arXiv:2601.07566v1 [cs.DS] 12 Jan 2026

动态（$\Delta+1$）顶点着色

Noam Benson-Tilsen

2026年1月12日

摘要

本文综述了动态和次线性图着色领域近期的若干研究成果。图着色问题受到广泛研究，其应用场景包括网络拓扑控制、约束满足和实时资源调度。图着色算法称为着色器。第 1 节定义了图着色、动态模型，以及动态模型中图算法性能的概念。特别地，文中介绍并阐释了（$\Delta+1$）着色、次线性性能，以及 oblivious 对手和自适应对手。第 2 节概述了 [BCKLVRV17] 提出的两个近似最优动态顶点着色器，作为（$\Delta+1$）着色器的预备知识。第 3 节介绍了动态（$\Delta+1$）着色的最新进展。本节涵盖两篇论文（[BCHN18] 和 [BGKLS22]），它们将动态（$\Delta+1$）着色从朴素算法的 $O(\Delta)$ 期望摊还更新时间改进至 $O(\log\Delta)$，进而达到高概率下的 $O(1)$。第 4 节介绍了 [BRW25], 的研究成果，该研究提出了一种用于（$\Delta+1$）着色的次线性算法，并将 oblivious 对手推广为自适应对手。

目录

1 预备知识 2
1.1 图着色 .2 1.2 动态模型 .2 1.3 性能 .2

2 预备知识：近似最优动态着色 3
2.1 顶点插入 .3 2.2 性能 .3

3 动态（$\Delta+1$）-着色（oblivious 对手） 4
3.1 对数时间动态 $(\Delta+1)$-着色 .4 3.1.1 层级数据结构 .4 3.1.2 算法 .6 3.1.3 分析 .7 3.2 常数时间动态 $(\Delta+1)$-着色 .8 3.2.1 算法 .9 3.2.2 分析 .10

4 动态（$\Delta+1$）-着色（自适应对手） 12

5 致谢 14

1 预备知识

1.1 图着色

图 $G := (V, E)$ 的顶点着色 $\chi : V \longrightarrow C$ 是将顶点 V 赋予颜色 $\mathcal{C} := \{1, \ldots, k\} \subseteq \mathbb{N}_+$ 的一种映射。若对于所有边 $\{u, v\} \in E$， χ 都满足 $\chi(u) \neq \chi(v)$，则称其为正常着色。若 χ 是 G 的一个正常着色，且最大颜色为 $\max_{c \in C} c = k$，则称 k 可着色。使得 G 存在正常着色的最小 k 称为色数 $\chi(G)$。

然而，一般而言，确定 $\chi(G)$ 是 NP-困难的，即使对于 $\chi(G) = 3$ 也是如此。这促使我们研究近似算法（§2），以及用于寻找 $(\Delta + 1)$-着色的算法（§3 及后续章节），因为这类着色总是存在。关于 $(\Delta + 1)$-着色性能的更多基础知识，请参见 §1.3 。

1.2 动态模型

在动态模型中， G 会经历一系列更新 $\mathcal{U}$，包括边和顶点的插入与删除。请注意，删除顶点或删除边都不会破坏恰当着色。在顶点着色的背景下，可以选择仅考虑（1）顶点（及其关联边）的插入，或（2）边插入。

1. 边更新可以通过以下方式分解

   (a) 通过删除并重新添加某条边的一个端点来模拟任意边插入；(b) 将任意边删除的处理延迟到其一个端点被删除时；或者

2. 假设（不失一般性） G 一开始就具有其在 $\mathcal{U}$ 期间可能达到的最大阶数 n ，即可将顶点更新剥离出来；在这种情况下，就维护正常顶点着色而言，顶点更新与边更新相同。

在§2 中采用第一种方法；在§3中采用第二种方法。

要在 $\mathcal{U}$ 期间维持正常着色，就需要对顶点重新着色；也就是说，在插入一条边时，动态图着色器必须通过修改某些顶点 $V' \subseteq V$ 的颜色 $\chi(V')$ 来更新当前着色 χ 。在 $\mathcal{U}$ 期间，任意顶点的最大度记为 Δ，即 $^{\mathcal{U}}\max_{v \in V(G^{\mathcal{U}})} d(V)$。

1.3 性能

动态图着色器的性能定义为每次更新的重新着色次数（期望摊销值），亦称更新时间，具体如下。对于某个底层输入集合 $\mathscr{U}$ ，令更新时间 $T_{\mathcal{A}}(\mathcal{U})$ 表示动态着色器 $\mathcal{A}$ 针对从对称群 $\mathfrak{S}_{\mathscr{U}}$ 中抽取的输入序列 $\mathcal{U} \sim \mathfrak{S}_{\mathscr{U}}$（其元素属于 $\mathscr{U}$）的性能。[1] 那么，接下来的任务是求出

$$\arg\min_{\mathcal{A}} \mathop{\mathbb{E}}_{\mathbf{U} \sim \mathfrak{S}_{\mathscr{U}}} [T_{\mathcal{A}}(\mathcal{U})].$$

线性时间动态 $(\Delta+1)$ 着色器很简单，但次线性动态 $(\Delta+1)$ 着色器并非如此。即使输入是非自适应的，情况依然如此——也就是说，输入序列的生成独立于着色器，例如从 $\mathfrak{S}_{\mathscr{U}}$ 中均匀随机抽取。本报告末尾将讨论一篇论文，该论文针对自适应对抗

[1] 如报告中的大多数部分一样，仅考虑边插入时， T 将特化为 $T(\mathcal{U}, n)$ ，其中 n 表示初始为空的图的阶。

输入序列，这些序列可以根据着色器迫使其表现出病态的最坏情况行为。在 [BCK-LVRV17], 在§2中介绍的着色器使用 $k(N) \cdot \chi(G)$ 种颜色，其中参数 $k > 0$ 取决于图 N 在所有更新过程中顶点数量的最大值 G 。每次更新的预期摊销重新着色次数与两个着色器的近似紧致性之间存在权衡。具体而言，对于参数为 $d > 0$的可 k着色图，给出了两种算法，其中因子 $O(N^{1/d})$ 在性能与近似紧致性之间进行权衡。

2 预备知识：近似最优动态着色

[BCKLVRV17] 提供了同一算法的两种变体。第一种算法， $\mathcal{A}_1$，采用单层顶点划分； $\mathcal{A}_2$ 则采用两层划分。[2] 在这两种算法中，第一层划分内的顶点都使用 $\chi(G)$ 种颜色进行适当着色。此外，在这两种算法中， V 被划分为若干桶 $\bigoplus_{i=1}^{d} V_i$ ，使得 V_i的容量为 $N_R^{(i-1)/d}$ ，其中 N_R 是 n 在上一次更新时的值。注意，

$$\sum_{i=1}^{d} N_R^{(i-1)/d} = \frac{N_R - 1}{N_R^{1/d} - 1} \in O(N_R^{1-1/d}) \subseteq O(n).$$

然而，桶并不会达到其容量。相反， V_i 的不变量高点至多为$h_i = {N^{i/}_R}^d - N_R^{(i-1)/d}$个顶点。在 $\mathcal{A}_2$ 中，每个第一层桶 V_i 中至多 h_i 个顶点被划分到 ${N^1/_R}^d - 1$ 个桶 $V_{i,1}, \ldots, V_{i,\lceil N_R^{1/d} - 1 \rceil}$ 每个桶的容量为 $N_R^{(i-1)/d}$ ，另有一个空的重置桶。也就是说，在 $\mathcal{A}_2$ 中，子桶从左向右紧凑排列，以便为重置桶留出空间。重置桶用于插入操作。

