Open University of Israel
Department of Mathematics and Computer Science

Locally-Iterative Distributed $(\Delta + 1)$-Coloring below Szegedy-Vishwanathan Barrier, and Applications to Self-Stabilization and to Restricted-Bandwidth Models

By
Uri Goldenberg

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

Prepared under the supervision of
Prof. Leonid Barenboim

December 21, 2020

Contents

List of Tables

List of Algorithms

Abstract

We consider graph coloring and related problems in the distributed message-passing model. *Locally-iterative algorithms* are especially important in this setting. These are algorithms in which each vertex decides about its next color only as a function of the current colors in its $1-hop-neighborhood$. In STOC'93 Szegedy and Vishwanathan showed that any locally-iterative $(\Delta+1)$-coloring algorithm requires $\Omega(\Delta \log \Delta + \log^* n)$ rounds, unless there exists "a very special type of coloring that can be very efficiently reduced" [42]. No such special coloring has been found since then. This led researchers to believe that Szegedy-Vishwanathan barrier is an inherent limitation for locally-iterative algorithms, and to explore other approaches to the coloring problem [4, 31, 2, 19]. The latter gave rise to faster algorithms, but their heavy machinery which is of non-locally-iterative nature made them far less suitable to various settings. In this thesis we obtain the aforementioned special type of coloring. Specifically, we devise a locally-iterative $(\Delta+1)$-coloring algorithm with running time $O(\Delta + \log^* n)$, i.e., *below* Szegedy-Vishwanathan barrier. This demonstrates that this barrier is not an inherent limitation for locally-iterative algorithms. As a result, we also achieve significant improvements for dynamic, self-stabilizing and bandwidth-restricted settings. This includes the following results.

- We obtain self-stabilizing distributed algorithms for $(\Delta+1)$-vertex-coloring, $(2\Delta-1)$-edge-coloring, maximal independent set and maximal matching with $O(\Delta + \log^* n)$ time. This significantly improves previously-known results that have $O(n)$ or larger running times [23].

- We devise a $(2\Delta-1)$-edge-coloring algorithm in the CONGEST model with $O(\Delta + \log^* n)$ time and $O(\Delta)$-edge-coloring in the Bit-Round model with $O(\Delta + \log n)$ time. The factors of $\log^* n$ and $\log n$ are unavoidable in the CONGEST and Bit-Round models, respectively. Previously-known algorithms had superlinear dependency on Δ for $(2\Delta-1)$-edge-coloring in these models.

- We obtain an arbdefective coloring algorithm with running time $O(\sqrt{\Delta}+\log^* n)$. Such a coloring is not necessarily proper, but has certain helpful properties. We employ it in order to compute a proper $(1+\epsilon)\Delta$-coloring within $O(\sqrt{\Delta}+\log^* n)$ time, and $(\Delta+1)$-coloring within $O(\sqrt{\Delta \log \Delta}\log^* \Delta + \log^* n)$ time. This improves the recent state-of-the-art bounds of Barenboim from PODC'15 [2] and Fraigniaud et al. from FOCS'16 [19] by polylogarithmic factors.

- Our algorithms are applicable to the SET-LOCAL model [25] (also known as the weak LOCAL model). In this model a relatively strong lower bound of $\Omega(\Delta^{1/3})$ is known for $(\Delta+1)$-coloring. However, most of the coloring algorithms do not work in this model. (In [25] only Linial's $O(\Delta^2)$-time algorithm and Kuhn-Wattenhofer $O(\Delta \log \Delta)$-time algorithms are shown to work in it.) We obtain the first linear-in-Δ algorithms that work also in this model.

1 Introduction

1.1 The Classical Model

Consider an n-vertex graph $G = (V, E)$ with maximum degree Δ whose vertices host processors. The vertices communicate with one another over the edges of G in *synchronous* rounds. The problem that we are studying is how many rounds (also known as *running time* in the *message-passing model of distributed computing*) are required for computing a proper[1] $(\Delta+1)$-coloring of G. This is one of the most fundamental and well-studied distributed symmetry-breaking problems [12, 21, 33, 42, 32, 5, 6, 7, 8, 10, 2, 19], and it has numerous applications to resource and channel allocation, scheduling, workload balancing, and to mutual exclusion [31, 23]. The study of distributed coloring algorithms on paths and cycles was initiated by Cole and Vishkin in 1986 [12], who devised a 3-coloring algorithm with $O(\log^* n)$ time[2]. The first distributed algorithm for the $(\Delta + 1)$-coloring problem on general graphs was devised by Goldberg and Plotkin in 1987 [21]. The running time of their algorithm is $2^{O(\Delta)}+O(\log^* n)$. ($\log^*$ is a very slow-growing function, defined formally in Section 2.) Goldberg, Plotkin and Shannon [22] improved this bound to $O(\Delta^2 + \log^* n)$. Linial [33] showed a lower bound of $\frac{1}{2}\log^* n - O(1)$. His lower bound applies to a more relaxed $f(\Delta)$-coloring problem, for any, possibly quickly-growing function $f()$. Linial also strengthened the upper bound of [22], and showed that an $O(\Delta^2)$-coloring can be computed in $\log^* n + O(1)$ time. (Via a standard color reduction, described e.g., in [7] Chapter 3, given an α-coloring one can compute a $(\Delta + 1)$-coloring in $\alpha - (\Delta + 1)$ rounds. Thus, Linial's algorithm also gives rise to $(\Delta + 1)$-coloring in $O(\Delta^2 + \log^* n)$ time.)

1.2 Locally iterative algorithms

In STOC'93, Szegedy and Vishwanathan [42] studied *locally-iterative* coloring algorithms. An algorithm $\mathcal{A}$ is an α-to-β locally-iterative, for a pair of parameters $\alpha > \beta$, if it maintains a sequence $\varphi_1, \varphi_2, ..., \varphi_T$ of *proper* α-colorings, where φ_i is the coloring on round i, for every $1 \leq i \leq T$, the coloring φ_T is a β-coloring, and T is the running time of the algorithm. On each round i, every vertex v computes its new color $\varphi_{i+1}(v)$ based only on the colors $\{\varphi_i(u) \mid u \in \hat{\Gamma}(v)\}$, where $\hat{\Gamma}(v) = \{v\} \cup \{u \in V \mid (u, v) \in E\}$ is the $1 - hop - neighborhood$ of v. Szegedy and Vishwanathan showed upper and lower bounds on the quantity $\Psi(n, \Delta, D)$, which is the number of colors into which an n-vertex graph G of maximum degree Δ can be properly recolored within one single round, assuming that it was properly D-colored in the beginning of the round. Note, however, that for the lower bound of [42] to hold, the proper D-coloring of G is assumed to be *arbitrary*. As a corollary of their upper bound on $\Psi(n, \Delta, D)$, Szegedy

[1] A coloring $\varphi : V \to [\Delta + 1]$ is called *proper*, if $\varphi(u) \neq \varphi(v)$, for every edge $e = (u, v) \in E$.

[2] Unless said otherwise, algorithms that we discuss are deterministic.

and Vishwanathan [42] derived an improved upper bound of $O(\Delta \log \Delta + \log^* n)$ for locally-iterative $(\Delta+1)$-coloring. Specifically, they devised an $O(\Delta^2)$-to-$(\Delta+1)$-locally-iterative algorithm with running time $O(\Delta \log \Delta)$. (This upper bound was later re-derived in a somewhat more explicit way by Kuhn and Wattenhofer [32].) As a corollary of their lower bound on $\Psi(n, \Delta, D)$, Szegedy and Vishwanathan [42] showed a *heuristic* lower bound on the number of rounds that a locally-iterative algorithm needs in order to compute a $(\Delta+1)$-coloring from an $O(\Delta^2)$-coloring. Their lower bound (Theorem 12 in [42], marked as "heuristic") is $\Omega(\Delta \log \Delta)$. By Linial's lower bound [33], $\frac{1}{2}\log^* n - O(1)$ rounds are required to compute an $O(\Delta^2)$-coloring. All $(\Delta+1)$-coloring algorithms developed before 2009 were locally iterative. (See Table 1 below for a summary of known locally-iterative algorithms.) In [4, 31] and independently Kuhn, devised an $O(\Delta + \log^* n)$-time $(\Delta+1)$-coloring algorithm, using *defective colorings*. (See Section 2 for the definition of this notion.) The algorithms of [4, 31, 8] are, however, not locally-iterative. This direction was further pursued by Barenboim in [2], who devised an algorithm with running time $\tilde{O}(\Delta^{3/4} + \log^* n)$, using *arbdefective colorings*. (See Section 2; the notion originates from [5].) This result was further improved by Fraigniaud at al. [19], who devised the current state-of-the-art $(\Delta+1)$-coloring with running time $O(\sqrt{\Delta}\log^{2.5}\Delta + \log^* n)$. The algorithms of [4, 31, 2, 19] are all not locally-iterative, as they all decompose the graph into many subgraphs, compute colorings for them, and carefully combine them into a single coloring for the original graph. In view of Szegedy-Vishwanathan's heuristic lower bound (henceforth, *SV barrier*), this seemed to be inevitable.

1.3 Our results

In the current thesis I show the first *locally-iterative* $(\Delta+1)$-coloring algorithm with running time $O(\Delta + \log^* n)$, i.e., *below the SV barrier* of $\Omega(\Delta \log \Delta + \log^* n)$. Unlike previously locally-iterative algorithms, our algorithm does not necessarily reduces the number of employed colors in every round. Instead, if the initial number of colors is Δ^2, it can keep being $\Omega(\Delta^2)$ for almost the entire execution of the algorithm, and then "suddenly" reduce to $\Delta+1$ in the last few rounds. The colorings $\varphi_1, \varphi_2, ..., \varphi_T$, $T = O(\Delta)$, that it computes on rounds $1, 2, ..., T$, respectively, are all proper, but they are *not at all arbitrary*. Rather they have some special properties that guarantee that in $O(\Delta)$ rounds the number of colors reduces to $(\Delta+1)$.

Interestingly, in their seminar paper [42], Szegedy and Vishwanathan mention a possibility of such a phenomenon. In the preamble to their aforementioned "heuristic" theorem (Theorem 12) they wrote:

"There is a possibility, however, that after a few steps of iteration we arrive at a very special type of coloring that can be very efficiently reduced in steps thereafter. Assuming that this does not happen, the results of the previous section give the following theorem:
Theorem 12 (heuristic): Let $1 \leq b < a \leq \Delta/2$. *To decrease the number of colors from* $a\Delta$ *to* $b\Delta$ *it takes*

$\Theta(\Delta \log(a/b))$ *steps. In particular, to decrease the number of colors from* $\Delta^2/2$ *to* Δ *requires* $\Theta(\Delta \log \Delta)$ *steps."*[1]

We also use our new locally iterative technique to devise improved *not* locally-iterative coloring algorithms. Specifically, we obtain $(1+\epsilon)\Delta$-coloring within $O(\sqrt{\Delta} + \log^* n)$ time, for an arbitrarily small constant $\epsilon > 0$, and a $(\Delta+1)$-coloring within $O(\sqrt{\Delta \log \Delta} \log^* \Delta + \log^* n)$ time. This improves the best previously-known running time $O(\sqrt{\Delta} \log^{2.5} \Delta + \log^* n)$ of Fraigniaud et al. [19].

