Distributed Graph Coloring Made Easy

Yannic Maus*
yannic.maus@cs.technion.ac.il
Technion
Haifa, Israel

ABSTRACT

In this paper we present a deterministic CONGEST algorithm to compute an $O(k\Delta)$-vertex coloring in $O(\Delta/k) + \log^* n$ rounds, where Δ is the maximum degree of the network graph and $1 \leq k \leq O(\Delta)$ can be freely chosen. The algorithm is extremely simple: Each node locally computes a sequence of colors and then it *tries colors* from the sequence in batches of size k. Our algorithm subsumes many important results in the history of distributed graph coloring as special cases, including Linial's color reduction [Linial, FOCS'87], the celebrated locally iterative algorithm from [Barenboim, Elkin, Goldenberg, PODC'18], and various algorithms to compute defective and arbdefective colorings. Our algorithm can smoothly scale between these and also simplifies the state of the art $(\Delta + 1)$-coloring algorithm. At the cost of losing the full algorithm's simplicity we also provide a $O(k\Delta)$-coloring algorithm in $O(\sqrt{\Delta/k}) + \log^* n$ rounds. We also provide improved deterministic algorithms for ruling sets, and, additionally, we provide a tight characterization for one-round color reduction algorithms.

CCS CONCEPTS

• **Mathematics of computing** → **Graph coloring**; • **Theory of computation** → **Distributed algorithms**.

KEYWORDS

graph coloring; deterministic; CONGEST; LOCAL; lower bound

ACM Reference Format:
Yannic Maus. 2021. Distributed Graph Coloring Made Easy. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3409964.3461804

1 INTRODUCTION

In the *C-vertex coloring* problem the objective is to assign each vertex of an n-node graph $G = (V, E)$ one of C colors such that adjacent vertices get different colors. In the distributed setting graph coloring is considered to be one of the core *symmetry breaking problems* with a huge amount of published work and even a whole book almost exclusively covering the problem [14]. In this setting, the usual goal is to compute a C-coloring with $\Delta + 1 \leq C \leq O(\Delta^2)$, where Δ is the maximum degree of the graph. The bound of $\Delta + 1$ stems from the fact that any graph can be colored with the respective number of colors and this can even be done with a simple sequential greedy algorithm. The bound of $O(\Delta^2)$ colors stems from an algorithm in Linial's seminal paper in which he introduces one of the core models for distributed graph algorithms, i.e., the LOCAL model [42]. In this model, the graph abstracts a communication network in which the nodes communicate through the edges in synchronous rounds and at the end of the computation each node needs to output its own part of the solution, e.g., its own color. The complexity measure is the number of synchronous rounds. Linial gave an extremely fast $O(\Delta^2)$-coloring algorithm that only uses $O(\log^* n)$ rounds. Further, he showed that $\Omega(\log^* n)$ rounds are needed to color rings ($\Delta = 2$) with $O(1) = \text{poly}\,\Delta$ colors. Due to the lower bound, a vast amount of published works, e.g., [9, 15, 16, 25, 41, 43, 51], study the setting that occurs after applying Linial's coloring algorithm, i.e., they ask: *Given an $O(\Delta^2)$-coloring, how fast can one reduce the number of colors where the runtime of the algorithm can only depend on Δ?* The similar question of finding a fast algorithm with complexity $f(\Delta) + \log^* n$ is also sometimes referred to as determining the *truly local complexity* of a problem [43].

In the current paper, we devise several results to advance the understanding of this setting. First, we provide a *simple* deterministic algorithm that scales between the two extremes of $\Delta + 1$ (or rather $O(\Delta)$ colors) and $O(\Delta^2)$ colors. In particular, for a parameter $1 \leq k \leq O(\Delta)$ of the user's choice the algorithm computes a $O(k\Delta)$ coloring in $O(\Delta/k)$ rounds. In the algorithm, each node v uses its input color, e.g., provided by Linial's algorithm, to (locally) compute a sequence $p_v(0), p_v(1), \ldots$ of colors. Then, node v *tries* to get colored with one of the first k colors in its sequence. If it tries a color c that is not *conflicting* with the neighbors' trials node v gets permanently colored with color c, otherwise v continues to the next round in which it tries the next k colors in its sequence, and so on.

Second, we show that this simple *mother algorithm* either immediately yields the core steps of the aforementioned papers, e.g., the algorithms in [15, 42], or can be mildly adapted to obtain crucial subroutines developed or used in [9, 10, 15, 16, 25, 37, 43]. E.g., a mild adaption yields *d-defective colorings* which were the crucial ingredient in [10, 16, 37], or *arbdefective colorings* which were crucial in [9, 15, 25, 43]. A core strength of our result is that the algorithms for each of these results are very similar. To get a feeling for this let us look at defective colorings. In a *d-defective coloring* a node is allowed to have at most d neighbors with the same color. Besides a suitable choice of k and sequences $p_v(0), p_v(1), \ldots$, the only change is in the execution of the algorithm: When deciding whether to keep a color c a node tolerates up to d neighbors with the same

*This work was supported by the European Union's Horizon 2020 Research And Innovation Programme under grant agreement no. 755839.

SPAA '21, July 6–8, 2021, Virtual Event, USA.

ACM ISBN 978-1-4503-8070-6/21/07…$15.00
https://doi.org/10.1145/3409964.3461804

color. This algorithm does not yet do the job as the defect of a node v might be larger than d at the end of the execution if one or more neighbors of v choose the same color as v in later rounds, but it still captures the essence of the adaptions that need to be performed.

Third, at the cost of losing the full simplicity of the algorithm, we show how to compute an $O(\Delta^{1+\varepsilon})$-coloring in $O(\Delta^{1/2-\varepsilon/2})$ rounds, and we improve the state of the art runtime for computing so called *ruling sets*.

Fourth, we provide a full characterization of one-round coloring algorithms. Given a graph with m colors, we precisely identify the integer number q such that there is a one-round algorithm to reduce an m coloring to a q coloring, while it is impossible to compute a $q-1$ coloring in one round.

Next, we describe why this result, combined with our $O(k\Delta)$-coloring algorithm in $O(\Delta/k)$ rounds might be of additional interest. Already [35, 41, 51] studied one-round color reduction algorithms and showed lower bounds like our fourth contribution; then in [51] these one-round lower bounds on the number of colors were used to prove a heuristic $\Omega(\Delta \log \Delta)$ runtime lower bound for computing a $(\Delta+1)$-coloring. As already pointed out by [51] the bound is *heuristic* in the following sense (the following example uses the result of our paper): Given a coloring with at most 2Δ colors, we can reduce exactly one color in a single round. By applying this tight bound twice, one would wish to claim that one cannot go from a 2Δ coloring to a $2\Delta-2$ coloring in two rounds. However, this claim cannot be deduced via this method, as the second application of the one-round lower bound assumes that the *intermediate* $2\Delta-1$ *coloring* is *worst case*. But, instead a two round algorithm might not produce an intermediate coloring at all, or it might output a very specific intermediate coloring which enables it to reduce more than one color in its second round. The heuristic lower bound in [51] is obtained by applying one-round lower bounds iteratively, purposely ignoring this important subtlety. Since the publication of [51] at least five different algorithms for $(\Delta+1)$-coloring that beat this lower bound were published: Given an $O(\Delta^2)$-coloring, a $(\Delta+1)$-coloring was computed in $O(\Delta)$ rounds [10, 16, 37], in $O(\Delta^{3/4})$ rounds in [9], in $O(\sqrt{\Delta \log \Delta} \log^* \Delta)$ rounds in [15, 25] and in $O(\sqrt{\Delta \log \Delta})$ rounds in [43]. Formally, only the *locally iterative* $O(\Delta)$-round algorithm by Barenboim, Elkin and Goldenberg [15] beats this lower bound and we explain why. It starts with an $O(\Delta^2)$ coloring and maintains a feasible coloring in each round. The color of a node in the next round only depends on the node's own color and the colors of its neighbors. The algorithm was celebrated as it is significantly simpler than the aforementioned faster algorithms and it breaks the aforementioned heuristic lower bound in a *clean way*, due to maintaining a feasible coloring in each round.

While we formally do not maintain a feasible coloring in each round in our algorithms[1], we provide a different insight. Our *tight* lower bounds for one-round algorithms provide a heuristic argument that shows that one needs $\Omega(\Delta)$ rounds to go from a $O(\Delta^2)$ coloring to a $\Delta^2/5$-coloring (the 5 is chosen somewhat arbitrarily), and for a suitable $k = O(\Delta)$, our mother algorithm provides a $O(\Delta/k) = O(1)$-round algorithm to perform such a color reduction. Thus the heuristic lower bound that is based on the repeated application of *tight* one-round lower bounds can be beaten significantly in the number of colors by a simple $O(1)$-round algorithm. In contrast, all the previous algorithms that provide such an insight require poly Δ rounds. This is in particular interesting, as there has been almost no progress in proving lower bounds for the $(\Delta+1)$-coloring problem since Linial's initial seminal $\Omega(\log^* n)$ lower bound. Just as we do in this paper, the only other known lower bounds in the LOCAL model study one-round algorithms [35, 41, 51]. Our paper suggests that even in constant-time coloring algorithms there are still results to be discovered. In fact, we do not even know a lower bound on the number of colors that a two-round algorithm must use. Surprisingly, there is not even a lower bound, that rules out that one can go from a $O(\Delta^2)$ coloring to a $\Delta+1$ coloring in two rounds. Further evidence that understanding constant-time or even just two-round algorithms is given by [43]. It provides a two-round algorithm for a *list coloring* variant of Linial's color reduction. In *list coloring* each node v has a list $L(v)$ of colors and needs to output a color from this list. Basically, the authors show that one can compute a list-coloring in two rounds if lists are of size $\tilde{\Omega}(\Delta^2)$—the 'equivalent' of the $O(\Delta^2)$ colors in Linial's coloring—, and interestingly for their choice of parameters the exact same problem cannot be solved in one round. Of course, just as applying a lower bound for one-round algorithms twice does not give a tight lower bound for two rounds it is unclear whether understanding two-round algorithms will yield a result for the holy grail, a tight runtime lower bound for $(\Delta+1)$-coloring

Our Contributions

While we have already explained our contributions from a high level point of view we use this section to state them formally, additional related work is presented afterwards. Our results hold in the LOCAL model and in the CONGEST model (both lower and upper bounds).

