

Online Graph Coloring for k -Colorable Graphs

Ken-ichi Kawarabayashi*

Hirotaka Yoneda[†]

Masataka Yoneda[‡]

Abstract

We study the problem of online graph coloring for k -colorable graphs. The best previously known deterministic algorithm uses $\tilde{O}(n^{1-1/k!})$ colors for general k and $\tilde{O}(n^{5/6})$ colors for $k = 4$, both given by Kierstead in 1998. In this paper, nearly thirty years later, we have finally made progress. Our results are summarized as follows:

1. **$k \geq 5$ case.** We provide a deterministic online algorithm to color k -colorable graphs with $\tilde{O}(n^{1-2/(k(k-1))})$ colors, significantly improving the current upper bound of $\tilde{O}(n^{1-1/k!})$ colors. Our algorithm also matches the best-known bound for $k = 4$ ($\tilde{O}(n^{5/6})$ colors).
2. **$k = 4$ case.** We provide a deterministic online algorithm to color 4-colorable graphs with $\tilde{O}(n^{14/17})$ colors, improving the current upper bound of $\tilde{O}(n^{5/6})$ colors.
3. **$k = 2$ case.** We show that for randomized algorithms, the upper bound is $1.034 \log_2 n + O(1)$ colors and the lower bound is $\frac{91}{96} \log_2 n - O(1)$ colors. This means that we close the gap to $1.09x$.

With our algorithm for the $k \geq 5$ case, we also obtain a deterministic online algorithm for graph coloring that achieves a competitive ratio of $O(n/\log \log n)$, which improves the best known result of $O(n \log \log \log n/\log \log n)$ by Kierstead.

For the bipartite graph case ($k = 2$), the limit of online deterministic algorithms is known: any deterministic algorithm requires $2 \log_2 n - O(1)$ colors. Our results imply that randomized algorithms can perform slightly better but still have a limit.

*National Institute of Informatics & The University of Tokyo, Tokyo, Japan, k.keniti@nii.ac.jp.
Supported by JSPS Kakenhi 22H05001, and by JST ASPIRE JPMJAP2302.

[†]The University of Tokyo, Tokyo, Japan, square37@gmail.com.
Supported by JST ASPIRE JPMJAP2302 and by JST ACT-X JPMJAX25CT.
[‡]The University of Tokyo, Tokyo, Japan, e869120@gmail.com.
Supported by JST ASPIRE JPMJAP2302.

Contents

1	Introduction	1
1.1	Our contributions	2
1.2	Overview of our techniques	3
1.2.1	$k \geq 5$ case, Theorem 1.1	3
1.2.2	$k = 4$ case, Theorem 1.2	4
1.2.3	$k = 2$ case upper bound, Theorem 1.3	5
1.2.4	$k = 2$ case lower bound, Theorem 1.4	5
1.3	Organization of this paper	5
2	Preliminaries	6
3	General bound	6
3.1	Algorithm for locally ℓ -colorable graph	7
3.2	Reduction to the subproblem	7
3.3	Base case for the subproblem ($d = \ell - 1$)	9
3.4	Inductive step on level d : Algorithm	10
3.5	Inductive step on level d : Analysis	12
3.6	The improvement	14
3.7	Comparison with Kierstead's algorithm	15
3.8	The competitive ratio	15
4	Improvement for 4-colorable graphs	17
4.1	The idea of the algorithm	17
4.2	Preliminaries	18
4.3	The first step: No-dense case	21
4.4	Correctness & Analysis	22
4.5	The second step: Avoiding dense case	24
4.6	The Common & Simplify technique	25
4.7	Correctness & Analysis	27
4.8	Conclusion	28
5	Randomized algorithm for bipartite graphs	29
5.1	The algorithm by Lovász, Saks, and Trotter	29
5.2	The randomized algorithm and its probabilistic model	30
5.3	Preliminaries for the analysis	31
5.4	Analysis 1: Considering the level-2 terminals	33
5.5	Analysis 2: Increasing the layers	35
6	Lower bound of the expected performance for bipartite graphs	38
6.1	The lower bound instance	38
6.2	Introducing potential	39
6.3	Two-phase analysis	41
6.4	Further improvement: Limiting the number of combinations	41
6.5	The final piece: Potential decomposition	44
7	Conclusion	45

1 Introduction

Graph coloring is arguably the most popular topic in graph theory. It is also one of the most well-studied topics in algorithms, because it is known to be one of the hardest problems to approximate. For general graphs, it is inapproximable within a factor $n^{1-\epsilon}$ for any $\epsilon > 0$, unless $ZPP = NP$ [10]. The best known approximation algorithm computes a coloring within a factor of $O(n(\log \log n)^2 / (\log n)^3)$ [13]. Even for 3-colorable graphs, the best approximation algorithm achieves a factor of $\tilde{O}(n^{0.1975})$ [17].

Formally, the *graph coloring problem* asks to color each vertex of a graph $G = (V, E)$ so that no two adjacent vertices have the same color. The number of colors should be as small as possible. Let $\chi(G)$ be the chromatic number of G , i.e., the number of colors needed for an optimal coloring.

In this paper, our primary focus is on the online version of graph coloring.

Online graph coloring. The *online coloring problem* is defined differently. In the online version of the graph coloring problem, the vertices of an unknown graph arrive one by one, together with edges adjacent to the already present vertices. Upon the arrival of each vertex v , a color must be immediately assigned to v before the next vertex arrives. The goal is to obtain a proper coloring of the resulting graph. The challenge is to design a coloring strategy that minimizes the total number of assigned colors. The online graph coloring problem appears to be even harder than the graph coloring problem.

The online graph coloring problem is indeed challenging because, sometimes, intuitive algorithms fail; for example, the FIRSTFIT algorithm, which repeatedly assigns the least-indexed available color to each arriving vertex, cannot even bound the number of colors (see Figure 1). This algorithm works well only for d -degenerate graphs (i.e., every induced subgraph has a vertex of degree at most $d - 1$), see [16].¹

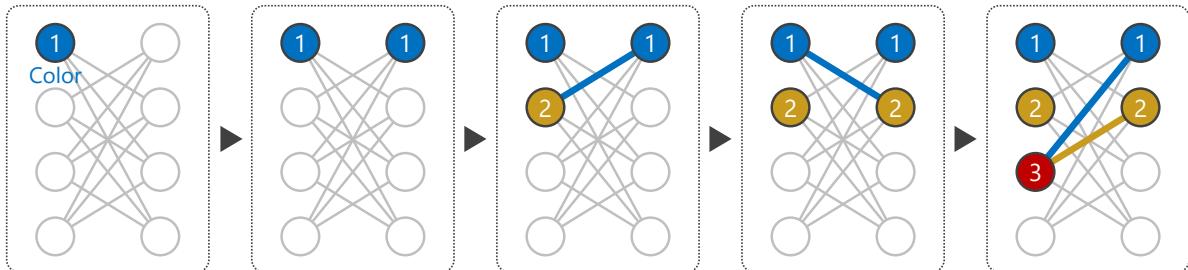


Figure 1: A worst-case input for FIRSTFIT, which uses $\frac{1}{2}n$ colors even for bipartite graphs [16].

Previous studies on online coloring. Lovász, Saks, and Trotter [21] in 1989 give a deterministic online coloring algorithm with a competitive ratio² of $O(n/\log^* n)$. Kierstead [18] in 1998 improves the competitive ratio (though not explicitly stated) to $O(n \log \log n / \log \log n)$. Vishwanathan in 1992 [23] gives a randomized algorithm that attains a competitive ratio of $O(n/\sqrt{\log n})$. His randomized algorithm is modified in [14] to improve the competitive ratio to $O(n/\log n)$. However, there is a $2n/\log^2 n$ lower bound (of the competitive ratio) for deterministic online graph coloring, and this bound also holds up to a constant factor for randomized algorithms [15]. Even for a tree, the lower bound is $\log n$, see [4, 12].

Given very strong negative results for approximating the chromatic number in the online graph coloring problem (even worse than the graph coloring case), it really makes sense to consider the online graph coloring for k -colorable graphs. For $k \geq 3$, a randomized algorithm that uses $\tilde{O}(n^{1-1/(k-1)})$ colors is found

¹FIRSTFIT only uses $O(d \log n)$ colors for such a graph.

²The competitive ratio is the maximum value of (the algorithm's solution [i.e., the number of colors used]) divided by (the optimal solution [i.e., chromatic number]) for all possible inputs.

by Vishwanathan [23], and a deterministic algorithm that uses $O(n^{1-1/k!})$ colors is given by Kierstead in 1998 [18]. For special cases when $k = 3$ and $k = 4$, deterministic algorithms that use $\tilde{O}(n^{2/3})$ colors and $\tilde{O}(n^{5/6})$ colors are given in 1998 by Kierstead [18]. However, no improvement has been made for deterministic algorithms since 1998, and a significant gap remains between the upper and lower bounds. Surprisingly, the only known lower bound is $\Omega(\log^{k-1} n)$ colors [23].

For $k = 2$, a deterministic algorithm that uses $2 \log_2(n + 1)$ colors is found by Lovász, Saks, and Trotter in 1989 [21]. For the lower bound of deterministic algorithms, after a series of research [4, 5, 11], Gutowski et al. in 2014 [11] finally showed that achieving $2 \log_2 n - 10$ colors is impossible. However, the performance of randomized algorithms remains open, where Vishwanathan [23] gives a lower bound of $\frac{1}{72} \log_2 n$ colors.

Deterministic and randomized online algorithms to color the following graph classes achieve $\Theta(\log n)$ colors: trees, planar, bounded-treewidth, and disk graphs [1]. For more references for online graph colorings, we refer the reader to the survey by Kierstead [19].

Deterministic vs Randomized. Generally speaking, deterministic online algorithms have large gaps compared to randomized online algorithms for many problems. For example, deterministic online caching algorithms cannot provide better worst-case guarantees than known trivial algorithms; however, randomization may yield significantly better results (see [22], Chapter 24). This is also the case for the online edge-coloring problem, see recent progress in [6, 7, 8]. We also predict that this is also the case for online coloring problems. Note that our deterministic online algorithms below do not match the randomized cases.

The difficulty for deterministic online algorithms in coloring problems lies in the so-called "adversarial attack" from the context of "machine learning" [3]. This means that whenever we assign a color to an arrived vertex, the next vertex to arrive puts us in the worst possible situation. This is very typical in the adversarial bandit problem [2], which assumes no stochastic assumptions about the process generating rewards for actions. This means that, in the deterministic case, we must always consider the worst-case scenario. In contrast, in the randomized case, it occurs only with very small probability, allowing us to present a randomized online algorithm with fewer colors.

1.1 Our contributions

In this paper, we study online coloring algorithms for k -colorable graphs. Our goal is to close the gap between the upper and lower bounds. So far, the best-known deterministic algorithm, proposed by Kierstead in 1998 [18], uses $O(n^{1-1/k!})$ colors. Nearly thirty years later, this paper finally makes progress. Our main contributions improve the results for all $k \geq 2$, except $k = 3$.

Theorem 1.1. *There exists a deterministic online algorithm to color k -colorable graphs with $\tilde{O}(n^{1-2/(k(k-1))})$ colors, for $k \geq 2$. This algorithm also achieves a competitive ratio of $O(n/\log \log n)$.*

This significantly improves the current upper bound of $O(n^{1-1/k!})$ colors [18] when $k \geq 5$. This also improves the competitive ratio for deterministic online algorithms, where the current best result is $O(n \log \log n / \log \log n)$ due to Kierstead [18].

Theorem 1.2. *There exists a deterministic online algorithm to color 4-colorable graphs with $\tilde{O}(n^{14/17})$ colors.*

This improves the current upper bound of $\tilde{O}(n^{5/6})$ colors [18]. Since the $\tilde{O}(n^{1-2/(k(k-1))})$ -color algorithm in **Theorem 1.1** works with $\tilde{O}(n^{5/6})$ colors when $k = 4$, **Theorem 1.2** is better. By combining **Theorem 1.1** and **Theorem 1.2**, we also improve the results for $k \geq 5$ (to $\tilde{O}(n^{1-6/(3k(k-1)-2)})$ colors; see **Theorem 4.17**). Our results, compared with the previous best, are shown in **Table 1**.

For $k = 2$, the lower bound and the upper bound exactly match at $2 \log_2 n \pm O(1)$ colors for deterministic algorithms [21, 11]. However, for randomized algorithms, we can do a little better, as follows:

k	3	4	5	6	7
Previous Results	$\tilde{O}(n^{0.6667})$	$\tilde{O}(n^{0.8334})$	$\tilde{O}(n^{0.9917})$	$\tilde{O}(n^{0.9987})$	$\tilde{O}(n^{0.9999})$
Our Results	—	$\tilde{O}(n^{0.8236})$	$\tilde{O}(n^{0.8966})$	$\tilde{O}(n^{0.9319})$	$\tilde{O}(n^{0.9517})$

Table 1: Comparison between the previous best results [18] and our results, for $k \leq 7$.

Theorem 1.3. *There exists a randomized online algorithm to color 2-colorable graphs with $1.034 \log_2 n + O(1)$ colors.*

Theorem 1.4. *Any randomized online algorithm to color 2-colorable graphs requires $\frac{91}{96} \log_2 n - O(1)$ colors.*

Our results imply that randomized algorithms can perform slightly better, but still have a limit. Theorem 1.3 and Theorem 1.4 imply that the gap is 1.09x.

1.2 Overview of our techniques

We now provide an overview of our techniques for the results stated above.

1.2.1 $k \geq 5$ case, Theorem 1.1

The key is to consider a “locally ℓ -colorable graph”, which is a graph without small (constant-order) non- ℓ -colorable subgraphs. Since every ℓ -colorable graph is locally ℓ -colorable but not vice versa, coloring locally ℓ -colorable graphs is more difficult than coloring ℓ -colorable graphs. For $\ell = 1$ and $\ell = 2$, there exist deterministic algorithms that use $O(1)$ colors and $O(n^{1/2})$ colors, respectively; for $\ell = 2$, an $O(n^{1/2})$ -color algorithm for graphs without odd cycles C_3 or C_5 is given by Kierstead [18]. However, no algorithm is known for $\ell \geq 3$.

Our main result is to achieve $O(n^{1-2/(\ell(\ell-1)+2)})$ colors for such graphs. As a consequence, we present a deterministic online coloring algorithm for k -colorable graphs with $\tilde{O}(n^{1-2/(k(k-1))})$ colors. Below, we explain how to derive an algorithm for locally ℓ -colorable graphs with $O(n^{1-\epsilon})$ colors, where $\epsilon = \frac{2}{\ell(\ell-1)+2}$.

First, we try the FIRSTFIT algorithm using colors $1, 2, \dots, n^{1-\epsilon}$, but for some vertices, there are no available colors. In such a case, we must pay for new colors. However, such a vertex v is adjacent to at least $n^{1-\epsilon}$ previous vertices. Specifically, let S be the set of vertices colored by FIRSTFIT, and let T be the rest of the vertices. Then, $|N_S(v)| \geq n^{1-\epsilon}$ holds for each $v \in T$ (otherwise, we can color v by FIRSTFIT). Using this large degree condition, we aim to obtain a “good subset S' ” in S that tells us about the upper bound of the chromatic number in $G[N(S')]$ (and hence the future vertices in $N(S')$). The best scenario is to consider the case when G is ℓ -colorable (not just locally ℓ -colorable), and to obtain a subset $S' \subseteq S$, in which only one color is used in S' in some ℓ -coloring of G . We call such a set S' a *1-color set*. Then the graph $G[N(S')]$ is guaranteed to be $(\ell-1)$ -colorable. In particular, all the “future” vertices that are adjacent to a vertex in S' can be $(\ell-1)$ -colored. This allows us to apply our algorithm inductively (on ℓ), and hence we can (inductively) color the vertices in $N(S')$.

To implement the above idea for locally ℓ -colorable graphs, we define a *level- d set*, which measures how close the set is to a 1-color set above. Formally, we say that a subset $S' \subseteq S$ is level- d if, for every small (constant-order) subset $X \subseteq V$, only $\ell - d$ or fewer colors are used in $X \cap S'$ in some ℓ -coloring of $G[X]$.³ The entire set S is just a level-0 set; but we aim to obtain a level-1 set, a level-2 set, and so on, and finally a level- $(\ell-1)$ set. This is because, once we obtain a level- $(\ell-1)$ set S' , as argued in the previous paragraph, $G[N(S')]$ is indeed a locally $(\ell-1)$ -colorable graph (see subsection 3.3).⁴

³Note that since G is locally ℓ -colorable, there exists some ℓ -coloring of $G[X]$.

⁴It means that we increase the level by $\Theta(\ell^2)$ in total (until reaching “locally 1-colorable”); it is why we achieve $O(n^{1-1/\Theta(\ell^2)})$.

To obtain a level-1 set, we pick a vertex $v \in T$, and then $S_1 = N_S(v)$ is indeed a level-1 set.⁵ To obtain a level-2 set, we pick two *adjacent* vertices $w_1, w_2 \in T$, and then $S_2 = N_{S_1}(w_1)$ or $S_3 = N_{S_1}(w_2)$ is indeed a level-2 set (see Figure 2).⁶ The important thing is that we cannot guarantee which of the two sets is a level-2 set, but we can guarantee that at least one of them is a level-2 set. Similarly, we can also obtain level-3 or higher sets in a similar way (see subsection 3.4). Note that the constructions of these level sets are for the sake of coloring “future” vertices. Note also that some readers may wonder that the obtained subsets (e.g., S_2 and S_3) can be empty (or very small) in the worst case, but due to the large degree condition, we are always guaranteed to obtain such subsets (see subsection 3.4 for details). In addition, some readers may also worry that many mistakes might occur (e.g., either S_2 or S_3 may not be a level-2 set), but we can bound the number of mistakes (Lemma 3.8), and therefore we do not pay many colors for mistakes.

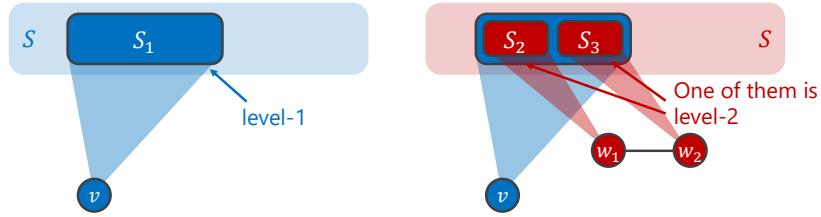


Figure 2: A sketch of our algorithm for $k \geq 5$ case. The left and right figures show the creation of level-1 and level-2 sets. Note that the entire set S is level-0, not even a level-1 set.

1.2.2 $k = 4$ case, Theorem 1.2

The key to our algorithm is the use of the second-neighborhood structure. In the algorithm used in Theorem 1.1 that achieves $\tilde{O}(n^{5/6})$ colors for $k = 4$, we obtain 1-color sets (see subsubsection 1.2.1) of $\Omega(n^{1/2})$ vertices, 2-color sets of $\Omega(n^{2/3})$ vertices, and 3-color sets of $\Omega(n^{5/6})$ vertices. If any of the larger sets (e.g., 3-color sets of $\Omega(n^{0.9})$ vertices) were obtained, there would be a good chance of improvement, but it is still challenging. We resolve this problem with our novel *double greedy method*, which applies FIRSTFIT twice to exploit the second-neighborhood structure. Let S be the vertices colored by the first FIRSTFIT, let T be the vertices colored by the second FIRSTFIT, and let U be the rest of the vertices.⁷ Both runs of FIRSTFIT use $O(n^{5/6})$ colors.⁸ The first step is to obtain a 1-color set of $\Omega(n^{1/2})$ vertices in T (using U). The second step is that, when $T' \subseteq T$ is the obtained 1-color set, $N_S(T')$ is a 3-color set of $\Omega(n)$ vertices, in most cases (Figure 3 left). Thus, we make progress (for the future vertices to arrive), unless an exceptional case arises.

However, if there exists a dense subgraph between S and T (which is an exceptional case), we cannot obtain a 3-color set of $\Omega(n)$ vertices. Indeed, in the worst case, the obtained 3-color set consists of $\Theta(n^{5/6})$ vertices. In such cases, we pick a pair of vertices $u_1, u_2 \in T$ where $Z = N_S(u_1) \cap N_S(u_2)$ is large, that is, $\Omega(n^{2/3+\epsilon})$ vertices for some $\epsilon > 0$ (Figure 3 right). If there exists a 4-coloring of the given graph in which distinct colors are used for u_1 and u_2 , then Z is indeed a 2-color set, leading to make progress (for the future vertices to arrive). We call this method *Common & Simplify technique*. With these two techniques, we eventually achieve $\tilde{O}(n^{14/17})$ colors, improving on Kierstead’s result [18] by a factor of $\tilde{O}(n^{1/102})$. We also believe that these techniques are useful for many other online coloring algorithms.

⁵If S_1 is not level-1, there exists a small subset $X \subseteq S_1$ where $\chi(G[X]) = \ell$, and therefore $\chi(G[X \cup \{v\}]) > \ell$ (and G is not locally ℓ -colorable).

⁶If both S_2 and S_3 are not level-2, there exist small subsets $X_2 \subseteq S_2$ and $X_3 \subseteq S_3$ where $\chi(G[X_2]) \geq \ell - 1$ and $\chi(G[X_3]) \geq \ell - 1$, and therefore $\chi(G[X_2 \cup X_3 \cup \{v, w_1, w_2\}]) > \ell$ (and G is not locally ℓ -colorable).

⁷It is equivalent to “applying FIRSTFIT using $2n^{1-\epsilon}$ colors, let S be the vertices colored by colors $1, 2, \dots, n^{1-\epsilon}$, let T be the vertices colored by colors $n^{1-\epsilon} + 1, n^{1-\epsilon} + 2, \dots, 2n^{1-\epsilon}$, and let U be the rest of the vertices”.

⁸Indeed, we only use $n^{14/17}, n^{13/17}$ colors in S, T in our algorithm, respectively, but for simplicity, we assume $O(n^{5/6})$ colors.



Figure 3: A sketch of our techniques in $k = 4$ case.

1.2.3 $k = 2$ case upper bound, **Theorem 1.3**

We apply randomization to Lovász, Saks, and Trotter’s algorithm [21] and show that this leads to better performance. By carefully analyzing how quickly the “third color” is forced to be used, we obtain an upper bound of $1.096 \log_2 n + O(1)$ colors. For a better result, we analyze how quickly the $(2L + 1)$ -th color is forced to be used for $L \geq 2$. We use a computer-aided approach based on dynamic programming. Eventually, with careful analysis up to $L = 10$, we improve the upper bound to $1.034 \log_2 n$.

1.2.4 $k = 2$ case lower bound, **Theorem 1.4**

We consider a binary-tree-like graph (Figure 4) and use the classic “potential function” argument. We introduce a potential function to represent the required number of colors, and show that any online algorithm would increase the potential by “ $\frac{3}{4}$ colors” per depth, leading to a lower bound of $\frac{3}{4} \log_2 n - O(1)$ colors. For a better result, we note that when the potential increase is smaller at one depth, a larger increase typically occurs at the next depth. We analyze how potential increases occur across two consecutive depths. However, the setting is more complex and requires a computer to brute-force all the essential cases. In the end, we obtain a lower bound of $\frac{91}{96} \log_2 n - O(1)$ colors.

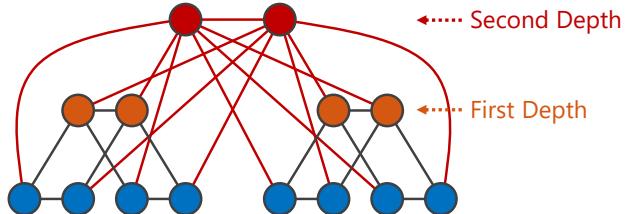


Figure 4: The input for the depth-2 case, which we consider for the lower bound. The vertices are arranged from bottom to top (the colors are used for readability purposes only).

1.3 Organization of this paper

This paper is organized as follows. In section 2, we present the preliminaries needed for the proof. In section 3, we present a deterministic online algorithm that obtains a general upper bound for $k \geq 5$ (Theorem 1.1), including a comparison with Kierstead’s previous algorithm (subsection 3.7). In section 4, we present a deterministic online algorithm for the special case $k = 4$ (Theorem 1.2). In section 5, we prove the upper bound $1.034 \log_2 n + O(1)$ for $k = 2$ (Theorem 1.3). In section 6, we prove the lower bound $\frac{91}{96} \log_2 n - O(1)$ for $k = 2$ (Theorem 1.4). Finally, in section 7, we conclude this paper with several remarks (including the reason why our proof techniques do not improve the $k = 3$ case) and several conjectures. Note that to understand section 4, the content of section 3 is required, but section 5 and section 6 are independent.

2 Preliminaries

Online coloring. We define the *online coloring problem* as follows. We are given an undirected graph $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_n\}$. We denote by n the number of vertices. We perform the following procedure for $i = 1, 2, \dots, n$: receive a vertex v_i together with its incident edges to some of v_1, v_2, \dots, v_{i-1} , and color v_i with a color $c(v_i) \in \mathbb{N}$. The next vertex v_{i+1} arrives only after coloring v_i . No two adjacent vertices are allowed to share the same color. The objective is to minimize the total number of colors used, that is, to minimize $\max\{c(v_1), c(v_2), \dots, c(v_n)\}$.

FIRSTFIT algorithm. The most straightforward strategy is to assign each new vertex the smallest available color. This strategy is often referred to as FIRSTFIT. We utilize it as a subroutine in our main algorithm.

