}(2^{6k}\cdot(3k)^{3k}\cdot n^{\mathcal{O}(1)})$ 时间内求解有根二叉树问题，即，对于以转置次数（=块交叉） k为参数的有根二叉树，OTBCM 是 *FPT*。

证明。 我们为 OTBCM 给出一种搜索树算法，该算法在成功的情况下还会提供一个见证。该搜索树算法假设已经穷尽地应用了归约规则 1，并以算法 1 中的递归函数形式给出。归约规则 1 可以在线性时间内实现，并且如前所述，应用归约规则后所得实例的见证可以转换为应用归约规则前的见证。此外，通过对所有叶节点 $x \in \text{leaf}(T)$ 设置 $f(x) = (x)$，并对所有内部节点设置 $f(v) = \perp$，OTBCM 的实例 T 被转换为 OTBCM-Extension 的实例 (T, f)。随后调用算法 Recursive-OTBCM-Extension (T, f)，若失败则返回 **false** 。若成功，则返回 $\pi \in \Pi(T)$ 以及一个恒等换位序列 $\tau^1, \ldots, \tau^\ell$，其中 $\ell \le k$ 对于 π。

正确性。在每个搜索树节点（算法 1 的函数调用）中，首先穷尽应用规约规则 2。根据引理 7，该规则的安全性已经得到讨论和证明。然后，如果 $\text{bp}(f) > 3k$，我们可以安全地忽略搜索树的当前分支，因为在由此产生的递归调用中

断点的数量只会增加，并且任何恒等换位序列都会比 k 更长（引理 1）。

若满足条件，则 $f(\text{root}(T)) \neq \perp$ 我们知道， T的所有子节点之间的顺序已固定。接着，我们应用第 2 节中概述的通过转置排序算法，尝试寻找一个转置序列 τ^1, ……， τ^ℓ，用 $\ell \leq k$ 次转置将 $f(\text{root}(T))$ 转换为恒等排列。若算法成功，则我们可以返回 $f(\text{root}(T))$ 和该转置序列；否则，我们报告此搜索树叶节点失败。

在剩余情况下，我们应用分支规则 1，为某个内部节点的两种可能子节点顺序创建新的递归调用。注意，在第 11 行中，分支规则 1 始终适用，因为归约规则 2 已经被穷尽地应用，而且至少有一个内部节点的子节点顺序尚未确定，例如 $\text{root}(T)$。

运行时间。注意，搜索树是一棵二叉树，因为分支规则 1 总是会创建两个子节点。此外，我们如下给出搜索树深度的上界。考虑搜索树中的任意叶实例 (T, f)。由于第 3 行，我们有 $\text{bp}(f) \leq 3k$。令 $V_b(T, f)$ 为沿搜索树中通往该叶实例的路径，由分支规则 1 固定了其子节点顺序的内部节点集合。因此，$|V_b(T, f)|$ 恰好是搜索树中叶实例 (T, f) 的深度。现在，令 $V(2)_b(T, f) \subseteq V_b(T, f)$ 为子节点均为叶节点的内部节点集合，令 $V(1)_b(T, f) \subseteq V_b(T, f)$ 为恰有一个子节点是叶节点的内部节点集合。换言之，对于所有 $v \in V(2)_b(T, f)$，我们有 $\text{ch}(v) \subseteq \text{leaf}(T)$ 且 $f(v) \neq \perp$。对于所有 $v \in V(1)_b(T, f)$，我们有 $|\text{ch}(v) \cap \text{leaf}(T)| = 1$ 且 $f(v) \neq \perp$。令 $V_b^i(T, f) = V_b(T, f) \setminus (V(1)_b(T, f) \cup V(2)_b(T, f))$。由于归约规则 1 已经被穷尽地应用， $V(2)_b(T, f)$ 中的每个节点都会使 $\text{bp}(f)$ 增加 1。此外，由于只有在归约规则 2 不再适用后我们才应用分支规则 1， $V(1)_b(T, f)$ 中的每个节点都会使 $\text{bp}(f)$ 增加 1。因此，我们有 $|V(2)_b(T, f)| + |V(1)_b(T, f)| \leq 3k$。现在，让我们在顶点集 $V(F_b) := V_b^i(T, f) \cup V(2)_b(T, f)$ 上构造一个森林 F_b，方法如下。对于每个顶点 $v \in V(F_b)$，令 $\text{anc}_{F_b}(v) = \text{anc}_T(v) \cap V(F_b)$。若 $\text{anc}_{F_b}(v) \neq \emptyset$，令 $p_v = \arg\max_{w \in \text{anc}_{F_b}(v)} \text{depth}_T(w)$，并添加边 $\{v, p_v\}$ 到 F_b。图 4 给出了 F_b 的构造示例。现在注意到， F_b 由一组互不相交的有根二叉树组成，它们的叶集并集恰好为 $V(2)_b(T, f)$。因此，$|V(F_b)| \leq 2|V(2)_b(T, f)|$，且 $|V_b(T, f)| \leq 2|V(2)_b(T, f)| + |V(1)_b(T, f)|$。由于 $|V(2)_b(T, f)| + |V(1)_b(T, f)| \leq 3k$，我们可以得出 $|V_b(T, f)| \leq 6k$，这给出了搜索树深度的上界。因此，搜索树至多有 $\mathcal{O}(2^{6k})$ 个叶节点。对于每个叶节点，都会应用一个通过转置排序的算法，其运行时间至多为 $\mathcal{O}(n(3k)^{3k})$（见第 2 节）。最后，我们可以得出，对于某个较小的常数 c，该算法可以在 $\mathcal{O}(2^{6k}(3k)^{3k}n^c)$ 时间内实现。□