2.1 顶点插入

插入顶点 v 时，将其放入 V_1。在 $\mathcal{A}_2$中， v 进入第一个空的子桶 $V_{1,\ j}$。如果V_1 的顶点数少于 h_i ，则为 v 分配一种在 V_1局部可用的剩余颜色。在 $\mathcal{A}_2$中，这种颜色在V_1局部可用，$_j$。但是，如果将 v 插入 V_1 会违反高点不变量，则将 V_1 中的所有顶点移至 V_2 （或者在 $\mathcal{A}_2$中，将其移至空子桶；根据不变量的保证，该空子桶必定存在）。如果可能，则对转移过来的顶点重新着色；如果它们也违反 V_2 中的不变量，那么V_2 就会被向上转移至 V_3。这一过程可能一直向上传播到最后一个划分 V_d。在这种情况下，整个图将被重置并重新着色。

2.2 性能

在最坏情况下， $\mathcal{A}_1$ 需要在所有 d 个桶中，每个桶移动并重新着色 O（$N^{1/d}$）个顶点。由于颜色数量绰绰有余，可以为每个顶点分配一种不同的颜色，因此每次重新着色都很简单。总计，重新着色次数为 O（$dN^{1/d}$）。类似地，在 $\mathcal{A}_2$中，每个第一级划分中的顶点最多移动并简单地重新着色一次，因此重新着色总次数为 O（d）。

对于 $\mathcal{A}_1$，每个桶使用 $O(\chi(G))$ 种颜色，并且共有 $O(d)$ 个桶，因此得到一个 $O(d)$-近似着色。对于 $\mathcal{A}_2$，每个划分中的每个桶使用 $O(\chi(G))$ 种颜色，并且每个 d 划分中有 $O(N^{1/d})$ 个桶，因此得到一个 $O(dN^{1/d})$-近似着色。

[2]论文以相反的顺序介绍了 $\mathcal{A}_1$ 和 $\mathcal{A}_2$，这样不够清晰。我也将两者的介绍合并在一起，因为它们只有一处不同。

表 1：性能与紧致性

	重新着色	近似紧致性
$\mathcal{A}_1$	$O(dN^{1/d})$	$O(d)$
$\mathcal{A}_2$	$O(d)$	$O(dN^{1/d})$

3 动态$(\Delta+1)$-着色与 oblivious 对手

本节将概述 [BCHN18] 和 [BGKLS22]所用的技术。在下文中，所有更新均为边插入，并且 $k := \Delta + 1$，因此 $\mathcal{C} = \{1, \ldots, \Delta + 1\}$。 χ^* 表示 $\mathcal{U}$期间的着色； $\chi^* = \chi$ 仅当 $\mathcal{U}$ 完成时，而在 $\mathcal{A}$ χ^* 的原子步骤之间，并不保证着色是合法的。

对数时间动态$(\Delta+1)$着色

存在一种更新时间为 $O(\Delta)$ 的简单算法。当插入边 uv 时，如果 $\chi^*(u) = \chi^*(v)$，则从 x 的 u 和 v 中选择一个，并为 x 分配一个其任一邻居都未使用的颜色。这种颜色称为 x的空白颜色，这类颜色的集合为 $\mathcal{B}_{N(x)}$。注意，对于任意 $x \in V$， $|N(x)| \leq \Delta < \Delta + 1 = |C|$，因此根据鸽巢原理，空白颜色总是存在。

3.1.1 层级数据结构

为了将更新时间改进为 $O(\log \Delta)$， [BCHN18] 提出了 $\mathcal{A}_{\log}$，这是一种具有对数更新时间的动态着色器，使用层级数据结构 ℓ： $V \longrightarrow \{4, \ldots, L\}$，其中对于足够大的常数 $\beta > 0 \in \mathbb{N}$， $L := \log_\beta \Delta$。换言之， $\ell(v)$ 是 v的层级，其中可能的层级数量为 $O(\log \Delta)$。引入 ℓ 的动机是为重新着色时应使用哪些颜色提供一种启发式方法，从而最小化级联重新着色链的预期长度。对于每个顶点 v，令 v 的下邻居 $\mathcal{L}(v)$ 为 v 的层级低于 $\ell(v)$ 的邻居；上邻居 $\mathcal{H}(v)$ 为层级至少为 $\ell(v)$ 的邻居；同层邻居 $\mathcal{S}(v)$ 为层级等于 $\ell(v)$ 的邻居。见图1。$\mathcal{A}_{\log}$ 维护两个不变量：

不变量 3.1. 对于每个$v \in V$，下邻居的数量 $\phi := |\mathcal{L}(v)|$ 至少为 $\beta^{\ell(v)-5} \in \Omega(\Delta)$ 。

不变量 3.2. 对于每个 $v \in V$，下邻居和同层邻居的数量 $|\mathcal{L}(v) \cup \mathcal{S}(v)|$ 至多为 $\beta^{\ell(v)} \in O(\Delta)$。

令 $\mathcal{C}_{\mathcal{H}(v)}$ 为某个上邻居所使用的颜色，这些上邻居属于 v， $\mathcal{U}_{\mathcal{L}(v)}$ 为恰好一个下邻居所使用的颜色，该下邻居属于 v，而 $\mathcal{M}_{\mathcal{L}(v)}$ 为至少两个下邻居所使用的颜色，这些下邻居属于 v。令 $\mathcal{D}_{N(v)} := \mathcal{C} \setminus \mathcal{C}_{\mathcal{H}(v)} \setminus \mathcal{M}_{\mathcal{L}(v)}$ 为没有上邻居使用且至多被一个下邻居使用的颜色，该上、下邻居属于 v 。令 τ_v 为 v 最后一次重新着色时的时间戳。关于这些定义，请参见表 2 。集合用花体字母表示。顶点集合以函数 $V \longrightarrow \mathbb{P}(V)$ 命名，而颜色集合的名称则以下标形式标注其定义所依赖的顶点集合。

$\mathcal{D}_{N(v)}$ 是最重要的对象。在最坏情况下的边插入中，可能会发生重新着色的递归级联，而 $\mathcal{D}_{N(v)}$ 将有助于将递归扇出限制为 1，方法是公开