ℓ -color set. In a graph $G = (V, E)$, a vertex set $S \subseteq V$ is “an ℓ -color set in some k -coloring of G ” if there exists a k -coloring of G such that ℓ or fewer colors appear in S . Otherwise, S is “not an ℓ -color set in any k -coloring of G ”. For example, the blue shaded area in Figure 5 left is a 1-color set in some 3-coloring of G , but the red shaded area in Figure 5 right is not a 1-color set in any 3-coloring of G . Note that if $\chi(G) > k$, S is *not* an ℓ -color set in any k -coloring of G .



Figure 5: The structure between S (shaded area) and the other vertices.

Notations. For a vertex $v \in V$, we denote its neighborhood by $N(v)$. For a vertex set $S \subseteq V$, we denote its neighborhood by $N(S) = \bigcup_{v \in S} N(v)$. For $X \subseteq V$, we denote $N_X(v) = N(v) \cap X$. In addition, we write $G[S]$ for the subgraph of G induced by $S \subseteq V$.

In general, we use O, Ω, Θ notations with respect to n , and ignore other constants. For example, in an online coloring algorithm for k -colorable graphs for a constant k , even if the number of colors used is $2^{2^k} \cdot n^{1/2}$, we write “ $O(n^{1/2})$ colors”. Also, we write $\tilde{O}(f(n)), \tilde{\Omega}(f(n))$ and $\tilde{\Theta}(f(n))$ to hide $\log n$ factors.

We assume all logarithms are base 2 unless stated otherwise (i.e., $\log n = \log_2 n$).

3 General bound

In this section, we present a deterministic online algorithm that colors any n -vertex k -colorable graph with $\tilde{O}(n^{1-2/(k(k-1))})$ colors, for any constant k . This significantly improves the previous results for $k \geq 5$. To this end, we first explain an online coloring algorithm that uses $O(n^{1-2/(k(k-1)+2)})$ colors, which is slightly worse than our best result. Generally speaking, we consider the situation when the number of vertices n is known in advance. This makes sense because of the following lemma (which is already proved in Lemma 1.10 of Kierstead [18]; for completeness, we give a proof here.)

Lemma 3.1 ([18]). *Suppose there exists a deterministic online coloring algorithm for a specific graph class (e.g., k -colorable graphs) with n vertices that uses $f(n)$ colors. Then, there exists a deterministic online coloring algorithm for this graph class that uses at most $4f$ (the current number of vertices) colors, at any moment. Here, we assume that $f(n)$ is an integer and $0 \leq f(n+1) - f(n) \leq 1$ for all n .*

Proof. We run the following algorithm, for $i = 0, 1, 2, \dots$:

Let t_i be the largest t such that $f(t) \leq 2^i$ (it satisfies $f(t_i) = 2^i$ by the assumption). For the next t_i vertices (or until all the vertices arrive), we use an algorithm for t_i -vertex graphs.⁹

Consider the moment when $i = i_0$. The $i_0 = 0$ case is obvious, so we suppose $i_0 \geq 1$. The current number of vertices m satisfies $m \geq t_{i_0-1}$, because the loop with $i = i_0 - 1$ is already finished. Thus, $f(m) \geq 2^{i_0-1}$. Also, the algorithm used at most $1 + 2 + \dots + 2^{i_0} = 2^{i_0+1} - 1$ colors, which is less than $4f(m)$. \square

Throughout this section, we denote n by the number of vertices in G .

3.1 Algorithm for locally ℓ -colorable graph

First, we consider an online algorithm for “locally” ℓ -colorable graphs, that is, graphs in which every small subgraph is ℓ -colorable. Formally, we give the following definition.

Locally ℓ -colorable. A graph $G = (V, E)$ is locally ℓ -colorable if for every subset $S \subseteq V$ with $|S| < 2^{2^\ell}$, we have $\chi(G[S]) \leq \ell$.

Note that locally ℓ -colorable graphs are not necessarily ℓ -colorable. Indeed, $\chi(G)$ could be arbitrarily large compared to ℓ , even for $\ell = 2$; cf., consider a graph G of large girth with $\chi(G)$ unbounded (it is well-known that such a graph does exist, by the classical result of Erdős, see [9]).

For $\ell = 1$, the graph has no edges, so we can color it online with 1 color. For $\ell = 2$, only $O(n^{1/2})$ colors are needed, by the following stronger theorem of Kierstead [18].

Theorem 3.2 ([18]). *There is a deterministic online algorithm that colors any graph that contains no odd cycles C_3 or C_5 using only $O(n^{1/2})$ colors.*

Our main contribution is to give a deterministic online algorithm for $\ell \geq 3$, with $O(n^{1-2/(\ell(\ell-1)+2)})$ colors. Since every ℓ -colorable graph is locally ℓ -colorable, this is indeed our main technical contribution (over Kierstead’s result [18]) for the general ℓ -colorable graphs. Below, we present our algorithm for $\ell \geq 3$.

3.2 Reduction to the subproblem

The first step is to apply the FIRSTFIT algorithm with colors $1, 2, \dots, n^{1-\epsilon}$, where $\epsilon = \frac{2}{\ell(\ell-1)+2}$.

However, sometimes there are no available colors for a vertex v . In this case, $|N_{S_{\text{now}}}(v)| \geq n^{1-\epsilon}$ holds, where S_{now} is the current set of vertices colored by FIRSTFIT. We now attempt to utilize the dense subgraph between S_0 and $T_0 = V \setminus S_0$ to color the remaining vertices online, where S_0 is the final set of vertices colored by FIRSTFIT. The overall framework for online coloring is shown in [Algorithm 1](#).

Algorithm 1 ONLINECOLORING($G = (V, E)$)

```

1: for each arrival of  $v \in V$  do
2:   if  $v$  can be colored by color  $1, 2, \dots, n^{1-\epsilon}$  then
3:     color  $v$  with FIRSTFIT
4:   else
5:     color  $v$  with a special algorithm                                 $\triangleright$  Uses color  $n^{1-\epsilon} + 1$  or bigger

```

⁹For example, for the $f(n) = \lceil \sqrt{n} \rceil$ case, we use an algorithm for 1-vertex graphs for the first 1 vertex, and we use an algorithm for 4-vertex graphs for the next 4 vertices, then we use an algorithm for 16-vertex graphs for the next 16 vertices, and so on.

The goal of this section is to present a deterministic online algorithm that colors T_0 with $O(n^{1-\epsilon})$ colors, that is, an algorithm referred to as the “special algorithm” in line 5. We denote such a problem as **else-problem**. Formally, the else-problem is the following:

- Color all vertices in T_0 (i.e., color all vertices not colored by FIRSTFIT). Indeed, each time a vertex in T_0 arrives, we apply this problem, but we never reset the algorithm (i.e., this algorithm leverages past iterations, see more details below).
- Let S_{now} and T_{now} be the “current” vertices which are subsets of S_0 and T_0 , respectively. When some vertex s that is colored by FIRSTFIT arrives, we add s to S_{now} .
- When some vertex t that is not colored by FIRSTFIT arrives, we add t to T_{now} and give a color to t . When coloring vertex $t \in T_0$, we are allowed to use structural information from $G[S_{\text{now}} \cup T_{\text{now}}]$, which is obtained from this algorithm previously applied to color vertices in T_{now} . Note that the information when coloring t can also be used for the “future” vertices that will not be colored by FIRSTFIT.

If an algorithm to solve the else-problem exists, our online coloring algorithm for locally ℓ -colorable graphs directly follows because the number of colors used is $n^{1-\epsilon} + O(n^{1-\epsilon}) = O(n^{1-\epsilon})$. Note that we use different colors for FIRSTFIT and for the else-problem (see comment on line 5).

However, solving the else-problem remains challenging. Indeed, if we do not take S_0 into account (i.e., we try to color T_0 online without using S_0), the problem remains the same. Therefore, we must use the fact that G is locally ℓ -colorable, and hence S_0 plays an important role when coloring vertices in T_0 . To this end, we first define the following *level-d set*:

Level-d set. We say that a subset $S \subseteq S_0$ is a *level-d set* if for every $X \subseteq V$ where $|X| < 2^{2^\ell - 2^d + 1}$, $X \cap S$ is an $(\ell - d)$ -color set in some ℓ -coloring of $G[X]$.

The reason we define this level-d set is that the level corresponds to the simplicity of the problem; the higher the level is, the easier it is to handle (see the overview in [subsubsection 1.2.1](#)). For example, the level- $(\ell - 1)$ set is the easiest to handle (see [subsection 3.3](#)).

Thus, we further break down the problem into the following simpler subproblems. In this subproblem, there is a parameter d called *level* (which corresponds to the above level-d set), and the $d = 0$ case corresponds to the else-problem. We try to solve this subproblem in the order of $d = \ell - 1, \ell - 2, \dots, 0$ (i.e., induction on the reverse order of d). See [Figure 6](#) for our objective.

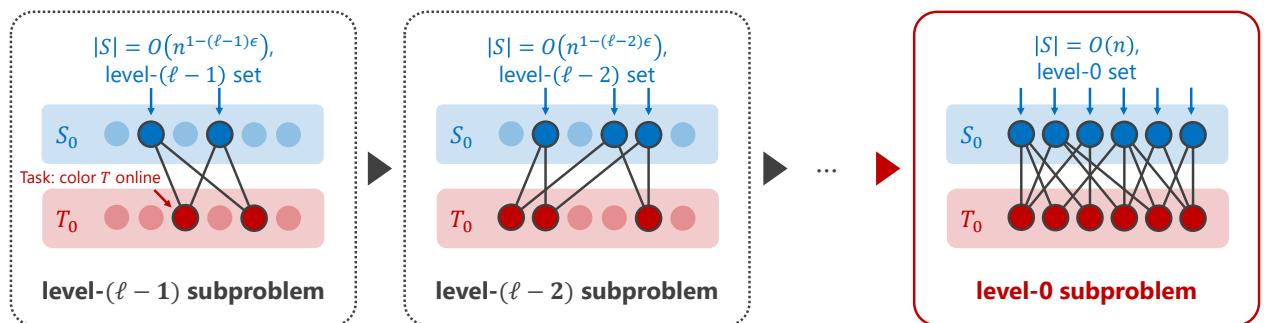


Figure 6: The sketch of the roadmap. We solve it in the order of $d = \ell - 1, \ell - 2, \dots, 0$. The bright vertices are S and T . Note that S and T are given in the order of arrival.

Level- d subproblem. We consider the subgraph $G[S \cup T]$ ($S \subseteq S_0, T \subseteq T_0$), where we need to color the vertices in T online. Initially, S and T are empty, and vertices are added to S and T in the order of arrival (i.e., sometimes vertices in S arrive after vertices in T). Every time a vertex v is added to T , we need to color the vertex v . It is guaranteed that $|S| \leq \gamma_d$ and $|N_S(v)| \geq \gamma_{d+1}$ ($v \in T$) at any moment, where $\gamma_i = n^{1-i\epsilon}/2^{i(i-1)/2}$ for each $i = 0, 1, \dots, \ell$. We can also assume that S is a level- d set.

However, at some point, we can *abort* the coloring when we can output a set $X \subseteq S \cup T$ that violates the condition for the level- d set, that is, $|X| < 2^{2^\ell - 2^d + 1}$ and $X \cap S$ is not an $(\ell - d)$ -color set in any ℓ -coloring of $G[X]$. Note that we must give a color to the last received vertex before aborting.

For the instance P of the subproblem, we denote S and T for this subproblem as $S(P)$ and $T(P)$.

Goal of subproblem. From the next section, we present an online algorithm for the subproblem that, at any point (until aborting), uses:

$$O\left(\frac{\max(|T|, n^{1-d\epsilon})}{n^\epsilon}\right) \text{ colors.}$$

Note that if the level-0 subproblem aborts, the given graph G is invalid (not locally ℓ -colorable). This is because there exists a set $X \subseteq V$ where $|X| < 2^{2^\ell}$ and $X \cap S$ is not even an ℓ -color set in any ℓ -coloring of $G[X]$, which means that $\chi(G[X]) > \ell$. Therefore, given that the final graph G is ℓ -colorable, we will show that our proposed algorithm for the else-problem uses $O(n^{1-\epsilon})$ colors.

Important Remarks. In this section, we present our algorithm inductively (i.e., recursively) on both ℓ and d . For this purpose, throughout this section, we assume that for all $\ell' < \ell$, there exists an online algorithm that colors any locally ℓ' -colorable graph $G' = (V', E')$ with $O(|V'|^{1-2/(\ell'(\ell'-1)+2)})$ colors, even when $|V'|$ is not known in advance (this is justified by [Lemma 3.1](#)). We denote such an algorithm by [LOCALLY](#)(ℓ'). Note that the base case for ℓ is $\ell = 1$ (this case obviously holds).

Moreover, we assume that for all $d' > d$, there exists an online algorithm that solves the level- d' subproblem with $O(\max(|T|, n^{1-d'\epsilon})/n^\epsilon)$ colors. We denote such an algorithm by [SUBPROBLEM](#)(d').

3.3 Base case for the subproblem ($d = \ell - 1$)

Lemma 3.3. *The base case of the subproblem (level $d = \ell - 1$) can be solved with*

$$O\left(\frac{\max(|T|, n^{1-(\ell-1)\epsilon})}{n^\epsilon}\right) \text{ colors.}$$

Proof. Consider coloring vertices in T by running [LOCALLY](#)($\ell - 1$). When $G[T]$ becomes not locally $(\ell - 1)$ -colorable, we can abort the procedure. To create an output for aborting, we first pick any $X_T \subseteq T$ such that $\chi(G[X_T]) \geq \ell$ and $|X_T| < 2^{2^{\ell-1}}$. Then, for each $t \in X_T$, let $\text{up}(t)$ be one vertex selected from $N_S(t)$ (if there are multiple vertices, we can select any). Let

$$X_S = \{\text{up}(t) : t \in X_T\}$$

and output $X = X_S \cup X_T$. The output X satisfies the abort conditions, because (i) $|X| < 2^{2^{\ell-1}+1}$ and (ii) $X \cap S = X_S$ is not a 1-color set in any ℓ -coloring of X ; if all the vertices in X_S have the same color in some ℓ -coloring of X , then $\chi(G[X_T]) \leq \ell - 1$ would hold, contradicting $\chi(G[X_T]) \geq \ell$. It remains to bound the number of colors used. By induction on ℓ , the algorithm uses $O(|T|^{1-2/((\ell-1)(\ell-2)+2)})$ colors. Since

$$|T|^{1-\frac{2}{(\ell-1)(\ell-2)+2}} \leq \frac{\max(|T|, n^{1-(\ell-1)\epsilon})}{n^\epsilon}$$

for $\epsilon = \frac{2}{\ell(\ell-1)+2}$, the number of colors used is within the required bound. Note that we need one additional color when aborting occurs, but it is negligible. \square

3.4 Inductive step on level d : Algorithm

Since we have solved the base case $d = \ell - 1$ of the subproblem, our next goal is to solve the $d \leq \ell - 2$ case, in the order of $d = \ell - 2, \ell - 3, \dots, 0$. Our algorithm for coloring a vertex when a vertex in T arrives, for the level- d subproblem, is described in [Algorithm 2](#).

Algorithm 2 SUBPROBLEM(d) for $d \leq \ell - 2

---$

```

1:  $\mathcal{P}, Q \leftarrow []$ 
2:  $D \leftarrow \emptyset$ 
3: for each arrival of  $v \in T$  do                                 $\triangleright S$  is also updated accordingly
4:    $N'_S(v) \leftarrow$  (the first  $\gamma_{d+1}$  elements in  $N_S(v)$ )
5:   if for some  $i, j$ ,  $|N'_S(v) \cap S(P_{i,j})| \geq \gamma_{d+2}$  and  $P_{i,j}$  is not aborted yet then
6:     color  $v$  using subroutine  $P_{i,j}$  (by adding  $v$  to  $T(P_{i,j})$ )            $\triangleright$  Use SUBPROBLEM( $d + 1$ )
7:   else
8:      $D \leftarrow D \cup \{v\}$ 
9:     color  $v$  with LOCALLY( $d$ ) (on  $D$ )
10:    if  $G[D]$  is not locally  $d$ -colorable anymore then
11:      find  $D' \subseteq D$  such that  $\chi(G[D']) > d$  and  $|D'| < 2^{2^d}$ 
12:      add  $D'$  to  $Q$ 
13:      for each  $q_{i,j} \in D'$  do
14:        initiate a new instance  $P_{i,j}$  of the level- $(d + 1)$  subproblem, and set  $S(P_{i,j}) \leftarrow N'_S(q_{i,j})$ 
15:        add the list of initiated subproblems  $[P_{i,1}, \dots, P_{i,|D'|}]$  to  $\mathcal{P}$ 
16:         $D \leftarrow \emptyset$ 
17:      if for some  $i$ , subroutines  $P_{i,1}, \dots, P_{i,|\mathcal{P}_i|}$  are all aborted then
18:        for  $j = 1, 2, \dots, |\mathcal{P}_i|$  do
19:           $X_j \leftarrow$  (the returned set for  $P_{i,j}$ )
20:      return  $X \leftarrow (X_1 \cup X_2 \cup \dots \cup X_{|\mathcal{P}_i|}) \cup Q_i$  and abort the procedure

```

Overview of SUBPROBLEM(d). Below, we describe how SUBPROBLEM(d) works. First, the key intuition is the following:

Assume that a small subset $D' \subseteq T$ with $\chi(G[D']) > d$ exists. Then, for at least one $x \in D'$, the subset $N_S(x)$ forms a level- $(d + 1)$ set, which is easier to handle (i.e., by induction, we can apply the level $d' > d$ subproblem), and therefore leads to some profit for future vertices (as we prove in [Lemma 3.4](#)). See [subsubsection 1.2.1](#) for examples (D' can be a vertex when $d = 0$, and D' can be two adjacent vertices when $d = 1$).

In our algorithm, to color the vertices in T , we normally use LOCALLY(d) ([Figure 7 left](#)). When $G[T]$ becomes not locally d -colorable, we can find a subset $D' \subseteq T$ such that $|D'| < 2^{2^d}$ and $\chi(G[D']) > d$. For each $x \in D'$, we simultaneously initiate a new instance P of the level- $(d + 1)$ subproblem where $S(P) = N'_S(x)$ ([Figure 7 middle](#)) and $N'_S(x)$ is the first γ_{d+1} vertices of $N_S(x)$. From now on, we refer to these initiated level- $(d + 1)$ subproblem instances as *subroutines*, for the sake of coloring the “future” vertices efficiently. The important fact is that, as long as S is a level- d set, at least one subroutine must survive ([Lemma 3.4](#)).

After the initiation, we now color a newly arrived vertex $v \in T$ in either of the following ways ([Figure 7 right](#)): (1) color v with an already initiated subroutine P when the number of neighbors of v to $S(P)$ is γ_{d+2} or more (otherwise, this would violate the input condition of the level- $(d + 1)$ subproblem), or (2) color

v with $\text{LOCALLY}(d)$ again, with completely new colors.¹⁰ Since the number of times that the initiation of new subroutines occurs is $O(n^\epsilon)$ due to the values γ_d, γ_{d+1} , and γ_{d+2} (Lemma 3.7), this algorithm uses the required number of colors.

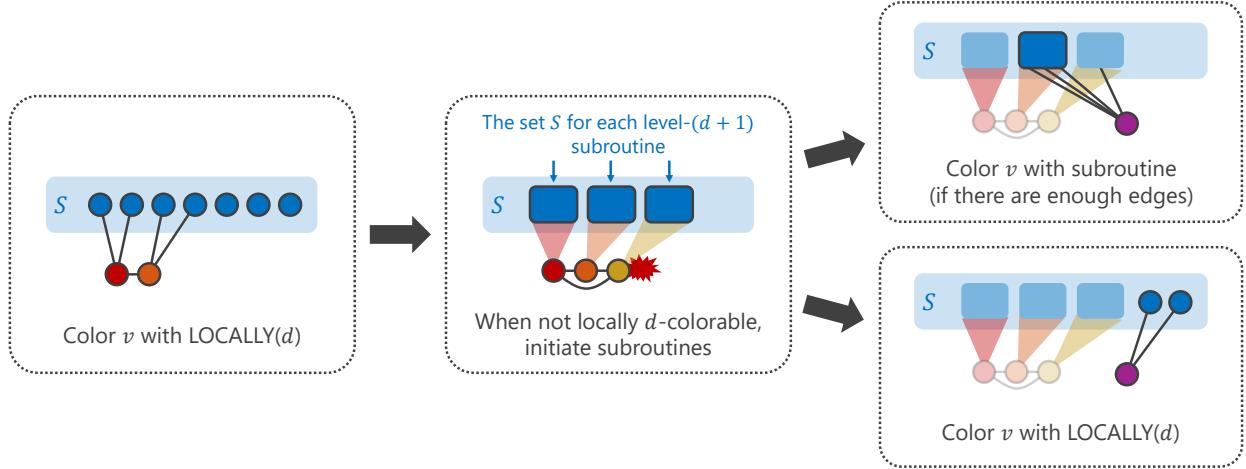


Figure 7: The sketch of the procedure when $d = 2$. In the middle and the right figure, each big rounded blue square in S is the set of adjacent vertices for each $v \in D'$.

Implementation. To implement this idea, let us first note that when a vertex $v \in T$ arrives, we have already recorded the current locally d -colorable subset $D \subseteq T$ (initially, D is empty).

We also recorded the list of currently initiated subroutines by a two-dimensional array $\mathcal{P} = [\mathcal{P}_1, \mathcal{P}_2, \dots] = [[P_{1,1}, \dots, P_{1,|\mathcal{P}_1|}], [P_{2,1}, \dots, P_{2,|\mathcal{P}_2|}], \dots]$ (again initially empty), where for each i, j , a subroutine $P_{i,j}$ was initiated, and the subroutines in the same \mathcal{P}_i were initiated at the same time.

We also recorded the list of small non- d -colorable subgraphs that caused the initiation of subroutines by an array of vertex sets $Q = [Q_1, Q_2, \dots]$, where each $Q_i = \{q_{i,1}, \dots, q_{i,|\mathcal{P}_i|}\}$ comes from $S(P_{i,j}) = N'_S(q_{i,j})$ for each j (Figure 8 left). Then, for each arrival of $v \in T$, we run the following procedure:

1. If there exists some i, j such that $|N'_S(v) \cap S(P_{i,j})| \geq \gamma_{d+2}$ and $P_{i,j}$ is not aborted, we color v with subroutine $P_{i,j}$ (lines 5–6).
2. Otherwise, color v with $\text{LOCALLY}(d)$, and add v to D (lines 8–9). If $G[D]$ becomes not locally d -colorable, we initiate new subroutines. We add the list of initiated subroutines to \mathcal{P} and the small non- d -colorable subgraph to Q . We also reset D (lines 10–16).

Note that at the timing when $G[D]$ becomes not locally d -colorable, we must pay one additional color to color v . From now on, we must use different colors at line 9.

Note that for 1, we use completely different colors among different subroutines, e.g., between $P_{1,1}$ and $P_{1,2}$. If all the subroutines initiated at the same time are aborted, it turns out that the assumption of the original subproblem is violated, so the original subproblem itself can be aborted (lines 18–20). Now for 2, after coloring v , we update \mathcal{P} and Q for the sake of coloring “future” vertices, but to be clear, the returned set $(X_1 \cup X_2 \cup \dots \cup X_{|\mathcal{P}_i|}) \cup Q_i$ is shown in Figure 8 right, where set X_j is the output of the aborted subroutine $P_{i,j}$ (see line 19).

¹⁰This means that we do not use the colors used in $\text{LOCALLY}(d)$ before the initiation of new subroutines. If the subgraph consisting of the vertices colored by $\text{LOCALLY}(d)$ (after the last initiation of subroutines) becomes not locally d -colorable, we initiate a new subroutine and restart $\text{LOCALLY}(d)$ again.

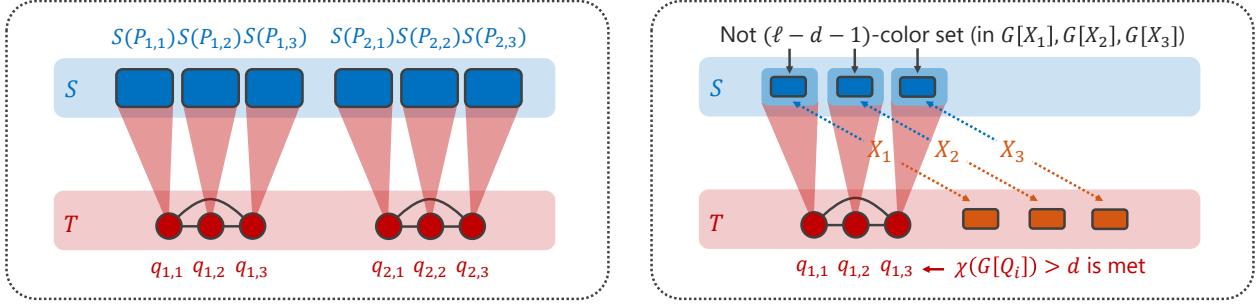


Figure 8: Left: the relationship between arrays \mathcal{P} and \mathcal{Q} . The red triangle shows the adjacency between vertex $q_{i,j}$ and the vertex set $S(P_{i,j})$. Note that subroutines $P_{1,1}, P_{1,2}, P_{1,3}$ are initiated earlier than subroutines $P_{2,1}, P_{2,2}, P_{2,3}$. Right: an example of graph $G[X]$ for the aborting output X , when $d = 2$. The output set X is the set of dark (red, dark blue, and orange) vertices.