The LOCAL *and* CONGEST *Model of distributed computing [42, 47].* In both models the graph is abstracted as an n-node network $G = (V, E)$ with maximum degree at most Δ. Communication happens in synchronous rounds. Per round, each node can send one message to each of its neighbors. At the end, each node has to know its own part of the output, e.g., its own color. In the LOCAL model there is no bound on the message size and in the CONGEST model messages can contain at most $O(\log n)$ bits. Usually, in both models nodes are equipped with $O(\log n)$ bit IDs and initially, nodes know their own ID or their own color in an input coloring but are unaware of the IDs of their neighbors.

Linial's algorithm treats the unique IDs as an input coloring to compute an $O(\Delta^2)$-coloring in $O(\log^* n)$ rounds, merely in one *color reduction step* he reduces an m-input coloring to an $O(\Delta^2 \operatorname{poly} \log m)$-coloring, which then serves as the input coloring for the next step. All of our algorithms do not make use of unique IDs but work in the more general setting where nodes are only equipped with some input coloring with m colors. Similarly to most previously-known results, we assume that m and Δ are global knowledge, which can at least partially be removed for our upper bounds with standard techniques [36]. It is easiest to grasp our results when

[1] It is straightforward to tweak the algorithms to actually achieve this at the cost of loosing some of the simplicity of the algorithm, e.g., by encoding the state of a node into a proper vertex coloring.

setting $m = O(\Delta^2)$, that is, one first applies Linial's algorithm. Our main technical result is the following theorem.

Theorem 1.1. *There is a deterministic CONGEST algorithm that, given a graph with an input m-coloring and maximum degree at most Δ and parameters $0 \leq d \leq \Delta - 1$ and $1 \leq k \leq X$ that are known to all vertices, uses $R = \lceil X/k \rceil$ rounds and computes a coloring with $C = X \cdot k$ colors, where $X = 4 \cdot (\Delta/(d+1)) \lceil \log_{\Delta/(d+1)} m \rceil$, and s.t.*

(1) the graph induced by each color class admits an orientation of its edges with outdegree at most d ,
(2) each color class can be partitioned into R induced subgraphs $P_1, \ldots, P_R$ of degree at most d .

The orientation and the partition can be computed with no additional cost in the round complexity.

While the algorithm for Theorem 1.1 is extremely simple (locally compute a permutation of the output colors, try them in batches of size k and tolerate up to d conflicts), the theorem, stated in its general form, has a rather technical appearance to fit various choices of parameters at once. But, we believe that it is very approachable as soon as one considers precise choices for its parameters. E.g., we can first use Linial's algorithm (or an algorithm derived from Theorem 1.1) to compute a poly Δ-coloring in $O(\log^* n)$ rounds. If we treat this coloring as an input coloring with $m = \text{poly}\,\Delta$ colors and if also $d = \Delta^{1-\Omega(1)}$ (e.g., when $d = 0$ or $d = \Delta^\varepsilon$ for constant $0 < \varepsilon < 1$), one can replace the term $\lceil \log_{\Delta/(d+1)} m \rceil$ with a constant.

One step of Linial's color reduction is based on a suitable so-called *low-intersecting set family* $S_1, \ldots, S_m$; he uses the probabilistic method to show that the respective families exist. In the algorithm a node with input color i simultaneously tries all colors in S_i, and as $S_i \cap S_j$ is *small* for each neighbor's input color $j \neq i$, it is guaranteed that at least one color in S_i is not tried by any neighbor. The low-intersecting set families obtained by the probabilistic method are not strong enough to go from a poly Δ-coloring to an $O(\Delta^2)$-coloring such that he also uses a different construction for such families, based on polynomials.[2] At its core is the observation that for two distinct polynomials p_1 and p_2 with degree d over a suitable finite set $\mathbb{F}_q$, the sets $S_i = \{(x, p_i(x)) \mid x \in \mathbb{F}_q\}$, $i = 1, 2$ intersect in at most d elements. Choosing m distinct polynomials yields the respective set family with m sets. This simple observation by Linial is also the core of our main result; in particular Linial's color one-round color reduction is a special case of our more general Theorem 1.1. Next, we discuss various settings for the parameters in Theorem 1.1 and explain which results it subsumes. An approachable summary is contained in Corollary 1.2. If $d = 0$, the computed coloring is proper and point (1) and (2) in Theorem 1.1 can be ignored. The parameter k trades the number of rounds versus the number of colors. For the extreme choice of $k = X = O(\Delta)$ we obtain the aforementioned color reduction by Linial (the one build on top of polynomials), and for $k = 1$ we obtain a generalization of the locally iterative algorithm of [15]. Other values of k scale between both algorithms and provide an extremely simple way to compute $O(\Delta \cdot k)$-coloring in $O(\Delta/k)$ rounds. While our algorithm for $k = 1$ only computes an $O(\Delta)$-coloring in $O(\Delta)$ rounds, we can use an additional $O(\Delta)$ rounds in each of which we remove a single color class to transform it into a $(\Delta + 1)$-coloring.

We now explain Theorem 1.1 in the case of $d > 0$. A *d-defective c-coloring* is a vertex coloring with c colors in which each vertex has at most d neighbors with the same color. A *β-out degree c-coloring* is a vertex coloring with c colors together with an orientation of the edges between neighbors with the same color such that each node has at most β-outgoing edges. Note that the edges between vertices with different colors do not need to be oriented.[3] For $d = \beta = \Delta^{1-\Omega(1)}$ and $k = 1$ and due to the first condition of Theorem 1.1, we obtain a simple β-out degree $O(\Delta/\beta)$-coloring algorithm that runs in $O(\Delta/\beta)$ rounds. Further, by assigning a vertex v with output color $\varphi(v)$ the color tuple $(\varphi(v), i)$ where i is the index of the subgraph P_i that it belongs to in (2), we obtain a d-defective $O((\Delta/d)^2)$-coloring in $O(\Delta/d)$ rounds. For $k = X = O(\Delta)$, the same defective coloring can be computed in one round. This simplifies and subsumes several results in the literature.

A β-out degree $O(\Delta/\beta)$-coloring algorithm is one of the two crucial components in the state of the art $(\Delta+1)$-coloring algorithm in [43]. Our simpler algorithm to compute such a coloring thus also simplifies the overall algorithm (see Section 2 for details).

The next corollary summarizes the parameter settings in Theorem 1.1, that are most interesting with our current knowledge. In the future, other settings of parameters might be of interest.

Corollary 1.2. *There are the following deterministic CONGEST algorithms with the presented runtimes, given an Δ^4-input coloring,*

(1) $256\Delta^2$-coloring in 1 round. (Linial's color reduction [42])
(2) $16\Delta \cdot k$-coloring in $O(\Delta/k)$ rounds. (subsumes results in [10, 16, 37])
(3) Δ^2-coloring in $O(1)$ rounds.

For $d = \beta = \Delta^{1-\Omega(1)}$ we have

(4) β-out degree $O(\Delta/\beta)$-coloring in $O(\Delta/\beta)$ rounds (subsumes a result in [15])
(5) d-defective $O(\Delta/d)^2 \log$-coloring in 1 round (subsumes a result in [16, 37])
(6) d-defective $O((\Delta/d)^2)$ coloring in $O(\Delta/d)$ rounds (subsumes some results in [10, 16])

The required Δ^4-input coloring can be computed with Linial's algorithm for a sufficiently large constant Δ. The precise choice of the constants in the O-notation in the defective coloring in Corollary 1.2 depends linearly on the exponent x in $d = \Delta^{1-\Omega(1)}$. The algorithm of (6) is clearly inferior to the one in (5), as it computes a d-defective coloring with the same number of colors but is slower. We merely state (6) for its proof which gives a slightly different perspective on Theorem 1.1; the details are deferred. While several of these colorings could be computed in the same runtime before, many of the previous algorithms are more involved and for example require carefully selected recursive applications of defective colorings. In contrast all of our algorithms are extremely simple and also the local computation of the sequence is simple. Each vertex is assigned a polynomial over a finite field. The polynomial depends on the node's input color and the sequence consists of the

[2] Szegedy and Vishnawatan show how to compute a coloring with poly Δ colors in $0.5 \log^* n$ rounds, if this algorithm is followed by $O(1)$ iterations of Linial's color reduction based on polynomials, this implies an $O(\Delta^2)$-coloring algorithm in $0.5 \log^* n + O(1)$ rounds [51].

[3] These colorings with a bound on the outdegree are closely related to *arbdefective* colorings which were introduced in [12] and have played a significant role in the development of sublinear in Δ algorithms (more details in [43]).

evaluation of the polynomial at all of the field's elements. At the cost of losing the full simplicity of the algorithm (see Section 2 for details), we also provide algorithms that are faster than the state of the art and show the following theorem.

