Distributed Delta-Coloring Under Bandwidth Limitations

Magnús M. Halldórsson
Reykjavik University, Iceland

Yannic Maus
TU Graz, Austria

Abstract

We consider the problem of coloring graphs of maximum degree Δ with Δ colors in the distributed setting with limited bandwidth. Specifically, we give a poly $\log\log n$-round randomized algorithm in the CONGEST model. This is close to the lower bound of $\Omega(\log\log n)$ rounds from [Brandt et al., STOC '16], which holds also in the more powerful LOCAL model. The core of our algorithm is a reduction to several special instances of the constructive Lovász local lemma (LLL) and the $deg+1$-list coloring problem.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Graph problems, Graph coloring, Lovász local lemma, LOCAL model, CONGEST model, Distributed computing

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.31

Related Version *Full Version*: https://arxiv.org/abs/2405.09975

Funding *Magnús M. Halldórsson*: Supported in part by Icelandic Research Fund grant 217965.
Yannic Maus: Supported by the Austrian Science Fund (FWF), Grant P36280-N.

Acknowledgements We thank Saku Peltonen for valuable input and discussions.

1 Introduction

The objective in the c-coloring problem is to color the vertices of a graph with c such that any two adjacent vertices receive different colors. In the distributed setting, the $\Delta+1$-coloring problem has long been the focus of interest as the natural *local* coloring problem: any partial solution can be extended to a valid full solution. It has fast poly$(\log\log n)$-round algorithms, both in LOCAL [12] and CONGEST [29], and so does the more general *deg+1*-list coloring problem (d1LC), which is what remains when a subset of the nodes has been $\Delta+1$-colored [30, 34].

The Δ-coloring problem, on the other hand, is *non-local*: fixing the colors of just two nodes can make it impossible to form a proper Δ-coloring, see Figure 1 for an example. Due to its simplicity, it has become the prototypical problem for the frontier of the unknown [27, 2]. Even the existence of such colorings is non-trivial: a celebrated result by Brooks from the '40s shows that Δ-colorings exist for any connected graph that is neither an odd cycle nor a clique on $\Delta+1$ nodes [10].

A poly$(\log\log n)$-round Δ-coloring algorithm was recently given in LOCAL [22], but no non-trivial algorithm is known in CONGEST. It is of natural interest to examine if the transition from local to non-local problems behaves differently in LOCAL and in CONGEST. Thus, we set out to answer the following question:

> Is there a sublogarithmic time distributed Δ-coloring algorithm using small messages?

In this work, we answer the question in the affirmative. We prove the following theorem.

▶ **Theorem 1.** *There is a randomized* $\operatorname{poly}\log\log n$*-round* CONGEST *algorithm to* Δ*-color any graph with maximum degree* $\Delta \geq 3$*. The algorithm works with high probability.*

Theorem 1 nearly matches the lower bound of $\Omega(\log\log n)$ that holds in LOCAL [9]. In [2], the authors claim that in order to make progress in our understanding of distributed complexity theory, we require a Δ-coloring algorithm that is genuinely different from the approaches in [41, 27]. This is due to the fact that the current state-of-the-art runtime for Δ-coloring lies exactly in the regime that is poorly understood. The approaches of [41, 27] are based on brute-forcing solutions on carefully chosen subgraphs of super-constant diameter. In contrast, our results are based on a bandwidth-efficient deterministic reduction to a constant number of "simple" Lovász Local Lemma (LLL) instances and $O(\log\Delta)$ instances of d1LC; the LLL is a general solution method applicable to a wide range of problems. It is known that LLL is complete for sublogarithmic computation on constant-degree graphs, but its role on general graphs is widely open [13]. Our algorithm adds to the small list of problems (see the related work section in [33]) that can be solved in sublogarithmic time with an LLL-type approach, even under the presence of bandwidth restrictions. Before continuing further, let us first detail the computational model.

In the CONGEST model, a communication network is abstracted as an n-node graph of maximum degree Δ, where nodes serve as computing entities and edges represent communication links. Initially, a node is unaware of the topology of the graph G, nodes can communicate with their neighbors in order to coordinate their actions. This communication happens in synchronous rounds where, in each round, a node can perform arbitrary local computations and send one message of $O(\log n)$ bits over each incident edge. At the end of the algorithm, each node outputs its own portion of the solution, e.g., its color in coloring problems. The LOCAL model is identical, except without restrictions on message size.

1.1 Technical Overview on Previous Approaches

Previous fast distributed Δ-coloring algorithms either use huge bandwidth [41, 27] or use limited bandwidth but only work in the extreme cases of either very high-degree [22] or super low-degree graphs [40]. Optimally, we would like to take any of these solutions and run them with minor modifications to obtain an algorithm that uses low bandwidth and works for all degrees. This approach is entirely infeasible for the highly specialized algorithms in [41, 27, 28]. These works crucially rely on learning the full topology of non-constant diameter subgraphs, which is impossible in CONGEST.

For graphs of super-low degree, i.e., at most $\operatorname{poly}\log\log n$, an efficient Δ-coloring algorithm with low bandwidth can be deduced from the results in [40]. In fact, the paper takes a complexity-theoretic approach and shows that any problem can be solved in sublogarithmic time with low bandwidth as long as 1) the problem is defined on low-degree graphs, 2) a given solution can be checked efficiently for correctness by a distributed algorithm, and 3) the problem admits a sublogarithmic time LOCAL model algorithm. As such, the results are not very constructive for any specific problem like the Δ-coloring problem. In fact, it is known that these generic techniques cannot be extended to problems defined on graphs with larger degrees [3], which is the main target of our work.

Our best hope is then the $\operatorname{poly}\log\log n$-round LOCAL model algorithm of [22]. We discuss it in detail throughout the next few pages as it motivates the design choices of our solution. Unfortunately, for maximum degrees that are at most poly-logarithmic, it relies on the prior $O(\log\Delta) + \operatorname{poly}\log\log n$-round LOCAL model algorithm from [27] in a black-box manner.

For large maximum degrees, however, when Δ is $\omega(\log^3 n)$, they provide a sophisticated constant-round randomized reduction to the $deg+1$-list coloring problem (d1LC) that also works with low bandwidth. The central ingredient in this reduction is the notion of slack.

Slack. To reduce the Δ-coloring problem to d1LC, it suffices to obtain a unit amount of *slack* for each node. Namely, if two neighbors of a node are assigned the same color, there are then more colors available to the node than its number of uncolored neighbors. Slack can be easily generated w.h.p. (for most, but not all, kinds of nodes) with a simple single-round procedure termed SlackGeneration, as long as the graph has high degree. This observation has been used in countless papers on various coloring problems, e.g., [18, 35, 12, 29, 22]. For intermediate-degree graphs, this slack generation problem can be formulated as an instance of the constructive Lovász Local Lemma (LLL), but one that seems inherently non-implementable in CONGEST, as we explain later.

Recall that the LLL is a general solution method applicable to a wide range of problems. Defined over a set of independent random variables, it asks for an assignment of the variables that avoids a set of "bad" events. The original theorem [19] shows that such an assignment exists as long as the probability of the events to occur is sufficiently small in relation to the dependence degree of the events, i.e., the number of other events that share a variable. There is now a general LOCAL algorithm running in $O(\log n)$ rounds of LOCAL [39, 15], but superfast $\text{poly}(\log\log n)$ algorithms are only known for restricted cases [21, 26, 17]. Even less is known about solvability in CONGEST [31, 33].

In the presented slack generation LLL, there is a bad event for each node that holds if the respective node does not obtain slack. The mentioned SlackGeneration works as follows. Each node gets activated with a constant probability, picks a random candidate color that it keeps if no neighbor wants to get the same color and discards otherwise (see Algorithm 3 in Section 3 for details). Hence, there are random variables for each node depicting its activation status and candidate color choice. The main reason why this LLL cannot be directly implemented in CONGEST is that events involve values of variables at distance 2 in the communication graph. This makes it impossible for an event node to obtain full information on the status of all its variables, an ingredient that essentially is crucial in all known sublogarithmic-time LLL algorithms. The formal meaning of the word "essential" in that sentence is extremely technical and is captured by the notion of a *simulatable* LLL (see the full version of this paper). In essence, it says that the LLL is easy enough such that event nodes can learn enough information about their variables to execute some simple primitives such as evaluating their status (does the event hold or not), resampling their variables, and computing certain conditional probabilities for the event to hold under partial variable assignments. The latter condition is the most challenging one to ensure.

1.2 Our Technical Approach

What we have discussed so far is only half the truth. In fact, the slack generation process only works for *sparse* nodes, i.e., nodes with many non-edges in their neighborhood. If the graph is locally too dense, then slack cannot be obtained via this LLL. Thus, the algorithm of [22] carefully analyzes the topological structure of the hard instances for Δ-coloring, combining several different (deterministic and randomized) methods to create slack. Such a treatment seems to be inherent to the Δ-coloring problem as a very similar classification was independently and currently discovered in the streaming model [1]. Additionally, it has also been shown to be useful in different models of computation. In the aftermath of these works, it has been used to obtain efficient massively parallel algorithms for the problem [16].