3.5 Inductive step on level d : Analysis

In this subsection, we discuss the correctness of our proposed algorithm and analyze the number of colors used. The goal is to show that we achieve $O(n^{1-2/(\ell(\ell-1)+2)})$ colors (Theorem 3.9).

Lemma 3.4. *If the procedure SUBPROBLEM(d) aborts, for the returned set X at line 20, $X \cap S$ is not a $(\ell - d)$ -color set in any ℓ -coloring of $G[X]$ (see Figure 8 right for the structure of $G[X]$).*

Proof. Suppose for a contradiction that $X \cap S$ is an $(\ell - d)$ -color set in some ℓ -coloring of $G[X]$, that is, there exists an ℓ -coloring of $G[X]$ that colors $X \cap S$ by colors $\{1, 2, \dots, \ell - d\}$. Then, a contradiction occurs between the following two:

1. By $\chi(G[Q_i]) > d$, at least one vertex in Q_i , say $q_{i,j}$, is colored by some color $r \in \{1, 2, \dots, \ell - d\}$, in that ℓ -coloring. Since $q_{i,j}$ is adjacent to all vertices in $X_j \cap S$, the vertices in $X_j \cap S$ are colored by colors $\{1, 2, \dots, \ell - d\} \setminus \{r\}$, so $X_j \cap S$ is an $(\ell - d - 1)$ -color set in some ℓ -coloring of $G[X_j]$.
2. By the assumption on the returned set for aborting, $X_j \cap S$ is not an $(\ell - d - 1)$ -color set in any ℓ -coloring of $G[X_j]$.

Therefore, Lemma 3.4 holds. □

Lemma 3.5. *For the returned set X , $|X| < 2^{2^\ell - 2^d + 1}$ holds.*

Proof. First, by line 20, $|X| \leq |X_1| + |X_2| + \dots + |X_{|\mathcal{P}_i|}| + |Q_i|$ holds. By $|X_i| < 2^{2^\ell - 2^{d+1} + 1}$ (by induction on d) and $|\mathcal{P}_i|, |Q_i| < 2^{2^d}$ (by induction on ℓ), we have:

$$|X| \leq \left(2^{2^\ell - 2^{d+1} + 1} - 1\right) \cdot \left(2^{2^d} - 1\right) + \left(2^{2^d} - 1\right) < 2^{2^\ell - 2^d + 1}$$

Therefore, Lemma 3.5 holds. □

Next, we analyze the number of colors used in this procedure. To this end, we first show the following lemma (which follows from Problem 13.13 in [20]; for completeness, we give a proof here.)

Lemma 3.6 ([20]). *Let $s, a \geq 1$ be real numbers such that $|S| \leq s$. Let $A_1, \dots, A_m \subseteq S$ be sets that $|A_i| \geq \frac{s}{a}$ for each i and $|A_i \cap A_j| \leq \frac{s}{2a^2}$ for each i, j ($i \neq j$). Then, $m < 2a$ must hold.*

Proof. First, $A_1 \cup \dots \cup A_m$ is the disjoint union of $A_i \setminus (A_1 \cup \dots \cup A_{i-1})$ for $i = 1, \dots, m$. Since $|A_i \setminus (A_1 \cup \dots \cup A_{i-1})| \geq \max(|A_i| - (|A_1 \cap A_i| + \dots + |A_{i-1} \cap A_i|), 0) \geq \max(\frac{s}{a} - (i-1) \cdot \frac{s}{2a^2}, 0)$:

$$|S| \geq |A_1 \cup \dots \cup A_m| \geq \sum_{i=1}^m \max\left(\frac{s}{a} - (i-1) \cdot \frac{s}{2a^2}, 0\right)$$

The right-hand side is an increasing function of m , and its value is $\frac{m(4a-(m+1))}{4a^2}s$ for $m \leq 2a+1$. This value exceeds s at $m = 2a$, so due to $|S| \leq s$, the inequality above does not hold for $m \geq 2a$. Hence, $m < 2a$. \square

Lemma 3.7. $|\mathcal{P}| = O(n^\epsilon)$ holds.

Proof. While the procedure is not aborted, for each i , there exists t_i such that P_{i,t_i} is not aborted. We let $A_i = S(P_{i,t_i})$ for such t_i . Obviously, $|S| \leq \gamma_d$, $|A_i| \geq \gamma_{d+1}$, and $A_i \subseteq S$.

Also, $|A_i \cap A_j| \leq \gamma_{d+2}$ holds for each i, j ($i < j$). This is because vertex q_{j,t_j} arrives after \mathcal{P}_i has been added to \mathcal{P} (by $i < j$ and the reset of D on line 16). This means that if $|A_i \cap A_j| > \gamma_{d+2}$, vertex q_{j,t_j} should have been colored by subroutine P_{i,t_i} due to line 5, which is a contradiction.

Thus, by applying Lemma 3.6 with $s = \gamma_d$ and $a = 2^d n^\epsilon$ (note that $|A_i| \geq \gamma_{d+1} = \frac{\gamma_d}{a}$ and $|A_i \cap A_j| \leq \gamma_{d+2} = \frac{\gamma_d}{2a^2}$), we obtain $|\mathcal{P}| < 2a = O(n^\epsilon)$. \square

Lemma 3.8. Suppose $d \leq \ell - 2$. Before the procedure for the level- d subproblem aborts, it uses

$$O\left(\frac{\max(|T|, n^{1-d\epsilon})}{n^\epsilon}\right) \text{ colors.}$$

Proof. First, it suffices to consider the case when $|T| \geq n^{1-d\epsilon}$. This is because if we only use $O(n^{1-(d+1)\epsilon})$ colors when $|T| = n^{1-d\epsilon}$, it is trivial that we only use $O(n^{1-(d+1)\epsilon})$ colors when $|T| < n^{1-d\epsilon}$, which matches the requirement. Let c_1 and c_2 be the number of colors used for coloring vertices in D (line 9) and level- $(d+1)$ subroutines (line 6), respectively. Then, when $|T| \geq n^{1-d\epsilon}$, the number of colors used for the subproblem is $c_1 + c_2$ colors, and each case is as follows:

- c_1 : By Lemma 3.7, we run LOCALLY(d) for at most $|\mathcal{P}| = O(n^\epsilon)$ times. By induction on ℓ , each run requires $O(|D|^{1-2/(d(d-1)+2)})$ colors. Since $|D|^{1-2/(d(d-1)+2)}$ is a concave function of $|D|$, the worst case is when $|\mathcal{P}| = \Theta(n^\epsilon)$ and $|D| = \Theta(\frac{|T|}{n^\epsilon})$ for all runs. In such a case, each run requires only $O(\frac{|D|}{n^\epsilon})$ colors because

$$|D|^{1-\frac{2}{d(d-1)+2}} \leq \frac{|D|}{n^\epsilon}$$

holds when $|T| \geq n^{1-d\epsilon}$ (i.e., $|D| = \Omega(n^{1-(d+1)\epsilon})$) and $\epsilon = \frac{2}{\ell(\ell-1)+2}$. Therefore, $c_1 = O(\frac{|T|}{n^\epsilon})$.

- c_2 : By induction on d :

$$c_2 = O\left(\sum_{i,j} \frac{\max(|T(P_{i,j})|, n^{1-(d+1)\epsilon})}{n^\epsilon}\right)$$

and by $i = O(n^\epsilon)$ (Lemma 3.7), $j = O(1)$ (by line 11), and $\sum_{i,j} |T(P_{i,j})| \leq |T|$, we have $c_2 = O(\frac{|T|}{n^\epsilon})$.

Therefore, $c_1 + c_2 = O(\frac{|T|}{n^\epsilon})$ when $|T| \geq n^{1-d\epsilon}$, and Lemma 3.8 follows. \square

Theorem 3.9. There exists a deterministic online algorithm that colors any locally ℓ -colorable graphs with $O(n^{1-2/(\ell(\ell-1)+2)})$ colors, for $\ell \geq 2$.

Proof. First, we show that $\text{SUBPROBLEM}(d)$ successfully solves the level- d subproblem. To prove the correctness, we must prove the following three: (1) upon aborting, $X \cap S$ is not a $(\ell - d)$ -color set in any ℓ -coloring of $G[X]$, (2) $|X|$ is within the limit, and (3) the number of colors used is as we claimed. (1) follows from [Lemma 3.3](#) and [Lemma 3.4](#), (2) follows from [Lemma 3.3](#) and [Lemma 3.5](#), and (3) follows from [Lemma 3.3](#) and [Lemma 3.8](#). Therefore, $\text{SUBPROBLEM}(d)$ works successfully.

By applying $\text{SUBPROBLEM}(d)$ for $d = \ell - 1, \ell - 2, \dots, 0$, $\text{SUBPROBLEM}(0)$ solves the else-problem using $O(n^{1-\epsilon})$ colors. Combining this with [Algorithm 1](#), by induction on ℓ , we obtain a deterministic online algorithm that colors any locally ℓ -colorable graph using $O(n^{1-\epsilon})$ colors. This result also holds when the number of vertices n is not known in advance, due to [Lemma 3.1](#). Therefore, [Theorem 3.9](#) holds. \square

3.6 The improvement

In the previous subsections, we showed a deterministic algorithm that colors any locally ℓ -colorable graph with $O(n^{1-2/(\ell(\ell-1)+2)})$ colors. This algorithm also colors k -colorable graphs with $O(n^{1-2/(k(k-1)+2)})$ colors, because k -colorable graphs are always locally k -colorable. Compared to the current best result of $O(n^{1-1/k!})$ colors, the exponential becomes quadratic, which is already a big improvement.

In this subsection, we further improve the result to $\tilde{O}(n^{1-2/(k(k-1))})$ colors. The key to the improvement is that, while only an $O(n^{1/2})$ -color algorithm is known for locally 2-colorable graphs, there is an $O(\log n)$ -color algorithm for 2-colorable (bipartite) graphs, by Lovász, Saks, and Trotter [21]. We define the following subproblem, where the $d = 0$ case is the same as the else-problem when the given graph G is k -colorable (not just locally k -colorable). Then, we can use the $O(\log n)$ -color algorithm as the base case ($k = 2$).

Subproblem+. We consider a modification of the subproblem,¹¹ which is specifically created for the case when G is k -colorable. We set $\epsilon = \frac{2}{k(k-1)}$, and guarantee the constraints on $|S|$ and $|N_S(v)|$ with the new ϵ . We can also assume that S is a $(k-d)$ -color set in some k -coloring of G . The aborting condition also changes; we can abort if it turns out that S is not a $(k-d)$ -color set. Note that, since the procedure only knows the information of $G[S \cup T]$, aborting is a valid move only when S is not a $(k-d)$ -color set in some k -coloring of $G[S \cup T]$. We are not required to return anything when aborting.

Our objective is to give an algorithm with the following number of colors (until aborting):

$$\tilde{O}\left(\frac{\max(|T|, n^{1-d\epsilon})}{n^\epsilon}\right).$$

Solution to subproblem+. First, we consider the base case $d = k - 1$. Let $\text{COLORING}(k - 1)$ be an online algorithm that colors any $(k - 1)$ -colorable graph $G' = (V', E')$ using $\tilde{O}(|V'|^{1-2/((k-1)(k-2))})$ colors. Note that such an algorithm exists by induction on k , where the base case is $k = 2$, in which case, as mentioned above, an $O(\log |V'|)$ -color algorithm exists [21]. Then, the base case can be solved by running $\text{COLORING}(k - 1)$, because, if S is a 1-color set, $\chi(G[T]) \leq k - 1$ holds. Since

$$|T|^{1-\frac{2}{(k-1)(k-2)}} \leq \frac{\max(|T|, n^{1-(k-1)\epsilon})}{n^\epsilon}$$

holds when $\epsilon = \frac{2}{k(k-1)}$, the required number of colors is satisfied.

Next, for the case of $d \in \{0, 1, \dots, k - 2\}$ inductively, the same algorithm as [Algorithm 2](#) works, because of the following [Lemma 3.10](#) and [Lemma 3.11](#).

¹¹In subproblem+, we consider the setting of subgraph $G[S \cup T]$ ($S \subseteq S_0, T \subseteq T_0$), similarly to the original subproblem.

Lemma 3.10. *If there is an i such that subroutines $P_{i,1}, \dots, P_{i,|\mathcal{P}_i|}$ are all aborted at line 17, S is not a $(k-d)$ -color set in any k -coloring of G .*

Proof. We prove this lemma in the same manner as [Lemma 3.4](#). Suppose for a contradiction that S is a $(k-d)$ -color set in some k -coloring of G , that is, there exists a k -coloring of G that colors S by colors $\{1, 2, \dots, k-d\}$. Then, a contradiction occurs between the following two:

1. By $\chi(G[Q_i]) > d$, at least one vertex in Q_i , say $q_{i,j}$, is colored by some color $r \in \{1, 2, \dots, k-d\}$, in that k -coloring. Since $q_{i,j}$ is adjacent to all vertices in $S(P_{i,j})$, vertices in $S(P_{i,j})$ are colored by colors $\{1, 2, \dots, k-d\} \setminus \{r\}$, so $S(P_{i,j})$ is a $(k-d-1)$ -color set.
2. By the assumption for aborting, $S(P_{i,j})$ is not a $(k-d-1)$ -color set.

Therefore, [Lemma 3.10](#) holds. \square

Lemma 3.11. *If the number of colors used in the base case $d = k-1$ satisfies the required number of colors, the number of colors used for $d \leq k-2$ also satisfies the required number of colors.*

Proof. Consider using the same manner as [Lemma 3.8](#). Obviously, c_2 in [Lemma 3.8](#) satisfies the required number of colors. c_1 also satisfies the required number of colors, because when $\epsilon = \frac{2}{k(k-1)}$,

$$|D|^{1-\frac{2}{d(d-1)+2}} \leq \frac{|D|}{n^\epsilon}$$

holds for all $d \leq k-2$ when $|D| = \Theta(\frac{|T|}{n^\epsilon})$. Therefore, [Lemma 3.11](#) holds. \square

By [Lemma 3.10](#) and [Lemma 3.11](#), we now complete the proof of an $\tilde{O}(n^{1-2/(k(k-1))})$ -color algorithm for k -colorable graphs, which is our best result obtained so far. Formally, we have the following theorem.

Theorem 3.12. *There exists a deterministic online algorithm that colors any k -colorable graph with $\tilde{O}(n^{1-2/(k(k-1))})$ colors.*

3.7 Comparison with Kierstead's algorithm

As mentioned in [section 1](#), Kierstead discovered an algorithm that uses $\tilde{O}(n^{2/3})$ colors for $k=3$ and $\tilde{O}(n^{5/6})$ colors for $k=4$. These algorithms use a framework similar to the algorithm described in [subsection 3.6](#). Indeed, although Kierstead employs his original *witness-tree* idea, what he does is essentially the same as in [subsection 3.6](#). However, his ideas are not enough to give an algorithm for locally ℓ -colorable graphs when $\ell \geq 3$, and as a result, it is only possible to obtain an algorithm with $O(n^{1-1/k!})$ colors. From [subsection 3.2](#) to [subsection 3.5](#), we ultimately find an algorithm that also works for $\ell \geq 3$, with $\tilde{O}(n^{1-2/(\ell(\ell-1)+2)})$ colors, leading to an improvement from exponential to quadratic.

3.8 The competitive ratio

We have shown a deterministic online algorithm to color locally ℓ -colorable graphs with $O(n^{1-2/(\ell(\ell-1)+2)})$ colors ([Theorem 3.9](#)). To analyze the competitive ratio of this algorithm, we need to examine the constant factor in the number of colors.

Lemma 3.13. *For the algorithms in [section 3](#), the following statements hold for all $\ell \geq 1$ and $0 \leq d \leq \ell-1$, where $\epsilon = \frac{2}{\ell(\ell-1)+2}$:*

- (a) *The algorithm for locally ℓ -colorable graphs uses at most $2^{2^{\ell+2}} \cdot n^{1-\epsilon}$ colors (when n is not known in advance), and*

(b) $\text{SUBPROBLEM}(d)$ uses at most $2^{2^{\ell+2}-2^{d+2}} \cdot \max(|T|, n^{1-d\epsilon})/n^\epsilon$ colors.

Proof. We prove by induction on ℓ . In order to prove the lemma for a pair (ℓ, d) , we assume that (a) holds for all smaller values of ℓ , and (b) holds for all larger values of d .

First, we prove (a). It is obvious for $\ell = 1$. For $\ell \geq 2$, in the case where n is known in advance, we use $2^{2^{\ell+2}-4} \cdot n^{1-\epsilon}$ colors for the else-problem (which is the same as $\text{SUBPROBLEM}(0)$) by assumption (b), and $n^{1-\epsilon}$ colors for FIRSTFIT . Since n is not known in advance, the number of colors is multiplied by 4 (Lemma 3.1). Since $4 \cdot (2^{2^{\ell+2}-4} + 1) \leq 2^{2^{\ell+2}}$, (a) holds.

Next, we prove (b). For the base case $d = \ell - 1$, by the proof of Lemma 3.3, we only use

$$2^{2^{\ell+1}} \cdot \frac{\max(|T|, n^{1-(\ell-1)\epsilon})}{n^\epsilon}$$

colors, which means that (b) holds. It remains to prove (b) for the cases where $d \leq \ell - 2$. We follow the proof of Lemma 3.8; we define c_1 and c_2 in the same way, and assume $|T| \geq n^{1-d\epsilon}$. By the proof of Lemma 3.8, $c_1 \leq 2^{2^{d+2}} \cdot \frac{|T|}{n^\epsilon} \cdot \max(\frac{|\mathcal{P}|}{n^\epsilon}, 1) \leq 2^{2^{d+2}+d+1}$ holds (by $|\mathcal{P}| \leq 2 \cdot \frac{\gamma_d}{\gamma_{d+1}} = 2^{d+1}n^\epsilon$; Lemma 3.7), and for c_2 ,

$$c_2 = 2^{2^{\ell+2}-2^{d+3}} \sum_{i,j} \frac{\max(|T(P_{i,j})|, n^{1-(d+1)\epsilon})}{n^\epsilon} \leq 2^{2^{\ell+2}-2^{d+3}} \sum_{i,j} \left(\frac{|T(P_{i,j})|}{n^\epsilon} + n^{1-(d+2)\epsilon} \right)$$

holds. Here, $i \leq 2 \cdot \frac{\gamma_d}{\gamma_{d+1}} = 2^{d+1}n^\epsilon$ (by Lemma 3.7), $j \leq 2^{2^d}$ (by line 11), and $\sum_{i,j} |T(P_{i,j})| \leq |T|$. Therefore:

$$\begin{aligned} c_2 &\leq 2^{2^{\ell+2}-2^{d+3}} \left\{ \frac{|T|}{n^\epsilon} + (2^{d+1}n^\epsilon) \cdot 2^{2^d} \cdot n^{1-(d+2)\epsilon} \right\} \\ &= 2^{2^{\ell+2}-2^{d+3}} \left(\frac{|T|}{n^\epsilon} + 2^{2^d+d+1}n^{1-(d+1)\epsilon} \right) \\ &\leq 2^{2^{\ell+2}-2^{d+3}} (2^{2^d+d+1} + 1) \frac{|T|}{n^\epsilon} \\ &\leq 2^{2^{\ell+2}-2^{d+2}-1} \cdot \frac{|T|}{n^\epsilon} \end{aligned}$$

In the first inequality, we use the fact that $\frac{|T|}{n^\epsilon} \geq n^{1-(d+1)\epsilon}$. In the second inequality, we use the fact that $2^{2^d+d+1} + 1 \leq 2^{2^{d+2}-1}$ for all $d \geq 0$. Finally, since $2^{2^{d+2}+d+1} + 2^{2^{\ell+2}-2^{d+2}-1} \leq 2^{2^{\ell+2}-2^{d+2}}$ for $d \leq \ell - 2$, the total number of colors satisfies $c_1 + c_2 \leq 2^{2^{\ell+2}-2^{d+2}} \cdot \frac{|T|}{n^\epsilon}$. Therefore, (b) holds. \square

Finally, we prove the main result of this subsection. The previous best-known competitive ratio is $O(n \log \log n / \log \log \log n)$ by Kierstead [18], which yields a deterministic online algorithm that colors k -colorable graphs with $n^{1-1/k!}$ colors. Theorem 3.14 improves this ratio by a factor of $\log \log \log n$.

Theorem 3.14. *There exists a deterministic online coloring algorithm with competitive ratio $O(\frac{n}{\log \log n})$.*

Proof. First, we prove that, for all $\ell \leq \frac{1}{2} \log \log n$, the following holds for sufficiently large n :

$$2^{2^{\ell+2}} \cdot n^{1-1/(\ell(\ell-1)+2)} < \frac{n}{\log \log n}$$

Taking the logarithm of each side, we get the following:

$$\begin{aligned} 2^{\ell+2} + \left(1 - \frac{1}{\ell(\ell-1)+2}\right) \log n &< \log n - \log \log \log n \\ (\Leftrightarrow) 2^{\ell+2} + \log \log \log n &< \frac{\log n}{\ell(\ell-1)+2} \end{aligned}$$

Here, for $\ell \leq \frac{1}{2} \log \log n$, the left-hand side is $o(\log n / (\log \log n)^2)$ due to $2^{\ell+2} = O(\sqrt{\log n})$. However, the right-hand side is $\Omega(\log n / (\log \log n)^2)$. Therefore, the inequality holds for sufficiently large n . Combining this with [Lemma 3.13](#), the algorithm for ℓ -colorable graphs where $\ell \leq \frac{1}{2} \log \log n$ only uses $\frac{n}{\log \log n}$ colors, when n is sufficiently large.

Now, we describe an algorithm that works even when we do not know ℓ . Consider the following algorithm: (1) run an algorithm for locally $(\frac{1}{2} \log \log n)$ -colorable graphs, and (2) if the algorithm aborts, give up and use a different color for each vertex. The number of colors used is at most

$$\begin{cases} \frac{n}{\log \log n} & (\chi(G) \leq \frac{1}{2} \log \log n) \\ n & (\chi(G) > \frac{1}{2} \log \log n) \end{cases}$$

when n is sufficiently large. Therefore, this algorithm achieves a competitive ratio of $O(\frac{n}{\log \log n})$. Note that this algorithm works in polynomial time. The non-trivial point is whether we can find $D' \subseteq D$ where $\chi(G[D']) > d$ and $|D'| < 2^{2^d}$ in polynomial time, but it can be done by looking at the output of the level-0 subproblem (of the problem for locally d -colorable graphs that colors vertices in D). \square

4 Improvement for 4-colorable graphs

In this section, we present a deterministic online algorithm to color a 4-colorable graph G of n vertices with $\tilde{O}(n^{14/17})$ colors, improving the previous bound of $\tilde{O}(n^{5/6})$. First, we describe the idea of our algorithm.

4.1 The idea of the algorithm

The first step follows the same approach as in [subsection 3.2](#) and uses the FIRSTFIT algorithm with $n^{14/17}$ colors. Then, we solve the else-problem under the following assumptions, which (2) is a new one.

Assumptions for the else-problem. (1) When $v \in T_0$ arrives, $|N_{S_{\text{now}}}(v)| \geq n^{14/17}$ is guaranteed.¹² (2) There is no edge (s, t) ($s \in S_0, t \in T_0$) in G that s arrives *later* than t .

For the assumption (2), this is not necessarily true for the “actual” graph, but we can ignore all such edges (s, t) . It does not cause a problem because $|N_{S_{\text{now}}}(v)| \geq n^{14/17}$ still holds and the deletion of edges between S_0 and T_0 does not change the validity of the coloring (we color the vertices in T_0 in the else-problem, with completely different colors from those in S_0). Then, $N_{S_{\text{now}}}(v) = N_{S_0}(v)$ holds at any moment for any already arrived vertex v . In this section, we often access $N_{S_0}(v)$ in the algorithms. This may seem to use “future” information, but it is not the case because of this assumption; we can just access $N_{S_{\text{now}}}(v)$.

Our task is to color the vertices in T_0 online using $\tilde{O}(n^{14/17})$ colors. However, it remains challenging. Now, we first consider the special case when there is no dense subgraph in G .

No-dense property. There are no subsets $S_D \subseteq S_0$ and $T_D \subseteq T_0$ that satisfy the following conditions: (i) $|T_D| \leq n^{8/17}$, (ii) $|S_D| \leq n^{8/17}|T_D|$, and (iii) $|N_{S_D}(v)| \geq \frac{1}{3}n^{14/17}$ for all $v \in T_D$.

The steps to solve the special case are shown in [Figure 9](#). The first step is to obtain a 1-color set $T_D \subseteq T_0$ in some 4-coloring of G , where $|T_D| = \Omega(n^{7/17})$ ([Figure 9A](#)). To obtain a large 1-color set, we use FIRSTFIT in T_0 (see the details in [subsection 4.2](#)). Since we run FIRSTFIT in both S_0 and T_0 , we call this technique the *double greedy method*. Then, assuming the no-dense property, we obtain a large 3-color set in S_0 ([Figure 9B](#)), because indeed, $N_{S_0}(T_D)$ is a 3-color set of $\Omega(n^{15/17})$ vertices. Moreover, repeating this process allows us

¹²As in [subsection 3.2](#), S_0 and T_0 is the “final” set of vertices that are colored and uncolored by FIRSTFIT, respectively, and S_{now} is a “current” set of vertices that are colored by FIRSTFIT.

to cover S_0 with $\tilde{O}(n^{2/17})$ 3-color sets (Figure 9C). Afterward, we color the remaining vertices using the subproblem where the subset $S \subseteq S_0$ is a 3-color set, as we did in subsection 3.6 (Figure 9D).