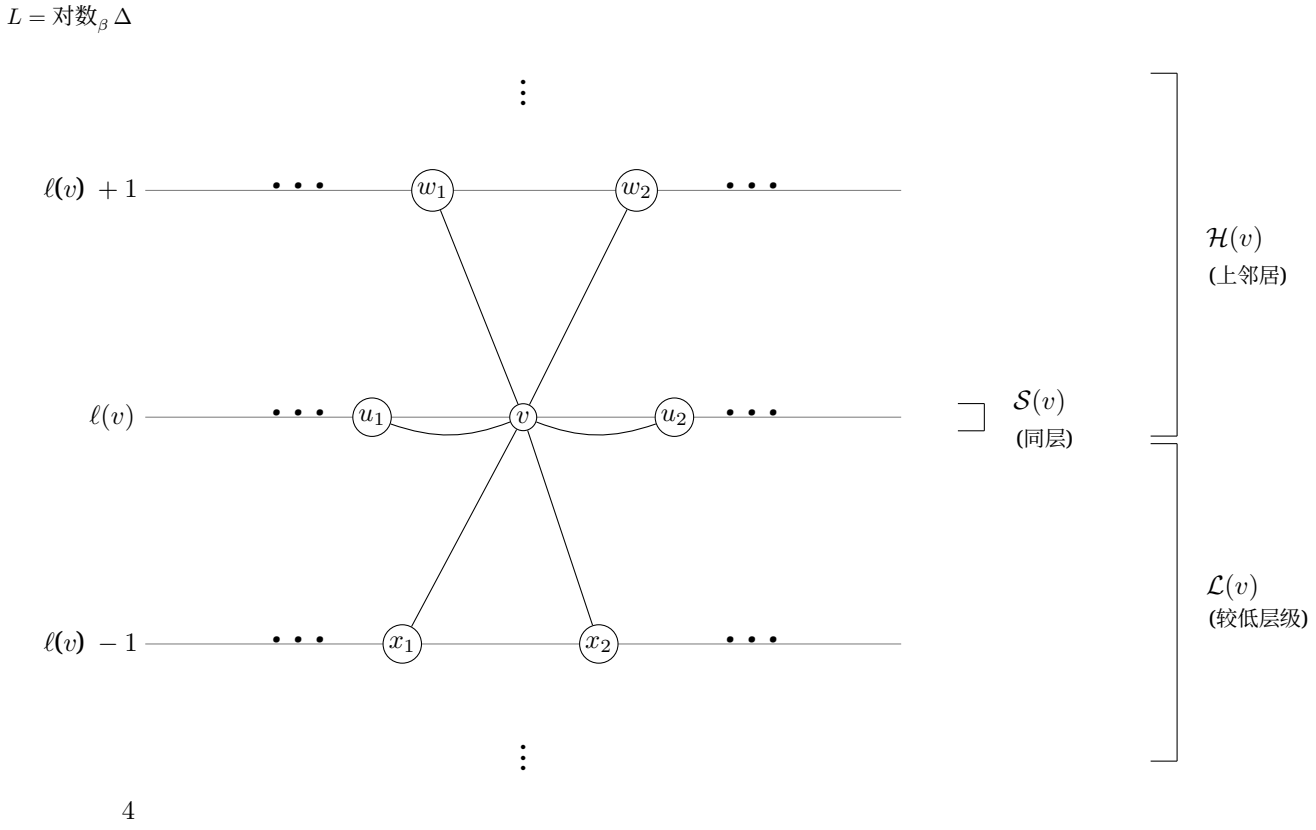

图1：每个 $v \in V$ 的 $\ell(v)$，其上邻居为 $\mathcal{H}(v)$，同层邻居为 $\mathcal{S}(v) \subseteq \mathcal{H}(v)$，下邻居为 $\mathcal{L}(v)$。

重新着色的顶点的下邻居中很少使用的颜色。根据引理 3.3， $\mathcal{D}_{N(v)}$ 足够大，因此从中均匀随机采样时，不太可能与正在重新着色的顶点发生冲突。

引理 3.3。 如果不变量*3.1* 和不变量 *3.2* 成立，$\left|\mathcal{D}_{N(v)}\right| \in \Omega(\Delta)$ 对每个 $v \in V$.

证明。 令 $x := \left|\{v \mid \chi^*(v) \in \mathcal{U}_{\mathcal{L}(v)}\}\right|$ 表示颜色在 $N(v)$ 中恰好使用一次的下邻居的数量。注意 $\left|\mathcal{M}_{\mathcal{L}(v)}\right| \leq \frac{\phi - x}{2}$： 除了颜色恰好使用一次的 x 个下邻居之外， v 的每个 ϕ 下邻居都有至少一个颜色相同的搭档。因此

$$\left|\mathcal{D}_{N(v)}\right| \geq \left|\mathcal{C} \setminus \mathcal{C}_{\mathcal{H}(v)}\right| - \frac{\phi - x}{2} = (\Delta + 1) - (\Delta - \phi) - \frac{\phi - x}{2} \leq \frac{\phi}{2} + 1,$$

因此， $\left|\mathcal{D}_{N(v)}\right| \in \Omega(\phi) = \Omega(\Delta)$。 □

表 2：每个 $v \in V$ 的层级数据结构数据

Name	定义	描述
L	$\log_\beta \Delta$	最高层级
$\ell(v)$	$\in \{4, \ldots, L\}$	v 的层级
$\deg(v)$	$\lvert N(v)\rvert$	v的度
τ_v	$\in \mathbb{N}$	v 上次重新着色的时间
$\mathcal{L}(v)$	$\{u \mid \{u,v\} \in E \wedge \ell(u) < \ell(v)\}$	v 的低层级邻居
$\mathcal{H}(v)$	$\{u \mid \{u,v\} \in E \wedge \ell(u) \geq \ell(v)\}$	v 的层级至少与 v相同的邻居
$\mathcal{S}(v)$	$\{u \mid \{u,v\} \in E \wedge \ell(u) = \ell(v)\}$[1]	与 v层级相同的 v 的邻居
$\mathcal{B}_{N(v)}$	$\{c \mid \forall u \in N(v)\colon\ \chi^*(u) \neq c\}$	v的所有邻居均未使用的颜色
$\mathcal{C}_{\mathcal{H}(v)}$	$\{c \mid \exists u \in \mathcal{H}(v) \mid \chi^*(u) = c\}$	v的上邻居的颜色
$\mathcal{U}_{\mathcal{L}(v)}$	$\{c \mid \exists!\ u \in \mathcal{L}(v) \mid \chi^*(u) = c\}$	v的恰好一个下邻居的颜色
$\mathcal{M}_{\mathcal{L}(v)}$	$\{c \mid \exists_{\geq 2} u \in \mathcal{L}(v) \mid \chi^*(u) = c\}$[2]	v的至少两个下邻居的颜色
$\mathcal{D}_{N(v)}$	$\mathcal{C} \setminus \mathcal{C}_{\mathcal{H}(v)} \setminus \mathcal{M}_{\mathcal{L}(v)}$	v没有上邻居且至多有一个下邻居的颜色

[1] 注意，这是 $\mathcal{H}(v)$ 的子集。[2] 注意，这是 $\mathcal{C} \setminus \mathcal{B}_{N(v)} \setminus \mathcal{C}_{\mathcal{H}(v)} \setminus \mathcal{U}_{\mathcal{L}(v)}$。

3.1.2 算法

算法 1： 边插入$^{\log}$

输入： G；删除或插入时的 $\{u,v\} \leftrightsquigarrow E(G)$； ℓ 以及相关数据结构

输出： 更新后的（$\Delta+1$）-着色 $\chi*$

更新层级；

如果 $\{u,v\}$ 正在被插入，并且 $\chi^*(u) = \chi^*(v)$ **那么**

$x \leftarrow \underset{x \in \{u,v\}}{\text{取最大值}}\ \tau_w$; `// More-recently recolored`

重新着色$^{\log}(x)$;

