Hexasort – 图上堆叠颜色的复杂性

LinusKlocker 维也纳
工业大学，维也纳，奥地利

Simon Dominik Fink 算法与复杂性系，维也纳工业大学，维也纳，奥地利

摘要

许多广受欢迎的益智和匹配游戏都曾从计算复杂性的角度得到分析。典型例子包括数独 [12]，糖果传奇 [7]，和 Flood-It [4]。这些广受欢迎的游戏有一个共同特点：它们的广义决策版本都是 NP-困难，这通常被认为是它们本身具有挑战性、并令玩家欲罢不能的原因。本文研究一种广受欢迎的单人堆叠游戏，通常称为 Hexasort。该游戏可以建模为将有色堆叠放置在图的顶点上；相邻且颜色相同的堆叠会按照确定性规则合并并消失。我们证明，即使将问题限制为单色堆，并逐步施加更严格的图类限制，Hexasort 仍是 NP-困难；进一步地，在有界高度或有界度的树上，该问题具有强 NP 困难性。针对固定参数可解算法，我们确定了问题可在多项式时间内求解的情形，并提出了动态规划算法。

2012 年 ACM 学科分类 计算理论 → 问题、归约与完备性

关键词与短语 Hexasort、图上的离线颜色堆叠、NP 完全、多项式时间可解、动态规划

资助 维也纳科学与技术基金（WWTF）项目 [10.47379/ICT22029] 和奥地利科学基金（FWF）项目 [10.55776/Y1329]

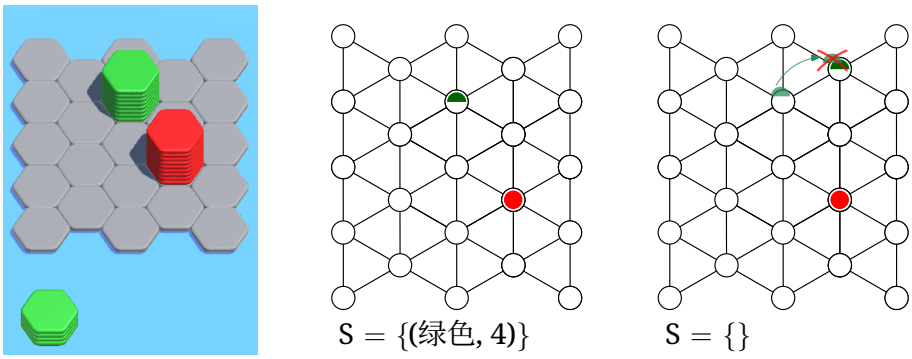
arXiv:2603.01244v1 [cs.CC] 1 Mar 2026

1 引言

我们研究单人堆叠游戏 Hexasort 的计算复杂性。该游戏有许多不同的浏览器版或手机应用实现，[1] 而且游戏机制也有各种变化。它们的共同点是都在六边形游戏棋盘上进行：给定一串由彩色六边形组成的堆，需要将它们放到空格上。相邻堆顶端的六边形颜色相同时，该颜色的所有六边形都会从一堆移到另一堆；见图 1。如果这样使得足够多同色六边形上下叠放，它们就会全部消失。游戏目标是在空格用尽之前放置完整个输入序列。

在我们的游戏模型中，我们将游戏棋盘推广为任意图，并将有色堆叠放置在其顶点上。我们只考虑单色堆，并规定：只要一个堆被放置在与同色堆相邻的位置，原有的堆就会完全移到新堆上。我们不考虑游戏机制的其他复杂情况，例如初始时棋盘非空，或阻挡物必须在完成一定次数的合并后才能移除，不过我们之后会说明，可以通过构件实现类似的机制。因此，Hexasort 的一个实例 $(G,\ S,\ t)$ 由一个图 $G = (V,\ E)$、一个阈值 $t \in \mathbb{N}^+$以及一个**堆叠**序列组成，这些堆叠的颜色为 C，其中每个堆 $s = (c,\ h) \in S$ 都有一个颜色 $c \in C$ 和一个高度 $h \in \mathbb{N}^+$。

[1] 例如，参见 hexasort.io、games-by-sam.com/bee-sort-by-sam 或 spielaffe.de/Spiel/Hexa-Sort-3D。



(a) 原始游戏截图 (b) 图抽象 (c) 放置后的结果

图 1 Fitting Hexasort 游戏示例及其图表示（未显示堆高）。放置 后，绿色堆叠合并，到达 t 并消失。

游戏的一个配置为每个顶点指定一个堆或空置，也就是说，它是一个函数 $\delta: V(G) \to C \times \mathbb{N}^+ \cup \perp$，其中 $\delta(v) = \perp$ 表示顶点 v 当前为空；否则，$\delta(v) = (c, h)$ 表示 v 上有一个颜色为 c、高度为 h 的堆。初始时，对于所有 $v \in V$，均有 $\delta(v) = \perp$。要从 S 中放置下一个堆 $s_i = (c, h)$，玩家选择一个空顶点x。令 $N_c[x]$ 表示 x 的非空邻居中颜色为 c 的顶点，并令 h' 为这些顶点上堆的高度总和。通过对 δ 执行以下操作，我们得到一个新配置 δ'：对所有 $u \in N_c[x]$，令 $\delta'(u) = \perp$；再根据累积高度更新 $\delta'(x)$。若 $h + h' \geq t$，则令$\delta'(x) = \perp$；否则令 $\delta'(x) = (c, h + h')$。

Empty Hexasort 问题的核心算法问题是：能否按照给定顺序将 S 中的所有堆放置到图上，使得 S 耗尽后所有顶点再次变为空。问题变体 Fitting Hexasort 放宽了这一要求，只需按顺序放置 S 中的所有堆，而不要求之后所有顶点都为空。注意，任何 Empty Hexasort 实例都可以通过向 S追加 $|V|$ 个堆，转化为一个等价的 Fitting 变体实例；这些堆的颜色彼此不同且均为新颜色，并且高度小于 t。反过来，任何 Empty Hexasort 的是实例显然也是 Fitting 变体的是实例。

本文探讨 Fitting 与 Empty Hexasort 的可处理实例与难处理实例之间的边界。在第 2 节中，我们证明即使在严格的结构限制下，Empty Hexasort 仍是 NP-困难的。具体而言，该问题在由两条独立边组成的图上、仅使用一种颜色时是弱 NP 困难的，并且在高度或度有界的树上是强 NP 困难的。面向固定参数可 tractable 算法，在第 3 节中，我们指出两种变体均可多项式时间可解的情形，并提出一种运行时间相对于输入图规模呈指数级的动态规划方法。

1.1 相关工作

已有若干游戏被研究，其机制模拟了 Hexasort 的部分方面，例如在图上放置物体、单色连通分量，以及合并与消除阈值。据我们所知，这些机制的组合，尤其是 Hexasort 的计算复杂性，此前尚未被研究。正如我们将看到的，游戏状态的非单调发展、会丢弃任何超额部分的基于阈值的消除，以及必须消除 Empty 中的所有堆叠