Compared to Kierstead's algorithm for $k = 4$, which generates $\tilde{O}(n^{1/6})$ 3-color sets, our algorithm generates fewer 3-color sets. This leads to the improvement from $\tilde{O}(n^{5/6})$ colors to $\tilde{O}(n^{14/17})$ colors.

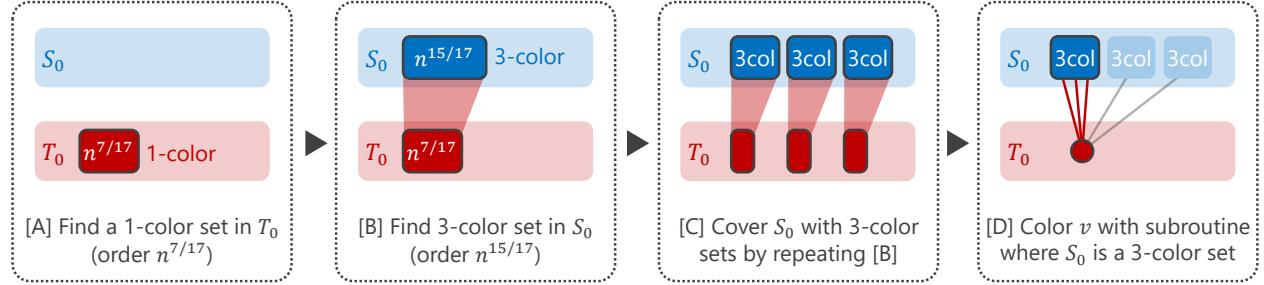


Figure 9: The sketch of our algorithm for the no-dense case.

However, in some cases, the no-dense property does not hold. Indeed, the worst case is $|S_D| = n^{14/17}$ even though $|T_D| = n^{8/17}$. To this end, we use our novel *Common & Simplify technique* to reduce the else-problem to one with the no-dense property. The core idea of the technique is to pick two highly common vertices $u_1, u_2 \in T_0$ that satisfy $|N_{S_0}(u_1) \cap N_{S_0}(u_2)| = \tilde{\Omega}(n^{12/17})$ (we can show that such a pair exists if the no-dense property does not hold; see Lemma 4.9). If there exists a 4-coloring of G in which different colors are used for u_1 and u_2 , $N_{S_0}(u_1) \cap N_{S_0}(u_2)$ is a 2-color set of $\tilde{\Omega}(n^{12/17})$ vertices (Figure 10), which is large enough to reduce to the 2-color version of subproblem+. Note that sometimes u_1 and u_2 may be assigned the same color, but we can force (u_1, u_2) to be different colors with at least $\frac{1}{4}$ ratio (see subsection 4.6).

In subsection 4.2, we first show the preliminaries that are required for the proof. In subsection 4.3 and subsection 4.4, we present an algorithm for the no-dense property case. In subsection 4.5 and subsection 4.6, we introduce the Common & Simplify technique and finally present a deterministic online algorithm that colors any 4-colorable graph with $\tilde{O}(n^{14/17})$ colors.

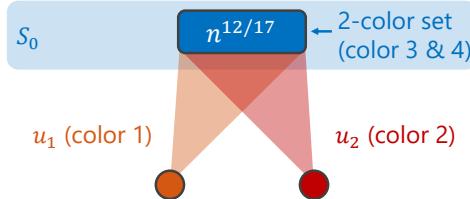


Figure 10: The core idea in Common & Simplify technique. If u_1 and u_2 are colored by color 1 and 2 in some 4-coloring of G , then $N_{S_0}(u_1) \cap N_{S_0}(u_2)$ is forced to be a 2-color set.

4.2 Preliminaries

First, we define three subproblems that our algorithm requires.

3-color set problem. We consider a modification of the level-1 subproblem+, where S is initially set and is fixed throughout the procedure. It is guaranteed that $|N_S(v)| = \tilde{\Omega}(|S|/n^{3/17})$ for each $v \in T$. We can assume that S is a 3-color set in some 4-coloring of G . If this assumption is violated, we can abort the procedure. Our task is to color the vertices in T online, using $\tilde{O}(\max(|T|, n^{15/17})/n^{3/17})$ colors. The algorithm for solving the 3-color set problem is denoted as $\text{3-COLOR}(S)$.

2-color set problem. We consider a modification of the level-2 subproblem+, where S is initially set and is fixed throughout the procedure. It is guaranteed that $|N_S(v)| = \tilde{\Omega}(|S|/n^{3/17})$ for each $v \in T$. We can assume that S is a 2-color set in some 4-coloring of G . If this assumption is violated, we can abort the procedure. Our task is to color the vertices in T online, using $\tilde{O}(\max(|T|, n^{12/17})/n^{3/17})$ colors. The algorithm for solving the 2-color set problem is denoted as $\text{2-COLOR}(S)$.

1-color candidates problem. We consider a modification of the level-0 subproblem+. Similar to the original subproblem+ (and unlike 3-color/2-color set problems), vertices are added to S and T in the order of arrival. It is guaranteed that $|S| \leq n^{16/17}$ and $|N_S(v)| \geq \frac{1}{2}n^{13/17}$ ($v \in T$) at any moment. Our task is to color the vertices in T online, using $O(\max(|T|, n^{16/17})/n^{5/17})$ colors, until we abort the procedure by outputting the following set:

A set $\mathcal{A} = \{A_1, A_2, \dots, A_m\}$ ($A_i \subseteq S$) that satisfies the following conditions: $|A_i| \geq \frac{1}{64}n^{7/17}$ for all $i, m \leq 10$, and there exists some i such that A_i is a 1-color set in some 4-coloring of G . (It means that no matter how “future” vertices arrive, at least one A_i must be a 1-color set as long as the graph is 4-colorable.)

The algorithm to solve the 1-color candidates problem is denoted as $\text{1-COLOR-CAND}(S)$. Note that sketches of the three defined problems are shown in Figure 11.

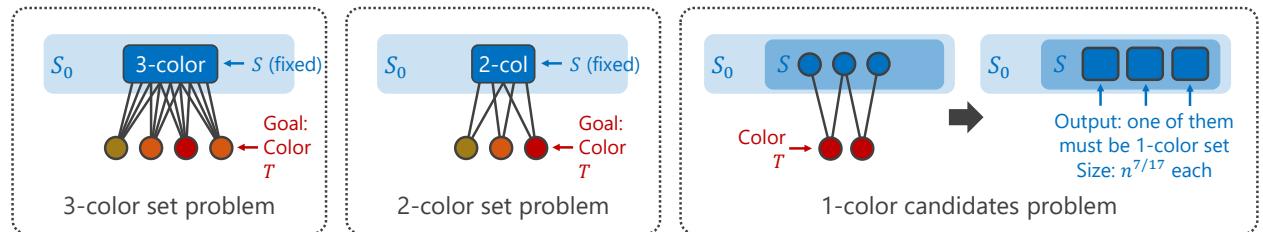


Figure 11: The sketches of the three problems.

Lemma 4.1. *A deterministic online algorithm that solves both the 3-color set problem and the 2-color set problem exists.*

Proof. We first consider the “general $|S|$ ” version of the level- d subproblem+ for k -colorable graphs, which makes the following modifications (consider $\epsilon = \frac{2}{k(k-1)}$ as in subproblem+):

S is initially a set and is fixed throughout the procedure, but there are no constraints on $|S|$ (ultimately, only $|S| \geq 1$ is required). It is guaranteed that $|N_S(v)| \geq |S|/(2^d n^\epsilon)$ for each $v \in T$.

In this modified problem, the same algorithm as subproblem+, except that we set $\gamma_i = |S|/(2^{i(i-1)/2} n^{i\epsilon})$, works (note that $\frac{\gamma_d}{\gamma_{d+1}} = 2^d n^\epsilon$, which is not modified). This is because, in the proof for subproblem+ (cf., Lemma 3.7), we only used the fact that $|N_S(v)| \geq |S|/(2^d n^\epsilon)$, not the size of $|S|$ itself. Therefore, we can still solve this problem with $\tilde{O}(\max(|T|, n^{1-d\epsilon})/n^\epsilon)$ colors. The same result also holds when the constraints are changed to $|N_S(v)| \geq \tilde{\Omega}(|S|/n^\epsilon)$, by the proof of subproblem+.

The 3-color/2-color set problems are the $(k, d) = (4, 1), (4, 2)$ cases of the modified problem, where “ n ” is replaced by $n^{18/17}$ (this corresponds to the case if G were a $n^{18/17}$ -vertex graph). Therefore, there exists a deterministic online algorithm that solves the 3-color/2-color set problems. \square

Lemma 4.2. *There exists a deterministic online algorithm that solves the 1-color candidates problem.*

Proof. We consider the following level- d subproblem for the 1-color candidate problem. The $d = 0$ case corresponds to the 1-color candidate problem.

It is guaranteed that $|S| \leq \gamma_d$ and $|N_S(v)| \geq \gamma_{d+1}$ ($v \in T$), where $\gamma_i = n^{(16-3i)/17}/2^{i(i+1)/2}$. The output \mathcal{A} when aborting must satisfy the following condition: assuming that S is a $(4-d)$ -color set in some 4-coloring of G , there exists some i that A_i is a 1-color set in some 4-coloring of G . Note that any \mathcal{A} meets the condition if we know that S is not a $(4-d)$ -color set. We only use $O(\max(|T|, n^{(16-3d)/17})/n^{5/17})$ colors (at any moment until aborting).

The $d = 2$ case. For the base case $d = 2$, we color vertices in T using an algorithm for graphs that contain no odd cycles C_3 or C_5 (Theorem 3.2), with $O(|T|^{1/2})$ colors. When a small odd cycle (c_1, c_2, \dots, c_m) ($m \in \{3, 5\}$) appears in $G[T]$, we abort the procedure by outputting the following (also see Figure 12 left):

$$\mathcal{A} = \{N_S(c_1), N_S(c_2), \dots, N_S(c_m)\}$$

Note that $|N_S(c_i)| \geq \gamma_3 = \frac{1}{64}n^{7/17}$ for each i . From the proof of Lemma 3.10 and by $\chi(G[\{c_1, c_2, \dots, c_m\}]) \geq 3$, assuming that S is a 2-color set, at least one of $N_S(c_i)$ is a 1-color set, and hence \mathcal{A} is valid. The number of colors used meets the requirement by the following inequality:

$$|T|^{1/2} \leq \frac{\max(|T|, n^{10/17})}{n^{5/17}}$$

The $d \leq 1$ case. Next, the $d \in \{0, 1\}$ case can be solved using the same way as in subproblem+, except that we need to output \mathcal{A} when aborting. When all the subroutines initiated at the same time abort (line 17 of Algorithm 2), and the output of the subroutine $P_{i,j}$ is \mathcal{A}_j (corresponds to line 19 of Algorithm 2), we can abort the procedure by outputting the following (also see Figure 12 middle/right):

$$\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \dots \cup \mathcal{A}_{|\mathcal{P}_i|}$$

The correctness of this algorithm follows from the proof of Lemma 3.10: assuming that S is a $(4-d)$ -color set, $S(P_{i,j})$ is a $(3-d)$ -color set for at least one j , and for such j , there is at least one 1-color set in \mathcal{A}_j . We can also prove that the number of colors used meets the requirement in the same manner as in the proof of Lemma 3.8. Finally, we estimate the order of the returned set \mathcal{A} . For $d = 2$, $|\mathcal{A}| \leq 5$ holds. For $d = 1$ and $d = 0$, the found ‘‘small subgraph’’ when running LOCALLY(d) always is of order 2 (an edge) and order 1 (a vertex), respectively. Hence, $|\mathcal{A}| \leq 5 \cdot 2 \cdot 1 = 10$. Therefore, we have solved the 1-color candidates problem deterministically. \square

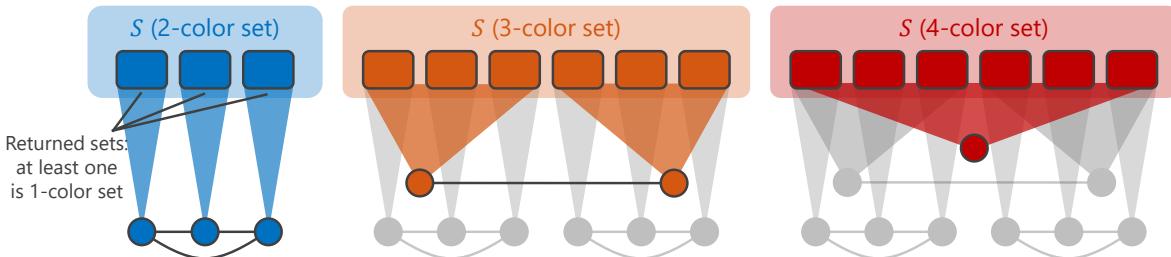


Figure 12: An example of returned sets when $d = 2, 1, 0$ in the order. The gray vertices are the vertices colored by level- $(d + 1)$ or higher subproblems.

Important Remarks. In the algorithm described in subsection 4.3, we use 1-COLOR-CAND(S) as a subroutine. Here, deletion of vertices in S may occur. Since the deletion is an invalid operation for the 1-color candidates problem, we need to “reset” the subroutine and start using new colors. Formally, if S changes from S_1 to S_2 ($\subseteq S_1$) by deletion, we finish the current subroutine of 1-COLOR-CAND, and start a new subroutine of 1-COLOR-CAND by setting $S = S_2, T = \emptyset$. In the new subroutine, we use completely new colors for coloring vertices coming to T (such a case may occur in 4. of **Implementation** in the next subsection).

4.3 The first step: No-dense case

In this section, we introduce an algorithm to solve the else-problem when the no-dense property holds. Our algorithm is described in [Algorithm 3](#) (No-DENSE-CASE).

Algorithm 3 No-DENSE-CASE

```

1:  $T_{FF} \leftarrow \emptyset$ 
2:  $\mathcal{R} \leftarrow []$ 
3:  $c \leftarrow 0$ 
4: for each arrival of  $v \in T_0$  do
5:   if for some  $i, j$ ,  $|N_{R_{i,j}}(v)| \geq \frac{1}{6400}n^{12/17}$  and 3-COLOR( $R_{i,j}$ ) is not aborted yet then
6:     color  $v$  with 3-COLOR( $R_{i,j}$ )
7:   else if  $|N_{T_{FF}}(v)| < n^{13/17}$  then
8:     color  $v$  with FIRSTFIT (among  $T_{FF}$ )
9:      $T_{FF} \leftarrow T_{FF} \cup \{v\}$ 
10:  else
11:     $T_{NG} \leftarrow \{v \in T_{FF} : |N_{\bigcup_{i,j} R_{i,j}}(v)| \geq \frac{1}{2}n^{14/17}\}$ 
12:    color  $v$  with 1-COLOR-CAND( $T_{FF} \setminus T_{NG}$ )
13:    if 1-COLOR-CAND( $T_{FF} \setminus T_{NG}$ ) returns a set  $\mathcal{A}$  then
14:      add  $\{N'_{S_0 \setminus (\bigcup_{i,j} R_{i,j})}(A) : A \in \mathcal{A}\}$  to  $\mathcal{R}$             $\triangleright N'_X(A)$ : the first  $\frac{1}{64}n^{15/17}$  elements of  $N_X(A)$ 
15:       $c \leftarrow c + 1$ 
16:    if  $c > n^{1/17}$  or  $|T_{FF}| > n^{16/17}$  then
17:       $T_{FF} \leftarrow \emptyset$  and reset FIRSTFIT
18:       $c \leftarrow 0$ 

```

Overview of No-DENSE-CASE. Below, we explain how No-DENSE-CASE works. Normally, to color vertices in T_0 online, we use FIRSTFIT (among T_0) with $n^{13/17}$ colors. However, some vertices in T_0 cannot be colored by FIRSTFIT. We instead color such a vertex $v \in T_0$ with 1-COLOR-CAND(T_{FF}), where $T_{FF} \subseteq T_0$ is the current set of vertices colored by FIRSTFIT ([Figure 13 left](#)). When 1-COLOR-CAND eventually returns a set of “candidate 1-color sets” \mathcal{A} , we initiate a new subroutine 3-COLOR($N_{S_0}(A)$) for each $A \in \mathcal{A}$. This is because, if A is a 1-color set, $N_{S_0}(A)$ is always a 3-color set in some 4-coloring of G , and we can also assume $|N_{S_0}(A)| \geq \frac{1}{64}n^{15/17}$ due to the no-dense property ([Figure 13 middle](#)).

Afterwards, we may color a newly arrived vertex by 3-COLOR($N_{S_0}(A)$) if there is a sufficient number of edges to $N_{S_0}(A)$, i.e., $\frac{1}{6400}n^{12/17}$ edges or more ([Figure 13 right](#)). Thus, the initiation of the new subroutines is for the sake of coloring the “future” vertices efficiently. Note that if a new vertex cannot be colored by either FIRSTFIT or 3-COLOR, we color it with 1-COLOR-CAND(T_{FF}) again.

Implementation. To implement this idea, when a vertex $v \in T_0$ arrives, we have again recorded the list of currently initiated 3-COLOR subroutines by a two-dimensional array $\mathcal{R} = [\mathcal{R}_1, \mathcal{R}_2, \dots] = [[R_{1,1}, \dots, R_{1,|\mathcal{R}_1|}], [R_{2,1}, \dots, R_{2,|\mathcal{R}_2|}], \dots]$, where each $R_{i,j}$ denotes that a subroutine 3-COLOR($R_{i,j}$) was initiated. \mathcal{R}_i denotes

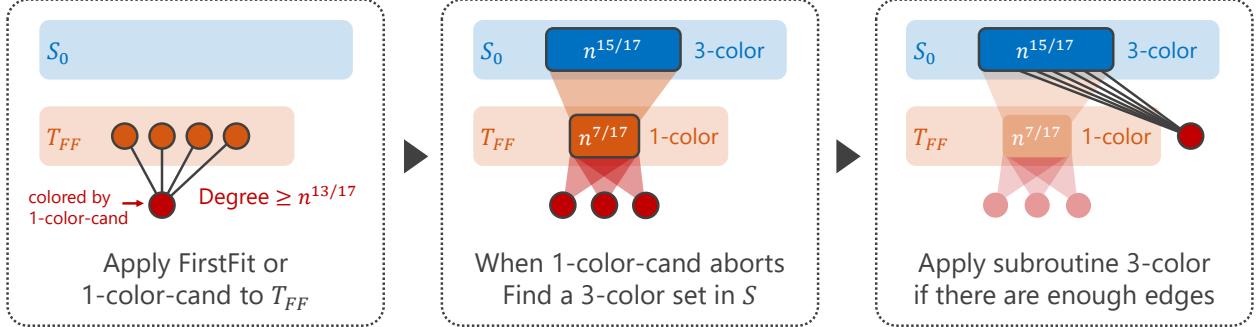


Figure 13: The sketch of the algorithm in No-DENSE-CASE.

the set of subroutines initiated at the same time. We also defined a counter c which is initially zero. Then, for each arrival of $v \in T_0$, we run the following procedure:

1. If there exists some i, j where $|N_{R_{i,j}}(v)| \geq \frac{1}{6400}n^{12/17}$ and $\text{3-COLOR}(R_{i,j})$ is not aborted, we color v with $\text{3-COLOR}(R_{i,j})$, as this meets the degree condition for the 3-color set problem (lines 5–6).
2. Otherwise, if v can be colored by FIRSTFIT, we use FIRSTFIT and add v to T_{FF} (lines 7–9).
3. Otherwise, color v with $\text{1-COLOR-CAND}(T_{\text{FF}} \setminus T_{\text{NG}})$ where

$$T_{\text{NG}} = \left\{ v \in T_{\text{FF}} : |N_{\bigcup_{i,j} R_{i,j}}(v)| \geq \frac{1}{2}n^{14/17} \right\}$$

The reason we do not simply apply $\text{1-COLOR-CAND}(T_{\text{FF}})$ is to avoid the situation of “finding a 3-color set that is already known”, e.g., $R_{1,1} = R_{2,1}$ occurs (lines 11–12).

4. When $\text{1-COLOR-CAND}(T_{\text{FF}} \setminus T_{\text{NG}})$ aborts, we receive a set of “candidate 1-color sets” \mathcal{A} . We use this obtained information to benefit the “future” vertices. Specifically, for each $A \in \mathcal{A}$, we initiate a subroutine $\text{3-COLOR}(N'_{S_0 \setminus (\bigcup_{i,j} R_{i,j})}(A))$ where $N'_X(A)$ is the first $\frac{1}{64}n^{15/17}$ elements of $N_X(A)$.

Then, we increase c by one, and add some elements to \mathcal{R} , for the sake of coloring “future” vertices efficiently (lines 13–15). Note that

$$|N_{S_0 \setminus (\bigcup_{i,j} R_{i,j})}(A)| \geq \frac{1}{64}n^{15/17}$$

always holds at line 14 due to the no-dense property and by $|A| \geq \frac{1}{64}n^{7/17}$. We can apply the no-dense property because $|N_{S_0 \setminus (\bigcup_{i,j} R_{i,j})}(v)| \geq \frac{1}{2}n^{14/17}$ holds for each $v \in T_{\text{FF}} \setminus T_{\text{NG}}$ (due to line 11).

5. When we reach $c > n^{1/17}$ or $|T_{\text{FF}}| > n^{16/17}$, we finally reset FIRSTFIT and counter c (lines 16–18). This is because if we continue, it violates the input condition $|S| \leq n^{16/17}$ in 1-COLOR-CAND , or $|T_{\text{NG}}|$ may become too large ([Lemma 4.4](#)). After the reset, we must use different colors in FIRSTFIT at line 8.

4.4 Correctness & Analysis

Next, we discuss the correctness and the number of colors used in No-DENSE-CASE. The goal is to show [Lemma 4.7](#) (correctness of our algorithm). We first show some required lemmas.

Lemma 4.3. $|\mathcal{R}| \leq 64n^{2/17}$ holds.

Proof. For each i , there is at least one t that $\text{3-COLOR}(R_{i,t})$ is not aborted. This is because at least one of $R_{i,t}$ is a 3-color set in some 4-coloring of G . For such t , let $R'_i = R_{i,t}$. Then, by line 14, $|R'_i \cap R'_j| = 0$ ($i \neq j$). Since $|R'_i| = \frac{1}{64}n^{15/17}$ for each i , **Lemma 4.3** holds. \square

Lemma 4.4. *If $c \leq n^{1/17}$, then $|T_{\text{NG}}| < n^{8/17}$ under the no-dense property.*

Proof. Let \mathcal{R}' be the elements that were added to \mathcal{R} before the last reset of FIRSTFIT. Since (i) $|\mathcal{R}'| = 64n^{2/17}$ (**Lemma 4.3**), (ii) $|\mathcal{R}'_i| \leq 10$ (**Lemma 4.1**), and (iii) $|N_{R'_{i,j}}(v)| < \frac{1}{6400}n^{12/17}$ for each $v \in T_{\text{FF}}$ (by line 5),

$$\left| N_{\bigcup_{i,j} R'_{i,j}}(v) \right| \leq \frac{1}{6400}n^{12/17} \cdot 64n^{2/17} \cdot 10 = \frac{1}{10}n^{14/17}$$

holds for each $v \in T_{\text{FF}}$. Therefore, by line 11,

$$\left| N_{(\bigcup_{i,j} R_{i,j}) \setminus (\bigcup_{i,j} R'_{i,j})}(v) \right| \geq \frac{1}{2}n^{14/17} - \frac{1}{10}n^{14/17} = \frac{2}{5}n^{14/17}$$

holds for each $v \in T_{\text{NG}}$. Now, consider applying the no-dense property where $T_D = T_{\text{NG}}$ and $S_D = (\bigcup_{i,j} R_{i,j}) \setminus (\bigcup_{i,j} R'_{i,j})$.¹³ By $c \leq n^{1/17}$, $|\mathcal{R}_i| \leq 10$, and $|R_{i,j}| = \frac{1}{64}n^{15/17}$,

$$|S_D| = \left| \left(\bigcup_{i,j} R_{i,j} \right) \setminus \left(\bigcup_{i,j} R'_{i,j} \right) \right| \leq n^{1/17} \cdot 10 \cdot \frac{1}{64}n^{15/17} = \frac{5}{32}n^{16/17}$$

holds, and if $|T_{\text{NG}}| \geq n^{8/17}$, it contradicts the no-dense property. Therefore, $|T_{\text{NG}}| < n^{8/17}$. \square

Lemma 4.5. *For each call of 1-COLOR-CAND($T_{\text{FF}} \setminus T_{\text{NG}}$), $|N_{T_{\text{FF}} \setminus T_{\text{NG}}}(v)| \geq \frac{1}{2}n^{13/17}$ holds, which satisfies the degree constraints of the 1-color candidate problem.*

Proof. By $|N_{T_{\text{FF}}}(v)| \geq n^{13/17}$ (line 7) and $|T_{\text{NG}}| < n^{8/17}$ (**Lemma 4.4**), **Lemma 4.5** follows. \square

Lemma 4.6. *No-DENSE-CASE uses $\tilde{O}(n^{14/17})$ colors under no-dense property.*

Proof. Let c_1, c_2, c_3 be the number of colors used in 3-COLOR (line 6), FIRSTFIT (line 8), and 1-COLOR-CAND (line 12), respectively. Then, the total number of colors used is $c_1 + c_2 + c_3$, and each value is as follows:

- c_1 : By **Lemma 4.1**, **Lemma 4.3** ($|\mathcal{R}| = O(n^{2/17})$) and therefore 3-COLOR is initiated for $O(n^{2/17})$ times, and the fact that the sum of $|T|$ for each 3-COLOR($R_{i,j}$) is at most n , $c_1 = \tilde{O}(n^{14/17})$.
- c_2 : By **Lemma 4.3** and line 16, the reset on FIRSTFIT only occurs $O(n^{1/17})$ times. Since each run of FIRSTFIT uses $n^{13/17}$ colors, $c_2 = O(n^{14/17})$.
- By **Lemma 4.2**, we only use $O(n^{12/17})$ colors for each procedure of 1-COLOR-CAND. In addition, vertices are deleted from the set S of 1-COLOR-CAND (i.e., $T_{\text{FF}} \setminus T_{\text{NG}}$) only when \mathcal{R} changes, which occurs $|\mathcal{R}| = O(n^{2/17})$ times. Therefore, $c_3 = O(n^{14/17})$.