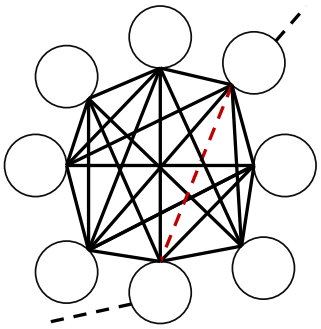

Figure 1 This is an example of an almost clique (AC). The depicted nice AC is a clique on $\Delta + 1$ nodes with a single missing (red) edge. It is essential that the two nodes incident to the missing edge receive the same color to solve the Δ-coloring problem. All non-nice ACs form proper cliques.

Our algorithm is based on a fine-grained version of this classification equipped with a sequence of various LLLs for eventual slack generation. Each LLL is easier to solve in the CONGEST model than the aforementioned slack generation LLL. In the following, we use the terminology of [22], and explain their algorithm and our solution in more detail.

Like in all recent randomized distributed graph coloring algorithms, they divide the graph into sparse and dense parts that are referred to as "almost-cliques" (ACs). Then, they partition the ACs further into different types – *ordinary, nice, difficult* – each of which admits a different coloring approach. See Figure 1 for an example of an AC. One challenge is that all these different types of tricky subgraphs may appear in the same graph and close to each other. For this overview it is best to imagine each AC as a proper clique on almost Δ nodes in which each node has a few external neighbors residing in other ACs and creating lots of dependencies between different ACs. Thus, their algorithm is fragile with regard to the order in which different types of ACs are colored. The starting point of our work is that the core step of their algorithm does not work in low-degree graphs. More detailed, the first step of their algorithm executes SlackGeneration (see Algorithm 3 in Section 3) on a carefully selected subset of nodes to achieve three objectives: **a)** giving slack to all sparse nodes, **b)** providing a slack-toehold[1] for a subclass of the difficult ACs that the authors term "runaway", and **c)** providing each ordinary clique with a node that has slack. Each of these probabilistic guarantees holds w.h.p. as long as $\Delta = \omega(\log^3 n)$. Their proof shows that, in essence, all three cases are LLLs but ones that are far from being simulatable. We discuss our solutions for a)–c), separately.

Solution for a). Providing slack to sparse graphs is the main application of the LLL algorithm in [33]. In essence, we adapt their techniques to provide slack to sparse nodes but provide additional guarantees that are needed for other parts of the graph.

Solution for b). For the difficult cliques we propose a solution that eliminates randomness and solely colors all the nodes via a sequence of d1LC instances. See Figure 2 for an illustration of our solution. First, we adjust the classification of difficult almost-cliques from [22]. All nodes in a given difficult clique have the same external degree. We associate with each such AC C a *special node* s_C on its outside that has many neighbors on the inside (namely, more than twice the external degree of C's nodes).

[1] A slack-toehold for an AC is an uncolored node that can be stalled to be colored later. All of its neighbors then lose one competitor for the remaining colors, providing them with temporal slack.

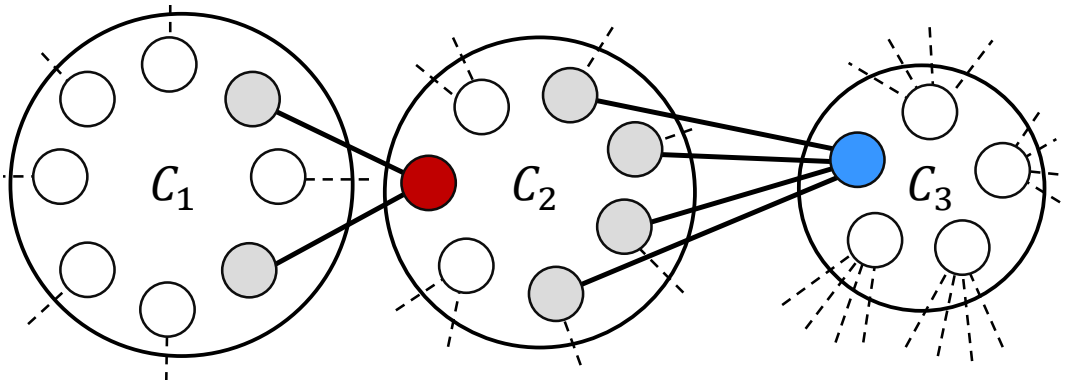

Figure 2 for part b): The illustration depicts three difficult cliques of different layers. The external degree of C_1 is 1, the external degree of C_2 is 2 and the external degree of C_3 is 4. C_1 has the lowest layer and its special node (the red node) is part of C_2. The blue special node of C_2 is part of C_3. So when we color C_1 the red node serves as an uncolored toehold providing slack to two gray nodes of C_1. Stalling the coloring of these gray nodes provides slack to the white nodes of C_1 so that they can be colored, followed by the gray ones. For illustration purposes, we chose Δ to be 9, but note that this would actually not classify C_3 as a difficult clique. A special node of C_3 would need $2e_{C_3} = 8$ neighbors in C_3, which is impossible due to C_3's size.

From here, we assign each difficult clique a *layer* that determines the step in which it gets colored. Those with a special node that is *not* contained in another difficult clique are treated separately and assigned to layer ∞, to be dealt with at the very end. The other difficult cliques are assigned to layers indexed by the base-2 logarithm of their external degree. The crucial property that follows is that the cliques in a given layer have their special node in a higher layer. This allows us to color the cliques layer by layer, starting with smaller layers. The special node s_C is stalled to be colored later, providing a toehold for C. This way, we color the cliques and special nodes in all layers besides ∞.

This leaves the problem of coloring ACs the ∞ layer and their still uncolored special nodes. In this exposition, we assume that special nodes are not shared by multiple difficult cliques. In that case, we pair the special node s_C up with some node $u_C \in C$ that is not adjacent to s_C with the objective to *same-color* the nodes: assigning both the same color. This is done via a virtual coloring problem capturing the dependencies between all selected pairs in the participating difficult cliques and the restrictions imposed by already colored vertices of the graph. We show that this virtual coloring instance is indeed a d1LC instance and can be solved efficiently in CONGEST despite being a problem on a virtual graph. As a result, the clique C obtains an uncolored node y_C that is adjacent to both s_C and u_C, has slack due to two same-colored neighbors, and can serve as a toehold for C.

Besides removing the need for randomization to solve the difficult cliques, our classification of difficult cliques also captures significantly more ACs than the definition of difficult cliques in [22]. The additional structure provided to the remaining ACs is exploited down the line in the most challenging part of the algorithm, dealing with the ordinary cliques in part c).

Solution for c). The most involved part by far is dealing with case c). We split the ordinary cliques into the small (of size less than $\Delta - \Delta/\operatorname{poly}\log\log(n)$) and large. The small ones can be handled just like the sparse nodes, as one can show that their induced neighborhoods are relatively sparse. The main effort then is to manually create slack for the large ordinary cliques. For this exposition, it is best to imagine an ordinary clique to be a clique on Δ nodes in which each node of the clique has exactly one external neighbor that is again a member of a large ordinary clique. See Figure 3 for an illustration.

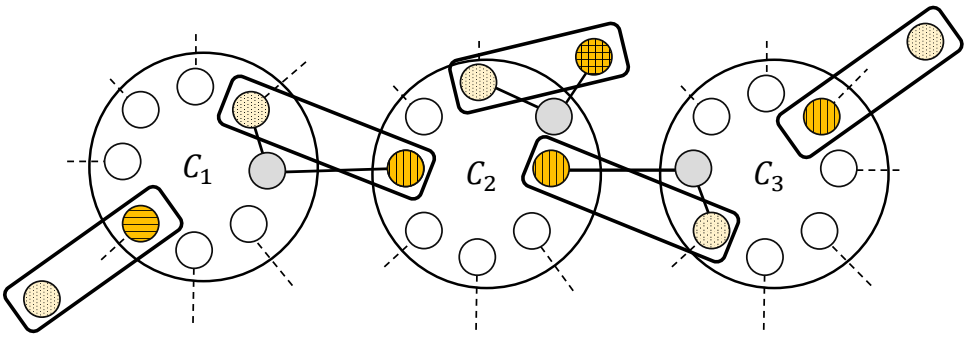

Figure 3 for Part c): For the large ordinary cliques, we find triples of nodes consisting of a yellow (striped), a light yellow (dotted), and a gray (solid) node. The two yellow nodes are non-adjacent while the gray node is adjacent to both of them. The goal is to same-color the pairs of yellow/light-yellow nodes, to which end we form a virtual coloring instance consisting of all pairs and their dependencies. After same-coloring the yellow nodes, the gray node provides a slack-toehold for the clique. An important aspect is that triples of different ordinary ACs are non-overlapping and no neighborhood of the graph contains too many nodes in such pairs, as otherwise we may run into unsolvable subinstances down the line. We find these triples by a sequence of "simple" LLLs.

In order to create slack-toehold in each large AC C, we compute a "vee-shaped" triple (x_C, y_C, z_C) of nodes, with $x_C, y_C \in C$ and $z_C \notin C$, but $z_C \in N(y_C)$ and z_C is also a non-neighbor of x_C. Then, we set up a virtual list coloring instance with a node for each such pair with the objective to same-color the pairs (x_C, z_C). As we ensure that y_C is uncolored, it serves as a slack-toehold for the AC. As many of the important ACs can be mutually adjacent, the main difficulty lies in finding non-overlapping triples for the ACs. We ensure this by first computing a suitable candidate set Z from which we then pick the third node z_C of the triple. Finding the set Z can be modeled as an "easy" LLL fitting the framework of [33]. Finding the node $z_C \in Z$ can also be modeled as a different type of "easy" LLL. In essence, the first LLL is easy (in CONGEST) as its bad events only consist of simple bounds on the number of neighbors in Z. Next, we elaborate on our LLL for finding $z_C \in Z$ with slightly more detail; due to further technicalities of the existing LLL algorithms from which we spare you in this technical overview, our actual solution differs slightly from the one presented here.