Hexasort 构成了一个特别有趣的挑战。

Iburi 等人 [8]分析了一个近期出现的、涉及在图上放置物体的游戏实例——Matching Match Puzzle。在这个谜题中，玩家会得到一个带颜色的图，以及一组火柴棍，每根火柴棍的两个端点各被指定一种颜色。任务是将火柴棍放到图的边上，使得每个顶点处所有相接的火柴棍端点颜色一致，并且与给定图的着色相符。即使在蜘蛛图上，该问题也是 NP-困难。有趣的是，这一结果是通过从 3-划分进行归约得到的，与第 2.2 节所用的归约类似。

单色连通分量相关游戏中，一个研究较为深入的例子是 Flood-It。Fellows 等人 [4] 的综述总结了名为 Flood-It 和 Free-Flood-It 的两个变体的复杂性结果。两者都在顶点着色的图上进行。在 Flood-It 中，玩家反复给一个固定的枢轴顶点重新着色，从而使其整个单色连通分量也随之改变颜色。在 Free-Flood-It 中，则可以自由选择要重新着色其单色连通分量的顶点。两种游戏的目标都是用尽可能少的步数让整个棋盘变成单一颜色。在棋盘大小为 $3 \times n$ 且使用四种颜色时，Flood-It 已经是 NP-困难的。

一种结合合并与消除机制的游戏是 Clickomania。在 Clickomania 中，玩家面对一个由彩色方块组成的网格。每一步都要选中一个由至少两个方块组成的单色连通分量，并将它们从棋盘上移除。被移除方块上方的方块会下落，空列则会消失。目标是彻底清空棋盘，或尽可能减少剩余方块的数量。即使将颜色限制为两种、列数限制为两列，Clickomania 仍然是 NP-困难问题 [1]。

在游戏的算法研究之外，Hexasort 中堆的合并与基于阈值的消除，与数字微流控平台（如 OpenDrop [2]）上物理液滴的行为有相似之处。在这类平台上，电极网格用于操控离散的液滴。通过激活特定电极，可以使液滴在表面移动、混合和合并。由于表面张力和物理限制，单个电极单元只能承载特定体积的液体；超过这一体积会导致液滴不稳定。在这个类比中，堆的高度对应液滴体积，而合并则取决于彼此的距离。除了用于医疗和化学应用，这种设备还通过彩色液滴被改编成游戏 [11]。

1.2 预备知识

我们区分集合（无序元素集合，记作 $\{\dots\}$）和序列（有序列表，记作 $\langle\dots\rangle$）。对于两个序列 $A = \langle a_1, \dots, a_n\rangle$ 和 $B = \langle b_1, \dots, b_m\rangle$，我们将它们的连接定义为 $A+B$，即将 B 中的元素追加到 A 末尾后形成的新序列，因此 $A + B = \langle a_1, \dots, a_n, b_1, \dots, b_m\rangle$。

我们用术语三合并 表示这样一步：放置当前的堆，使其恰好有两个相邻的、非空且颜色相同的顶点。对于颜色 c，我们将$sum(c, S)$ 表示为当前 S 中剩余颜色 c 的高度总和。

我们假设对于所有 $(c, h) \in S$， $h \leq t$，因为任何超过 t 的堆叠都可以等效地截断为高度 t。注意，在配置 δ 中， G 上相邻的两个堆叠不能具有相同的颜色。最后，在 Empty Hexasort 中，我们假设对于任意$c \in C$，sum$(c, S) \geq t$，否则该实例显然是否定的。

2 NP困难性

我们首先通过从弱 NP-困难问题 Partition问题 归约，证明 Empty Hexasort 是 NP-困难的：在第 2.1 节中，将该问题归约到仅含一种颜色、且图由两条独立边构成的 Empty Hexasort。随后，我们将这一困难性结果扩展到具有九个顶点的连通图。我们还在第 2.2 节中通过从 3-划分归约到高度或度数有界的树上的 Empty Hexasort，证明其具有强 NP 困难性。由此，我们证明，即使图的规模和颜色数量都是较小的常数，或者所有数值输入都关于输入规模为多项式有界，该问题仍然是困难的。

本节中的图采用有色堆叠的视觉编码以确保可访问性：、、、和分别表示红色、蓝色、绿色和黑色堆叠。请注意，精确的堆高通常与这些图无关，因此未予显示。

2.1 从Partition问题的归约

在Partition问题中，给定正整数的多重集 P，需要判定 P 能否被划分为两个不相交的子集 P_1 和 $P_2 = P \setminus P_1$，使得$\sum P_1 = \sum P_2$。Partition问题是子集和问题的一个著名特例，并且是（弱）NP-困难的 [6]。我们将注意力限制于满足以下条件的划分实例：(1) 元素之和 $\sum P$为偶数，(2) 不存在元素 $p \in P$满足 $p \geq \sum P/2$，以及 (3) 所需划分（若存在）并非由 P的某个前缀之和构成。请注意，违反上述任一条件的实例都可以在多项式时间内平凡地判定。

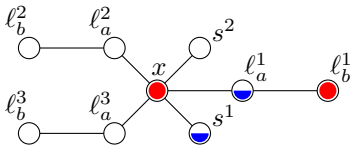
Partition问题可按如下方式归约到 Empty Hexasort。设 $P = \{p_1, p_2, \ldots, p_n\}$ 为 Partition问题的任意实例。我们构造 Empty Hexasort 的一个对应实例 $I_H = (G, S, t)$，其阈值为 $t = \sum P/2$，其中图 $G = (V, E)$ 有四个顶点 $V = \{v_1, v_2, v_3, v_4\}$ 以及两条边 $E = \{v_1v_2, v_3v_4\}$。输入堆序列 S定义为 $S = (c, p_1), (c, p_2) \ldots, (c, p_n)$，其中 c是唯一的颜色。

▶**定理 1.** 在两条独立边和单一颜色上的 *Empty Hexasort* 是弱 *NP* 困难的。

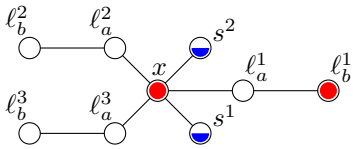
证明。 在上述构造中，容易看出，任意一个关于 P 的解都可以转化为导出的 I_H 的一个解，方法是将第一个划分的堆叠交替放置在 v_1和 v_2 上，同时将第二个划分的那些堆叠放置在 v_3 和 v_4上。由于每个划分的总和恰好为$\sum P/2 = t$，这些堆叠恰好在该划分的最后一个元素被放置时消失。

对于逆向证明，我们首先证明，将 I_H 的所有堆叠都只放在两条边之一的端点上，无法清空该图。回想一下，我们已排除某个前缀恰好构成所需 Partition问题的平凡情形。因此，将所有堆叠放在一条边上，最终会形成一个高度至少超过 t 1 的堆，并因此消失。之后，剩余堆叠的高度总和至多为 $t - 1$，因此无法消失。所以，I_H 的堆叠需要分配到两条独立边上。假设一条边接收到的堆叠超过 t，那么留给另一条边的就不足 t。◀