Running time	Reference
$2^{O(\Delta)} + O(\log^* n)$	Goldberg, Plotkin [21]
$O(\Delta^2) + \log^* n$	Linial [33]
$O(\Delta) \cdot \log n$	Goldberg at el. [22]
$O(\Delta^2) + \log^* n$	Goldberg et al. [22]
$O(\Delta \log \Delta) + \frac{1}{2} \log^* n$	Szegedy, Vishwanathan [42]
$O(\Delta \log \Delta) + \log^* n$	Kuhn, Wattenhofer [32]
$O(\Delta) + \log^* n$	**This thesis**

Table 1: Known results for locally-iterative $(\Delta+1)$-coloring.

1.4 Applications

In the Conclusions section of the paper [32] by Kuhn and Wattenhofer, the authors explain why locally-iterative algorithms are particularly important from practical perspective. They mention "emerging dynamic and mobile distributed systems such as peer-to-peer, ad-hoc, or sensor networks" as examples of networks for which such algorithms can be especially suitable. They also point out that locally-iterative algorithms are typically communication-efficient ones.

In this thesis I demonstrate that our novel locally-iterative algorithms indeed provide dramatically improved bounds for both the dynamic Self-Stabilizing scenarios and for scenarios in which communication-efficiency is crucial. In the next three subsections I discuss these applications of our locally-iterative technique one after another.

[1] The argument of [42] applies, in fact, to reducing the number of colors to $\Delta+1$, as opposed to Δ.

1.5 Tight bounds

Very recently, Balliu, Brandt, Hirvonen, Olivetti, Rabie And Suomela proved in [3] that MIS (Maximal independent set) and MM (Maximal matching) require $\Omega(\Delta + \log^* n)$ rounds in general graphs. Together with the results of the current work and [8], several tight bounds are obtained. Specifically, in LOCAL and CONGEST models the running time of MIS and maximal matching is $\Theta(\Delta + \log^* n)$, However, in one bit model our technique still requires $O(\Delta + \log n)$ rounds when using deterministic methods. Nevertheless, as explained in [35], the factor of $O(\log n)$ is unavoidable in the case of one-round MIS algorithms.

1.6 Self-Stabilizing Symmetry Breaking

The Self-Stabilizing setting was introduced by Dijkstra [13], and is being intensively studied since then. See, e.g., Dolev's monograph [14] and surveys by Herman [26], by Guelleti and Kheddouci [23]. Self-stabilization in dynamic systems was defined in [15]. In the context of $(\Delta + 1)$-coloring, the setting we consider is the following one. Every vertex v of a graph $G = (V, E)$ of maximum degree at most Δ and at most n vertices has a unique ID number. The memory of each vertex consists of two areas. The *Read Only Memory* (henceforth, ROM) consists of hard-wired data such as vertex ID, degree bound Δ, vertices bound n, and program code. The ROM is faultless, but its contents cannot be changed during execution. The other area of the memory is *Random Access Memory* (henceforth, RAM). This memory may change during execution, and it is appropriate for storing variables, such as vertex colors. However, this memory area may change not only as a result of an algorithm instruction, but also as a result of faults. Such faults may make arbitrary and completely unpredictable changes in any round in the entire RAM. Moreover, in the *Fully-Dynamic Self-Stabilizing setting*, in each round vertices may crash, new vertices may appear and communication links between vertices may change arbitrarily, as long as the bounds on n and Δ hold[1]. For example, colors are stored in RAM, and as long as faults occur, vertices may hold arbitrary colors, possibly the same as those of their neighbors, no matter what operations are performed by an algorithm. The objective is to devise algorithms in which once faults stop occurring, the algorithm *self-stabilizes* quickly to a proper solution. The relevant notion of running time in this context is called *stabilization time* (also known as "quiescence" time), which is the maximum number T of rounds, so that T rounds after the last fault or dynamic change of the graph we are guaranteed that an algorithm arrives to a proper solution, e.g., the coloring of the graph is a proper $(\Delta + 1)$-coloring. One can define analogously self-stabilizing variants of $(2\Delta - 1)$-edge-coloring (see Section 1.2.2), of Maximal

[1] In fact, since the dependence of our algorithms' running time on n is just $\log^* n$, the bound for the number of vertices may be double- or triple-exponential in the real number of vertices, and still the running time will be affected by just an additive constant term.

Independent Set (henceforth, MIS) and of Maximal Matching (henceforth, MM)[2].

Self-stabilizing symmetry-breaking problems were extensively studied [27, 28, 29, 41]. See also [23] for an excellent survey of self-stabilizing symmetry-breaking algorithms. However, all of them have prohibitively large stabilization time of $O(n)$ or more. In this thesis I devise the first self-stabilizing algorithms with stabilization time of $O(\Delta + \log^* n)$ for all these four fundamental problems. We note that the fact that our algorithms are *deterministic* is particularly useful in this setting. Indeed, this prevents the possibility that adversarial faults will manipulate random bits of the algorithm.

1.7 Edge-Coloring

Another classical and extremely well-studied symmetry breaking problem is that of $(2\Delta-1)$-edge-coloring [38, 6, 9, 10, 17, 16, 20, 18, 39]. An *edge-coloring* φ of a graph $G = (V, E)$ is a function $\varphi : E \to N$. It is said to be *proper* if for every pair of incident edges $e, e' \in E$, $e \neq e'$, we have $\varphi(e) \neq \varphi(e')$. The classical theorem of Vizing [43] states that every graph is $(\Delta + 1)$-edge-colorable. However, existing distributed deterministic solutions [38, 6, 9, 10, 17, 18] employ $(2\Delta - 1)$ colors or more in general graphs. (There are efficient randomized distributed algorithms [10, 17] that compute $(1 + \epsilon)\Delta$-edge-colorings in time close to $(\log n)/\Delta^{1-o(1)}$. This running time is incomparable to running time of the form $f(\Delta) + O(\log^* n)$, for some function $f()$, achieved by deterministic algorithms that we discuss here.) The first efficient deterministic algorithm for $(2\Delta - 1)$-edge-coloring was devised by Panconesi and Rizzi [38]. Its running time is $O(\Delta + \log^* n)$.

In the LOCAL model of distributed computing, messages of arbitrary size are allowed. The $(2\Delta - 1)$-edge-coloring problem for a graph G reduces to $(\Delta + 1)$-vertex-coloring problem for the line graph $L(G)$ of G, and in the LOCAL model this reduction can be implemented without any overhead in running time. Therefore, the novel sublinear-in-Δ time algorithms for $(\Delta+1)$-vertex-coloring [2, 19] immediately give rise to sublinear-in-Δ time algorithms for $(2\Delta - 1)$-edge-coloring. However, all these edge-coloring algorithms [38, 2, 25] are not locally iterative. Moreover, they do not apply (or require significantly more time) in the CONGEST model of distributed computing. In the latter model, every vertex v is allowed to send $O(\log n)$ bits of information to each of its neighbors in every round. Implementing Panconesi-Rizzi algorithm in the CONGEST model requires $O(\Delta^2 + \log^* n)$ time. Simulating vertex-coloring for a line graph also yields a multiplicative overhead of factor at least Δ in the running time. Therefore, to the best of our understanding, the state-of-the-art solution for $(2\Delta - 1)$-edge-coloring in the CONGEST model requires $\tilde{O}(\Delta^{3/2} + \log^* n)$ time, and it is not locally iterative. The best currently-known locally-iterative solution is even slower, and requires $O(\Delta^2 \log \Delta + \log^* n)$ time. (It is achieved by simulating the locally-

[2] A subset $U \subseteq V$ of vertices is an *MIS* if there are no edges between pairs of vertices in U, and for every vertex $v \in V \setminus U$, there exists a neighbor $u \in U$. A subset $M \subseteq E$ of edges is an *MM* if no tow edges of M are incident, and for every $e \in E \setminus M$, there exists an edge $e' \in M$ incident on it.

iterative $O(\Delta \log \Delta)$-time algorithm of [32, 42] in the line graph in the CONGEST model.) The problem of devising communication-efficient algorithms for symmetry-breaking problems was raised in a recent work by Pai et al. [37].

We adapt our locally-iterative algorithm for $(\Delta+1)$-vertex-coloring to work for $(2\Delta-1)$-edge-coloring directly, i.e., without simulation of the line graph. As a result we obtain a locally-iterative $(2\Delta-1)$-edge-coloring algorithm with running time $O(\Delta+\log^* n)$ in the CONGEST model. Moreover, we show that unlike previous solutions (that require stabilization time of $\Omega(n)$), our algorithm works in the self-stabilizing setting, still with small messages, with stabilization time $O(\Delta + \log^* n)$. Moreover, our algorithm is also applicable to the more restricted Bit-Round [30] model in which each vertex is only allowed to send 1 bit in each round over each edge.

1.8 SET-LOCAL Model

An additional application of our algorithms is in the SET-LOCAL model [25] that represents restricted networks in which vertices do not have IDs (but start from a proper coloring), and are not capable to distinguish between identical messages received from different neighbors. Since our algorithms are locally-iterative and compute the next colors based only on sets of current colors of 1-hop-neighborhoods, our algorithms are directly applicable to the SET-LOCAL model. Thus our algorithms compute proper $(\Delta + 1)$-coloring (and solve related problems) in $O(\Delta)$ time in the SET-LOCAL model starting from a proper $O(\Delta^2)$ coloring. The best previous algorithms in this model required $O(\Delta \log \Delta)$ time [42, 32, 25]. A lower bound of $\Omega(\Delta^{1/3})$ for $(\Delta + 1)$-coloring in this setting was obtained by Hefetz et al. [25].

1.9 Summary

We believe that these applications demonstrate the power of locally-iterative coloring. Bypassing Szegedy-Vishwanathan barrier via a locally-iterative algorithm does not only provide a surprising answer to a quarter-century-old open problem, but also provides new precious insights into distributed coloring in general. We are confident that these insights will be instrumental in achieving further breakthroughs in this important field.