虽然我们在这里给出了一种 FPT 算法，但我们认为其渐近运行时间仍有进一步优化的空间。尽管如此，该算法的瓶颈……

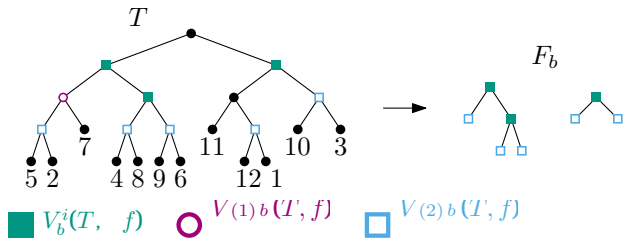


图 4：森林 F_b 的构造示例，其中对 11 的父节点应用了归约规则 2，并且 f(根节点$(T)) = \perp$。

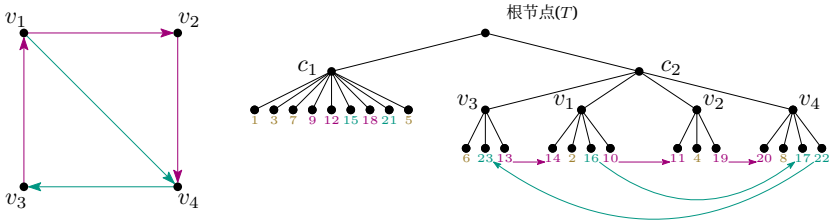


图 5：定理 5 中归约的示例。

该算法仍是用于通过转置排序的 FPT 子程序。因此，任何更优的通过转置排序算法也会改进定理 4。

我们还认为，可以为非二叉树构造类似的算法。在这种情况下，该算法的运行时间应为 $\mathcal{O}(f(k, \Delta) \cdot n^c)$，其中$\Delta$ 是输入树 T的最大度，c 是一个常数，而 f 是一个仅取决于 k 和 Δ的函数。

7 超越二叉树

在第 5 节中，我们给出了两个针对二叉树上 OTBCM 的近似算法。关键步骤是在多项式时间内找到一个与输入树相容、且使断点数量最少的排列。接下来我们将证明，对于非二叉树，这是不可能的。我们首先证明，寻找一个块数小于或等于 k 的排列是 NP 完全问题。

定理 5. 对于一个具有叶集 $[n]$ 的有根树 T 和一个整数 k，判定是否存在满足块$(\pi) \leq k$ 的 $\pi \in \Pi(T)$ 是 NP-完全问题。

显然，该问题属于 NP。对于困难性，我们从有向图上的哈密顿路径问题进行归约，其中任意一对顶点之间至多有一条弧；该问题是 NP 完全的 [18]。设 $G = (V, E)$ 是哈密顿路径问题的一个实例，其有向弧为 E。问题是要在 G 中寻找一条路径 P，使其恰好经过每个顶点一次，这样的路径称为 哈密顿路径。令 $V = \{v_1, v_2, \ldots, v_{|V|}\}$ 和 $E = \{e_1, e_2, \ldots, e_{|E|}\}$。不失一般性，我们假设 G 中的每个顶点都至少关联一条边。我们构造一棵有根树 T，使得