请注意，如果允许给定一个初始配置，其中已有一个异色堆放置在适当位置，并使两条边一直保持分离直至最后，那么上述方法很容易扩展到连通图。相反，我们构造一个构件，在一个结构更复杂但连通的图上充分限制堆的放置方式。我们的目标是论证：给定的堆序列只能以这样一种方式放置，才能让两条特定的不相邻长度为二的路径保持可用，以处理 Partition 输入序列。我们调整上述构造，得到一个实例 $I_H^* = (G', S', t)$，其中 G'是一个连通的蜘蛛图



(a) 放置 $S_{initial}$，$S_{reservation}$ 和 S 后下一个红色堆无法在 $S_{connector}$ 中进行三合并这些红色堆，违反了引理 2。



(b) 引理 4 的可容许配置。

图 2 标记图 G'，其中放置了红色和蓝色堆。图 (a) 所示的配置无法使图变为空图，而图 (b) 所示的配置可以。

图，也就是说，它是一棵只有一个度大于二的顶点的树。我们的特定蜘蛛图 G' 有九个顶点：一个度为五的中心顶点 x，两条短臂 分别由顶点 s^1、s^2组成，以及三条长度为二的长臂；见图 2。对于长臂 $i \in \{1、2、3\}$，令 ℓ_a^i 为 x的邻居，而 ℓ_b^i为叶节点。

现在我们将使用四种颜色：c_{black}（用于 S 中 Partition 输入的颜色），以及 c_{red}、c_{blue}、c_{green}（用于构件堆叠）。将初始构件堆叠的序列定义为 $S_{initial}$ =($c_{\text{red}}, t-1$), ($c_{\text{red}}, t-1$), ($c_{\text{blue}}, t-1$), ($c_{\text{blue}}, t-1$)。将预留构件堆叠定义为 $S_{reservation}$ =($c_{\text{green}}, \lfloor \frac{t}{2} \rfloor + 1$), ($c_{\text{green}}, \lfloor \frac{t}{2} \rfloor + 1$), ($c_{\text{green}}, \lceil \frac{t}{2} \rceil - 1$), ($c_{\text{green}}, \lceil \frac{t}{2} \rceil - 1$)。将用于释放初始构件堆叠所占构件的连接构件栈序列定义为 $S_{connector}$ ($c_{\text{red}}, t-1$), ($c_{\text{blue}}, t-1$)。我们将完整的输入序列构造为 $S' = S_{initial} + S_{reservation} + S + S_{connector}$。我们的构造思路是：要使图变为空图，构件堆叠（非 c_{black}）必须消失；但在处理 S 的 c_{black} 堆叠时，它们会阻挡图中的大部分区域，因此这些堆叠必须放置在恰好两条独立边的四个端点上。为了对此进行形式化证明，我们需要以下引理；这些引理在后续情形中也会派上用场。

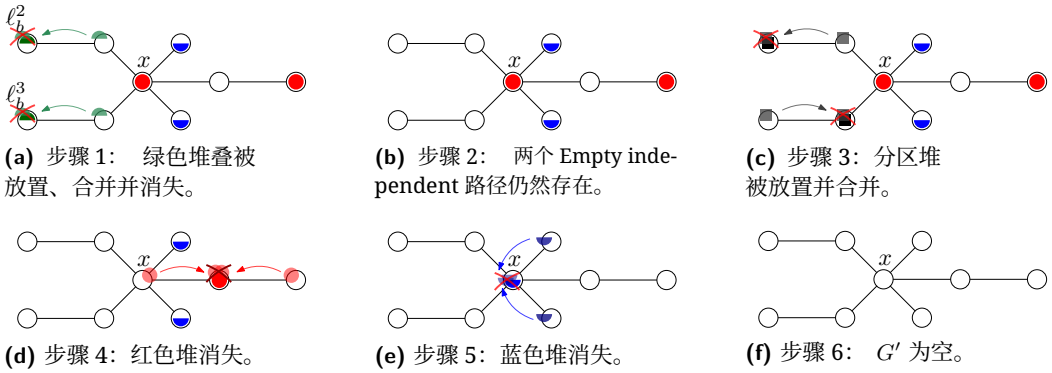
▶**引理 2** (强制三栈合并)。设 c 是一个 *Empty Hexasort* 实例中的一种颜色$I = (G, S, t)$，其中 S 恰好包含三个颜色为 c 的堆 (c, h_1)、(c, h_2)、(c, h_3)，顺序如给定，并且 $h_i < t$ 对于 $i \in$ 1,2,3 成立，而 $h_1 + h_2 \geq t$。在 I的任意解中，必须放置第一和第二个堆，使它们不相邻，但有一个公共邻居；放置第三个堆时，该邻居必须为空。

证明。如果前两个堆放在相邻顶点上，它们会立即合并并消失，导致第三个堆在最后仍未被移除。如果它们放置时相隔超过一个顶点，则之后无法形成三合并。因此，它们之间必须恰好隔着一个顶点，并将第三个堆放在该顶点上，从而一次性移除所有堆。◀

因此，适用引理 2 的两种颜色不能在如图 2a 所示的路径上交替出现。

▶ **引理 3.** 设 c是 *Empty Hexasort* 实例 $I = (G, S, t)$ 中的一种颜色，使得 S恰好包含以下四个堆 (c, h_ℓ),(c, h_ℓ), (c, h_s), (c, h_s)，其颜色均为 c，且顺序如上，并且 $h_s < \frac{t}{2} < h_\ell$ 使得 $h_s + h_\ell \geq t$。在 I的任何解中，这些堆都可以通过以下方式消失：*(a)* 将一个 h_ℓ-堆与一个 h_s-堆进行两次独立合并；或 *(b)* 在一步中合并全部四个堆。在情况 *(a)* 中，这些堆位于两条独立边的端点上；在情况 *(b)* 中，它们位于度为 *3* 的星形图上。

证明。每个颜色-c 堆都小于 t，因此没有任何堆会自行消失。任何涉及恰好三个颜色-c 堆的合并，都会使这三个堆恰好消失，只留下一个永远不会消失的堆。合并前两个堆（高度为 $2h_\ell > t$）



(a) 步骤 1： 绿色堆叠被放置、合并并消失。

(b) 步骤 2： 两个 Empty independent 路径仍然存在。

(c) 步骤 3：分区堆被放置并合并。

(d) 步骤 4：红色堆消失。

(e) 步骤 5：蓝色堆消失。

(f) 步骤 6： G' 为空。

图 3 蜘蛛图的堆放置过程，最终得到一个空图。

会使它们消失，留下后两个总高度为 $2h_s < t$ 、无法消失的堆。相比之下，两个高度不同的堆相加至少为 t ，因此会消失。因此，只有后两个 h_s 堆可以在合并后不消失，但这样一来，就没有机会再将它们与前面 h_ℓ 的堆合并并使其消失了。总而言之，这表明颜色-c 堆必须通过以下方式消失：（a）分成两对，每对两个；或（b）一次性合并全部四个。对于情况（a），我们首先需要放置前两个堆，使它们不发生合并，然后放置第三个堆，使它只与其中一个合并，这就需要第一条边。最后，将最后一个堆放在与剩下的 h_ℓ 堆相邻的位置，这需要第二条独立的边。对于情况（b），一次移动就合并全部四个堆的唯一方法，是将它们放在一个度为 3 的星形图上，并将最后一个堆放在中心。◀