1.10 Technical Overview

We start with describing our most basic subroutine, which we call *Additive Group algorithm*, or shortly, AG algorithm. The subroutine starts with a proper $(\Delta+1)^2$-vertex-coloring φ of the input graph G, and

produces its proper $(\Delta+1)$-coloring in $O(\Delta)$ rounds, in a locally-iterative way. Assume (for simplicity of presentation) that $\Delta+1=p$ is a prime number. We represent every initial color $\varphi(v)=\varphi_0(v)$ as a pair $\langle a_v, b_v\rangle$, where a_v, b_v are from the field of integers with characteristic p, i.e., $a_v, b_v \in GF(p)$. Then every vertex $v \in V$ (in parallel) checks if there exists a neighbor $u \in \Gamma(v)$, with $b_u = b_v$. If there is no such a neighbor, then the vertex v *finalizes* its color, i.e., sets it to $\langle 0, b_v\rangle$. Otherwise, the vertex v sets its color to $\langle a_v, b_v + a_v\rangle$, where the addition is performed in $GF(p)$. We show (see Section 3) that when all vertices run this simple iterative step for $2p+1 = 2(\Delta+1)+1$ rounds, the ultimate coloring ψ is a proper $(\Delta+1)$-coloring. Moreover, at all times the graph is properly colored.

The simplicity and the uniformity of this iterative step makes it very powerful. In dynamic self-stabilizing environments vertices run this step forever in conjunction with an appropriate "check-and-fix" procedure, no matter what changes or faults occur in the network. It turns out that still, once faults stop occurring, within additional $O(\Delta)$ rounds the coloring converges to a proper $(\Delta+1)$-coloring. In the edge-coloring scenario, every edge $e=(u,v)$ has a color $\varphi(e)=\langle a_e, b_e\rangle$, known to both endpoints. The endpoint u checks locally if there is an edge e_u incident on u, $e_u \neq e$, with $b_{e_u} = b_e$, and v makes an analogous test among edges incident on it. Then u and v communicate to one another *one single bit* each, which enables both of them to update the color of e. Therefore, this algorithm gives rise to the first communication- and time-efficient $(2\Delta-1)$-edge-coloring algorithm. Moreover, this algorithm is extremely well suited to dynamic and self-stabilizing scenarios.

Some subtleties arise when $(\Delta+1)$ is not prime, and we overcome them by showing that in some cases the proof goes through even if the arithmetics are performed in an additive group $Z_{\Delta+1}$, rather than in a Galois field $GF(p)$. Another difficulty stems from the need to combine the AG algorithm with Linial's algorithm. The latter algorithm reduces the number of colors to $O(\Delta^2)$, and from there the AG algorithm takes over. However, in the self-stabilizing setting some vertices may run Linial's algorithm, while others have already proceeded to AG algorithm. Careful adaptations to both algorithms are required to handle such situations.

Finally, we also extend the AG algorithm to computing *arbdefective coloring*. For a pair of parameters α and β, a coloring φ is said to be α-*arbdefective* β-*coloring* if the β color classes of G induce subgraphs of arboricity at most α each. Arbdefective colorings were introduced by the first- and the second-named authors in [5], and they were shown to be extremely useful for efficient computation of proper colorings in [5, 2, 19]. Our extension of AG algorithm from proper to arbdefective colorings (we call the extended algorithm *ArbAG*) works very similarly to the AG algorithm. The only difference is that on each round, each vertex v tests if it has at most a certain number of neighbors u with $b_u = b_v$. (Recall that in AG algorithm, this threshold number is 0.) Other than that ArbAG has the same simple locally-iterative structure as algorithm AG, but the number of iterations of ArbAG is significantly smaller. (Note,

however, that strictly speaking, a locally iterative algorithm is required to maintain a proper coloring on each round, while algorithm ArbAG maintains an arbdefective coloring.) This is in sharp contrast to previous methods [5, 2] of computing arbdefective colorings. The latter are far more involved, far less communication-efficient, and less time-efficient by polylogarithmic factors. As a result we also obtain improved (again, by polylogarithmic factors) algorithms for general (not necessarily locally-iterative) $(\Delta+1)$-coloring and $(1+\epsilon)\Delta$-coloring.

2 Preliminaries

2.1 Definitions

The function $\log^* n$

The function $\log^* n$ is the number of times the $\log_2$ function has to be applied iteratively starting from n, until we arrive at a number smaller than 2.

Vertex ID

The unique identity number (ID) of a vertex v in a graph G is denoted $id(v)$.

Graph diameter

The diameter $Diam(G)$ of a graph $G=(V,E)$ is the maximum (unweighted) distance between vertices $u,v \in V$.

Arboricity

The *arboricity* $a=a(G)$ of a graph $G=(V,E)$ is the minimum number of forests into which the edge set E can be partitioned.

Defective coloring

A *d-defective p-coloring* is a vertex coloring using p colors such that each vertex has at most d neighbors colored by its color.

Arbdefective coloring

A *b-arbdefective p-coloring* is a vertex coloring using p colors, such that each subgraph induced by vertices of the same color has arboricity at most b.

2.2 The algorithm of Cole and Vishkin [12]

One of the simplest configurations in the distributed setting is an oriented tree. An oriented tree $T=(V,E)$ is rooted at a vertex $r \in V$, and every vertex $v \in V$ knows the identity of its parent $\pi(v)$ in the rooted tree (T,r). On the other hand, in an unoriented tree there is no distinguished root, and there is no parent-child relationship between neighbors. Notice that in oriented trees each vertex has information that allows it to orient the edges adjacent on it towards the parents. (For each edge (u,v)

exactly one of the endpoints is the parent of the other one. Thus each edge can be oriented towards the parent endpoint.) Consequently, an acyclic orientation of out-degree at most 1 is obtained. Therefore, an oriented tree can be 2-colored using BFS algorithm starting from r where each round switches the color. Unoriented trees can be 2-colored as well, since trees are bipartite. However, 2-coloring a tree (even an oriented one) in the distributed model requires $\Theta(n)$ rounds [33]. Moreover, if the tree is unoriented, then even with any constant number of colors one still needs at least $\Theta(\log n)$ rounds to color a tree [33]. However, an oriented tree can be 3-colored within just $\log^* n + O(1)$ rounds. This is a fundamental result by Cole and Vishkin [12], and Goldberg, Plotkin and Shannon [22]. We start with describing a 6-coloring algorithm for oriented trees. It will be later refined to a 3-coloring one. Initially, each vertex v has an identity number $Id(v)$ from the set $[1..n]$, where n is the number of vertices. It initializes its color $\phi(v)$ to be equal to its identity number $Id(v)$. Denote by $|\phi(v)|$ the number of bits used to represent the color $\phi(v)$ of v. The algorithm works iteratively. In each iteration each vertex $v \neq r$ compares the bit string $\phi(v)$ which represents its current color with the bit string $\phi_\pi(v)$ which represents the color of its parent. It finds an index i such that $\phi(v)[i] \neq \phi(\pi)(v)[i]$, and sets $\phi'(v) =< i, \phi(v)[i] >$. Specifically, $\phi'(v)$ is the new color of the vertex v, and it consists of two fields. The first field contains the binary representation of the bit string i, and the second field contains the single bit $\phi(v)[i]$. The color $\phi'(v)$ is the concatenation of these two fields. The root r of the tree T picks an arbitrary index i and sets $\phi'(r) =< i, \phi(r)[i] >$. The algorithm is executed for $\log^* n$ iterations. (For simplicity we assume that all vertices know the value of n. However, this assumption can be omitted using a slightly more delicate argument.)

Lemma 2.1. *Given a proper coloring ϕ, the resulting coloring ϕ' is proper as well.*

Proof. Consider an edge $(v, u) \in E$, and suppose without loss of generality that $u = \pi(v)$. By the assumption of the lemma, $\phi(v) \neq \phi(u)$. Let $i(v)$ (respectively, $i(u)$) be the index selected by v (respectively, u). If $i(v) \neq i(u)$ then $\phi'(v) \neq \phi'(u)$ by the difference of the first index. Otherwise, if $i(v) = i(u)$ then $\phi'(v) \neq \phi'(u)$, since the index i is selected in such a way that $\phi(v)[i] \neq \phi(u)[i]$. Thus the values of the i-th bits of the new colors $\phi'(v), \phi'(u)$ are different one from another. □

Lemma 2.2. *The algorithm above can produce a proper coloring using at most 6 colors*

Proof. Note that in each iteration if the number of colors used by ϕ is α then number of colors used by ϕ' is $2 \cdot \lceil \log(\alpha) \rceil$. This iterative process can proceed until $\alpha = 6$ because $2 \cdot \lceil \log(6) \rceil = 6$. □

2.3 Shift down

The number of colors is further reduced to 3 by a different technique, called the shift-down. This phase of the algorithm requires 3 additional iterations, with $O(1)$ rounds each. (Generally, it can be used to

reduce the number of colors from α to 3 within α - 3 rounds).

In each iteration the number of colors is reduced by 1 within two steps.

In the first step each vertex $v \neq r$ adopts the color $\phi(\pi(v))$ of its parent $\pi(v)$. The root changes its color to an arbitrary color from $\{1, 2, 3\}$, different from the color it used to have. After this phase all siblings nodes have the same color.

The second step is to choose a proper coloring ϕ' where each node with color α chooses colors from $\{1, 2, 3\}$ different from the color of its children and the color of its parent.

2.4 Linial's coloring using $O(\Delta^2)$ colors, within $O(\log^* n)$ rounds [33]

Linial suggested the state of the art algorithm reducing a proper coloring ϕ using n colors into $O(\Delta^2)$ coloring within $O(\log^* n)$ rounds. The algorithm employs an algebric technique that will be described next. Denote $X = GF(q) \cdot GF(q)$, representing a 2-dimentional discreet graph with size of q for each axis. Let $|X| = m$, be a ground-set, where $m = q^2$ for a prime number q. We consider the Galois field $GF(q)$ of q^2. For any positive parameter d, let $Poly(d, q)$ the set of polynomials of degree d over $GF(q)$. For each polynomial $g() \in Poly(d, q)$, let $S_g = \{S_g | g() \in Poly(d, q)\}$, representing the points on X over $Poly(d, q)$. Let $F_{d,q} = \{S_g | g() \in Poly(d, q)\}$, reperesenting the family of possible polynomials in $Poly(d, q)$. Observe that $|S_g| = q$. for every $g() \in Poly(d, q)$. Note that two such distinct sets may intersect in at most d elements (This is because every polynomial can be represented in a form of $g() = (x - i_1) \cdot ... \cdot (x - x_{i_d})$). Hence to cover a fixed set S_g, one needs at least q/d other sets S_h from the family $F_{d,q}$. Let $\Delta \leq \lceil (q/d) - 1 \rceil < q/d$. It follows that $F_{d,q}$ is a Δ cover free family, which means that for every set in $F_{d,q}$ there is always an item not covered by another set of $F_{d,q}$ as long as $|F_{d,q}| \leq \Delta$. Its cardinally is $|F| = q^{d+1} \geq n$. If we express q and d by terms of n - (number of nodes) and m - (ground set size) and Δ. We obtain $m \cdot log^2 m \leq 4 \cdot (\Delta + 1)^2 \cdot \log^2 n$. hence, $m \leq 4 \cdot (\Delta + 1)^2 \cdot \log^2 n$. Now that we have a ground set m and an upper bound for the 1st round, it follows that $m = O(\Delta^2 \cdot \log^2 m)$. Next consider the next round. Let us mark $|X| = m'$ where m' represents the number of colors after the 1st round reduction. That is $|X| = O(\Delta^2 \cdot \log^2 m)$. Then we need to choose a polynomial constructed by the new $q' < 2 \cdot |X|$. Repeating the idea $\log^* n$ times will produce an $O(\Delta^3)$ coloring. By using appropriate parameters for the case $d = 2$ we get $n = q^3$, $m = q^2$. This reduces the number of colors to $O(\Delta^2)$ as expected.

