




Optimal (degree+1)-Coloring in Congested Clique

Sam Coy   

University of Warwick, UK

Artur Czumaj   

University of Warwick, UK

Peter Davies   

Durham University, UK

Gopinath Mishra   

University of Warwick, UK

Abstract

We consider the distributed complexity of the (degree+1)-list coloring problem, in which each node u of degree $d(u)$ is assigned a palette of $d(u) + 1$ colors, and the goal is to find a proper coloring using these color palettes. The (degree+1)-list coloring problem is a natural generalization of the classical $(\Delta + 1)$ -coloring and $(\Delta + 1)$ -list coloring problems, both being benchmark problems extensively studied in distributed and parallel computing.

In this paper we settle the complexity of the (degree+1)-list coloring problem in the Congested Clique model by showing that it can be solved deterministically in a constant number of rounds.

2012 ACM Subject Classification Theory of computation \rightarrow Massively parallel algorithms; Theory of computation \rightarrow Distributed algorithms; Theory of computation \rightarrow Pseudorandomness and derandomization; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Distributed computing, graph coloring, parallel computing.

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.99

Category Track A: Algorithms, Complexity and Games

Funding *Sam Coy*: Research supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP), by an EPSRC studentship, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

Artur Czumaj: Research supported in part by the Centre for Discrete Mathematics and its Applications, by EPSRC award EP/V01305X/1, by a Weizmann-UK Making Connections Grant, by an IBM Award, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

Gopinath Mishra: Research supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP), by EPSRC award EP/V01305X/1, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

1 Introduction

Graph coloring problems are among the most extensively studied problems in the area of distributed graph algorithms. In the distributed graph coloring problem, we are given an undirected graph $G = (V, E)$ and the goal is to properly color the nodes of G such that no edge in E is monochromatic. In the distributed setting, the nodes of G correspond to devices that interact by exchanging messages throughout some underlying communication network such that the nodes communicate with each other in synchronous rounds by exchanging messages over the edges in the network. Initially, the nodes do not know anything about G (except possibly for some global parameters, e.g., the number of nodes n or the maximum



© Sam Coy, Artur Czumaj, Peter Davies, and Gopinath Mishra;
licensed under Creative Commons License CC-BY 4.0

50th International Colloquium on Automata, Languages, and Programming (ICALP 2023).

Editors: Kousha Etessami, Uriel Feige, and Gabriele Puppis; Article No. 99; pp. 99:1–99:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

degree Δ). At the end of computation, each node $v \in V$ should output its color (from a given palette) in the computed coloring. The *time or round complexity* of a distributed algorithm is the total number of rounds until all nodes terminate.

If adjacent nodes in G can exchange arbitrarily large messages in each communication round (and hence the underlying communication network is equal to the input graph G), this distributed model is known as the LOCAL model [30], and if messages are restricted to $O(\log n)$ bits per edge (limited bandwidth) in each round, the model is known as the CONGEST model [38]. If we allow all-to-all communication (i.e., the underlying network is a complete graph and thus the communication is independent of the input graph G) using messages of size $O(\log n)$ bits then the model is known as the CongestedClique model [31].

The most fundamental graph coloring problem in distributed computing (studied in the seminal paper by Linial [30] that introduced the LOCAL model) is $(\Delta + 1)$ -coloring: assuming that the input graph G is of maximum degree Δ , the objective is to color nodes of G using $\Delta + 1$ colors from $\{1, 2, \dots, \Delta + 1\}$. The $(\Delta + 1)$ -coloring problem can be easily solved by a sequential greedy algorithm, but the interaction between local and global aspects of graph coloring create some non-trivial problems in a distributed setting. The problem has been used as a benchmark to study distributed symmetry breaking in graphs, and it is at the very core of the area of distributed graph algorithms. $(\Delta + 1)$ -list coloring is a natural generalization of $(\Delta + 1)$ -coloring: each node has an arbitrary palette of $\Delta + 1$ colors, and the goal is to compute a legal coloring in which each node is assigned a color from its own palette. A further generalization is the *(degree+1)-list coloring (DILC)* problem, which is the same as the $(\Delta + 1)$ -list coloring problem except that the size of each node v 's palette is $d(v) + 1$, which might be much smaller than $\Delta + 1$. These three problems always have a legal coloring (easily found sequentially using a greedy approach), and the main challenge in the distributed setting is to find the required coloring in as few rounds as possible.

These three graph coloring problems have been studied extensively in distributed computing, though $(\Delta + 1)$ -coloring, as the simplest, has attracted most attention. However, one can also argue that (degree+1)-list coloring, as the most versatile, is more algorithmically fundamental than $(\Delta + 1)$ -coloring. For example, given a partial solution to a $(\Delta + 1)$ -coloring problem, the remaining coloring problem on the uncolored nodes is an instance of the (degree+1)-list coloring problem. The (degree+1)-list coloring problem is self-reducible: after computing a partial solution to a (degree+1)-list coloring problem, the remaining problem is still a (degree+1)-list coloring problem. It also naturally appears as a subproblem in more constrained coloring problems: for example, it has been used as a subroutine in distributed Δ -coloring algorithms (see, e.g., [19]), in efficient $(\Delta + 1)$ -coloring and edge-coloring algorithms (see, e.g., [28]), and in other graph coloring applications (see, e.g. [4]).

Following an increasing interest in the distributed computing community for (degree+1)-list coloring, it is natural to formulate a central challenge relating it to $(\Delta + 1)$ -coloring:

Can we solve the (degree+1)-list coloring problem in asymptotically the same round complexity as the simpler $(\Delta + 1)$ -coloring problem?

This challenge has been elusive for many years and only in the last year the affirmative answer was given for randomized algorithms in LOCAL and CONGEST. First, in a recent breakthrough, Halldórsson, Kuhn, Nolin, and Tonoyan [24] gave a randomized $O(\log^3 \log n)$ -round distributed algorithm for (degree+1)-list coloring in the LOCAL model, matching the state-of-the-art complexity for the $(\Delta + 1)$ -coloring problem [10, 40]. This has been later

extended to the CONGEST model by Halldórsson, Nolin, and Tonoyan [25], who designed a randomized algorithm for $(\text{degree}+1)$ -list coloring that runs in $O(\log^5 \log n)$ -round, matching the state-of-the-art complexity for the $(\Delta + 1)$ -coloring problem in CONGEST [23].

The main contribution of our paper is a complete resolution of this challenge in the CongestedClique model, and in fact, even for deterministic algorithms. We settle the complexity of the $(\text{degree}+1)$ -list coloring problem in CongestedClique by showing that it can be solved deterministically in a constant number of rounds.

► **Theorem 1.** *There is a deterministic CongestedClique algorithm which finds a $(\text{degree}+1)$ -list coloring of any graph in a constant number of rounds.*

1.1 Background and Related Works

Distributed graph coloring problems have been extensively studied in the last three decades, starting with a seminal paper by Linial [30] that introduced the LOCAL model and originated the area of local graph algorithms. Since the $(\Delta + 1)$ -coloring problem can be solved by a simple sequential greedy algorithm, but it is challenging to be solved efficiently in distributed (and parallel) setting, the $(\Delta + 1)$ -coloring problem became a benchmark problem for distributed computing and a significant amount of research has been devoted to the study of these problems in all main distributed models: LOCAL, CONGEST, and CongestedClique. The monograph [6] gives a comprehensive description of many of the earlier results.

It is long known from research on parallel algorithms that $(\Delta + 1)$ -coloring can be computed in $O(\log n)$ rounds by randomized algorithms in the LOCAL model [2, 32]. Linial [30] observed that for smaller values of Δ , one can do better: he showed that it is possible to deterministically color arbitrary graphs of maximum degree Δ with $O(\Delta^2)$ colors in $O(\log^* n)$ rounds; this can be easily extended to obtain a deterministic LOCAL algorithm for $(\Delta + 1)$ -coloring that runs in $O(\Delta^2 + \log^* n)$ rounds, and thus in bounded degree graphs, a $(\Delta + 1)$ -coloring can be computed in $O(\log^* n)$ rounds. Improve results have since been found for general values of Δ : the current state-of-the-art for the $(\Delta + 1)$ -coloring problem in LOCAL is $O(\log^3 \log n)$ rounds for randomized algorithms [10, 40] and $O(\log^2 \Delta \cdot \log n)$ for deterministic algorithms [22]. Furthermore, the fastest algorithms mentioned above can be modified to work also for the more general $(\Delta + 1)$ -list coloring problem in the LOCAL model. (In fact, many of those algorithms critically rely on this problem as a subroutine.)