算法 2： 更新层级

输入： G; χ^*； ℓ 及相关数据结构
输出： ℓ 及满足不变量 3.1 和不变量 3.2 的相关数据结构

当 不变量 *3.1* 或不变量*3.2* 被违反时 **执行**
  如果 存在 $x \in V$违反不变量 *3.2* （拥有多于 $\beta^{\ell(x)}$ 个下邻居）**则**
    找到使 x 具有 $|\mathcal{L}(x)| + |\mathcal{S}(x)| \leq \beta^k$的最小层级 $k > \ell(x)$;
    $\ell(x) \leftarrow k$;
    更新所有辅助数据结构;
  else
    找到一个顶点 $x \in V$违反不变量 3.1 (拥有少于 $\beta^{\ell(x)-5}$ 同层邻居);
    如果 存在一个层级 $k < \ell(x)$ 对于 x 而言，其将拥有 $|\mathcal{L}(x)| \geq \beta^{k-1}$ **那么**
      令 k' 为满足条件的最大层级;
      $\ell(x) \leftarrow k'$;
    else
      $\ell(x) \leftarrow 4$;
  更新任意辅助数据结构;

算法 3： 重新着色log

输入： G; $v \in V(G)$ 用于重新着色； χ^*； ℓ 及相关数据结构
输出： χ^*，其颜色为新颜色，适用于 v

$c \sim \mathcal{D}_{N(v)}$ 均匀随机地;
$(v) \leftarrow c$;
更新相关的辅助数据结构;
如果 $\exists w \in \mathcal{L}(v) : \chi^*(w) = c$ **那么**
  重新着色log(w);

3.1.3 分析

插入（算法 1）依赖两个子程序：更新层级（算法 2）和重新着色log（算法 3）。要将一条边 $\{u, v\}$ 插入到 E（G）中，首先修复层级数据结构 ℓ，使不变量 3.1 和 3.2 得到满足，然后将 $\{u, v\}$ 添加到 E（G）。如果 χ^*（u） $= \chi^*$（v），则对最近一次重新着色的那个进行重新着色。此外，还维护一个由违反不变量的 脏 顶点组成的辅助列表。

UPDATE LEVELS 的思路是移动 ℓ 中的顶点，以确保不变量成立，然后在需要时修复 χ^*。首先，只要存在一个顶点 x 的下邻居过多（超过 $\beta^{\ell(x)}$），就将 x 移动到能够使不变量 3.2 在该层级对其成立的最低层级。注意， x 在 ℓ 中逐层上移时， x 的下邻居数量增长得比 $\beta^{\ell(x)}$ 慢；特别是，始终可以通过这种方式修复 x。另一方面，提升 x 可能会使其他一些顶点变脏，但在 while 循环结束时，相对于不变量 3.2，所有顶点都是干净的。（证明见 [BCHN18] 引理 3.1。）在针对不变量 3.1 清理每个顶点后，UPDATE LEVELS 会针对不变量 3.1 清理这些顶点。具体而言，只要存在一个脏顶点 x，就将 x 下移到使其保持干净的最高层级。不变量 3.2 的清理循环所具备的性质在此循环中同样成立。

UPDATE LEVELS 是 EDGE INSERTION 的来源$^{\log}$的 $O(\log \Delta)$ 运行时间。关于 UPDATE LEVEL 的 $O(\beta) = O(\log \Delta)$ 运行时间的证明细节见 [BGKLS22] 定理 3.2。更重要的是，EDGE INSERTION 的另一组成部分$^{\log}$ — RECOLOR$^{\log}$ — 可以在常数期望摊销时间内实现。

在 RECOLOR$^{\log}$ 中，如果 $\chi^*(u) \neq \chi^*(v)$，则 χ^* 仍保持合法。否则，将 x （ u 和 v 中较近一次重新着色的顶点）记录在 τ_u， τ_v 中，并对其重新着色。为了对 x 重新着色，将 $\chi^*(x)$ 设置为一个随机样本$c \sim \mathcal{D}_{N(x)}$。注意， $\mathcal{D}_{N(x)}$ 中的每种颜色都由 χ^* 分配给 w 的邻居 x，且至多分配给一个邻居。如果 $c = \chi^*(w)$，则 $\{x, w\}$ 是一条单色边。为修复这一冲突，RECOLOR$^{\log}$ 在 w 上递归执行。注意，这可能沿着 $O(\beta^{\ell(x)})$ 层级向下级联，每一层都需要从 $\mathcal{D}_{N(x)}$ 中采样，并扫描 $\mathcal{L}(x)$ 以获取 w。由于通常情况下 $\left|\mathcal{D}_{N(x)}\right| \gg \beta^{\ell(x)}$，采样在技术上较为困难，但在 $\mathcal{D}_{N(x)}$ 的前 $\beta^{\ell(x)}$ 个元素处截断样本即可奏效。

无论如何，每次初始调用 RECOLOR$^{\log}$ 的耗时为 $O(\beta^{\ell(x)}) = O(\Delta)$。要证明 RECOLOR$^{\log}$的期望摊还时间为 $O(1)$，必须证明在给定 χ^* 及 x的新颜色时，调用它的概率足够低。根据引理 3.3 和 $\mathcal{D}_{N(x)}$的定义， $\mathcal{D}_{N(x)}$的 $\Omega(\Delta)$ 个元素中至多有一个与 x的某个下邻居 $w \in \mathcal{L}(x)$ 冲突；因此， $c \sim \mathcal{D}_{N(x)}$ 发生冲突的概率为 $\Pr[\kappa_{\chi^*,c,w}] \in \underline{O}(\frac{1}{\Delta})$，所以 RECOLOR$^{\log}$ 的期望摊还运行时间 $T_{\text{RECOLOR}^{\log}}$为

$$\mathbb{E}[T_{\text{RECOLOR}^{\log}}] = \frac{\Pr[\kappa_{\chi^*,c,w}]}{\Delta} \in O\left(\frac{\Delta}{\Delta}\right) = O(1).$$

因此， $\mathcal{A}_{\log}$ 的期望摊还总更新时间为 $O(\log \Delta)$。

3.2 常数时间动态 $(\Delta+1)$-着色

为了将动态 $(\Delta+1)$-着色的性能从 $O(\log \Delta)$ 提升至 $O(1)$ 更新时间，[BGKLS22] 在 [BCHN18] 的基础上，将 RECOLOR$_{\text{constant}}$ 分为两种情况。第一种情况对应于 RECOLOR$_{\text{constant}}$ 来自 [BCHN18]；在另一种情况下，当重新着色调色板足够小时，使用运行时间为 $O(n)$的确定性重新着色子程序。

对于该算法 $\mathcal{A}_{\text{constant}}$，层级数据结构 ℓ 在本质上与 $\mathcal{A}_{\log}$中的相同。它包含 $\{-1, \ldots, \lceil \log_3(n) \rceil\}$ 个层级，并配有与 $\mathcal{A}_{\log}$中类似的数据。请注意，在 $\mathcal{A}_{\text{constant}}$和 $\mathcal{A}_{\log}$ 中，都有许多论文中详细阐述的实现细节。在本报告中，我将重点放在关键的算法和形式细节上，这些细节使这些方法区别于其他方法，也使它们彼此不同。$\mathcal{A}_{\text{constant}}$ 中的分析及其所用工具比 $\mathcal{A}_{\log}$中的复杂得多，但关键不变量相似。

