Distributed Partial Coloring via Gradual Rounding

Avinandan Das
Institut de Recherche en Informatique Fondamentale (IRIF),
CNRS and Université Paris Cité, France

Pierre Fraigniaud
Institut de Recherche en Informatique Fondamentale (IRIF),
CNRS and Université Paris Cité, France

Adi Rosén
Institut de Recherche en Informatique Fondamentale (IRIF),
CNRS and Université Paris Cité, France

Abstract

For $k \geq 0$, k-partial $(k+1)$-coloring asks to color the nodes of an n-node graph using a palette of $k+1$ colors such that every node v has at least $\min\{k, \deg(v)\}$ neighbors colored with colors different from its own color. Hence, proper $(\Delta+1)$-coloring is the special case of k-partial $(k+1)$-coloring when $k = \Delta$. Ghaffari and Kuhn [FOCS 2021] recently proved that there exists a deterministic distributed algorithm that solves proper $(\Delta+1)$-coloring of n-node graphs with maximum degree Δ in $O(\log n \cdot \log^2 \Delta)$ rounds under the LOCAL model of distributed computing. This breakthrough result is achieved via an original iterated rounding approach. Using the same technique, Ghaffari and Kuhn also showed that there exists a deterministic algorithm that solves proper $O(a)$-coloring of n-node graphs with arboricity a in $O(\log n \cdot \log^3 a)$ rounds. It directly follows from this latter result that k-partial $O(k)$-coloring can be solved deterministically in $O(\log n \cdot \log^3 k)$ rounds.

We develop an extension of the Ghaffari and Kuhn algorithm for proper $(\Delta+1)$-coloring, and show that it solves k-partial $(k+1)$-coloring, thus generalizing their main result. Our algorithm runs in $O(\log n \cdot \log^3 k)$ rounds, like the algorithm that follows from Ghaffari and Kuhn's algorithm for graphs with bounded arboricity, but uses only $k+1$ color, i.e., the smallest number c of colors such that every graph has a k-partial c-coloring. Like all the previously mentioned algorithms, our algorithm actually solves the general list-coloring version of the problem. Specifically, every node v receives as input an integer demand $d(v) \leq \deg(v)$, and a list of at least $d(v)+1$ colors. Every node must then output a color from its list such that the resulting coloring satisfies that every node v has at least $d(v)$ neighbors with colors different from its own. Our algorithm solves this problem in $O(\log n \cdot \log^3 k)$ rounds where $k = \max_v d(v)$. Moreover, in the specific case where all lists of colors given to the nodes as input share a common colors c^* known to all nodes, one can save one $\log k$ factor. In particular, for standard k-partial $(k+1)$-coloring, which corresponds to the case where all nodes are given the same list $\{1, \ldots, k+1\}$, one can modify our algorithm so that it runs in $O(\log n \cdot \log^2 k)$ rounds, and thus matches the complexity of Ghaffari and Kuhn's algorithm for $(\Delta+1)$-coloring for $k = \Delta$.

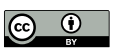
2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed graph coloring, partial coloring, weak coloring

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.30

Funding *Avinandan Das*: Additional support from ERC Consolidator Grant Distributed Biological Algorithms.
Pierre Fraigniaud: Additional support from the ANR Project ANR-20-CE48-0006 (DUCAT).
Adi Rosén: Most of the work of this author was done while with CNRS, FILOFOCS, Israel.

Acknowledgements We thank Alkida Balliu and Dennis Olivetti for useful discussions, and for pointing to us an error in a first draft of our result. We also thank Baruch Schieber for useful discussion in the early stage of this work. We finally thank the anonymous reviewers for their detailed comments.



1 Introduction

1.1 Partial Coloring

Proper coloring of the vertices of an arbitrary graph G with maximum degree Δ using $\Delta+1$ colors is one of the most studied symmetry-breaking problems in the context of distributed computing in networks [3, 17]. A natural generalization of proper coloring is *partial* coloring [2, 5, 11, 15]. Given two integers $k \geq 0$ and $c \geq 1$, k-partial c-coloring asks for a coloring of the vertices with colors in $\{1,\ldots,c\}$, such that every vertex v has at least $\min\{k, \deg(v)\}$ neighbors with a color different from its own color. In particular, k-partial $(k+1)$-coloring is equal to proper $(\Delta+1)$-coloring for $k=\Delta$, 0-partial coloring is trivial, and, for $0<k<\Delta$, k-partial $(k+1)$-coloring relaxes the requirement of proper coloring for vertices of degree larger than k. The case $k=1$ is referred to as *weak* coloring in [15].

Note that for every integer $k \geq 0$ there exists a k-partial $(k+1)$-coloring of G. Such a coloring can be constructed by a simple centralized greedy algorithm as follows. Initialize all vertices with color 1, and consider all vertices sequentially, one by one. For each considered vertex v, let $C(v)$ be the set of colors present in the neighborhood of v. If $|C(v)| = k+1$ then v keeps color 1, otherwise it recolors itself with an arbitrary color in $\{1,\ldots,k+1\} \setminus C(v)$. The resulting coloring is k-partial because (1) node v has at least $\min\{k, \deg(v)\}$ neighbors with a color different from its own color when it adopts its final color, and (2) two neighboring vertices with different colors at some time t during the execution of the greedy algorithm will remain with different colors at any time $t' \geq t$. This paper studies the design of a *distributed, deterministic* algorithm for k-partial $(k+1)$-coloring, that works for every $k \geq 0$.

We consider the standard LOCAL model of distributed computing [13]. This model assumes an n-node network modeled as a graph $G=(V,E)$, where each node is a computer, and the nodes communicate by exchanging messages along the edges of the graph. Each node is assigned an identifier in $\{1,\ldots,n^c\}$, for some $c \geq 1$, which is unique in the network. Initially, every node knows solely its identifier, the range of identifiers, and potentially some inputs, e.g., a non-negative integer k in the case of k-partial coloring. All nodes execute the same algorithm, which proceeds in synchronous rounds. At each round, every node sends one message to each of its neighbors, receives the messages from its neighbors, and performs some individual computation. After a certain number of rounds, every node outputs, and terminates. In the case of coloring, every node must eventually output a color in a prescribed range of colors, e.g., $\{1,\ldots,k+1\}$.

1.2 Previous Work

For many years, the best known (deterministic) algorithms for proper $(\Delta+1)$-coloring graphs with maximum degree Δ were running in essentially $2^{O(\sqrt{\log n})}$ rounds in n-node networks [1, 16]. This was the state of the art for almost a quarter of a century, and it is only a few years ago that an algorithm for $(\Delta+1)$-coloring running in a polylogarithmic number of rounds was proposed [18]. All these algorithms were based on a specific graph decomposition, and the efficient algorithm in [18] actually shows how to compute such a decomposition in a polylogarithmic number of rounds. Recently, a breakthrough result has been achieved, stating that $(\Delta+1)$-coloring can be solved in $O(\log n \cdot \log^2 \Delta)$ rounds [10], without using graph decomposition. Moreover, for large Δ, the algorithm sill runs in $O(\log^3 n)$ rounds – this has been very recently improved to $\tilde{O}(\log^2 n)$ rounds [8]. There are indeed algorithms for $(\Delta+1)$-coloring running in $\widetilde{O}(\sqrt{\Delta}) + O(\log^* n)$ rounds (see [4, 7, 14]). However, such algorithms are efficient for small Δ only, that is, their complexities are polylogarithmic in n only when Δ is itself growing polylogarithmically with n.

The algorithm in [10] is based on an original gradual rounding technique which, starting from a uniform selection of a color by each node, produces a non-necessarily proper coloring which leaves only a linear number of edges monochromatic (i.e., with both end-points having the same color). This technique has even been extended to computing a maximal independent set (MIS) [6], and to computing network decomposition [9]. In the present paper we show that the $O(\log n \cdot \log^2 \Delta)$-round $(\Delta+1)$-coloring algorithm from [10] can be modified to solve the generalized partial coloring problem. We note the gradual rounding technique can also be applied to coloring graphs with given *arboricity* a. Indeed, it was shown [10] that there exists a deterministic algorithm that solves proper $O(a)$-coloring of n-node graphs with arboricity a in $O(\log n \cdot \log^3 a)$ rounds. As a consequence, k-partial $O(k)$-coloring can be solved deterministically in $O(\log n \cdot \log^3 k)$ rounds: one can let every node v pick $\min\{k, \deg(v)\}$ incident edges arbitrarily, and then remove from the input graph G all edges that were not picked. The resulting graph G' has *degeneracy* at most $2k$ (simply because it has at most kn edges, and therefore minimum degree $2k$), and therefore it has arboricity at most $2k$ as well. By running in G' the algorithm from [10] for proper coloring graphs with bounded arboricity, we get that a k-partial $O(k)$-coloring of G', and therefore of G too, in $O(\log n \cdot \log^3 k)$ rounds. The question remains however whether one can reduce the number of colors from $O(k)$ to $k+1$. Indeed, for every $k \geq 1$, $k+1$ is the smallest number c of colors such that every graph has a k-partial c-coloring.

Note that the coloring algorithms in [10] actually solve the general list-coloring versions of the various problems. In the proper c-list-coloring problem, every node is also given a list of at least c colors as input, and its output color is bounded to belong to its list. We also slve the list-coloring version of partial coloring. In addition, we allow the *demand* of the nodes to vary, i.e., every node can have a different demand regarding how many of its neighbors must have a different color than its own.