For the CONGEST model, the parallel algorithms mentioned above [2, 32] can be implemented in the CONGEST model to obtain randomized algorithms for both the $(\Delta+1)$ -coloring and $(\Delta + 1)$ -list coloring problems that run in $O(\log n)$ rounds. Only recently this bound has been improved for all values of Δ : In a seminal paper, Halldórsson et al. [23] designed a randomized CONGEST algorithm that solves the $(\Delta + 1)$ -coloring and $(\Delta + 1)$ -list coloring problems in $O(\log^5 \log n)$ rounds. For deterministic computation, the best LOCAL algorithm [22] works directly in CONGEST, running in $O(\log^2 \Delta \cdot \log n)$ rounds.

As for the lower bounds, one of the first results in distributed computing was a lower bound in LOCAL of $\Omega(\log^* n)$ rounds for computing an $O(1)$ -coloring of a graph of maximum degree $\Delta = 2$, shown by Linial [30] for deterministic algorithms, and by Naor [35] for randomized ones. Improved coloring lower bounds have not been forthcoming, and $\Omega(\log^* n)$ rounds is still the best known lower bound complexity for the $(\Delta + 1)$ -coloring problem in LOCAL and CONGEST.

This lower bound does not hold in the CongestedClique model, and in fact we can color faster. After years of gradual improvements, Parter [36] exploited the LOCAL shattering approach from [10] to give the first sublogarithmic-time randomized $(\Delta+1)$ -coloring algorithm

for CongestedClique, which runs in $O(\log \log \Delta \cdot \log^* \Delta)$ rounds. This bound was later improved by Parter and Su [37] to $O(\log^* \Delta)$ rounds. Finally, Chang et al. [9] settled the randomized complexity of $(\Delta + 1)$ -coloring (and also for $(\Delta + 1)$ -list coloring) and obtained a randomized CongestedClique algorithm that runs in a constant number of rounds. This result has been later simplified and extended into a deterministic constant-round CongestedClique algorithm by Czumaj et al. [15].

(degree+1)-list coloring (D1LC). The D1LC problem in distributed setting has been studied both on its own, and also as a tool in designing distributed algorithms for other coloring problems, like $(\Delta + 1)$ -coloring, $(\Delta + 1)$ -list coloring, and Δ -coloring. The problem is more general than the $(\Delta + 1)$ -coloring and the $(\Delta + 1)$ -list coloring problems, and the difficulty of dealing with vertices having color palettes of significantly different sizes adds an additional challenge. As the result, until very recently the obtained complexity bounds have been significantly weaker than the bounds for the $(\Delta + 1)$ -coloring problem, see, e.g., [5, 20, 28]. This changed last year, when in a recent breakthrough Halldórsson et al. [24] gave a randomized $O(\log^3 \log n)$ -round distributed algorithm for D1LC in the LOCAL distributed model. Observe that this bound matches the state-of-the-art complexity for the (easier) $(\Delta + 1)$ -coloring problem [10]. This work has been later extended to the CONGEST model by Halldórsson et al. [25], who designed a randomized CONGEST algorithm for D1LC that runs in $O(\log^5 \log n)$ rounds. Similarly as for the LOCAL model, this bound matches the state-of-the-art complexity for the $(\Delta + 1)$ -coloring problem in CONGEST [23].

Specifically for the CongestedClique model, the only earlier D1LC result we are aware of is by Bamberger et al. [5], who extended their own CONGEST algorithm for the problem to obtain a deterministic D1LC algorithm requiring $O(\log \Delta \log \log \Delta)$ rounds in CongestedClique. However, the randomized state-of-the-art D1LC bound in the CongestedClique model follows from the aforementioned $O(\log^5 \log n)$ -round CONGEST algorithm by Halldórsson et al. [25], which works directly in CongestedClique. This should be compared with the state-of-the-art $O(1)$ -round CongestedClique algorithms for $(\Delta + 1)$ -coloring [9, 15].

Recent work in D1LC on MPC. Various coloring problems have been also studied in a related model of parallel computation, the so-called *Massively Parallel Computation* (MPC) model. The MPC model, introduced by Karloff et al. [27], is now a standard theoretical model for parallel algorithms. The MPC model with $O(n)$ local space and n machines is essentially equivalent to the CongestedClique model (see, e.g., [7, 26]), and this implies that many MPC algorithms can be easily transferred to the CongestedClique model. (However, this relationship requires that the local space of MPC is $O(n)$ words, not more.)

Both the $(\Delta + 1)$ -coloring and $(\Delta + 1)$ -list coloring problems have been studied in MPC extensively (see, e.g., [5, 15] for linear local space MPC and [5, 9, 14] for sublinear local space MPC). We are aware only of a few works for the D1LC problem on MPC, see [5, 11, 24]. The work most relevant to our paper is the result of Halldórsson et al. [24]. They give a constant-round MPC algorithm assuming the local MPC space is *slightly superlinear*, i.e., $\Omega(n \log^4 n)$ [24, Corollary 2]. This result relies on the palette sparsification approach due to Alon and Assadi [1] (see also [3]) to the D1LC problem, which reduces the problem to a sparse instance of size $O(n \log^4 n)$; hence, on an MPC with $\Omega(n \log^4 n)$ local space one can put the entire graph on a single MPC machine and then solve the problem in a single round. Given the similarity of CongestedClique and the MPC model with *linear local space*, one could hope that the use of “slightly superlinear” MPC local space in [24] can be overcome and the approach can allow the problem to be solved in linear local space, resulting in a

CongestedClique algorithm with a similar performance. Unfortunately, we do not think this is the case. The palette sparsification approach of Alon and Assadi does not reduce the number of vertices in an input graph, and can only hope to reduce the maximum degree of the graph down to, at best, $\Theta(\log n)$. It has no effect on graphs that already have $\Delta = O(\log n)$, and these sparse graphs are still hard instances for D1LC, with no better known upper bound than on general graphs.

Further, we have recently seen a similar situation in $(\Delta + 1)$ -coloring. The palette sparsification by Assadi et al. [3] trivially implies a constant-round MPC algorithm for $(\Delta + 1)$ -coloring with local space $\Omega(n \log^2 n)$, but does not give a constant-round algorithm for $(\Delta + 1)$ -coloring in CongestedClique. Only by using a fundamentally different approach were Chang et al. [9] and then (deterministically) Czumaj et al. [13] able to obtain constant-round $(\Delta + 1)$ -coloring algorithms in CongestedClique. Hence, despite having a constant-round algorithm for D1LC in MPC with local space $\Omega(n \log^4 n)$, possibly a different approach than palette sparsification is needed to achieve a similar performance for D1LC in CongestedClique.

Derandomization tools for distributed coloring algorithms. In our paper we rely on a recently developed general scheme for derandomization in the CongestedClique model (and used also extensively in the MPC model) by combining the methods of bounded independence with efficient computation of conditional expectations. This method was first applied by Censor-Hillel et al. [8], and has since been used in several other works for graph coloring problems, (see, e.g., [15, 36]), and for other problems in CongestedClique and MPC.

The underlying idea begins with the design of a randomized algorithm using random choices with only *limited independence*, e.g., $O(1)$ -wise-independence. Then, each round of the randomized algorithm can be simulated by giving all nodes a shared random seed of $O(\log n)$ bits. Next, the nodes deterministically compute a seed which is at least as good as a random seed is in expectation. This is done by using an appropriate estimation of the local quality of a seed, which can be aggregated into a global measure of the quality of the seed. Combining this with the techniques of conditional expectation, pessimistic estimators, and bounded independence, this allows selection of the bits of the seed “batch-by-batch,” where each batch consists of $O(\log n)$ bits. Once all bits of the seed are computed, we can use it to simulate the random choices of that round, as it would have been performed by a randomized algorithm. A more detailed explanation of this approach is given in Section 2.2.

1.2 Technical Overview

The core part of our constant-round deterministic CongestedClique D1LC algorithm (given as BUCKETCOLOR, Algorithm 4) does *not* follow the route of recent D1LC algorithms for LOCAL and CONGEST due to Halldórsson et al. [24, 25]. Instead, it uses fundamentally different techniques, extending the approach developed recently in a simple deterministic $O(1)$ -round CongestedClique algorithm for $(\Delta + 1)$ -list coloring of Czumaj, Davies, and Parter [15]. Their $(\Delta + 1)$ -list coloring algorithm works by partitioning nodes into Δ^ε buckets, for a small constant ε . (This partitioning is initially at random, but then it is derandomized using the method of conditional expectations). The available colors are also distributed among all buckets, except for one ‘leftover’ bucket, which is left without colors and is set aside to be colored later. Then, each node’s palette is restricted to only the colors assigned to its bucket (except those in the leftover bucket, whose palettes are not restricted). This ensures that nodes in different buckets have entirely disjoint palettes, and therefore edges between different buckets can be removed from the graph, since they would never cause a coloring conflict. One important property is that nodes still have sufficient colors when their

palettes are restricted in this way. This is achieved in [15] using two main arguments: firstly, the fact that colors are distributed among one fewer buckets than nodes provides enough ‘slack’ to ensure that with reasonably high probability, a node would receive more colors than neighbors in its bucket. Secondly, the few nodes that do *not* satisfy this property induce a small graph (of size $O(n)$), and therefore can be collected onto a single network node and colored separately.