3 Additive-Group Coloring

3.1 Basic additive-group coloring

In this section we present our main algorithm that computes a proper $O(\sqrt{k})$-coloring from a proper k-coloring, where $k = \Omega(\Delta^2)$. Consider a graph $G = (V, E)$ with a proper k-coloring ψ. For all vertices $v \in V$, we represent a color $\psi(v) = i$ by a pair $\langle a_v, b_v \rangle$. We do it by finding a prime number $q, \sqrt{k} \leq q \leq 2\sqrt{k}$. The color $\psi(v) = i$ is represented by the following pair $\psi(v) = \langle \lfloor i/q \rfloor, i \bmod q \rangle$. Our final goal is to eliminate the first coordinate, i.e., to change all nodes colors such that for every vertex $v \in V$, it will hold that $\psi(v) = \langle 0, b_v \rangle$, $0 \leq b_v < q$, and ψ is a proper q-coloring. Our algorithm proceeds in iterations, starting from the initial coloring ψ. In each iteration colors may change, but the coloring remains proper. We employ the following definition.

Definition 3.1. *Two neighbors u, v in G* conflict *with one another if and only if $\psi(v) = \langle a, b \rangle$ and $\psi(u) = \langle a', b \rangle$, where $0 \leq a, b, a' < q$.*

Denote $\psi(v) = \langle a, b \rangle$. We will refer to a as the first coordinate and to b as the second coordinate. Denote by $\psi_i(v)$ the color of $v \in V$ in round i. Our algorithm starts from a proper $k = \Omega(\Delta^2)$ coloring of the input graph $G = (V, E)$. In each round the algorithm performs the following step. For all $v \in V$ in parallel, if a node v conflicts with a neighboring node u, then the new color of v in the end of this round is $\psi_{i+1}(v) = \langle a, (b + a) \bmod q \rangle$. Otherwise (this means v does not conflict with any neighbor), we set $\psi_{i+1}(v) = \langle 0, b \rangle$, and the color of v becomes final and will not change anymore.[1] This completes the description of the algorithm. Note that a node does not have to send its new color to all of its neighbors. Rather it is enough to send only one bit indicating whether its color became final or that it changed according to the rule specified above. We will use this property later. Next, we prove correctness.

[1] Note, however, that a *finalized* vertex v, i.e., a vertex with $\psi_i(v) = \langle 0, b \rangle$, can keep running the same iterative step, and still its colors will stay unchanged.

Lemma 3.2. *For each iteration i, the coloring $\psi_i(G)$ is proper.*

Proof. The proof is by induction on i. **Base:** $(i = 0)$: holds trivially, since the initial coloring is proper. **Step:** Assuming that in iteration i the coloring is proper, we prove that in iteration $i + 1$ it is proper as well. If a color of a node $v \in V$ is $\psi_i(v) = \langle a, b\rangle$, then for the next iteration the color is either $\psi_{i+1}(v) = \langle 0, b\rangle$ or $\psi_{i+1}(v) = \langle a, (b + a) \bmod q\rangle$. Consider an adjacent node u, i.e., $(u, v) \in E$. If $\psi_i(u) = \langle c, b\rangle$, where $0 \leq c < q$, then $c \neq a$, by the induction hypothesis. In this case, the new colors of the nodes will be $\psi_{i+1}(v) = \langle a, (b+a) \bmod q\rangle$ and $\psi_{i+1}(u) = \langle c, (b+c) \bmod q\rangle$ and since $c \neq a$ this means that the new colors of u and v are distinct. Otherwise, $\psi_i(u) = \langle c, d\rangle$, where $d \neq b$. If in iteration $i + 1$ it holds that $\psi_{i+1}(v) = \langle 0, b\rangle$ and $\psi_{i+1}(u) = \langle 0, d\rangle$, we are done since $b \neq d$. Otherwise, u or v had conflicts in iteration i. If exactly one of them had a conflict, then their colors in iteration $i + 1$ are distinct. (One of them has 0 in the first coordinate, while the other has not, in iteration $i + 1$.) It is left to consider the case that both had conflicts. Thus, $\psi_{i+1}(v) = \langle a, (b + a) \bmod q\rangle$ and $\psi_{i+1}(u) = \langle c, (d + c) \bmod q\rangle$. If $a \neq c$, we are done. Otherwise, $a = c$ and $b \neq d$, because ψ_i is proper. Thus, $b + a \not\equiv d + c \pmod q$, and $\psi_{i+1}(v) \neq \psi_{i+1}(u)$. □

We say that a vertex is in a *working* stage as long as its color $\langle a, b\rangle$ satisfies $a \neq 0$. Once a becomes 0, the vertex is in the *final* stage. In order to analyze the running time of the algorithm we observe in Lemmas 3.3, 3.4 and Corollary 3.5, assuming that q is sufficiently large, a pair of neighbors can conflict at most twice in q rounds. (Once in a working stage, and once in a final stage of one of the vertices.) Therefore, a vertex with less than $q/2$ neighbors will have a round out of q in which it conflicts with no neighbor. In this round it will select a final color. Since $q > 2 \cdot \Delta$, all vertices in the graph will select a color within q rounds. This gives rise to the following Corollary.

Lemma 3.3. *For $t \leq q$, suppose that our algorithm is executed for t rounds, and consider two neighboring nodes u, v in G that are in their respective working stages during these entire t rounds. Then u, v have the same second coordinate in their colors in the same round i, $0 \leq i < t$ (that is, $\psi_i(u) = \langle a, b\rangle$ and $\psi_i(v) = \langle c, b\rangle$, for some $0 \leq a, b, c < q$) at most once during these t consequent rounds.*

Proof. Assume that in some iteration i it holds that $\psi_i(u) = \langle a, b\rangle$ and $\psi_i(v) = \langle c, b\rangle$. For each of the following iterations $j = i+1, i+2, ...$, the difference between the second coordinates is $(c-a)\cdot(j-i) \bmod q$. Note that since q is a prime and $a \neq c$ (since, by Lemma 3.2, the coloring is proper in all iterations, and in particular, ψ_i is a proper coloring), the equality $(c - a)(j - i) \bmod q = 0$ can only hold when $(j - i) \bmod q = 0$, i.e., only after additional q iterations. □

In the following lemma we complement Lemma 3.3.

Lemma 3.4. *For $t \leq q$, suppose that our algorithm is executed for t rounds, and consider two neighboring nodes u, v in G, such that u is in working stage and v is in final stage during these entire t rounds. Then u, v have the same second coordinate in their colors in the same round i, $0 \leq i < t$ (that is, $\psi_i(u) = \langle a, b\rangle$ and $\psi_i(v) = (0, b)$, for some $0 \leq a, b < q$) at most once during these t consequent rounds.*

Proof. Since v is in final stage, its color does not change during these t rounds. Indeed, it holds that $\langle 0, b\rangle = \langle 0, (b + 0) \bmod q\rangle$. On the other hand, u is in the working stage. If initially the color of v is $\langle c, d\rangle$, for some $0 \leq c, d < q$, then in the following t rounds it changes as follows: $\langle c, (d + c) \bmod q\rangle$, $\langle c, (d + 2c) \bmod q\rangle, \ldots, \langle c, (d + tc) \bmod q\rangle$. Since q is prime, all these values of the second coordinate are distinct in the field of integers modulo q. In other words, the equality $d + xc \equiv b \pmod q$ holds for exactly one element x of this field. Thus v conflicts with u at most once, in the round i where $d + ic \equiv b \pmod q$. □

Corollary 3.5. *Given a graph $G = (V, E)$ with a proper k-coloring, where $k = \Theta(\Delta^2)$, our Additive-Group Coloring algorithm produces a proper $O(\sqrt{k})$ coloring within $O(\Delta)$ rounds.*

Proof. By Lemma 3.3, for $q > 2\Delta$, two adjacent nodes in the working stage (whose colors are not final) cannot conflict with one other more than once during the first q rounds of the algorithm. However, two adjacent nodes can also conflict if exactly one of them has selected a final color. Once this happens, it will conflict with its neighbor that is still in the working stage at most once during these q rounds. (See Lemma 3.4.) Since any node starts from a working state, and once the state transits to final its color does not change anymore, a node cannot conflict with each of its neighbors more than twice. Therefore, for each node, within $q > 2 \cdot \Delta$ rounds, there must be a round in which it does not conflict with any of its neighbors. Hence, all nodes will reach a final stage within q rounds. Since $q \leq 2\sqrt{k} = O(\Delta)$, the statement about the running time of the corollary follows. A final color is of the form $\langle 0, b\rangle$, $0 \leq b < q$. Thus the number of employed colors is at most $q = O(\sqrt{k})$. □

Corollary 3.6. *Any graph $G = (V, E)$ can be colored with $\Delta + 1$ colors within $O(\Delta) + \log^* n$ rounds, by a locally-iterative algorithm.*

Proof. Running Linial's algorithm [33] on the input graph $G = (V, E)$ will produce a coloring $\varphi(G)$ using $O(\Delta^2)$ colors within $\log^* n + O(1)$ rounds. (Recall that Linial's algorithm is locally-iterative.)
At the second stage we run our Additive-Group algorithm on $\varphi(G)$. This results in a new proper coloring $\psi(G)$ that employs $O(\Delta)$ colors. Computing the coloring ψ from φ requires $O(\Delta)$ rounds, by Corollary 3.5. At the last stage we reduce the number of colors to $\Delta + 1$ using the standard color reduction. This also requires $O(\Delta)$ time. Note that the standard color reduction is a locally-iterative algorithm as well. Therefore, the overall running time is $\log^* n + O(1) + O(\Delta) + O(\Delta) = O(\Delta + \log^* n)$. □

3.2 Halving the number of colors using 1-bit-messages per-round

In this section we devise a more bit-efficient algorithm than the algorithm presented in the previous section. Specifically, while the Additive-Group coloring stage requires just 1 bit per edge per round, the standard reduction performed in the last stage may require $O(\log \Delta)$ bits for color updates for each round. We devise an improved method that requires messages of just 1 bit. Specifically, we devise an algorithm reducing the number of colors from $O(\Delta^2)$ to $\Delta + 1$ within $O(\Delta)$ rounds using messages of 1 bit per edge per round. Consequently, the overall bit complexity of the $(\Delta + 1)$ coloring algorithm is $O(\log n + \Delta)$ in the one bit model.