Theorem 1.3 ($O(k\Delta)$-coloring in $O(\sqrt{\Delta/k})$ rounds). *For any constant $\varepsilon > 0$, there is a deterministic* CONGEST *algorithm to compute a $O(\Delta^{1+\varepsilon})$-coloring in $O(\Delta^{1/2-\varepsilon/2}) + \log^* n$ rounds.*

For an integer $r \geq 1$, a $(2, r)$-ruling set of a graph $G = (V, E)$ is a subset $S \subseteq V$ of the vertices that is an independent set and satisfies that for any vertex $v \in V$ there is a vertex $s \in S$ in hop distance at most r [1]. Ruling sets and its extensions (larger distance between nodes in S) have played an important role as subroutines in several algorithms, e.g., [1, 23, 29, 46]. We provide a faster algorithm for $(2, r)$-ruling sets.

Theorem 1.4. *For any constant integer $r \geq 2$ a $(2, r)$-ruling set can be computed in $O(\Delta^{\frac{2}{r+2}}) + \log^* n$ rounds, deterministically in the* CONGEST *model.*

The fastest previous algorithm used $O(\Delta^{2/r}) + \log^* n$ rounds [49]. So, e.g., for $r = 2$ the Δ-dependency improves from $O(\Delta)$ to $O(\sqrt{\Delta})$ and for $r = 3$ it improves from $O(\Delta^{2/3})$ to $O(\Delta^{3/5})$. For $r = 1$ the problem is equivalent to the *maximal independent set problem* and has a $\Omega(\Delta)$ lower bound, if the dependency on n is limited to $O(\log^* n)$ round [3].

Additionally, we give tight characterization for one-round color reduction algorithms, given an m-input coloring with $\Delta + 1 \leq m \leq \Delta^2/4 + 3/2\Delta + 9/4$ and no unique IDs.

Theorem 1.5. *For any integer $\Delta \geq 1$ and $\Delta + 1 \leq m \leq \Delta^2/4 + 3/2\Delta + 9/4$ let $1 \leq k \leq \min\{\Delta - 1, \Delta/2 + 3/2\}$ be the largest integer such that $m \geq k(\Delta - k + 3)$. Then, there is a one-round* CONGEST *algorithm that reduces an m-coloring to an $(m-k)$-coloring, but there is no one-round* LOCAL *algorithm to compute a $(m - k - 1)$-coloring, given an m-coloring.*

Theorem 1.5 roughly states that reducing k colors requires $k\Delta - \Theta(k^2)$ input colors. For concrete choices of k the bound in Theorem 1.5 says that to reduce 1 color one needs at least $\Delta + 2$ input colors, to reduce 2 colors one needs $2\Delta + 2$ input colors, to reduce 3 colors one needs 3Δ input colors, and to reduce 4 colors one needs $4\Delta - 4$ input colors, and so on … .

The fastest randomized algorithms compute $O(\Delta)$ colorings in $O(\log^* n)$ rounds for $\Delta \geq \operatorname{poly}\log n$ [22, 50] and they can be adapted to also compute $(1 + \varepsilon)\Delta$-colorings. However, it seems that the hardest part of $(\Delta + 1)$-coloring is to reduce a $(1 + \varepsilon)\Delta$ coloring to a $(\Delta + 1)$-coloring. We show, that given an algorithm with runtime T that reduces an input coloring with $(1 + \varepsilon)\Delta$ colors to a $(\Delta + 1)$ coloring can be used with $O(\log_{1+\varepsilon} \Delta)$ overhead to reduce a $O(\Delta^2)$-coloring to a $(\Delta + 1)$-coloring. If $T = \Delta^{\Omega(1)}$, there is only a constant factor overhead (for details see Section 4).

Related Work

The state of the art for $(\Delta + 1)$-coloring when the runtime is expressed as $f(\Delta) + \log^* n$ is $O(\sqrt{\Delta \log \Delta}) + \log^* n$ rounds and given by [43]. Just, as the slightly slower algorithms [9, 15, 25] the result of [43] works for the more general $(deg + 1)$-list coloring problem in which the size of the list of each node exceeds its degree. The result in [25] even applies for the even more general *local conflict coloring problem* in which one can specify for each edge of the graph which colors are not allowed to be adjacent. For an extensive overview on algorithms whose runtime is $f(\Delta) + \log^* n$ as well as an overview on the influence of arbdefective colorings during the last decade we refer to the related work section in [43]. Further, almost all published works until 2013 are discussed in the excellent monograph Barenboim and Elkin [14], and another very detailed overview on more recent results on coloring works is contained in [38]. Detailed overviews on randomized algorithms are contained in [22, 31]. Due to the sheer amount of published work on distributed coloring we focus on selected results that have not been discussed in detail in [14, 22, 31, 38, 43], are most related to the current work, or indicate in which direction future research should continue, or should probably not continue.

Unlike as for $(\Delta+1)$-vertex coloring, we already know subpolynomial in Δ algorithms for the corresponding edge coloring problem in the LOCAL model. The $(2\Delta - 1)$-edge coloring problem—as an edge coloring is a vertex coloring of the line graph, a $2\Delta - 1$ edge coloring is the edge coloring version of $(\Delta + 1)$-vertex coloring—has a $f(\Delta) + \log^* n$ LOCAL algorithm with $f(\Delta) = \text{quasi poly}\log\Delta$ [7]. In the CONGEST model the state of the art deterministic for $(2\Delta - 1)$-edge coloring is an $O(\Delta + \log^* n)$ round algorithm [15].

Besides optimizing the dependency on Δ after spending only $\log^* n$ rounds on Linial's coloring algorithm, another big branch of research has tried to settle the complexity of the problem as a function of n. For almost thirty years the best deterministic algorithm in this regime was a $2^{O(\sqrt{\log n})} \gg \operatorname{poly}\log n$ round algorithm [1, 46], that has been improved to $O(\log^5 n)$ rounds [27, 48] in the LOCAL model and in the CONGEST model [8, 27] (slightly slower). The crucial building block of all of these results is a decomposition of the graph into $O(\log n)$ classes $C_1, \ldots, C_{O(\log n)}$ of small diameter clusters. To solve the $(\Delta+1)$-coloring problem one iterates through the $O(\log n)$ classes and solves each cluster $C \in C_i$ in parallel in time that is (at least) linear in the cluster diameter. Even existentially, such decompositions require that the cluster diameter is at least $\Omega(\log n)$ and as a result these methods can probably not yield runtimes that are $o(\log^2 n)$. Thus, the fastest algorithm [30] that needs $O(\log n \log^2 \Delta)$ rounds uses a different approach: Similar to [8] it derandomizes a simple randomized one-round algorithm. Output colors are represented as bit strings of length $O(\log \Delta)$ and in one round of the algorithm each node flips a (suitably weighted) coin to determine the next bit in the string. In expectation, after all $O(\log \Delta)$ bits are fixed a constant fraction of the vertices can be colored. Bamberger, Kuhn and Maus derandomize this algorithm for each cluster of a given network decomposition [8]. In contrast, instead of computing a network decomposition and derandomizing within a cluster, Ghaffari and Kuhn derandomize the algorithm *globally* with the help of a special kind of a defective coloring [30]. In this work, the derandomization step takes $O(\log \Delta)$ rounds for each of the $O(\log \Delta)$ bits and the $O(\log n)$ factor follows as only a constant fraction of the vertices get colored in each phase, yielding a total runtime of $O(\log^2 \Delta \log n)$ rounds. Similar methods, also yielding $\log n \cdot \operatorname{poly}\log\Delta$ runtimes, have been successful for edge-coloring [28, 33] and computing maximal matchings [24]. If one

allows $O(\Delta^{1+\varepsilon})$ colors for a constant $\varepsilon > 0$ a $O(\log\Delta\log n)$ round algorithm has been known for more than a decade [12].

As shown in [19, 21] a logarithmic dependency on n ($\log\log n$-dependency for randomized algorithms) is unavoidable if one colors with fewer than $\Delta+1$ colors, that is, Δ-coloring requires at least $\Omega(\log n)$ rounds. Similar bounds hold for the edge coloring problem for coloring with fewer than $(2\Delta-1)$-colors [6] and for coloring trees and bounded arboricity graphs with significantly fewer than Δ colors [11, 42].

Little is known on lower bounds for C-coloring when $C \geq \Delta+1$ (in contrast to other symmetry breaking problems, e.g., maximal matching, MIS or ruling sets [3, 4, 40]). Linial's $\Omega(\log^* n)$ deterministic lower bound has recently been re-proven in a topological framework [26]. A $\Omega(\Delta^{1/3})$ lower bound for $O(\Delta)$-coloring holds in a weak variant of the LOCAL model [35]. Several works analyzed special cases of coloring algorithms which can only spend a single communication round [35, 41, 51]. Just, as the lower bounds in this paper, none of these results gives anything non-trivial for two rounds. Also, the *speedup* technique (e.g., [2–4, 18–20]), which proved very successful, e.g., for MIS and ruling set lower bounds, is not yet helpful for graph coloring. To make full use of the technique, one uses a computer program [45] to automatically transfer a problem P_0, e.g., the $(\Delta+1)$-coloring problem, into a problem P_1 that requires exactly one communication round less in the LOCAL model. Then, one iterates the process to obtain problems $P_0, P_1, \ldots, P_t$, and if P_t cannot be solved with a 0-round algorithm, problem P_0 has a lower bound of t rounds. Usually, the program is applied for small values of Δ, and in a second step, the gained insights are transferred into a formal proof for general Δ. For graph coloring the description of the problems grows so quickly with t that even for small values of Δ one cannot even compute P_1 with current computers.