Using the approach from [15] sketched above, a $(\Delta + 1)$ -list coloring instance can be reduced into multiple smaller $(\Delta + 1)$ -list coloring instances (i.e., on fewer nodes and with a new, lower maximum degree) that are *independent* (since they had disjoint palettes), and so can be solved in parallel without risking coloring conflicts. The final part of the analysis of [15] was to show that, after recursively performing this bucketing process $O(1)$ times, these instances are of $O(n)$ size and therefore they could be collected to individual nodes and solved in a constant number of rounds in `CongestedClique`.

There are major barriers to extending this approach to (degree+1)-list coloring. Crucially, it required the number of buckets to be dependent on Δ , and all nodes’ palette sizes to be at least Δ . Dividing nodes among too few buckets would cause the induced graphs to be too large, and the algorithm would not terminate in $O(1)$ rounds; using too few buckets would fail to provide nodes with sufficient colors in their bucket. In the DILC problem, we no longer have a uniform bound on palette size, so it is unclear how to perform this bucketing.

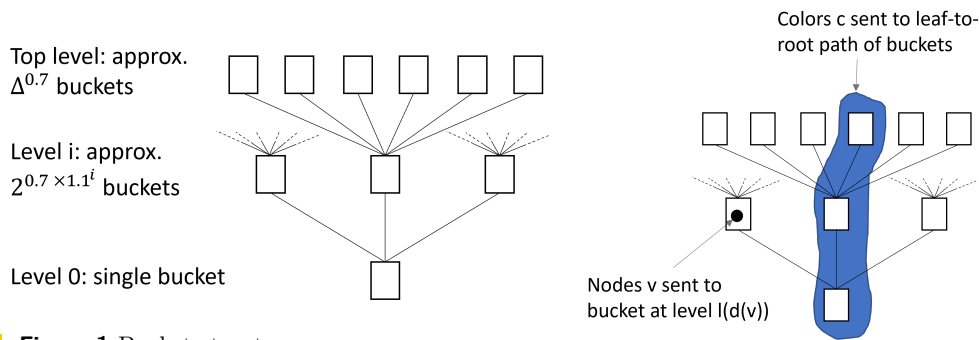
Our first major conceptual change is that, rather than simply partitioning among a number (dependent on Δ) of equivalent buckets, we instead use a tree-structured hierarchy of buckets, with the $O(\log \log \Delta)$ levels in the hierarchy corresponding to doubly-exponentially increasing degree ranges (see Figure 1). Nodes with degree $d(v)$ will be mapped to a bucket in a level containing (very approximately) $d(v)^{0.7}$ buckets. Colors will be mapped to a top-level (leaf) bucket, but will also be assigned to every bucket on the leaf-root path in the bucket tree (see Figure 2). We can therefore discard all edges between different buckets that do not have an ancestor-descendant relationship, since these buckets will have disjoint palettes.

This change allows nodes to be bucketed correctly according to their own degree. However, it introduces several new difficulties:

- We no longer get a good bound on the number of lower-degree neighbors of a node v that may share colors with it. We can only hope to prove that v receives enough colors relative to its higher (or same)-degree neighbors.
- The technique of having a leftover bucket which is not assigned colors no longer works to provide slack (nor even makes sense - we would need a leftover bucket at every level, but, for example, level 0 only contains one bucket).

In order to cope with these challenges, firstly, we employ the observation that if we were to greedily color in non-increasing order of degree, we would require nodes to have a palette size of $d^+(v) + 1$ (where $d^+(v)$ is the number of v ’s neighbors of equal or higher degree), rather than $d(v) + 1$ (since $d^+(v)$ of v ’s neighbors will have been colored at the point v is considered). Therefore, we argue that we can still show that the graph is colorable even though our bucketing procedure may leave nodes with many more lower-degree neighbors than palette colors. (It is not necessarily clear how to find such a coloring in a parallel fashion, but in our analysis, we will be able to address this issue.)

This observation also helps us with the problem of generating slack without a leftover bucket. We show that, since lower-degree neighbors are now effectively providing slack, only nodes with very few lower-degree neighbors may not receive enough colors (relative to higher-degree neighbors) in their bucket. It transpires that we can easily generate slack



■ **Figure 1** Bucket structure

■ **Figure 2** Partitioning nodes and colors

for these nodes prior to BUCKETCOLOR via derandomizations of fairly standard procedures (COLORTRIAL, Algorithm 2, and SUBSAMPLE, Algorithm 3). The randomized bases for all these procedures would inevitably result in some nodes failing to meet the necessary properties for the next stage. To overcome this, we derandomize all of these procedures, using the method of conditional expectations. As well as making the algorithm deterministic, this has the important property of ensuring that *failed* nodes form an $O(n)$ -size induced graph, which can be easily dealt with later.

Having solved the problem of slack for the bucketing process (by showing that nodes have received palettes of size at least $d^+(v) + 1$ within their buckets), it remains to find a parallel analog to greedily coloring in non-increasing order of degree. Our approach here is to repeatedly move all nodes from their current bucket to a child of that bucket in the bucket tree (which further restricts their neighborhood and available palette). We show that, by correct choice of bucket and order of node consideration, we will always be able to find child buckets such that each node still has palette size at least $d^+(v) + 1$ according to the new bucket assignment. We also show that, after $O(1)$ iterations of this process, nodes only have one palette color in their bucket, and zero neighbors earlier in the coloring order. Then, all nodes can safely color themselves this palette color, and the coloring is complete.

The overall structure of the main algorithm COLOR (Algorithm 1) is more complicated, since SUBSAMPLE produces a graph G' of leftover nodes that are deferred to be colored later. We recursively run Algorithm 1 on this graph G' , and show that it is sufficiently smaller than the original input graph that after $O(1)$ recursive calls, the remaining graph has size $O(n)$ and can be collected and solved on a single node.

If we combine all these tools together then we will be able to obtain a randomized CongestedClique algorithm that finds a $(\text{degree}+1)$ -list coloring of any graph in $O(1)$ rounds. Using the *method of conditional expectation* using *bounded-independence hash function* (see Section 2.2 and Appendix A), each randomized step of our algorithm can be derandomized.

2 Preliminaries

The main model considered in this paper is CongestedClique, as introduced by Lotker et al. [31]. It is a variant of CONGEST, in which nodes can send a message of size $O(\log n)$ to each neighboring node in the graph in each communication round: the difference is that CongestedClique allows all-to-all communication, and hence the underlying communication network is a complete graph on the nodes V . In particular, this allows the communication to be performed between all pairs of nodes rather than being restricted to the edges of the

input graph. CongestedClique has been introduced as a theoretical model to study overlay networks: an abstraction that separates the problems emerging from the topology of the communication network from the problems emerging from the structure of the problem at hand. It allows us to study a model in which each pair of nodes can communicate, and we do not consider any details of how this communication is executed by the underlying network.

The **degree+1 list coloring (D1LC) problem** is for a given graph $G = (V, E)$ on n nodes and given color palettes $\Psi(u)$ assigned to each node $u \in V$, such that $|\Psi(u)| \geq d(u) + 1$, the objective to find a proper coloring of nodes in G such that each node is assigned to a color from its color palette (and, as in proper coloring, no edge in G is monochromatic). The input to the D1LC problem in CongestedClique is a graph G , where each node v of G has assigned a network node and this network node knows $\Psi(v)$ and all neighbors of v in G .

A useful property of the CongestedClique model is that thanks to the constant-round routing algorithm of Lenzen [29], information can be redistributed essentially arbitrarily in the communication network, so there is no need to associate the computational entities with nodes in the input graph G . (This is in stark contrast to the related LOCAL and CONGEST distributed models in which the link between computation and input graph locality is integral.) In particular, this allows us to collect graphs of size $O(n)$ on a single node in $O(1)$ rounds. Because of this “decoupling” of the computation from the input graph, where appropriate we will distinguish the nodes in their roles as computational entities (“network nodes”) from the nodes in the input graph (“graph nodes”).