With a given set Z, we model the problem of selecting $z_C \in Z$ as an LLL as follows. Each AC C sends a proposal (to serve as its z_C node) to each outside neighbor inside Z with probability $\operatorname{poly}\log\log n/\Delta$. The proposal is *successful* if no other AC proposes to that node. We show that with a constant probability, no other AC proposes to the same node and that this is independent for different nodes in Z. Since we ensure C has many neighbors in Z, we obtain that the probability that none of C's proposals are successful is bounded above by $p = \exp(-\Omega(\operatorname{poly}\log n))$. The main benefit is that this LLL and also the LLL for finding the set Z are simple enough to be simulatable (in contrast to LLLs based on randomized slack generation for those ACs that can be derived from the proofs in [33]).

Once we have found z_C, the structure of large ordinary ACs implies that we can deterministically find the other two nodes x_C and y_C of the triple. Additional complications arise in ensuring that the list coloring instance of the pairs is a d1LC instance, i.e., that the size of the joint available color palette of x_C and z_C exceeds the maximum degree in the virtual graph induced by the pairs. The last difficulty that appears is solving the d1LC instance, as the bandwidth between the nodes within a pair is very limited and existing d1LC algorithms cannot be run in a black-box manner.

Further related work. Graph coloring is fundamental to distributed computing as an elegant way of breaking symmetry and avoiding contention, and was, in fact, the topic of the original paper introducing the LOCAL model [37]. There is an abundance of efficient deterministic and randomized $\Delta+1$-coloring algorithms in LOCAL and CONGEST for various settings, e.g., [6, 35, 23, 12, 5, 43, 39, 29, 32, 30, 24]. The excellent monograph on distributed graph coloring by Barenboim and Elkin is still a great resource for older results [7].

There are significantly fewer results for coloring with fewer than $\Delta+1$ colors. A LOCAL algorithm is known for $\Delta-k$-coloring in graphs not containing too large cliques [4]. An $O(\log\log n)$-round Δ-coloring algorithm in the LOCAL model is known for trees [11], matching the lower bound [9] within a constant factor. Additionally, there are works coloring special graph classes such as coloring planar graphs with 6 or 5 colors in $O(\log n)$ rounds with a deterministic LOCAL algorithm [14, 42].

Outline. In Section 2, we define the notion of slack and state required results from prior work on solving d1LC and computing an almost clique decomposition (ACD). In Section 3, we present our Δ-coloring algorithm with essentially all proofs. The algorithm consists of 5 phases and all phases except for Phases 1 (ACD computation) and Phase 2 are deterministic reductions to various d1LC instances. In Phase 2, we provide slack to sparse nodes and the nodes in ordinary cliques; this refers to part a) and part c) described in Section 1.2. For ease of presentation, the (involved) Phase 2 is presented in a top down manner. In Section 4 we present the high level overview of this phase. In essence its a reduction to solving four different subproblems. The details of the reduction are deferred to Appendix A. The heart of our approach is actually solving each of these subproblems via an instance of the constructive Lovász Local Lemma. As this part is extremely technical and cannot fit into the space constraints of a conference publication we defer this part to the full version of the paper.

2 Preliminaries: d1LC, Slack, Almost-Clique Decomposition, Graytone

In the $deg+1$-list coloring (d1LC) problem, each node of a graph receives as input a list of allowaed colors whose size exceeds its degree. The goal is to compute a proper vertex coloring in which each node outputs a color from its list. The problem can be solved with a simple centralized greedy algorithm, and it also admits efficient distributed algorithms.

▶ **Lemma 2** (List coloring [30, 34]). *There is a randomized* CONGEST *algorithm to $(deg+1)$-list-color (d1LC) any graph in $O(\log^5\log n)$ rounds, w.h.p. This reduces to $O(\log^3\log n)$ rounds when the degrees and the size of the color space is* $\mathrm{poly}(\log n)$.

The slack of a node (potentially in a subgraph) is defined as the difference between the size of its palette and the number of uncolored neighbors (in the subgraph).

▶ **Definition 3** (Slack). *Let v be a node with color palette $\Psi(v)$ in a subgraph H of G. The slack of v in H is the difference $|\Psi(v)|-d$, where d is the number of uncolored neighbors of v in H.*

We use the following helpful terminology.

▶ **Definition 4** (Graytone [22]). *Consider an arbitrary step of the algorithm. A node is* gray *if it has unit-slack or a neighbor that will be colored in a later step of the algorithm. A node is* grayish *if it is not gray but has a gray neighbor. A set of gray and grayish nodes is said to be* graytone.

Any graytone set can be colored as two d1LC instances: first the grayish nodes and then the gray. We emphasize that the graytone property depends on the order in which nodes are processed. It always refers to a certain step of the algorithm in which we color the respective set. Throughout our algorithm we aim at making more and more nodes graytone.

A proof of the following construction central to our approach is given in Appendix B.

▶ **Lemma 5** (ACD computation [1, 22]). *For any graph $G = (V, E)$, there is a partition (*almost-clique decomposition (ACD)*) of V into sets V_{sparse} and $C_1, C_2, \ldots, C_t$ such that each node in V_{sparse} is $\Omega(\epsilon^2\Delta)$-sparse and for every $i \in [t]$,*

(i) $(1-\varepsilon/4)\Delta \leq |C_i| \leq (1+\varepsilon)\Delta$,

(ii) *Each $v \in C_i$ has at least $(1-\varepsilon)\Delta$ neighbors in C_i: $|N(v) \cap C_i| \geq (1-\varepsilon)\Delta$,*

(iii) *Each node $u \notin C_i$ has at most $(1-\varepsilon/2)\Delta$ neighbors in C_i: $|N(u) \cap C_i| \leq (1-\varepsilon/2)\Delta$.*

Further, there is an $O(1)$-round CONGEST *algorithm to compute a valid ACD, w.h.p.*

We say that nodes in V_{sparse} are *sparse* and other nodes are *dense*. It is immediate from Lemma 5 that each dense node has external degree (or neighbors outside its AC) at most $\varepsilon\Delta$ and at most $2\varepsilon\Delta$ non-neighbors in its AC. Also, any pair of nodes in C_i have at least $(1-3\varepsilon)\Delta \geq 3\Delta/4$ common neighbors in C_i.

Notation. For a graph $G = (V, E)$ and two nodes $u, v \in V$, let $\text{dist}_G(u, v)$ denote the length of a shortest (unweighted) path between u and v in G. For a set $S \subseteq V$ we denote $\text{dist}_G(v, S) = \min_{u \in S} \text{dist}_G(v, u)$. $N(v)$ denotes the set of neighbors of a node $v \in V$.

3 Δ-Coloring in CONGEST

In this subsection, we prove the following theorem.

▶ **Theorem 1.** *There is a randomized* $\text{poly}\log\log n$*-round* CONGEST *algorithm to Δ-color any graph with maximum degree $\Delta \geq 3$. The algorithm works with high probability.*

The extreme cases of very large Δ and very small Δ can be solved in the claimed runtime with prior work [22, 40], see the proof of Theorem 1 in Section 3.3. Here, we present an algorithm for the most challenging regime where $\Delta \in O(\text{poly}\log n) \cap \Omega(\text{poly}\log\log n)$.

In the extreme case that $\Delta = \omega(\log^{21} n)$, the Δ-coloring algorithm from [22] even runs in $O(\log^* n)$ rounds. A lower bound of $\Omega(\log_\Delta \log n)$ rounds in the LOCAL model for the Δ-coloring problem [9] rules out a $O(\log^* n)$ algorithm for small Δ. Hence, in this section, we aim for an algorithm using $\text{poly}\log\log n$ rounds. In fact, we reduce the Δ-coloring problem to a few list coloring instances and a few LLL instances, each of which we solve in $\text{poly}\log\log n$ rounds.

3.1 Fine-Grained ACD Partition

The following definitions of types of almost-cliques are crucial for all results of the paper. The reader is hereby warned to read them slowly!

▶ **Definition 6** (Types of almost-cliques). *For an AC C, let $e_C = \Delta - |C| + 1$. An AC is* easy *if it contains a non-edge or a node of degree less than Δ. A node $v \notin C$ is an* intrusive *neighbor of a non-easy C if v has at least $2e_C$ neighbors in C. A non-easy AC is* difficult *if it has an intrusive neighbor. Each difficult AC C arbitrarily selects one of its intrusive neighbors as its* special *node s_C. An AC is* nice *if it is easy or if it is both non-difficult and contains a special node (necessarily for another AC). An AC is* ordinary *if it is neither nice nor difficult.*

Note that all ACs except the easy are proper cliques and all nodes in such a clique C have external degree e_C. We say that a node is ordinary (difficult, nice) if it belongs to an ordinary (difficult, nice) AC, respectively. The difficult ACs are divided into *levels*.