Under restricted hypotheses, fast partial coloring algorithms exists. For instance, in regular graphs, there is a $O(\log^\star n)$-round deterministic algorithm for k-partial 3-coloring whenever $\Delta \geq 3k-4$ and $k \geq 3$, and for k-partial k-coloring whenever $\Delta \geq k+2$ and $k \geq 4$ (see [2]). Moreover, the same paper shows that computing 2-partial 2-coloring in Δ-regular graphs requires $\Omega(\log n)$ rounds deterministically, and $\Omega(\log \log n)$ rounds randomized, for any $\Delta \geq 2$. More generally, computing a k-partial c-coloring in Δ-regular graphs with $k \geq \frac{\Delta(c-1)}{c}+1$ requires $\Omega(\log n)$ rounds deterministic, and $\Omega(\log \log n)$ rounds randomized [2]. It was recently shown [5] that 1-partial 2-coloring in odd-degree graphs cannot be solved in $o(\log^\star \Delta)$ rounds, thereby providing a matching lower bound to the 30-year old upper bound in [15]. Finally, k-partial $O(k^2)$-coloring as well as d-defective $O(\Delta^2/d^2)$-coloring can both be computed in $O(\log^* n)$ rounds [11].

1.3 Our Results

For a non-negative integer k, the k-partial list-coloring problem in a graph $G=(V,E)$ is defined as follows.

Input of node v: A demand $d(v) \in \mathbb{N}$ satisfying $d(v) \leq \deg(v)$, a list $L(v)$ of colors, of size at least $d(v)+1$, and the value $k = \max_{v \in V} d(v)$.

Output of node v: A color $\delta(v) \in L(v)$ such that

$$|\{u \in V : \{u,v\} \in E \wedge \delta(v) \neq \delta(u)\}| \geq d(v)$$

.

The k-partial $(k+1)$-coloring problem is the special case of k-partial list-coloring problem in which every node has demand $\min\{k, \deg(v)\}$, and all lists are all equal to $\{1, \ldots, k+1\}$. Our main results are the following.

▶ **Theorem 1.** *There exists a deterministic distributed algorithm solving k-partial list-coloring in all n-node networks, and running in $O(\log n \cdot \log^3 k)$ rounds under the* LOCAL *model.*

Note that the round-complexity of our algorithm scales as a function of the upper bound k on the demands, and not as a function of Δ. In particular, for $k = O(1)$, our algorithm runs in just $O(\log n)$ rounds, even in graphs with large maximum degrees. Also note that even if it is possible to design faster algorithms for specific values of the demand, e.g., for 1-partial 2-coloring, a.k.a. *weak coloring* [15], our algorithm is generic, and works for all $k \geq 0$.

If the lists given to the nodes satisfy $\cap_{v \in V} L(v) \neq \varnothing$, and the nodes are given a color $c^* \in \cap_{v \in V} L(v) \neq \varnothing$ as input, one can modify our algorithm so as to solve k-partial list-coloring in $O(\log n \cdot \log^2 k)$ rounds. An important application is k-partial $(k+1)$-coloring in which every node has demand $\min\{k, \deg(v)\}$, and all lists are all equal to $\{1, \ldots, k+1\}$. Indeed, $1 \in \cap_{v \in V} L(v)$ in k-partial $(k+1)$-coloring, and this holds even if, for every node v, $L(v) = \{1, \ldots, d(v)+1\}$ for a demand $d(v)$.

▶ **Theorem 2.** *There exists a deterministic distributed algorithm solving k-partial $(k+1)$-coloring in all n-node networks, and running in $O(\log n \cdot \log^2 k)$ rounds under the* LOCAL *model.*

1.4 Our Techniques

We base ourselves on the main ideas of the algorithm by Ghaffari and Kuhn [10] for $(\Delta+1)$-coloring, which actually works for list-coloring as well, and on the core rounding procedure of that algorithm. The algorithm in [10] proceeds in $O(\log n)$ iterations of a procedure whose objective is to fix the color of a constant fraction of the remaining uncolored nodes. Specifically, at each iteration, if $G_i = (V_i, E_i)$ denotes the subgraph of the input graph G whose nodes are still uncolored at the ith iteration, the procedure starts from a fractional coloring of G_i, where each color $c \in L_i(v)$ in the current list of colors assigned to every node $v \in V_i$ appears with "weight" $1/|L_i(v)|$. Given this fractional coloring, the procedure produces, in $O(\log^2 \Delta)$ rounds, an integral coloring γ_i of the nodes of G_i, that is, for every $v \in V_i$, all colors $c \in L_i(v)$ but one have weight 0, and the remaining color has weight 1. This coloring γ_i is not necessarily proper. However, it is shown (see Corollary 3.6 in [10]) that the total number of monochromatic edges F_i in E_i is $O(|V_i|)$. As a consequence, a constant fraction of the nodes in the graph $G'_i = (V_i, F_i)$ have degrees upper bounded by a constant, from which it follows that a maximal independent set (MIS) I_i in the subgraph H_i of G'_i induced by these low degree nodes can be computed in $O(\log^\star n)$ rounds [12]. Moreover, since the maximum degree of H_i is bounded, there is a constant fraction of its nodes in I_i. The nodes in I_i adopt their colors given by γ_i, and terminate. The remaining nodes update their lists by removing the colors adopted by neighboring nodes that terminated, and carry on with yet another iteration. The algorithm in [10] uses other clever tricks, but the above summarizes the core of the algorithm, which is the part that we shall modify for handling partial coloring.

Our first change is in the preprocessing stage where we compute a k-partial $O(k^2)$-coloring α of the graph and then remove all monochromatic edges. Since the computed coloring α is k-partial, the removal of the monochromatic edges still leaves us with a graph on which a k-partial coloring is a k-partial coloring of the original graph. On the other hand we have a proper coloring of the remaining graph, a condition that is needed to apply the core procedure in [10].

At the start of each iteration i, a preprocessing phase selects arbitrarily, for each node, a number of "outgoing" edges equal to the size of this node's color list (or demand), and removes all edges that were not selected by any of their endpoints. The remaining graph $G_i = (V_i, E_i)$ has the property that a partial coloring on it is also a corresponding partial coloring on the original graph (for that iteration), and the (remaining) degree of each node is not bigger that the size of its (remaining) list of colors. The rounding procedure of Ghaffari and Kuhn [10] is applied on G_i, providing each node $v \in V_i$ with a color $\gamma_i(v) \in L_i(v)$. We then introduce the notion of *saturated* nodes w.r.t. γ_i, which are nodes v such that, for each color c in their lists, they have a neighbor u_c colored c by γ_i. These nodes might get "saturated" with respect to the following consideration. Let S_i denotes the set of saturated nodes, and let $v \in S_i$. If a neighbor u_c of v, colored c, belongs to the MIS computed at iteration i, it is expected that u_c terminates with color c, and v removes c from its list $L_i(v)$ so that to eventually provide u_c with a neighbor with a color different from c. But if v is saturated, this would result in exhausting all possible colors in the list of $L_i(v)$, preventing v from eventually choosing a color during further iterations.

To overcome the issue of saturated nodes, we proceed differently from [10], by first letting all nodes that are saturated w.r.t. γ_i adopting their current color, and terminate. Then, we also introduce the notion of *idle* nodes. Roughly, an idle node is a node which is not saturated, but which has so many saturated neighbors with potentially the same color as its own color that it has not sufficiently many other neighbors for guaranteeing that its demand will eventually be satisfied. In our algorithm, idle nodes abort the search for a color during the remaining iterations. They do not participate to the subsequent iterations, but update their demand and list of colors according to the colors picked by neighboring (active) nodes. Finally, if K_i denotes the set of nodes that become idle at the ith iteration, then all nodes in $I_i \setminus (S_i \cup K_i)$, that is, all nodes in the MIS that are neither saturated nor idle, adopt their current colors. In this way, k-partial coloring is guaranteed for the non-idle nodes, and each of the $O(\log n)$ iterations performs in $O(\log^2 k)$ rounds.

After all iterations are completed, it remains to assign colors to the nodes which became idle during some of the iterations, during a post-processing stage. We note that the subgraph induced by the idle nodes can be viewed as "layered", where the nodes that became idle at iteration i form layer K_i of the subgraph, $i = 1, \ldots, O(\log n)$. We then show that the preconditions required for applying Lemma 5.4 in [10], which provides an algorithm for proper coloring layered graphs, are fulfilled (in particular, the degree towards the upper layers is bounded by k). Indeed, it occurs that the threshold for the number of saturated neighbors, which defines idle nodes, precisely corresponds to the ability to properly color the idle nodes. By application of Lemma 5.4 in [10], this post-processing stage runs in $O(\log n \cdot \log^3 k)$ rounds. Note that, up to the post-processing stage, our algorithm runs in $O(\log n \cdot \log^2 k)$ rounds as each of the $O(\log n)$ iterations consumes $O(\log^2 k)$ rounds, and the extra $\log k$ factor is only due to the post-processing stage.

Partial Coloring

Assuming that all lists contain a common color c^*, which fits with the setting of k-partial $(k+1)$-coloring, enables to simplify the coloring procedure performed at each of the $O(\log n)$ iterations of the algorithm, and to avoid the costly post-processing stage. Roughly, we can avoid introducing the notion of idle nodes as follows. The color c^* is removed from all lists, and will play the role of a backup color in case a node has exhausted all the colors in its list for satisfying neighbors. Specifically, as in [10], we serve first all the nodes in the maximal independent set (MIS) I_i computed at iteration i, i.e., for every $v \in I_i$, node v adopts its

color $\gamma_i(v)$. Then, if a node v gets saturated (i.e., all the colors in its list are picked by at least one of its neighbor, which also happens to be in the MIS), then v adopts color c^* – recall that c^* was removed from all lists. Letting the saturated nodes adopting color c^* does not cause other nodes to saturate at further iterations even if it may seem that a node v with many neighbors that have fixed their color to c^* at previous iterations might be prevented from using color c^* itself (in case it becomes saturated). Indeed, we show that while the number of neighbors of a node v may decrease at a given iterations as these neighbors adopt color c^* and terminate, the set of colors currently in the list of that node v is not affected by these neighbors. As a consequence, we can show that if a node has "many" neighbors colored c^* then the size of its list actually becomes larger than its current degree, and thus this node cannot become saturated anymore. This guarantees the correctness of the algorithm. Finally, avoiding the post-processing stage enables saving one $\log\ k$ factor in the round complexity, since each iteration performs in $O(\log^2 k)$ rounds. Due to lack of space, the details are moved to Appendix A.