2.1 Notation

For $k \in \mathbb{N}$ we let $[k] := \{1, \dots, k\}$. We consider a graph $G = (V(G), E(G))$ with $V(G)$ as the node set and $E(G)$ as the edge set. The size of a graph G refers to the number of edges in G and is denoted by $|G|$. The set of neighbors of a node v is denoted by $N_G(v)$ and the degree of a node v is denoted by $d_G(v)$. The maximum degree of any node in G is denoted by Δ_G . For any node v , we partition its neighbors into two sets $N_G^+(v) := \{u \in N_G(v) : d_G(u) \geq d_G(v)\}$ and $N_G^-(v) := \{u \in N_G(v) : d_G(u) < d_G(v)\}$, and let $d_G^+(v) := |N_G^+(v)|$ and $d_G^-(v) := |N_G^-(v)|$. When G is clear from the context, we suppress G from the subscripts of the notation.

For the coloring problem, for a node v , $\Psi_G(v)$ denotes the list of colors in the color palette of v and $p_G(v) := |\Psi_G(v)|$. As we proceed in coloring the nodes of the input graph G the graph will be changing and the color palettes of the nodes may also change. We will ensure that at any moment, if G denotes the current graph then we have $p_G(v) \geq d_G(v) + 1$. We use \mathcal{C} to denote the set of all colors present in the palette of any node (in a given moment).

For binary strings a and a' in $\{0, 1\}^*$, $a \sqsubseteq a'$ denotes that a is a prefix of a' , and $a \sqsubset a'$ denotes that it is a *strict* prefix of a' . Furthermore, $a' \supseteq a$ iff $a \sqsubseteq a'$, and $a' \supset a$ iff $a \sqsubset a'$.

Due to the space constraint, the missing proofs are deferred to the full version.

2.2 Derandomization in CongestedClique

The *method of conditional expectations* using *bounded-independence hash functions* is a nowadays classical technique for the derandomization of algorithms [17, 33, 34, 39]. Starting with the recent work of Censor-Hillel et al. [8], this approach has been found very powerful also in the setting of distributed and parallel algorithms, see e.g., [5, 12, 13, 14, 15, 16, 18, 21, 36].

This technique requires that we show that our randomized algorithm can be made to work in expectation using only bounded-independence. It is known that small families of bounded-independence hash functions exist, and that hash functions in these families can be specified by a short seed. It is also known that such a family must contain a hash function

which beats the expectation due to the probabilistic method. Using these facts, we can perform an efficient search for a hash function which beats the expectation by iteratively setting a larger and larger prefix of the seed of the hash function.

We give a more detailed explanation of bounded-independence hash functions and the method of conditional expectations with its implementation in Appendix A.

3 The D1LC Algorithm

The framework of our `CongestedClique` algorithm is $\text{COLOR}(G, x)$ (Algorithm 1), which colors graph G relying on three main procedures: `COLORTRIAL`, `SUBSAMPLE`, and `BUCKETCOLOR`.

`COLORTRIAL` is a derandomized version of a simple and frequently used coloring procedure: all nodes nominate themselves with some constant probability, and nominated nodes then pick a color from their palette. If no neighbors choose this same color, the node is successful and takes this color permanently. For our algorithm, the goal of `COLORTRIAL` is to provide permanent slack for nodes whose neighbors mostly have higher degree than their own.

`SUBSAMPLE` is a derandomized version of sampling: nodes v defer themselves to S (to be colored later) with probability $d(v)^{-0.1}$. The purpose of this is to provide temporary slack to nodes whose neighbors mostly have similar degrees to their own. We will then recursively run the whole algorithm on S , and we will show that after $O(1)$ recursive calls the remaining graph will be of size $O(n)$, which can be trivially colored in `CongestedClique` in $O(1)$ rounds.

`BUCKETCOLOR` is our main coloring procedure, and is designed to color all nodes for which `COLORTRIAL` and `SUBSAMPLE` have generated sufficient slack, as well as all nodes whose neighbors mostly have lower degree than their own.

All these three algorithms begin with a randomized procedure, and use the method of conditional expectations on a family of $O(1)$ -wise independent hash functions to derandomize it. Note that *this derandomization is an essential part of the algorithm* even if one is only concerned with probabilistic success guarantees. This is because in low-degree graphs, we cannot obtain the necessary properties with high probability, and some nodes will fail. The method of conditional expectations ensures that these graphs of failed nodes are of $O(n)$ size (and hence can be collected onto a single network node in $O(1)$ rounds to color sequentially).

Using `COLORTRIAL`, `SUBSAMPLE`, and `BUCKETCOLOR`, we can present our main algorithm $\text{COLOR}(G, x)$ (Algorithm 1) to color graph G . The algorithm assumes that $\Delta_G \leq O(\sqrt{n})$ and it uses a parameter x , $0 \leq x \leq 0.9$, that quantifies the size of the remaining graph over recursive calls (the algorithm starts with $x = 0$ and recursively increases by 0.1 until $x = 1$). In Section 6 (Lemma 19), we extend the analysis to arbitrary graphs, allowing arbitrary Δ_G .

■ **Algorithm 1** $\text{COLOR}(G, x)$: $\Delta_G \leq O(\sqrt{n})$; $0 \leq x \leq 0.9$; C is a sufficiently large constant

-
- 1 If $|G| = O(n)$, then collect G in a single network node and solve the problem locally.
 - 2 Set $L_0 := \{v \in G : p_G(v) < C\}$ and $G_0 := G \setminus L_0$.
 - 3 $G_1, F \leftarrow \text{COLORTRIAL}(G_0)$.
 - 4 $G_2, G' \leftarrow \text{SUBSAMPLE}(G_1, x)$.
 - 5 $\text{BUCKETCOLOR}(G_2)$.
 - 6 $\text{COLOR}(G', x + 0.1)$.
 - 7 Collect and solve L_0 and then F at a single node.
-

Step 1 in $\text{COLOR}(G, x)$ uses the fact that if G is of size $O(n)$, then in `CongestedClique`, the entire graph can be collected onto a single network node in $O(1)$ rounds and the coloring can be done locally. In the same way, since L_0 consists of vertices of constant degree, we can color them in step 7 in $O(1)$ rounds. Similarly, we will argue that the graph F (of failed

99:10 Optimal (degree+1)-Coloring in Congested Clique

nodes in `COLORTRIAL`) is of size $O(n)$, and hence it can be colored in step 7 in $O(1)$ rounds. The central part of our analysis will be to show that after a constant number of recursive calls the algorithm terminates with a correct solution to `D1LC` of G .

To prove the correctness of our algorithm, we show the following properties of `COLOR`(G, x):

1. `COLORTRIAL`, `SUBSAMPLE`, and `BUCKETCOLOR` run deterministically in $O(1)$ rounds.
2. The size of F is $O(n)$.
3. Each node in G_2 has sufficient slack to be colored by `BUCKETCOLOR`. For each node v of G_2 , either $p_{G_2}(v) \geq d_{G_2}(v) + \frac{1}{4}d_{G_2}(v)^{0.9}$, or $|N_{G_2}^-(v)| \geq \frac{1}{3}d_{G_2}(v)$.
4. The size of the (remaining) graph reduces over recursive calls in the following sense:

$$\sum_{v \in G'} d_{G_1}(v)^{x+0.1} \leq Cn + 2 \sum_{v \in G_1} d_{G_1}(v)^x . \quad (1)$$

Observe that when $x = 0.9$, expression (1) bounds the number of edges of G' . In particular, we show that the total size of the remaining graph is $O(n)$ after 10 recursive calls.

In Section 4, we describe the procedures `COLORTRIAL` and `SUBSAMPLE`. Also, we prove the desired properties of F , G_2 , and G' in Section 4. In Section 5, we describe the procedure `BUCKETCOLOR`. Finally, we prove our main theorem (Theorem 1) in Section 6.

For simplicity of the presentation, in the pseudocode of our algorithms in the following sections, we will only present the randomized bases of each procedure. In each case, the full deterministic procedure comes from applying the method of conditional expectations to the randomized bases, with some specific cost function we will make clear in the analysis.

4 ColorTrial and Subsample

We describe procedure `COLORTRIAL` and `SUBSAMPLE` in Section 4.1 and Section 4.2, respectively, along with some of their crucial useful properties. In particular, we show that $|F| = O(n)$ in Lemma 9 of Section 4.1 and show that graph G' has the desired property in Lemma 11 of Section 4.2. Finally, we give a lemma capturing the desired property of graph G_2 (which is the input to `BUCKETCOLOR`).

4.1 ColorTrial

We first note that, because nodes with palette size less than C are removed immediately prior to `COLORTRIAL`(G_0) in `COLOR`(G, x), we may assume that all nodes v in G_0 have $p_{G_0}(v) \geq C$. The randomized procedure on which `COLORTRIAL` is based is Algorithm 2. `COLORTRIAL` has two major steps: nomination step (line 1) and coloring step (line 3). The coloring of a node can be deferred if it is a *failed* node either in nomination step and coloring step. Note that the notions of failed nodes are different in nomination step and coloring step, and we will define both the notions in the following part of this section.