不变量 3.4. 如果在重新着色期间， $v \in V$ 被移动到层级 $\ell(v) \neq -1$，那么此次重新着色使用的调色板大小至多为 $\frac{3^{\ell(v)}+1}{2}$。如果 $\ell(v) = 1$，则调色板大小为 *1*。

$\mathcal{A}_{\text{constant}}$的更新时间为 $O(1 + n^{\frac{\log n + \Delta}{t}})$，且以高概率成立。对于 $t \in \Omega(n(\log n + \Delta))$，这是 $O(1)$。因子 Δn 来源于初始化构成层级数据结构的相关数据结构，特别是对于每个 $v \in V$，都有 $O(\Delta)$ 个 $\mathcal{D}_{N(v)}$的元素。因子 $n \log n$ 来源于以下顶点的总更新时间：这些顶点所在的层级中，至少有固定比例的顶点会被重新着色，因为它们曾在某个时刻成为冲突的插入边中最近一次被重新着色的端点。层级数据结构在图 2 中重新给出，并加入了若干新定义，其中包括一些分析产物，对此我不作解释；带颜色的术语定义于 3.2.2。除了层级编号不同于 $\mathcal{A}_{\log}$之外，证明常数级更新

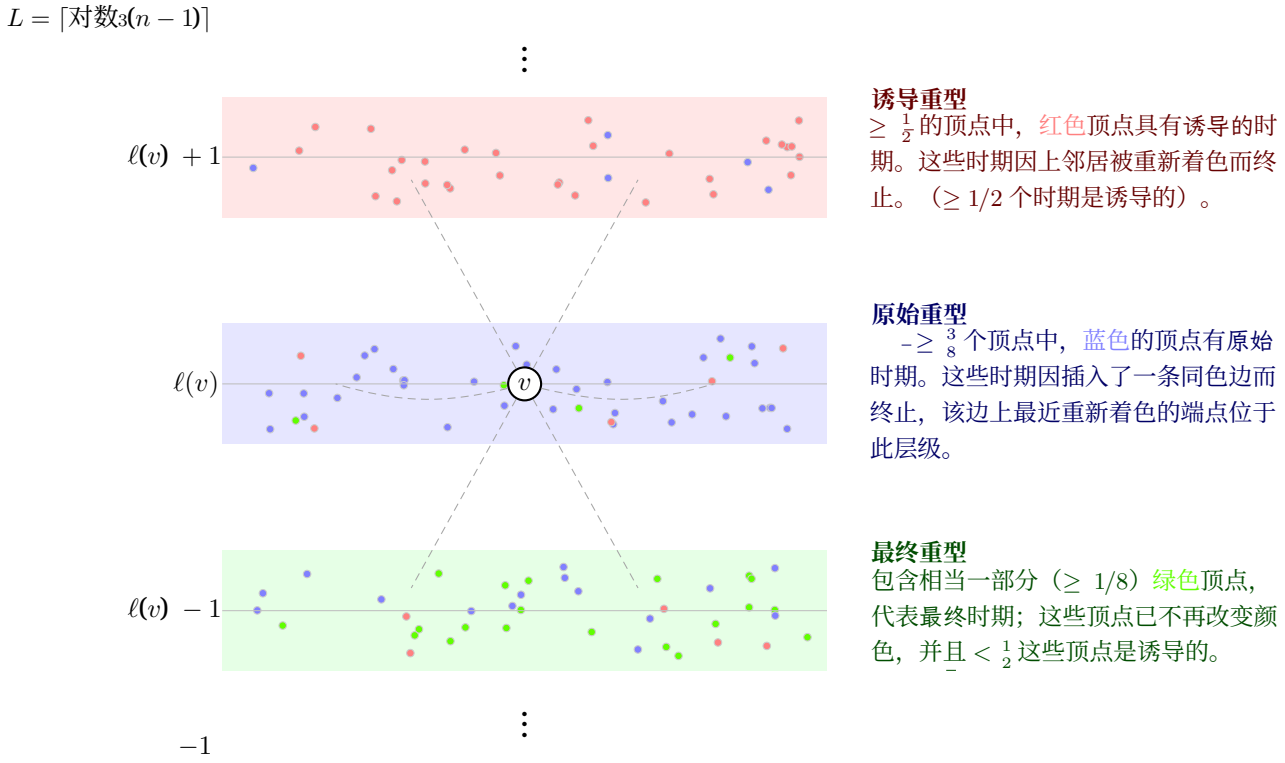

图 2：根据时期类型对层级进行分类。小型彩色节点表示各个时期，其颜色表示终止原因。顶点 v 位于层级 ℓ（v），弱虚线表示其在不同层级中的邻域。

以高概率计算 $\mathcal{A}_{\text{constant}}$ 的时间，需要跟踪哪些顶点在 RECOLOR$_{\text{constant}}$ 的原始根调用中被重新着色——即插入边的端点（该边与 χ^* 冲突）——或在 RECOLOR$_{\text{constant}}$ 的诱导后代调用中被重新着色。

3.2.1 算法

令 $\phi(v, \ell^*) = |\{u \in N(v) \mid \ell(u) < \ell^*\}|$ 表示对于每个 $v \in V$，v 在层级 ℓ^* 以下的邻居数量。在冲突边这一非平凡情形下，边插入$_{\text{constant}}$ 与边插入$_{\log}$几乎相同，只是省略了移动那些会破坏与层级间 $\mathcal{A}_{\log}$相关不变量的顶点这一步，并且当 ϕ 过大时会重试。

算法4： 边插入$_{\text{constant}}$

输入： G；用于删除或插入的 $\{u, v\} \leftrightsquigarrow E(G)$；$\ell$ 和相关数据结构
输出： 更新后的 $(\Delta+1)$ 着色 $\chi*$
如果 $\{u, v\}$ 正在插入，且 $\chi^*(u) = \chi^*(v)$ **则**
  $x \leftarrow$ 参数最大值$_{x \in \{u,v\}} \tau_w$; // More-recently recolored
  重新着色$_{\text{constant}}(x)$;

算法5： 重新着色$_{\text{constant}}$

输入： G； $v \in V(G)$ 以进行重新着色； χ^*； ℓ 和相关数据结构
输出： χ^*，其颜色为 v的新颜色

如果 $\phi\left(x,\ \ell\left(x\right)\right) < 3^{\ell(x)+2}$ **则**
    确定性重新着色(x);
    返回null;
else
    返回随机重新着色(x);

算法6： 确定性重新着色

输入： G；用于重新着色的 $v \in V(G)$； χ^*； ℓ 和相关数据结构
输出： χ^*，其颜色为 v的新颜色

对于 $c \in \mathcal{D}_{N(v)}$ **执行**
    如果不存在颜色为 $\chi^*(w) = c$ 的顶点 $w \in \mathcal{L}(v)$ **则**
        设置 $\chi^*(v) \leftarrow c$;
        更新 v的下邻居数据结构;
        设置 $\ell(v) = -1$;
        返回;