2 The Algorithm

We describe our distributed algorithm for solving partial list-coloring in an arbitrary graph. The algorithm is to be executed in the LOCAL model by the nodes of an n-node graph $G = (V, E)$. Due to lack of space, we do not re-explain some of the features of the algorithm in [10], but mostly focus on the parts that differ significantly from this latter algorithm for extending it from $(\Delta + 1)$-list coloring to k-partial list coloring. Our algorithm proceeds as follows.

2.1 Preprocessing Stage

The nodes compute a k-partial $O(k^2)$-coloring of G, which we denote by α. This can be done in $O(\log^* n)$ rounds (see [11]). Then, the nodes remove the monochromatic edges to obtain a sub-graph of G, denoted by $G_1 = (V_1, E_1)$, where $V_1 = V$. Note that in G_1 the degree of any node $v \in V$ is at least $\min\{k, \deg_G(v)\}$, and therefore $d(v) \leq \deg_{G_1}(v)$. Note also that α is a proper $O(k^2)$-coloring of G_1.

2.2 Core of the Algorithm

The coloring produced by our algorithm is denoted $\delta(\cdot)$. Initially, $\delta(v) = \perp$ for every $v \in V$. For computing their colors, the nodes perform $O(\log n)$ iterations of a series of phases, called preprocessing, derandomization, color assignment, and update. At each iteration, some nodes $v \in V$ fix their colors $\delta(v) \in L(v)$. Once a node v becomes colored (i.e., it adopts a color $\delta(v) \neq \perp$), its color never changes. At each iteration, we classify the nodes of G into three categories: *active, idle*, and *terminated*.

- A terminated node v is a node that has adopted its color $\delta(v)$.
- An idle node is a node that has not yet adopted its color, but it stops participating in the subsequent iterations after it became idle. However, its demand, as well as its list of colors are updated in each subsequent iteration.
- An active node is a node that is neither idle nor terminated (such a node will carry on to the next iteration).

An active node may, in a subsequent iteration, be colored (in which case it becomes terminated), or become idle. Only active nodes proceed with another iteration. An idle node remains idle until the post-processing stage starts[1]. When this is the case, no more iterations are performed, and the idle nodes perform a specific post-processing stage of computation for finalizing their colors (this stage to be described further in the text, in Section 2.3). Once this is done, the algorithm terminates. Initially, i.e., before the first iteration starts, all nodes are active.

We now describe one iteration of the algorithm. The input to iteration $i \geq 1$ is a graph $G_i = (V_i, E_i)$, a demand function $d_i : V_i \to \mathbb{N}$, and a color list function $L_i : V_i \to 2^{\mathbb{N}}$. If, for a node v still active at iteration i, $|L_i(v)| > d_i(v) + 1$, then v prunes its list arbitrarily for keeping exactly $d_i(v) + 1$ colors in its list. For every node v, we set $d_1(v)$ as the input demand $d(v)$, and $L_1(v)$ as the input list $L(v)$.

2.2.1 Preprocessing Phase

Every vertex v arbitrarily selects $d_i(v)$ many edges incident to it – we shall show later, in the proof of correctness, that v has sufficiently many edges for performing this selection. Note that an edge can be selected by one or two of its endpoints. Any non-selected edge is removed from G_i. In what follow, we slightly abuse notation, and $G_i = (V_i, E_i)$ still refers to the actual graph after the removal of the non-selected edges. Note that G_i has average degree at most $2k$, and therefore $O(kn)$ edges.

2.2.2 Derandomization Phase

Every vertex v picks an arbitrary set $L'_i(v) \subseteq L_i(v)$ of $2^{\lfloor \log |L_i(v)| \rfloor}$ colors. Observe that $\frac{|L_i(v)|}{2} \leq |L'_i(v)| \leq |L_i(v)|$. For any set $p = \{p_v : v \in V_i\}$ of random distributions over the lists $L'(v)$, $v \in V_i$, and for $L = \cup_{v \in V} L'_i(v)$, we define a weight function $w_p : (V_i \times L)^2 \to [0, 1]$ as follows:

$$w_p\big(\{(u, a), (v, b)\}\big) = \begin{cases} 0 & \text{if } \{u, v\} \notin E_i \\ 0 & \text{if } \{u, v\} \in E_i \text{ and } a \neq b \\ p_u(a) \cdot p_v(b) & \text{if } \{u, v\} \in E_i \text{ and } a = b \end{cases}$$

Moreover, we define

$$W(p) = \sum_{((u,a),(v,b)) \in (V_i \times L)^2} w_p\big((u, a), (v, b)\big).$$

Corollary 3.6 in [10] provides a way to "derandomize" the uniform distribution p^{unif}, defined as

$$p_v^{\text{unif}}(c) = \frac{1}{|L'_i(v)|}$$

for every node v and every color $c \in L'_i(v)$, while almost preserving $W(p)$. We use Corollary 3.6 in [10] with $\epsilon = 1$, using the proper $O(k^2)$-coloring α computed during the pre-processing stage (cf. Section 2.1), and observing that our node "labeling" uses at most $2^{\lfloor \log(k+1) \rfloor}$ colors.

[1] Note that since all nodes know $k = \max_{v \in V} d(v)$ and the (polynomial) range of IDs, they can individually compute how long lasts each phase, each iteration, and each stage of the algorithm. In particular, a node becoming idle at a given iteration knows when it has to start executing the post-processing stage.

▶ **Lemma 3** (Corollary 3.6 in [10]). *A coloring γ_i satisfying $\gamma_i(v) \in L'_i(v)$ for every $v \in V_i$ can be computed in $O(\log^2 k)$ rounds, such that $W(p^{\gamma_i}) \leq 2 \cdot W(p^{\text{unif}})$, where the distribution $p^{\gamma_i} = (p_v^{\gamma_i})_{v \in V_i}$ is defined as*

$$p_v^{\gamma_i}(c) = \begin{cases} 1 & \text{if } \gamma_i(v) = c \\ 0 & \text{otherwise .} \end{cases}$$

2.2.3 Color Assignment Phase

Given the graph $G_i = (V_i, E_i)$, and the coloring γ_i from Lemma 3, let us consider the graph $G'_i = (V_i, F_i)$ where $F_i \subseteq E_i$ is the set of monochromatic edges w.r.t γ_i. As we will prove later in the proof of correctness (see Lemma 10), we have $|F_i| \leq 4 \cdot |V_i|$. As a consequence, the number of nodes $v \in V_i$ with $\deg_{G'_i}(v) \geq 16$ is at most $|V_i|/2$. Let us denote by H_i the subgraph of G'_i induced by the nodes with degree less than 16 in G'_i. By construction, H_i has at least $|V_i|/2$ nodes (again, see Lemma 10), and it has maximum degree at most 15. The facts that H_i has bounded degree, and that it has a $O(k^2)$ proper coloring thanks to the pre-processing stage imply that its nodes can compute a maximal independent set (MIS) I_i in H_i in $O(\log^\star k)$ rounds [12] . Note that, since each of the at least $|V_i|/2$ nodes in H_i has maximum degree 15, we have $|I_i| \geq \frac{|V_i|}{32}$.

We now introduce a notion that plays an important role in our algorithm. For every node v, let $N_{G_i}(v)$ denote the set of v's neighbors in G_i.

▶ **Definition 4.** *A node $v \in V_i$ is* saturated *with respect to the coloring γ_i if, for every $c \in L_i(v)$, there exists a neighbor $u \in N_{G_i}(v)$ with $\gamma_i(u) = c$.*

Note that some nodes in the maximal independent set may be saturated, whereas some other nodes in the independent set may not be saturated, and the same holds for nodes outside the independent set. At this point, nodes may turn from active to idle or terminated according to one of the following three rules.

- **Saturated Node Rule:** Every saturated node w.r.t. γ_i fixes its final color as $\delta(v) = \gamma_i(v)$, and terminates. Let us denote by S_i the set of nodes that become saturated w.r.t. γ_i at the ith iteration.
- **Idle Node Rule:** For every node $v \in V_i \setminus S_i$, if

$$|N_{G_i}(v)| - |\{u \in N_{G_i}(v) \cap S_i \ : \ \gamma_i(u) = \gamma_i(v)\}| < d_i(v) \tag{1}$$

  then v becomes idle. Let us denote by K_i the set of nodes that become idle at the ith iteration.
- **MIS Node Rule:** Every vertex $v \in I_i \setminus (S_i \cup K_i)$ fixes its final color as $\delta(v) = \gamma_i(v)$, and terminates. We denote by J_i the set of non-saturated node nodes in the independent set that terminate at iteration i, i.e., $J_i = I_i \setminus (S_i \cup K_i)$.