▶ **Definition 7** (Levels of difficult ACs). *The* maximum level ∞ *contains all difficult ACs whose special node is not contained in a difficult AC. A difficult AC C that is not at the maximum level has* level $\ell(C) = \lceil \log_2 e_C \rceil$.

Observe that $\ell(C) \leq \log_2 \Delta = O(\log \log n)$ for all difficult ACs.

▶ **Definition 8** (Node classification). *The nodes are partitioned into the following sets:*

1. $\mathcal{S}$*: the set of special nodes that are not in difficult ACs,*
2. $\mathcal{D}_\ell$*: nodes in difficult ACs of level ℓ, $\ell \in [\lg \Delta] \cup \{\infty\}$ (might include special nodes),*
3. $\mathcal{N}$*: nodes in nice ACs, excluding those in $\mathcal{S}$,*
4. $\mathcal{O}$*: nodes in ordinary ACs, and*
5. V_**: nodes in V_{sparse}, excluding those in $\mathcal{S}$.*

Our classification is built on [22] but is subtly different and more fine-grained. We are driven by a need to limit the reach of probabilistic arguments, being that we are in the challenging sub-logarithmic degree range. Thus, a strictly smaller set of dense nodes (the ordinary) needs probabilistic slack in our formulation. On the other hand, the easy, difficult, and nice definitions are more inclusive here. The difficult ones are here divided into super-constant number of levels, as opposed to only two types in [22].

The underlying idea is to ensure that every node gets at least one unit of slack, ensuring that it can be colored as part of a d1LC instance. Easy nodes have such slack from the start; difficult ones get it from their special nodes (special nodes are used in several different ways to provide slack); sparse and ordinary nodes get it from probabilistic slack generation; and non-easy nice ones get it from same-coloring a non-edge it contains. The most challenging part of the low-degree regime is the probabilistic part. That has guided our definition, resulting in the ordinary ACs being defined as restrictively as possible and, in fact, much more restrictive than the ordinary ACs in [22].

3.2 Algorithm for Δ-coloring

Our Δ-coloring algorithm consists of the following five phases.

Algorithm 1 Δ-coloring.

1: Compute an ACD ($\varepsilon = 1/172$) and form the ordered partition of the nodes.
2: Color sparse nodes V_* and ordinary nodes $\mathcal{O}$
3: Color nice nodes $\mathcal{N}$
4: For increasing $1 \leq \ell < \infty$:
    Color difficult nodes $\mathcal{D}_\ell$ in level ℓ
5: Color difficult nodes in $\mathcal{D}_\infty$ and special nodes in $\mathcal{S}$

The remainder of the paper describes these phases in detail. Only Phases 1 and 2 are randomized. Phase 2 is also the most involved part of our algorithm. For ease of presentation, we defer its details when Δ is at most logarithmic to Section 4. In this section, we present Phase 2 in the case of $\Delta \geq c \log n$ for a sufficiently large constant c, where Phase 2 does not require any LLL and which is sufficient to understand how Phase 2 interacts with the remaining phases. The remaining phases are identical in both cases.

3.2.1 Phase 1: Partitioning the Nodes

We first apply Lemma 5 to compute an ACD for $\varepsilon = 1/172$ and break the graph into nice ACs, difficult ACs, ordinary ACs, and the remaining nodes in V_* according to Definition 8.

3.2.2 Phase 2: Sparse and Ordinary Nodes ($\Delta \gg \log n$)

In this subsection, we prove the following lemma.

▶ **Lemma 9.** *There exists a* $\operatorname{poly}\log\log n$*-round* CONGEST *algorithm that w.h.p. colors the sparse nodes and nodes in ordinary cliques if* $\Delta \geq c \log n$ *for a sufficiently large constant* c*.*

Lemma 9 essentially follows from the proof of Lemma 3.5 in [22, arxiv version]. However, as we have changed the definition of ordinary cliques, we spell out the required details.

Slack generation is based on trying a random color for a subset of nodes. Sample a set of nodes and a random color for each of the sampled nodes. Nodes keep the random color if none of their neighbors choose the same color. See Algorithm 3 for a pseudocode. If there are enough non-edges in a node's neighborhood, then it probabilistically gets significant slack.

Algorithm 2 Phase 2: Coloring Sparse and Ordinary Nodes (when $\Delta \gg \log n$).

1: Run SlackGeneration on $V_* \cup \mathcal{O}$
2: Color the remaining ordinary nodes $\mathcal{O}$
3: Color the remaining sparse nodes V_*

Algorithm 3 SlackGeneration.

Input: $S \subseteq V$

1: Each node in $v \in S$ is active w.p. $1/20$
2: Each active node v samples a color r_v u.a.r. from $[\chi]$.
3: v keeps the color r_v if no neighbor tried the same color.

We also require the following lemma from [22].

▶ **Lemma 10** ([22])**.** *Let* C *be a non-easy AC,* $S \subseteq V$ *be a subset of nodes containing* C*, and* M *be an arbitrary matching between* C *and* $N(C) \setminus C$*. Then, after* SlackGeneration *is run on* S*,* C *contains* $\Omega(|M|)$ *uncolored nodes with unit-slack in* $G[S]$*, with probability* $1 - \exp(-\Omega(|M|))$*.*

There exists a large matching satisfying the hypothesis of Lemma 10,

▶ **Lemma 11.** *For each ordinary AC* C*, there exists a matching* M_C *between* C *and* $N(C) \setminus C$ *of size* $2\Delta/5$*.*

Proof. We use the following combinatorial result that is proven in Appendix B for completeness.

▷ Claim 12. Let $B = (Y, U, E_B)$ be a bipartite graph where nodes in Y have degree at least k and nodes in U have degree at most $2k$. There exists a matching of size $|Y|/2$ in B.

Proof. Let M be a maximum matching in B and suppose that more than half the nodes in Y are unmatched. Let S be the set of nodes reachable from the unmatched nodes $Y \setminus V(M)$. Since M has no augmenting path, S contains no unmatched node of U. All of the $|Y \cap S| \cdot k$

edges incident on $Y \cap S$ have their other endpoint in $U \cap S$. By the degree bound on U, there are fewer than $|U \cap S|2k$ such edges. Thus, $|Y \cap S| < 2|U \cap S|$. Every node in $U \cap S$ is matched to a node in $Y \cap S$, while all unmatched nodes in Y are in $Y \cap S$. Thus, the number of unmatched nodes in Y is at most $|Y \cap S| - |U \cap S| < |U \cap S| \leq |M|$. This is a contradiction, and hence, at least half the nodes in Y are matched. ◁

As C is not easy, all its nodes have external degree e_C, while nodes in $N(C) \setminus C$ are by assumption not intrusive neighbors of C, so they have at most $2e_C$ neighbors in C. Claim 12 then implies that there exists a matching between C and $N(C) \setminus C$ of size $|C|/2 \geq (1-\epsilon)\Delta/2 \geq 2\Delta/5$. ◀

The properties of Phase 2 are summarized in the following lemma.

▶ **Lemma 13.** *If $\Delta \geq c \log n$ for a sufficiently large constant c, the following properties hold w.h.p. after Step 1 of Algorithm 2:*
(†) *Each sparse node has unit-slack in $G[V_*]$,*
(††) *Each ordinary AC has an uncolored unit-slack node in $G[V_* \cup \mathcal{O}]$.*

Proof. We run SlackGeneration on the node set $S = V^* \cup \mathcal{O}$. Nodes with neighbors outside $V^* \cup \mathcal{O}$ have slack while the rest of the graph is stalled. We focus on the remaining nodes. Each sparse node gets the respective slack with probability at least $1 - \exp(\Omega(\Delta))$ [18, Lemma 3.1], implying (†). By Lemma 11, there is a matching between C and $N(C) \setminus C$ of size $2\Delta/5$. Thus, (††) holds with probability at least $1 - \exp(-\Omega(\Delta))$, by Lemma 10.

Both probabilities become w.h.p. guarantees if $\Delta \geq c \log n$ for a sufficiently large constant c. For $\Delta \geq \Delta_0$ for a sufficiently large constant Δ_0 we obtain an LLL. ◀

Proof of Lemma 9. By Lemma 13 w.h.p. all sparse nodes become gray as they have unit slack. Also, the unit-slack node in each ordinary AC becomes gray and all other nodes of the AC become grayish as ordinary ACs induce cliques. This is sufficient to color all nodes with $O(1)$ d1LC instances. ◀

Forward pointer. The main difficulty of Phase 2 for smaller values of Δ is to mimic the properties of Lemma 13. Section 4 are devoted to ensuring these properties via several LLLs and d1LC instances that can be solved in a bandwidth-efficient manner.

3.2.3 Phase 3: Nice ACs

We give a simpler treatment than [22]. We want a *toehold* in each nice AC: a node with permanent or temporary slack. With a toehold, the rest is easy. Namely, ACs have all nodes of internal degree at least $(1-\varepsilon)\Delta$, of which none are colored in previous phases. The neighbors of a toehold are gray, and there are at least $(1-\varepsilon/4)\Delta$ of them by Lemma 5, all uncolored. The remaining nodes in the AC are then grayish, so the AC is graytone.

Nice ACs come in three types, depending on if they contain a special node, a non-edge, or a degree-below-Δ node. The first and third types immediately give us a toehold. It remains then to consider nice ACs with a non-edge but with no special node, which we call *hollow*.