叶节点$(T) = [3|E| + 2|V|]$。图5给出了一个示例。 T的根节点有两个子节点 c_1 和 c_2。顶点 c_1 包含 $|V| + |E|$ 个子节点

$$\{1, 3, \ldots, 2|V| - 1\} \cup \{2|V| + 1, 2|V| + 4, 2|V| + 7, \ldots, 2|V| + 3|E| - 2\}.$$

顶点 c_2 包含 $|V|$ 个子节点，对应于顶点集 V。设 $\delta_G(v)$ 为 G 中顶点 v 的度。在 T 中，顶点 v_i 有 $1 + \delta_G(v_i)$ 个子节点，它们是以下叶节点：一个子节点为 $2i$；对于与 v_i 相 incident 的每条边 e_j，如果 v_j 是 e_j 的源点，则一个子节点为 $2|V|+3(j-1)+2$，否则为 $2|V|+3(j-1)+3$。注意， T 包含 $2|V| + 3|E|$ 个叶节点，并且叶节点$(T) = [2|V| + 3|E|]$，正如预期。直观来说，每条边 $e \in E(G)$ 对应于 T 中的一个叶节点三元组（ℓ、 $\ell+1$、 $\ell+2$），其中 ℓ 是 c_1 的子节点，而 $\ell + 1$、$\ell + 2$ 是对应于 e 的源点和目标点的顶点的子节点。叶节点 ℓ 排除了 T 中顶点 v_i 的子节点之间存在大小大于一的块的可能性。叶节点 1 、 2 、 . . . 、 $2|V|$ 使得可以存在一个块，该块由 $T(c_1)$ 中最右侧的叶节点与 $T(c_2)$ 中最左侧的叶节点构成。这使我们能够证明， G 包含哈密顿路径，当且仅当存在一个 $\pi \in \Pi(T)$，使得块(π) $\leq 2|V| + 3|E| - 1 - (|V| - 1)$。

引理 8 如果 G 包含一条哈密顿路径，那么存在 $\pi \in \Pi(T)$，使得块(π) $\leq 2|V| + 3|E| - 1 - (|V| - 1)$。

证明。 设 $(v_{i_1}, v_{i_2}, \ldots, {}^{v_{i_{|V|}}})$ 是 G 中的一条哈密顿路径。我们通过指定 T 中每个内部节点的子节点顺序，来描述置换 $\pi \in \Pi(T)$

– 根节点 root(T) 的子节点按如下方式排列： c_1 在 c_2 之前。– c_1 的子节点按如下方式排列： $2i_1 - 1$ 位于最右侧，其余子节点任意排列。– c_2 的子节点按照哈密顿路径排列。也就是说，v_{i_1} 在 v_{i_2} 之前， v_{i_2} 在 v_{i_3} 之前，依此类推。– 对于顶点 v_{i_k}，其子节点按如下方式排列。若 $k = 1$，则 v_{i_k} 最左侧的子节点是 $2i_k$。否则，设 e_j 为对应于有向弧 $(v_{i_{k-1}}, v_{i_k})$ 的边。那么， v_{i_k} 最左侧的子节点是 $2|V| + 3(j - 1) + 3$，而 $v_{i_{k-1}}$ 最右侧的子节点是 $2|V| + 3(j - 1) + 2$。

根据构造，我们有 $\pi \in \Pi(T)$。接下来只需证明 blocks(π) $=2|V| + 3|E| - 1 - (|V| - 1)$。注意， π 至多有 $2|V| + 3|E|$ 个块，而每个大小为二的块都会使这个数量减少一。因此，我们要证明存在 $1 + (|V| - 1)$ 个大小为二的块。其中第一个块由 $T(c_1)$ 中最右侧的叶节点和 $T(c_2)$ 中最左侧的叶节点构成。其余 $|V| - 1$ 个块由 v_{i_k} 的最右侧子节点和 $v_{i_{k+1}}$的最左侧子节点构成。这些子节点在 π 中依次相邻，对应于 G 中的有向弧 $(v_{i_k}, v_{i_{k+1}})$，因此构成一个块。□

引理 9 如果存在 $\pi \in \Pi(T)$，且具有块$(\pi) \leq 2|V| + 3|E| - 1 - (|V| - 1)$，那么 G 包含一条哈密顿路径。