There has also been a lot of progress in randomized coloring algorithm, e.g, [17, 22, 31, 34] where the state of the art for $(\Delta+1)$-vertex coloring is a $O(\log^3\log n)$ algorithm in the LOCAL model [22, 30] and $O(\log^6\log n)$ in the CONGEST model [31]. Remarkably, there is a randomized $O(\log^*\Delta)$ round algorithm to compute a coloring with $\Delta+\log^\gamma n$ colors for a large enough constant $\gamma > 0$ [22]. Prior to this Schneider and Wattenhofer [50] showed that one can compute a $O(\Delta+\log^{1.1} n)$-coloring in $O(\log^*\Delta)$ rounds of LOCAL. Very recently, Halldórsson and Nolin showed that these results can be extended to the CONGEST model [32]. All of these latter randomized algorithm make use of the concept of trying several colors in one round, similar to our algorithm for $k > 1$. While Naor extended Linial's $\Omega(\log^* n)$ lower bound to randomized algorithms [44] (on rings with $\Delta = 2$) it is not known whether the bounds for $O(\Delta)$-coloring for large Δ are tight. When $\Delta \geq \operatorname{poly}\log n$ holds, our current knowledge does not rule out $O(1)$-round algorithms for $O(\Delta)$-coloring. This question is even more of interest as in this setting a poly Δ-coloring can be computed in one round from unique IDs from a space of size poly n.

Roadmap

In Section 2 we present the $O(\Delta/k)$-round $O(k\Delta)$-coloring algorithm, its implications and modifications to compute defective and out degree colorings, our $O(k\Delta)$-coloring in $O(\sqrt{\Delta/k})$-rounds, and our results on computing ruling sets. In Section 3 we analyze one-round color reduction algorithms. In Section 4 we conclude and explain why reducing a $(1+\varepsilon)\Delta$-input coloring to a $(\Delta+1)$-coloring might be the hardest part of the $(\Delta+1)$-coloring problem.

2 MAIN ALGORITHM: COLORING MADE EASY

The objective of this section is to prove Theorem 1.1 where the emphasis is on the fact that the algorithm is extremely simple if one ignores the precise choice of parameters. Before we prove Theorem 1.1, we prove Corollary 1.2 with useful settings of the respective parameters in Theorem 1.1; while Theorem 1.1 is the technical result, the corollary is supposed to be the framework to the outer world. To formally state the corollary, we begin with two definitions.

Proof of Corollary 1.2. In each of the results we apply Theorem 1.1 with different parameters. **Proof of (1).** Choose $d = 0$ which implies $X = 16\Delta$. With $k = X$ we obtain a proper $C = X\cdot k = 256\Delta^2$-coloring in one ($R = X/k = 1$) round. **Proof of (2).** Choose $d = 0$, which implies $X = 16\Delta$. Then we obtain a $16\Delta\cdot k$ coloring in $R = 16\Delta/k$ rounds. **Proof of (3).** Choose all parameters as in 2., but set $k = \lceil\Delta/16\rceil$, which implies Δ^2 colors in $R = 16\Delta/k = O(1)$ rounds.

In (4)–(6). the condition $d = \beta = \Delta^{1-\Omega(1)}$ implies

$$X = O(\Delta/(\beta+1)\cdot\log_{\Delta/(\beta+1)}\Delta^4) = O(\Delta/\beta).$$

Proof of (4). Let $k = 1$ which implies the claimed number of colors ($X\cdot k = O(\Delta/(\beta+1))$) and the claimed round complexity ($R = O(\Delta/\beta)$). The coloring is a β-out degree coloring due to part (1) of Theorem 1.1. **Proof of (5).** With $k = X$ the runtime is $R = X/k = 1$ rounds and we obtain $C = X\cdot k = O((\Delta/d)^2)$) colors. Theorem 1.1 says that the coloring has defect at most d as there is only one subgraph ($R = 1$). **Proof of (6).** Choose $k = 1$. Let $P_1, \ldots, P_R$ be the partition of part (2). If vertices consider their color and the index of their part of the partition as a color tuple, i.e., if a vertex $v \in P_j$ with color $\varphi(v)$ colors itself if color $(\varphi(v), j)$ we obtain a d-defective coloring with $O((\Delta/d)^2)$ colors in $O(\Delta/d)$ rounds. □

Corollary 1.2 shows that one algorithm is sufficient for many of the essential steps of several previous important works, and it further allows to smoothly scale between these results. Additionally, the algorithm for computing β-out degree colorings is simpler and more direct, e.g.,the algorithm in [15] first needs to computes a certain defective coloring and only then can proceed to compute a low outdegree coloring, the slower algorithm in [9] uses a more involved recursive algorithm. The algorithm(s) [11, 13] are more involved and require $\Omega(\log n)$ rounds.

We continue with explaining the algebraic basics to construct the sequences for the algorithm for Theorem 1.1 (Algorithm 1). Given a prime q let $\mathbb{F}_q$ denote the field of size q over the elements $[q] = \{0, \ldots, q-1\}$ and let

$$P_q^f = \{p : \mathbb{F}_q \to \mathbb{F}_q \mid p \text{ is polynomial of degree} \leq f\}$$

be the set of all polynomials over $\mathbb{F}_q$ of degree at most f. To run the algorithm on a graph with maximum degree Δ, an input m-coloring and a *defect parameter* d fix $f = \lceil\log_{\Delta/(d+1)} m\rceil$ and a prime q with

$$2f\cdot\Delta/(d+1) < q < 4f\cdot\Delta/(d+1), \tag{1}$$

which exists due to Bertrand's postulate. Then we can locally and without communication assign each input color $i \in [m]$ a distinct polynomial $p_i \in P_q^f$ as $m \leq |P_q^f| = q^{f+1}$ and since all vertices know m and f.[4] Given these polynomials, uncolored nodes try the colors in their sequences in batches of size k (k is an integer parameter that can be freely chosen). To *try a color* it is sent to its neighbors. A node gets permanently colored with a color c or *adopts* color c if it causes conflicts with at most d neighbors. To this end call a color c *d-proper* in some iteration if at most d neighbors try color c in the same iteration or are already permanently colored with color c. The details are given by Algorithm 1 and the paragraph thereafter.

Algorithm 1: for vertex with color i. Parameters d, k, m, Δ.

Locally compute:
polynomial $p_i : \mathbb{F}_q \to \mathbb{F}_q$ with q chosen by (1)
sequence s_i: $(x \mod k, p_i(x) \mod q)$, $x = 0, \ldots, q-1$
Process s_i in disjoint batches B_j of size k, for $j = 1, \ldots, \lceil \frac{q}{k} \rceil$
Try the colors in batch B_j (in a single round)
if $\exists$ (d-*proper* $c \in B_j$) **then** adopt c, join P_j, and **return**;

Processing one batch takes one round of communication. If k does not divide q, a node that is uncolored before the last iteration tries less than k tuples in the last iteration, i.e., $|B_{\lceil q/k \rceil}| < k$. In fact, it will try $q - k\lfloor q/k \rfloor$ tuples. When a vertex picks a tuple as its permanent color c, it orients all edges towards neighbors that have previously chosen c as a permanent color. If two neighbors both pick the same permanent color c in the same iteration, the edge between them is oriented arbitrarily (e.g., using the input coloring for symmetry breaking from smaller input color to larger input color). A node joins the subgraph P_j where j is the index of the iteration in which it decides for a permanent color.

Note that vertices with the same input color compute the same sequence and that all steps that are related to orientations and subgraphs are obsolete when $d = 0$. We will show that the algorithm is well defined, i.e., that every vertex is colored before it reaches the end of its sequence. To prove the result we need the following well known algebraic result on the number of intersections of two degree bounded polynomials over finite fields.

Lemma 2.1. *Let q be a prime, $f \in \mathbb{N}_0$ and let $p_1, p_2 \in P_q^f$ distinct polynomials of degree f_1, f_2, respectively. Then there are at most* $\max\{f_1, f_2\}$ *points in which p_1 and p_2 intersect, i.e.,* $|\{x \in \mathbb{F}_q \mid p_1(x) = p_2(x)\}| \leq \max\{f_1, f_2\}$.

Proof of Theorem 1.1. Recall, that the prime number q is the size the of the field $\mathbb{F}_q$ from which the coefficients of the polynomials are taken. Due to the choice of q and f (see Equation (1)) the set P_q^f contains at least one distinct polynomial for each input color $i \in [m]$. Recall, the choice of parameters: $X = 4\Delta/(d+1)\lceil \log_{\Delta/(d+1)} m \rceil$, $R = \Delta/k$ and $C = X \cdot k$, and note that $X \geq q$. We first prove all statements under the assumption that all vertices are colored after R iterations of the while loop; afterwards we show that this is indeed the case.

[4]For example we can represent each element $p(x) = \sum_{i=0}^{f} a_i x^i$ of P_q^f as a tuple $(a_0, \ldots, a_f)$, order the tuples lexicographically and assign the polynomial corresponding to the i-th tuple with input color i.

 # Colors: Each color is of the form $(x \mod k, p(x) \mod q)$ for some polynomial p that is evaluated over $\mathbb{F}_q$. Thus the number of colors C can be upper bounded by $|x \mod k| \cdot |p(x) \mod q| \leq k \cdot q \leq k \cdot X$.

Proof of (1): When a vertex v is colored with a color $\varphi(v)$ in iteration j there are at most d other vertices that try to get the same color in this round or are already colored with this color. Since only edges to these vertices are oriented outwards from v the outdegree of v is bounded by d. Further, each edge between vertices with the same permanent color is oriented: Either they picked the color in the same iteration and the edge is oriented from the node with the smaller input color to the vertex with larger input color, or the edge is oriented outwards from the vertex that got permanently colored in a later iteration.

Proof of (2): Recall, that vertex v joins P_j if it is colored in iteration j of the while loop. As a vertex v only gets colored with a color φ in iteration j if there are at most d neighbors of v that want to get colored in iteration j the maximum degree of the graph induced by all vertices in P_j with color φ is at most d.