算法 7： 随机重新着色

输入： G；为重新着色准备的 $v \in V(G)$； χ^*； ℓ 和相关数据结构
输出： χ^* 使用 v的新颜色

设置 $\ell' \leftarrow \ell(v)$;
当 $\phi(v,\ \ell'+1) \geq 3^{\ell'+2}$ **时**
    递增 ℓ';
设置 $\ell(v) \leftarrow \ell'$;
均匀随机绘制 $c \sim \mathcal{D}_{N(v)}$；
如果 $c \neq \chi^*(v)$ **则**
    将 $\chi^*(v)$ 设为 $\leftarrow c$;
    更新 v的下邻居数据结构;
如果 $c \in \mathcal{U}_{\mathcal{L}(v)} \setminus \mathcal{C}_{\mathcal{H}(v)}$ **则**
    令 $w \in \mathcal{L}(v)$满足 $\chi^*(w) = c$;
    重新着色$_{\text{constant}}(w)$;

3.2.2 分析

对 $\mathcal{A}_{\text{constant}}$ 进行分析的核心是时期 $\mathcal{E}$这一概念，它刻画了某些顶点被重新着色的低频程度。具体而言，时期 $\mathcal{E}$ 对应于顶点 $x = v(\mathcal{E})$，在此期间， x 保持其颜色 $\chi^*(\mathcal{E})$和层级 $\ell(v)$不变。时期的成本 $c(\mathcal{E})$是调用 RECOLOR$_{\text{constant}}$所需的时间；该调用作用于 $v(\mathcal{E})$，并在 $\mathcal{E}$ 开始改变 $\chi^*(v(\mathcal{E}))$和 $\ell(v(\mathcal{E}))$时启动。当 $\ell(x)$为 $x = v(\mathcal{E})$执行 $= -1$，因此 x 完成重新着色后，分析便记录 $c(\mathcal{E})$。换言之，每个顶点对 $\mathcal{A}_{\text{constant}}$ 总更新时间的贡献，来自其各个时期成本之和。一个时期由以下调用界定：调用时标记为原始调用的 RECOLOR$_{\text{constant}}$，或调用时标记为诱导调用的 RECOLOR$_{\text{constant}}$。参见图3。

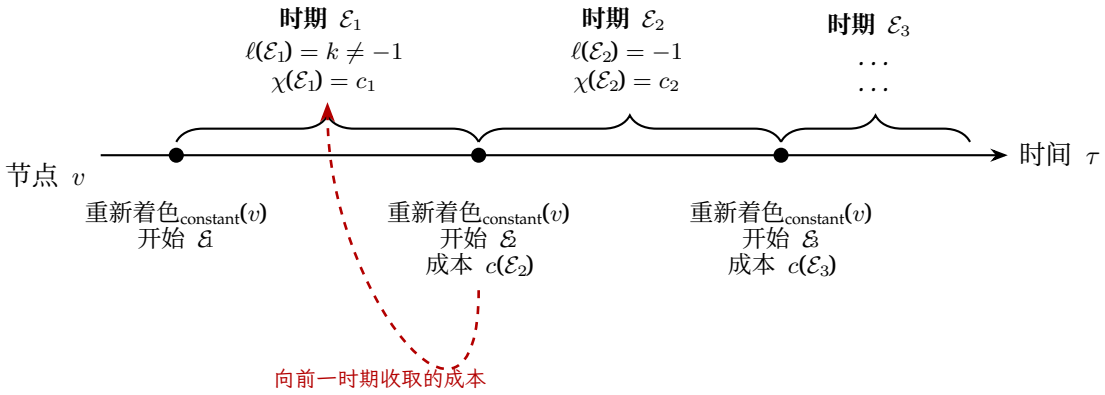

图3：顶点 $x = v(\mathcal{E}_1) = v(\mathcal{E}_2) = v(\mathcal{E}_3)$ 的时期 $\mathcal{E}_1$、$\mathcal{E}_2$、$\mathcal{E}_3$。 $\mathcal{E}$对应于 x 两次连续重新着色之间的最大时间间隔。层级为 $\ell(\mathcal{E}) = -1$ ($\mathcal{E}_2$) 的时期成本计入前一时期（$\mathcal{E}_1$）。此处 $\mathcal{E}_3$ 可能是诱导的。

由于在 $\mathcal{E}$ 期间没有顶点 $v(\mathcal{E})$ 改变层级，因此，时期集合 $\mathscr{E}$ 可以跨越ℓ进行划分。 ℓ 中的瓶颈层级集合是原始重型层级；在这些层级中，拥有诱导时期的顶点少于一半，拥有最终时期的顶点少于八分之一；也就是说，原始重层级中至少有3 8 的顶点对 RECOLOR$_{\text{constant}}$ 进行了原始调用。用于成本的记号 c 扩展为一个层级中各时期成本之和。

因此，分析的主要任务是界定原始重层级的成本，而这并非易事，因为原始重层级的成本取决于其为原始重型这一事实。这意味着，不变量 3.4 无法像不变量 3.1 和 3.2 那样，在插入冲突边之前对插入次数的期望值给出下界；后两者确保了在 $\mathcal{A}_{\log}$中每次边插入发生冲突的概率足够小。

为克服分析中的这一障碍，我们定义了两个时间函数。首先，时期 $\mathcal{E}$ 的持续时间$dur(\mathcal{E})$是 $\mathcal{E}$ 期间边插入的数量。其次， $\mathcal{E}$ 的伪持续时间 $psdur(\mathcal{E})$是指在 $\mathcal{E}$ 期间， $v(\mathcal{E})$ 的邻居所取的不同颜色数量，直到其中一个邻居取到 $v(\mathcal{E})$ 在 $\mathcal{E}$ 选择的颜色 ℓ。例如，自适应对手会确保，与 $v(\mathcal{E})$ 相连的每次插入（$v(\mathcal{E})$ u）中，另一端点 u 的颜色都与 $v(\mathcal{E})$ 在 $\mathcal{E}$ 开始时所取的颜色相同。参见图 4。

根据定义， $psdur(\mathcal{E}) \leq dur(\mathcal{E})$，并且事实证明， $\mathcal{E}$ 较短的概率上界较低（[BGKLS22] 引理 3.11）。此外，某一层级包含过多时期、且其中短时期数量超过少数几个的概率上界也较低。事实上，粗略地说，对于较大的常数 a，任何层级具有这种不良特征的概率为$O(\frac{\log n}{n^a})$。这导致总运行时间中出现 log n因子，而足够多的边插入会将其摊还掉。若没有这一关于伪持续时间过短的时期出现概率的上界， $\mathcal{A}_{\log}$ 的确定性运行时间将为 $O(tn^2 + \Delta n)$；在这种情况下，即使 $t \in \Omega(n$ (log $n + \Delta$))， $\mathcal{A}_{\log}$的更新时间也至少会多出两个 n因子。