In this algorithm there is no need for a prime parameter, any integer greater than Δ will do. Given a graph with a proper k coloring, $k \geq 2\Delta + 2$, we use the parameter $\lceil q = \frac{k}{2} \rceil$ where $q \geq \Delta + 1$ and produce a proper q coloring. Initially, each color c, $0 \leq c < k$, is represented as an ordered pair: $\langle \lfloor c/q \rfloor, c \bmod q \rangle$. Note that $\lfloor c/q \rfloor \in \{0, 1\}$. The pseudocode is provided below.

Algorithm 1 One-bit AG halving reduction

```
1: for i = 0, 1, , ... ⌈Δ + 1⌉ do
2:    let ψ_i(v) = ⟨a_v, b_v⟩ be the color of v in iteration i    // a_v ∈ {0, 1}
3:    ∀v ∈ V such that ψ_i(v) = ⟨1, b_v⟩ in parallel do:
4:    if not exists (v, u) ∈ E where ψ_i(u) = ⟨0, b_v⟩ then
5:       ψ_{i+1}(v) = ⟨0, b_v⟩
6:    else
7:       ψ_{i+1}(v) = ⟨1, b_v + 1 mod q⟩
8:    end if
9: end for
```

We analyze the algorithm using the following lemmas.

Lemma 3.7. *Given an arbitrary graph $G = (V, E)$ with a proper $k \geq 2\Delta + 2$ coloring, one-bit AG halving reduction preserves a proper coloring of the input graph in every round.*

Proof. Assume that in iteration i the coloring is proper. Therefore, $\forall (u, v) \in E$, $\langle a_u, b_u \rangle = \psi_i(u) \neq \psi_i(v) = \langle a_v, b_v \rangle$. In iteration $i + 1$ there are 2 options.

Option 1: $\psi_{i+1}(v) = \langle 0, b_v \rangle$, and this means that $\psi_i(u) \neq \langle 0, b_v \rangle$, since in this case $\psi_i(v)$ is either $\langle 0, b_v \rangle$ or $\langle 1, b_v \rangle$. Moreover, this means that $\psi_{i+1}(u)$ cannot become $\langle 0, b_v \rangle$ during this iteration. Thus, $\psi_{i+1}(u) \neq \psi_{i+1}(v)$.

Option 2: $\psi_{i+1}(v) = \langle 1, b_v + 1 \bmod q \rangle$. From the proper coloring assumption we know that if $\psi_i(u) = \langle 1, b_u \rangle$ then $\psi_i(v)$ is $\langle 1, b_v \rangle$ with $b_v \neq b_u$. Therefore, either $\psi_{i+1}(u) = \langle 1, b_u + 1 \bmod q \rangle \neq \psi_{i+1}(v)$ or $\psi_{i+1}(u) = \langle 0, b_u \rangle \neq \psi_{i+1}(v)$. On the other hand, if $\psi_i(u) = \langle 0, b_u \rangle$, then $\psi_{i+1}(u) = \langle 0, b_u \rangle$ as well, and again $\psi_{i+1}(u) \neq \psi_{i+1}(v)$. □

Lemma 3.8. *Given any graph $G = (V, E)$ with a proper $k \geq 2\Delta + 2$ coloring, one-bit AG halving reduction will cause every node to have a final color in the range $\{0, 1, ..., p-1\}, p = \lceil k/2 \rceil$, after $\Delta + 1$ rounds.*

Proof. Note the following: A node u can conflict with another node v in One-bit AG halving if $\psi(u) = \langle 0, b_v \rangle$ and $\psi(v) = \langle 1, b_v \rangle$. After that these nodes may conflict again only once p additional rounds have passed. Therefore, within p rounds, a node can have a conflict at most once with every adjacent node. So, if $p \geq \Delta + 1$, from the pigeonhole principle there will always be a round where v becomes final. □

Now we discuss the scenario when $(\Delta + 1)$-coloring is computed from scratch. To this end, Linial's algorithm is executed first. In each of its $O(\log^* n)$ rounds, vertices exchange their colors in that round with their neighbors. The ranges of colors in round $1, 2, 3, ...$ are $O(n)$, $O(\Delta \log n)^2$, $O(\Delta \log \log n)^2$, etc. Consequently, the bit complexity per edge is $O(\log n + \log \Delta + \log \log n + \log \Delta + \log \log \log n + ...) = O(\log n + \log \Delta \cdot \log^* n)$. Note that $\log \Delta \log^* n \leq O(\log n + \Delta)$. We summarize this in the next corollary.

Corollary 3.9. *Coloring any input graph properly with $\Delta+1$ colors can be computed within $O(\Delta+\log n)$ rounds in the one-bit model. Moreover, obtaining a $(\Delta + 1)$-coloring from $O(\Delta^2)$-coloring in this model requires $O(\Delta)$ rounds.*

3.3 Computing $O(\Delta \cdot k)$ coloring within $O(\Delta/k)$ rounds

In this section we describe a minor change in AG algorithm that works in the CONGEST, LOCAL and SET-LOCAL models. (It will not work in the one-bit model). This way, a faster computation is performed, in the expense of increasing the number of colors. Specifically, for an integer k, such that, $1 \leq k < \Delta$ we compute $O(\Delta \cdot k)$-coloring within $O(\Delta/k)$ rounds, starting from an $O(\Delta^2)$-coloring. The change we suggest is to use triplets instead of ordered pairs for representing colors. Next, we provide the pseudocode of the algorithm. Below, we analyze the algorithm. The algorithm starts with a proper $O(\Delta^2)$ coloring, where each color is represented by a triplet $\langle a_v, b_v, c_v \rangle$, such that $a_v, b_v \in \{0, 1, ..., q-1\}, q = O(\Delta)$, $c_v = 0$. During an execution the colors change, but it always holds that $0 \leq a_v, b_v < q$ and $0 \leq c_v < k$.

Algorithm 2 Refine-AG

1: Let $\psi_i(v) = \langle a_v, b_v, c_v \rangle$ be the current color // $0 \leq c_v < k$, initially $c_v = 0$
   // Invariant: $a_v = 0$ or $c_v = 0$
2: **if** $a_v \neq 0$ **then**
3: **if** exists an index $j, 0 \leq j < k$, such that:
4: 1. for all neighbors u of v with $a_u \neq 0$:
5: $\langle (b_v + j \cdot a_v) \bmod p, j \rangle \neq \langle (b_u + j \cdot a_u) \bmod q, j \rangle$
6: and
7: 2. for all neighbors u of v with $a_u = 0$:
8: $\langle (b_v + j \cdot a_v) \bmod q, j \rangle \neq \langle b_u, c_u \rangle$
  then
9: $\psi_{i+1}(v) = \langle 0, (b_v + j \cdot a_v) \bmod q, j \rangle$
10: **else**
11: $\psi_{i+1}(v) = \langle a_v, (b_v + k \cdot a_v) \bmod q, 0 \rangle$
12: **end if**
13: **end if**
14: **if** $a_v = 0$ **then**
15: $\psi_{i+1}(v) = \psi_i(v)$
16: **end if**

Lemma 3.10. *Given an arbitrary graph $G = (V, E)$ with a proper $O(\Delta^2)$ coloring, Refine AG produces a proper coloring after every round.*

Proof. The proof is by induction.

Base: The initial coloring is proper.

Step: We assume that in iteration i the coloring is proper. Next, we show that it is also proper in iteration $i+1$. For a positive integer x, we denote the values of a_v, b_v, c_v in iteration x by $a_{v_x}, b_{v_x}, c_{v_x}$, respectively. If line 9 of the algorithm was executed then $\psi_{i+1}(v) = \langle 0, (b_v + j \cdot a_v) \bmod q, j \rangle$. Since $a_{v_{i+1}} = 0$, $\psi_{i+1}(v)$ cannot be equal to the chosen colors in iteration $i+1$ of any of v's neighbors u that executed line 11, simply because $a_{u_{i+1}} \neq 0$. Thus, assume that another node executed line 9 and caused a conflict. This means that both nodes have the same index j, and $\langle (b_{v_i} + j \cdot a_{v_i}) \bmod q, j \rangle = \langle (b_{u_i} + j \cdot a_{u_i}) \bmod q, j \rangle$. But this is impossible, since the if statement in lines 3-5 prevents it.

It is left to analyze the case that both neighbors execute line 11. This means that $a_{v_{i+1}} \neq 0$ and $a_{u_{i+1}} \neq 0$. Thus u and v have not executed line 9 before. The value of c_u and c_v can become non-zero only in line 9. Therefore, $c_{u_i} = c_{v_i} = 0$. Hence, if $\langle a_{v_i}, (b_{v_i} + k \cdot a_{v_i}) \bmod q, 0 \rangle = \langle a_{u_i}, (b_{u_i} + k \cdot a_{u_i}) \bmod q, 0 \rangle$, then $\langle a_{v_i}, b_{v_i}, c_{v_i} \rangle = \langle a_{v_i}, b_{v_i}, c_{v_i} \rangle$. This is a contradiction to the correctness of the coloring in round i. □

Lemma 3.11. *For $(u, v) \in E$ with $a_u \neq 0, a_v \neq 0$, in each round there can be at most one index j, such that $(b_v + j \cdot a_v) \bmod q = (b_u + j \cdot a_u) \bmod q$.*

Proof. Assume for contradiction that there are tow indices $j_1 > j_2$, such that

(1) $(b_v + j_1 \cdot a_v) \bmod q = (b_u + j_1 \cdot a_u) \bmod q$
and
(2) $(b_v + j_2 \cdot a_v) \bmod q = (b_u + j_2 \cdot a_u) \bmod q$.
By subtracting (2) from (1) we get $(j_1 - j_2)(a_v - a_u) = 0(\bmod q)$. Since $0 < j_1 - j_2 < k < q$, it follows that $j_1 - j_2 \neq 0 \ (\bmod q)$, and thus $a_u = a_v \ (\bmod q)$. Then, from (1) it follows that $b_u = b_v \ (\bmod q)$. But this means that $\langle a_u, b_u, c_u \rangle = \langle a_v, b_v, c_v \rangle$, since $a_u \neq 0, a_v \neq 0$ implies $c_u = c_v = 0$. However, this is a contradiction to the correctness of the coloring in each round. □

We say that a node u *conflicts* with its neighbor v, if v caused u to execute line 11 of Refine-AG. Next, we analyze how many times a node u can conflict with a neighbour v during an execution of $O(\Delta/k)$ rounds of Refine-AG.