由此，我们可以证明以下引理；另见图 2b。

▶ **引理 4**。 放置 $S_{initial} + S_{reservation}$ 于 $I_H^* = (G',\ S',\ t)$之后，任何解都需要将 c_{red} 堆放在 x 和 ℓ_b^1上，将 c_{blue} 堆放在 s^1上，s^2 并使所有其他顶点为空。

证明。 首先，假设顶点 x 在放置 $S_{initial}$之后仍为空，即前两个c_{red} 和两个 c_{blue} 堆。引理 2 表明， x 的四个邻居必须被 c_{red} 和 c_{blue} 堆占据，而所有叶顶点均为空。然而，占据 x 的四个邻居会导致没有空间放置 c_{green} 堆，从而不可避免地违反引理 3。因此，中心顶点 x 必须承载颜色为 c_{red} 或 c_{blue}的堆，这意味着我们只需考虑引理 3 的情况 (a)。

放置 $S_{initial}$ 顶点后， x 持有堆 $(c_a, t-1)$，其中 $c_a \in \{c_{\text{red}}, c_{\text{blue}}\}$。由引理 2 可知，必须将 $S_{initial}$ 的第二个 c_a 堆放置在 ℓ_b^i ，其中$i \in$ 属于{1, 2, 3}；不失一般性，设为 ℓ_b^1。现在证明， $S_{initial}$的另外两个堆（其颜色记为 c_b）必须分别放置在 s^1 和 s^2上。首先，如果将一个 c_b 堆放置在 ℓ^2 或 ℓ^3的任一顶点上，就只剩下一条长度为二的路径，这违反了引理 3。其次，如果将一个 c_b 堆放置在 ℓ_a^1上，即现有两个 c_a 堆之间，就会违反引理 2，因为为进行 c_a 三合并， ℓ_a^1 必须保持为空。因此，只剩下 s^1 和 s^2 。

最后，我们证明 $c_a = c_{\text{red}}$ 和 $c_b = c_{\text{blue}}$。回想一下，引理 2 特别意味着，在将第三个堆放到公共邻居上时，前两个堆的公共邻居必须为空。由于第三个 c_{red} 堆在第三个 c_{blue} 堆之前出现在 $S_{connector}$中，因此在放置 $S_{connector}$之前，那两个先前 c_{red} 堆的公共邻居在该时刻必须为空。只有当 x 和 ℓ_b^1 是 c_{red} ，而 s^1 和 s^2 是 c_{blue}时，才会如此。移除 c_{red} 堆后， x便成为 c_{blue} 堆的公共邻居。◀

图 3 展示了引理 4 之后空出的两条边如何用于求解 S 中编码的划分实例，并通过 $S_{connector}$清空图。利用引理 4，我们证明任何解都必须执行这些步骤，从而得到以下定理。

▶**定理5.** 具有九个顶点和四种颜色的蜘蛛图上的 *Empty Hexasort* 是弱 *NP* 困难的。

证明。 设 P 是一个任意的 Partition问题实例。我们使用上文定义的蜘蛛图 EMPTY HEXASORT实例 $I_H^* = (G',\ S',\ t)$ 以及扩展堆栈序列 G' 来构造。我们证明，Partition问题的实例 S' 有解，当且仅当 P 存在一个解 I_H^*，即 EMPTY HEXASORT的解。

(⇒) 假设 P是一个是实例，其划分为 P_1、P_2，且满足 $\sum P_1 = \sum P_2 = t$。根据引理 4，$S_{initial}$ 的放置位置固定，占据 x、ℓ_b^1、s^1 和 s^2。我们将 $S_{reservation}$ 的 c_{green} 个堆放在剩余边 ℓ^2 和 ℓ^3的端点上，它们会在此合并并消失。这样，序列 S恰好有两条互不相连的边可用。我们将与 P_1 对应的堆放在 ℓ^2 上，并将与 P_2 对应的堆放在 ℓ^3上。由于 $\sum P_1 = \sum P_2 = t$，两条边都被清空。最后，通过将一个c_{red} 堆放在 ℓ_a^1 上，并将一个 c_{blue} 堆放在 x上，序列 $S_{connector}$ 清除剩余的构件堆叠。因此，I_H^* 是一个是实例。

(⇐) 反过来，假设 I_H^* 是一个是实例。整个图 G' 必须被清空。根据上文给出的引理 4，只有当放置 c_{black}堆叠 S 成功清除在构件堆叠放置并消失后剩下的两条边时，才有可能做到这一点。如定理 1 的证明中的论证所示，清除 ℓ^2 和ℓ^3 要求将集合 S 划分为两个子集 S_1 和 S_2，并分别放置到 ℓ^2 和 ℓ^3上，使得 $\sum S_1 = \sum S_2 = t$。这一构造立即得到一个有效的划分 P_1，P_2适用于 P，从而证明 P是一个是实例。

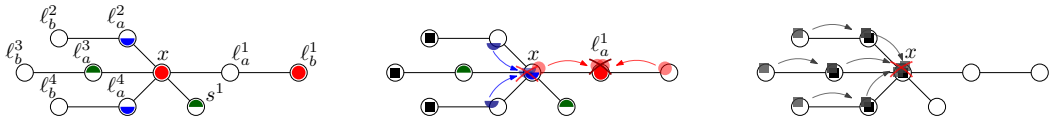
由于前述归约是多项式的，且构件至多增加五个顶点、六条边和十个大小不超过 t 的堆叠，因此，从 Partition问题到 I_H^* 的这一归约也可在多项式时间内完成。◀

2.2 从 3-划分归约

请注意，上一节通过从 Partition问题 进行归约，依赖了较大的数值，因此只能证明弱 NP-困难性。本节我们要证明强 NP 困难性，即即使 t （因此所有堆高也）受限于输入规模 [5] 的多项式，问题仍然困难。为此，我们将从强 NP-困难问题 3-划分进行归约。该问题给定由 $3m$ 个正整数组成的多重集 A，以及一个界限$B \in Z^+$。问题是能否将 A 划分为 m 个三元组 A_1，A_2，……，A_m，使得每个三元组的和恰好为 B。正如 Garey等人 [6, 第 224] 页所指出的，该问题在强意义下是 NP 完全的，并且我们可以假设每个输入值 $a \in A$ 都受到如下限制：$B/4 < a < B/2$ 且 $\sum_{a \in A} a = mB$。

对于从3-划分到Empty Hexasort的归约，我们将调整之前的构件，使其现在恰好留下三个彼此不同且孤立的顶点，以供放置单个三元组中的三个元素。通过使用 m 个彼此不相交的该构件副本，我们便可以据此对3-划分解中的 m 个三元组进行建模。更具体地说，我们的归约如下所示。一个构件由一个 x 度为五的中心顶点、四条长度为二且顶点为ℓ_a^i, l_b^i（其中 $i \in \{1,2,3,4\}$）的臂，以及一条长度为一的臂 s^1组成；见图4。