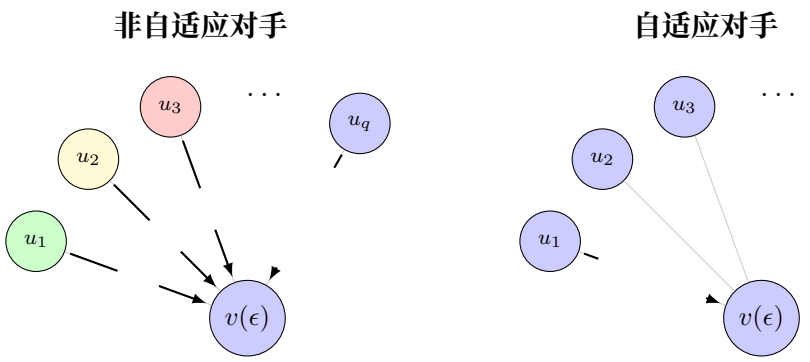

图 4：伪持续时间示例：首先是 $\mathcal{A}_{\log}$ 中的非自适应情形，然后展示自适应对手如何利用伪持续时间，试图缩短轮次长度。

4 针对自适应对手的动态 $(\Delta+1)$ 着色

$\mathcal{A}_1$、 $\mathcal{A}_2$、 $\mathcal{A}_{\log}$和 $\mathcal{A}_{\text{constant}}$ 假设输入是无自适应的。也就是说，它们假设插入的边是从 G的非边集合 $\binom{V(G)}{2} \setminus E(G)$ 中均匀随机抽取的。非自适应性与多种应用范式相符：在这些范式中，输入来自复杂的现实世界系统，且其分布难以处理。 [BRW25] 研究了 $(\Delta+1)$-着色，使其能够稳健地应对自适应对抗性输入；这类输入会破坏输入的均匀性假设，并在最坏情况下导致最差性能。自适应对手 $\forall$ 可以暗中检查或模拟 $(\Delta+1)$-着色器 $\mathcal{A}$，确定 $\mathcal{A}$在每种排列 $\mathcal{U} \in \mathfrak{S}_{\mathscr{U}}$ 下对所需输入 $\mathscr{U}$的性能，并将使 $\mathcal{A}$的摊还更新时间达到最大值的 $\mathcal{U}$ 输入给 $\mathcal{A}$。具体而言，[BRW25] 提出了一种动态 $(\Delta+1)$-着色器 $\mathcal{A}_\forall$，面对这种自适应对手时，其更新时间为 $\tilde{O}(n^{8/9})$。为了在面对 $\forall$时实现次线性时间，$\mathcal{A}_\forall$ 采用基于阶段的方法，分别处理所谓的稀疏顶点和稠密顶点。稀疏顶点会按照确定性间隔随机重新着色，因此其调色板通常有足够多的剩余颜色；稠密顶点则会反复重新着色，直到正确着色为止，而期望的重复次数有界。

在之前的（$\Delta+1$）着色器中，预先知道 Δ 后，算法便能确定桶的数量或分层顶点划分的高度。在这种情况下，使用全部 O（Δ）种可用颜色是一项显式不变量。

> ˜**不变量 4.1**。 每种颜色 $c \in \{1,\dots,\Delta+1\}$ 都会分配给 $O\binom{n}{\Delta}$ 中的一定数量的顶点。

˜借助不变量 4.1，验证某种颜色是否适合分配给某个顶点所需的时间为 $O(\binom{n}{\Delta})$，而不是 $\Theta(\Delta)$。$\mathcal{A}_\forall$ 采用稀疏–稠密顶点分解，以便更容易跟踪每个顶点可用的颜色。该分解大致是 Harris–Schneider–Su [HSS16](HSS) 分解，它将 $V(G)$ 划分为稀疏顶点 V_S 和稠密顶点 V_D。令G_S：$= G[V_S]$，以及 $G_D := G[V_D]$。 V_S 是这样一组顶点：其中每个 v 顶点都具有 $\left|E(G) \cap N(v)^2\right| \le (1-\epsilon^2)\binom{\Delta}{2}$；也就是说，每个稀疏顶点的邻域距离成为一个 Δ团还相差 ϵ^2。它进一步将 $G[V_D]$ 划分为 C_1、.....、C_k:

$$V(G) = V_S \oplus \bigoplus_{i=1}^{k} V(C_i).$$

每个 $C_i \in \{C_1,\dots,C_k\}$ 都是一个近似团，因此对于某个 $\epsilon > 0$

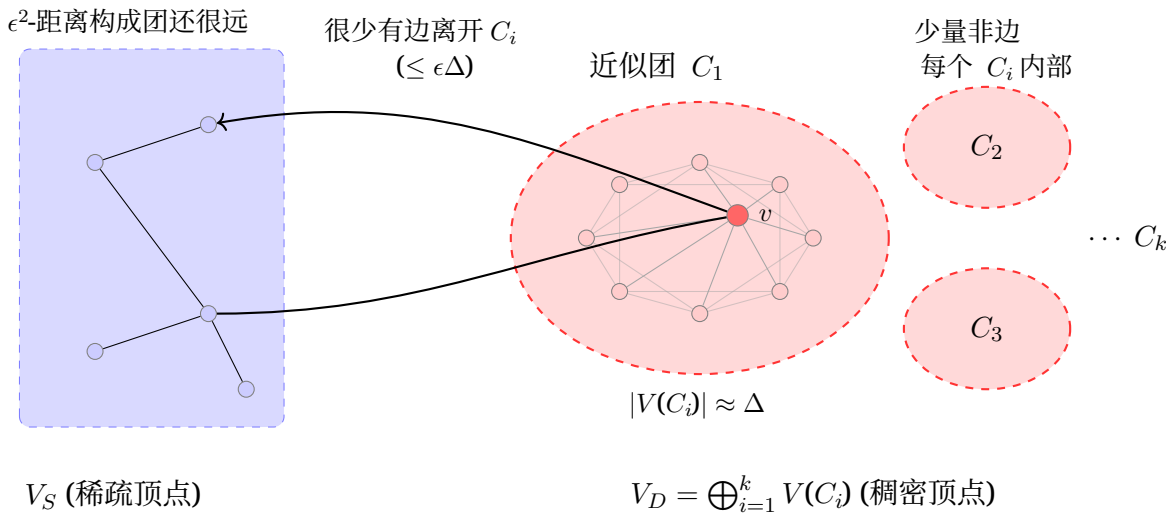

图 5：HSS 分解方案。 $V(G)$ 被划分为稀疏顶点 V_S 和稠密顶点 V_D。 V_D 是近似团 C_1、C_2、.....、C_k 的不交并。顶点 $v \in C_1$ 展示了这样一种性质：它与自身所在团中的几乎所有顶点相连，且只有很少的边离开它。

- $|V(C_i)| \in [(1-\epsilon)\Delta, (1+\epsilon)\Delta]$ (C_i has order close to Δ)

并且对于任意 $v \in C_i$

- $|V(C_i)| - |N(v)| \leq \epsilon\Delta$ （v 与 C_i 中的几乎所有顶点相邻）并且
- $|N(v) \cap V(G) \setminus V(C_i)| \leq \epsilon\Delta$ （很少有边离开 C_i）。

见图 5。（关于近似团还有一些与调整复杂度有关的性质（参见 [BRW25] 定理 2）。）

分解很有用，因为 $\mathcal{A}_\forall$ 独立维护 G_S 和 C_i 的着色。对于 G_S，$\mathcal{A}_\forall$ 运行单轮刷新着色器，以确保每个稀疏顶点的剩余颜色期望为 $\Omega(\epsilon^2\Delta)$。具体而言，单轮稀疏着色为每个顶点均匀随机地抽取一个颜色，然后分配互不冲突的颜色。参见算法 8。通常，这一过程能成功地为 G_S 中的常数比例重新着色，并为典型顶点的许多邻居赋予相同颜色，从而为每个顶点的调色板留下 $\Omega(\epsilon^2\Delta)$ 个剩余颜色。为确保这些颜色可用，每经过 $\Theta(\epsilon^2\Delta)$ 次更新后就运行单轮稀疏重着色；每次更新至多为一个顶点重新着色，因此顶点的剩余颜色最多减少一个，使得剩余颜色仍为 $\Omega(\epsilon^2\Delta) - \Theta(\epsilon^2\Delta) = \Omega(\epsilon^2\Delta)$。换言之，对于稀疏顶点，每 $\Theta(\epsilon^2\Delta)$ 个更新构成一批更新，并由单轮稀疏重着色划分。接着，为稀疏顶点重新着色时，均匀随机选中这些剩余颜色之一的概率为 $\Omega(\epsilon^2\Delta)/O(\Delta+1) = \Omega(\epsilon^2)$。由此，进行 $O(\epsilon^{-2})$ 次这样的抽样，并且每次都进行一次 $\tilde{O}(\frac{n}{\Delta})$ 可行性检查（这得益于不变量 4.1），即可得到