Lemma 3.12. *A node $u \in V$ can conflict with a neighbor v of u at most twice during $\lfloor q/k \rfloor$ rounds of refine-AG.*

Proof. If $a_u = a_v \neq 0$, then $b_u \neq b_v$ and no conflict occurs. Otherwise, once a conflict of u with v occurs, the next conflict is going to occur after at least q/k rounds, as long as both u and v are in non-final states, i.e., $a_u \neq 0, a_v \neq 0$. This is because in each round the second coordinate of u advances k times by a value of a_u, while the second coordinate of v advances k times by a value of a_v. A conflict can also occur if u is in non-final state and v is in a final state. But this can happen only once. □

Corollary 3.13. *Refine-AG produces a proper coloring using $O(\Delta \cdot k)$ colors within $O(\Delta/k)$ rounds.*

Proof. From Lemma 3.10 - Lemma 3.12 we observe that in order for a node not to select a final color in a certain round, it must conflict with at least k neighbors in that round. Moreover, within $\lfloor q/k \rfloor$ rounds, this node can conflict with a certain neighbor only twice. Let $q > 2\Delta + k$ be a prime number. By the Pigeonhole principle, the number of rounds out of $\lfloor q/k \rfloor$ in which a node conflicts with k different neighbors is at most $2 \cdot \Delta/k < \lfloor q/k \rfloor$. Therefore, within $\lceil 2 \cdot \Delta/k + 1 \rceil$ rounds each node must find a final color. □

3.4 Arbdefective $O(\frac{\Delta}{p})$-coloring with defect $O(p)$

Lovasz [34] showed that in a graph with maximum degree Δ, there exists a p-defective $\frac{\Delta}{p}$-coloring, where $1 \leq p \leq \Delta$. In this section we devise an algorithm for $O(\sqrt{\Delta})$-*arbdefective* $O(\sqrt{\Delta})$-coloring within $O(\sqrt{\Delta} + \log^* n)$ rounds. More generally, our algorithm computes an $O(p)$-arbdefective $O(\Delta/p)$-coloring within time $O(\Delta/p + \log^* n)$. (Definitions of defective- and arbdefective-colorings are found in Section 2.) Our algorithm starts with computing an $O(\sqrt{\Delta})$-defective $O(\Delta)$-coloring. This is done using the algorithm

of [8] within $O(\log^* n)$ rounds. (More generally, the algorithm of [8] computes a p-defective $O((\Delta/p)^2)$-coloring, for any positive parameter p, in $\log^* n + O(1)$ time.) Then we perform $O(\Delta/p) = O(\sqrt{\Delta})$ rounds of color updates, rather than $O(\Delta)$ as in our Additive-Group algorithm. The update rule for arbdefective coloring is different from the rule for proper coloring. Specifically, we tolerate up to p conflicts. In other words, instead of setting $\psi_{i+1}(v) = \langle 0, b\rangle$ only if there are no neighbors with the same value b in the second coordinate, we set this if there are at most $p = \Theta(\sqrt{\Delta})$ neighbors of different original ψ-color with the same second coordinate b. We will show in the sequel that after $O(\Delta/p) = O(\sqrt{\Delta})$ rounds all colors are of the form $\langle 0, b\rangle$, and each color class induces a subgraph of arboricity $O(p)$. Thus, as a result we have an $O(\sqrt{\Delta})$-arbdefective $O(\sqrt{\Delta})$-coloring. The pseudocode of the algorithm is provided below. The next lemmas analyze its running time and show its correctness.

Algorithm 3 Arbdefective-Color$(G, v, p = \sqrt{\Delta})$

ψ = compute an $O(p)$-defective $O((\Delta/p)^2)$-coloring of G using [8] /* $O(\sqrt{\Delta})$-defective $O(\Delta)$-coloring */
represent $\psi_0(v)$ as an ordered pair $\langle a, b\rangle$, such that $a, b \in O(\Delta/p)$. /* $a, b \in O(\sqrt{\Delta}) * /$
let $q = \Theta(\Delta/p)$ be the smallest prime such that q is greater than $2\lceil\Delta/p\rceil + 1$
for $i = 0, 1, , ...2\lceil\Delta/p\rceil$ **do**
    if v has at most p neighbors u of a different ψ-color, such that the second coordinate of $\psi_i(u)$ equals the second coordinate of $\psi_i(v)$ **then**
        $\psi_{i+1}(v) = \langle 0, b\rangle$
    else
        $\psi_{i+1}(v) = \langle a, (a + b) \bmod q\rangle$
    end if
end for

Lemma 3.14. *The produced coloring $\psi_{2\lceil\Delta/p\rceil+1}$ is of the form $\langle 0, b\rangle$, $0 \le b < q = \Theta(\Delta/p)$, for all $v \in V$.*

Proof. Consider a vertex $v \in V$. The vertex v can conflict at most twice with each neighbor u of different ψ-color within q rounds, i.e., at most once before u finalizes its color, and at most once after that. If v conflicts with more than p neighbors in each round, it means it has more than $\frac{1}{2} \cdot p \cdot (2\lceil\Delta/p\rceil + 1) > \Delta$ neighbors. This is a contradiction. Therefore, there is a round $i \in \{0, 1, ..., 2\lceil\Delta/p\rceil\}$ in which v conflicts with at most p neighbors. In this round its color finalizes, i.e., becomes of the form $\langle 0, b\rangle$. □

Lemma 3.15. *The resulting coloring $\psi_{2\lceil\Delta/p\rceil+1}$ has arbdefect at most $O(p) = O(\sqrt{\Delta})$.*

Proof. For the purpose of analysis, orient each edge $(u, v) \in E$ towards the endpoint that first set ψ_{i+1} to $\langle 0, b\rangle$. If both endpoints u, v did it in the same round, orient (u, v) towards the endpoint with greater ID. Let i denote the round in which v selects a color of the form $\langle 0, b\rangle$ for the first time. Observe that once a vertex v selects a color of the form $\langle 0, b\rangle$, its outgoing neighbors have already colors of the form $\langle 0, b'\rangle$. Thus, they will never change their colors from this moment and on. Moreover, the number of such neighbors of v of different original ψ-color and the same second coordinate of ψ_i is at most $p = \sqrt{\Delta}$. In addition, v may have at most $O(p)$ neighbors with the same original ψ-color, since the coloring ψ computed in line 1 is $O(p)$-defective. Thus, upon termination all vertices of the same $\psi_{2\lceil\Delta/p\rceil+1}$-color induce a subgraph with arboricity $O(p)$. This is because each vertex in such a subgraph has $O(p)$ outgoing edges, each of which can be assigned a distinct label from a range of size $O(p)$. Then, all edges of the same label form a forest, and the number of forests is $O(p)$. In other words, the resulting coloring has arbdefect at most $O(p)$. □

Lemma 3.16. *The running time of the algorithm is $O(\Delta/p + \log^* n) = O(\sqrt{\Delta} + \log^* n)$.*

Proof. Computing a defective-coloring in line 1 requires $O(\log^* n)$ time. Each iteration of the for-loop requires a single round. There are $O(\Delta/p) = O(\sqrt{\Delta})$ such iterations. □

The latter result gives rise to improved $(1 + \epsilon)\Delta$-coloring and $(\Delta + 1)$-coloring algorithms. This is summarized in the next theorem.

Theorem 3.17. *We compute $(1 + \epsilon)\Delta$-coloring within $O(\sqrt{\Delta} + \log^* n)$ deterministic time, for an arbitrarilly small constant $\epsilon > 0$, and $(\Delta + 1)$-coloring within $O(\sqrt{\Delta \log \Delta} \log^* \Delta + \log^* n)$ deterministic time.*

Proof. In [2] it was shown that given an $O(\sqrt{\Delta})$-arbdefective $O(\sqrt{\Delta})$-coloring one can compute a proper $(1+\epsilon)\Delta$-vertex-coloring within $O(\sqrt{\Delta}+\log^* n)$ deterministic time. (For more details, we refer the reader to the discussion in Section 3.4 of [2]. However, such an arbdefective coloring is computed in [2] only within time $(\sqrt{\Delta}\log^3 \Delta + \log^* n)$. See Lemma 3.5, Corollary 3.12, and the discussion preceding it in [2]. Consequently, the overall running time of the algorithm of [2] for $(1+\epsilon)\Delta$-coloring is $(\sqrt{\Delta}\log^3 \Delta+\log^* n)$ as well.) Our improved running time of arbdefective coloring (cf. Lemma 3.16) in conjunction with the procedure of [2] (i.e., by replacing the invocation of line 1 of Algorithm 1 of [2] by an invocation of our new algorithm Arbdefective-Color), gives rise to a deterministic $(1+\epsilon)\Delta$-coloring within $O(\sqrt{\Delta}+\log^* n)$ time.

It is shown in [19] that a deterministic $(\Delta+1)$-coloring is obtained in $O(\sqrt{\Delta}\log^{2.5} \Delta+\log^* n)$ time using arbdefective colorings. Specifically, the proof of Lemma 4.2 of [19] shows that given an β-arbdefective k-

coloring algorithm with $O(k)$ running time, a proper $(\Delta+1)$-coloring is computed within time $O(\log^* n + T_A)$, where T_A is given by the recursive formula $T_A(\Delta) = O(k \log^* \Delta) + T_A(O(\beta^2 \log \Delta))$. By setting $\beta = \sqrt{\Delta/(c \log \Delta)}$ and $k = \sqrt{c\Delta \log \Delta}$, for a sufficiently large constant c, this recursive formula evaluates to $O(\sqrt{\Delta \log \Delta} \log^* \Delta)$. Moreover, we compute such β-arbdefective k-coloring within $O(\sqrt{\Delta \log \Delta} + \log^* n)$ time. (See Lemma 3.16.) Thus by using our Arbdefective-Color algorithm in conjunction with the procedure of [19], we obtain $(\Delta + 1)$-coloring in $O(\sqrt{\Delta \log \Delta} \log^* \Delta + \log^* n)$ time. □

4 Fully-Dynamic Self-Stabilizing algorithms with $O(\Delta + \log^* n)$ rounds

4.1 Fully-Dynamic Self-Stabilizing $O(\Delta)$-Coloring

In this section we employ a variant of Linial's algorithm for $O(\Delta^2)$-coloring that allows a vertex v to avoid being colored by colors from a given set $R(v)$ of size at most $O(\Delta)$ [2]. (This is useful when selecting a new color, to avoid collisions with some neighbors that have already obtained final colors.) We refer to this algorithm as Algorithm Excl-Linial.