Remark. The intuition guiding the above three rules is that a saturated node v has all colors in $L_i(v)$ present its neighborhood and hence it has at least $d_i(v)$ neighbors with colors, in γ_i, different than $\gamma_i(v)$. It can therefore safely set its final color to its current color: for each neighbor either that neighbor adopts its current (different) color, or will be "instructed" not to use v's color is subsequent iterations. On the other hand, nodes satisfying Eq. (1) are nodes that are in some sense "stuck" because they have too many neighbors with the same color as their own color, which do not leave them enough "space" for accommodating their demand. This is why the algorithm "freezes" them, as idle nodes, to be treated after the

termination of all iterations, during the post-processing stage (see Section 2.3). Finally, nodes in the MIS fix their colors. For making sure that its demand will be eventually satisfied, each MIS node will require its neighbors to remove its color from their lists (see the updating phase hereafter). Here comes the main reason of introducing saturated nodes. It may indeed be the case that a node v is surrounded by MIS nodes, with colors covering its list entirely. All these MIS nodes will prevent v from using any of its available colors, resulting in color starvation. For avoiding this, a saturated node fixes its color anyway, and terminates. We shall see that this does not prevent its adjacent MIS nodes to have their demands eventually satisfied. Essentially, the reason is that if they could not be satisfied, then they would have become idle.

2.2.4 Updating Phase

Towards the next iteration we perform a series of list and demand updates, applying to both the active *and* the idle nodes. Let us denote by K the entire set of idle nodes, i.e., $K = \bigcup_{j=1}^{i} K_j$.

List Update. The lists are updated as follows:

- For every active node v, i.e., for every $v \in V_i \setminus (S_i \cup K_i \cup J_i)$,

$$L_{i+1}(v) = L_i(v) \setminus \{\delta(u) \ : \ u \in N_{G_i}(v) \cap (S_i \cup J_i)\}.$$

- For every idle node v, i.e., for all $v \in K$ (and not only $v \in K_i$),

$$L_{i+1}(v) = L_i(v) \setminus \{\delta(u) \ : \ u \in N_{G_1}(v) \cap (S_i \cup J_i)\}.$$

  Note that for idle nodes we consider $u \in N_{G_1}(v)$, and not only $u \in N_{G_i}(v)$, since $v \in K$ may have became idle before iteration i, say at iteration $j < i$, i.e., $v \in K_j$, and thus v may not belong to V_i.

Demand Update. The demands are updated as follows:

- For every active node v, i.e., for every $v \in V_i \setminus (S_i \cup K_i \cup J_i)$,

$$d_{i+1}(v) = \max\{0, \ d_i(v) - |N_{G_i}(v) \cap (S_i \cup J_i \cup K_i)|\}.$$

- For every idle node v, i.e., for all $v \in K$ (and not only for those in K_i),

$$d_{i+1}(v) = \max\{0, \ d_i(v) - |N_{G_1}(v) \cap (S_i \cup J_i)|\}.$$

  Note again that we consider for idle nodes $N_{G_1}(v)$, and not only $N_{G_i}(v)$ for the same reasons as above.

Graph Update. The graph G_i is updated to $G_{i+1} = (V_{i+1}, E_{i+1})$ as follows:

- $V_{i+1} = V_i \setminus (S_i \cup J_i \cup K_i)$
- $E_{i+1} = \big\{\{u, v\} \in E_i \ : u \in V_{i+1} \wedge v \in V_{i+1}\big\}$

The nodes that remain active at the end of the iteration proceed to the next iteration.

2.3 Post-Processing Stage

After all active nodes eventually become either terminated (with their final color) or idle, it remains to provide the idle nodes with a color. Every idle node belongs to one of the set K_i, $1 \leq i \leq T$, where T is the number of iterations of the four phases described in Section 2.2 – we shall prove later that $T = O(\log n)$. Therefore, for $K = \cup_{i=1}^{T} K_i$, the graph $G_1[K]$ induced by the idle vertices can be viewed as *layered*, with layers $K_1, \ldots, K_T$. Every idle node has a list of available colors equal to $L_{T+1}(v)$, as computed during the updating phase of the last iteration T. We will apply Lemma 5.4 in [10] with $h = T$, $\hat{\Delta} = k$, and the lists $M(v) = L_{T+1}(v)$ for all $v \in K$. Let $i \in \{1, \ldots, T\}$, and let $v \in K_i$. We denote by $\deg^+(v)$ the number of neighbors of v belonging to layers $K_i, \ldots, K_T$.

▶ **Lemma 5** (Lemma 5.4 in [10]). *Let us consider a layered graph $G_1[K]$ with h layers $K_1, \ldots, K_h$ such that, for every node $v \in K$, v is assigned a list of colors $M(v)$ with $|M(v)| > \deg^+(v)$. Let $\hat{\Delta} = \max_{v \in K} \deg^+(v)$. There is a deterministic distributed algorithm that solves proper list-coloring with input lists M in $G[K]$, running in $O(\log n \log^2 \hat{\Delta} + h \log^3 \hat{\Delta})$ rounds.*

In the proof of correctness, we shall show that the conditions in Lemma 5 hold for the post-processing stage with $\hat{\Delta} = k$. Every idle node adopts the color returned by the proper list-coloring algorithm in Lemma 5.

This completes the description of our algorithm. In the next section, we shall prove its correctness.

3 The Proof

We first prove that our algorithm does solve k-partial list-coloring. Next, we will prove that it runs in the prescribed number of rounds.

3.1 Proof of Correctness

The lemma below shows the correctness of the preprocessing stage described in Section 2.1.

▶ **Lemma 6.** *The k-partial list-coloring problem is well defined on the graph $G_1 = (V_1, E_1)$. Moreover, any k-partial list-coloring on the graph $G_1 = (V_1, E_1)$ is also a k-partial list-coloring on the original graph $G = (V, E)$.*

Proof. To show that the k-partial list-coloring problem is well defined on the graph G_1, we must show that, for every node, its demand does not exceed its degree (in G_1). Recall that the graph $G_1 = (V_1, E_1)$ (i.e., the graph after the preprocessing stage) satisfies $V_1 = V$, so the demands and lists are defined on the same set of vertices. Let $v \in V$ be a vertex. We have $d(v) \leq \deg_G(v)$, $d(v) < |L(v)|$, and $d(v) \leq k = \max_{u \in V} d(u)$.

- If $\deg_G(v) \leq k$, then none of the edges incident to v in G is monochromatic in the k-partial coloring α of G computed in the preprocessing phase (see Section 2.1). It follows that $\deg_{G_1}(v) = \deg_G(v)$.
- If $\deg_G(v) > k$, then $\deg_{G_1}(v) \geq k$ as the coloring α computed in the preprocessing phase is a k-partial coloring of G.

Therefore, either $\deg_{G_1}(v) = \deg_G(v)$ or $\deg_{G_1}(v) \geq k$. As a consequence, $d(v) \leq \deg_{G_1}(v)$, as claimed. Now, let ψ be a k-partial list-coloring on the graph G_1. We have $\psi(v) \in L(v)$, and v has at least $d(v)$ neighbors in G_1 with another color. Since $E_1 \subseteq E$, v has also at least $d(v)$ neighbors in G with another color. Therefore ψ is a k-partial list-coloring on the original graph G. ◀

By Lemma 6, it is sufficient to prove the correctness of the algorithm on the graph G_1 (i.e., the graph after the preprocessing stage). Let us start by establishing a series of invariants which hold throughout the execution of the algorithm. The following holds by construction.

▶ **Observation 7.** *For every iteration $i \geq 1$, S_i, J_i, K_i, and V_{i+1} form a partition of V_i.*

The next lemma shows that the conditions necessary for partial list-coloring are preserved.

▶ **Lemma 8.** *For every iteration $i \geq 1$, and every node $v \in V_i$, we have that $d_i(v) \leq \deg_{G_i}(v)$, and $d_i(v) < |L_i(v)|$.*

Proof. We prove the lemma by induction on the iteration number, i. The basis of the induction holds for $i = 1$ by Lemma 6. Let us assume that the lemma holds for $i \geq 1$, and let $v \in V_{i+1}$. We have $V_{i+1} = V_i \setminus (S_i \cup J_i \cup K_i)$, and

$$d_{i+1}(v) = \max\{0,\ d_i(v) - |N_{G_i}(v) \cap (S_i \cup J_i \cup K_i)|\},$$

from which it follows that $d_{i+1}(v) \leq \deg_{G_{i+1}}(v)$. For the list update, we have

$$L_{i+1}(v) = L_i(v) \setminus \{\delta(u) : u \in N_{G_i}(v) \cap (S_i \cup J_i)\}.$$

Therefore, since

$$|\{\delta(u) : u \in N_{G_i}(v) \cap (S_i \cup J_i)\}| \leq |N_{G_i}(v) \cap (S_i \cup J_i \cup K_i)|,$$

we have $d_{i+1}(v) < |L_{i+1}(v)|$. Thus the lemma holds for $i+1$, which completes the proof. ◀

The idle nodes carry on updating their lists and demands after they become idle, but the invariant $d_i(v) < |L_i(v)|$ remains true for idle nodes too, as shown below.

▶ **Lemma 9.** *Let $i \geq 1$, and let $v \in K_i$. For every $j \geq i$, $d_j(v) < |L_j(v)|$.*

Proof. Let us fix $i \geq 1$. We prove the claim by induction on $j \geq i$. Since $v \in K_i$, the base case holds by Lemma 8, as $v \in V_i$. For the induction step, let us assume that $d_j(v) < |L_j(v)|$, for $j \geq i$. We have

$$L_{j+1}(v) = L_j(v) \setminus \{\delta(u) : u \in N_{G_1}(v) \cap (S_j \cup J_j)\}.$$

and

$$d_{j+1}(v) = \max\{0,\ d_j(v) - |N_{G_1}(v) \cap (S_j \cup J_j)|\},$$

Therefore, since

$$|\{\delta(u) : u \in N_{G_1}(v) \cap (S_j \cup J_j)\}| \leq |N_{G_1}(v) \cap (S_j \cup J_j)|,$$

we get that $d_{j+1}(v) < |L_{j+1}(v)|$, as desired. ◀

We now prove that the number of iterations performed during the core of the algorithm (see Section 2.2) is finite. We start with the following lemma that we obtain by modifying a similar proof from [10].