We define some notions that will be useful to define failed nodes in both the nomination step and the coloring step of `COLORTRIAL`.

► **Definition 2.** $N^*(v) \subseteq N_{G_0}(v)$ is defined as the subset of neighbors u of v that have $d_{G_0}(u) \geq 3d_{G_0}(v)$. $\text{Nom}_v \subseteq N_{G_0}(v)$ is defined as the subset of v 's neighbors that self-nominate, and $\text{Nom}_v^* := \text{Nom}_v \cap N^*(v)$.

Next, we define the notion of failed nodes in the nomination step.

Algorithm 2 COLORTRIAL(G_0) - Randomized Basis

- 1 Each node v in G_0 independently self-nominates with probability $\frac{1}{4}$.
- 2 Each node v decides if it is successful or failed in the nomination step.
- 3 For each self-nominated node v (that is successful in the nomination step):
 - v chooses a random palette color $c(v) \in \Phi_{G_0}(v)$;
 - v colors itself with color $c(v)$ if no neighbor u of v choose $c(u) = c(v)$;
 - v decides if it is successful or failed in the coloring step.

Return

- G_1 , the induced graph of remaining (non-failed) uncolored nodes, with updated palettes,
 - F , the induced graph of failed nodes (either in nomination step or in the coloring step), with updated palettes.
-

► **Definition 3.** A node v is successful during the nomination step of COLORTRIAL if both of the following hold (if either condition does not hold, node v fails):

- $|\text{Nom}_v| \leq \frac{1}{4}d_{G_0}(v) + p_{G_0}(v)^{0.7}$;
- $|\text{Nom}_v^*| \geq \frac{1}{4}|N^*(v)| - p_{G_0}(v)^{0.7}$.

To derandomize COLORTRIAL, we replace each of the random choices of lines 1 and 3 (the nomination step and the coloring step respectively) with choices determined by a random hash function from a $O(1)$ -wise independent family $[n^{O(1)}] \rightarrow [n^{O(1)}]$. We show that, under such a choice of hash function, the subgraph induced by the failed nodes in the nomination step is of size $O(n)$ in expectation (Lemma 4). We are then able to derandomize this selection using the method of conditional expectations to obtain Lemma 5.

► **Lemma 4.** When nomination choices of COLORTRIAL are determined by a random hash function from a $O(1)$ -wise independent hash family $[n^{O(1)}] \rightarrow [n^{O(1)}]$, any node v fails in the nomination step of COLORTRIAL with probability at most $1/p_{G_0}(v)$.

► **Lemma 5.** We can deterministically choose a hash function in $O(1)$ rounds, from a $O(1)$ -wise independent family $[n^{O(1)}] \rightarrow [n^{O(1)}]$, to run the nomination step of COLORTRIAL such that the size of the subgraph induced by the failed nodes (in the nomination step) is $O(n)$.

Besides the nomination step, a node can also fail in the coloring step of COLORTRIAL. Now we formally define what it means for a node to fail in the coloring step.

► **Definition 6.** A node v is successful during the coloring step of COLORTRIAL if any of the following hold (if none hold, node v fails):

- $p_{G_0}(v) \geq 1.1d_{G_0}(v)$;
- $|N^*(v)| < \frac{1}{3}d_{G_0}(v)$;
- at least $0.03d_{G_0}(v)$ of v 's neighbors failed in the nomination step;
- at least $0.01p_{G_0}(v)$ of v 's neighbors successfully color themselves a color not in v 's palette.

Notice that the first three properties are already determined by the nomination step. Here, we need to handle the fourth property. Similar to our analysis for the nomination step, we are able to show that choosing a hash function uniformly at random from a $O(1)$ -wise independent family to make decisions in the coloring step yields a subgraph of failed nodes of size $O(n)$ in expectation (Lemma 7). We can then derandomize this result using the method of conditional expectations, achieving Lemma 8.

99:12 Optimal (degree+1)-Coloring in Congested Clique

► **Lemma 7.** *When color choices in the coloring step of COLORTRIAL are determined by a random hash function from a $O(1)$ -wise independent hash family $[n^{O(1)}] \rightarrow [n^{O(1)}]$, any node v that did not fail in the nomination step fails in the coloring step of COLORTRIAL with probability at most $1/p_{G_0}(v)$.*

► **Lemma 8.** *We can deterministically choose a hash function in $O(1)$ rounds, from a $O(1)$ -wise independent family $[n^{O(1)}] \rightarrow [n^{O(1)}]$, to run the coloring step of COLORTRIAL such that the size of the subgraph induced by the failed nodes (in the coloring step) is at most n .*

Note that Lemma 8 is the only lemma whose prove requires the assumption $\Delta_G = O(\sqrt{n})$.

Recall that F denotes the subgraph of G induced by the nodes that is either failed in the nomination step or in the coloring step of COLORTRIAL. The following lemma bounds $|F|$, and follows immediately from Lemma 5 and Lemma 8.

► **Lemma 9.** *We can deterministically choose hash functions in $O(1)$ rounds, from a $O(1)$ -wise independent hash family $[n^{O(1)}] \rightarrow [n^{O(1)}]$, to run each step of COLORTRIAL such that the size of the subgraph induced by the failed nodes is at most $O(n)$, i.e., $|F| = O(n)$.*

4.2 Subsample

After executing COLORTRIAL(G_0), COLOR(G, x) executes procedure SUBSAMPLE(G_1, x). The randomized procedure on which SUBSAMPLE is based is Algorithm 3. To derandomize SUBSAMPLE, we replace the random choice of line 1 (to generate a set S of vertices) with a choice determined by a hash function from a $O(1)$ -wise independent family $[n^{O(1)}] \rightarrow [n^{O(1)}]$.

■ **Algorithm 3** SUBSAMPLE(G_1, x) - Randomized Basis

-
- 1 Each node v in G_1 independently joins S with probability $d_{G_1}(v)^{-0.1}$
 - 2 Each node v decides whether it succeeds or fails. Let F_1 be the set of failed nodes.
 - 3 Let L denote the nodes with $p_{G_1}(v) < C$. Return:
 - G_2 , consisting of $G_1 \setminus (F_1 \cup S \cup L)$
 - $G' = (S \cup F_1 \cup L)$
-

Note that while x is not used explicitly in Algorithm 3, it increases by 0.1 in each recursive call to COLOR, and this plays a significant role in the analysis (see Lemma 11). Our aim is to show that after 10 levels of recursion of COLOR($G, 0$), the remaining graph is of size $O(n)$.

We start by defining the notion of failed nodes in SUBSAMPLE:

► **Definition 10.** *Let us define $N^{\approx}(v) \subseteq N_{G_1}(v)$ to be the subset of v 's neighbors u with $\frac{1}{2}d_{G_1}(v) \leq d_{G_1}(u) \leq 6d_{G_1}(v)$. A node v is classed as successful during SUBSAMPLE if either*

- $p_{G_1}(v) \geq 1.1d_{G_1}(v)$; or
- $|N^{\approx}(v)| \leq \frac{1}{3}d_{G_1}(v)$; or
- at least $\frac{1}{4}p_{G_1}(v)^{0.9}$ of v 's neighbors join S .

v is classed as failed if none of the above three conditions hold.

In a similar way to the analysis in Section 4.1, we can express the analysis of SUBSAMPLE in terms of bounded-independence hash functions and derandomize it, obtaining the following:

► **Lemma 11.** *Let x be such that $0 \leq x \leq 0.9$. We can deterministically choose a hash function in $O(1)$ rounds, from a $O(1)$ -wise independent hash family, to execute line 1 of SUBSAMPLE to generate set S such that the following holds:*

$$\sum_{v \in G'} d_{G_1}(v)^{x+0.1} \leq Cn + 2 \sum_{v \in G_1} d_{G_1}(v)^x.$$

We end with a lemma which explains what properties the graph G_2 has. Recall that G_2 is the graph of successful nodes that results from running `COLORTRIAL` and `SUBSAMPLE` on our input graph, and it is the input graph to our main coloring procedure `BUCKETCOLOR` in Section 5. Here, we show that each node in G_2 has sufficient slack to be colored in $O(1)$ rounds by `BUCKETCOLOR`.

► **Lemma 12.** *For any $v \in G_2$, either $p_{G_2}(v) \geq d_{G_2}(v) + \frac{1}{4}d_{G_2}(v)^{0.9}$, or $|N_{G_2}^-(v)| \geq \frac{1}{3}d_{G_2}(v)$.*

5 BucketColor

In this section, we describe our core coloring procedure `BUCKETCOLOR`(G_2). Note that, each node in the input graph G_2 to `BUCKETCOLOR` has sufficient slack as mentioned in Lemma 12. Throughout this section, the graph under consideration is always G_2 , so we omit the subscript G_2 from $N_{G_2}(v)$, $d(v)$, $N_{G_2}^+(v)$, $d_{G_2}^+(v)$, $N_{G_2}^-(v)$, $d_{G_2}^-(v)$, $\Psi(v)$, $p(v)$ and Δ_G .