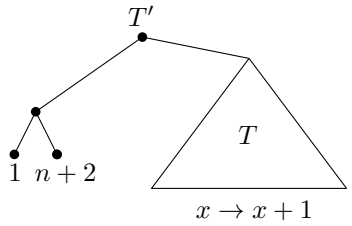


图 6：推论 1 中树 T' 的构造。

证明。设 $\pi \in \Pi(T)$具有 $\leq 2|V|+3|E|-1-(|V|-1)$个块。首先注意，根据构造，π 不可能包含大小大于二的块：ch($T(c_1)$)中的每一对相邻子节点都会构成一个断点，因为不存在构成连续编号的子节点；并且如上所述，单个顶点的两个子节点之间不存在非断点。因此，π 包含$1+(|V|-1)$个大小为二的块。c_1 的子节点要么全部出现在叶节点$T(c_2)$的左侧，要么全部出现在其右侧。因此，大小为二的块可以描述如下：至多有一个块由 c_1 的一个子节点和 v_i 的一个子节点构成，且满足 $i \in [|V|]$。其余块均由两个叶节点 x、y 构成，满足 $x+1=y$，并且 x 是某个 v_{i_α} 的子节点，y 是某个v_{i_β}的子节点。注意，根据构造，只有当 G中存在一条从v_{i_α} 指向v_{i_β} 的有向弧时，这样的块才可能存在。由于这类邻接关系共有 $|V|-1$ 个，我们定义哈密顿路径 P，使得 v_{i_α} 是v_{i_β}的前驱。□

定理 *5* 的证明。注意，给定的归约是多项式的，因为 T 恰好有 $2|V|+3|E|$ 个叶节点。正确性由引理 8 和引理 9 得出。□

对上述问题进行一次简单的归约，即可得到以下推论。

推论 1 对于一棵有 $[n]$ 叶节点的有根树 T，判定是否存在 $\pi \in \Pi(T)$ 使得 bp(π) $\leq k$ 是 NP- 完全的。

证明。设 T 是一棵具有 $=[n]$ 个叶节点的树 T。如下构造树 T'（示意图见图 6）：

– 对于 T 中的每个叶节点，将 x 设为 $x+1$。– 创建一个新根节点，将其连接到 root(T) 和一个新顶点 u。– 添加叶节点 1 和 $n+2$，并将它们连接到 u。

注意，存在 $\pi \in \Pi(T)$，使得 blocks(π) $\leq k$，当且仅当存在$\pi' \in \Pi(T')$，使得 bp(π') $\leq k+2$。因此，该结论可由定理 5 得出。

□

不过，我们认为，如果对树的最大度或块/断点的数量设定一个上界，这两个问题都会变得易处理。

因此，第 5 节中用于获得多项式时间近似算法的技术无法扩展到非二叉树。不过，这并不意味着 OTBCM 不存在多项式时间近似算法。

8 总结与开放问题

我们从不同的理论角度分析了最小化单侧（二叉）纠缠图中块交叉的复杂性。仍有许多开放问题和有趣的研究方向：

– 我们假设其中一棵树固定不动。如果两棵树的叶节点都可以重新排列，会发生什么？– 通过转置排序可以实现比本文所述更好的近似。不过，这些算法依赖一种更复杂的结构，称为断点图（例如参见 $[2, 13, 19, 26]$）。这些近似算法能否用于纠缠图？– 我们的算法生成的纠缠图中，一对边可能会多次交叉。能否借助类似单调块交叉（参见 [17]）这样的概念来避免这种情况？– 我们的算法在实际应用中的表现如何？实际应用中，两两交叉的数量与块交叉的数量之间有何关系？