▶ **Lemma 10.** *Let $i \geq 1$. The graph $G'_i = (V_i, F_i)$ induced by the monochromatic edges in $G_i = (V_i, E_i)$ w.r.t. γ_i has a linear number of edges. Specifically, $|F_i| \leq 4\,|V_i|$.*

Proof. Let us analyze the total "weight" $W(p^{\text{unif}})$ of the uniform distribution p^{unif} before applying rounding (all notations except one are defined at the beginning of Section 2.2.2). We introduce just one new notation: $N_{G_i}^{select}(v)$ denotes the set of selected neighbours of v according to the operation performed in Section 2.2.1 where v selects $d_i(v)$ many edges incident to it. We have

$$\begin{aligned} W(p^{\text{unif}}) &= \sum_{(u,a),(v,b)\in(V_i\times L)^2} w_{p^{\text{unif}}}((u,a),(v,b)) \quad = \sum_{\substack{\{u,v\}\in E_i \\ c\in L_i'(u)\cap L_i'(v)}} p_u(c)\cdot p_v(c) \\ &= \sum_{\substack{\{u,v\}\in E_i \\ c\in L_i'(u)\cap L_i'(v)}} \frac{1}{|L_i'(u)|}\cdot\frac{1}{|L_i'(v)|} \quad = \sum_{\{u,v\}\in E_i} |L_i'(u)\cap L_i'(v)|\cdot\frac{1}{|L_i'(u)|}\cdot\frac{1}{|L_i'(v)|} \\ &\le \sum_{v\in V_i}\sum_{u\in N_{G_i}^{select}(v)} |L_i'(u)\cap L_i'(v)|\cdot\frac{1}{|L_i'(u)|}\cdot\frac{1}{|L_i'(v)|} \quad \le \sum_{v\in V_i}\sum_{u\in N_{G_i}^{select}(v)}\frac{1}{|L_i'(v)|}. \end{aligned}$$

Now, $|L_i'(v)| \ge d_i(v)/2$, and $|N_{G_i}^{select}(v)| = d_i(v)$. Thus

$$W(p^{\text{unif}}) \le \sum_{v\in V_i}\sum_{u\in N_{G_i}^{select}(v)}\frac{1}{d_i(v)/2} = \sum_{v\in V_i}\frac{d_i(v)}{d_i(v)/2} = 2\,|V_i|.$$

By applying lemma 3, we have that $W(p^{\gamma_i}) \le 4\,|V_i|$. It follows from the definition of p^{γ_i} that the size of the set F_i of monochromatic edges in G_i w.r.t. γ_i is at most $4\,|V_i|$. ◀

As a consequence of the previous lemma, the core of our algorithm terminates.

▶ **Lemma 11.** *The algorithm described in Section 2 terminates.*

Proof. Lemma 10 states that, for every $i \ge 1$, the subgraph $G_i' = (V_i, F_i)$ of $G_i = (V_i, E_i)$ satisfies $|F_i| \le 4\,|V_i|$. Therefore, the subset $V_i' \subseteq V_i$ of the nodes $v \in V_i$ with $\deg_{G_i'}(v) \le 16$ satisfies $|V_i'| \ge |V_i|/2$. As a consequence, the subgraph H_i of $G_i' = (V_i, F_i)$ induced by the nodes in V_i' has at least $|V_i|/2$ nodes. It follows that there is a non-empty maximal independent set in H_i, which guarantees that $I_i \neq \varnothing$. Therefore, either $S_i \neq \varnothing$, or $J_i = I_i \smallsetminus (S_i \cup K_i) \neq \varnothing$, which ensures that at least one vertex terminates at iteration i. As a consequence, the number of iterations performed during the core of the algorithm is at most n (we shall show later that it is actually at most $O(\log n)$). ◀

Let T be the number of iterations of the algorithm until there are no more active nodes left, and let us first focus on the status of the idle nodes at the end of the T iterations. For any idle node v, let $i \ge 1$ be the iteration during which v becomes idle, i.e., $v \in K_i$, and let us then denote by

$$\deg^+(v) = |N_{G_1}(v) \cap (K_i \cup \cdots \cup K_T)|,$$

the number of neighbors of v in G_1 that belong to the set $K_i \cup \cdots \cup K_T$. Note that the idle nodes update their demands and lists at the end of the Tth iteration, so $d_{T+1}(v)$ and $L_{T+1}(v)$ are well defined for an idle node $v \in K = \cup_{i=1}^{T} K_i$. The following justifies the use of Lemma 5 in the post-processing stage (see Section 2.3).

▶ **Lemma 12.** *For every $v \in K$, $\deg^+(v) < |L_{T+1}(v)|$ and $\deg^+(v) \le k$.*

Proof. Let i be such that $v \in K_i$. By definition of idle nodes, v satisfies that

$$|N_{G_i}(v)| - d_i(v) < |S_i^v|,$$

where $S_i^v = \{u \in N_{G_i}(v) \cap S_i \ : \ \gamma_i(u) = \gamma_i(v)\}$. It follows that

$$|N_{G_i}(v)| - |S_i^v| < d_i(v) < |L_i(v)|$$

where the second inequality follows from Lemma 9. Since $S_i^v \subseteq N_{G_i}(v)$, we get

$$|N_{G_i}(v) \setminus S_i^v| = |N_{G_i}(v)| - |S_i^v|,$$

and $d_i(v) \leq |L_i(v) \setminus \{\gamma_i(v)\}|$, we get that

$$|N_{G_i}(v) \setminus S_i^v| < |L_i(v) \setminus \{\gamma_i(v)\}|.$$

Therefore, since each time the algorithm removes a color from $L_i(v)$, it also removes a node from $N_{G_i}(v)$, we get that

$$|N_{G_i}(v) \setminus (S_i \cup J_i)| < |L_{i+1}(v)|.$$

For $j = i, \ldots, T$, let us denote by

$$\deg_j^+(v) = N_{G_1}(v) \cap (K_i \cup \cdots \cup K_j \cup V_{j+1})$$

the degree of v in $K_i \cup \cdots \cup K_j \cup V(G_{j+1})$ at the end of iteration j. For $j = i$, we have just shown that $\deg_i^+(v) < |L_{i+1}(v)|$. For $j \geq i$, assuming $\deg_j^+(v) < |L_{j+1}(v)|$, we have

$$\deg_{j+1}^+(v) = \deg_j^+(v) - |N_{G_1}(v) \cap (S_j \cup J_j)|$$

because the nodes in $S_j \cup J_j$ are removed from G_j to get G_{j+1}. Similarly, by the list updates performed by the algorithm, we have

$$L_{j+1}(v) = L_j(v) \setminus \{\delta(u) : u \in N_{G_1}(v) \cap (S_j \cup J_j)\}.$$

Since a node is removed each time a color is removed, we get that $\deg_{j+1}^+(v) < |L_{j+2}(v)|$. Therefore $\deg_j^+(v) < |L_{j+1}(v)|$ holds for every $j \geq i$. The lemma follows from taking $j = T$, because $\deg_T^+(v) = \deg^+(v)$ is the degree of v in $K_i \cup \cdots \cup K_T$ as $V_{T+1} = \varnothing$.

Finally, since, at every iteration i, every active node v truncates its list $L_i(v)$ to be of size exactly $d_i(v) + 1$ at the beginning of the iteration, all lists involved in the algorithm are of size at most $k + 1$. It follows that $\deg^+(v) \leq k$. ◀

The next lemma states that the idle nodes eventually become properly colored.

▶ **Lemma 13.** *For every idle node $v \in K$, and for every $u \in N_{G_1}(v)$, we have $\delta(u) \neq \delta(v)$.*

Proof. Let $v \in K$, and let $u \in N_{G_1}(v)$. If $u \in V \setminus K$, then u terminated at some iteration $i, 1 \leq i \leq T$. By the list update applied at node v, $\delta(u) \notin L_{i+1}(v)$, and thus $\delta(u) \notin L_{T+1}(v)$. By lemma 12, the graph $G_1[K]$ is properly colorable, and is properly colored by the algorithm in Lemma 5. ◀

The next technical lemma concerns the nodes which did not become idle.

▶ **Lemma 14.** *For every $i \geq 1$, and every $v \in S_i \cup J_i$, let*

$$X_i = \{u \in N_{G_i}(v) \cap (S_i \cup J_i) : \delta(u) \neq \delta(v)\}, \text{ and } Y_i = N_{G_i}(v) \setminus (S_i \cup J_i).$$

Then $|X_i| + |Y_i| \geq d_i(v)$.

Proof. Let $i \geq 1$, and $v \in S_i \cup J_i$. From Observation 7, the neighbors of node v can be partitioned into S_i, J_i, and $Y_i = K_i \cup V_{i+1}$. The set $X_i \subseteq S_i \cup J_i$ is the subset of nodes u which adopts their final color as $\delta(u) = \gamma_i(u) \neq \gamma_i(v) = \delta(v)$. The nodes in $(S_i \cup J_i) \setminus X_i$ adopts the same final color as v, i.e., for every $u \in (S_i \cup J_i) \setminus X_i$, we have $\gamma_i(u) = \gamma_i(v)$.

If $v \in S_i$, then the coloring γ_i is such that the neighborhood of v contains all the colors in $L_i(v)$, and thus, thanks to Lemma 8, the neighborhood of v contains at least $d_i(v) + 1$ colors. These colors are those assigned by γ_i to nodes either in Y_i or X_i. Therefore, the neighborhood of v contains at least $d_i(v)$ colors different from $\gamma_i(v)$. Hence the lemma holds for $v \in S_i$.