In Section 5.1, we first formalize the bucket structure of nodes (as discussed in Section 1.2), and then introduce some useful definitions. Then we describe algorithm `BUCKETCOLOR` in Section 5.1. In Sections 5.2, we analyze the correctness of `BUCKETCOLOR`.

5.1 Assigning nodes to buckets

We use two special functions in the description of our algorithm in this section: $l : V(G_2) \rightarrow \mathbb{N}_{\geq 0}$ and $b : \mathbb{N}_{\geq 0} \rightarrow \mathbb{N}_{\geq 0}$. l is defined as $l(v) := \max\{\lfloor \log_{1.1} \log_2 d(v) \rfloor, 0\}$ for node v , and b is defined as $b(i) := \lfloor 0.7 \cdot 1.1^i \rfloor$ for $i \in \mathbb{N}_{\geq 0}$. If $d(v)$ is at least a suitable constant, then $b(l(v)) = \Theta(\log d(v))$ and $b(l(v)) \leq 0.7 \log_2 d(v)$.

We consider a partition of the nodes of G_2 into $O(\log \log \Delta)$ levels, with the *level* of a node v equal to $l(v) = \max\{\lfloor \log_{1.1} \log_2 d(v) \rfloor, 0\}$. The nodes of a particular level will be further partitioned into buckets. The *level of a bucket x* is the level of a possible node that can be put into this bucket, and is denoted by $level(x)$. The buckets of *level i* (or *level- i buckets*) are identified by binary strings of length $b(i)$, where $i \in \mathbb{N}_{\geq 0}$, as well as their level.¹ So, there are $2^{b(i)}$ level- i buckets. To put a node v to a bucket, (in our algorithm) we generate a random binary string of length $b(l(v))$.

The set of buckets forms a hierarchical tree structure as described below. We say that a bucket a' is a *child* of a (and a is the *parent* of a') if $level(a') = level(a) + 1$ and $a \sqsubseteq a'$. We say that a' is a *descendant* of a (and a is an *ancestor* of a') if $level(a') \geq level(a)$ and $a \sqsubseteq a'$ (note that by definition a is a descendant and ancestor of itself). The buckets form a rooted tree structure: the root is the single level 0 bucket, specified by the empty string; each bucket in level $i > 0$ has one parent in level $i - 1$ and multiple children in level $i + 1$.

We also put colors into the buckets. For any color c , we put c into a bucket of level $\lfloor \log_{1.1} \log_2 \Delta \rfloor$ by generating a random binary string of length $b(\lfloor \log_{1.1} \log_2 \Delta \rfloor + 20)$. Consider a bucket a and a color c which is put in a . We say c is assigned to bucket a' (and that bucket a' contains c) iff $a' \sqsubseteq a$. Note that a' is a leaf, since the string generated for c is of maximum length; note also that c is assigned to all buckets on the path from a to the root bucket.

Our algorithm uses a hash function to generate the binary strings (and hence the buckets) for the colors and nodes. Based on the partition of the nodes and colors into buckets, it is

¹ Note that, at low levels, buckets in different levels can be identified by the same string, because the function $b(i) = \lfloor 0.7 \cdot 1.1^i \rfloor$ is not injective for $i \leq 24$. Therefore, for example, $b(0) = b(1) = 0$, and so levels 0 and 1 both in fact contain a single bucket specified by the empty string. We treat these as different buckets in order to conform to a standard rooted tree structure, and therefore must identify buckets by their level as well as their specifying string.

99:14 Optimal (degree+1)-Coloring in Congested Clique

sufficient to color a set of reduced instances (one per bucket) of the original D1LC instance. The following definition formalizes the effective palettes and neighborhoods of a node under any function mapping nodes and colors to strings.

► **Definition 13.** Let $h : (\mathcal{C} \cup V(G_2)) \rightarrow \{0, 1\}^*$ be a function mapping colors and nodes to binary strings. For each node $v \in G_2$, define:

- the graph $G_{h(v)}^+$ to contain all edges $\{u, w\} \in G_2$ with $d(u) \leq d(w)$ for which $h(u) = h(v)$ and $h(w) \supseteq h(v)$, and all nodes which are endpoints of such edges;
- $\Psi_{h(v)}(v) = \{c \in \Psi(v) : h(c) \supseteq h(v)\}$ $\{\Psi_{h(v)}(v)$ is the set of palette colors v has in $h(v)\}$;
- $N_{h(v)}^+(v) := \{w \in N^+(v) : h(w) \supseteq h(v)\}$ $\{d_{h(v)}^+(v)$ is the number of neighbors u that v has in descendants of $h(v)$ with $d(v) \leq d(u)\}$;
- $p_{h(v)}(v) = |\Psi_{h(v)}(v)|$ and $d_{h(v)}^+(v) = |N_{h(v)}^+(v)|$.

Observe that there is a reduced instance for each bucket. Notice that each node u is present in only one reduced instance, i.e., in $G_{h(u)}^+$ (the reduced instance corresponding to the bucket where u is present); and each edge $\{u, w\}$ with $d_{G_2}(u) \leq d_{G_2}(w)$ is present in at most one reduced instance, i.e., possibly in $G_{h(u)}^+$ only when $h(u) \sqsubseteq h(w)$ (i.e., w is present in some descendent bucket of u). Consider a node u in the reduced instance G_x^+ (i.e., $h(u) = x$). $N_x^+(u)$ and $d_x^+(u)$ denote the set of neighbors and the degree of u in G_x^+ , respectively. Moreover, for coloring the reduced instance G_x^+ , let $\Psi_x(u)$ and $p_x(u)$ denote the color palette and the size of the the color palette of u , respectively.

Observe that the reduced G_x^+ instances are not independent, and they can be of size $\omega(n)$. Also, it may be the case that G_x^+ may not be a valid D1LC instance. To handle the issue, we define the notion of *bad nodes* in Definition 14. Intuitively, bad nodes are those who do not behave as expected when mapped to their bucket (e.g. have too many neighbors or too few colors therein), and we will show that the subgraphs of buckets restricted to good nodes are of size $O(n)$ and can be colored in $O(1)$ rounds. If we choose our hash function uniformly at random from a $O(1)$ -wise independent family of hash functions, the subgraph G_{bad} (induced by the bad nodes) has size $O(n)$ in expectation. We also show that it is possible to choose a of hash function deterministically in $O(1)$ rounds such that the size of G_{bad} is $O(n)$.

► **Definition 14.** Given a hash function $h : (\mathcal{C} \cup V(G_2)) \rightarrow \{0, 1\}^*$ mapping colors and nodes to binary strings, define a node v to be bad if any of the following occur:

1. $d_{h(v)}^+(v) \geq d^+(v)2^{-b(l(v))} + \frac{1}{8}d(v)^{0.9}2^{-b(l(v))}$;
2. $p_{h(v)}(v) \leq p(v)2^{-b(l(v))} - \frac{1}{8}d(v)^{0.9}2^{-b(l(v))}$;
3. any of v 's level $l(v) + 20$ descendant buckets contain more than one of v 's palette colors;
4. more than $2n2^{-b(l(v))}$ nodes v' have $h(v) = h(v')$.

(1) and (2) ensure that each reduced instance (after removing the bad nodes) are valid D1LC instances; (3) ensure that the dependencies among the reduced instances are limited; and (4) when combined with (1) ensures that the subgraph induced by bad nodes is $O(n)$.

Now we are ready to discuss our algorithm BUCKETCOLOR. The randomized procedure on which BUCKETCOLOR is based is Algorithm 4. Note that only line 1 of Algorithm 4 is a randomized step, and it can be derandomized by replacing its random choices with choices determined by a hash function from a $O(1)$ -wise independent family $[n^{O(1)}] \rightarrow [n^{O(1)}]$. The subgraph induced by the bad nodes, G_{bad} , is deferred to be colored later. Then in Lines 3 to 11, BUCKETCOLOR colors the (good) nodes in $G_2 \setminus G_{bad}$ in $O(1)$ rounds deterministically.