参考文献

1. Alam, M.J., Fink, M., Pupyrev, S.：《捆绑交叉数》。载于：Hu, Y.、Nöllenburg, M.（编）第24届图绘制与网络可视化研讨会（GD）论文集。《计算机科学讲义》，第9801卷，第399–412页。施普林格（2016）。https://doi.org/ 10 1007 978 3 319 50106 2 _31 . / - - - - . Bafna, V., Pevzner, P.A.：通过转置排序。《 SIAM 离散数学杂志》 **11**(2), 224–240 （1998）。https://doi.org/10.1137/S089548019528280X 3. Bar-Joseph, Z., Demaine, E.D., Gifford, D.K., Srebro, N., Hamel, A.M., Jaakkola, T.S.：针对基因表达数据进行最优叶节点排序的K叉聚类。《生物信息学》**19**030 4 9 1070 1078 2003 10 1093 ( ), – ( ). https://doi.org/ . / bioinformatics/btg . Baumann, F., Buchheim, C., Liers, F.：树约束下的精确二部交叉最小化。载于：Festa, P.（编）第9届实验算法研讨会（SEA）论文集。《计算机科学讲义》，第6049卷，第118–128页。施普林格（2010）。https://doi.org/ 10 1007 978 3 642 13193 6 _11 . / - - - - . Böcker, S., Hüffner, F., Truß, A., Wahlström, M.：一种更快速的固定参数方法，用于绘制二叉纠缠图。载于：Chen, J., Fomin, F.V.（编）第4届参数化与精确计算研讨会（IWPEC）论文集。《计算机科学讲义》，第5917卷，第 38 49 2009 10 1007 978 3 642 11269 0 _3 –页。施普林格（）。https://doi.org/ . / - - - - . Brandes, U.：完全二叉树的最优叶节点排序。《离散算法杂志》**5**003 7 3 546 552 2007 10 1016 2006 09 ( ), – ( ). https://doi.org/ . /j.jda. . . . Buchin, K., Buchin, M., Byrka, J., Nöllenburg, M., Okamoto, Y., Silveira, R.I., Wolff, A.：绘制（二叉）纠缠图——困难性、近似算法与固定参数可处理性。《算法》**62**(1-2), 309–332（2012）。https://doi.org/10 3 8 1007 00453 010 9456 . /s - - - . Bulteau, L., Fertin, G., Rusu, I.：通过转置排序是困难的。《SIAM离散数学杂志》**26**110851390 9 3 1148 1180 2012 10 1137 ( ), – ( ). https://doi.org/ . / . Bulteau, L., Gambette, P., Seminck, O.：根据叶节点上的顺序重新排列树。载于：Bannai, H., Holub, J.（编）第33届组合模式匹配研讨会（CPM）论文集。《信息学论文集》，第223卷，第24:1–24:15页（2022）。https://doi.org/10.4230/LIPIcs.CPM.2022.24

10. Christie, D.A.：《基因组重排问题》。博士论文，格拉斯哥大学（1998），https://theses.gla.ac.uk/74685/ 11. van Dijk, T.C., Fink, M., Fischer, N., Lipp, F., Markfelder, P., Ravsky, A., Suri, S., Wolff, A.：故事线可视化中的块交叉。《图算法与应用期刊》**2**100443 12 5 873 913 2017 10 7155 ( ), – ( ). https://doi.org/ . /jgaa. . Dwyer, T., Schreiber, F.：用于二维半系统发育树可视化的最优叶节点排序。载于：Churcher, N., Churcher, C.（编）《澳大拉西亚信息可视化研讨会（InVis.au）》。CRPIT，第35卷，第109–115页。澳大利亚计算机学会（2004），http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV35Dwyer.html 13. Elias, I., Hartman, T.：一种用于通过转置排序的 1.375 近似算法。《IEEE/ACM计算生物学与生物信息学汇刊》**3**(4), 369–379 (2006). https://doi.org/10 44 14 1109 2006 . /TCBB. . . Eriksson, H., Eriksson, K., Karlander, J., Svensson, L.J., Wästlund, J.：扑克牌桥牌手牌排序。《离散数学》**241**(1-3), 289–300 (2001). https://doi.org/10 9 15 1016 0012 365 01 00150 . /S - X( ) - . Fernau, H., Kaufmann, M., Poths, M.：通过交叉最小化比较树。《计算机与系统科学杂志》**76**(7), 593–608 (2010). https://doi.org/10 014 16 1016 2009 10 . /j.jcss. . . . Fink, M., Hershberger, J., Suri, S., Verbeek, K.：嵌入图中的捆绑交叉。载于：Kranakis, E., Navarro, G., Chávez, E.（编）《第12届理论信息学研讨会论文集（LATIN）》。《计算机科学讲义》，第9644卷，第454–468页。施普林格（2016）。https://doi.org/ 10 1007 978 3 662 49529 2_34 . / - - - - . Fink, M., Pupyrev, S., 17 Wolff, A.：通过块交叉对地铁线路排序。《图算法与应用期刊》**19**00351 18 1 111 153 2015 10 7155 ( ), – ( ). https://doi.org/ . /jgaa. . Garey, M.R., Johnson, D.S.：《计算机与难解性：NP 完全理论指南》。W. H. Freeman（1979）19. Hartman, T., Shamir, R.：一种更简单、更快速的用于通过转置排序的 1.5 近似算法。《信息与计算》**204**(2), 275–290 (2006). https://doi.org/10 002 20 1016 2005 09 . /j.ic. . . . Holten, D., van Wijk, J.J.：层次组织数据的可视化比较。《计算机图形论坛》**27**(3), 759–766 (2008). https://doi.org/10.1111/j.1467- 8659.2008.01205.x 21. Mahajan, M., Rama, R., Raman, V., Vijaykumar, S.：近似块排序。《计算机科学基础国际期刊》**17**(2), 337–356 (2006). https://doi.org/10 0129054106003863 22 1142. /S . Nöllenburg, M.：非平面图绘制中的交叉布局。载于：Hong, S., Tokuyama, T.（编）《超越平面图》，NII湘南会议通讯，第 187 209 2020 10 1007 978 981 15 6533 5_11 –页。 23 施普林格（）。https://doi.org/ . / - - - - . Nöllenburg, M., Völker, M., Wolff, A., Holten, D.：二叉纠缠图绘制：实验评估。载于：Finocchi, I., Hershberger, J.（编）《第11届算法工程与实验研讨会论文集（ALENEX）》。第106–119页。SIAM（2009 11 24 10 1137 1 9781611972894）。https://doi.org/ . / . . . Page, R.D.M.：《纠缠之树：系统发育、共物种形成与协同进化》。芝加哥大学出版社（2003）25. Scornavacca, C., Zickmann, F., Huson, D.H.：有根系统发育树和网络的纠缠图。《生物信息学》**27**(13), 248–256 (2011). https://doi.org/10 210 26 1093. /bioinformatics/btr . Silva, L.A.G., Kowada, L.A.B., Rocco, N.R., Walter, M.E.M.T.：一种新的用于通过转置排序的 1.375 近似算法。《分子生物学算法》**17**(1), 1 (2022). https://doi.org/10.1186/s13015-022-00205-z