If $v \in J_i$, then $v \notin K_i$, i.e., v does not become idle at iteration i. As a consequence,

$$|N_{G_i}(v)| - |\{u \in N_{G_i}(v) \cap S_i : \gamma_i = \gamma_i(v)\}| \geq d_i(v).$$

Since $v \in J_i$, we have $N_{G_i}(v) \cap J_i = \varnothing$. Therefore

$$|N_{G_i}(v)| - |\{u \in N_{G_i}(v) \cap (S_i \cup J_i) : \gamma_i = \gamma_i(v)\}| \geq d_i(v).$$

The lemma follows from the fact that the left hand side of the latter inequality is precisely equal to $|X_i| + |Y_i|$. ◀

For computing the total contribution of the neighbors of a node v to the original demand $d(v)$ of that node, we introduce the following notion, which applies to nodes that have not yet terminated, i.e., they are still active, or became idle at previous iterations. For every $i \geq 1$, and $v \in V_i \cup (\cup_{j=1}^{i-1} K_j)$, we denote by

$$\mathsf{good}_i(v) = \{u \in N_{G_1}(v) : \exists j \in \{1, \ldots, i-1\},\ u \in S_j \cup J_j\}$$

the set of neighbors of v in G_1 that have terminated at a round less than i. Note also that, by the list update rules, for every $u \in \mathsf{good}_i(v)$, $\delta(u) \notin L_i(v)$.

▶ **Lemma 15.** *For every $i \geq 1$, and every vertex $v \in V_i \cup (\cup_{j=1}^{i-1} K_j)$, we have*

$$|\mathsf{good}_i(v)| + d_i(v) + |N_{G_1}(v) \cap (\cup_{j=1}^{i-1} K_j)| \geq d(v).$$

Proof. The proof is by induction on $i \geq 1$. For $i = 1$, $\mathsf{good}_1(v) = \varnothing$, $d_1(v) = d(v)$, and $\cup_{j=1}^{0} K_j = \varnothing$, so the lemma holds. Let us now assume that this lemma is true for $i \geq 1$. By the definition of good vertices, $\mathsf{good}_{i+1}(v) = \mathsf{good}_i(v) \cup (N_{G_i}(v) \cap (S_i \cup J_i))$. Since, $\mathsf{good}_i(v) \cap (N_{G_i}(v) \cap (S_i \cup J_i)) = \varnothing$, it follows that

$$|\mathsf{good}_{i+1}(v)| = |\mathsf{good}_i(v)| + |(N_{G_i}(v) \cap (S_i \cup J_i))|. \tag{2}$$

Now, by the updates of the demands performed in the algorithm,

$$d_{i+1}(v) = d_i(v) - |N_{G_i}(v) \cap (S_i \cup J_i \cup K_i)|.$$

Since S_i, J_i, and K_i are disjoint, we can rewrite the previous equation as

$$d_{i+1}(v) = d_i(v) - (|N_{G_i}(v) \cap S_i| + |N_{G_i}(v) \cap J_i| + |N_{G_i}(v) \cap K_i|). \tag{3}$$

Finally, we have that

$$|N_{G_1}(v) \cap (\cup_{j=1}^{i} K_j)| = \sum_{j=1}^{i} |N_{G_1}(v) \cap K_j|. \tag{4}$$

Adding Equations (2), (3), and (4), and using the fact that S_i, J_i, and K_i are disjoint, we get

$$|\mathsf{good}_{i+1}(v)| + d_{i+1}(v) + |N_{G_1}(v) \cap (\cup_{j=1}^{i} K_j)| = |\mathsf{good}_i(v)| + d_i(v) + \sum_{j=1}^{i} |N_{G_1}(v) \cap K_j| - |N_{G_i} \cap K_i|.$$

Since $N_{G_1}(v) \cap K_i = N_{G_i} \cap K_i$ for every $i \geq 1$, we obtain that

$$\begin{aligned} |\mathsf{good}_{i+1}(v)| + d_{i+1}(v) + |N_{G_1}(v) \cap (\cup_{j=1}^{i} K_j)| &= |\mathsf{good}_i(v)| + d_i(v) + \sum_{j=1}^{i-1} |N_{G_1}(v) \cap K_j| \\ &= |\mathsf{good}_i(v)| + d_i(v) + |N_{G_1}(v) \cap (\cup_{j=1}^{i-1} K_j)|. \end{aligned}$$

The claim then follows from the induction hypothesis. ◀

We can now conclude with the correctness of our algorithm.

▶ **Proposition 16.** *The algorithm described in Section 2 terminates, and the coloring δ returned by the algorithm is a solution to k-partial list-coloring in G.*

Proof. The fact that the algorithm execute a finite number of iterations has been established in Lemma 11. For every vertex $v \in V$, either v terminates at some iteration $i \in \{1, \ldots, T\}$, or v becomes idle at some iteration $i \in \{1, \ldots, T\}$.

Let us consider a node v that terminates at some iteration $i \in \{1, \ldots, T\}$. By Lemma 14, and the fact that if $u \in N_{G_i}(v) \setminus (S_i \cup J_i)$, then $\delta(v) \notin L_i(u)$, we deduce that the number of neighbors of v in G_i which will take different color that v is at least $d_i(v)$. As v is active in the beginning of round i, it follows from Lemma 15 that

$$|\mathsf{good}_i(v)| + d_i(v) + |N_{G_1}(v) \cap K| \geq d(v).$$

Thanks to the definition of good vertices, Lemma 13, and the fact that $d(v) \leq |N_{G_1}(v)|$, it results that the lemma holds for terminating nodes.

Let us now consider a node that becomes idle at some iteration $i \in \{1, \ldots, T\}$. In that case, the results immediately follows from Lemma 13. ◀

3.2 Complexity Analysis

We now prove that our algorithm terminates in the prescribed number of rounds.

▶ **Proposition 17.** *The algorithm described in Section 2 terminates in $O(\log n \cdot \log^3 k)$ rounds.*

Proof. The algorithm starts with a preprocessing stage which takes $O(\log^\star n)$ rounds [11]. The removal of the monochromatic edges from the graph takes $O(1)$ rounds. For each iteration i of the algorithm, the prepocessing phase takes $O(1)$ rounds for each vertex to select incident edges and remove unselected edges. The derandomization phase takes $O(\log^2 k)$ rounds by Lemma 3. Finally, for the color assignment phase, computing the graph G'_i takes $O(1)$ rounds, and computing the MIS I_i on H_i takes $O(\log^\star k)$ rounds [12]. Therefore, each iteration takes at most $O(\log^2 k)$ rounds. The assignment of colors to the terminated vertices, and the update of the graph from G_i to G_{i+1} takes $O(1)$ rounds.

Thanks to Lemma 10, the graph H_i induced by the nodes with degree less than 16 in $G'_i = (V_i, F_i)$ has at least $\frac{|V_i|}{2}$ vertices. For every maximal independent set I_i in H_i, every $v \in I_i$ dominates at most 16 nodes (itself, plus its at most 15 neighbors). Therefore, at least

$\frac{|V_i|}{32}$ vertices are in I_i during the ith iteration. Each vertex in I_i either terminate (in S_i or J_i) or become idle (in K_i). Therefore, at most $\frac{31|V_i|}{32}$ vertices participate in the next iteration. Therefore, the number of iterations for the core of the algorithm is at most $O(\log n)$. It follows that, in total, the core of the algorithm takes at most $O(\log^2 k \cdot \log n)$ rounds.

For the post processing phase, by lemma 12, $G_1[K]$ is a layered graph for which $\hat{\Delta} = \max_{v \in K} \deg^+(v) \leq k$. By lemma 5, $G_1[K]$ it takes at most $O(\log^3 k \cdot \log n)$ rounds to properly list-color the vertices of this graph. ◀

Theorem 1 directly follows from Propositions 16 and 17.

4 Conclusion

We have shown that the breakthrough result of Ghaffari and Kuhn [10], stating that $(\Delta+1)$-list-coloring can be computed in $O(\log n \cdot \log^2 \Delta)$ rounds, can be generalized to k-partial list coloring, where k is the maximum demand, for all $k \geq 0$. Our algorithm for k-partial list-coloring runs in $O(\log n \cdot \log^3 k)$ rounds. The extra $\log k$ factor is due to the post-processing stage, for assigning colors to the idle nodes, which itself comes from the extra $\log k$ factor in proper list-coloring of layered graphs in [10]. However, in the specific case of k-partial $(k+1)$-coloring, this extra $\log k$ factor can be avoided, and our algorithm runs in $O(\log n \cdot \log^2 k)$ rounds. It would be interesting to know whether this extra $\log k$ factor could be avoided for general partial list-coloring, and of course whether $O(\log n \cdot \log^2 k)$ rounds is the best that can be achieved for k-partial $(k+1)$-coloring for general k.

Another interesting generalization of proper coloring is to consider the extension of proper Δ-coloring to k-partial k-coloring, for $k = 1, \ldots, \Delta$. Brook's theorem states that, in a connected graph with maximum degree Δ, the vertices can be properly colored with only Δ colors in all graphs, except for complete graphs, and cycle graphs of odd length, which require $\Delta+1$ colors. The algorithm in [10] can be used to properly Δ-color every Δ-colorable graph in $O(\log^2 n \log^2 \Delta)$ rounds. We do not know whether this can be generalized to k-partial k-coloring every graph for which there exist a k-partial coloring using a palette with only k colors.

References

1 Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 364–369, 1989. doi:10.1109/SFCS.1989.63504.

2 Alkida Balliu, Juho Hirvonen, Christoph Lenzen, Dennis Olivetti, and Jukka Suomela. Locality of not-so-weak coloring. In *26th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 11639 of *LNCS*, pages 37–51. Springer, 2019. doi:10.1007/978-3-030-24922-9_3.