All vertices are colored in R iterations of the while loop: A node is only not colored in one iteration if for all of the k tuples, i.e., colors of the form $(x, p_i(x)) \in [k] \times [q]$, that it tries in that iteration there are strictly more than d neighbors that try the same tuple in the current iteration or already colored with the tuple.

Conflicts with an active node: Let us bound the number of entries in which two neighbors u, v with polynomials p_u and p_v, respectively, try the same tuple in some iteration $j \in [\lceil q/k \rceil]$ (conditioned on both nodes not being permanently colored yet). In iteration j the nodes simultaneously try all of the following tuples (where we omit the $j \cdot k \mod k = 0$ term in the first coordinate).

$$u \text{ tries: } \big(l, p_u(j \cdot k + l)\big) \text{ with } l \in [k] \text{ and} \tag{2}$$

$$v \text{ tries: } \big(l, p_v(j \cdot k + l)\big) \text{ with } l \in [k]. \tag{3}$$

Two tuples tried by u and v in iteration j can only cause a conflict, i.e, be the same, if they are the same in both coordinates. As all k tuples that are simultaneously tried by a node differ in the first coordinate, any conflict in iteration j between u and v implies that $p_u(j \cdot k + l) = p_v(j \cdot k + l)$ holds for some $l \in [k]$. Since p_u and p_v are polynomials of degree at most f, Lemma 2.1 implies that there are at most at most f combinations of j and l for which this holds.

Conflicts with an inactive node: Consider a neighbor u that has chosen some permanent color $(x_u, y_u) \in [k] \times [q]$. For v to try this tuple in iteration $j \in [q/k]$ we need

$$\big((j \cdot k + l) \mod k, p_v(j \cdot k + l) \mod q\big) = (x_u, y_u)$$

for some $l \in [k]$. This can only be the case if $p_v(j \cdot k + l)$ equals the fixed number y_u, which is the case for at most f different choices of j and l due to Lemma 2.1 (y_u is a polynomial of degree 0).

Thus, for fixed u and v, there are at most f tuples causing a *conflict* while u and v are active, and at most f tuples causing a *conflict* after (at least) one of the nodes has chosen a permanent color. A node v cannot get permanently colored with a tuple $\big(l, p_v(j \cdot k + l)\big)$ if there are strictly more than d conflicts for the tuple, i.e., strictly more than d neighbors try the same tuple in the same iteration or have already permanently adapted the color. In this case we call the tuple *blocked*. As each of the at most Δ neighbors contributes at most $2f$ such conflicts there can be at most $z = 2f\Delta/(d+1)$

blocked tuples. As the length q of the sequence (of tried tuples) is strictly larger than z, there is at least one tuple that is not blocked and each node is colored at the end of the algorithm.

During the execution of the algorithm all nodes have knowledge of m, q, f, d and k and all nodes can construct the set of polynomials P_q^f locally according to the same lexicographic order. Thus, for an uncolored node to send k trials in iteration j, it is sufficient to send its input color (together with k and j which are global knowledge). A node that gets colored can inform its neighbors about the choice in one round. All these steps can be executed in the CONGEST model. □

We remark, that it becomes significantly easier to read the algorithm if $m = \text{poly}\,\Delta$ and if $\Delta/d = \Delta^{\Omega(1)}$, as this implies $f = O(1)$, which simplifies many of the parameters. However, we chose to present the result in a more general form.

Remark 2.2. *With a tighter analysis for special cases one can reduce the constants in Corollary 1.2, e.g., in the case of $k = X$, the size of the field $\mathbb{F}_q$ can be chosen smaller. Due to such a tighter analysis and by assuming $m = \Delta^3$ the leading constant in the $O(\Delta^2)$-coloring by Linial is some $1 < \alpha < 10$ [42]. In contrast, the lower bound for the one-round algorithms from Section 3 only provides impossibilities below $\Delta^2/2 + \Theta(\Delta)$ colors, that is, for a constant $\alpha < 1$. Thus, there is a large regime for α where we neither have one-round upper bounds nor lower bounds. As even optimized constants in Theorem 1.1 and Corollary 1.2 leave a gap for the regime of α where lower bounds are known, we focus on having simple proofs that cover all cases of the theorem, instead of optimizing these constants.*

The condition $d = \beta = \Delta^{1-\Omega(1)}$ in Corollary 1.2: One would wish to use a variant of Corollary 1.2 to compute a $\Delta/2$-defective $O(1)$-coloring in one round, given a $O(\Delta^2)$-coloring. Note, that this setting would require the finite field over which we operate—the field size essentially determines the number of colors—to contain only $q = O(1)$ elements, and to obtain a distinct polynomial for each input color we would have to choose polynomials of degree $f = O(\log \Delta) \gg q$. However, then the proof of Theorem 1.1 breaks as we might have $O(f\Delta/d) = O(f) = \omega(1)$ blocked tuples while only having $q = O(1)$ tuples in the sequence. While slightly weaker requirements on d are possible without breaking the proof, our requirement ensures that we do not run into these issues. See [16, 37] for the parameter-heavy details on how to iterate the result of Theorem 1.1 for $O(\log^* \Delta)$ iterations to obtain defective a d-defective $O((\Delta/d))^2$ with no condition on d (essentially Corollary 1.2 (5) can also take a defective coloring as input coloring, and then defects add up).

We point out that the sequences required for Theorem 1.1 need not be constructed via polynomials. The proof only requires that the elements of the sequence are from a small enough domain, sequences are long enough, there is one sequence for each input color, and any two sequences intersect in few positions. In [43, arxiv version] such sequences are constructed greedily. Here, we chose to use a construction based on polynomials as the dependency on the m-input coloring is better, in particular, when $m = \text{poly}\,\Delta$, it implies that $f = O(1)$, instead of $f = O(\log \Delta)$ for the greedy-based construction. s

2.1 $(\Delta + 1)$-Coloring and $O(\Delta^{1+\varepsilon})$-Coloring

In this section, we begin with sketching (according to [43]) how the simple β-out degree coloring algorithm from Theorem 1.1 and Corollary 1.2 can be used in the state of the art $(\Delta + 1)$-vertex coloring algorithm in the LOCAL model (the CONGEST algorithm in [9, 15] profits from the same simplification). In the *list coloring problem* each node is given a list $L(v)$ of colors from some color space C. If the list of each node is of size at least $\deg(v) + 1$ the problem can be solved with a sequential greedy algorithm. If a bound β on the outdegree is given, then there exist a $(\beta + 1)$-coloring but it is not known how to compute such a coloring with an efficient distributed algorithm. A coloring with $(2+\varepsilon)\beta$ colors (with constant $\varepsilon > 0$) can be computed in $O(\beta \log n)$ rounds [11]. The following result by [43] shows that the list coloring problem can be solved in two rounds if the lists are large enough with respect to the out degree of a given orientation, the size of the color space and the number of colors in the input coloring.

Theorem 2.3 (Linial for Lists [43]). *In a directed graph with max. degree Δ, max. outdegree β, and an input m-coloring, list coloring with lists L_v from a color space C and of size $|L_v| \geq l = 4e\beta^2(4 \log \beta + \log\log |C| + \log\log m + 8)$ can be solved in 2 rounds in* LOCAL.

If one ignores the color space dependence, Theorem 2.3 has a close resemblance to Linial's color reduction generalized to the out degree setting. Given Theorem 2.3, we can easily sketch the state of the art for $(\Delta + 1)$-coloring in the LOCAL model: First, use [15] to compute a β-outdegree $O(\Delta/\beta)$-coloring ψ with $\beta = O(\sqrt{\Delta/\log \Delta})$. Then iterate through the coloring ψ and in each iteration apply Theorem 2.3 to all nodes who satisfies the theorems requirements, that is, whose list is of size $\Omega(\beta^2(\log \beta \ldots)) \geq \Delta/2$. Any node who does not satisfy this requirement has at most $\Delta/2$ uncolored neighbors. After each iteration, the lists of nodes are updated. Once the last color class of ψ has been processed the uncolored degree has halved for all uncolored nodes and one can recurse. As the runtime is polynomial in Δ, a geometric sum argument shows that the first recursion level dominates the total runtime.

Originally, this recursion scheme was developed in [25] and has been used in several works afterwards [7, 9, 15, 38, 43]. There are a few (simple) details that need to be taken care of for the whole algorithm to work: First, one needs to use Linial's algorithm to compute an initial $m = O(\Delta^2)$-coloring [42, 51], then one needs to adjust the m-coloring in each recursion level such that the m coloring and the uncolored degree do not drift apart too far, and then one needs a clean-up stage as soon as the uncolored degree and the color space size drift far apart, e.g., once the uncolored degree d is $O(\Delta^{1/4})$, one can simply list-color the remaining graph in $O(d^2) = O(\sqrt{\Delta})$ rounds, by computing an $O(d^2)$ coloring φ, iterate through the color classes of φ and let each vertex pick a color from its list that is not used by one of its already colored neighbors.

While our results do not improve the runtime to compute a $(\Delta + 1)$-coloring, we simplify the step of [15] of computing a β-outdegree $O(\Delta/\beta)$ coloring. Thus, afterall, the state of the art $(\Delta+1)$ coloring algorithms *only* consists of steps that are in the spirit of our mother algorithm: Linial's coloring algorithm is a special case of it, computing β-outdegree coloring is a special case of it, and

Theorem 2.3 is an extension of the setting $d = 0$ and $k = X$ to lists. We want to point out that unlike Linial's one-round color reduction, the list version of Theorem 2.3 uses two rounds, and as observed in [43], already an early lower bound by [51] showed that these two rounds are needed unless one wants to have an exponentially larger dependence on m in the lower bound on the list size. However,in aforementioned state of the art algorithm such a larger dependence on m would just appear as a $\log m = \Theta(\log \beta)$ dependence and would not hurt the recursion scheme. Thus, we are even far from understanding the difference between one-round and two-round coloring algorithms.