27. Venkatachalam, B., Apple, J., John, K.S., Gusfield, D.：《解开缠结图：通过绘图比较树》。IEEE/ACM计算生物学与生物信息学汇刊 **7**57 28 4 588 597 2010 10 1109 2010 ( ), – ( )。https://doi.org/ 。/TCBB. . . Walter, M.E.T., Dias, Z., Meidanis, J.：一种近似转置距离的新方法。载于：de la Fuente, P.（编）, 《第七届字符串处理与信息检索研讨会》（SPIRE）论文集。第199–208页。IEEE Computer Society（2000）。https://doi.org/10.1109/SPIRE.2000.878196

方程(1) 引理 6 的证明

下面给出公式 (1) 的证明。首先注意，我们的 $\ominus$-操作可以定义如下。设 $\pi \in \Pi_n$ 和 $x \in [n]$，则 $\pi \ominus x \in \Pi_{n-1}$ 是定义如下的排列

$$(\pi \ominus x)(j) = \begin{cases} \pi(j) & \text{if } j < \mathsf{inv}(\pi)(x) \text{ and } \pi(j) < x \\ \pi(j) - 1 & \text{if } j < \mathsf{inv}(\pi)(x) \text{ and } \pi(j) > x \\ \pi(j+1) & \text{if } j \geq \mathsf{inv}(\pi)(x) \text{ and } \pi(j+1) < x \\ \pi(j+1) - 1 & \text{if } j \geq \mathsf{inv}(\pi)(x) \text{ and } \pi(j+1) > x \end{cases} \tag{3}$$

我们通过对 ℓ 进行归纳来证明方程（1）。基本情形（$\ell = 0$）显然成立。接下来进行归纳步骤，因此 $\ell \geq 1$ 是任意的。考虑方程（1）的右侧。首先，根据归纳假设，我们有

$$(\pi \ominus \pi_i) \circ \upsilon^1 \circ \cdots \circ \upsilon^\ell = ((\pi \circ \tau^1 \circ \cdots \circ \tau^{\ell-1}) \ominus \pi_i) \circ \upsilon^\ell. \tag{4}$$