3 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

4 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta+1)$: -coloring below Szegedy-Vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 437–446, 2018. URL: https://dl.acm.org/citation.cfm?id=3212769.

5 Sebastian Brandt. An automatic speedup theorem for distributed problems. In *38th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 379–388, 2019. doi:10.1145/3293611.3331611.

6 Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhon. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *34th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4409–4447, 2023. doi:10.1137/1.9781611977554.CH168.

7 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In Irit Dinur, editor, *57th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 625–634, 2016. doi:10.1109/FOCS.2016.73.

8 Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed MIS and approximate matching. In *55th ACM Symposium on Theory of Computing (STOC)*, pages 1777–1790, 2023. doi:10.1145/3564246.3585243.

9 Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhon. Improved distributed network decomposition, hitting sets, and spanners, via derandomization. In *34th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2532–2566, 2023. doi:10.1137/1.9781611977554.CH97.

10 Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *62nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1009–1020, 2021. doi:10.1109/FOCS52979.2021.00101.

11 Fabian Kuhn. Weak graph colorings: distributed algorithms and applications. In *21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 138–144, 2009. doi:10.1145/1583991.1584032.

12 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 331–335, 1987. doi:10.1109/SFCS.1987.20.

13 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

14 Yannic Maus and Tigran Tonoyan. Local conflict coloring revisited: Linial for lists. In *34th International Symposium on Distributed Computing (DISC)*, volume 179 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.16.

15 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

16 Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *J. Algorithms*, 20(2):356–374, 1996. doi:10.1006/JAGM.1996.0017.

17 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

18 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *52nd ACM Symposium on Theory of Computing (STOC)*, pages 350–363, 2020. doi:10.1145/3357713.3384298.

A Partial List-Coloring with a Common Color

In this section, we consider k-partial list-coloring with the additional assumption that all lists share a common "special" color c^* that is known to all nodes. This setting includes the important case of k-partial $(k+1)$-coloring, with $c^* = 1$. In this context, it is easier to assume that each node v has a specific list of colors of size at least $d(v)$ not containing c^*, and the color outputted by each node must be either c^* or a color from its list $L(v)$.

A.1 The Algorithm

Our algorithm remains based on the structure of the algorithm in [10] for $(\Delta+1)$-list-coloring, as our algorithm for general k-partial list-coloring, but the color assignment phase is much easier, thanks to the presence of the special color c^*, which can be used as a backup for saturated nodes. Specifically, our algorithm works as follows.

- **Preprocessing Stage.** This stage is identical to what was done in the algorithm for general k-partial list-coloring. Specifically, we compute a k-partial $O(k^2)$-coloring of G, which we denote by α. Again, this can be done in $O(\log^* n)$ rounds [11]. Remove the monochromatic edges to obtain a sub-graph of G, denoted by $G_1 = (V_1, E_1)$. Note that in G_1 the degree of any node v is at least $\min\{k, \deg(v)\}$, and therefore at least $d(v)$. We further arbitrarily truncate each list $L(v)$ to be of size $d(v)$.
- **Computing the coloring δ.** Initially, $\delta(v) = \perp$ for every $v \in V$. For computing their colors, the nodes performs $O(\log n)$ iterations. At each iteration, some nodes $v \in V$ fix their colors $\delta(v) \in L(v) \cup \{c^*\}$. Once a node becomes colored (i.e., different from $\perp$), its color never changes. We proceed with another iteration as long as there is at least one node v with $\delta(v) = \perp$. Note that, as opposed to the previous algorithm for general k-partial list-coloring, there is no post-processing stage.

  For every $v \in V$, we set $d_1(v)$ to be the input demand $d(v)$, and we set $L_1(v)$ to be the input list $L(v)$. The input to iteration $i \geq 1$ is a graph $G_i = (V_i, E_i)$, a demand function $d_i : V_i \to \mathbb{N}$, and a color list function $L_i : V_i \to 2^{\mathbb{N}}$.

  We now describe one iteration of the algorithm.

  1. **Preprocessing Phase.** Every vertex v arbitrarily selects $d_i(v)$ many edges incident on it, and orient them as outgoing edges, as we did before in the general k-partial list-coloring algorithm. Note that an edge can be oriented both ways, by its two endpoints. Any unoriented edge is removed from G_i. In what follow, we slightly abuse notations, and $G_i = (V_i, E_i)$ now refers to the graph after the removal of these edges. We define

     $$N^{out}_{G_i}(v) \coloneqq \{u \in N_{G_i}(v) \ : \ v \text{ selected } e = \{u, v\} \text{ as outgoing edge}\}.$$

  2. **Derandomization Phase.** This phase is identical to the derandomization phase of the general k-partial list-coloring, excepted that we are using lists of $d(v)$ colors at nodes v, instead of lists of $d(v) + 1$ colors, as nodes can use the wildcard color c^* in addition to the colors in their list. Corollary 3.6 in [10] provides a way to "derandomize" the uniform distribution while almost preserving the weights assigned by w_p (cf. Lemma 3).
  3. **Color Assignment Phase.** Given the graph $G_i = (V_i, E_i)$, and the coloring γ from Lemma 3, we consider the graph $G'_i = (V_i, F_i)$ where

     $$F_i \coloneqq \big\{\{u, v\} \in E_i \ : \ \gamma(u) = \gamma(v)\big\}.$$

     As we have seen in Lemma 10, we have $|F_i| \leq 4 \cdot |V_i|$. As a consequence, the number of nodes $v \in V_i$ with degree $\deg_{G'_i}(v) \geq 16$ is at most $|V_i|/2$. We denote by H_i the subgraph of G'_i induced by the nodes with $\deg_{G'_i}(v) < 16$. By construction, H_i has maximum degree at most $15 = O(1)$. In addition, H_i has a proper $(O(k^2))$ coloring (by the preprocessing stage operations). It follows that the nodes of H_i can compute a maximal independent set (MIS) I_i in H_i in $O(\log^* k)$ rounds [12] . Note that, since each of the at least $|V_i|/2$ nodes in H_i has maximum degree 15, we have $|I_i| \geq \frac{|V_i|}{32}$.

     We now modify the notion of *saturated* nodes, for adapting it to the setting in which all lists share a common color c^*.

     ▶ **Definition 18.** *We say that a node $v \in V_i$ is saturated by I_i with respect to the coloring γ if the following condition holds: for every $c \in L_i(v)$, there exists a neighbor $u \in N_{G_i}(v) \cap I_i$ with $\gamma(u) = c$.*

     Let us denote by S_i the set of nodes saturated by I_i w.r.t. γ. It is easy to see that $S_i \cap I_i = \varnothing$ (cf. Lemma 22).

All nodes in $I_i \cup S_i$ fix their final δ-colors in a way different from the color assignment in the case of general k-partial list-coloring, using the existence of the common "wildcard" color c^*, as follows:

- **MIS Node Rule:** if $v \in I_i$, then $\delta(v) \leftarrow \gamma(v)$, i.e., v adopts its γ-color as its final color, and
- **Saturated Node Rule:** if $v \in S_i$, then $\delta(v) \leftarrow c^*$, i.e., v adopts the wildcard color c^* as its final color.

Towards the next iteration we perform the following updates.

- The list of colors available to each node $v \in V_i \setminus (I_i \cup S_i)$ is updated as follows.

$$L_{i+1}(v) \leftarrow L_i(v) \setminus \{\delta(u) : \ u \in N_{G_i}(v) \cap I_i\} \ .$$

- The graph itself is updated to be $G_{i+1} = (V_{i+1}, E_{i+1})$ as
  - **a.** $V_{i+1} = V_i \setminus (S_i \cup I_i)$
  - **b.** $E_{i+1} = \big\{\{u, v\} \in E_i : u \notin S_i \cup I_i \text{ and } v \notin S_i \cup I_i\big\}$

- The (remaining) demands for each node $v \in V_i \setminus (I_i \cup S_i)$ are defined as follows.
  - **a.** If $\deg_{G_{i+1}}(v) < |L_{i+1}(v)|$ then

$$d_{i+1}(v) \leftarrow \max\{0, d_i(v) - |N_{G_i}(v) \cap I_i| - |N_{G_i}(v) \cap S_i|\};$$

  - **b.** Otherwise

$$d_{i+1}(v) \leftarrow \max\{0, d_i(v) - |N_{G_i}(v) \cap I_i|\}.$$

This completes the description of our algorithm.

A.2 The Proof

We first prove that our algorithm does solve k-partial list coloring under the assumption of the existence of the special color c^*. Next, we will prove that it runs in the prescribed number of rounds. Thanks to Lemma 6, we merely prove the correctness of the algorithm on the graph G after the preprocessing stage. We start by a claim on the relation between the size of the list of each node and its demand over the iterations.

▶ **Lemma 19.** *For any $j \geq i$, if $d_j(v) > 0$ then $|L_j(v)| - d_j(v) \geq |L_i(v)| - d_i(v)$.*

Proof. We prove the claim by induction on $j \geq i$. The base case being $j = i$ is obviously true. Assume the induction hypothesis for $j \geq i$. According to the algorithm

$$L_{j+1}(v) \coloneqq L_j(v) \setminus \{\delta(u) : \ u \in N_{G_j}(v) \cap I_j\}.$$

Now, according to the algorithm, if $d_{j+1}(v) > 0$ then

$$d_{j+1}(v) \leq d_j(v) - |N_{G_j}(v) \cap I_j|.$$

The induction claim follows from

$$|\{\delta(u) : \ u \in N_{G_j}(v) \cap I_j\}| \leq |N_{G_j}(v) \cap I_j|,$$

and from the induction hypothesis. ◀

▶ **Lemma 20.** *For every iteration i and every node $v \in V_i$, $|L_i(v)| \geq d_i(v)$.*

Proof. The claim follows from the definition of the input to the algorithm and from Lemma 19. ◀

▶ **Lemma 21.** *For every iteration i and every node $v \in V_i$, $\deg_{G_i}(v) \geq d_i(v)$.*

Proof. In each iteration i of the algorithm the degree of a node v is reduced by exactly

$$|N_{G_i}(v) \cap I_i| + |N_{G_i}(v) \cap S_i|,$$

and its demand is reduced by at most that number. ◀

The following establishes that our algorithm is well defined.