For a hollow AC C, we identify an arbitrary non-edge (u_C, w_C) and call it *the pair for C*. We color the pairs for hollow ACs as a d1LC instance. The two nodes in a pair have at least $\Delta/2$ common neighbors within C and any of them can function as a toehold. It remains to argue that we can find a valid coloring of the pairs efficiently.

▶ **Lemma 14.** *The pairs of hollow ACs can be colored in the* CONGEST *model in $O(\log^3 \log n)$ rounds.*

Proof. As the nodes of a hollow C were uncolored, the only nodes that can conflict with the coloring of the pair are the at most $2 \cdot \varepsilon\Delta \leq \Delta/2$ external neighbors. The $\Delta + 1$ colors we have to work with significantly exceed that. Thus, the pairs are $deg + 1$-list colorable.

Both nodes of the pair (u_C, w_C) have at least $(1-\epsilon)\Delta$ neighbors in C, so they have at least $(1-\epsilon)\Delta - (|C| - (1-\epsilon)\Delta) > (1-3\epsilon)\Delta \geq \Delta/2$ common neighbors in C. They provide the bandwidth to transmit to one node all the colors adjacent to the other node. Also, all messages to and from u_C *vis-a-vis* its external neighbors can be forwarded in two rounds. Hence, we can simulate any CONGEST coloring algorithm on the pairs with $O(1)$-factor slowdown; in particular, we can simulate the algorithm from Lemma 2. ◀

3.2.4 Phase 4: Difficult ACs in a Non-Maximum Level

By Definition 7, the special node s_C of any difficult AC C at a level other than D_∞ is contained in another difficult AC $C' \neq C$. The next lemma shows that the level of C' must be strictly larger than the level of C, which allows us to color C fast while C' remains uncolored.

▷ Claim 15. For an AC C with $\ell(C) < \infty$, let C' be the difficult AC that contains the special node s_C. Then we have $\ell(C) < \ell(C')$.

Proof. The special node s_C has external degree of at least $2e_C$ as it is connected to at least $2e_C$ nodes of C that do not lie within C'. Hence, we obtain that the external degree $e_{C'}$ in AC C' is at least $e_{C'} \geq 2e_C$, so $\ell(C') > \ell(C)$. ◁

We color all ACs of a level in parallel, in increasing order of levels. Due to the previous claim, the special node of an AC is contained in a difficult clique in a larger level or not contained in a difficult clique at all. Hence, the special node is uncolored when the clique is processed. So, when processing some level $1 \leq i \leq O(\log \log n)$, we color all nodes in ACs of that level, but we do not color their respective special nodes. Thus, the respective special node provides a toehold for the respective clique.

3.2.5 Phase 5: Difficult ACs in the Maximum Level

The maximum level is processed last and differently from the other levels. By definition, the special node s_C of an AC in ∞ level is not contained in a difficult AC. Also, all nodes in D_∞ and their special nodes are still uncolored at the beginning of this phase.

The algorithm has four steps: (1) Form pairs of selected non-adjacent nodes, (2) Color the nodes in each pair consistently, (3) Graytone color the remaining nodes of the AC, and (4) Color the special nodes $\mathcal{S}$. We explain each step in detail.

First, we form the following pairs. For each special node s_C that is special for only one AC C at level ∞: Form a *type-1* pair $T_s = (s_C, u_C)$ with a non-neighbor of s_C in C. For each special node s that is special for more than one ACs at level ∞, form a *type-2* pair $T_s = (w_1, w_2)$, where w_1 and w_2 are arbitrary non-adjacent nodes in two of the ACs for which s is special. Let $\mathcal{E}$ be the set of the latter special nodes.

▷ Claim 16. The pairs can be properly formed.

Proof. *Type-1:* An (uncolored) non-neighbor u_C of p_C exists as p_C can have at most $(1-\varepsilon/2)\Delta$ neighbors in C by Lemma 5 (4), but the AC C has at least $(1-\varepsilon/4)\Delta$ vertices.

Type-2: Let C_1 and C_2 be two ACs at level ∞ for which s is special, where $e(C_1) \leq e(C_2)$. By definition, s has at least $2e(C_1)$ ($2e(C_2)$) neighbors in C_1 (C_2), respectively. Pick w_1 to be any neighbor of s in C_1. Node w_1 has at most $e(C_1)$ neighbors in C_1. Thus, there are at least $2e(C_2) - e(C_1) > 0$ nodes in C_2 that are neighbors of s and non-neighbors of w_1, and we can pick any such node as w_2. ◁

▶ **Lemma 17.** *Coloring the pairs is a $(deg+1)$-list coloring instance that can be solved in* $\operatorname{poly}\log\log n$ *rounds in* CONGEST*, w.h.p.*

Proof. *Type-1 pair $T=\{s_C,u_C\}$, $s_C\notin C$, $u_C\in C$:* We say that a node *conflicts* with the pair $\{s_C,u_C\}$ if the node is already colored or is contained in an adjacent pair of the same phase. As C does not contain a special node, u_C is the only node of C participating in the phase and all other nodes of C are still uncolored. The node u_C can only be adjacent to e_C conflicting nodes as it has external degree at most e_C. As s_C has at least $2e_C$ neighbors in C, it can conflict with at most $\Delta-2e_C$ nodes. Thus, the pair conflicts with at most $e_C+\Delta-2e_C=\Delta-e_C$ nodes, which is less than Δ, the number of colors initially available. Thus, the problem of coloring such pairs is a $(deg+1)$-list coloring problem.

Type-2 pair $T=\{w_1,w_2\}$: Each such pair (w_1,w_2) is adjacent to at most $e(C_1)+e(C_2)\leq 2\epsilon\Delta$ nodes in other ACs. Further, all nodes in the ACs C_1 and C_2 are still uncolored, so both nodes have at least $(1-2\epsilon)\Delta$ colors in their palette, and each pair is adjacent to at most $2\varepsilon\Delta$ other pairs or already colored neighbors, that is, the palette exceeds the degree.

CONGEST *Implementation.* A type-1 pair has at least $e(C)$ common neighbors (the special node s_C has $2e(C)$ neighbors inside the clique by its definition that are all connected to u_C), which suffices to communicate the colors and all messages of external neighbors of u_C to s_C (u_C has at most e_C external neighbors). Hence, the coloring can be achieved in CONGEST.

Let s be the common special node of a type-2 pair $\{w_1,w_2\}$ and let C_1 and C_2 be the respective cliques. For $i=1,2$ the node w_i has at most e_{C_i} outside neighbors and s has $2e_{C_i}\geq e_{C_i}$ neighbors in C_i, denote these by X_i. We simulate the pair by s. The node w_i can forward all initial colors of outside neighbors as well as all messages from them to s by relaying them through X_i. ◀

After coloring the pairs, each difficult AC C has a node with unit-slack in $G[V\setminus\mathcal{E}]$, either because the clique contains an uncolored node with two neighbors appearing in a consistently colored type-1 pair $T=\{s_C,u_C\}$, or because it contains an uncolored node with a neighbor in $\mathcal{E}$. In the former case, the uncolored node exists because s_C has at least one neighbor in C that is also a neighbor of u_C. In the latter case, the special node s with type-2 pair $T=\{w_1,w_2\}$ has by definition further neighbors besides w_1 and w_2 in each clique that are all uncolored.

Thus, we color all nodes in difficult cliques via the graytone property. At the end, we color the nodes in $\mathcal{E}$, which have unit-slack as they are adjacent to a type-2 pair.

3.3 Proof of Theorem 1

Proof of Theorem 1. There are five cases, depending on the relation of Δ and n. Generally, we use Lemma 2 to solve d1LC instances in $\operatorname{poly}\log\log n$ rounds. Whenever the d1LC instances require additional arguments to be solved in the respective time, e.g., because they are defined on a virtual graph, we reason their runtime when they are introduced.

- If $\Delta=\omega(\log^4 n)$, we use the algorithm from [22] to Δ-color the graph.
- For $c\log n\leq\Delta=O(\log^4 n)$ for a sufficiently large constant c, the result follows by executing Algorithm 1 with the arguments of this section. Phases 1–3 only require $O(1)$ rounds and a constant number of d1LC instances. In Phase 4, we iterate through the $O(\log\Delta)=O(\log\log n)$ levels and solve a constant number of d1LC instances for each level. Phase 5 can be executed in $\operatorname{poly}\log\log n$ time by Lemma 17.

- When $\text{poly}\log\log n \leq \Delta \leq c\log n$, we use Algorithm 1 from this section and replace Phase 2 with Algorithm 4 (presented in Section 4) whose correctness and runtime we prove in Section 4.
- If $\Delta_0 \leq \Delta \leq \text{poly}\log\log n$, we use the algorithm of this section together with the LLL representation from the proof of Lemma 13. The LLL can be solved with the CONGEST LLL solver of [40] in $\text{poly}\,\Delta\,\text{poly}\log\log n = \text{poly}\log\log n$ rounds. Here, Δ_0 is a sufficiently large constant such that the LLL guarantees from Lemma 13 hold.
- If $3 \leq \Delta \leq \Delta_0$, that is, for constant Δ, there is an existing algorithm from [40].

In all cases, the algorithm runs in $\text{poly}\log\log n$ rounds. ◀

4 Phase 2 ($\Delta = O(\log n)$): Sparse Nodes and Ordinary Cliques

In this section, we deal with Phase 2 for the most challenging regime of $\Delta \in O(\log n) \cap \Omega(\text{poly}\log\log n)$. The following lemma follows from all proofs in this section, together with Lemmas 20, 22, and 23 stated in Appendix A and proven in the full version of this paper.