Also, the state of the art $(\Delta+1)$-coloring CONGEST algorithm works according to a similar scheme, but with a different choice of parameters [9, 15]: For $\beta = O(\Delta^{1/4})$ [9] develops a method to color all vertices in a single run through the $O(\Delta/\beta) = O(\Delta^{3/4})$ color classes of the β-out degree coloring, and further recursion is not needed. However, the approach only computes a coloring with $\Delta + O(\Delta^{3/4})$ colors and to obtain a $(\Delta+1)$-coloring the $O(\Delta^{3/4})$ excess colors need to be removed in an additional $O(\Delta^{3/4})$ rounds via a simple post processing procedure that eliminates one color class per round. If one aims for $O(\Delta)$ colors the approach of [9] works for the larger choice of $\beta = O(\sqrt{\Delta})$ without any further recursion. Using the improved (in comparison with the β-outdegree (aka arbdefective) coloring procedure in [9]) runtime for β-out degree coloring, either from Corollary 1.2 or from [15], into the framework of [9] one obtains the following theorem for CONGEST, as stated in [15].

Theorem 2.4 ([9, 15]). *There is a deterministic* CONGEST *algorithm to compute a $O(\Delta)$-coloring in $O(\sqrt{\Delta} + \log^* n)$ rounds.*

We use this algorithm and our defective coloring algorithm from Corollary 1.2 to improve the trade-off between the number of colors and the runtime from $O(k\Delta)$ vs $O(\Delta/k)$ (Corollary 1.2) to $O(k\Delta)$ vs $O(\sqrt{\Delta/k})$.

Theorem 1.3 ($O(k\Delta)$-coloring in $O(\sqrt{\Delta/k})$ rounds). *For any constant $\varepsilon > 0$, there is a deterministic* CONGEST *algorithm to compute a $O(\Delta^{1+\varepsilon})$-coloring in $O(\Delta^{1/2-\varepsilon/2}) + \log^* n$ rounds.*

Proof. First, compute an $O(\Delta^2)$-coloring in $\log^* n + O(1)$ rounds using Linial's algorithm [42]. Set $d = \Delta^{1-\varepsilon}$, then use Corollary 1.2 (part 6) to compute a d-defective coloring ψ with $O((\Delta/d)^2)$ colors in $O(\Delta/d) = O(\Delta^{\varepsilon})$ rounds. Then, on each color class in parallel compute a $O(d)$-coloring in $O(\sqrt{d}) = \Delta^{1/2-\varepsilon/2}$ rounds (the $\log^* n$ of Theorem 2.4 vanishes as we already have a $O(\Delta^2) = \operatorname{poly} d$-coloring) via Theorem 2.4 using a distinct color space for each color class of ψ, that is, each node v gets a color $\varphi(v)$ from this second step and the final output color of node v is set to be the tuple $(\psi(v), \varphi(v))$. In total we use $O((\Delta/d)^2 \cdot d) = O(\Delta^2/d) = O(\Delta^{1+\varepsilon})$ colors. □

While the above way is a simple way to prove Theorem 1.3 when using Theorem 2.4 as a blackbox, an alternative algorithm can be obtained by using the β-outdegree coloring result from Corollary 1.2 (with $\beta = \Delta^{1/2+\varepsilon/2}$) and carefully choosing the remaining parameters in the framework of [9].

2.2 Ruling Sets

A $(2,r)$-ruling set of a graph $G = (V, E)$ is a subset $S \subseteq V$ of the vertices that is an independent set and satisfies that for any vertex v in V there is a vertex $s \in S$ in hop distance at most r [1]. The following result uses colorings to compute a ruling sets.

Lemma 2.5 ([39, arxiv version]). *For any $B \geq 2$ there exists a deterministic distributed* CONGEST *algorithm that, given a C-coloring, computes a $(2, \lceil \log_B C \rceil)$-ruling set in $O(B \log_B C)$ rounds.*

We use Lemma 2.5 and Theorem 1.3 to compute $(2,r)$-ruling sets by adjusting the number of colors such that the runtime of computing the coloring and using it via Lemma 2.5 are balanced.

Theorem 1.4. *For any constant integer $r \geq 2$ a $(2,r)$-ruling set can be computed in $O(\Delta^{\frac{2}{r+2}}) + \log^* n$ rounds, deterministically in the* CONGEST *model.*

Proof. Set $C = O(\Delta^{2r/(r+2)}) = O(\Delta^{1+\frac{r-2}{r+2}})$. Then, we can compute a C-coloring of G in $O(\Delta^{1/2-\frac{r-2}{2r+4}}) + \log^* n$ rounds via Theorem 1.3. Now, set $B = O(C^{1/r})$ such that $\lceil \log_B C \rceil = r$ and apply Lemma 2.5 to compute a $(2,r)$-ruling set in $O(B \log_B C) = O(B) = O(C^{1/r})$ rounds. Ignoring the $\log^* n$ term, the total runtime is

$$O(\Delta^{1/2-\frac{r-2}{2r+4}} + C^{1/r}) = O(\Delta^{1/2-\frac{r-2}{2r+4}} + \Delta^{2/(r+2)}) = O(\Delta^{2/(r+2)}) .$$ □

Interestingly, for $r = 2$ the state of the art runtimes for $(2,r)$-ruling set is the same as the complexity for computing an $O(\Delta)$ coloring. Note that Theorem 1.4 does not hold for $r = 1$. It would yield a $O(\Delta^{2/3}) + \log^* n$ round complexity for computing a $(2,1)$-ruling set. $(2,1)$-ruling sets are better known under the name *maximal independent sets* for which a $\Omega(\Delta)$-round lower bound is known if the runtime's n-dependency is limited to $O(\log^* n)$ [3].

An (α, r)-*ruling set* is a subset $S \subseteq V$ that is an independent set in the power graph $G^{\alpha-1}$ that satisfies that each vertex $v \in V$ has a vertex in S in distance at most r. In the LOCAL model the results of Theorem 1.4 can be extended to (α, r)-ruling sets as one can simulate any algorithm on $G^{\alpha-1}$ in the original network graph. For details on these black-box extensions see, e.g., [17, 39].

3 ONE-ROUND COLOR REDUCTION

The objective of this section is to show the following theorem

Theorem 1.5. *For any integer $\Delta \geq 1$ and $\Delta + 1 \leq m \leq \Delta^2/4 + 3/2\Delta + 9/4$ let $1 \leq k \leq \min\{\Delta - 1, \Delta/2 + 3/2\}$ be the largest integer such that $m \geq k(\Delta - k + 3)$. Then, there is a one-round* CONGEST *algorithm that reduces an m-coloring to an $(m-k)$-coloring, but there is no one-round* LOCAL *algorithm to compute a $(m-k-1)$-coloring, given an m-coloring.*

In addition to the values for k that are stated in Section 1 ($k = 1, 2, 3, 4$) note that one requires $5\Delta - 10$ input colors to reduce 5 colors, and $6\Delta - 18$ input colors to reduce 6 colors. The proof is split into two lemmas, in Lemma 3.1 we provide a one-round color reduction algorithm, and in Lemma 3.2 we show that the algorithm is tight up to each single color. For a function $f : V \to [m]$ and a set $S \subseteq V$ we denote $f(S) = \{f(v) \mid v \in S\}$. For a node $v \in V$ of a given graph $G = (V, E)$ we denote the set of its neighbors by $N(v)$.

The following result reduces more colors than the one-round algorithms in [41, 51].

Lemma 3.1 (Color Reduction). *For an integer $1 \leq k \leq \Delta/2+3/2$ there is a one-round color reduction procedure from $m \geq k(\Delta-k+3)$ to $k(\Delta-k+2)$ (reduces k colors).*

The intuitive idea of the algorithm is that only vertices with one of the k largest color classes change their color. Each of these *recoloring colors* has its own small hardcoded output color regime from which it can pick a *free color*. However, the size of the regime is smaller than Δ and it might be that all of its colors are blocked by neighbors that do not recolor themselves. But this implies that there are some recoloring colors that do not appear in the node's neighborhood and it can steal colors from those recoloring color's regimes to gain the desired freedom.

Proof of Lemma 3.1. We may assume that $m = k(\Delta-k+3)$ is the number of input colors as for $m' \geq m$ input colors, one can leave $m'-m$ colors unchanged and apply the algorithm to the remaining m colors. Let $\ell = k(\Delta-k+2) \geq \Delta+1$ the number of (desired) output colors, $\varphi : V \to [m]$ the input coloring and $\psi : V \to [\ell]$ the to be computed output coloring.

Fix k disjoint color regimes of output colors $R_0, \ldots R_{k-1}$, each of size $\Delta-k+2$ as follows: Let $R_i(j) = i \cdot (\Delta-k+2)+j$ and $R_i = \{R_i(j) \mid j \in [\Delta-k+2]\}$.

Algorithm. Nodes perform the following mutual exclusive case distinction (decreasing priority from top to bottom):

- **Option 1:** Each node v with color $\varphi(v) \in [\ell]$ keeps its color, that is, it sets $\psi(v) = \varphi(v)$, else
- **Option 2:** Each nodes v with $\varphi(N(v)) \subseteq [\ell]$ sets $\psi(v)$ to be the smallest color in in $[\Delta+1] \setminus \varphi(N(v))$, else
- **Option 3:** Nodes v with color $\varphi(v) = i$ pick a free color in

$$\begin{aligned} F(v) =& R_i \cup \{R_j(i) \mid j \notin \varphi((N(v)), j > i\} \\ & \cup \{R_j(i-1) \mid j \notin \varphi((N(v)), j < i\}, \end{aligned}$$

that is, $\psi(v)$ is set to the smallest color in $F(v) \setminus \varphi(N(v))$.