Algorithm Excl-Linial is identical to Linial's original algorithm, except for the final stage that transforms a proper $O(\Delta^3)$-coloring into a proper $O(\Delta^2)$-coloring. In this stage each vertex v computes a polynomial $P_v(x)$ of degree 2 in a field of size $O(\Delta)$, and selects a color $\langle x, P_v(x)\rangle$, such that $\langle x, P_v(x)\rangle \neq \langle y, P_u(y)\rangle$, for any neighbor u of v and any y in that field. Since the degree of the polynomials in this stage is 2, each polynomial intersects with a neighboring node's polynomial in at most two points. Hence, there are at most 2Δ points on P_v that may intersect with some neighbor. If the field is of size $2\Delta + 1$, there must be a point such that $\langle x, P_v(x)\rangle \neq \langle y, P_u(y)\rangle$ for all neighbors u of v and all elements y in the field. Such a pair is selected by the original algorithm of Linial. In the modified variant, on the other hand, the field is of size greater than 3Δ. Consequently, if a set $R(v)$ of at most Δ forbidden colors is provided, there still exists an element x in that field, such that $\langle x, P_v(x)\rangle$ is not equal to any of the colors in the set $R(v)$, and neither to any $\langle y, P_u(y)\rangle$, for a neighbor u and an element y. Such a color is selected as a final color. Thus, we obtain an $O(\Delta^2)$-coloring were all colors belong to sets that exclude $O(\Delta)$ colors each, within $\log^* n + O(1)$ time. This completes the description of algorithm Excl-Linial.

Before describing our self-stabilizing algorithm, we define some notation, and describe yet another useful variant of Linial's algorithm, which we call Algorithm Mod-Linial. Let $r = \log^* n + O(1)$ denote the number of iterations in Linial's algorithm. Let $t_r = O((\Delta \log n)^2), t_{r-1} = O((\Delta(\log \Delta + \log\log n)^2), ..., t_1 = O(\Delta^2)$ denote upper bounds on the number of colors in the different iterations of Linial's algorithm. Define the intervals $I_0, I_1, I_2, ...$ as follows. $I_0 = [0, t_1 - 1], I_1 = [t_1, t_1 + t_2 - 1], ..., I_{r-1} = [t_1 + t_2 + ... + t_{r-1},\ t_1 + t_2 + ... + t_r - 1]$, $I_r = [t_1 + t_2 + ... + t_r,\ t_1 + t_2 + ... + t_r + n - 1]$. Since each such interval contains sufficient number of colors, we can map each color palette of each iteration of

Linial's algorithm to one of the intervals defined above. Specifically, the palette of the first iteration is mapped to I_{r-1} (which is of size t_r), the palette of the second iteration is mapped to I_{r-2} (which is of size t_{r-1}), and so on, up to the last palette that is mapped to I_0. This way Linial's algorithm is modified, so that in each iteration $i = 1, 2, ..., r$ a coloring using a palette I_{r-i+1} is transformed into a coloring using the palette I_{r-i}. (The actual number of colors used from this palette is $O((\Delta \log^{(i)} n)^2)$.) The modified algorithm will be referred to as Mod-Linial. It accepts as input a color of a vertex v, a (sub)set of its neighbors colors, and a set of $O(\Delta)$ forbidden colors, and returns a new color for v. The range $I_r = [t_1 + t_2 + ... + t_r,\ t_1 + t_2 + ... + t_r + n - 1]$ will be used for an initial n-coloring obtained from IDs.

Our fully-dynamic self-stabilizing algorithm works as follows. The RAM of each vertex consists of a variable that holds a color in a range $\{0, 1, ..., t_1 + t_2 + ... + t_r + n - 1\}$. The ROM of each vertex holds the algorithm, the number of vertices n and the maximum degree Δ. In each round each vertex v checks whether it is in a proper state, i.e., its color is distinct from all colors of its neighbors. (See the pseudocode of Procedure Check-Error below.) If v is not in a proper state, the vertex returns to its initial state. (See lines 1- 3 of Procedure Self-Stabilizing-Coloring.) We define the initial state of a vertex with ID $j \in 0, 1, ..., n - 1$ by the color $t_1 + t_2 + ... + t_r + j$. Otherwise, the vertex is in a proper state. Then, the vertex v computes its next color or finalizes the current one. (See lines 4 - 20 of Procedure Self-Stabilizing-Coloring.) Specifically, as long as the vertex color belongs to an interval I_j for $j \geq 2$, i.e., the color is significantly larger than Δ^2, the vertex computes the next color from a smaller range using the algorithm Mod-Linial (lines 6-7 of Procedure Procedure Self-Stabilizing-Coloring). Once a color is in the interval I_1, the vertex must select a new color in the interval I_0 that is distinct from any neighboring color that is also in I_0. This is done in lines 9 - 11 of the procedure. The set S', computed in line 10 and provided as the third parameter of Procedure Mod-Linial in line 11, contains all possible colors that neighbors u of v that run already lines 12 - 18 (i.e., their colors are small enough) may obtain in the current iteration. Note that for each such $u \in \Gamma(v)$ there are at most 2 such colors. Finally, a color that is in the range I_0 either becomes final or changes to another color in I_0 according to Algorithm AG. See lines 12 - 18. This completes the description of the algorithm. Its pseudocode is provided below. Next, we analyze the algorithm.

Lemma 4.1. *Given an arbitrary graph $G = (V, E)$, our self-stabilizing algorithm produces a proper coloring $\psi(G)$ in each round, once faults no longer occur.*

Proof. Consider a round i. If a node $v \in V$ has a color that is equal to that of a neighbor u, i.e., $\psi_i(u) = \psi_i(v)$, then $\psi_{i+1}(v) = t_r + t_{r-1} + ... + t_1 + id(v) \neq \psi_{i+1}(u) = t_r + t_{r-1} + ... + t_1 + id(u)$. Otherwise, lines 3 - 20 are executed. Since it is assumed that no more faults will occur, we prove that lines 3-20 provide a proper coloring. If $j \geq 2$ (line 7) then $\psi_{i+1}(v)$ will be in the range I_{j-1}. (Any element in I_j is greater than any element in I_{j-1}, and thus numerical values of colors decrease as the algorithm proceeds.

Also, all intervals are disjoint.) Therefore, all neighbors u with $\psi_i(u) \notin I_j$ will not select a new color $\psi_{i+1}(u)$ from I_{j-1}. For a neighbor u with $\psi_i(u) \in I_j$, its color belongs to Q, and Mod-Linial algorithm will produce a proper coloring.

If $j = 1$ then Mod-Linial algorithm works in the following way. It computes a new color from t_0, such that it is distinct from all neighbors' colors that transit from I_1 to I_0 in round i, and from all colors of the set S'. The latter set contains all possible colors that can be used in round $i + 1$ by neighbors of v with colors in the range I_0 in round i. Consequently, the new color of $\psi_{i+1}(v)$ of v is distinct from the new colors of such neighbors. Moreover, the new color is also distinct from new colors of the rest of the neighbors, since they were either in I_1 in round i, and do not collide with v in round $i + 1$ due to correctness of Mod-Linial, or in a higher range, and thus are not in I_0 in round $i + 1$.

If $j = 0$, then lines 12 - 19 execute our Additive-Group algorithm (see Corollary 3.5), and produce a proper coloring for neighbors with $j = 0$. For neighbors with $j > 0$, the coloring is proper as well, by analysis of previous cases in this proof. □

Check-Error (my_color, [$neighbors\_colors$])

if $my\_color \in neighbors\_colors$ **then**
  return error
end if
return valid

Algorithm 4 Self-Stabilizing-Coloring

if Check-Error(my_color, [$neighbors\_colors$]) = error **then**
  $my\_color = t_1 + t_2 + ... + t_r + my\_ID$ /* initial state */
else
  Let I_j denote the range that my_color belongs to
  Let Q denote the subset of [$neighbors\_colors$] of all colors that belong to I_j
  if $j \geq 2$ **then**
    my_color = Mod-Linial($my\_color, Q, \emptyset$)
  else if $j = 1$ **then**
    Let S denote the subset of [$neighbors\_colors$] of all colors that belong to I_0, represented as ordered pairs
    Let $S' = \{\langle a, (b + a) \bmod q\rangle \mid \langle a, b\rangle \in S\} \cup \{\langle 0, b\rangle \mid \langle a, b\rangle \in S\}$
    my_color = Mod-Linial(my_color, Q, S') /* avoid collisions with S' */
  else if $j = 0$ **then**
    represent my_color as an ordered pair $\langle a, b\rangle$
    if $\langle a, b\rangle$ conflicts with a color in Q **then**
      $my\_color = \langle a, (a + b) \bmod q\rangle$
    else
      $my\_color = \langle 0, b\rangle$ /* final color */
    end if
  end if
end if

Lemma 4.2. *Given an arbitrary graph $G = (V, E)$, our fully-dynamic self-stabilizing algorithm produces a proper $O(\Delta)$-coloring with $O(\Delta + \log^* n)$ stabilization-time.*

Proof. In the end of each round $i = 1, 2, ...$, counting from the moment that faults stop occurring, all colors are in the range $I_0 \cup I_1 \cup ... \cup I_{r+1-i}$. Therefore, within $r + 1 = \log^* n + O(1)$ rounds, all colors are in the range I_0. From this moment and on, the procedure executes our Additive-Group algorithm in all vertices. Therefore, by Corollary 3.5, within $O(\Delta)$ additional rounds the number of colors becomes $O(\Delta)$. □

We also obtain a self-stabilizing algorithm that employs exactly $(\Delta + 1)$ colors. To this end, in each round each vertex v with a color of the from $< 0, b_v >$ whose all neighbors also have 0 in their first color coordinate performs the following. If b_v is greater then the colors of all v's neighbors, then v selects a new color $< 0, b'_v >$ such that $0 \leq b'_v \leq \Delta$, and $< 0, b'_v >$ is distinct from all colors of v's neighbors. Consequently, once all colors are of the form $< 0, b >$, $b = O(\Delta)$, at most $O(\Delta)$ additional rounds are required to arrive to a $(\Delta + 1)$- coloring, because at least one color is eliminated in each round. (This is the greatest color, as long as there are colors greater than Δ.) Further discussion about $(\Delta + 1)$-coloring appears in the full version of this thesis [11]. I'll summarize this below.