一个合适的颜色，使稀疏顶点的重新着色时间为 $\tilde{O}(\frac{n}{\epsilon^2\Delta}) = n^{1-\Omega(1)}$ ——次线性。[3]

算法 8： 单轮稀疏着色

输入： 图 G；稀疏顶点 $V_S \subset V(G)$

输出： 重新着色的顶点 $V' \subseteq V_S$

初始化 $V' \leftarrow \emptyset$;

对于 $v \in V_S$，**执行**

从均匀随机分布中采样 $\chi'(v) \sim [\Delta + 1]$;

对 $v \in V_S$ **执行**

如果 $c(v) \neq \chi^*(w)$ **对每个** $w \in N(v)$ **则**

将 $\chi^*(v)$ 赋给 $\leftarrow c(v)$;

将 v 添加到 V';

返回 V'_S;

对于 V_D，动态着色的处理更为复杂。关键技术是将 [ACK19] 中的静态算法动态化，其中每个近似团 C_i 按如下方式着色。首先，找到非边 $\binom{C_i}{2} \setminus E(C_i)$的近似最大匹配，并为匹配中的顶点赋予相同颜色。换言之，尽可能多地为非邻接顶点着色，每种颜色用于两个顶点。其次，在剩余顶点与剩余颜色之间找到一个完美匹配。该匹配定义在一个二分图 $\mathcal{H}$ 上，其两侧分别为 $\mathcal{V}$ （剩余顶点）和 $\mathcal{C}$ （剩余颜色）；当且仅当 $c \not\in N(v) \setminus \mathcal{V}$时， $\{v, c\} \in E(\mathcal{H})$。在 $\mathcal{H}$ 上找到完美匹配，可将每个顶点匹配到一个未使用的颜色，从而得到正常着色。匹配的存在性并不显然；而要将这一步动态化，又不花费与 Δ 成线性关系的时间（因此时间可能与 n成线性关系，这与 $\forall$相悖），极具挑战性。

要使这个静态算法动态化，有两种情况。第一种情况，对于阶至少为 $\Delta + 1$ 的近似团，颜色被划分为重颜色和轻颜色。如果在近似团 C_i 中，存在 $\Omega(\Delta)$ 条边 (a_i, b_i)，满足 $a_i \in V(C_i)$ $b_i \not\in V(C_i)$ $\chi^*(a_i) = c$，那么颜色 c 就是重颜色。轻颜色不是重颜色。重颜色不如轻颜色有用。幸运的是，即使从 $\mathcal{C}$ 中排除重颜色， $\mathcal{H}$ 仍然可以着色，因为存在一条以较高常数概率出现的增广路径。第二种情况，对于大小小于 $\Delta + 1$ 的近似团，又分为两个子情况。如果非边匹配足够大（至少为 $\frac{\Delta}{10}$），那么 $\mathcal{A}_\forall$ 的行为与大型近似团的情况类似，因为足够多的顶点已经得到处理。否则，为了重新着色 $v \in V(C_i)$，会从 C_i 中均匀随机地选取一个尚未使用的随机颜色 c。如果 v 有稀疏邻居， c 可能会引发冲突，其概率为 $\Omega(\frac{1}{k})$，其中 $k = \Delta + 1 - |V(C_i)|$。于是，经过 $O(\epsilon\Delta)$ 次这样的采样，$\mathcal{A}_\forall$ 通常会找到一个可用颜色。如果这行不通，那么一条经过 $\mathcal{H}$ 、长度为五的增广路径，使用三个顶点 v_1、 v_2、 $v_3 \in \mathcal{V}$ 以及 $\mathcal{C}$ 中的三种颜色，就能以足够大的常数概率正确地为 v_1、 v_2、 v_3 着色。寻找这条增广路径最耗费运行时间： $O(k)$ 次采样，每次都要进行耗时 $\frac{n}{\Delta}$ 的可行性检查，总共耗时，属于 $O(\epsilon n)$ ——这是次线性时间。

5 致谢

感谢各位作者对该领域的贡献，并感谢刘全全提出的意见。也感谢 OpenAI 和 Google 的生成式预训练 Transformer 大语言模型（Large LaTeX Mud-

[3]每个批次开始时，都会运行单轮稀疏重新着色，这是一种来自 [AY25] 的额外贪心算法，它会对 V_S 中尚未成功重新着色的恒定比例部分进行着色。

投石手），这有助于制作图形。

参考文献

[ACK19] Sepehr Assadi、Yu Chen 和 Sanjeev Khanna。“(Δ+ 1) 顶点着色的次线性算法”。载于：第三十届 ACM-SIAM 离散算法年会论文集。SIAM。2019 年，第 767–786 页（见第 14 页）。[AY25] Sepehr Assadi 和 Helia Yazdanyar。“通过非对称调色板稀疏化实现 (Δ+ 1) 顶点着色的简单次线性算法”。载于：2025 年算法简约性研讨会（SOSA）。SIAM。2025 年，第 1–8 页（见第 14 页）。[BCKLVRV17] Luis Barba、Jean Cardinal、Matias Korman、Stefan Langerman、André Van Renssen、Marcel Roeloffzen 和 Sander Verdonschot。“动态图着色”。载于：算法与数据结构研讨会。施普林格。2017 年，第 97–108 页（见第 1、3 页）。[BRW25] Soheil Behnezhad、Rajmohan Rajaraman 和 Omer Wasim。“针对自适应对手的全动态 (Δ +1)-着色”。载于：2025 年 ACM-SIAM 离散算法年会论文集（SODA）。SIAM。2025 年，第 4983–5026 页（见第 1、12、13 页）。[BCHN18] Sayan Bhattacharya、Deeparnab Chakrabarty、Monika Henzinger 和 Danupon Nanongkai。“图着色的动态算法”。载于：第二十九届 ACM-SIAM 离散算法年会论文集。SIAM。2018 年，第 1–20 页（见第 1、4、7、8 页）。[BGKLS22] Sayan Bhattacharya、Fabrizio Grandoni、Janardhan Kulkarni、Quanquan C Liu 和 Shay Solomon。“O(1) 更新时间的全动态 (Δ+ 1)-着色”。载于：ACM 算法汇刊（TALG）18.2（2022），第 1–25 页（见第 1、4、8、11 页）。[HSS16] David G Harris、Johannes Schneider 和 Hsin-Hao Su。“次对数轮数的分布式 (Δ+ 1)-着色”。载于：第四十八届 ACM 计算理论研讨会年会论文集。2016 年，第 465–478 页（见第 12 页）。