Algorithm 4 BUCKETCOLOR(G_2) - Randomized Basis

- 1 Each node v uniformly randomly chooses a $b(l(v))$ -bit binary string $h(v)$, and each color is uniformly randomly assigned a $b(\lceil \log_{1.1} \log_2 \Delta \rceil + 20)$ -bit binary string $h(c)$.
 - 2 Each node v decides whether it is bad or good. Let G_{bad} be the subgraph induced by the bad nodes.
 - 3 **for** $O(1)$ iterations **do**
 - 4 Each node $v \in G_2 \setminus G_{bad}$ restricts its palettes to colors c with $h(v) \sqsubseteq h(c)$, i.e.,
 $\Psi_{h(v)}(v) = \{c \in \Psi(v) : h(c) \sqsupseteq h(v)\}$ is the current palette of v .
 - 5 **for each** $i \in [\lceil \log_{1.1} \log_2 \Delta \rceil + 20]$ and each string $x \in \{0, 1\}^{b(i)}$ **do**
 - 6 collect the graph G_x^+ to a dedicated network node $node_x$.
 - 7 **end**
 - 8 **for each node** $v \in G_2 \setminus G_{bad}$ in a bucket $h(v)$, in non-increasing order of degree, performed on $node_{h(v)}$ **do**
 - 9 $h(v) \leftarrow h^*$, where $h^* \sqsupseteq h(v)$ is a child bucket of $h(v)$ with $d_{h^*}^+(v) < p_{h^*}(v)$.
 - 10 **end**
 - 11 **end**
 - 12 Color each node $v \in G_2 \setminus G_{bad}$ with the only palette color in its current bucket.
 - 13 Update the palettes of G_{bad} , collect to a single node, and color sequentially.
-

Overview of coloring good nodes: To color the (good) nodes in $G_2 \setminus G_{bad}$, we proceed in non-increasing order of node degree and start with the hash function h chosen in Line 1 of BUCKETCOLOR. Recall that $h(v)$ denotes the bucket in which node v is present. The algorithm goes over iterations and the bucket status of the nodes change over iterations.

In every iteration, for every node v , we restrict the color palettes of v to the colors present in the descendant bucket of (current) $h(v)$. Also, for every binary string x such that the bucket x has at least one node, we gather the graph having set of edges with one endpoint in bucket x and the other endpoint in some descendent bucket of x , i.e., G_x^+ (of size $O(n)$) at a network node in $O(1)$ rounds. Though all G_x^+ 's are a valid D1LC instance in any iteration, G_x^+ 's are not necessarily independent: there can be an edge between a node v in G_x^+ and a node w outside G_x^+ such that the current palettes of v intersects with the current palette of w . We can show that every node v satisfies $d_{h(v)}^+(v) < p_{h(v)}(v)$ in the first iteration — i.e., each node has enough colors in its bucket to be greedily colored in the non-increasing order of degree. That is, (in first iteration) each graph G_x^+ is a valid D1LC instance.

In each iteration, we move each node down to a child bucket h^* of its current bucket $h(v)$, in such a way that we maintain this colorability property (having more colors in the palette than the degree). This will imply that, when we find graphs G_x^+ in the next iteration, those are also valid D1LC instances. We will show that after $O(1)$ iterations, each node has only 1 palette color in its bucket (and therefore zero higher-degree neighbors in descendant buckets, since $d_{h^*}^+(v) < p_{h^*}(v)$). At this point, nodes can safely color themselves the single palette color in their bucket. To decide on child buckets for the nodes in any iteration, it is essential that each G_x^+ will always fit onto a single network node (which is in fact the case).

5.2 Correctness of BucketColor

To prove the correctness of BUCKETCOLOR formally, we give Lemma 15 and Lemma 16, which jointly imply Lemma 17.

99:16 Optimal (degree+1)-Coloring in Congested Clique

► **Lemma 15.** *All network nodes can simultaneously choose a hash function (in line 1 of BUCKETCOLOR) such that the size of G_{bad} is $O(n)$.*

► **Lemma 16.** *After 20 iterations of the outer for-loop of BUCKETCOLOR, all nodes in $G_2 \setminus G_{bad}$ can be colored without conflicts.*

► **Lemma 17.** *BUCKETCOLOR successfully colors graph G_2 in $O(1)$ rounds.*

6 Proof of the main theorem

Now, we are ready to complete our analysis of a constant-round CongestedClique and prove Theorem 1. We begin with a theorem summarizing the properties of $\text{COLOR}(G, 0)$.

► **Theorem 18.** *$\text{COLOR}(G, 0)$ colors any D1LC instance G with $\Delta_G \leq O(\sqrt{n})$ in $O(1)$ rounds.*

Proof. From the description of $\text{COLOR}(G, 0)$ and its subroutines, it is evident that $\text{COLOR}(G, 0)$ colors a graph G successfully when $\Delta_G \leq O(\sqrt{n})$. It remains to analyze the total number of rounds spent by $\text{COLOR}(G, 0)$.

Note that the steps of $\text{COLOR}(G, 0)$, other than the call to subroutines COLORTRIAL , SUBSAMPLE , BUCKETCOLOR and recursive call, can be executed in $O(1)$ rounds. COLORTRIAL and SUBSAMPLE can be executed in $O(1)$ rounds by Lemma 9 and Lemma 11, respectively. Also, $O(1)$ rounds are sufficient for BUCKETCOLOR due to Lemma 17.

To analyze the round complexity of recursive calls in $\text{COLOR}(G, 0)$, let G^i denote the graph on which the i^{th} -level recursive call of COLOR , i.e., $\text{COLOR}(G^i, 0.1i)$ is made. $\text{COLOR}(G^i, 0.1i)$ does $O(1)$ rounds of operations and makes a recursive call $\text{COLOR}(G^{i+1}, 0.1(i+1))$.

We show by induction that $\sum_{v \in G^i} d_{G^i}(v)^{0.1i} \leq 3^i Cn$ for $i \leq 10$. This is true for $G^0 = G$, since $\sum_{v \in G} d_G(v)^0 = n$. For the inductive step, for $1 \leq i \leq 9$, by Lemma 11 using $x = 0.1i$,

$$\sum_{v \in G^{i+1}} d_{G^{i+1}}(v)^{0.1(i+1)} \leq \sum_{v \in G^{i+1}} d_{G^i}(v)^{0.1(i+1)} \leq Cn + 2 \sum_{v \in G^i} d_{G^i}(v)^{0.1i} \leq Cn + 2 \cdot 3^i Cn \leq 3^{i+1} Cn .$$

So, $|E(G^{10})| \leq \sum_{v \in G^{10}} d_{G^{10}} \leq 3^{10} Cn = O(n)$. Therefore, after 10 recursive calls, the remaining uncolored graph can simply be collected to a single network node and solved. ◀

While Theorem 18 requires that $\Delta_G \leq O(\sqrt{n})$, we note that we can generalize this result to any maximum degree:

► **Lemma 19.** *In $O(1)$ rounds of CongestedClique, we can recursively partition an input D1LC instance into sub-instances, such that each sub-instance has maximum degree $O(\sqrt{n})$. The sub-instances can be grouped into $O(1)$ groups where each group can be colored in parallel.*

Proof. We use the LOWSPACEPARTITION procedure from [14], which reduces a coloring instance to $O(1)$ sequential instances of maximum degree n^ε for any constant $\varepsilon > 0$. The procedure is for $\Delta + 1$ -coloring, but it extends immediately to D1LC, as discussed in Section 5 of [11]. Since we can simulate low-space MPC in CongestedClique, we can execute LOWSPACEPARTITION , setting ε appropriately to reduce the maximum degree of all instances to $O(\sqrt{n})$. By subsequent arguments in [14], $O(1)$ sequential sets of base cases are created. ◀

Now the proof of Theorem 1 follows immediately from Theorem 18 and Lemma 19. ◀

References

- 1 Noga Alon and Sepehr Assadi. Palette sparsification beyond $(\Delta + 1)$ vertex coloring. In *Proceedings of the 24th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 6:1–6:22, 2020.
- 2 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.
- 3 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 767–786, 2019.
- 4 Étienne Bamas and Louis Esperet. Distributed coloring of graphs with an optimal number of colors. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 10:1–10:15, 2019.
- 5 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 243–252, 2020.
- 6 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013.
- 7 Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Brief announcement: Semi-MapReduce meets Congested Clique. *Preprint arXiv:1802.10297*, 2018.
- 8 Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Computing*, 33(3):349–366, 2020.
- 9 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\Delta + 1)$ coloring in Congested Clique, Massively Parallel Computation, and centralized local computation. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 471–480, 2019.
- 10 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. Distributed $(\Delta + 1)$ -coloring via ultrafast graph shattering. *SIAM Journal on Computing*, 49(3):497–539, 2020.
- 11 Sam Coy, Artur Czumaj, Peter Davies, and Gopinath Mishra. Fast parallel degree+1 list coloring. *Preprint arXiv:2302.04378*, 2023.
- 12 Artur Czumaj, Peter Davies, and Merav Parter. Component stability in low-space massively parallel computation. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 481–491, 2021.
- 13 Artur Czumaj, Peter Davies, and Merav Parter. Graph sparsification for derandomizing massively parallel computation with low space. *ACM Transactions on Algorithms*, 17(2), May 2021.
- 14 Artur Czumaj, Peter Davies, and Merav Parter. Improved deterministic $(\Delta + 1)$ coloring in low-space MPC. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 469–479, 2021.
- 15 Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in Congested Clique and MPC. *SIAM Journal on Computing*, 50(5):1603–1626, 2021.
- 16 Janosch Deurer, Fabian Kuhn, and Yannic Maus. Deterministic distributed dominating set approximation in the CONGEST model. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 94–103, 2019.
- 17 Paul Erdős and John L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14(3):298–301, 1973.
- 18 Manuela Fischer, Jeff Giliberti, and Christoph Grunau. Improved deterministic connectivity in massively parallel computation. In *Proceedings of the 36th International Symposium on Distributed Computing (DISC)*, pages 22:1–22:17, 2022.
- 19 Manuela Fischer, Magnús M. Halldórsson, and Yannic Maus. Fast distributed Brooks’ theorem. In *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2567–2588, 2023.