记住，$\upsilon^\ell = \tau^\ell \ominus \mathsf{inv}(\pi \circ \tau^1 \circ \cdots \circ \tau^{\ell-1})(\pi_i)$。令 $\phi = \pi \circ \tau^1 \circ \cdots \circ \tau^{\ell-1}$。将其代入方程（1），得到

$$(\phi \circ \tau^\ell) \ominus \pi_i = (\phi \ominus \pi_i) \circ (\tau^\ell \ominus \mathsf{inv}(\phi)(\pi_i)) \tag{5}$$

我们考虑方程（5）左侧和右侧的所有情形，并证明它们得到相同的排列。先考虑左侧。

$$((\phi \circ \tau^\ell) \ominus \pi_i)(j) = \\ \begin{cases} (\phi \circ \tau^\ell)(j) & \text{if } j < \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i) \text{ and } (\phi \circ \tau^\ell)(j) < \pi_i \\ (\phi \circ \tau^\ell)(j) - 1 & \text{if } j < \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i) \text{ and } (\phi \circ \tau^\ell)(j) > \pi_i \\ (\phi \circ \tau^\ell)(j+1) & \text{if } j \geq \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i) \text{ and } (\phi \circ \tau^\ell)(j+1) < \pi_i \\ (\phi \circ \tau^\ell)(j+1) - 1 & \text{if } j \geq \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i) \text{ and } (\phi \circ \tau^\ell)(j+1) > \pi_i \end{cases} \tag{6}$$

现在让我们考虑方程（5）的右侧。我们需要考虑 16 种情况，但其中一些情况会产生矛盾，已用红色标出。

$$((\phi \ominus \pi_i) \circ (\tau^\ell \ominus \mathsf{inv}(\phi)(\pi_i)))(j) = \\ \begin{cases} (\phi \ominus \pi_i)(\tau^\ell(j)) & \text{if } j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i)) \text{ and } \tau^\ell(j) < \mathsf{inv}(\phi)(\pi_i) \\ (\phi \ominus \pi_i)(\tau^\ell(j) - 1) & \text{if } j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i)) \text{ and } \tau^\ell(j) > \mathsf{inv}(\phi)(\pi_i) \\ (\phi \ominus \pi_i)(\tau^\ell(j+1)) & \text{if } j \geq \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i)) \text{ and } \tau^\ell(j+1) < \mathsf{inv}(\phi)(\pi_i) \\ (\phi \ominus \pi_i)(\tau^\ell(j+1) - 1) & \text{if } j \geq \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i)) \text{ and } \tau^\ell(j+1) > \mathsf{inv}(\phi)(\pi_i) \end{cases} \tag{7}$$

接下来的 4 个段落将更详细地讨论该方程的 4 种情况。

情形 *1* $j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$ 和 $\tau^\ell(j) < \mathsf{inv}(\phi)(\pi_i)$。

$$(\phi \ominus \pi_i)(\tau^\ell(j)) = \begin{cases} \phi(\tau^\ell(j)) & \text{if } \tau^\ell(j) < \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j)) < \pi_i \\ \phi(\tau^\ell(j)) - 1 & \text{if } \tau^\ell(j) < \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j)) > \pi_i \\ \text{impossible} & \text{if } \tau^\ell(j) \geq \mathsf{inv}(\phi)(\pi_i) \end{cases} \tag{8}$$

最后一种情形是不可能的，因为 $\tau^\ell(j) \geq \mathsf{inv}(\phi)(\pi_i)$，但根据情形1，$\tau^\ell(j) < \mathsf{inv}(\phi)(\pi_i)$。否则，我们有

- $\phi(\tau^\ell(j))$, if $j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$ which is equivalent to $j < \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$, and $\phi(\tau^\ell(j)) < \pi_i$.
- $\phi(\tau^\ell(j))$, if $j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$ which is equivalent to $j < \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$, and $\phi(\tau^\ell(j)) > \pi_i$.

如果与方程（6）进行比较，我们会发现，情形1中排列的结果是等价的。因此，方程（5）在这种情形下成立。

情况 *2* $j <$逆序(τ^ℓ)(逆序$(\phi)(\pi_i))$ 以及 $\tau^\ell(j) >$逆序$(\phi)(\pi_i)$。

$$(\phi \ominus \pi_i)(\tau^\ell(j) - 1) = \begin{cases} \text{impossible} & \text{if } \tau^\ell(j) - 1 < \mathsf{inv}(\phi)(\pi_i) \\ \phi(\tau^\ell(j)) & \text{if } \tau^\ell(j) - 1 \geq \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j)) < \pi_i \\ \phi(\tau^\ell(j)) - 1 & \text{if } \tau^\ell(j) - 1 \geq \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j)) > \pi_i \end{cases} \tag{9}$$