给定一个 3-划分实例 (A, B)，我们得到一个 Empty Hexasort 实例 $I_H = (G, S, t)$，其中图 G 包含 $m = \frac{|A|}{3}$ 个互不相交的构件图副本，阈值为 $t = B + 4$。对于 S，我们使用 $3m + 1$ 种颜色：c_{black} 种用于三元组元素，另外三种构件颜色 c^j 红色、c^j 蓝色和 c^j_{green}，用于 $j \in \{1, \ldots, m\}$。输入序列按如下方式构造。



(a) 可容许配置引理 6。 **(b)** 移除构件堆叠 , 最后 . **(c)** 拉动 堆叠以合并并在 x上消失。

图4 我们的3-划分归约中所使用的蜘蛛构件。

初始构件栈序列 $S^j_{initial} = (c^j$ 红色$t-1), (c^j$ 红色$t-1), (c^j$ 蓝色$t-1), (c^j$ 蓝色$t-1), (c^j_{\text{green}}, t-1), (c^j_{\text{green}}, t-1)$ 强制隔离三个顶点。序列 $S_A = (c_{\text{black}}, a_1) \ldots, (c_{\text{black}}, a_{3m})$ 表示 $A = \{a_1, \ldots, a_{3m}\}$。连接器构件栈序列 $S^j_{connector} = (c^j$ 红色$t-1), (c^j$ 蓝色$t-1), (c^j_{\text{green}}, t-1)$ 使每个构件脱离（初始）构件堆叠。三元组连接器序列 $S^j_{final} = (c_{black}, 1), (c_{black}, 1), (c_{black}, 1), (c_{black}, 1)$ 取出已放置的三元组，使其所有堆合并并消失。

我们将完整的输入序列构造为 $S = S_{initial} + S_A + S_{connector} + S_{final}$，其中$S_{initial}$，$S_{connector}$ 以及 S_{final} 是分别通过拼接所有 $S^j_{initial}$，$S^j_{connector}$，以及S^j_{final}得到的，对于所有 $j \in \{1, \ldots\ldots, m\}$。请注意，引理 2仍然适用于所有 c^j 蓝色，c^j 红色，以及 c^j_{green}。与引理 4类似，我们再次证明 $S_{initial}$ 会强制某种配置；见图 4a。

▶ **引理 6**。将 $S_{initial}$ 放入 I_H后，对于每个构件，任何解都要求顶点 ℓ_a^i以及所有 $\ell_b^{j\neq i}$保持为空，且某个 $i \in \{1、2、3、4\}$满足此条件。此外，ℓ_a^i 的两个邻居颜色相同，均为 c；每个构件恰好由三种颜色组成，其中 c 在 S_{final}中最先出现。

证明。首先，我们分析一个构件最多能有多少堆叠。假设一个构件至少包含七个堆叠。由于一个构件有十个顶点，至少有一个堆叠必须放在叶节点 ℓ_b^i 上，其中 $i \in \{1、2、3、4\}$。那么，根据引理 2，中心 x必须容纳一个颜色相同的堆，而顶点 ℓ_a^i 必须保持为空。由于 x 已被占用，顶点$\ell_b^{j\neq i}$ 必须保持为空。这样就只剩下恰好六个可用顶点（x、 ℓ_b^i、 s^1以及三个 $\ell_a^{j\neq i}$），这与我们假设一个构件能容纳七个堆相矛盾。因此，每个构件最多只能包含六个堆。由于 $S_{initial}$ 由 $6m$ 个堆组成，且共有 m 个构件，每个构件都必须恰好包含六个堆；否则，根据鸽巢原理，某个构件将包含至少七个堆，而我们已证明这是不可能的。

此外，由于引理 2 不允许在构件之间拆分颜色，这六个堆必须恰好由三种颜色组成。

接下来，我们分析每个构件中的堆的放置。假设在一个构件中放置六个堆后，中心顶点 x 为空。根据引理 2， $S_{initial}$ 中的每对颜色必须恰好相隔一个顶点放置，这会迫使 x 的五个邻居全部被占据。因此，第六个堆无法放置而不违反该引理。由此， x 必须包含一个堆。根据引理 2，一个叶节点 ℓ_b^i 必须包含一个相同颜色的堆，于是得到上述指定配置。

最后，我们分析每个构件中的颜色。我们称一个构件具有颜色 c、 c' 和 c''，其中 c 位于 x 和 ℓ_b^i 上。注意， c' 和 c'' 在 $\ell_a^{i\neq j}$ 和 s^1 上的排列无关紧要。如上所述，根据引理 2，为了能够进行三合并，顶点 ℓ_a^i 必须保持为空。因此，只要 c 在 S_{final} 中先于 c' 和 c'' 出现，构件颜色在 $S_{initial}$ 和 S_{final} 中的确切放置位置并不重要。◀

我们再次不失一般性地假设， ℓ_a^1 以及 ℓ_b^2、 l_b^3、 l_b^4始终是空顶点。由于每个构件恰好接收三种颜色，而每种颜色都需要

放置到单个构件上，因此颜色在 $S_{initial}$ 和 S_{final} 中的确切位置并不重要，只要 c 在 S_{final}中最先出现。我们不失一般性地假设，第 j个构件接收了颜色c^j 红色、 c^j 蓝色， c^j_{green} 使得其 x 为 c^j 红色，而 s^1 和 ℓ^2_a 则为 c^j_{green}；见图 4a。现在我们可以证明如下结论。

▶**定理 7.** 在每个含十个顶点的蜘蛛图的不交并上，*Empty Hexasort* 是强 *NP* 困难的。

证明。根据任意一个解 (A, B)，我们构造 I_H 的有效放置策略如下。放置 $S_{initial}$，使每个构件都处于图 4a 所示的配置，并留出 ℓ^2_b、l^3_b 和 ℓ^4_b。对于每个构件 j，我们将三元组 A_j 的三个元素放在这三个顶点上。接下来，处理 $S_{connector}$，将 c^j 红色堆放在 ℓ^1_a 上，并将 c^j 蓝色堆和 c^j_{green} 连接堆放在 x 上。这样，除了叶子上的 S_A 堆外，该构件就空了。最后，我们均匀分配序列 S_{final}：对于每个构件，将三个 c_{black} 堆放在 ℓ^2_a、l^3_a、l^4_a 上，将第四个放在 x 上。这样就会把三元组堆拉到 x 上。由于三元组之和恰好为 B，而四个黑色堆的总和为 4，因此 x 上的总高度变为 $B+4=t$，该堆便会消失。由于我们将此策略应用于所有 m 个构件，整个图就被清空了，且 I_H 是一个是实例。