By $c_1 + c_2 + c_3 = \tilde{O}(n^{14/17})$, the algorithm No-DENSE-CASE uses $\tilde{O}(n^{14/17})$ colors. \square

Lemma 4.7. *No-DENSE-CASE solves the else-problem with $\tilde{O}(n^{14/17})$ colors under no-dense property.*

Proof. To prove the correctness, we must show the following two: (1) the assumption of 1-COLOR-CAND (e.g. $|N_S(v)| \geq \frac{1}{2}n^{13/17}$ for each $v \in T$) is satisfied, and (2) the number of colors used is as required. (1) follows from **Lemma 4.5**, while (2) follows from **Lemma 4.6**. Therefore, **Lemma 4.7** holds. \square

¹³Note that since $\frac{2}{5}n^{14/17} \geq \frac{1}{3}n^{14/17}$, we can apply the no-dense property.

4.5 The second step: Avoiding dense case

The last step is to reduce the else-problem to “the else-problem with the no-dense property”. First, consider using FIRSTFIT with $2n^{14/17}$ colors (instead of the previous $n^{14/17}$). For each vertex that cannot be colored by FIRSTFIT (i.e., vertices in T_0), we apply the following [Algorithm 4](#) (BRANCH).

Algorithm 4 BRANCH

```

1: for each arrival of  $v \in T_0$  do
2:   select either  $T_A$  or  $T_B$  “wisely”
3:   if  $T_A$  is selected then
4:     color  $v$  with No-DENSE-CASE
5:   else
6:     color  $v$  with a “special procedure”

```

In this BRANCH procedure, the no-dense property must hold between S_0 and T_A , and there must exist a deterministic online algorithm that colors the vertices in T_B (at line 6) using $\tilde{O}(n^{14/17})$ colors. Formally, we must solve the following problem:

Division problem. Consider the else-problem in which $|N_{S_0}(v)| \geq 2n^{14/17}$ for each arrival of $v \in T_0$. Our task is to divide T_0 into T_A and T_B online, that is, once we receive a vertex $v \in T_0$, we must immediately decide whether v should belong to T_A or T_B (see [Figure 14](#)). If we select T_B , we must color v . Our objective is that the no-dense property holds between S_0 and T_A , and that the vertices in T_B are colored with $\tilde{O}(n^{14/17})$ colors. To achieve the no-dense property, it is allowed to delete some edges between S_0 and T_0 , but $|N_{S_0}(v)| \geq n^{14/17}$ for each $v \in T_0$ must still hold after deletion (otherwise, this would violate the assumption of the else-problem in [subsection 4.1](#)).

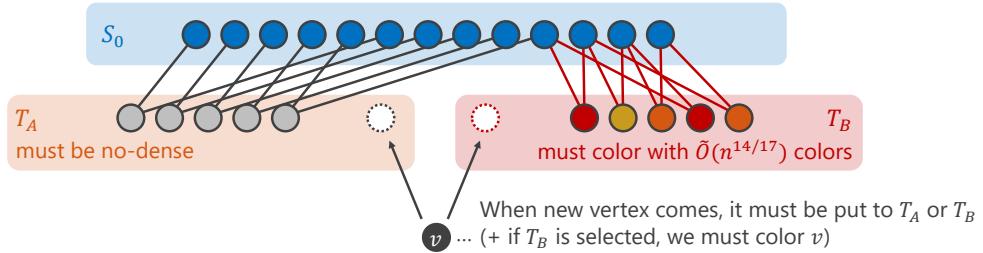


Figure 14: A sketch of the division problem.

If the division problem is solved, we obtain a deterministic online algorithm that uses $\tilde{O}(n^{14/17})$ colors for 4-colorable graphs (by the BRANCH procedure). Now, to solve the division problem, we first show an important definition and lemma.

Definition 4.8. We say that two vertices $u_1, u_2 \in T_0$ are β -common if $|N_{S_0}(u_1) \cap N_{S_0}(u_2)| \geq n^\beta$. We say T_A is (α, β) -free if there is no vertex set $\{t_1, t_2, \dots, t_{n^\alpha+1}\} \subseteq T_A$ where $t_1, t_2, \dots, t_{n^\alpha+1}$ arrive in this order, and for all i , t_i and $t_{n^\alpha+1}$ are β -common.

Lemma 4.9. Suppose that the no-dense property does not hold between S_0 and T_A . Then, T_A is not (α_i, β_i) -free for at least one i , where $\Delta = \log n$, $K = \frac{2\log\log n}{\log n}$, and:

$$(\alpha_i, \beta_i) = \left(\frac{8}{17} - \left(\frac{2}{17} + K \right) \cdot \frac{\Delta - i}{\Delta}, \frac{14}{17} - \left(\frac{2}{17} + K \right) \cdot \frac{i}{\Delta} \right) \quad (i = 0, 1, \dots, \Delta)$$

Proof. Since the no-dense property does not hold, there exist subsets $S_D \subseteq S_0, T_D \subseteq T_A$ such that $|T_D| = n^\alpha$ and $|S_D| \leq n^{\alpha+8/17}$ hold for some $\alpha \leq \frac{8}{17}$, and $|N_{S_D}(t)| \geq \frac{1}{3}n^{14/17}$ for each $t \in T_D$. It suffices to prove the case that $|N_{S_D}(t)| = \frac{1}{3}n^{14/17}$ for each $t \in T_D$, because this is the “hardest” case.¹⁴

Suppose for a contradiction that T_A is (α_i, β_i) -free for all i .

We consider the sum of $c(t_1, t_2) = |N_{S_D}(t_1) \cap N_{S_D}(t_2)|$ over all $\{t_1, t_2\} \subseteq T_D$ (with t_1 arriving earlier than t_2) in two ways. First, the sum is equal to the number of tuples (t_1, t_2, s) ($\{t_1, t_2\} \in T_D, s \in S_D$ that s is adjacent to both t_1 and t_2). For each s , there are $\frac{1}{2}|N_{T_D}(s)|(|N_{T_D}(s)| - 1)$ ways to choose $\{t_1, t_2\}$ to meet the condition. Hence, the following equation holds:

$$\sum_{\{t_1, t_2\} \subseteq T_D} c(t_1, t_2) = \sum_{s \in S_D} \frac{1}{2}|N_{T_D}(s)|(|N_{T_D}(s)| - 1).$$

There are $\frac{1}{3}n^{a+14/17}$ edges between S_D and T_D . Therefore, $\sum_{s \in S_D} |N_{T_D}(s)| = \frac{1}{3}n^{a+14/17}$. Combining with $|S_D| \leq n^{\alpha+8/17}$, we obtain $\sum_{s \in S_D} |N_{T_D}(s)|^2 \geq \frac{1}{9}n^{a+20/17}$.¹⁵ Therefore,

$$\sum_{\{t_1, t_2\} \subseteq T_D} c(t_1, t_2) = \Omega(n^{a+20/17}).$$

Second, given that T_A is (α_i, β_i) -free, for each t_2 , the number of t_1 's that $c(t_1, t_2) \geq n^{\beta_i}$ is at most n^{α_i} . It means that the number of $\{t_1, t_2\}$'s that $c(t_1, t_2) \geq n^{\beta_i}$ is at most $n^{\alpha+\alpha_i}$. Let $[c_1, c_2, \dots, c_m]$ be a list of $c(t_1, t_2)$'s for all $\{t_1, t_2\}$'s, sorted in descending order. Then, $c_j \leq n^{\beta_i}$ must hold for $j = n^{\alpha+\alpha_i} + 1, \dots, n^{\alpha+\alpha_{i+1}}$. For $j = 1, 2, \dots, n^{\alpha+\alpha_0}$, $c_j \leq n^{\beta_0}$ must hold, by the assumption $|N_{S_D}(t)| = \frac{1}{3}n^{14/17}$ ($\leq n^{\beta_0}$). Therefore, the following holds (note that $m \leq n^{\alpha+\alpha_\Delta}$, and $n^{\alpha_i+\beta_i} = n^{a+20/17-K} = O(n^{a+20/17}/(\log n)^2)$ for each i):

$$\begin{aligned} \sum_{i=1}^m c_i &\leq n^{\alpha+\alpha_0} \cdot n^{\beta_0} + \sum_{i=0}^{\Delta-1} (n^{\alpha+\alpha_{i+1}} - n^{\alpha+\alpha_i}) \cdot n^{\beta_i} \\ &= n^{a+20/17-K} + \sum_{i=0}^{\Delta-1} (n^{(2/17+K)/\Delta} - 1) \cdot n^{a+20/17-K} \\ &= O(n^{a+20/17}/\log n). \end{aligned}$$

We obtain that the sum of $c(t_1, t_2)$'s is both $\Omega(n^{a+20/17})$ and $O(n^{a+20/17}/\log n)$, which is a contradiction. Therefore, [Lemma 4.9](#) follows. \square

By [Lemma 4.9](#), the division problem is solved if T_A is (α_i, β_i) -free for all $i = 0, 1, \dots, \Delta$. Now, we aim to achieve (α_i, β_i) -free, instead of no-dense property.

4.6 The Common & Simplify technique

Since handling multiple pairs at once is quite challenging, we first consider an algorithm that achieves (α, β) -free for a single pair (α, β) . Our algorithm is described in [Algorithm 5](#) (DIVISION(α, β)).

Overview of DIVISION(α, β). Below, we explain how the algorithm works. The important intuition is:

Using the vertices in T_0 that have a lot of common neighbors, we aim to obtain a large n^β -vertex 2-color set. Indeed, if two β -common vertices $u_1, u_2 \in T_0$ have different colors in some 4-coloring of G , $N_{S_0}(u_1) \cap N_{S_0}(u_2)$ is a n^β -vertex 2-color set in some 4-coloring of G .

¹⁴This is because T_A cannot newly become non- (α_i, β_i) -free by deleting edges.

¹⁵For any real numbers x_1, \dots, x_k , the inequality $x_1^2 + \dots + x_k^2 \geq \frac{1}{k}(x_1 + \dots + x_k)^2$ holds.

Algorithm 5 DIVISION(α, β)

```

1:  $\mathcal{F}, \mathcal{I}, C \leftarrow []$ 
2:  $\mathcal{R} \leftarrow []$ 
3: for each arrival of  $v \in T_0$  do
4:   if for some  $i, j$ ,  $|N_{R_{i,j}}(v)| \geq \frac{1}{8\log n}n^{\beta-3/17}$  and 2-COLOR( $R_{i,j}$ ) is not aborted yet then
5:     color  $v$  with 2-COLOR( $R_{i,j}$ )
6:   else
7:     delete all edges from  $v$  to vertices in  $\bigcup_{i,j} R_{i,j}$ 
8:     if  $v$  and  $F_{i,j}$  are  $\beta$ -common for some  $i, j$  then
9:        $I_i \leftarrow I_i \cup \{v\}$ 
10:      color  $v$  with the special color for  $I_i$ 
11:      if some  $u_1, u_2 \in I_i$  are adjacent then
12:         $w_1 \in \{w \in F_i : u_1 \text{ and } w \text{ are } \beta\text{-common}\}$ 
13:         $w_2 \in \{w \in F_i : u_2 \text{ and } w \text{ are } \beta\text{-common}\}$ 
14:        add  $\{N_{S_0}(u_1) \cap N_{S_0}(w_1), N_{S_0}(w_1) \cap N_{S_0}(c_i), N_{S_0}(c_i) \cap N_{S_0}(w_2), N_{S_0}(w_2) \cap N_{S_0}(u_2)\}$  to
             $\mathcal{R}$  ▷ In this operation, we cap the size of each added set in  $\mathcal{R}$  to  $n^\beta$ 
15:       $\mathcal{F} = \mathcal{I} = C = []$ 
16:    else
17:       $F' \leftarrow \{u \in T_A : u \text{ and } v \text{ are } \beta\text{-common}\}$ 
18:      if  $|F'| \geq n^\alpha$  then
19:        add  $F'$  to array  $\mathcal{F}$ 
20:        add  $\emptyset$  to array  $\mathcal{I}$ 
21:        add  $v$  to array  $C$ 
22:        color  $v$  with a special color
23:      else
24:        put  $v$  in  $T_A$ 

```

In this algorithm, we normally put a new vertex $v \in T_0$ to T_A , but sometimes, T_A becomes not (α, β) -free without selecting T_B . In such a case, we select T_B instead and create a *dense group* F where

$$F = \{u \in T_A : u \text{ and } v \text{ are } \beta\text{-common}\}$$

which is $|F| \geq n^\alpha$. Note that we color v with a special unique color (Figure 15 left). Afterwards, if a newly arrived vertex $v' \in T_0$ is β -common with at least one vertex $u \in T_A$ that is already in a (previously created) dense group F_i , we color v' with “the special color prepared for F_i ” (and put v' to T_B). At this point, the structure between T_A and T_B and the color used in T_B are as shown in Figure 15 middle. Note that since the dense groups are disjoint, the number of dense groups that exist simultaneously is at most $n^{1-\alpha}$.

When coloring v' with “the special color for F_i ”, a problem occurs when two vertices that share the same special color are adjacent (Figure 15 right). In this case, we need to color v' with a new color just for this vertex. However, we can obtain a 2-color set in S_0 instead. Assume that $(u_1, u_2) \in T_B$ are adjacent, u_1 is β -common with $w_1 \in F_i$, u_2 is β -common with $w_2 \in F_i$, and each vertex in F_i is β -common with x . Then, at least one of

$$\{N_{S_0}(u_1) \cap N_{S_0}(w_1), N_{S_0}(w_1) \cap N_{S_0}(x), N_{S_0}(x) \cap N_{S_0}(w_2), N_{S_0}(w_2) \cap N_{S_0}(u_2)\}$$

must be an n^β -vertex 2-color set. This is because u_1 and u_2 always have different colors in any 4-coloring of G , and therefore at least one of $(u_1, w_1), (w_1, x), (x, w_2), (w_2, u_2)$ has different colors in some 4-coloring of G (see Lemma 4.10). Afterwards, like subsection 4.3, we can reduce to the 2-color set problem. Note that this procedure is for the sake of coloring “future” vertices efficiently.

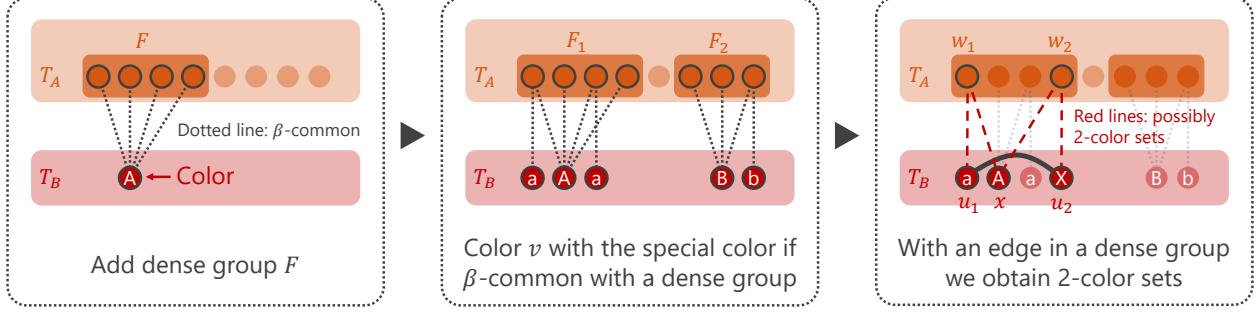


Figure 15: The sketch of the algorithm between T_A and T_B . The dotted line indicates that two vertices are β -common (it does not always mean they are adjacent). Colors A, a, B, b, and X are different.

Implementation. To implement this idea, when a vertex $v \in T_0$ arrives, we have recorded three arrays: the list of dense groups as an array of sets $\mathcal{F} = [F_1, F_2, \dots]$, another array $\mathcal{I} = [I_1, I_2, \dots]$ where I_i is the set of vertices colored by the special color for F_i , and another array $C = [c_1, c_2, \dots]$ where c_i is β -common to each vertex in F_i (i.e., the cause of creating F_i). We also recorded the list of initiated 2-COLOR subroutines in array \mathcal{R} (the same details as [Algorithm 3](#)). Then, upon the arrival of $v \in T_0$, we run the following procedure:

1. If there exist i, j such that $|N_{R_{i,j}}(v)| \geq \frac{1}{8\log n}n^{\beta-3/17}$ and $\text{2-COLOR}(R_{i,j})$ is not aborted, we color v with $\text{2-COLOR}(R_{i,j})$, as it meets the condition for T in the 2-color set problem (lines 4–5).
2. Otherwise, to ensure the no-dense property between S_0 and T_A , we delete the edges between v and $\bigcup_{i,j} R_{i,j}$ (line 7). Then,
 - If v and $x \in F_i$ are β -common, we color v with the special color for F_i and update I_i (lines 9–10). If there is an edge inside I_i , we obtain four candidates for 2-color sets (lines 11–14), update \mathcal{R} , and then reset all dense groups (line 15). Note that after coloring v , we update \mathcal{R} , but this is for the sake of coloring “future” vertices efficiently.
 - Otherwise, put v in T_A if it does not violate (α, β) -free (line 24). If not, select T_B and create a dense group (lines 19–22).

4.7 Correctness & Analysis

Next, we discuss the correctness of and the number of colors used in $\text{DIVISION}(\alpha, \beta)$. The goal is to show that the algorithm correctly solves the division problem ([Lemma 4.15](#)). We first show some required lemmas.

Lemma 4.10. At line 14 of $\text{DIVISION}(\alpha, \beta)$, at least one of $N_{S_0}(u_1) \cap N_{S_0}(w_1)$, $N_{S_0}(w_1) \cap N_{S_0}(c_i)$, $N_{S_0}(c_i) \cap N_{S_0}(w_2)$, $N_{S_0}(w_2) \cap N_{S_0}(u_2)$ is a 2-color set in some 4-coloring of G .

Proof. Let $p_1, p_2, p_3, p_4, p_5 \in \{1, 2, 3, 4\}$ be the colors of u_1, w_1, c_i, w_2, u_2 in a 4-coloring of G . Since $u_1 u_2 \in E(G)$ and hence $p_1 \neq p_5$, there exists some i with $p_i \neq p_{i+1}$. If $p_1 \neq p_2$, then $N_{S_0}(u_1) \cap N_{S_0}(w_1)$ is a 2-color set. If $p_2 \neq p_3$, then $N_{S_0}(w_1) \cap N_{S_0}(c_i)$ is a 2-color set. If $p_3 \neq p_4$, then $N_{S_0}(c_i) \cap N_{S_0}(w_2)$ is a 2-color set. If $p_4 \neq p_5$, then $N_{S_0}(w_2) \cap N_{S_0}(u_2)$ is a 2-color set. \square

Lemma 4.11. $|\mathcal{R}| \leq n^{1-\beta}$.

Proof. By [Lemma 4.10](#), for each i , there must be at least one t that $\text{2-COLOR}(R_{i,t})$ is not aborted yet. For such t , let $R'_i = R_{i,t}$. Then, $|R'_i \cap R'_j| = 0$ ($i \neq j$) by line 7 in $\text{DIVISION}(\alpha, \beta)$. Since $|R'_i| = n^\beta$ for each i , [Lemma 4.11](#) holds. \square

Lemma 4.12. In line 7 of DIVISION(α, β), at most $\frac{1}{2\log n}n^{14/17}$ edges are deleted for each $v \in T_0$.

Proof. By $|\mathcal{R}| \leq n^{1-\beta}$ (Lemma 4.11), $|\mathcal{R}_i| = 4$ for each i , and the threshold in line 4 is $\frac{1}{8\log n}n^{\beta-3/17}$,

$$\frac{1}{8\log n}n^{\beta-3/17} \cdot |\mathcal{R}| \cdot |\mathcal{R}_i| \leq \frac{1}{2\log n}n^{14/17}$$

edges are deleted from each $N_S(v)$, and therefore Lemma 4.12 holds. \square

Lemma 4.13. DIVISION(α, β) uses $\tilde{O}(n^{14/17})$ colors, if $\alpha + \beta = \frac{20}{17} - K$ and $\beta \geq \frac{12}{17} - K$.

Proof. Let c_1, c_2, c_3 be the number of colors used in lines 5, 10, and 22 in DIVISION(α, β), respectively. Then, the number of colors used in DIVISION(α, β) is $c_1 + c_2 + c_3$. Each value is as follows:

- c_1 : By $|\mathcal{R}| = \tilde{O}(n^{5/17})$ (Lemma 4.11 and the value of β) and Lemma 4.1, $c_1 = \tilde{O}(n^{14/17})$.
- c_2 : Since $|F_i \cap F_j| = 0$ ($i \neq j$) by lines 8 and 17, and $|F_i| \geq n^\alpha$ for each i , we have $|\mathcal{F}| = |\mathcal{I}| = O(n^{1-\alpha})$. Since only two colors are used in each I_i , and the resetting of \mathcal{I} occurs $O(n^{1-\beta})$ times (Lemma 4.11), $c_2 = O(n^{2-\alpha-\beta}) = \tilde{O}(n^{14/17})$.
- c_3 : From the same reason as for c_2 , we have $c_3 = \tilde{O}(n^{14/17})$.

As $c_1 + c_2 + c_3 = \tilde{O}(n^{14/17})$, the algorithm DIVISION(α, β) uses only $\tilde{O}(n^{14/17})$ colors. \square

Lemma 4.14. The set T_A generated by DIVISION(α, β) is (α, β) -free.

Proof. It follows directly from lines 17–18 of DIVISION(α, β). \square

At this point, we have successfully obtained an algorithm that achieves (α, β) -free in $\tilde{O}(n^{14/17})$ colors for a single pair (α, β) . Now, we finally show the solution to the division problem.

Lemma 4.15. There is an algorithm that solves the division problem.

Proof. Consider an algorithm that applies lines 4–22 of DIVISION(α, β) for each pair (α_i, β_i) for each $i \in \{0, 1, \dots, \Delta\}$. We use independent arrays $\mathcal{F}, \mathcal{I}, C$ for each pair (α_i, β_i) , and for each arrival of $v \in T$, we repeat lines 4–22 in DIVISION(α, β) for $i \in \{0, 1, \dots, \Delta\}$. If all the results of the “if-conditions” are false, we put v in T_A .

To prove the correctness, we must prove the following three: (1) the no-dense property between S_0 and T_A , (2) the number of colors used for T_B , and (3) $|N_{S_0}(v)| \geq n^{14/17}$ holds for each $v \in T_0$ even after the edge deletion. For (1), by Lemma 4.14, T_A is obviously (α_i, β_i) -free for each i , and by Lemma 4.9, the no-dense property holds between S_0 and T_A . For (2), by Lemma 4.13 and $\Delta = \log n$, we only use $\tilde{O}(n^{14/17}) \cdot \Delta = \tilde{O}(n^{14/17})$ colors. For (3), by Lemma 4.12 and $\Delta = \log n$, at most $\frac{\Delta+1}{2\log n} \cdot n^{14/17} \leq n^{14/17}$ edges in $N_{S_0}(v)$ are deleted in total for each $v \in T_0$, which means that $n^{14/17}$ or more edges remain. Therefore, the algorithm above solves the division problem. \square

4.8 Conclusion

Finally, by Lemma 4.15, Lemma 4.7, and subsection 4.1, we obtain the following theorem.

Theorem 4.16. There is a deterministic online algorithm that colors any 4-colorable graph with $\tilde{O}(n^{14/17})$ colors.