- 20 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proceedings of the 57th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 625–634, 2016.
- 21 Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 29:1–29:17, 2018.
- 22 Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *Proceedings of the 62nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1009–1020, 2021.
- 23 Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. Efficient randomized distributed coloring in CONGEST. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1180–1193, 2021.
- 24 Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan. Near-optimal distributed degree+1 coloring. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 450–463, 2022.
- 25 Magnús M. Halldórsson, Alexandre Nolin, and Tigran Tonoyan. Overcoming congestion in distributed coloring. In *Proceedings of the 41st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 26–36, 2022.
- 26 James W. Hegeman and Sriram V. Pemmaraju. Lessons from the Congested Clique applied to MapReduce. *Theoretical Computer Science*, 608:268–281, 2015.
- 27 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.
- 28 Fabian Kuhn. Faster deterministic distributed coloring through recursive list coloring. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1244–1259, 2020.
- 29 Christoph Lenzen. Optimal deterministic routing and sorting on the Congested Clique. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013.
- 30 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, February 1992.
- 31 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.
- 32 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.
- 33 Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.
- 34 Rajeev Motwani, Joseph Naor, and Moni Naor. The probabilistic method yields deterministic parallel algorithms. *Journal of Computer and System Sciences*, 49(3):478–516, 1994.
- 35 Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991.
- 36 Merav Parter. $(\Delta + 1)$ coloring in the Congested Clique model. In *Proceedings of the 45th Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pages 160:1–160:14, 2018.
- 37 Merav Parter and Hsin-Hao Su. Randomized $(\Delta + 1)$ -coloring in $O(\log^* \Delta)$ Congested Clique rounds. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 39:1–39:18, 2018.
- 38 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, 2000.
- 39 Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

- 40 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 350–363, 2020.

A Derandomization in CongestedClique

In this section we first give some useful lemmas regarding $O(1)$ -wise independence and the existence of small families of $O(1)$ -wise independent hash functions, and then we give a formal description of the method of conditional expectations and how it is implemented in the CongestedClique model.

A.1 Bounded Independence

Our algorithm will be finding a hash function of sufficient quality from a family of $O(1)$ -independent hash functions. In the following, we recall the standard notions of k -wise independent hash functions and k -wise independent random variables. Then, we recall that we can construct small families of bounded-independence hash functions, and that each hash function in this family can be specified by a short seed.

► **Definition 20.** Let $k \geq 2$ be an integer. A set $\{X_1, \dots, X_n\}$ of n random variables taking values in S are said to be k -wise independent if for any $I \subset [n]$ with $|I| \leq k$ and any $x_i \in S$ for $i \in I$, we have

$$\Pr \left[\bigwedge_{i \in [k]} X_i = x_i \right] = \prod_{i=1}^k \Pr [X_i = x_i] .$$

► **Definition 21.** A family of hash functions $\mathcal{H} : \{h : X \rightarrow Y\}$ is said to be k -wise independent if $\{h(x) : x \in X\}$ are k -wise independent when h is drawn uniformly at random from \mathcal{H} .

We use the property that small families of $O(1)$ -wise independent hash functions can be constructed, and each hash function in such a family can be specified with a small number of bits:

► **Remark 22.** For all positive integers c_1, c_2 , there is a family of k -wise independent hash functions $\mathcal{H} = \{h : [n^{c_1}] \rightarrow [n^{c_2}]\}$ such that each function from \mathcal{H} can be specified using $O(k \log n)$ bits.

A.2 The Method of Conditional Expectations

We now describe in more detail the method of conditional expectations and its implementation in CongestedClique. We briefly recall the setup to the problem: we have a randomized algorithm which “succeeds” if a “bad” outcome occurs for less than some number T of nodes. This algorithm succeeds in expectation using bounded-independence randomness. We would like to derandomize this algorithm. In order to achieve this, given a family of $O(1)$ -independent hash functions \mathcal{H} , we need to find a “good” hash function $h^* \in \mathcal{H}$ which solves our problem, when used to make decisions for nodes instead of randomness.

First, we define some cost function $f : \mathcal{H} \times V \rightarrow \{0, 1\}$ such that $f(h, v) = 1$ if the node v has a “bad” outcome when h is the selected hash function, and $f(h, v) = 0$ if the outcome is “good”. We further define $F(h) = \sum_{v \in V} f(h, v)$ as the total cost of the hash function h : i.e., the number of bad nodes when h is the selected hash function. Finally, we use $\mathbb{E}_{h \in \mathcal{H}} x(h)$

99:20 Optimal (degree+1)-Coloring in Congested Clique

to denote the expected value of some function $x(h)$ when h is drawn uniformly at random from \mathcal{H} .

To successfully derandomize our algorithm, we need to find a hash function $h^* \in \mathcal{H}$ such that $F(h^*) \leq T$. We need the following conditions to hold for our derandomization to work:

- $\mathbb{E}_{h \in \mathcal{H}}[F(h)] \leq T$ (i.e., the expected cost of a hash function selected uniformly at random from \mathcal{H} is at most T); and
- Node v can locally (i.e., without communication) evaluate $f(h, v)$ for all $h \in \mathcal{H}$.

We can now use the method of conditional expectations to find a $h^* \in \mathcal{H}$ for which $F(h^*) \leq T$. We first recall that each hash function in our family of $O(1)$ -wise independent hash functions \mathcal{H} can be specified using $O(\log n)$ bits, by Remark 22. Next, let $\Pi = \{0, 1\}^{\log n}$ be the set of binary strings of length $\log n$, and for each $\pi \in \Pi$, let \mathcal{H}_π denote the hash functions in \mathcal{H} whose seeds begin with the prefix π .

Our goal is to find some seed-prefix $\pi \in \Pi$ for which $\mathbb{E}_{h \in \mathcal{H}_\pi}[F(h)] \leq T$: the existence of such a prefix is guaranteed by the probabilistic method. Since each node v can locally evaluate $f(h, v)$ for all $h \in \mathcal{H}$, nodes can also compute $\mathbb{E}_{h \in \mathcal{H}_\pi}[f(h, v)]$ for all $\pi \in \Pi$. Since $|\Pi| = n$, each node v can be made responsible for a prefix $\pi_v \in \Pi$. Node v can then collect the value of $\mathbb{E}_{h \in \mathcal{H}_{\pi_v}}[f(h, u)]$ for each $u \in V \setminus \{v\}$: since this requires all nodes sending and receiving $O(n)$ messages it can be done in $O(1)$ rounds using Lenzen's routing algorithm [29]. Now, by linearity of expectation:

$$\sum_{v \in V} (\mathbb{E}_{h \in \mathcal{H}_{\pi_v}} [f(h, v)]) = \mathbb{E}_{h \in \mathcal{H}_{\pi_v}} [F(h)] \quad .$$

Therefore v can compute the expected value of F for the sub-family of hash functions which are prefixed with π_v . Nodes can broadcast this expected value to all other nodes in $O(1)$ rounds, again using Lenzen's routing algorithm [29]. All nodes then know the expected value of F for all $(\log n)$ -bit prefixes and can, without communication (breaking ties in a predetermined and arbitrary way), pick the prefix with the lowest expected value of F . Recall that this prefix is guaranteed to have an expected value of at most T by the probabilistic method.

We have now fixed the first $(\log n)$ bits of the prefix and obtained a smaller set $\mathcal{H}_1 \subset \mathcal{H}$ of hash functions. We can then perform the same procedure described above on \mathcal{H}_1 to set the next $(\log n)$ bits of the seed, obtaining a smaller set $\mathcal{H}_2 \subset \mathcal{H}_1 \subset \mathcal{H}$ of hash functions. After repeating this procedure $O(1)$ times we will have fixed the entire seed, since we fix $(\log n)$ bits each time and the seeds of hash functions in \mathcal{H} were $O(\log n)$ bits in length.