In the first two options node v either does not change its color and neighbors ensure that the coloring is proper, or node v does not have a neighbor that recolors itself. In the latter case node v always has a color in $[\Delta+1]$ that is not among the colors $\varphi(N(v)) = \psi(N(v))$ that are used by its at most Δ neighbors.

So, let v be a node a node that uses the third option and let $d(v) < \Delta$ be the number of neighbors of v that do not recolor themselves. First note that the set $F(v)$ is well defined as all indices are in the range in which sets are defined, in particular we have $|R_j| = \Delta-k+2 \geq k-1$ because $k \leq \Delta/2+3/2$ holds. Further, by construction we have $F(v) \cap F(w) = \emptyset$ for any neighbor w that recolors itself. To ensure that v can select a free color in $F(v)$ we need to show that $|F(v)| \geq d(v)+1$ holds. Having $d(v) < \Delta$ neighbors that do not recolor themselves implies that node v has at most $\min\{k-1, \Delta-d(v)\}$ different input color in its neighborhood that recolor themselves ($k-1$ as the color of node v cannot appear in its neighborhood). We next lower bound the size of $F(v)$ in two cases.

Case $\Delta-d(v) \leq k-1$: For each of the at least $X = (k-1) - \min\{k-1, \Delta-d(v)\} = d(v)+k-\Delta-1$ input colors $j \in ([m] \setminus [\ell])$ that do not appear in the neighborhood of v node v steals a color from their set R_j, i.e, the set $F(v)$ of colors that v picks from contains at least X distinct colors of the form $R_j(i)$. Thus, the set $F(v)$ has size at least

$$|F(v)| \geq |R_j| + X \geq \Delta-k+2+d(v)+k-\Delta-1 = d(v)+1.$$

Case $\Delta-d(v) > k-1$: In this case, all colors in $[m] \setminus [\ell]$ might appear in the neighborhood of v, but the condition implies that $d(v) < \Delta-k+1$ holds and we obtain

$$|F(v)| \geq |R_j| = \Delta-k+2 > d(v)+1.$$

In both cases we obtain $|F(v)| \geq d(v))+1$. Thus v can pick a color in $F(v)$ not used by the $d(v)$ neighbors that do not recolor themselves, and as reasoned before there is no conflict with a neighbor that recolors itself. □

Next, we show that the result of Lemma 3.1 is tight. The next lemma can be seen as a generalization of a result in [35]), which proved a result of a similar flavor but only for $m \geq \Delta^2/4+\Delta/2+1$.

Lemma 3.2 (Lower Bound for One-Round Algorithms). *Let $1 \leq k \leq \Delta-1$ be an integer and let $G = (V, E)$ be a graph with maximum degree Δ and an input m-coloring. If $m \leq k(\Delta-k+3)-1$ then G cannot be colored with $q = m-k$ colors in a single communication round.*

Proof. We may assume that $m = k(\Delta-k+3)-1$, as an impossibility result for this choice of m implies the same result for any $m' \leq m$. Assume for contradiction that there is a one-round algorithm $\mathcal{A}$ that colors a graph with an input coloring with m colors with $q = m-k$ output colors. Then, among the m input colors there are at most q input colors $S = \varphi_1, \ldots, \varphi_q$ that are *strictly hardcoded* colored with one of the q output colors regardless of which 1-hop neighbors they have in the graph. So, let $T = \{\varphi_{q+1}, \ldots, \varphi_{q+k}\}$ be k of the input colors that do not have this property, i.e., for $\varphi \in T$ and for any output color $c \in q$ there is an input color φ' such that if a node v input colored with φ has a neighbor with input color φ' then node v does not output c.

Next, we construct a 1-hop neighborhood that cannot be colored with one of the q output colors and thus leads to a contradiction. For that purpose consider the partial neighborhoods $N_x = (x, T \setminus \{x\})$ with one node $x \in T$ in the center and the $k-1$ other nodes in $T \setminus \{x\}$ as 1-hop neighbors of x. We call a color $c \in [q]$ a *candidate color* for N_x if there exists some $B \subseteq [m]$ with $T \setminus \{x\} \subseteq B$ such that the algorithm $\mathcal{A}$ colors a node with input color x and neighbors with input colors B with c.

Claim 3.3. *For $x \neq x' \in T$ the sets of candidate colors of N_x and $N_{x'}$ are disjoint.*

Proof. Assume for contradiction, that $c \in [q]$ is a candidate color for N_x and $N_{x'}$ and let $B \supseteq T \setminus \{x\}$ and $B' \supseteq T \setminus \{x'\}$ be the respective sets such that $\mathcal{A}$ colors (x, B) and (x', B') with c. As $x \in B'$ and $x' \in B$ the neighborhoods (x, B) and (x', B') can occur next to each other in a graph, and $\mathcal{A}$ would not be correct, a contradiction. ■

By Claim 3.3 and by the pigeonhole principle there exists one $x_* \in T$ for which N_x has at most α candidate colors where

$$\begin{aligned}\alpha &= \left\lfloor \frac{q}{|T|} \right\rfloor = \left\lfloor \frac{q}{k} \right\rfloor = \left\lfloor \frac{m-k}{k} \right\rfloor = \left\lfloor \frac{k\Delta - k^2 + 2k - 1}{k} \right\rfloor \\ &= \left\lfloor \frac{k(\Delta - k + 2) - 1}{k} \right\rfloor = (\Delta - k + 2) - 1 \\ &= \Delta - (k-1).\end{aligned}$$

Now, let $C_* \subseteq [q]$ be the set of candidate colors of N_{x_*}. As x_* is in T, for each $c \in C_*$ there exists some $\varphi_c \in [m]$ such that $\mathcal{A}$ does not color a node v with input color x_* with color c if φ_c is the input color of one of v's neighbors. Hence, the 1-hop neighborhood $\tilde{N}_{x_*} = (x_*, \{\varphi_c \mid c \in C_*\} \cup (T \setminus \{x\}))$ cannot be colored by $\mathcal{A}$, a contradiction. The choice of parameters is important, as the $\tilde{N}_{x_*}$ is only a feasible neighborhood if $|\{\varphi_c \mid c \in C_*\} \cup (T \setminus \{x\})| \leq \alpha + k - 1 = \Delta$. □

Lemma 3.2 implies a heuristic lower bound of $\Omega(\Delta)$ to reduce a $\Delta^2/2$-coloring to a $\Delta^2/5$-coloring (if you have $\leq \Delta^2/4$ input colors you can remove at most $\Delta/2$ colors per iteration). In contrast, the algorithm from Corollary 1.2 (for a suitable choice of k) can reduce a Δ^4-coloring to a $\Delta^2/5$ coloring in $O(1)$ rounds. Thus, the iterative application of *tight* bounds for one-round algorithms can be beaten significantly by a simple $O(1)$-round algorithm. This suggests that it is important to understand constant-time algorithms to settle the complexity of distributed graph coloring problems.

4 CONCLUSION

In the current paper we have seen a simple algorithm for distributed graph coloring in which each vertex locally computes a permutation of the output colors and then *tries* them in batches. A a trial is *successful* if there is no *conflict*, that is, no neighbor tries the same color in the same round and no neighbor is already permanently colored with that color. Depending on the size of the batches, this algorithm scales between Linial's famous color reduction [42] and the locally iterative algorithm by Barenboim, Elkin and Goldenberg [15]. If nodes tolerate conflicts up to a certain threshold the same algorithm can be used to obtain the defective coloring algorithms of [16, 37] and [10, 16], as well as obtaining a simpler algorithm (as compared to [9, 15]) to compute low out degree colorings aka *arbdefective colorings*. The latter are one of the two crucial ingredients in the state-of-the art $(\Delta+1)$-coloring algorithm in [43]. The second ingredient is a 2-round *list version* of Linial's color reduction, together with the observation that the degree bound can be replaced with a bound on the out-degree. One can also see our algorithm as an extension of Linial's algorithm, or the other way around: In the setting where nodes can only try one color per round ($k = 1$) one wants to get colored with one out of $O(\Delta)$ output colors; this process is guaranteed to be successful in $O(\Delta)$ rounds if vertices try colors in a suitable order. Now, if you want to try more than one color per iteration, that is, you want to compress several rounds of the original algorithm into one iteration, you need to also mark each trial with the round number in which you would have tried it in the original algorithm, yielding an $O(\Delta^2)$-coloring if you want to execute all $O(\Delta)$ rounds in one iteration. We find it astonishing, that in hindsight many crucial results in this area can be related to the algorithm that was presented in Linial's seminal paper [42] roughly 30 years ago.

On the lower bound side his initial $\Omega(\log^* n)$ bound is still the state of the art. The only progress is in terms of understanding one-round algorithms or weak variants of the LOCAL model [35]. Our paper showed that there is a large discrepancy between iterating the best one-round algorithm for $O(1)$ times and what can be achieved by a *'smart'* algorithm that uses $O(1)$ rounds. This suggests that we first need to understand constant-time algorithms, both from the upper and lower bound side, before we can settle the complexity of the $(\Delta+1)$-coloring problem. Another approach would be attack the coloring problem through lower bounds for ruling sets, as there is recent progress for the latter [5]. A large lower bound for a $(2, r)$-ruling set would imply a lower bound for graph coloring via Lemma 2.5; however, it is unclear whether large enough lower bounds for ruling sets exist, as there currently is at least an exponential gap in the known lower bound and the complexity that would be needed.