对于反向证明，我们从引理 6 的论证继续。根据引理 2，ℓ^1_a 不能被 S_A 使用，因此恰好剩下三个顶点 ℓ^2_b、l^3_b和 ℓ^4_b可供放置 S_A。显然，根据鸽巢原理，必须将S_A 中的恰好三个堆放到这些顶点上。接下来放置 $S_{connector}$ 中的堆。由于现在每个构件j 中只有 ℓ^1_a 是空的，相应的 c^j 红色连接堆只能放到这个顶点上，从而合并红色三元组。随后，c^j 蓝色和 c^j_{green} 连接堆被迫在 x上进行三合并，见图 4b。这样一来，该构件中只剩下 ℓ^2_b、l^3_b和 ℓ^4_b，其中包含来自 S_A的堆。要清空该构件，至少需要从 S_{final}取出四个高度为 1 的堆 c_{black}，将三元组堆移到 x上；见图 4c。根据鸽巢原理，每个构件最多能接收四个这样的堆，因此对 $t=B+4$所需高度的贡献为 4。因此，只有当来自 S_A 的三个堆之和至少为 B时，才能清空每个构件。分布在所有 m 个构件中的 S_A 总高度恰好为 mB，因此每个三元组的和都必须恰好为 B，这就意味着存在一个针对(A, B)的 3-划分解。

我们的归约可在多项式时间内完成，并且保持数值不变，因为它构造了 m 个常数规模的构件和一个序列，其中堆叠和颜色的总数为$\mathcal{O}(m)$，而阈值和所有堆高均有$\mathcal{O}(B)$的上界，从而证明了强 NP 困难性。◀

我们再次希望将结果推广到连通图。为此，我们将以类似树的方式连接所有构件的 s^1 顶点：设 T为一个包含 m 个顶点 $t_1, \ldots, t_m$的树。对于 Empty Hexasort 实例 $I^*_H = (G', S, t)$，连通图 G' 由上文的 G 中的 m 个构件与 T组合得到：对于所有 $j \in \{1, \ldots, m\}$，将 G 中第 j个构件的 s^1顶点与 T中的树节点 t_j 合并。注意，如果 T 具有有界高度或度，那么 G'也具有有界高度或度。剩下要证明的是引理 6 仍然适用；在这种情况下，关键在于同一颜色的堆不会分布在多个构件中。

▶ **引理 8.** 将 $S_{initial}$ 放入 I^*_H后，每个构件上的所有堆都必须恰好包含三种颜色，解才能成立。

证明。 我们首先分析一个构件最多能有多少堆。假设某个构件至少包含七堆。由于一个构件有十个顶点，因此至少有一堆必须放在某个叶节点上 ℓ^i_b 对于某个 $i \in \{1,2,3,4\}$。接着，引理 2 迫使中心 x 放置一堆相同颜色的堆。具体来说，按照 G' 的构造方式，同一个引理会迫使

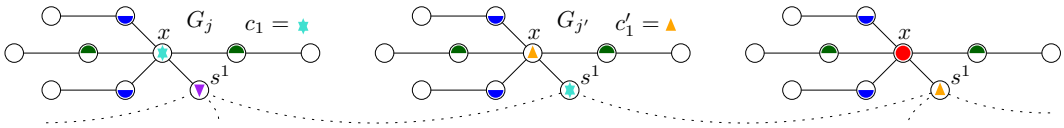


图5 一个连通的　　图 G' 中放置着跨越多个构件的堆叠　　由引理 8 排除 .

顶点 ℓ_a^i 以及所有 $\ell_b^{j\neq i}$ 也都保持为空。因此，没有构件能有七堆或更多堆；由鸽巢原理可知，所有构件都恰好有六堆。

然而，由于这些构件彼此连通，它们可能包含超过三种颜色，且颜色分散在多个构件中。请注意，根据引理 2，唯一能够容纳跨越多个构件的（非黑色）堆叠的顶点是 x 和 s^1，见图5。

▷ 断言9. 如果一个构件中有一种颜色分散在多个构件中，那么对于所有 $i \in \{1,2,3,4\}$，ℓ_b^i 必须为空，并且 x 和 s^1 都必须包含分散的颜色。

证明. 假设某个构件在 x上包含一种分散的颜色。那么，根据引理 2，对于所有 $i \in \{1, 2, 3, 4\}$，任何堆都不会放置在 ℓ_b^i 顶点上。因此，为满足每个构件包含六个堆叠的要求，除了 x之外，s^1 以及所有 $i \in \{1, 2, 3, 4\}$对应的 ℓ_a^i 都必须被占据。由于 $S_{initial}$ 只包含成对的有色堆叠，这五个顶点上至少要分布三种颜色；因此，其中至少有一个顶点包含一种在该构件中只出现一次的颜色。根据引理 2，对于任何 $i \in \{1, 2, 3, 4\}$，ℓ_a^i 都不能包含只出现一次的颜色。因此，两对有色堆叠必须分散在这些 ℓ_a^i 顶点上，并且第四种颜色的一个堆必须放置在 s^1上。如果我们假设分散的颜色放置在 s^1上，同样的逻辑也适用。◁

由此我们可以得出：如果 x 与 s^1 中的任意一个承载了一种在相应构件上仅出现一次的颜色，那么另一个也会承载一种在该构件上仅出现一次的不同颜色。

现在，我们任取这样一对，将其从图中移除，并证明更小的剩余图必定总是包含另一对这样的对，这与 G' 是有限无环图相矛盾。考虑构件 j 的一个 x 顶点，它具有一种在该构件上仅出现一次的颜色 c_1。根据引理 2，c_1 的第二个堆必须位于构件 j' 的 s^1 顶点上，使得 t_j 与 $t_{j'}$ 在 T 中相邻；见图5。如上文在断言9的证明中所述，构件 j' 的顶点 x 必须具有一种在该构件上仅出现一次的不同颜色 c'_1。特别地，构件 j 的 s^1 顶点不能具有颜色 c'_1，否则它会与 c_1 交替，从而违反引理 2。我们移除构件 j，并继续考虑位于更小连通图上的颜色 c'_1 对，该图是剩余构件 j' 的连通分量。这产生了矛盾，因为该论证可以无限继续，而图是有限的。◀

引理 6 的其余部分以及由此得到的下述定理现可得证。

▶ **定理 10.** 有界高度或度数有界的树上的 *Empty Hexasort* 是强 *NP* 困难的。

我们证明，尽管增加了连通性，这些构件仍必须独立求解。在放置 $S_{initial}$、S_A 和 $S_{Connector}$ 之后，清空每个构件仍至少需要四个c_{black} 高度为1的堆叠。合并不同构件的三元组必然需要每个构件使用来自 S_{final} 的超过四个 c_{black} 高度为1的堆叠，因为中心顶点 x 上的三元组必须被拉到 s^1，从而还需要另一个 c_{black} 高度为1的堆。然而，根据鸽巢原理，另一个构件届时将拥有少于四个 c_{black} 高度为1的堆叠

可用，并且至少会剩下一个堆。由于这实际上排除了构件之间的任何交互，该归约的正确性直接由定理 7得出，从而在每个构件上局部强制实现一个有效的3-划分解。此外，该归约保留了该定理中所确立的多项式界。

3 算法

本节探讨解决 Hexasort 的算法方法。注意，判定一个 Fitting 或 Empty Hexasort 实例的朴素穷举方法需要 $O(|V|^{|S|})$ 时间：对于序列 S 中的每个堆叠，我们尝试将其放置在每个可能的顶点上，然后检查最终配置是否可行。我们改为系统地探索可达棋盘配置的空间，从而设计出一种 Hexasort 动态规划方法，其运行时间关于图的规模呈指数级。

▶ **定理 11.** *Fitting* 和 *Empty Hexasort* 可以在 $O(|S| \cdot (|C| \cdot t)^{|V|})$ 时间内判定 .