第一种情形不可能，因为 $\tau^\ell(j) - 1 < \mathsf{inv}(\phi)(\pi_i) \iff \tau^\ell(j) \leq \mathsf{inv}(\phi)(\pi_i)$，但根据情形2，$\tau^\ell(j) > \mathsf{inv}(\phi)(\pi_i)$。否则，我们有

- $\phi(\tau^\ell(j))$, if $j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$ which is equivalent to $j < \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$, and $\phi(\tau^\ell(j)) < \pi_i$.
- $\phi(\tau^\ell(j))$, if $j < \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$ which is equivalent to $j < \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$, and $\phi(\tau^\ell(j)) > \pi_i$.

如果与方程（6）比较，我们会注意到，在情形2中，排列的结果是等价的。因此，方程（5）在这种情况下成立。

情况 *3* $j \geq$求逆(τ^ℓ)(求逆$(\phi)(\pi_i))$ 以及 $\tau^\ell(j+1) <$求逆$(\phi)(\pi_i)$。

$$(\phi \ominus \pi_i)(\tau^\ell(j+1)) = \begin{cases} \phi(\tau^\ell(j+1)) & \text{if } \tau^\ell(j+1) < \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j+1)) < \pi_i \\ \phi(\tau^\ell(j+1)) - 1 & \text{if } \tau^\ell(j+1) < \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j+1)) > \pi_i \\ \text{impossible} & \text{if } \tau^\ell(j+1) \geq \mathsf{inv}(\phi)(\pi_i) \end{cases} \tag{10}$$

根据情形3，最后一种情况不可能成立，因为 $\tau^\ell(j+1) \geq \mathsf{inv}(\phi)(\pi_i)$，但 $\tau^\ell(j+1) < \mathsf{inv}(\phi)(\pi_i)$。否则，我们有

– $\phi(\tau^\ell(j+1))$，若 $j \geq \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$，这等价于 $j \geq \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$，且 $\phi(\tau^\ell(j+1)) < \pi_i$。– $\phi(\tau^\ell(j+1)) - 1$，若 $j \geq \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$，这等价于 $j \geq \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$，且 $\phi(\tau^\ell(j+1)) > \pi_i$。

若與方程（6）比較，我們會注意到，在情形3中，排列的結果是等價的。因此，方程（5）在此情形下成立。

情况 4 $j \geq$逆序(τ^ℓ)(逆序$(\phi)(\pi_i)$) 以及 $\tau^\ell(j+1) >$逆序$(\phi)(\pi_i)$。

$$(\phi \ominus \pi_i)(\tau^\ell(j+1) - 1) = \begin{cases} \text{impossible} & \text{if } \tau^\ell(j+1) - 1 < \mathsf{inv}(\phi)(\pi_i) \\ \phi(\tau^\ell(j+1)) & \text{if } \tau^\ell(j+1) - 1 \geq \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j+1)) < \pi_i \\ \phi(\tau^\ell(j+1)) - 1 & \text{if } \tau^\ell(j+1) - 1 \geq \mathsf{inv}(\phi)(\pi_i) \text{ and } \phi(\tau^\ell(j+1)) > \pi_i \end{cases} \tag{11}$$

第一种情况不可能，因为 $\tau^\ell(j+1) - 1 < \mathsf{inv}(\phi)(\pi_i) \iff \tau^\ell(j+1) \leq \mathsf{inv}(\phi)(\pi_i)$，但根据情况 4，$\tau^\ell(j+1) > \mathsf{inv}(\phi)(\pi_i)$。否则，我们有

– $\phi(\tau^\ell(j+1))$，如果 $j \geq \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$，这等价于 $j \geq \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$，且 $\phi(\tau^\ell(j+1)) < \pi_i$。– $\phi(\tau^\ell(j+1)) - 1$，如果 $j \geq \mathsf{inv}(\tau^\ell)(\mathsf{inv}(\phi)(\pi_i))$，这等价于 $j \geq \mathsf{inv}(\phi \circ \tau^\ell)(\pi_i)$，且 $\phi(\tau^\ell(j+1)) > \pi_i$。

与方程（6）对比可知，在情况 4 中，排列的结果是等价的。因此，方程（5）在这种情况下成立。

方程（5）在所有情况下都成立，由此完成归纳步骤。 □