In addition, combining with the results in section 3, we can also improve the results for $k \geq 5$. We show the following theorem:

Theorem 4.17. *There exists a deterministic online algorithm that colors any k -colorable graph with $\tilde{O}(n^{1-6/(3k(k-1)-2)})$ colors, for any $k \geq 4$.*

Proof. The case for $k = 4$ is proved in [Theorem 4.16](#). For $k \geq 5$, we prove that the algorithm to solve the level- d subproblem+ can achieve $\tilde{O}(\max(|T|, n^{1-d\epsilon})/n^\epsilon)$ colors, where $\epsilon = \frac{6}{3k(k-1)-2}$. We follow the proof in [subsection 3.6](#). For the $d = k - 1$ case, it suffices to prove the following inequality:

$$|T|^{1-\frac{6}{3(k-1)(k-2)-2}} \leq \frac{\max(|T|, n^{1-(k-1)\epsilon})}{n^\epsilon}$$

and it holds when $\epsilon \leq \frac{6}{3k(k-1)-2}$. For the $d \leq k - 2$ case, it suffices to prove the following inequality, for $|T| \geq n^{1-d\epsilon}$ and $|D| = \frac{|T|}{n^\epsilon}$:

$$|D|^{1-\frac{2}{d(d-1)+2}} \leq \frac{|D|}{n^\epsilon}$$

and it holds when $\epsilon \leq \frac{2}{d(d+1)+4}$, which is above $\frac{6}{3k(k-1)-2}$ even with $d = k - 2$, for all $k \geq 5$. Therefore, [Theorem 4.17](#) holds. \square

By [Theorem 4.17](#), the results for $k \geq 5$ are slightly improved over [Theorem 3.12](#). Our best results, compared with Kierstead's algorithm [18], are shown in [Table 2](#).

k	3	4	5	6	7
Previous Results	$\tilde{O}(n^{0.6667})$	$\tilde{O}(n^{0.8334})$	$\tilde{O}(n^{0.9917})$	$\tilde{O}(n^{0.9987})$	$\tilde{O}(n^{0.9999})$
Our Results	—	$\tilde{O}(n^{0.8236})$	$\tilde{O}(n^{0.8966})$	$\tilde{O}(n^{0.9319})$	$\tilde{O}(n^{0.9517})$

Table 2: Comparison between the previous best results [18] and our results, for $k \leq 7$.

5 Randomized algorithm for bipartite graphs

In this section, we show an efficient randomized online coloring algorithm for bipartite graphs, against an oblivious adversary (the case where the graph is determined before the coloring begins).

5.1 The algorithm by Lovász, Saks, and Trotter

Firstly, we review a previously-known deterministic online coloring algorithm for bipartite graphs by Lovász, Saks, and Trotter (1989) [21]. This algorithm uses $2 \log(n + 1)$ colors in the worst case, and is an optimal deterministic algorithm up to a constant number of colors [11]. The idea of this algorithm is to always try to achieve the “bipartite coloring”, and once it becomes impossible, we start using a new color. Note that the hard case is when there are many components of bipartite graphs, and some of them start merging when a new vertex v arrives. In this case, we may need a new color for v .

Formally, we assign a *level* to each connected component, so that a level- ℓ component uses colors $1, 2, \dots, 2\ell$. The meaning of the level is how many times the algorithm fails to achieve the “bipartite coloring” and is forced to use a new color. Therefore, we would like to estimate the maximum level.

We call a level- ℓ component *matched* if the colors $2\ell - 1$ and 2ℓ are based on the “correct” 2-coloring; that is, distances of any two vertices with color $2\ell - 1$ are even, distances of any two vertices with color 2ℓ are even, and distances of any vertex with color $2\ell - 1$ and any vertex with color 2ℓ are odd. At any time, every component is supposed to be matched. So, when a new vertex v arrives and some component becomes no longer matched, we increase the component's level by 1 and use a new color for vertex v . Otherwise, we color v with either color $2\ell - 1$ or 2ℓ , whichever is matched. Examples are shown in [Figure 16](#).

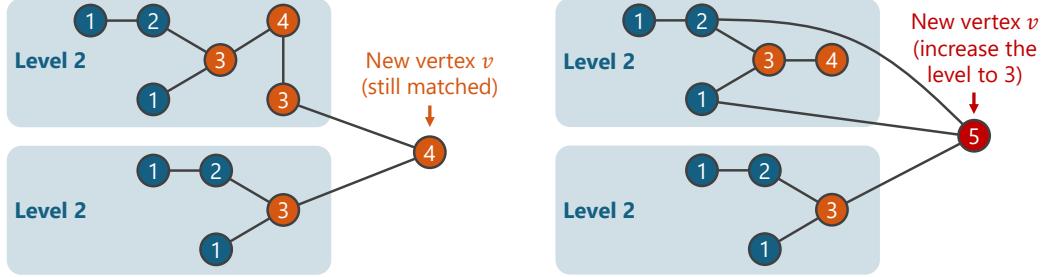


Figure 16: Two examples of the algorithm by Lovász, Saks, and Trotter [21]. Left figure shows an example when the component is still matched, and the right figure shows an example when we must increase the level from 2 to 3. The number inside each vertex is the color used.

Lovász, Saks, and Trotter's algorithm is described in [Algorithm 6](#) (LST89). We suppose that $V = \{v_1, \dots, v_n\}$, arriving in the order v_1, \dots, v_n . When vertex v_i arrives, the resulting component including v_i is denoted as C_i , and its level, after coloring v_i , is denoted as ℓ_i .

Algorithm 6 LST89($G = (V, E)$)

```

1:  $S \leftarrow \emptyset$                                 ▷ the set of indices of connected components of the current graph
2: for  $i = 1, \dots, n$  do
3:    $x_{i,1}, \dots, x_{i,k_i} \leftarrow \{j \in S : \exists u \in C_j, uv_i \in E\}$     ▷ the indices of components that are adjacent to  $v_i$ 
4:    $C_i \leftarrow C_{x_{i,1}} \cup \dots \cup C_{x_{i,k_i}} \cup \{v_i\}$ 
5:    $S \leftarrow (S \setminus \{x_{i,1}, \dots, x_{i,k_i}\}) \cup \{i\}$ 
6:   if  $k_i = 0$  then
7:     Color  $v_i$  by color 1
8:      $\ell_i \leftarrow 1$ 
9:   else
10:     $\ell_i^* \leftarrow \max(\ell_{x_{i,1}}, \dots, \ell_{x_{i,k_i}})$ 
11:    if  $C_i$  is matched (as a level- $\ell_i^*$  component) then
12:      Color  $v_i$  by either color  $2\ell_i^* - 1$  or  $2\ell_i^*$  so that  $C_i$  remains matched
13:       $\ell_i \leftarrow \ell_i^*$ 
14:    else
15:      Color  $v_i$  by color  $2\ell_i^* + 1$ 
16:       $\ell_i \leftarrow \ell_i^* + 1$ 

```

Theorem 5.1 ([21]). [Algorithm 6](#) (LST89) uses at most $2 \log(n+1)$ colors for any graph.

Proof. Let a_ℓ be the minimum n that level- ℓ components can appear. For $\ell \geq 2$, a level- ℓ component can be formed only when two or more level- $(\ell-1)$ components are merged into one component. Therefore, $a_\ell \geq 2a_{\ell-1} + 1$. Then, $a_\ell \geq 2^\ell - 1$ follows, so the algorithm uses at most $2 \log(n+1)$ colors. \square

5.2 The randomized algorithm and its probabilistic model

In this subsection, we present a randomized algorithm for online coloring of bipartite graphs. This algorithm is essentially the randomized version of LST89, which makes the following modifications to [Algorithm 6](#):

- Change line 7 to “Color v_i by color 1 or 2 with probability $\frac{1}{2}$ each”

- Change line 15 to “Color v_i by color $2\ell_i^* + 1$ or $2\ell_i^* + 2$ with probability $\frac{1}{2}$ each”

The new algorithm is denoted by **RANDOMIZEDLST**. The number of colors will be improved from LST89, because the adversarial case, which requires a new color every time, can be avoided in expectation. The goal of this section is to prove that **RANDOMIZEDLST** uses at most $1.034 \log n + O(1)$ colors in expectation.

Probabilistic model of the performance. We analyze the performance of **RANDOMIZEDLST** using a rooted forest T . When a vertex v_i arrives, the component C_i is formed by merging v_i and some existing connected components, say $C_{x_{i,1}}, \dots, C_{x_{i,k_i}}$. We represent in T this relation of how the components are merged. This is defined to be an n -vertex forest where the vertices are labeled $1, \dots, n$, and the children of vertex i are $x_{i,1}, \dots, x_{i,k_i}$.¹⁶ Then, the probability distribution of (ℓ_1, \dots, ℓ_n) in **RANDOMIZEDLST** can be simulated by [Algorithm 7](#), which is shown in the following lemma.

Lemma 5.2. *The probability distributions of (ℓ_1, \dots, ℓ_n) generated by **RANDOMIZEDLST** and by [Algorithm 7](#) are the same.*

Algorithm 7 Alternative algorithm to generate levels ℓ_1, \dots, ℓ_n

```

1: for  $i = 1, \dots, n$  do
2:   if vertex  $v_i$  is a leaf in  $T$  then
3:      $\ell_i \leftarrow 1$ 
4:   else
5:      $\ell_i^* \leftarrow \max(\ell_{x_{i,1}}, \dots, \ell_{x_{i,k_i}})$ 
6:      $c_i \leftarrow (\text{number of } j\text{'s that } \ell_{x_{i,j}} = \ell_i^*)$ 
7:      $\ell_i \leftarrow (\ell_i^* \text{ with probability } 2^{-(c_i-1)}, \text{ and } \ell_i^* + 1 \text{ with probability } 1 - 2^{-(c_i-1)})$ 

```

Proof. It suffices to show that in **RANDOMIZEDLST**, the conditional probability that $\ell_i = \ell_i^*$ given $(\ell_1, \dots, \ell_{i-1})$ is always $2^{-(c_i-1)}$, where $\ell_i^* := \max(\ell_{x_{i,1}}, \dots, \ell_{x_{i,k_i}})$ and $c_i := (\text{the number of } j\text{'s that } \ell_{x_{i,j}} = \ell_i^*)$. We relate a coloring of v_1, \dots, v_{i-1} to the colorings obtained by “flipping” the color of all the vertices in an arbitrary $C' \subseteq \{C_{x_{i,j}} : \ell_{x_{i,j}} = \ell_i^*\}$. Formally, flipping the color means that odd-numbered color $2\ell - 1$ becomes color 2ℓ , and even-numbered color 2ℓ becomes color $2\ell - 1$. Then, we obtain 2^{c_i} colorings. These colorings have the same $(\ell_1, \dots, \ell_{i-1})$, and all of them appear with the same probability because the related coloring appears when all of the probabilistic decisions in C' are inverted. However, C_i is matched for only two of them. Therefore, the probability that $\ell_i = \ell_i^*$ is $\frac{2}{2^{c_i}} = 2^{-(c_i-1)}$, and otherwise ℓ_i becomes $\ell_i^* + 1$. \square

The expected number of colors in **RANDOMIZEDLST** is, obviously, between $2\mathbb{E}[\max(\ell_1, \dots, \ell_n)] - 1$ and $2\mathbb{E}[\max(\ell_1, \dots, \ell_n)]$. Especially when T is a tree, $\ell_n = \max(\ell_1, \dots, \ell_n)$ because vertex n is the root of T ; in this case, it is crucial to estimate $\mathbb{E}[\ell_n]$. It turns out that, when we search for the graphs with the worst expected number of colors, we only have to consider the case when G is connected (i.e., T is a tree). This is because, when G is not connected, we can modify the graph by adding edges between v_n and all the other connected components, and the levels will not decrease.

5.3 Preliminaries for the analysis

In this subsection, we prove the following lemma, which shows that it is sufficient to consider when T is a binary tree. Hereafter, we call ℓ_v the level of vertex v (of T) and denote the root of T as $\text{root}(T)$.

¹⁶The constructed T is indeed a rooted forest. It has no cycles due to $x_{i,j} < i$. No vertex is a child of multiple vertices because once a connected component is merged, it is no longer a connected component of the graph.

Lemma 5.3. *There is a binary tree T that maximizes $\mathbb{E}[\ell_{\text{root}(T)}]$ among all trees with m leaves.*

First, we define the following preorder \preceq on the set of rooted trees \mathcal{T} :

Definition 5.4. *For $T_1, T_2 \in \mathcal{T}$, $T_1 \preceq T_2$ if $\Pr_{T=T_1}[\ell_{\text{root}(T_1)} \geq t] \geq \Pr_{T=T_2}[\ell_{\text{root}(T_2)} \geq t]$ for all t .*

It is easy to see that the relation \preceq satisfies reflexivity and transitivity. Also, if $T_1 \preceq T_2$, then $\mathbb{E}[\ell_{\text{root}(T_1)}] \geq \mathbb{E}[\ell_{\text{root}(T_2)}]$ holds because $\mathbb{E}[X] = \sum_{t=1}^{\infty} \Pr[X \geq t]$ for any random variable X that takes a positive integer.

Lemma 5.5. *Let T_1 be a tree, and let T_2 be a tree created by replacing a subtree T'_1 (of T_1) with a tree T'_2 . If $T'_1 \preceq T'_2$, then $T_1 \preceq T_2$ holds.*

Proof. Let $r = \text{root}(T_1)$ ($= \text{root}(T_2)$). We first consider the case that $\text{root}(T'_1)$ (also $\text{root}(T'_2)$) is a child of r . Let x_1, \dots, x_k ($x_1 = \text{root}(T'_1)$) be the children of r . If $k = 1$, then $\ell_r = \ell_{x_1}$, so it is obvious that $T_1 \preceq T_2$ holds given that $T'_1 \preceq T'_2$. So suppose $k \geq 2$. If the levels of x_2, \dots, x_k are fixed, the level of r is decided in the following way, where $\ell' = \max(\ell_{x_2}, \dots, \ell_{x_k})$, and $c' = (\text{number of } j\text{'s } (j \geq 2) \text{ that } \ell_{x_j} = \ell')$:

$$\ell_r = \begin{cases} \ell' \text{ with probability } 2^{-(c'-1)}, \text{ and } \ell' + 1 \text{ with probability } 1 - 2^{-(c'-1)} & (\ell_{x_1} < \ell') \\ \ell' \text{ with probability } 2^{-c'}, \text{ and } \ell' + 1 \text{ with probability } 1 - 2^{-c'} & (\ell_{x_1} = \ell') \\ \ell_{x_1} & (\ell_{x_1} > \ell') \end{cases}$$

Therefore:

$$\Pr[\ell_r \geq t] = \begin{cases} 1 & (t \leq \ell') \\ (1 - 2^{-(c'-1)}) + 2^{-c'} \cdot \Pr[\ell_{x_1} \geq \ell'] + 2^{-c'} \cdot \Pr[\ell_{x_1} \geq \ell' + 1] & (t = \ell' + 1) \\ \Pr[\ell_{x_1} \geq t] & (t \geq \ell' + 2) \end{cases}$$

When T'_1 is replaced by T'_2 , $\Pr[\ell_{x_1} \geq t]$ does not decrease for all t , so $\Pr[\ell_r \geq t]$ does not decrease for all t , which means that $T_1 \preceq T_2$.

Next, we prove the general case. Let v_0, v_1, \dots, v_k ($v_k = r$) be a path from $\text{root}(T'_1)$ to r , and let $T_{i,j}$ be the subtree of v_j in T_i . The assumption $T'_1 \preceq T'_2$ means $T_{1,v_0} \preceq T_{2,v_0}$. Also, if $T_{1,v_j} \preceq T_{2,v_j}$, then $T_{1,v_{j+1}} \preceq T_{2,v_{j+1}}$, as shown above. Therefore, $(T_1 =) T_{1,v_k} \preceq T_{2,v_k} (= T_2)$ is shown by induction. \square

Proof of Lemma 5.3. Let T_0 be a tree that $\mathbb{E}[\ell_{\text{root}(T_0)}]$ takes the maximum value among all trees with m leaves. If T_0 has some vertex v that has one child or three or more children, we perform the following operation to make a new tree T_1 . Let p be the parent of v (if it exists), and let x_1, \dots, x_k be the children of v .

- Case $k = 1$: Delete vertex v , and set the parent of x_1 to p (if it exists). We define this operation as *contraction*. See the left of Figure 17.
- Case $k \geq 3$: Delete vertex v , and instead create vertices w_1, \dots, w_{k-1} . Set the parents of x_1 and x_2 to w_1 , and for $j = 1, \dots, k-2$, set the parents of w_j and x_{j+2} to w_{j+1} , and finally set the parent of w_{k-1} to p (if it exists). See the right of Figure 17.

We prove that $T_0 \preceq T_1$ holds.

- Case $k = 1$: The operation is to replace T'_0 with T'_1 , where T'_0 is the subtree of v in T_0 , and T'_1 is the subtree of x_1 in T_1 . Since v has one child, $\ell_v = \ell_{x_1}$ holds; therefore, $T'_0 \preceq T'_1$. By Lemma 5.5, $T_0 \preceq T_1$.
- Case $k \geq 3$: The operation is to replace T'_0 with T'_1 , where T'_0 is the subtree of v in T_0 , and T'_1 is the subtree of w_{k-1} in T_1 . Let $\ell^* = \max(\ell_{x_1}, \dots, \ell_{x_k})$, and $c = (\text{number of } j\text{'s that } \ell_{x_j} = \ell^*)$. Then, in T_0 , the probability that $\ell_v = \ell^*$ is $2^{-(c-1)}$, and otherwise $\ell_v = \ell^* + 1$. However, in T_1 , to become $\ell_{w_{k-1}} = \ell^*$, at least $c-1$ events that “a vertex is leveled ℓ^* from two level- ℓ^* children” must happen inside w_1, \dots, w_{k-1} . The scenario happens with probability $2^{-(c-1)}$ or less. Otherwise, $\ell_{w_{k-1}}$ becomes $\ell^* + 1$ or more. Therefore, $T'_0 \preceq T'_1$. By Lemma 5.5, $T_0 \preceq T_1$.

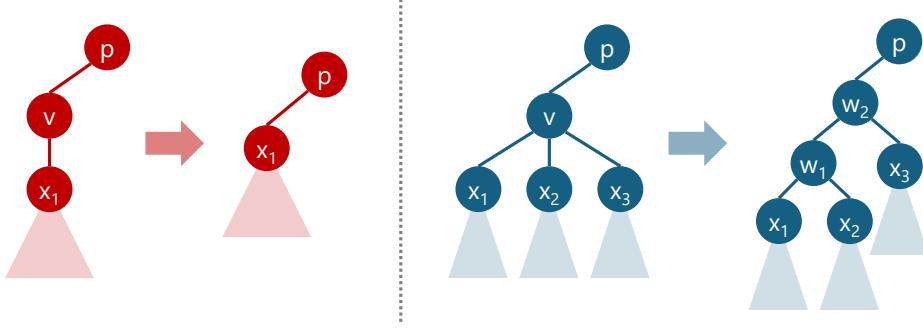


Figure 17: The operations to modify a tree, for $k = 1$ case (left) and $k \geq 3$ case (right)

We repeatedly perform the operations, starting from T_0 , until the tree becomes a binary tree. Note that the number of leaves remains unchanged by the operations. Let T be the resulting tree. By the transitivity of (\mathcal{T}, \preceq) , $T_0 \preceq T$ holds, which means that $\mathbb{E}[\ell_{\text{root}(T_0)}] \leq \mathbb{E}[\ell_{\text{root}(T)}]$. By the maximality of T_0 , the binary tree T is another tree that $\mathbb{E}[\ell_{\text{root}(T)}]$ takes the maximum value among all trees with m leaves. \square

Now, it is crucial to estimate the maximum value of $\mathbb{E}[\ell_{\text{root}(T)}]$ among all binary trees T with at most m leaves; let this value be $f(m)$. Then, we know that the expected performance of RANDOMIZEDLST is upper-bounded by $2f(n)$ colors because $n \geq m$.

5.4 Analysis 1: Considering the level-2 terminals

We start analyzing the worst-case performance of RANDOMIZEDLST. Instead of directly analyzing the maximum value of $\mathbb{E}[\ell_{\text{root}(T)}]$, we examine how quickly the level-2 vertices appear in the worst case.

Definition 5.6. For $i \geq 1$, a vertex is a level- i terminal if it has level i and both of its two children have level $i - 1$ (or is a leaf if $i = 1$).

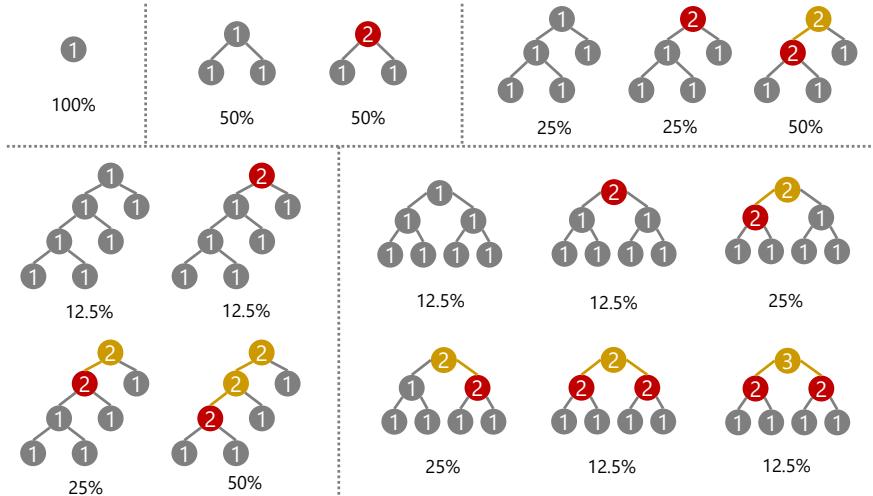


Figure 18: The outcomes of levels for all possible trees with four or fewer leaves (percentage = probability of the corresponding outcome, red vertices = level-2 terminals, yellow vertices = other level 2+ vertices). There are two binary trees with $m = 4$ leaves, and the expected number of level-2 terminals are $\frac{7}{8}$ and $\frac{9}{8}$ for the bottom-left and the bottom-right cases, respectively.

Let a_m be the maximum possible expected number of level-2 terminals for a tree with exactly m leaves. For example, $a_1 = 0$, $a_2 = \frac{1}{2}$, $a_3 = \frac{3}{4}$, and $a_4 = \frac{9}{8}$, as in [Figure 18](#). The following lemma shows that these values also navigate the number of level- i terminals for $i \geq 3$.

Lemma 5.7. *Let γ be a real number that satisfies $a_{m'} \leq \gamma \cdot m'$ for all $m' \geq 1$. Then, the expected number of level- i terminals for any tree T with m leaves is at most $\gamma^{i-1} \cdot m$.*

Proof. Let X_i be the number of level- i terminals. By the assumption, $\mathbb{E}[X_1] = m$. Let T_i be a subgraph induced by the set of vertices with level i or higher.¹⁷

We show a relation between the number of leaves in T_i (which is equal to X_i) and the number of level- $(i+1)$ terminals (which is equal to X_{i+1}). For a vertex v that has one child in T_i , the level of v is the same as the level of the only child because the level of “another child” in T is $i-1$ or less; therefore, contracting such a vertex v does not affect the analysis. After repeating the contraction, T_i becomes a binary tree with X_i leaves, and the way that levels are assigned in this binary tree is identical to that of the normal binary tree case, except that levels start from i . Therefore, the expected number of level- $(i+1)$ terminals is at most a_{X_i} . It follows that $\mathbb{E}[X_{i+1}] \leq \mathbb{E}[a_{X_i}] \leq \gamma \cdot \mathbb{E}[X_i]$. By induction, $\mathbb{E}[X_i] \leq \gamma^{i-1} \cdot m$ is shown. \square

By this lemma, we can upper-bound the expected level of the root vertex using γ .

Lemma 5.8. $\mathbb{E}[\ell_{\text{root}(T)}] \leq \frac{1}{\log(1/\gamma)} \cdot \log m + O(1)$.

Proof. If the level of the root is i or higher, there exists at least one level- i terminal, so $\Pr[\ell_{\text{root}(T)} \geq i] \leq \min(\mathbb{E}[X_i], 1) \leq \min(\gamma^{i-1} \cdot m, 1)$ (the last inequality is from [Lemma 5.7](#)). Therefore:

$$\begin{aligned}\mathbb{E}[\ell_{\text{root}(T)}] &= \sum_{i=1}^{\infty} \Pr[\ell_{\text{root}(T)} \geq i] \\ &\leq \sum_{i=1}^{\infty} \min(\gamma^{i-1} \cdot m, 1) \\ &\leq \frac{1}{\log(1/\gamma)} \cdot \log m + \left(1 + \frac{1}{1-\gamma}\right)\end{aligned}$$

which shows that $\mathbb{E}[\ell_{\text{root}(T)}] \leq \frac{1}{\log(1/\gamma)} \cdot \log m + O(1)$. \square

The remaining work for this subsection is to estimate the value of γ .

Theorem 5.9. *The minimum possible value of γ is given by the following:*

$$\gamma = \sum_{i=1}^{\infty} 2^{-(2^i - 1 + i)} = \frac{1}{2^2} + \frac{1}{2^5} + \frac{1}{2^{10}} + \frac{1}{2^{19}} + \dots < 0.282229$$

Proof. For a binary tree T , let $a(T)$ be the expected number of level-2 terminals. This can be calculated by:

$$a(T) = \sum_{v \in V(T)} 2^{-(s_v - 1)}$$

where s_v is the number of leaves in the subtree rooted at vertex v . This is because the probability for v to become a level-2 terminal is $2^{-(s_v - 1)}$; all non-leaf vertices of the subtree except v (there are $s_v - 2$ such vertices) should be leveled 1 from two level-1 children, and v should be leveled 2 from two level-1 children.

Let T be a tree that maximizes $a(T)$ among all trees with $m = 2^k$ leaves ($k \geq 1$). We say that two leaves are *paired* if they share the parents. Suppose that there exist at least two *unpaired* leaves, and let u_1 and u_2 be two of them (since 2^k is an even number, the number of unpaired leaves is always even.) Then, we perform the following operations on T :

¹⁷In [Figure 18](#), T_2 corresponds to the subgraph with red/yellow vertices and yellow edges.