证明。我们方法的核心是一个布尔表，若且唯若棋盘配置 δ 在放置了 S中的前 i 个堆叠后可以达到，则表项 DP$[i, \delta]$ 为真。我们初始化动态规划：仅将唯一的空配置 δ_0对应的 DP $[0, \delta_0] =$ 设为真，其中对所有 $v \in V$均有 $\delta_0(v) = \perp$ 。递推关系逐步处理该序列：对于从 1 到 m 的每个索引 i，以及每个满足 DP$[i-1, \delta] =$ 为真的配置 δ，我们考虑所有可能的有效步。一步是将当前堆叠 $s_i = (c, h)$ 放置在任意空顶点上，并将所得配置 δ'对应的 DP$[i, \delta'] =$ 设为真。当序列 S处理完毕时，若 DP$[m, \delta_0] =$ 为真，则该 Empty Hexasort 实例为肯定实例。若对任意 δ，DP$[m, \delta_0] =$ 均为真，则该 Fitting Hexasort 实例为肯定实例。

关于运行时间，注意到共有 $\mathcal{O}((|C| \cdot t)^{|V|})$ 种配置，因为每个 $|V|$ 顶点都有 $|C| \cdot (t-1) + 1$ 种可能状态。由于算法会遍历该序列，并针对每个可达状态考虑至多 $|V|$ 种放置选项，因此总时间复杂度为 $\mathcal{O}(|S| \cdot (|C| \cdot (t-1) + 1)^{|V|} \cdot |V|) = \mathcal{O}(|S| \cdot (|C| \cdot t)^{|V|})$。◀

虽然图越大，该算法运行得越慢，但我们现在将证明，一旦图足够大，能够为每种颜色提供专用空间，问题就显然有解。为此，我们需要针对这两种问题变体分别采用不同的方法。

▶**引理 12.** 任何 G 包含大小至少为 $|C|$ 的匹配的 *Fitting Hexasort* 实例都有解。

证明。 对于每种颜色 $c \in C$，我们从匹配中预留一条边。当一个堆 (c, h)到达时，就将它交替放置在两个端点顶点之一上。这样，在任何时刻，每种颜色至少都有一个空闲顶点。因此，所有堆都可以放置。◀

▶ **引理 13。** 任何包含度为 $|C| \cdot t$的顶点的 *Fitting Hexasort* 实例 G 都是肯定的。

证明。 如果 G 包含一个顶点 x 其度至少为 $|C| \cdot t$，我们可以把 $|S|$ 中的堆放到它的任意一个邻居上；只有当某种颜色已经有相邻的堆时，才将堆直接放到 x 上，从而消除这种颜色的所有堆。由于任何颜色在图上都不会超过 t 个堆，而放在 x 上的任何堆都会立即消失（其 t 个邻居合并后的大小肯定超过 t），因此所有堆都可以放置，所以该实例总是肯定的。◀

▶**观察 14。** 任何一种 *Empty Hexasort* 实例，如果忽略所有高度为 t 的堆后，某种颜色 c 的堆叠总和小于 t，且最后一个颜色为 c 的堆高度小于 t，则该实例为负。

12 Hexasort – 图上的颜色堆叠的复杂性

▶**引理 15**。 任何一种 *Empty Hexasort* 实例，若其根据观察 *14* 并非显然为负，且 G 包含 $|C|$ 个互不重叠的蜘蛛图，每个蜘蛛图都有两条长度为二的腿和一条长度为一的腿，则该实例为正。

证明。对于每种颜色 $c \in C$，我们预留一个度为3的蜘蛛，其中心 x 与三条腿 s、$\ell_a^1\ell_b^1$、$\ell_a^2\ell_b^2$ 的首个顶点相邻。令 $z = (c, h_z)$ 为 S 中最后一个颜色为 c 的堆叠。若 $h_z = t$ 或 $sum(c, S) - h_z < t$，则将所有颜色为 c 的堆叠交替放在蜘蛛上任意两个相邻顶点上。否则，令 X 为颜色为 c 的堆叠的任意一个极小集合，其中不包括任何高度为 t 的堆叠，且满足 $\Sigma X \geq t$。注意，假设观察 14 不会使该实例显然为否，则这样的集合必定存在。现在令 $y = (c, h_y)$ 为 S 中 X 的第一个堆叠，并将其放在 s 上。将 X 的所有其他堆叠（z 除外）交替放在 ℓ_a^1 和 ℓ_b^1 上，使得 X 的最后一个堆叠放在 ℓ_a^1 上。将颜色为 c 的所有其他堆叠（z 除外）交替放在 ℓ_a^2 和 ℓ_b^2 上，使得其中最后一个放在 ℓ_a^2 上。将最后一个堆叠 z 放在中心 x 上。根据 X 的极小性，无论 $z \in X$ 是否成立，此时 s 都包含 y，并且包含一个在与 y（或与 z 和 y）合并时高度至少达到 t 的堆叠。当 z 放在 x 上时，它会与 ℓ_a^2 上任何剩余的堆叠合并 ℓ_a^1 和 s，从而使颜色 c 消失。因此，可以放置 S 中的所有堆叠，并最终清空该图。◀

对于 Fitting Hexasort，结合这些结果，我们现在可以证明，以颜色数量和阈值为参数时，该问题具有固定参数可解性。

▶ **定理 16**。 *Fitting Hexasort* 可在 $O(f(|C|, t) \cdot |S|^2)$ 时间内判定，其中 f 为某个函数。

证明。 首先注意，如果 G 足够小，使得 $|V|$ 被 $|C|$所界定，那么根据定理 11，该实例可在 $\mathcal{O}((|C| \cdot t)^{|C|} \cdot |S|)$ 时间内求解。反之，如果 G 较大且包含一个大小至少为 $|C|$的匹配，那么根据引理 12，该实例必然为肯定实例。可以在多项式时间内检查是否存在这样的匹配；算法综述见 [3] 。此外，根据引理 13，所有包含度至少为 $|C| \cdot t$ 的顶点的实例也必然为肯定实例。剩下需要处理的是 G 较大、但度有界且不包含足够大的匹配的情况。等价地，后一个条件意味着 G 具有一个大小不超过 X $2 \cdot |C|$ [9,10]的顶点覆盖。