We end with an additional (informal) observation that might be helpful to guide lower bound search to the correct questions.

Observation 4.1 (informal)**.** *Modulo a* $\log\Delta$*-factor the difficult part of the* $(\Delta+1)$*-coloring problem is to reduce a* $(1+\varepsilon)\Delta$ *coloring to a* $(\Delta+1)$*-coloring.*

Proof sketch. Assume an algorithm $\mathcal{A}$ that reduces the number of colors from $(1+\varepsilon)\Delta$ to $\Delta+1$. Now, assume an input coloring with $m \gg (1+\varepsilon\Delta)$ colors is given. Then, one can chop $[m]$ into $x \approx m/((1+\varepsilon)(\Delta+1))$ disjoint color spaces, each of size $(1+\varepsilon)\Delta$ and run $\mathcal{A}$ on each of them in parallel, using a disjoint output color space for each application. This uses $x \cdot (\Delta+1) \approx m/(1+\varepsilon)$ output colors, i.e., we have reduced the number of colors by a constant factor (if ε is constant). Thus, if we begin with $m = O(\Delta^2)$ colors, we obtain a $(\Delta+1)$-coloring with a $O(\log_{1+\varepsilon}\Delta)$ multiplicative overhead. □

If Δ is a large enough poly $\log n$, then randomized algorithm can very efficiently compute $(1+\varepsilon)\Delta$-colorings [22, 32, 50].

5 ACKNOWLEDGMENTS

We thank the several people with whom we have discussed the presented algorithm, in particular, Fabian Kuhn and Janosch Deurer.

REFERENCES

[1] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. 1989. Network Decomposition and Locality in Distributed Computation. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 364–369.

[2] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. 2020. Classification of Distributed Binary Labeling Problems. In *Proceedings of the International Symposium on Distributed Computing (DISC)*.

[3] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2019. Lower Bounds for Maximal Matchings and Maximal Independent Sets. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 481–497.

[4] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. 2020. Distributed Lower Bounds for Ruling Sets. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*.

[5] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. 2020. Distributed Lower Bounds for Ruling Sets. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. 365–376. https://doi.org/10.1109/FOCS46700.2020.00042

[6] Alkida Balliu, Juho Hirvonen, Christoph Lenzen, Dennis Olivetti, and Jukka Suomela. 2019. Locality of Not-so-Weak Coloring. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. 37–51.

[7] Alkida Balliu, Fabian Kuhn, and Dennis Olivetti. 2020. Distributed Edge Coloring in Time Quasi-Polylogarithmic in Delta. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*.

[8] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. 2020. Efficient Deterministic Distributed Coloring with Small Bandwidth. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*.

[9] Leonid Barenboim. 2016. Deterministic (Δ + 1)-Coloring in Sublinear (in Δ) Time in Static, Dynamic, and Faulty Networks. *Journal of the ACM* 63, 5 (2016), 47:1–47:22.

[10] Leonid Barenboim and Michael Elkin. 2009. Distributed (Delta+1)-Coloring in linear (in Delta) Time. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. 111–120. https://doi.org/10.1145/1536414.1536432

[11] Leonid Barenboim and Michael Elkin. 2010. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Comput.* 22, 5-6 (2010), 363–379.

[12] Leonid Barenboim and Michael Elkin. 2011. Deterministic Distributed Vertex Coloring in Polylogarithmic Time. *Journal of the ACM* 58, 5 (2011), 23:1–23:25.

[13] Leonid Barenboim and Michael Elkin. 2011. Deterministic Distributed Vertex Coloring in Polylogarithmic Time. *Journal of the ACM* 58, 5 (2011), 23:1–23:25.

[14] Leonid Barenboim and Michael Elkin. 2013. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers.

[15] Leonid Barenboim, Michael Elkin, and Uri Goldenberg. 2018. Locally-Iterative Distributed (Δ+ 1)-Coloring below Szegedy-Vishwanathan Barrier, and Applications to Self-Stabilization and to Restricted-Bandwidth Models. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 437–446.

[16] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. 2014. Distributed (Delta+1)-Coloring in Linear (in Delta) Time. *SIAM J. Comput.* 43, 1 (2014), 72–95.

[17] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The Locality of Distributed Symmetry Breaking. *Journal of the ACM* 63, 3 (2016), 20:1–20:45.

[18] Sebastian Brandt. 2019. An Automatic Speedup Theorem for Distributed Problems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 379–388.

[19] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. 2016. A lower bound for the distributed Lovász local lemma. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 479–488.

[20] Sebastian Brandt and Dennis Olivetti. 2020. Truly Tight-in-Δ Bounds for Bipartite Maximal Matching and Variants. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*.

[21] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2019. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. *SIAM J. Comput.* 48, 1 (2019), 122–143.

[22] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. 2020. Distributed (Δ + 1)-Coloring via Ultrafast Graph Shattering. *SIAM J. Comput.* 49, 3 (2020), 497–539.

[23] Michael Elkin and Shaked Matar. 2019. Near-Additive Spanners In Low Polynomial Deterministic CONGEST Time. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. 531–540. https://doi.org/10.1145/3293611.3331635

[24] Manuela Fischer. 2020. Improved deterministic distributed matching via rounding. *Distributed Comput.* 33, 3-4 (2020), 279–291. https://doi.org/10.1007/s00446-018-0344-4

[25] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. 2016. Local Conflict Coloring. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 625–634.

[26] Pierre Fraigniaud and Ami Paz. 2020. The Topology of Local Computing in Networks. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*. 128:1–128:18. https://doi.org/10.4230/LIPIcs.ICALP.2020.128

[27] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. 2021. Improved Deterministic Network Decomposition. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. arXiv:2007.08253 [cs.DS]

[28] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. 2018. On Derandomizing Local Distributed Algorithms. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 662–673.

[29] Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. 2018. Improved Distributed Delta-Coloring. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 427–436.

[30] Mohsen Ghaffari and Fabian Kuhn. 2020. Deterministic Distributed Vertex Coloring: Simpler, Faster, and without Network Decomposition. *CoRR* abs/2011.04511 (2020). arXiv:2011.04511 https://arxiv.org/abs/2011.04511

[31] Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. 2021. Efficient Randomized Distributed Coloring in CONGEST, In Proceedings of the ACM Symposium on Theory of Computing (STOC). *CoRR* abs/2012.14169. arXiv:2012.14169 https://arxiv.org/abs/2012.14169

[32] Magnús M. Halldórsson and Alexandre Nolin. 2021. Superfast Coloring in CONGEST via Efficient Color Sampling. In *28th International Colloquium on Structural Information and Communication Complexity*. arXiv:2102.04546

[33] David G. Harris. 2019. Distributed Local Approximation Algorithms for Maximum Matching in Graphs and Hypergraphs. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 700–724.

[34] David G. Harris, Johannes Schneider, and Hsin-Hao Su. 2018. Distributed (Δ +1)-Coloring in Sublogarithmic Rounds. *Journal of the ACM* 65, 4 (2018), 19:1–19:21.

[35] Dan Hefetz, Fabian Kuhn, Yannic Maus, and Angelika Steger. 2016. Polynomial Lower Bound for Distributed Graph Coloring in a Weak LOCAL Model. In *Proceedings of the International Symposium on Distributed Computing (DISC)*. 99–113.

[36] Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. 2011. Toward more localized local algorithms: removing assumptions concerning global knowledge. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*. 49–58. https://doi.org/10.1145/1993806.1993814

[37] Fabian Kuhn. 2009. Weak graph colorings: distributed algorithms and applications. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architecture (SPAA)*. 138–144.

[38] Fabian Kuhn. 2020. Faster Deterministic Distributed Coloring Through Recursive List Coloring. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1244–1259.

[39] Fabian Kuhn, Yannic Maus, and Simon Weidner. 2018. Deterministic Distributed Ruling Sets of Line Graphs. In *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*. 193–208. https://doi.org/10.1007/978-3-030-01325-7_19

[40] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2016. Local Computation: Lower and Upper Bounds. *Journal of the ACM* 63, 2 (2016).

[41] Fabian Kuhn and Roger Wattenhofer. 2006. On the complexity of distributed graph coloring. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. 7–15.

[42] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM J. Comput.* 21, 1 (1992), 193–201.

[43] Yannic Maus and Tigran Tonoyan. 2020. Local Conflict Coloring Revisited: Linial for Lists. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. 16:1–16:18. https://doi.org/10.4230/LIPIcs.DISC.2020.16

[44] Moni Naor. 1991. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM J. Discret. Math.* 4, 3 (1991), 409–412.

[45] Dennis Olivetti. 2020. Brief Announcement: Round eliminator: a tool for automatic speedup simulation. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*. 352–354. https://doi.org/10.1145/3382734.3405694

[46] Alessandro Panconesi and Aravind Srinivasan. 1992. Improved Distributed Algorithms for Coloring and Network Decomposition Problems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 581–592.

[47] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. SIAM.

[48] Václav Rozhon and Mohsen Ghaffari. 2020. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 350–363.

[49] Johannes Schneider, Michael Elkin, and Roger Wattenhofer. 2013. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theor. Comput. Sci.* 509 (2013), 40–50. https://doi.org/10.1016/j.tcs.2012.09.004

[50] Johannes Schneider and Roger Wattenhofer. 2010. A new technique for distributed symmetry breaking. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, Andréa W. Richa and Rachid Guerraoui (Eds.). ACM, 257–266. https://doi.org/10.1145/1835698.1835760

[51] Mario Szegedy and Sundar Vishwanathan. 1993. Locality based graph coloring. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 201–207.