1. Delete u_1 and contract the parent p of u_1 . This operation decreases $a(T)$ by at most $2^{-(3-1)} = \frac{1}{4}$ because this deletes p , and since u_1 is unpaired, $s_p \geq 3$ holds. (Figure 19 left to middle)
2. Create two new vertices and set their parents to u_2 . This operation increases $a(T)$ by at least $\frac{1}{2} - (\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \dots + \frac{1}{2^m}) = \frac{1}{4} + \frac{1}{2^m}$ because by this operation, s_{u_2} becomes 2, and s_v for all ancestors of u_2 increases by 1. Note that, since u_2 is unpaired, $s_v \geq 3$ holds for the parent of u_2 . (Figure 19 middle to right)

Overall, $a(T)$ increases by at least $\frac{1}{2^m}$, contradicting the maximality of $a(T)$. Therefore, every leaf is paired to make a subtree with two leaves.

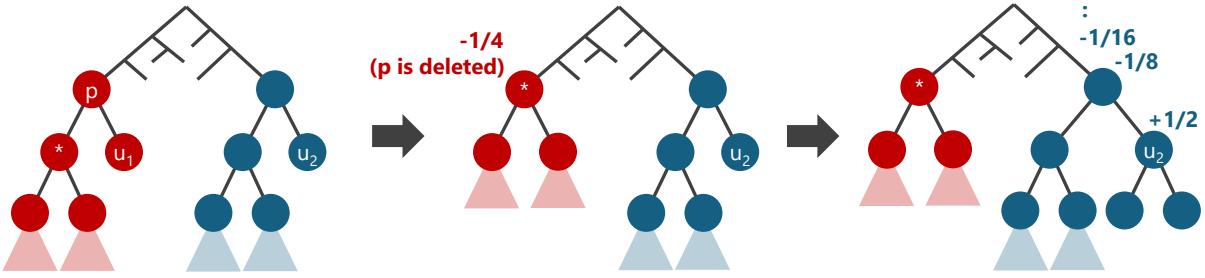


Figure 19: The sketch of operations.

Using the same argument, we can show inductively that every subtree with 2^i leaves must be paired, for $i = 1, 2, \dots, k-1$. Therefore, T is a complete binary tree; a binary tree that all 2^k leaves are at depth k . Calculating $a(T)$ for the complete binary tree gives:

$$a_{2^k} = \sum_{i=1}^k 2^{-(2^i-1)} \cdot 2^{k-i} = 2^k \cdot \sum_{i=1}^k 2^{-(2^i-1+i)}$$

because there are 2^{k-i} vertices such that $s_v = 2^i$. It follows that $a_{2^k} \leq \gamma \cdot 2^k$ for γ given in the statement, and the given γ is the minimum possible value.

Suppose $a_m > \gamma \cdot m$ for some m . Then, for a large enough k , $a_m \cdot \lfloor \frac{2^k}{m} \rfloor > \gamma \cdot 2^k$ holds. For such k , we can make a binary tree T with 2^k leaves, which contains $\lfloor \frac{2^k}{m} \rfloor$ subtrees with m leaves. This tree can satisfy $a(T) > \gamma \cdot 2^k$, which is a contradiction to $a_{2^k} \leq \gamma \cdot 2^k$. Therefore, $a_m \leq \gamma \cdot m$ for all m . \square

By assigning this γ to Lemma 5.8, we make partial progress to the main problem of this section.

Corollary 5.10. *For any graph G , RANDOMIZEDLST uses at most $1.096 \log n + O(1)$ colors in expectation.*

5.5 Analysis 2: Increasing the layers

In this subsection, we further improve the upper bound. To this end, we analyze how quickly the level- $(L+1)$ vertices appear in the worst case for a constant $L \geq 2$. Similar to the $L=1$ case, we consider the maximum expected number of level- $(L+1)$ terminals for a tree with m leaves, $a_m^{(L)}$. Let $\gamma^{(L)}$ be the minimum γ that $a_m^{(L)} \leq \gamma \cdot m$ for every $m \geq 1$; for example, we have shown $\gamma^{(1)} \approx 0.282228$ (see Theorem 5.9). We can show the following generalization of Lemma 5.8.

Lemma 5.11. $\mathbb{E}[\ell_{\text{root}}(T)] \leq \frac{L}{\log_2(1/\gamma^{(L)})} \cdot \log m + O(1)$.

Proof. We can prove this fact similarly to the proofs of Lemma 5.7 and Lemma 5.8. \square

Estimating the value of $\gamma^{(L)}$ is difficult for $L \geq 2$, so we try to obtain a good upper bound for $\gamma^{(L)}$ by computer check. Obviously, simple brute force is impossible because trees can be infinitely large. Instead, our idea is to sum up the performances of small enough subtrees. The following lemma demonstrates that the results of small trees can be used to upper-bound the value of $\gamma^{(L)}$:

Lemma 5.12. *Let $b_m^{(L)}$ be the maximum value of $(\text{expected number of level-} (L+1) \text{ terminals}) + \Pr[\ell_{\text{root}(T)} \leq L]$ among all binary trees T with m leaves. Then,*

$$\gamma^{(L)} \leq \max \left(\frac{b_B^{(L)}}{B}, \dots, \frac{b_{2B-1}^{(L)}}{2B-1} \right)$$

holds for any $B \geq 1$.

Proof. Consider any binary tree T . For explanation, we color vertex v red if $s_v \geq B$ and $s_x < B$ for all the children x (of v). Note that as in the proof of [Theorem 5.9](#), s_v is the number of leaves in the subtree rooted at vertex v . For each red-colored vertex v , let $U_v = (\text{the set of vertices in the subtree of } v) \cup (\text{the set of ancestors of } v)$. We claim that the expected number of level- $(L+1)$ terminals in U_v is bounded by $b_{s_v}^{(L)}$. Since there can be at most one level- $(L+1)$ terminals in the ancestors of v (only when $\ell_v \leq L$), the claim follows. Since $B \leq s_v \leq 2B-1$ and the subtrees of red vertices are disjoint, the expected number of level- $(L+1)$ terminals in $\bigcup_{v:\text{red}} U_v$ is at most:

$$\max \left(\frac{b_B^{(L)}}{B}, \dots, \frac{b_{2B-1}^{(L)}}{2B-1} \right) \cdot \sum_{v:\text{red}} s_v.$$

Some vertices are in $V(T) \setminus \bigcup_{v:\text{red}} U_v$, but they are composed of subtrees with $B-1$ or fewer leaves. Thus, the expected number of level- $(L+1)$ terminals in these vertices is at most:

$$\max \left(\frac{a_1^{(L)}}{1}, \dots, \frac{a_{B-1}^{(L)}}{B-1} \right) \cdot \left(m - \sum_{v:\text{red}} s_v \right).$$

Overall, the expected number of level- $(L+1)$ terminals in T can be upper-bounded by

$$\max \left(\frac{a_1^{(L)}}{1}, \dots, \frac{a_{B-1}^{(L)}}{B-1}, \frac{b_B^{(L)}}{B}, \dots, \frac{b_{2B-1}^{(L)}}{2B-1} \right) \cdot m,$$

and since $a_m^{(L)}/m \leq \gamma^{(L)}$, the statement of the lemma holds. \square

So, how do we calculate $b_m^{(L)}$? Brute-forcing all binary trees to calculate $b_m^{(L)}$ is realistic only for $m \leq 40$, even using a computer check. Instead, we attempt to obtain a good upper bound for $b_m^{(L)}$. First, we use the following lemma as a tool.

Lemma 5.13. *Let $p_{m,t}$ be the maximum value of $\Pr[\ell_{\text{root}(T)} \geq t]$ among all binary trees T with m leaves. Define $p'_{m,t}$ ($m \geq 1, t \geq 1$) by the recurrence relation that $p'_{m,1} = 1$ and*

$$p'_{m,t} = \max_{m_l+m_r=m} \left\{ 1 - (1 - p'_{m_l,t})(1 - p'_{m_r,t}) + \frac{1}{2}(p'_{m_l,t-1} - p'_{m_l,t})(p'_{m_r,t-1} - p'_{m_r,t}) \right\}$$

for $m \geq 2$. Then, $p_{m,t} \leq p'_{m,t}$ holds for all m, t .

Proof. Let v_l, v_r be the children of $\text{root}(T)$, and let m_l, m_r the number of leaves in subtree of v_l and v_r , respectively. Here, the following equation holds:

$$\Pr[\ell_{\text{root}(T)} \geq t] = (1 - \Pr[\ell_{v_l} < t] \cdot \Pr[\ell_{v_r} < t]) + \frac{1}{2} \cdot \Pr[\ell_{v_l} = t - 1] \cdot \Pr[\ell_{v_r} = t - 1]$$

This is because, $\ell_{\text{root}(T)} \geq t$ if exactly one of the following happens:

- Either $\ell_{v_l} \geq t$ or $\ell_{v_r} \geq t$ (probability $1 - \Pr[\ell_{v_l} < t] \cdot \Pr[\ell_{v_r} < t]$)
- $\ell_{v_l} = \ell_{v_r} = t - 1$ and $\ell_{\text{root}(T)} = t$ (probability $\frac{1}{2} \cdot \Pr[\ell_{v_l} = t - 1] \cdot \Pr[\ell_{v_r} = t - 1]$)

By Lemma 5.5, $\Pr[\ell_{\text{root}(T)} \geq t]$ is a (non-strictly) increasing function with respect to $\Pr[\ell_{v_l} \geq i]$ and $\Pr[\ell_{v_r} \geq i]$ for each $i \in \mathbb{N}$. Since $\Pr[\ell_v < t] = 1 - \Pr[\ell_v \geq t]$ and $\Pr[\ell_v = t] = \Pr[\ell_v \geq t - 1] - \Pr[\ell_v \geq t]$ hold for any vertex v , the following holds:

$$\Pr[\ell_{\text{root}(T)} \geq t] \leq \{1 - (1 - p'_{m_l, t})(1 - p'_{m_r, t})\} + \frac{1}{2}(p'_{m_l, t-1} - p'_{m_l, t})(p'_{m_r, t-1} - p'_{m_r, t})$$

which gives an upper bound of $p_{m, t}$. We note that $p'_{m, t}$ can be calculated in $O(m^2 t)$ time. \square

Next, we show an alternative way to count the expected number of level- k terminals. Let $q_v = \Pr[\ell_v \geq L + 1]$. Then, the probability that v is a level- $(L + 1)$ terminal is the following, where $\text{left}(v), \text{right}(v)$ are children of v :

$$q_v - (1 - (1 - q_{\text{left}(v)})(1 - q_{\text{right}(v)}))$$

This is because v is a level- k terminal when v has level $L + 1$ or more (probability q_v) but not “either $\text{left}(v)$ or $\text{right}(v)$ have level $L + 1$ or more (probability $1 - (1 - q_{\text{left}(v)})(1 - q_{\text{right}(v)})$ ”). The number of level- $(L + 1)$ terminals in T is the sum of this value for all $v \in V(T)$.

For $b_m^{(L)}$, we considered (expected number of level- $(L + 1)$ terminals) + $\Pr[\ell_{\text{root}(T)} \leq L]$. The value of $\Pr[\ell_{\text{root}(T)} \leq L]$ ($= 1 - q_{\text{root}(T)}$) is equal to the sum of

$$-q_v + q_{\text{left}(v)} + q_{\text{right}(v)}$$

across all non-leaf vertex v , plus 1. This is because $q_v = 0$ when v is a leaf, and q_v for all non-root, non-leaf vertices are canceled out, only $-q_{\text{root}(T)}$ to remain. Therefore:

$$(\text{expected number of level-}(L + 1) \text{ terminals}) + \Pr[\ell_{\text{root}(T)} \leq L] = 1 + \sum_{v:\text{non-leaf}} q_{\text{left}(v)} q_{\text{right}(v)}$$

because $(q_v - (1 - (1 - q_{\text{left}(v)})(1 - q_{\text{right}(v)}))) + (-q_v + q_{\text{left}(v)} + q_{\text{right}(v)}) = q_{\text{left}(v)} q_{\text{right}(v)}$. So, we must estimate the maximum value of $1 + \sum_{v:\text{non-leaf}} q_{\text{left}(v)} q_{\text{right}(v)}$.

Here, q_v can be between 0 and $p'_{s_v, L+1}$, where s_v is the number of leaves in the subtree of v . We consider the relaxed problem to maximize $1 + \sum_{v:\text{non-leaf}} q_{\text{left}(v)} q_{\text{right}(v)}$ under $0 \leq q_v \leq p'_{s_v, L+1}$ for all $v \in V(T)$. Let $b'_m^{(L)}$ be the maximum value for the relaxed problem among all tree T .

If the tree T is fixed, it is obvious that $q_v = p'_{s_v, L+1}$ is the optimal solution. Using this fact, the dynamic programming formula for the optimal value $b'_m^{(L)}$ can be easily found:

Lemma 5.14. $b'_m^{(L)}$ satisfies the following recurrence relation:

$$b'_1^{(L)} = 1, \quad b'_m^{(L)} = 1 + \max_{m_l + m_r = m} ((b'^{(L)}_{m_l} - 1) + (b'^{(L)}_{m_r} - 1) + p'_{m_l, L+1} \cdot p'_{m_r, L+1})$$

We note that $b'_m^{(L)}$ can be calculated in $O(m^2)$ time. The calculated $b'_m^{(L)}$ is an upper bound for $b_m^{(L)}$. Hence, assigning $b'_m^{(L)}$ to $b_m^{(L)}$ in Lemma 5.12, we give an upper bound (or “upper bound of upper bound”) of $\gamma^{(L)}$, leading to the better analysis for the number of colors in RANDOMIZEDLST.

Tackling numerical errors. In order to upper bound $\gamma^{(L)}$, we must calculate $p'_{m,t}$ and b'_m . Conventionally, these values are represented by floating-point numbers, but it may cause numerical errors and create a hole in the proof. It is ideal to calculate everything with integers. So, we *round up* each calculation of $p'_{m,t}$ and b'_m to a rational number of the form $\frac{n}{D}$ ($n \in \mathbb{Z}$), where D is a fixed integer parameter. When we run the dynamic programming program, the calculated $p'_{m,t}$ and b'_m will not be lower than the actual $p'_{m,t}$ and b'_m . Therefore, we can calculate an upper bound of $\gamma^{(L)}$, and becomes more precise when D is larger.

Results. We computed b'_m for $L = 1, \dots, 10$ and $m \leq 2 \cdot 2^{22} + 1$ with $D = 2^{30}$. We note that it is reasonable to set $B = 2^k + 1$ for some integer k , because $b'_m/(2B-1)$ tends to be especially large when m is a power of two. The resulting upper bounds on the expected number of colors in **RANDOMIZEDLST**, is shown in [Table 3](#).

Table 3: Computed $\gamma' = \max(b'_B^{(L)}/B, \dots, b'_{2B-1}^{(L)}/(2B-1))$ and the corresponding upper bounds on the number of colors in **RANDOMIZEDLST** (divided by $\log n$), rounded up to 6 decimal places

L	B	γ'	colors
1	$2^4 + 1$	2.822285e-1	1.095852
2	$2^6 + 1$	7.373281e-2	1.063392
3	$2^8 + 1$	1.912694e-2	1.051111
4	$2^{10} + 1$	4.957865e-3	1.044924
5	$2^{12} + 1$	1.284998e-3	1.041231

L	B	γ'	colors
6	$2^{14} + 1$	3.330478e-4	1.038783
7	$2^{16} + 1$	8.632023e-5	1.037042
8	$2^{18} + 1$	2.237284e-5	1.035741
9	$2^{20} + 1$	5.798723e-6	1.034731
10	$2^{22} + 1$	1.502954e-6	1.033925

Now, we obtain $\gamma^{(10)} \leq 1.502954 \times 10^{-6}$. Therefore, by [Lemma 5.11](#), the following theorem holds:

Theorem 5.15. *For any graph G , **RANDOMIZEDLST** uses at most $1.034 \log n + O(1)$ colors in expectation.*

Performance of **RANDOMIZEDLST.** We estimate $\gamma^{(L)}$ for $L \leq 10$, but if we can further increase L , it would improve the upper bound on the performance of **RANDOMIZEDLST**. It seems to us that the worst-case input graph for **RANDOMIZEDLST** is a complete binary tree. In this case, the experiment with dynamic programming shows that it uses around $1.027 \log n$ colors. Therefore, we conjecture the following:

Conjecture 5.16. ***RANDOMIZEDLST** uses at most $1.027 \log n + O(1)$ colors for any bipartite graph G .*

Computer check and programs. The programs used for the analysis and their results in this section can be downloaded at <https://github.com/square1001/online-bipartite-coloring>.

6 Lower bound of the expected performance for bipartite graphs

6.1 The lower bound instance

In this section, we show the limit of randomized algorithms (against an oblivious adversary). We prove the following theorem.

Theorem 6.1. *Any randomized online coloring algorithm for bipartite graphs requires at least $\frac{91}{96} \log n - O(1)$ colors in expectation in the worst case.*

By Yao's lemma [24], the goal is to give a distribution of bipartite graphs that any *deterministic* algorithm uses at least $\frac{91}{96} \log n - O(1)$ colors in expectation. We construct such input graphs as in Figure 20, having structures similar to a binary tree. The grade- h instance, which is constructed by merging two disjoint grade- $(h-1)$ instances with two extra vertices, contains $4 \cdot 2^h - 2$ vertices.

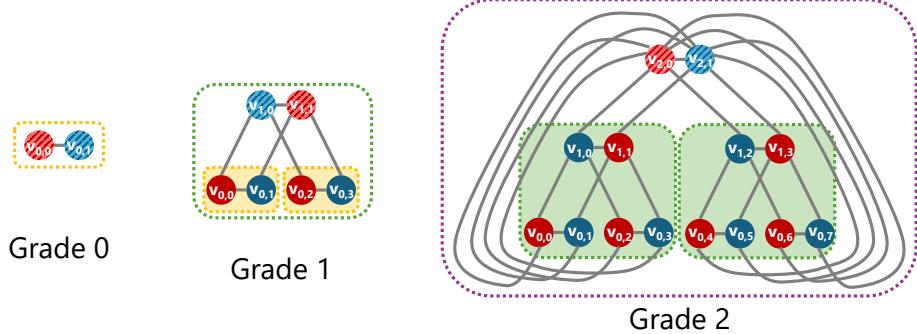


Figure 20: The instances to give a lower bound for $h = 0, 1, 2$. The orange, green, and purple regions correspond to grade-0, grade-1, and grade-2 graphs, respectively. The two extra vertices are in stripe. The labels of vertices can be changed, depending on the random choice.

We formally explain how to construct the grade- k instance. The vertices are $v_{i,j}$ ($0 \leq i \leq h, 0 \leq j < 2^{h-i+1}$). We refer to the phase that $v_{i,0}, \dots, v_{i,2^{h-i+1}-1}$ arrive as “phase i ”. At phase i , there are 2^{h-i+1} components of grade- $(i-1)$ graphs. For $j = 0, 2, \dots, 2^{h-i+1} - 2$, we randomly select two of the remaining grade- $(i-1)$ components, and “merge” them with two new vertices $v_{i,j}$ and $v_{i,j+1}$. Then, after phase i , there are 2^{h-i} components of grade- i graphs.

Formally, when we merge two connected components (say C_1 and C_2) with two new vertices (say v_a and v_b), for each vertex in C_1 and C_2 , we add an edge between it and *either* v_a or v_b in a way that the resulting graph remains bipartite, and we also add an edge between v_a and v_b . There are four possible resulting graphs because we can choose which side of bipartition of C_i will be linked to v_a (and to v_b), independently for $i = 1, 2$. The four choices will be selected with probability $\frac{1}{4}$ each. We denote this procedure as $\text{MERGE}(C_1, C_2, v_a, v_b)$, which returns the resulting component.

The goal is to prove the following result, which implies Theorem 6.1.

Theorem 6.2. *Any deterministic online coloring algorithm requires at least expected $\frac{91}{96}h - O(1)$ colors for the grade- h instance.*

6.2 Introducing potential

A classic idea to lower-bound the performance of algorithms is to define a value called the *potential* for the current state and say that the potential always increases by a certain amount in each operation.

We define the potential for each connected component C . Define the state of C to be (X, Y) , where X and Y are the sets of colors used in each bipartition of C . Later, we may also use (X, Y) to represent C itself. We consider the following potential ϕ_1 :

$$\phi_1(C) := \frac{1}{2}(|X| + |Y|)$$

We will show that, for each phase, the average potential of the components will increase by at least $\frac{3}{4}$ in expectation. Since $|X \cup Y| \geq \phi_1(C)$ always holds, this proves that the expected number of colors used for the grade- h instance is at least $\frac{3}{4}h + 1$. To this end, we first prove that X and Y are not in the inclusion relation.

Lemma 6.3. Let C_1 and C_2 be two connected components, and let $C := \text{MERGE}(C_1, C_2, v_a, v_b)$. Then, the state (X, Y) of C neither satisfies $X \subseteq Y$ nor $Y \subseteq X$.

Proof. To this end, we will show that $X \subseteq Y$ is impossible. Let (X_0, Y_0) be the state of C before coloring v_a and v_b (they are “uncolored” at this moment). Let c_a and c_b be the colors used for v_a and v_b , respectively. Without loss of generality, v_a is adjacent to vertices with a color in Y_0 . Thus, $c_a \notin Y_0$. Since $c_a \neq c_b$, $c_a \notin Y_0 \cup \{c_b\} = Y$. Therefore, $X = X_0 \cup \{c_a\} \not\subseteq Y$. By symmetry, $Y \subseteq X$ is also impossible. \square

This lemma implies that, for the grade- h instance, the state (X, Y) of any component at any time neither satisfies $X \subseteq Y$ nor $Y \subseteq X$. Now, we start proving the $\frac{3}{4}$ lower bound. It suffices to prove the following lemma.

Lemma 6.4. Let C_1 and C_2 be two connected components, and let $C := \text{MERGE}(C_1, C_2, v_a, v_b)$. Then, $\mathbb{E}[\phi_1(C)] \geq \frac{1}{2}(\phi_1(C_1) + \phi_1(C_2)) + \frac{3}{4}$ when the state (X, Y) for each C_i neither satisfies $X \subseteq Y$ nor $Y \subseteq X$.

Proof. Let (X_1, Y_1) and (X_2, Y_2) be the states of C_1 and C_2 , respectively. Let (X_0, Y_0) and (X, Y) be the states of C before and after coloring $\{v_a, v_b\}$, respectively. Here, (X_0, Y_0) can be any of $(X_1 \cup X_2, Y_1 \cup Y_2)$, $(X_1 \cup Y_2, Y_1 \cup X_2)$, $(Y_1 \cup Y_2, X_1 \cup X_2)$, $(Y_1 \cup X_2, X_1 \cup Y_2)$, with probability $\frac{1}{4}$ each. We consider the following value Δ_0 . Since $X_0 \subseteq X$ and $Y_0 \subseteq Y$, Δ_0 gives a lower bound of $\mathbb{E}[\phi_1(C)] - \frac{1}{2}(\phi_1(C_1) + \phi_1(C_2))$.

$$\Delta_0 := \mathbb{E}[\phi_1((X_0, Y_0))] - \frac{1}{2}(\phi_1(C_1) + \phi_1(C_2)) \quad \left(\phi_1((X_0, Y_0)) = \frac{1}{2}(|X_0| + |Y_0|) \right) \quad (1)$$

We see how much each color c contributes to Δ_0 , for the first and the second terms of Equation 1. Excluding the symmetric patterns, there are six cases to consider, shown in Table 4:

Case	1st term	2nd term	Δ_0
$c \notin X_1, Y_1, X_2, Y_2$	0	0	0
$c \in X_1, c \notin Y_1, X_2, Y_2$	1/2	1/4	1/4
$c \in X_1, X_2, c \notin Y_1, Y_2$	3/4	1/2	1/4
$c \in X_1, Y_1, c \notin X_2, Y_2$	1	1/2	1/2
$c \in X_1, Y_1, X_2, c \notin Y_2$	1	3/4	1/4
$c \in X_1, Y_1, X_2, Y_2$	1	1	0

Table 4: For each case, contribution to the 1st and 2nd term of (1), along with their difference

By Lemma 6.3, $|X_1 \oplus Y_1| \geq 2$ and $|X_2 \oplus Y_2| \geq 2$, where $X_i \oplus Y_i = (X_i \setminus Y_i) \cup (Y_i \setminus X_i)$ is the symmetric difference of X_i and Y_i . The 2nd, 3rd, and 5th patterns in Table 4 correspond to the case that $c \in X_i \oplus Y_i$ for an $i \in \{1, 2\}$ (the 3rd pattern is double-counted); for each case, each element in $X_i \oplus Y_i$ ($i = 1, 2$) contributes to Δ_0 by $\frac{1}{4}, \frac{1}{8}, \frac{1}{4}$, respectively. Therefore, $\Delta_0 \geq 4 \times \frac{1}{8} = \frac{1}{2}$, and except for the case that “two c ’s are in the 3rd pattern and no c ’s are in the 2nd, 4th, and 5th patterns,” $\Delta_0 \geq \frac{3}{4}$, which means that the expected average potential increases by $\frac{3}{4}$. Note that there is no case that $\Delta_0 = \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{5}{8}$ because the number of “ $\frac{1}{8}$ ” must be even.