令 V^0 表示孤立顶点的集合。容易看出，若 $|V^0| \geq |S|$，则该实例总是正实例——将每个堆放在单个顶点上。因此， G 具有有界的顶点覆盖 X、没有高度数顶点，并且孤立顶点 V^0的数量有界。这意味着，若不考虑孤立顶点 V^0，图 G 的规模以 $|C|^2 \cdot t$为界。这是因为 X 中至多 $2 \cdot |C|$ 个顶点的每一个在 X之外至多可以有 $|C| \cdot t$ 个邻居，并且任一非孤立顶点都必须要么位于 X 中，要么在 X中有一个邻居。定理 11 中动态规划的运行时间可分解为 $O(|S| \cdot (|C| \cdot t)^{|V \setminus V^0|} \cdot (|C| \cdot t)^{|V^0|})$。由于孤立顶点在结构上相同且不能相互影响，我们无需跟踪它们的具体配置。相反，我们只需统计非空孤立顶点的数量，从而用线性因子$|V^0|$简化指数因子 $((|C| \cdot t)^{|V^0|})$。由此得到运行时间 $O(|S| \cdot (|C| \cdot t)^{|V \setminus V^0|} \cdot |V^0|)$。由于 $|V \setminus V^0|$ 以 $|C|^2 \cdot t$为界，且 $|V^0|$ 以 $|S|$为界，复杂度简化为 $O((|C| \cdot t)^{|C|^2 \cdot t} \cdot |S|^2)$，在这最后一种情形下满足我们的界。◀

4 结论与未来工作

在本研究中，我们考察了热门游戏 Hexasort 在 Empty 和 Fitting 两种变体中的计算复杂性。在第 2 节中，我们介绍了多种 NP 困难性归约，

首先，我们将 Partition问题归约为 Empty Hexasort，其中只有一种颜色，且有两条大小为二的独立路径。通过向这样的实例添加四个带有新颜色的堆，将 Empty 归约为 Fitting Hexasort，这一结果也适用于允许使用五种颜色的后一种变体。我们进一步利用 3-划分证明，Empty Hexasort 是强 NP 困难的，即使将其限制在高度或度数有界的树上也不例外。这表明，即便对图施加严格的结构限制，并且颜色数量为常数或数值受输入规模多项式界定，该问题仍然难以求解。

与此同时，在第 3 节中，我们确定了几类能够使用高效算法求解的实例。我们提出了一种动态规划算法，可在任意图上用时 $\mathcal{O}\big(|S| \cdot (|C| \cdot t)^{|V|}\big)$ 求解这两个变体。请注意，这立即意味着，当阈值 t 相对于输入规模为多项式有界时，常数规模图上的问题具有可处理性。我们还证明，对于规模足够大的图，如果每种颜色都可以在图中各自独占的部分上独立求解，那么相应实例总是可处理的。基于此，我们证明了以颜色数量和阈值为参数时，Fitting Hexasort 是固定参数可处理的。

因此，对于 Fitting Hexasort，我们得到了对图规模$|V|$、颜色数量 $|C|$ 和阈值 t三者中任意两者进行界定的结果。当 $|V|$ 和 $|C|$为常数时，该问题是（弱）NP-困难的；而当 $|V|$ 为常数且t 为多项式有界，或当 $|C|$为常数且 t时，问题则变得可处理。 t 被常数界定时的复杂性仍是一个开放问题。此外，若能将我们的结果拓展为证明在 $|C|$为常数时具有强 NP 困难性，将会很有意义。类似地，我们关于 Empty Hexasort 的结果也是紧确的：当图规模被常数界定，且消除阈值被输入规模的多项式界定时，我们可以得到一个多项式时间算法。相比之下，只采用其中任一界定都会导致 NP 困难性。我们将以下问题留作开放问题：能否通过同时界定阈值和颜色数量（例如，像处理 Fitting 情形时那样使用引理 15），使 Empty Hexasort 变得可处理？这里的主要挑战在于论证：任何仅包含少数几个这种更复杂结构副本的图，其自身也必定具有简单结构。

关于在游戏可玩版本中发现的 Fitting Hexasort 实例，其六边形网格相对于颜色数量而言通常足够大，因此能够容纳足够大的匹配。相较之下，使这些实例变得复杂的是，这里的堆叠还允许包含多种不同颜色的六边形；这也是我们的结果可以进一步扩展的一个方向。

参考文献

1 Aviv Adler、Erik D. Demaine、Adam Hesterberg、Quanquan Liu 和 Mikhail Rudoy。即使只有两种颜色和列，Clicko-mania 也很难。在 各种娱乐主题的数学（*MOVES 2015*），第 2 卷，第 325–363 页。普林斯顿大学出版社，2017 年 . doi:10.23943/princeton/9780691171920.003.0018。2 Mirela Alistar 和 Urs Gaudenz。Opendrop：一个供个人使用的生物芯片集成式 DIY 平台。《生物工程》，4(2)，2017 年。 doi:10.3390/bioengineering4020045。3 Ran Duan 和 Seth Pettie。最大权匹配的线性时间近似算法。《*ACM*杂志》，61(1)，2014年1月。 doi:10.1145/2529989。4 Michael R. Fellows、Frances A. Rosamond、Maise Dantas da Silva 和 Uéverton S. Souza . 洪水填充游戏复杂性综述，第357–376页。施普林格国际出版公司，查姆，2018年。 doi:10.1007/978-3-319-98355-4_20。5 Michael R. Garey 和 David S. Johnson。“强”NP完全性结果：动机、示例与启示。《*ACM*学报》（*JACM*），25(3):499–508，1978年。

14 Hexasort – 图上的颜色堆叠的复杂性

6 Michael R. Garey 和 David S. Johnson。《计算机与难解性：*NP* 完全性理论指南》。W. H. Freeman and Company，旧金山，1979年。参见附录 A3，问题 SP12（PARTITION问题），第223页。7 Luciano Gualà、Stefano Leucci 和 Emanuele Natale。宝石迷阵、糖果传奇及其他三消游戏均为（NP）难题。载于*2014*年 *IEEE* 计算智能与游戏会议（*CIG 2014*）论文集，第1–8页。IEEE，2014年。 doi:10.1109/CIG.2014.6932866。8 Yuki Iburi 和 Ryuhei Uehara。匹配消除谜题的计算复杂性。载于 Andrei Z. Broder 和 Tami Tamir 编，第*12*届算法趣味国际会议（*FUN 2024*）论文集，第291卷，*LIPIcs*，第17:1–17:10页。Schloss Dagstuhl - Leibniz-Zentrum für Informatik，2024年。 doi:10.4230/LIPICS.FUN.2024.17。9 Hugo Jacob、William Lochet 和 Christophe Paul。关于带宽的基于树的变体及禁止简单拓扑子式，2025年。 arXiv:2502.11674。10 Robert Sasak。比较17个图参数。硕士论文，卑尔根大学，2010年。11 Steve Mould。Opendrop——电润湿器件演示 [视频]。 https://www.youtube.com/watch?v=rf-efIZI_Dg，2015年。12 Takayuki Yato 和 Takahiro Seta。寻找另一个解的复杂性与完备性及其在谜题中的应用。*IEICE*基础电子、通信与计算机科学汇刊，E86- A(5):1052–1060，2003年。网址：https://globals.ieice.org/en_transactions/fundamentals/10.1587/e86-a_5_1052/_p。