▶ **Lemma 18** (Phase 2). *There exists a* $\text{poly}\log\log n$*-round* CONGEST *algorithm that w.h.p. color the sparse nodes and nodes in ordinary cliques if* $\log^{10}\log n \leq \Delta \leq O(\log n)$.

We first give high-level ideas of our method. We divide the ordinary cliques into the *small*, of size at most $\Delta(1 - 1/(10\log^3\log n))$, and the *large*. Nodes in small ordinary cliques have significant sparsity (i.e., non-edges in their induced neighborhood), which means that the one-round procedure of trying a random color has a good probability of successfully generating slack. The natural LLL formulation of that step is therefore well-behaved enough that it can be solved fast in CONGEST with a few additional tweaks.Large nodes need a different approach.

For each large AC, we produce unit slack for a single node. See Figure 3 for an illustration of the process we will describe. We identify for each such AC a triplet of nodes (x, y, z) with the objective to color x and z with the same color, while y remains uncolored. This way, y receives unit slack, which gives us a toehold to color the whole AC.

Computing such triplets is non-trivial. We do so by breaking it into three steps, each solvable by a different LLL formulation. In brief, we first compute a set Z of candidate z-nodes; next partition Z into two sets; and then select the actual z-nodes to be used from these two sets. The split of Z into two sets is required to make the process of finally finding the z-nodes fit the LLL solver from [33]. The properties of the set Z imply that it is then much easier to identify compatible x- and y-nodes, and once we find such triplets, we set up a virtual coloring instance for same-coloring x- and z-nodes in each triple. We show that this instance is d1LC and can be solved with low bandwidth despite being defined on a virtual graph. This provides a slack-toehold to the y-node of each triple and the coloring can be extended via d1LC instances to the whole instance.

Algorithm. The first step of the algorithm is to compute a large matching M_C between each ordinary clique C and $N(C) \setminus C$ in parallel. We then classify the ordinary cliques as follows. Fix the parameter $q(n) = 10\log^3\log n$ throughout this section.

▶ **Definition 19** (Small, Large, Unimportant and Important Ordinary cliques.). *An ordinary AC is* large *if it contains more than* $\Delta - \Delta/q(n)$ *nodes, and* small *otherwise. A large AC is* important *if* $|(V(M_C) \setminus C) \cap \mathcal{O}_l| \geq \Delta/12$*, and* unimportant *otherwise.*

We say that a node is small/large/important/unimportant if it belongs to an AC of the corresponding type. Let $\mathcal{O}_i$, $\mathcal{O}_u$, $\mathcal{O}_l = \mathcal{O}_i \cup \mathcal{O}_u$, and $\mathcal{O}_s$ be the set of important, unimportant, large, and small nodes, respectively.

Next, we summarize the high level steps of the algorithm.

Algorithm 4 Phase 2: Coloring Sparse and Ordinary Nodes ($\Delta = O(\log n)$).

1: Step 0: For each ordinary AC C in parallel, compute a matching $M_C \subseteq C \times (N(C) \setminus C)$. Classify ordinary ACs into important, unimportant, and small ACs.
2: Step 1: Generate slack for sparse and small nodes (via LLL, see full version)
3: Step 2: Compute candidate sets $Z = Z_1 \cup Z_2 \subseteq \mathcal{O}_l$ (via LLL, see full version)
4: Step 3: Form triples $(x_C, y_C, z_C) \in C \times C \times Z$ (via LLL, see full version)
5: Step 4: Same-color (x, z)-pairs via virtual coloring instance
6: Step 5: Color the remainder of $V^* \cup \mathcal{O}$ (via d1LC instances).

Steps 0,4, and 5 are detailed in Appendix A. While Steps 1–3 are the heart of this paper, their treatment is extremely technical and requires setting up several involved LLLs that are then solved with the LLL solvers of [33]. Thus, the complete treatment of these steps is defered to the full version of the paper and Appendix A only focuses on presenting the guarantees provided by these steps (Lemmas 20, 22, and 23).

References

1 Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for Δ-coloring. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 234–247, 2022.

2 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed Δ-coloring plays hide-and-seek. In *Proc. 54th ACM Symp. on Theory of Computing (STOC)*, 2022.

3 Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally checkable labelings with small messages. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.8.

4 Étienne Bamas and Louis Esperet. Distributed coloring of graphs with an optimal number of colors. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126 of *LIPIcs*, pages 10:1–10:15. LZI, 2019. doi:10.4230/LIPICS.STACS.2019.10.

5 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 243–252, 2020. doi:10.1145/3382734.3404504.

6 L. Barenboim. Deterministic $(\Delta + 1)$-coloring in sublinear (in Δ) time in static, dynamic and faulty networks. In *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 345–354, 2015.

7 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

8 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM*, 63(3):20:1–20:45, 2016. doi:10.1145/2903137.

9 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In *Proc. 48th ACM Symposium on Theory of Computing (STOC 2016)*, pages 479–488. ACM, 2016. doi:10.1145/2897518.2897570.

10 R. Leonard Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941. doi:10.1017/S030500410002168X.

11 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed edge coloring and a special case of the constructive Lovász local lemma. *ACM Trans. Algorithms*, 2020. doi:10.1145/3365004.

12 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed (Δ+1)-coloring algorithm? In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 445–456, 2018. doi:10.1145/3188745.3188964.

13 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM J. Comput.*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

14 Shiri Chechik and Doron Mukhtar. Optimal distributed coloring algorithms for planar graphs in the LOCAL model. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 787–804. SIAM, 2019. doi:10.1137/1.9781611975482.49.

15 Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the Lovász local lemma and graph coloring. *Distributed Comput.*, 30(4):261–280, 2017. doi:10.1007/S00446-016-0287-6.

16 Sam Coy, Artur Czumaj, Peter Davies, and Gopinath Mishra. Parallel derandomization for coloring, 2024. Note: https://arxiv.org/abs/2302.04378v1 contains the Delta-coloring algorithm. arXiv:2302.04378.

17 Peter Davies. Improved distributed algorithms for the Lovász local lemma and edge coloring. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4273–4295. SIAM, 2023. doi:10.1137/1.9781611977554.CH163.

18 Michael Elkin, Seth Pettie, and Hsin-Hao Su. $(2\Delta - 1)$-edge-coloring is much easier than maximal matching in the distributed setting. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 355–370, 2015. doi:10.1137/1.9781611973730.26.

19 Paul Erdös and László Lovász. Problems and Results on 3-chromatic Hypergraphs and some Related Questions. *Colloquia Mathematica Societatis János Bolyai*, pages 609–627, 1974.

20 Manuela Fischer. Improved deterministic distributed matching via rounding. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.17.

21 Manuela Fischer and Mohsen Ghaffari. Sublogarithmic Distributed Algorithms for Lovász Local Lemma, and the Complexity Hierarchy. In *the Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, pages 18:1–18:16, 2017. doi:10.4230/LIPIcs.DISC.2017.18.

22 Manuela Fischer, Magnús M. Halldórsson, and Yannic Maus. Fast distributed Brooks' theorem. In *Proceedings of the SIAM-ACM Symposium on Discrete Algorithms (SODA)*, pages 2567–2588, 2023. doi:10.1137/1.9781611977554.ch98.

23 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 625–634, 2016. doi:10.1109/FOCS.2016.73.

24 Marc Fuchs and Fabian Kuhn. List defective colorings: Distributed algorithms and applications. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 22:1–22:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.22.

25 Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proc. 30th Symp. on Discrete Algorithms (SODA)*, pages 805–820, 2019. doi:10.1137/1.9781611975482.50.

26 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 662–673, 2018. doi:10.1109/FOCS.2018.00069.

27 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved distributed delta-coloring. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 427–436, 2018. URL: https://dl.acm.org/citation.cfm?id=3212764.

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

29 Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. Efficient randomized distributed coloring in CONGEST. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 1180–1193, 2021. Full version at CoRR abs/2105.04700. doi:10.1145/3406325.3451089.

30 Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan. Near-optimal distributed degree+1 coloring. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 450–463. ACM, 2022. doi:10.1145/3519935.3520023.

31 Magnús M. Halldórsson, Yannic Maus, and Alexandre Nolin. Fast distributed vertex splitting with applications. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 26:1–26:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DISC.2022.26.

32 Magnús M. Halldórsson and Alexandre Nolin. Superfast coloring in CONGEST via efficient color sampling. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2021. doi:10.1007/978-3-030-79527-6_5.

33 Magnús M. Halldórsson, Yannic Maus, and Saku Peltonen. Distributed Lovász local lemma under bandwidth limitations, 2024. arXiv:2405.07353, doi:10.48550/arXiv.2405.07353.

34 Magnús M. Halldórsson, Alexandre Nolin, and Tigran Tonoyan. Overcoming congestion in distributed coloring. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 26–36. ACM, 2022. doi:10.1145/3519270.3538438.

35 David G. Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\Delta + 1)$-coloring in sublogarithmic rounds. *Journal of the ACM*, 65:19:1–19:21, 2018. doi:10.1145/3178120.

36 Öjvind Johansson. Simple distributed $\Delta + 1$-coloring of graphs. *Inf. Process. Lett.*, 70(5):229–232, 1999.