The only exceptional case is, without loss of generality, $X_1 = \{1, \dots, c, c+1\}, Y_1 = \{1, \dots, c, c+2\}, X_2 = \{1, \dots, c, c+1\}, Y_2 = \{1, \dots, c, c+2\}$ for some c . In this case:

- If $(X_0, Y_0) = (X_1 \cup X_2, Y_1 \cup Y_2)$, we can color v_a, v_b by color $c+1, c+2$, respectively, and $(X, Y) = (\{1, \dots, c, c+1\}, \{1, \dots, c, c+2\})$. The average potential stays at $c+1$.
- If $(X_0, Y_0) = (X_1 \cup Y_2, Y_1 \cup X_2)$, $X_0 = Y_0 = \{1, \dots, c+2\}$, so we need two extra colors to color v_a and v_b . The average potential increases from $c+1$ to $c+3$.

- The cases of $(X_0, Y_0) = (Y_1 \cup Y_2, X_1 \cup X_2), (Y_1 \cup X_2, X_1 \cup Y_2)$ are symmetric to the first and the second cases, respectively.

Therefore, the expected average potential increases by 1 in this case. \square

6.3 Two-phase analysis

In the previous subsection, we showed that any randomized algorithm requires at least $\frac{3}{4} \log n - \frac{1}{2}$ colors, using potential ϕ_1 . Unfortunately, this is the best possible potential among functions of $|X \cap Y|, |X \setminus Y|, |Y \setminus X|$. In order to improve the lower bound, we need to develop more sophisticated analysis methods.

In this subsection, we see how much the expected average potential of the components increases in two phases. We consider the model with four connected components, C_1, C_2, C_3, C_4 . Two pairs of components are merged in the first phase, and the two “merged” components are merged in the second phase. There are essentially $3 \times 2 \times 2^3 = 48$ outcomes, considering how components are paired (3 ways) and are merged in which order (2 ways), along with how merges happen ($2^3 = 8$ ways, because 3 merges happen).¹⁸ The “player” (algorithm) must decide the color of two added vertices right after every merge. Therefore, the minimum expected increase in potential ϕ when the player plays optimally, denoted by $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi)$, can be computed in the expected minimax algorithm ([Algorithm 8](#)). The complex nature of this procedure makes it more difficult to create cases with low increases in potential.

We introduce the following potential ϕ_2 , which slightly modifies ϕ_1 .¹⁹ The goal is to prove [Lemma 6.5](#), which directly leads to a lower bound of $\frac{89}{96}h - O(1)$ colors.

$$\phi_2(C) = |X \cap Y| + \frac{11}{21}|X \oplus Y|$$

Lemma 6.5. $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_2) \geq \frac{89}{48}$ for any components C_1, C_2, C_3, C_4 where state (X, Y) neither satisfies $X \subseteq Y$ nor $Y \subseteq X$.

Proof. This lemma can be shown by a computer search. First, we need to make the number of candidates for the combination of states of C_1, C_2, C_3, C_4 computable (or at least finite). With all the ideas explained in the next subsection, we reduce the number of combinations to consider to 16829. We compute $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_2)$ for all of them, and each call returned a result of $\frac{89}{48}$ or more. \square

Remarks on Algorithm 8. When brute-forcing colors $(c_a^{(i)}, c_b^{(i)})$ ($i = 1, 2, 3$) in [Algorithm 8](#), we may assume that we do not “jump” the colors; when there are only colors $1, \dots, c$ in the current graph, we only color the next vertex using any of color $1, \dots, c+1$. This makes the number of search states finite. Also, when $X_{i+4} \setminus Y_{i+4}$ is not empty, it is obvious that choosing $c_a^{(i)}$ from $X_{i+4} \setminus Y_{i+4}$ is an optimal strategy (then, it becomes $X'_{i+4} = X_{i+4}$). Similarly, when $Y_{i+4} \setminus X_{i+4}$ is not empty, it is optimal to choose $c_b^{(i)}$ from $Y_{i+4} \setminus X_{i+4}$. Then, we can further reduce the number of possibilities to consider.

6.4 Further improvement: Limiting the number of combinations

In this subsection, we complete the proof of [Lemma 6.5](#); indeed, we explain how to enumerate all candidates of combinations of states of C_1, C_2, C_3, C_4 such that $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_2)$ is less than $\frac{89}{48}$, using a computer search.

¹⁸When merging components with states (X_1, Y_1) and (X_2, Y_2) , there are essentially two possible resulting states: $(X_1 \cup X_2, Y_1 \cup Y_2)$ and $(X_1 \cup Y_2, Y_1 \cup X_2)$, as we can regard (X, Y) and (Y, X) as the same states.

¹⁹Even with ϕ_1 , we can prove the lower bound of $\frac{59}{64} \log_2 n - O(1)$, which is only slightly worse than $\frac{89}{96} \log_2 n - O(1)$.

Algorithm 8 POTENTIALINCREASE(C_1, C_2, C_3, C_4, ϕ)

```

1:  $x \leftarrow 0$ 
2: for all  $(C'_1, C'_2, C'_3, C'_4)$ , a permutation of  $(C_1, C_2, C_3, C_4)$ , that  $C'_1$  and  $C'_2$  are merged first and  $C'_3$  and  $C'_4$  are merged second (there are  $3 \times 2 = 6$  ways) do
3:   Let  $(X'_i, Y'_i)$  be the state of  $C'_i$  for  $i = 1, 2, 3, 4$ 
4:    $a_0 \leftarrow 0$ 
5:   for  $(X_5, Y_5) = (X'_1 \cup X'_2, Y'_1 \cup Y'_2), (X'_1 \cup Y'_2, Y'_1 \cup X'_2)$  do
6:      $b_0 \leftarrow +\infty$ 
7:     for all  $(c_a^{(1)}, c_b^{(1)})$ , the colors of two extra vertices when merging  $C'_1$  and  $C'_2$  do
8:        $(X'_5, Y'_5) \leftarrow (X_5 \cup \{c_a^{(1)}\}, Y_5 \cup \{c_b^{(1)}\})$ 
9:        $a_1 \leftarrow 0$ 
10:      for  $(X_6, Y_6) = (X'_3 \cup X'_4, Y'_3 \cup Y'_4), (X'_3 \cup Y'_4, Y'_3 \cup X'_4)$  do
11:         $b_1 \leftarrow +\infty$ 
12:        for all  $(c_a^{(2)}, c_b^{(2)})$ , the colors of two extra vertices when merging  $C'_3$  and  $C'_4$  do
13:           $(X'_6, Y'_6) \leftarrow (X_6 \cup \{c_a^{(2)}\}, Y_6 \cup \{c_b^{(2)}\})$ 
14:           $a_2 \leftarrow 0$ 
15:          for  $(X_7, Y_7) = (X'_5 \cup X'_6, Y'_5 \cup Y'_6), (X'_5 \cup Y'_6, Y'_5 \cup X'_6)$  do
16:             $b_2 \leftarrow +\infty$ 
17:            for all  $(c_a^{(3)}, c_b^{(3)})$ , the colors of two extra vertices in the final merge do
18:               $(X'_7, Y'_7) \leftarrow (X_7 \cup \{c_a^{(3)}\}, Y_7 \cup \{c_b^{(3)}\})$ 
19:               $\Delta \leftarrow \phi((X'_7, Y'_7)) - \frac{1}{4}(\phi(C_1) + \phi(C_2) + \phi(C_3) + \phi(C_4))$ 
20:               $b_2 \leftarrow \min(b_2, \Delta)$ 
21:               $a_2 \leftarrow a_2 + \frac{1}{2}b_2$ 
22:               $b_1 \leftarrow \min(b_1, a_2)$ 
23:               $a_1 \leftarrow a_1 + \frac{1}{2}b_1$ 
24:               $b_0 \leftarrow \min(b_0, a_1)$ 
25:               $a_0 \leftarrow a_0 + \frac{1}{2}b_0$ 
26:       $x \leftarrow x + \frac{1}{6}a_0$ 
27: return  $x$ 

```

The searching framework. The following lemma, the “potential ϕ_2 version” of [Lemma 6.4](#), shows that ϕ_2 always increases by $\frac{31}{42}$ in the second phase.

Lemma 6.6. *Let C_1 and C_2 be connected components, and let $C := \text{MERGE}(C_1, C_2, v_a, v_b)$. Then, $\mathbb{E}[\phi_2(C)] \geq \frac{1}{2}(\phi_2(C_1) + \phi_2(C_2)) + \frac{31}{42}$ holds when the state (X, Y) for each C_i neither satisfies $X \subseteq Y$ nor $Y \subseteq X$.*

We can prove this lemma similarly to [Lemma 6.4](#), but we can also prove by running [Algorithm 9](#) (by calling $g((\emptyset, \emptyset), (\emptyset, \emptyset))$, which will be explained later). Therefore, it suffices to search all the cases where, in the first phase, the expected average potential increases by less than $\frac{89}{48} - \frac{31}{42} = \frac{125}{112}$. Let $f(C_i, C_j)$ be the expected increase of a potential ϕ_2 by merging components C_i and C_j . We enumerate all the cases such that:

$$\Delta_1 = \frac{1}{6} (f(C_1, C_2) + f(C_1, C_3) + f(C_1, C_4) + f(C_2, C_3) + f(C_2, C_4) + f(C_3, C_4)) < \frac{125}{112}$$

Implementation and the state matrix. In order to implement the brute force of the cases such that $\Delta_1 < \frac{125}{112}$, we define *state matrix* M to represent (X_i, Y_i) , the state of C_i , for $i = 1, 2, 3, 4$. M is a $4 \times m$ matrix

where m is the number of colors, defined in the following way.

$$M_{i,j} = \begin{cases} 0 & (j \notin X_i, Y_i) \\ 1 & (j \in X_i, j \notin Y_i) \\ 2 & (j \in Y_i, j \notin X_i) \\ 3 & (j \in X_i, Y_i) \end{cases}$$

Here, we can assume that each column is not “all 0” or “all 3” (so there are 254^m matrices with m columns). This is because if a color is in none or all of $X_1, Y_1, \dots, X_4, Y_4$, there is no effect on the increase of potential. We attempt to brute-force these matrices by DFS (depth-first search), which appends one column to the right in each step (corresponds to adding a new color).

Lower-bounding for the DFS. In order to execute the DFS in a finite and realistic time, we need to apply “pruning” by showing that, for the current state matrix, it is impossible to achieve $\Delta_1 < \frac{125}{112}$ no matter how subsequent columns are appended.

Let $g(C_i, C_j)$ be the minimum value of $f(C_i, C_j)$ when we can freely add subsequent colors to (X_i, Y_i) and (X_j, Y_j) . Then, $LB := \frac{1}{6}(g(C_1, C_2) + g(C_1, C_3) + g(C_1, C_4) + g(C_2, C_3) + g(C_2, C_4) + g(C_3, C_4))$ gives the lower bound of Δ_1 for any subsequent state matrices. Therefore, once $LB \geq \frac{125}{112}$ is met, we do not need to perform the DFS for subsequent columns and can apply pruning. We can compute $g(C_i, C_j)$ using [Algorithm 9](#), which is given by $\text{LOWERBOUND}(C_i, C_j, m, +\infty)$.

Algorithm 9 $\text{LOWERBOUND}(C_1, C_2, m, \beta)$: Function to calculate the minimum value of $f(C_1, C_2)$ when colors $m+1, m+2, \dots$ are added, or report that the minimum value is β or more

```

1:  $d_0 \leftarrow$  (the value of  $\Delta_0$  defined in Lemma 6.4 (for potential  $\phi_2$ ) for the current  $C_1, C_2$ )
2: if  $d_0 \geq \beta$  then
3:   return  $\beta$                                  $\triangleright f(C_1, C_2) \geq \beta$  no matter how subsequent colors are added
4:    $(X_1^*, Y_1^*), (X_2^*, Y_2^*) \leftarrow$  (states of components which adds minimum possible subsequent colors into  $C_1, C_2$  to make it satisfy  $X_1 \not\subseteq Y_1, Y_1 \not\subseteq X_1, X_2 \not\subseteq Y_2, Y_2 \not\subseteq X_2$ )
5:    $\beta \leftarrow \min(\beta, f((X_1^*, Y_1^*), (X_2^*, Y_2^*)))$ 
6: for all  $(X'_1, Y'_1), (X'_2, Y'_2)$ , the states of  $C_1, C_2$  after adding color  $m+1$  (there are  $2^4 - 2 = 14$  ways) do
7:    $d \leftarrow \text{LOWERBOUND}((X'_1, Y'_1), (X'_2, Y'_2), m+1, \beta)$ 
8:    $\beta \leftarrow \min(\beta, d)$ 
9: return  $\beta$ 

```

Remarks on Algorithm 9. The idea of this algorithm is that Δ_0 defined in [Lemma 6.4](#) not only gives a lower bound to $f(C_1, C_2)$; no matter how subsequent colors are added to (X_1, Y_1) and (X_2, Y_2) , $f(C_1, C_2)$ will not be lower than Δ_0 . To compute $g(C_1, C_2)$, we search all possibilities on subsequent colors by DFS, but once Δ_0 exceeds or equals to the current minimum value of $f(C_1, C_2)$, we do not need to perform the DFS for subsequent colors and can apply pruning. Also, since most of the calls to LOWERBOUND are identical (up to swapping colors), we can apply memoization to reduce redundant calculations of LOWERBOUND .

Utilizing symmetry. Next, we reduce the number of combinations by utilizing the “symmetry” that some state matrices represent essentially the same state. Specifically, the following operations on a state matrix M do not essentially change the state:

1. Swap two rows of M . (Corresponds to swapping C_i and C_j)

2. Swap two columns of M . (Corresponds to swapping the indices of two colors)
3. Choose one row, and for each element of the row, change 1 to 2 and 2 to 1. (Corresponds to swapping X_i and Y_i)

We say that M is in *standard form* if $(M_{1,1}, \dots, M_{4,1}, \dots, M_{1,m}, \dots, M_{4,m})$ is lexicographically earliest among the state matrices that can be obtained by repeating these three kinds of operations. It is easy to see that, once M becomes a matrix not in standard form, it will never be in standard form again after adding subsequent columns. In this case, we can apply pruning in the DFS.

We can check if M is in standard form by brute force. We brute-force the choice of how the rows are permuted and which rows we flip 1 and 2 in (there are $4! \times 2^4 = 384$ ways), and for each choice, we sort columns in lexicographical order to obtain a candidate of the standard form.

Results. In the DFS, we search 62195 state matrices that satisfy $\text{LB} < \frac{125}{112}$ and are in standard form. Among them, 22558 state matrices contain 1 and 2 in every row, and 16829 of them actually satisfy $\Delta_1 < \frac{125}{112}$.

6.5 The final piece: Potential decomposition

To further improve the analysis of the lower bound, we introduce the idea that the potentials in the first and second phases can be different. We consider decomposing potential ϕ as $\phi = \phi_A + \phi_B$. Let p_i, a_i, b_i ($i = 0, \dots, h$) be the average potentials ϕ, ϕ_A, ϕ_B of components after phase i . Then, the following equation holds due to the telescoping sum:

$$p_h - p_0 = (a_h - a_{h-1}) + (b_1 - b_0) + \sum_{i=1}^{h-1} \{(a_i - a_{i-1}) + (b_{i+1} - b_i)\}$$

The term $(a_i - a_{i-1}) + (b_{i+1} - b_i)$ represents the increase of ϕ_A at phase i plus the increase of ϕ_B at phase $i + 1$. Therefore, when merging C_1, C_2, C_3, C_4 as in subsection 6.3, if we know that the expected (increase of average ϕ_A in the first phase) + (increase of average ϕ_B in the second phase) is always x or more, we know that $p_k \geq xk - O(1)$. The new ‘‘potential increase’’ can be calculated by modifying Line 19 of Algorithm 8 to $\Delta \leftarrow \{\phi_B((X'_7, Y'_7)) - \frac{1}{2}(\phi_B((X'_5, Y'_5)) + \phi_B((X'_6, Y'_6)))\} + \{\frac{1}{2}(\phi_A((X'_5, Y'_5)) + \phi_A((X'_6, Y'_6))) - \frac{1}{4}(\phi_A(C_1) + \phi_A(C_2) + \phi_A(C_3) + \phi_A(C_4))\}$. The new algorithm is denoted as $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_A, \phi_B)$. In Subsection 6.3, we only considered the case that $\phi_A = \phi_B$, so by increasing the degrees of freedom, we can expect a better lower bound.

The potential setting. We consider setting ϕ_A, ϕ_B in the following way:

$$\begin{aligned} \phi_A(C) &= \frac{1}{2}|X \cap Y| + \begin{cases} \frac{17}{24} & ((|X \setminus Y|, |Y \setminus X|) = (2, 1), (1, 2)) \\ \frac{5}{6} & ((|X \setminus Y|, |Y \setminus X|) = (3, 1), (1, 3)) \\ \frac{1}{4}|X \oplus Y| & (\text{otherwise}) \end{cases} \\ \phi_B(C) &= \frac{1}{2}|X \cap Y| + \frac{1}{3}|X \oplus Y| \end{aligned}$$

Then, the following lemma holds:

Lemma 6.7. $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_A, \phi_B) \geq \frac{91}{96}$ for any components C_1, C_2, C_3, C_4 where state (X, Y) neither satisfies $X \subseteq Y$ nor $Y \subseteq X$.

We assume that in Lemma 6.7, the function $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_A, \phi_B)$ is calculated in the way explained in ‘‘Remarks on Algorithm 8’’ in subsection 6.3. We note that the result may change when we allow to choose a new color for $c_a^{(i)}$ when $X_{i+4} \setminus Y_{i+4}$ is not empty (and similarly for $c_b^{(i)}$), even if this is not an optimal strategy, because of the difference of ϕ_A and ϕ_B . Below, we give a computer assisted proof.

The searching framework. We prove [Lemma 6.7](#) in a similar way to [subsection 6.4](#). In the second phase, it can be proven that ϕ_B always increases by $\frac{1}{3}$, similarly to [Lemma 6.6](#). Therefore, we have to enumerate all the cases such that ϕ_A increases by at most $\frac{91}{96} - \frac{1}{3} = \frac{59}{96}$ in the first phase.

The lower-bounding for ϕ_A . We aim to calculate $g(C_1, C_2)$, defined in [subsection 6.4](#). However, the difference is that ϕ_A is no longer a linear function of $|X \oplus Y|$, so each $c \in X_i \oplus Y_i$ does not directly contribute to Δ_0 . In order to cope with this issue, we consider another potential function $\phi'_A(C) = \frac{1}{2}|X \cap Y| + \frac{1}{4}|X \oplus Y|$ (which is identical to $\frac{1}{2}\phi_1(C)$). Note that $\phi'_A(C) - \frac{1}{6} \leq \phi_A(C) \leq \phi'_A(C)$ always holds. Then, if we define Δ_0 for ϕ'_A (referring to [Table 4](#), the contribution to Δ_0 for each case (from the top) becomes $0, \frac{1}{8}, \frac{1}{8}, \frac{1}{4}, \frac{1}{8}, 0$), the increase of ϕ_A can be lower-bounded by $\Delta_0 - \frac{1}{6}$. Therefore, we can calculate $g(C_1, C_2)$ by changing Line 1 of [Algorithm 9](#) to $d_0 \leftarrow (\text{the value of } \Delta_0 \text{ for potential } \phi'_A \text{ for the current } C_1, C_2) - \frac{1}{6}$.

Results. In the DFS, we search 1773334 state matrices that satisfy $\text{LB} < \frac{59}{96}$ (where LB is defined for ϕ_A) and are in standard form. Among them, 700415 state matrices contain 1 and 2 in every row, and 415942 of them actually satisfy $\Delta_1 < \frac{59}{96}$ (where Δ_1 is defined for ϕ_A). We compute $\text{POTENTIALINCREASE}(C_1, C_2, C_3, C_4, \phi_A, \phi_B)$ for all of them, and each call returned a result of $\frac{91}{96}$ or more. In conclusion, it is shown that any randomized online coloring algorithm for bipartite graphs requires at least $\frac{91}{96} \log n - O(1)$ colors ([Theorem 6.1](#)).

Computer checks and programs. The programs used for the analysis and their results in this section can be downloaded at <https://github.com/square1001/online-bipartite-coloring>, which is the same URL as that in [section 5](#).

7 Conclusion

In this paper, we studied the online coloring of k -colorable graphs for $k \geq 5$, $k = 4$, and $k = 2$.

In [section 3](#), we presented a deterministic online algorithm to color k -colorable graphs with $\tilde{O}(n^{1-2/(k(k-1))})$ colors. The key was to create an online algorithm for locally ℓ -colorable graphs with $O(n^{1-2/(\ell(\ell-1)+2)})$ colors. The new algorithm also improved the competitive ratio of online coloring to $O(n/\log \log n)$.

In [section 4](#), we presented a deterministic online algorithm to color 4-colorable graphs with $O(n^{14/17})$ colors. The key was to make use of the second neighborhoods with the *double greedy method* that uses FIRSTFIT twice. We also applied the *Common & Simplify Technique* to take advantage of dense subgraph structures.

In [section 5](#), we showed that a randomization of the algorithm by Lovász, Saks, and Trotter [21] improves the performance to $1.034 \log_2 n + O(1)$ colors. We also showed that, in [section 6](#), no randomized algorithms can achieve $\frac{91}{96} \log_2 n - O(1)$ colors.

We were unable to improve the state-of-the-art bound $\tilde{O}(n^{2/3})$ colors for $k = 3$ by [18]. The main reason is that the graph of degree $n^{2/3}$ is too sparse to use the Common & Simplify technique, which we described in [subsection 4.5](#). Indeed, to achieve $\tilde{O}(n^{2/3-\epsilon})$ colors for some $\epsilon > 0$ with this technique, we must set parameters (α, β) satisfying $\alpha + \beta = 1 - \epsilon$, but the present technique requires $\tilde{O}(n^{2-\alpha-\beta})$ colors, which is much above the requirement. Especially, we are still unable to solve the special case where there exists a 3-coloring of G that for every $i \in \{1, 2, \dots, n/k\}$ ($k = n^{1/3-\epsilon}$), the $(i-1)k+1, (i-1)k+2, \dots, ik$ -th vertices (in the arrival order) have the same color.

Finally, we conclude this paper by highlighting several important open problems related to online graph coloring. We conjecture the following:

Conjecture 7.1. *There exists a deterministic online algorithm to color 3-colorable graphs with $\tilde{O}(n^{2/3-\epsilon})$ colors (for some $\epsilon > 0$).*

Conjecture 7.2. *There exists a deterministic online algorithm to color k -colorable graphs with $n^{1-1/o(k^2)}$ colors.*

Conjecture 7.3. *RANDOMIZEDLST uses at most $1.027 \log_2 n + O(1)$ colors for any bipartite graph G .*

Conjecture 7.4. *The optimal randomized online algorithm for coloring bipartite graphs is RANDOMIZEDLST, up to constant number of colors; therefore, no algorithm can achieve $1.026 \log_2 n - O(1)$ colors.*

We hope that future researchers (or possibly AIs) solve these questions.

Acknowledgments

The authors thank our laboratory members Kaisei Deguchi, Tomofumi Ikeura, and Yuta Inoue for their helpful suggestions for improving this paper.

References

- [1] Susanne Albers and Sebastian Schraink. Tight bounds for online coloring of basic graph classes. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPICS*, pages 7:1–7:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- [2] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.
- [3] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.
- [4] Dwight R Bean. Effective coloration. *The Journal of Symbolic Logic*, 41(2):469–480, 1976.
- [5] Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, and Lucia Keller. Online coloring of bipartite graphs with and without advice. *Algorithmica*, 70(1):92–111, 2014.
- [6] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Online edge coloring is (nearly) as easy as offline. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 36–46, 2024.
- [7] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Deterministic online bipartite edge coloring. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1593–1606. SIAM, 2025.
- [8] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Online edge coloring: Sharp thresholds. *to appear in FOCS 2025, arXiv preprint arXiv:2507.21560*, 2025.
- [9] Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959.
- [10] Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, 1998.
- [11] Grzegorz Gutowski, Jakub Kozik, Piotr Micek, and Xuding Zhu. Lower bounds for on-line graph colorings. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 507–515. Springer, 2014.

- [12] András Gyárfás and Jenö Lehel. On-line and first fit colorings of graphs. *Journal of Graph theory*, 12(2):217–227, 1988.
- [13] Magnús M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993.
- [14] Magnús M Halldórsson. Parallel and on-line graph coloring. *Journal of Algorithms*, 23(2):265–280, 1997.
- [15] Magnús M Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring. *SODA'92, Theoretical Computer Science*, 130(1):163–174, 1994.
- [16] Sandy Irani. Coloring inductive graphs on-line. *FOCS'90 and Algorithmica*, 11:53–72, 1994.
- [17] Ken-ichi Kawarabayashi, Mikkel Thorup, and Hirotaka Yoneda. Better coloring of 3-colorable graphs. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 331–339, 2024.
- [18] Hal A Kierstead. On-line coloring k-colorable graphs. *Israel Journal of Mathematics*, 105:93–104, 1998.
- [19] Hal A Kierstead. Coloring graphs on-line. *Online algorithms: the state of the art*, pages 281–305, 2005.
- [20] László Lovász. *Combinatorial Problems and Exercises*, volume 361. American Mathematical Society, 1979.
- [21] László Lovász, Michael Saks, and William T Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1-3):319–325, 1989.
- [22] Tim Roughgarden. *Beyond the worst-case analysis of algorithms*. Cambridge University Press, 2021.
- [23] Sundar Vishwanathan. Randomized online graph coloring. *FOCS'90 and Journal of algorithms*, 13(4):657–669, 1992.
- [24] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 222–227. IEEE Computer Society, 1977.