▶ **Lemma 22.** *For every iteration i, we have $S_i \cap I_i = \varnothing$.*

Proof. if $v \in S_i$, then v has a neighbor $u \in N^{\gamma}_{G_i}(v) \cap I_i$ with $\gamma(u) = \gamma(v)$. As a consequence, if $v \in I_i$ too, then both v and u were vertices of H_γ that were adjacent in this graph, a contradiction with the fact that I_i is an independent set of H_γ. ◀

We now give two lemmas that will allow us to establish the number of bichromatic edges in G_i, according to the final coloring δ.

▶ **Lemma 23.** *For every iteration i, every node $u \in I_i$, and every node $v \in N_{G_i}(u)$, we have*

$$\begin{cases} \delta(u) \neq \delta(v) & \text{if } v \in I_i \cup S_i \\ \delta(u) \notin L_{i+1}(v) & \text{otherwise} \end{cases}$$

Proof. First note that it can be the case that u and v are both in I_i and adjacent in G_i, because I_i is an independent set in H_i and not in G_i.

- If $v \in I_i$, then $\delta(u) = \gamma(u) \neq \gamma(v) = \delta(v)$, as otherwise the edges (u, v) would have been in H_i (because H_i is obtained from G_i by removing bichromatic edges and nodes), and therefore it cannot be the case that both u and v are in I_i.
- If $v \in S_i$ then $\delta(v) = c^* \notin L(u)$ and therefore $\delta(u) \neq \delta(v)$. ◀

▶ **Lemma 24.** *For any iteration i and any node $u \in S_i$,*

$$|\{v : v \in N_{G_i}(u), \delta(u) \neq \delta(v)\}| \geq d_i(u) \ .$$

Proof. By the definition of S_i it follows that for every $c \in L_i(u)$, there exists a neighbor $v \in N_{G_i}(v) \cap I_i$ with $\gamma(v) = c$. By the algorithm for all nodes $x \in I_i$, $\delta(x) = \gamma(x) \neq c^*$, and $\delta(u) = c^*$. By Lemma 20, $|L_i(u)| \geq d_i(u)$, and the lemma follows. ◀

We proceed with two definitions. First, we define the notion of *slackness*.

▶ **Definition 25.** *Let $i \geq 1$, and let $v \in V_i$ be a vertex. The* slackness *of v is defined as* $\texttt{slack}_i(v) := \deg_{G_i}(v) - d_i(v)$.

Second, we introduce the notion of *free* nodes.

▶ **Definition 26.** *Let $i \geq 1$, and let $v \in V_i$ be a vertex. We say that v is* free *whenever $\deg_i(v) < |L_i(v)|$. We denote by free_i the set of free nodes in G_i.*

▶ **Lemma 27.** *For $v \in V_i$, if $|N_{G_i}(v) \cap S_i| > \texttt{slack}_i(v)$ then, for any $j > i$,*

$$v \in V_j \Rightarrow v \in \mathsf{free}_j.$$

Proof. By the definition of the algorithm

$$\deg_{i+1}(v) = \deg_i(v) - |N_{G_i}(v) \cap I_i| - |N_{G_i}(v) \cap S_i|.$$

Using the conditions of the Lemma we then have that

$$\begin{aligned}\deg_{i+1}(v) &< \deg_i(v) - |N_{G_i}(v) \cap I_i| - \mathtt{slack}_i(v) \\ &= \deg_i(v) - |N_{G_i}(v) \cap I_i| - (\deg_i(v) - d_i(v)) \\ &= d_i(v) - |N_{G_i}(v) \cap I_i|.\end{aligned}$$

Using Lemma 20 we have then

$$\deg_{i+1}(v) < |L_i(v)| - |N_{G_i}(v) \cap I_i| \leq |L_{i+1}(v)|.$$

We therefore have that $\deg_{i+1}(v) < |L_{i+1}(v)|$. By Lemma 19 we have that, for all $i < j$, $\deg_j(v) < |L_j(v)|$. By the definition of a free node, for all $i < j$, if $v \in V_j$ then $v \in \mathsf{free}_j$. ◀

We need one last definition, which adapts the notion of good nodes to the case of lists with a common color. The new notion of *good nodes for* v (at iteration i) is intended to allow us to count the number of nodes adjacent to v in G that are already certain to be colored differently than v by the final coloring.

▶ **Definition 28.** *For iteration i and node $v \in V_i$ we define* $\mathsf{good}_i(v)$ *as follows.*

$$\mathsf{good}_i(v) = \begin{cases} \{u \in V \setminus V_i \mid \delta(u) \notin L_i(v)\} & \text{if } v \in \mathsf{free}_i \\ \{u \in V \setminus V_i \mid \delta(u) \notin L_i(v) \cup \{c^*\}\} & \text{otherwise.} \end{cases}$$

The following observation directly follows from the definition of $\mathsf{good}_i(v)$. It formally states which of the neighbors of v in G_{i-1} are added, at the end of iteration $i-1$, to the set of "good neighbors" of v.

▶ **Observation 29.** *For $i > 1$, if $v \in \mathsf{free}_i$, then*

$$N_{G_{i-1}}(v) \cap (S_{i-1} \cup I_{i-1}) \subseteq \mathsf{good}_i(v),$$

else

$$N_{G_{i-1}}(v) \cap I_{i-1} \subseteq \mathsf{good}_i(v).$$

▶ **Lemma 30.** *For every iteration i and every node $v \in V$*

$$|\mathsf{good}_i(v)| + d_i(v) \geq d(v)\ .$$

Proof. We prove the claim by induction on i. The base case being $i = 1$ holds since $d_1(v) = d(v)$ by the definition of the algorithm. Assume the induction hypothesis for $i \geq 1$. First observe that for all $v \in V$, $\mathsf{good}_i(v) \subseteq \mathsf{good}_{i+1}(v)$ (and hence $|\mathsf{good}_i(v)| \leq |\mathsf{good}_{i+1}(v)|$), because

$$V \setminus V_i \subseteq V \setminus V_{i+1},\ \ L_{i+1} \subseteq L_i, \text{ and } \mathsf{free}_i \subseteq \mathsf{free}_{i+1}.$$

However, it could be that $d_{i+1}(v) < d_i(v)$. According to the algorithm there are two cases for the update of $d_{i+1}(v)$.

– If $\deg_{G_{i+1}}(v) < L_{i+1}(v)$ then $d_{i+1}(v)$ is reduced, compared to $d_i(v)$, by at most

$$|N_{G_i}(v) \cap I_i| + |N_{G_i}(v) \cap S_i|.$$

Moreover, if $\deg_{G_{i+1}}(v) < L_{i+1}(v)$ then $v \in \mathsf{free}_{i+1}$, and by Observation 29,

$$|\mathsf{good}_{i+1}(v) \setminus \mathsf{good}_i(v)| \geq |N_{G_i}(v) \cap I_i| + |N_{G_i}(v) \cap S_i|.$$

– If $\deg_{G_{i+1}}(v) \geq L_{i+1}(v)$ then $d_{i+1}(v)$ is reduced, compared to $d_i(v)$, by at most $|N_{G_i}(v) \cap I_i|$. Using again Observation 29,

$$|\mathsf{good}_{i+1}(v) \setminus \mathsf{good}_i(v)| \geq |N_{G_i}(v) \cap I_i|,$$

which completes the proof. ◀

We now have all the necessary ingredients to establish the correctness of our algorithm.

▶ **Proposition 31.** *If the algorithm in Section A.1 terminates, then the final coloring δ is a solution to k-partial list-coloring on G, whenever all lists include a common color c^*.*

Proof. Let vertex v terminate at the end of iteration i of the algorithm (i.e., $v \in V_i$ but $v \notin V_{i+1}$).

- If $v \in I_i$, then, by Lemma 23, for every $u \in N_{G_i}(v)$, either $\delta(u) \neq \delta(v)$ or $\delta(u) \notin L_{i+1}(v)$. Now, observe that $\mathsf{good}_i(v) \cap V_i = \varnothing$. Hence, the number of nodes $u \notin \mathsf{good}_i(v), u \in_{G_i} (v)$ colored differently than v is at least $|N_{G_i}(v)| \geq d_i(v)$, using Lemma 21.
- If $v \in S_i$, then, by Lemma 24,

  $$|\{u \in N_{G_i}(v) \ : \ \delta(u) \neq \delta(v)\}| \geq d_i(v).$$

  Similarly to the previous case, because $\mathsf{good}_i(v) \cap V_i = \varnothing$, the number of nodes $u \notin \mathsf{good}_i(v), u \in_{G_i} (v)$ is at least $d_i(v)$.

By Lemma 30 we get that the total number of nodes $u \in N_G(v)$ colored differently than v is at least $d(v)$. ◀

We finally prove that our algorithm terminates in the prescribed number of rounds.

▶ **Proposition 32.** *The algorithm in Section A.1 terminates in $O(\log n \cdot \log^2 k)$ rounds.*

Proof. By the same arguments as in the proof of Proposition 17, excepted from the post-processing stage, which is absent from the algorithm for lists with a common color. ◀

Theorem 2 directly follows from Propositions 31 and 32.