37 Nati Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

38 Yannic Maus, Saku Peltonen, and Jara Uitto. Distributed symmetry breaking on power graphs via sparsification. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, PODC '23, pages 157–167, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3583668.3594579.

39 Yannic Maus and Tigran Tonoyan. Local conflict coloring revisited: Linial for lists. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.16.

40 Yannic Maus and Jara Uitto. Efficient CONGEST algorithms for the Lovász local lemma. In Seth Gilbert, editor, *Proceedings of the International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 31:1–31:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.31.

41 Alessandro Panconesi and Aravind Srinivasan. The local nature of Δ-coloring and its algorithmic applications. *Combinatorica*, 15(2):255–280, 1995. doi:10.1007/BF01200759.

42 Luke Postle. Linear-time and efficient distributed algorithms for list coloring graphs on surfaces. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 929–941. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00060.

43 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 350–363, 2020.

A Details of Phase 2

Step 0: Classifying ACs and computing matchings. We compute a matching M_C for each ordinary clique C between the vertices in C and the ones in $N(C) \setminus C$. We use a 2.5-approximate algorithm of [20] running in $O(\log^2 \Delta + \log^* n) = O(\log^2 \log n)$ rounds, obtaining that $|M_C| \geq (2\Delta/5)/2.5 = \Delta/10$, using Lemma 11.

We view the edges of M_C as being directed arcs with a head in C and tail in $V \setminus C$. Each AC can determine its size and the size of $V(M_C) \cap \mathcal{O}_l$ in $O(1)$ rounds and hence the classification of Definition 19 can be computed in $O(1)$ rounds.

Step 1: Slack for sparse and small nodes. In this step, we create slack for sparse nodes and all nodes in $\mathcal{O}_s$. The key property of small nodes is that they are relatively sparse (with many non-edges in their neighborhoods), so randomly trying colors is likely to produce slack. That leads to an LLL formulation that we can make simulatable and can therefore implement in CONGEST.

The properties are summarized by the following lemma. Besides providing slack to all sparse nodes and the nodes in small ordinary ACs, it also guarantees that each neighborhood (and hence also each AC) does not have too many nodes colored and that the matching M_C of each AC does not get too many nodes colored.

▶ **Lemma 20.** *Assume that we are given a matching M_C of size at least $\Delta/10$ between C and $N(C) \setminus C$ for each ordinary AC C. There is a* $\operatorname{poly}\log\log n$*-round (LLL-based)* CONGEST *algorithm that w.h.p. colors a subset $S \subseteq V^* \cup \mathcal{O}$ and ensures that:*

1. *Each uncolored node in $V^* \cup \mathcal{O}_s$ has unit-slack in $G[V^* \cup \mathcal{O}]$.*
2. *In each of the following subsets, at most $O(\log^4 \log n \cdot \log \Delta)$ nodes are colored: $N(v)$ for each $v \in V^* \cup \mathcal{O}$ and $V(M_C)$ for each AC C .*

Step 2: Compute triple candidate set via LLL. Let $X = \mathcal{O}_l \setminus \{v \in \mathcal{O}_l : v \text{ colored in Step 1}\}$.

The goal of this step is to compute two disjoint sets Z_1, Z_2 of uncolored nodes such that each important AC has sufficiently many matching edges satisfying the following definition of usefulness.

▶ **Definition 21** (useful edge)**.** *Given a subset $Z \subseteq X$ and important AC C, a matched arc $\overrightarrow{vu} \in M_C$ is* useful *for C if $v \in (X \setminus Z)$ and $u \in Z$. Refer to Z as the* black *nodes and $X \setminus Z$ as the* white *nodes. An edge is* white *if both endpoints are* white*.*

An arc $\overrightarrow{vz}$ cannot be useful for the AC containing v; only the one containing z.

Formally, Step 2 provides the following lemma. For an AC C and set Z, let $U(C, Z)$ denote the arcs of M_C with one endpoint in Z (and the other in C).

▶ **Lemma 22.** *Let $q = 1/30$. There is a* poly log log n*-round (LLL-based)* CONGEST *algorithm computing disjoint subsets $Z_1, Z_2 \subseteq \mathcal{O}_l$ satisfying the following properties, w.h.p.:*

1. $|U(C, Z_i)| \geq q^2(1-q)^3\Delta/60$*, for $i = 1, 2$ and for each important AC C, and*
2. $|(Z_1 \cup Z_2) \cap N(v)| \leq \Delta/10$*, for all $v \in \mathcal{O}$.*

Step 3: Forming triples via LLL. The goal of this step is to compute a triple $(x_C, y_C, z_C) \in C \times C \times Z$ of nodes that satisfy the conditions of the next lemma. These triple nodes are distinct for different ACs.

▶ **Lemma 23.** *Given sets $Z_1, Z_2 \subseteq \mathcal{O}_l$ with the properties as in Lemma 22, there is a* poly log log n*-round (LLL-based)* CONGEST *algorithm that computes for each large important AC C a triple (x_C, y_C, z_C) of uncolored nodes such that w.h.p.:*

1. *$x_C, y_C \in C$ and $z_C \notin C$,*
2. *$y_C x_C$, $y_C z_C \in E$, $x_C z_C \notin E$ (x_C and z_C are non-adjacent; y_c is adjacent to both x_C and z_C) and*
3. *the graph induced by $\{z_C : C$ is important$\}$ has maximum degree $\leq \Delta/10$.*

We model the problem of selecting z_C for each important AC C as a disjoint variable set LLL.

Step 4: Same-coloring (x_C, z_C) pairs. Given a triple (x_C, y_C, z_C), we will create a toehold for the AC C at y_C by coloring its non-adjacent neighbors x_C and z_C with the same color.

Let H_P (P for pair) be the virtual graph consisting of one vertex for each pair (s_C, z_C) and an edge between two pairs (s_C, z_C) and $(s_{C'}, z_{C'})$ if there is any edge in G between $\{s_C, z_C\}$ and $\{s_{C'}, z_{C'}\}$. The *list of available colors* $L((s_C, z_C))$ consists of all colors that are not used by the already colored neighbors in G of s_C and z_C.

▶ **Lemma 24.** *The maximum degree Δ_{H_P} of H_P is upper bounded by $\Delta/9$.*

Proof. By Lemma 23, each node has at most $\Delta/10$ neighbors in Z. Define the set $X' = \{x_C : C$ is an important AC$\}$. As X' contains at most one node per AC, the number of neighbors that a node in $\mathcal{O}_l$ can have in U is upper bounded by its external degree plus 1, which is upper bounded by $\Delta/q(n) + 1$. Thus, the maximum degree Δ_{H_P} of the virtual graph H_P is at most $\Delta/10 + \Delta/q(n) + 1 \leq \Delta/9$ for sufficiently large n. ◀

▶ **Lemma 25.** *Coloring H_P – i.e., same-coloring the pairs – is a $deg+1$-list coloring instance.*

Proof. By Lemma 24 we obtain $\Delta_{H_P} \leq \Delta/9$. As we colored at most $x = O(\log^5 \log n)$ vertices in each neighborhood in Step 1, the list of available colors of each pair has at least $\Delta - 2x \gg \Delta/9 = \Delta_{H_P}$ colors available in their joint list. Hence, we obtain a $deg + 1$-list coloring instance. ◀

CONGEST implementation. Our algorithm is based on the $deg + 1$-list coloring algorithm from [25, 8]. Before we show how to color the nodes in H_P, we need to define a slow (it takes $O(\log n)$ rounds) randomized algorithm. The algorithm is used in our analysis and it works as follows. In each iteration, each uncolored pair executes the following procedure that may result in the pair to try to get colored with a color or to not try a color (also see Algorithm 5 for pseudocode of the algorithm). Throughout the algorithm, nodes x_C and z_C maintain lists $L(x_C)$ and $L(z_C)$ consisting of all colors not used by their respective neighbors in G. Then, in one iteration node x_C selects a color c u.a.r. from its list of available colors $L(x_C)$, and sends it to the other endpoint through node y_C. The other endpoint z_C checks whether

$c \in L(z_C)$; if so, both nodes agree on trying color c, and the color is sent to their neighbors. If no incident pair tries the same color, the pair gets permanently colored with the color. Lastly, both nodes individually update their lists by removing colors from adjacent vertices that got colored from their respective list. There is no explicit coordination between the two vertices in maintaining a joint list of available colors.

Algorithm 5 Randomized Pair Coloring.

1: Each node x_C selects a color c u.a.r. from $L(x_C)$ and sends c to z_C
2: If $c \in L(z_C)$ then TryColor(c)
3: Update lists $L(x_C) \leftarrow L(x_C) \setminus \{c(v) : v \in N_G(x_C)\}$ and $L(z_C) \leftarrow L(z_C) \setminus \{c(v) : v \in N_G(z_C)\}$

The next lemma shows that each pair gets colored with constant probability.

▶ **Lemma 26.** *Consider an arbitrary iteration of Algorithm 5 and an arbitrary pair (x_C, z_C) for a hiding AC C that is uncolored at the start of the iteration. Then, we have*

$$\Pr((x_C, z_C) \text{ gets colored in the iteration}) \geq 1/2 \ . \tag{1}$$

The bound on the probability holds regardless of the outcome of previous iterations.

Proof. Note [2] that throughout the execution of Algorithm 5 the respective lists of nodes x_C and z_C are always of size at least $\Delta - \Delta_{H_P} - \Omega(\log^5 \log n) \geq 4\Delta/5$ as $\Delta = \omega(\log^5 \log n)$ and $\Delta_{H_P} \leq \Delta/9$, by Lemma 24. Note, that both nodes keep their individual list of available colors in which they only remove the colors of immediate neighbors in G from the list of available colors. Thus, at all times we have $|L(x_C)| \cap L(z_C)| \geq 3\Delta/5$. Let X be the set of colors tried by one of the $\Delta_{H_P} \leq \Delta/9$ pairs incident to (s_C, z_C) in the current iteration. We obtain $|(L(s_C) \cap L(z_C)) \setminus X| \geq \Delta/2$. As these colors are at least half of $L(x_C)$'s palette, the probability that the pair (x_C, z_C) gets colored is at least $1/2$. ◀

▶ **Lemma 27.** *There is a randomized* poly log log n*-round* CONGEST *algorithm that w.h.p. colors the pairs of H_P.*

Proof. Consider the well-understood color trial algorithm in which nodes repeatedly try a color from their list of available colors, keep their color permanently if no neighbor tries the same color, and remove colors of permanently colored neighbors from their list of available colors. It is known that this algorithm colors each node with a constant probability in each iteration [8, 36]. Thus, it requires $O(\log n)$ rounds to color all vertices of a graph. The shattering-based CONGEST algorithm from [25] for d1LC runs in poly log log n rounds. It requires three subroutines: a) A color trial algorithm like the one from [8, 36], b) a network decomposition algorithm that can run on small subgraphs (the ones in [43, 40, 38] do the job), and c) the possibility to run $O(\log n)$ instances of the color trial algorithm in parallel. In our setting we want to solve the same problem, but on the virtual graph H_p while the communication network is still the original graph G. The subroutine for part b) can be taken from prior work as the same issue is dealt with formally in [40, 38, 33]. We refer to these works for the details and also the definition of a network decomposition. Let us sketch the main ingredient for the informed reader. Instead of computing a network decomposition of small subgraphs of H_P, the subgraphs are first projected to G, and a network decomposition

[2] The constants in this proof are not chosen optimally in order to improve readability.

of G is computed afterwards. This only requires an increased distance between clusters such that the preimage of the decomposition induces a proper network decomposition of H_P.

For ingredients a) and c), we observe that Ghaffari's algorithm only requires the following properties for the color trial algorithm: i) one iteration can be executed in constant time and with $\operatorname{poly}\log\log n$ bandwidth, allowing to execute $O(\log n)$ instances in parallel in the CONGEST model, and ii) each node gets colored with a constant probability in each iteration. Thus, we can replace the color trial algorithm with the color trial algorithm for H_P given in Algorithm 5. We have already argued that it can be implemented with $\operatorname{poly}\log\log n$ bandwidth showing *i*) and Lemma 26 provides its constant success probability for ii). ◀

Step 5: Completing the coloring. To finish the coloring, we first color the unimportant nodes and then the important, small, and sparse nodes.

▶ **Lemma 28.** *Unimportant nodes are graytone as long as the other ordinary nodes (small, sparse, important) are inactive.*

Proof. The only steps so far in which we colored vertices are Steps 1 and 4. In Step 1 we color at most $O(\log^5 \log n)$ vertices per AC and per matching M_C of each ordinary AC C. In Step 4 we only color (a subset of) the vertices in Z and one vertex per important AC (the vertex x_C for AC C). As $|Z \cap C| \leq \Delta/10$, we color at most $\Delta/10 + O(\log^5 \log n) \leq \Delta/9$ vertices in each unimportant AC.

Fix some unimportant AC C. Recall that the algorithm of [20] finds a 2.5-approximate matching, which by Lemma 11 implies that $|M_C| \geq \Delta/10$. As an unimportant AC has fewer than $\Delta/12$ nodes in $(V(M_C) \setminus C) \cap \mathcal{O}_l$, we obtain that $V(M_C) \setminus C$ contains at least $\Delta/10 - \Delta/12 = 7\Delta/60$ nodes that are not contained in $\mathcal{O}_l$. By Lemma 20, at most $O(\log^5 \log n)$ of these get colored in Step 1; denote the uncolored nodes of these by S and let $S' = N(S) \cap C$. By the earlier argument, at most $\Delta/9$ nodes of S' are already colored, that is, there exists some $v \in S'$ that is still uncolored and has an uncolored neighbor $u \notin \mathcal{O}_l$. As u is stalled to be colored later, v is gray and other nodes of the AC are grayish. ◀

▶ **Lemma 29.** *Small, sparse, and important nodes are graytone.*

Proof. By Lemma 20, each small or sparse node has slack in $G[V^* \cup \mathcal{O}]$ and is therefore gray (and stays gray until colored).

For an important AC C with triple (x_C, y_C, z_C), the node y_C is gray as x_C and z_C are same-colored. Hence, the remaining uncolored nodes of C are either already colored or graytone as they are adjacent to v. ◀

B Computing the ACD

We adapt a proof from [22] that, as stated, applies only to the case when Δ is sufficiently large. Technically, the argument differs only in that we build on [34] instead of [29] in the first step of the argument, where we compute a decomposition with weaker properties. We have opted to rephrase it, given the different constants in the definitions of these works and in order to make it more self-contained.

▶ **Lemma 5** (ACD computation [1, 22])**.** *For any graph $G = (V, E)$, there is a partition (*almost-clique decomposition (ACD)*) of V into sets V_{sparse} and $C_1, C_2, \ldots, C_t$ such that each node in V_{sparse} is $\Omega(\epsilon^2\Delta)$-sparse and for every $i \in [t]$,*

- **(i)** $(1 - \varepsilon/4)\Delta \leq |C_i| \leq (1 + \varepsilon)\Delta$,
- **(ii)** *Each $v \in C_i$ has at least $(1 - \varepsilon)\Delta$ neighbors in C_i: $|N(v) \cap C_i| \geq (1 - \varepsilon)\Delta$,*

(iii) *Each node $u \notin C_i$ has at most $(1-\varepsilon/2)\Delta$ neighbors in C_i: $|N(u) \cap C_i| \leq (1-\varepsilon/2)\Delta$. Further, there is an $O(1)$-round* CONGEST *algorithm to compute a valid ACD, w.h.p.*

Proof. We first use a $O(1)$-round CONGEST algorithm of [HNT22] to compute a weaker form of ACD with parameter $\varepsilon/4$.[3] Namely, it computes w.h.p. a partition $(V', D_1, D_2, \ldots, D_k)$ where nodes in V' are $\Omega(\Delta)$-sparse and we have, for each $i \in [k]$:
(a) $|D_i| \leq (1+\varepsilon/4)\Delta$, and
(b) $|N(v) \cap D_i| \geq (1-\varepsilon/4)\Delta$, for each $v \in D_i$.
What this construction does not satisfy is condition (iii).

We form a modified decomposition $(V_{sparse}, C_1, \cdots, C_k)$ as follows. For each $i \in [t]$, let C_i consist of D_i along with the nodes in V' with at least $(1-\varepsilon)\Delta$ neighbors in D_i. Let $V_{sparse} = V \setminus \cup_i C_i$. Observe that the decomposition is well-defined, as a node $u \in V'$ cannot have $(1-\varepsilon)\Delta > \Delta/2$ neighbors in more than one D_i.

We first bound from above the number of nodes added to each part C_i. Each node in D_i has at most $\varepsilon\Delta/4$ outside neighbors, so the number of edges with exactly one endpoint in D_i is at most $\varepsilon\Delta|D_i|/4 \leq \varepsilon(1+\varepsilon/4)\Delta^2/4$, using (a) to bound $|D_i|$. Each node in $C_i \setminus D_i$ is incident on at least $(1-\varepsilon)\Delta$ such edges (by definition). Thus,

$$|C_i \setminus D_i| \leq \varepsilon/4 \cdot (1+\varepsilon/4)\Delta/(1-\varepsilon) \leq \varepsilon\Delta/2 \; . \tag{2}$$

Now, (iii) holds since a node outside C_i has at most $(1-\varepsilon)\Delta$ neighbors in D_i (by the definition of C_i) and at most $|C_i \setminus D_i| \leq \epsilon\Delta/2$ other neighbors in C_i (by Equation (2)). Also, (ii) holds for nodes in D_i by (b) and for nodes in $C_i \setminus D_i$ by the definition of C_i. For the lower bound in (i), $|C_i| \geq |D_i| \geq (1-\varepsilon/4)\Delta$, by (b). For the upper bound of (i), we have $|C_i| \leq |D_i| + |C_i \setminus D_i| \leq (1+3\epsilon/4)\Delta$ (by (a) and Equation (2)).

Finally, the claim about V_{sparse} follows from the definition of V', as $V_{sparse} \subseteq V'$. ◀

[3] While such a statement is used in the paper, it is not explicitly stated. Alternatively, we may use an alternative (slower) implementation (Lemma 4.4) in [Flin et al (FGHKN22), arXiv:2301.06457]] that runs $O(\log\log n)$ rounds for $\Delta = \text{poly}(\log n)$ and still suffices for our main result. The slowdown in [FGHKN22] comes from working with sparsified graphs, while a CONGEST version also runs in $O(1)$ rounds.