Theorem 4.3. *Given an arbitrary graph $G = (V, E)$, our fully-dynamic self-stabilizing algorithm produces a proper $(\Delta + 1)$-coloring with $O(\Delta + \log^* n)$ stabilization time.*

4.2 Fully-Dynamic Self-Stabilizing MIS, MM, and $(2\Delta - 1)$-Edge-Coloring

We employ our self-stabilizing coloring algorithm from the previous section in order to compute MIS as follows. We add a bit μ_v to the RAM of each vertex $v \in V$. This bit represents whether v is in the MIS (if $\mu_v = 1$) or not in the MIS (if $\mu_v = 0$). We add the following instruction in the end of Procedure Self-Stabilizing-Coloring. If all neighbors u of v with smaller colors than that of v have $\mu_u = 0$, then we set $\mu_v = 1$. Otherwise, we set $\mu_v = 0$. This completes the description of the changes required to compute an MIS. The next lemma shows that within i rounds, for $i > 0$, after the stabilization of coloring, all vertices with colors $1, 2, ..., i$ induce a subgraph with a properly computed MIS. Consequently, within $O(\Delta)$ additional rounds an MIS of the entire input graph is achieved.

Lemma 4.4. *Consider a graph G after the stabilization of coloring. For $i > 0$, within i additional rounds, all vertices with colors $1, 2, ..., i$ induce a subgraph with a properly computed MIS. Consequently, within $O(\Delta)$ additional rounds an MIS of the entire input graph is achieved.*

Proof. The proof is by induction on i. **Base (i = 0):** All vertices of color $< 0, 0 >$ do not have neighbors with smaller colors, and thus their μ bits become equal to 1. Since in this stage the coloring is proper,

the set of such vertices is independent. Since it does not contain vertices of other colors in the current stage, the set is also maximal. (in other words, this set is an MIS of itself.)

Step: We consider the subset of vertices with the following colors:
$\langle 0,0\rangle, \langle 0,1\rangle, ..., \langle 0,i\rangle$. By induction hypothesis, within i rounds from stabilization of the coloring, the subgraph induced by vertices of colors $\langle 0,0\rangle, \langle 0,1\rangle, ..., \langle 0,i-1\rangle$ has a properly computed MIS. Since colors do not change after stabilization, the μ bits of the vertices of this subgraph will not change in round $i+1$. In this round each vertex of color $\langle 0,i\rangle$ sets its μ bit to 1 if it has no neighbors with a smaller color in the MIS, and sets it to 0 otherwise. Consequently, there are no pair of neighbors with colors from $\{\langle 0,0\rangle, \langle 0,1\rangle, ..., \langle 0,i\rangle\}$, for which both their μ bits are set to 1. Moreover, each vertex of color smaller than $\langle 0,i\rangle$ for which $\mu = 0$ must have a neighbor with $\mu = 1$, by induction hypothesis. Each vertex of color $\langle 0,i\rangle$ for which $\mu = 0$ must have a neighbor with a smaller color and $\mu = 1$, according to the instruction that is executed in round $i+1$ after the stabilization of the coloring. Hence, after that round, the subgraph induced by $\langle 0,0\rangle, \langle 0,1\rangle, ..., \langle 0,i\rangle$ has a properly computed MIS. □

Theorem 4.5. *Given an arbitrary graph $G = (V, E)$, our self-stabilizing algorithm produces a proper MIS within $O(\Delta + \log^* n)$ rounds after the last fault.*

Proof. Let $t_{cd} = O(\Delta + \log^* n)$ be the stabilization time of the coloring algorithm. (See Theorem 4.3.) Denote by U_i, $i = 1, 2, ..., \Delta + 1$, the set of vertices v that belong to MIS (i.e., have $\mu_v = 1$) at round $t_{cd} + i$ after faults stop occurring. Let ψ be the $(\Delta+1)$-coloring maintained by the algorithm. (We know that t_{cd} rounds after the last fault occurred, ψ is indeed a proper $(\Delta+1)$-coloring.)

We prove by induction on i that at time $t_{cd} + i$ after faults stop occurring, for $i = 1, 2, ..., \Delta + 1$, U_i is an MIS for the set $\hat{V}_i = \{v \mid 1 \leq \psi_i(v) \leq i\}$, where ψ_i is the coloring ψ maintained by the algorithm at that time.

Base (i = 1): All vertices of $\hat{V}_1$ form an independent set (because φ_1 is a proper $(\Delta+1)$-coloring, because it is the coloring ψ more than t_{cd} rounds after the last fault occurred), and each of them joins MIS because they have no neighbors of smaller color.

Step: For some $i \leq \Delta$ we assume that U_i is an MIS for $\hat{V}_i$. Consider a vertex $v \in U_{i+1}$, i.e., $\psi_{i+1}(v) = i+1$. This vertex had the same color $i+1$ for all the rounds $t_{cd}+1, t_{cd}+2, ..., t_{cd}+i+1$, counting from the moment T when faults stopped occurring. By end of round $T + t_{cd} + i$ or earlier, all its neighbors of smaller color (they also did not change their colors during the time interval $[T + t_{cd}, T + t_{cd} + i]$) have set their values μ_u. So in round $T + t_{cd} + i + 1$, if v has no smaller color neighbor in the MIS, it joins MIS. (It might have joined earlier, but it will anyway check again whether it has to join in round $T + t_{cd} + i + 1$.) Since vertices of $V_{i+1} = \{v \mid \psi_{i+1}(v) = i+1\}$ form an independent set, the resulting set U_{i+1} is an independent set for $\hat{V}_i \cup V_{i+1} = \hat{V}_{i+1}$. □

In the ordinary (non-stabilizing) setting it is possible to compute a maximal matching and an edge coloring by simulating the line-graph of the input graph, and computing an MIS and vertex-coloring of it. These solutions on the line graph directly provide solutions for maximal matching and edge coloring of the input graph within the same running time. This technique is applicable also to the self-stabilizing setting. Specifically, each vertex v simulates virtual vertices, one virtual vertex per edge adjacent on v. In the beginning of each round each vertex verifies whether the state of each of its virtual vertices that correspond to some edge equals to the state in the other endpoint of that edge. If this is not the case, the endpoint with a greater ID copies the state of the other endpoint for that virtual vertex. Consequently all edges have consistent representations, i.e., the same state in both their endpoints, in the entire graph. Now, a self-stabilizing MIS or vertex-coloring algorithm can be simulated correctly on the line graph in order to produce self-stabilizing maximal matching and edge-coloring of the input graph. In conjunction with Theorems 4.3, 4.5 this leads to the following result.

Theorem 4.6. *Given an arbitrary graph $G = (V, E)$, our self-stabilizing algorithms produce a maximal matching and a proper $(2\Delta - 1)$-edge-coloring within $O(\Delta + \log^* n)$ stabilization time.*

5 Edge Coloring within $O(\Delta + \log^* n)$ Rounds in the CONGEST Model and $O(\Delta + \log n)$ Rounds in the Bit-Round Model

We employ our techniques in order to compute *edge colorings* using small messages. The algorithm consists of two stages. The first stage constructs an $O(\Delta^2)$-edge-coloring from scratch, and the second stage computes an $O(\Delta)$-coloring from this $O(\Delta^2)$-coloring. We remark that we cannot use the algorithm of Linial for the first stage, since its message complexity in the case of edge-coloring is quite large. Instead, we do the following. We invoke Kuhn's algorithm [31] for 2-defective Δ^2-edge coloring. This algorithm orients all edges towards endpoints with greater IDs. Then, each vertex assigns its outgoing edges distinct colors from the set $\{1, 2, ..., \Delta\}$. It also assigns its incoming edges distinct colors from the same range. Consequently, each edge obtains a pair of colors, one color from each of its endpoints. This is done within a single round by sending a message of size $O(\log n)$ per edge (in both directions).

Each color of an edge $e \in E$ can be represented as an ordered pair $\psi(e) = \langle i, j \rangle$, where $i, j \in \{1, 2, ..., \Delta\}$. Note that a set of edges with the same ψ-color consists of paths and cycles, since each vertex on such an edge has at most one another edge adjacent on it in this set. This is because the defect of ψ is 2. To remove the defect we run Cole and Vishkin coloring algorithm [12] on edges of each color class in parallel and assign a new color to each $e \in E$ in the form $\psi(e) = (i, j, k)$. The first two indices i, j are the result of the first stage, and the rightmost index $k \in \{1, 2, 3\}$ is the result of Cole-Vishkin's algorithm invocation.

Next, we compute an $O(\Delta)$-edge-coloring from the $O(\Delta^2)$-edge-coloring as follows. In each round both endpoints of an edge hold its color, that will be from now on represented as an ordered pair $\langle a, b\rangle$, $a, b \in O(\Delta)$, rather than a tuple. Consequently, each endpoint can check for conflicts of edges adjacent on it. For each edge with a conflict at an endpoint, the endpoint sends a message over this edge (consisting of a single bit) to notify the other endpoint about the conflict. Then, for each edge, both of its endpoint know whether it has a conflict with some adjacent edge or not. If the current edge color is $\langle a, b\rangle$, and there is a conflict, the new color becomes $\langle a, (a+b) \bmod q\rangle$. Otherwise, it becomes $\langle 0, b\rangle$. Both endpoints update the new color of their edge. This is done within a single round and by exchanging just a single bit on each edge. Then all vertices of the graph are ready to proceed to the next round and perform it in a similar way. The algorithm stops once all edges have colors of the from $\langle 0, b\rangle$, $0 \leq b < q = O(\Delta)$. (Here q is a prime number that satisfies that the original number of colors is at most q^2 and $q \geq 2\Delta - 1$.)

Lemma 5.1. *A proper $O(\Delta)$-edge coloring is obtained in $O(\Delta + \log^* n)$ rounds in the CONGEST model.*

Proof. The algorithm starts with the invocation of Kuhn's algorithm that results in a 2-defective Δ^2-edge-coloring within $O(1)$ time. Then it is turned into a proper coloring using Cole-Vishkin algorithm within $O(\log^* n)$ time. Indeed, if prior to the execution of the latter algorithm a pair of adjacent edges had the same color $\langle i, j\rangle$, they now have distinct colors $\langle i, j, k\rangle$ and $\langle i, j, k'\rangle$, since Cole-Vishkin algorithm produces a proper 3-coloring of the edges in the set of color class $\langle i, j\rangle$. Next, in each round each color of an edge of the form $\langle a, b\rangle$ is transformed either into $\langle a, (a + b) \bmod q\rangle$ or into $\langle 0, b\rangle$. In both cases the new coloring is proper. See Lemma 3.5. Within $O(\Delta)$ rounds all colors obtain the form $\langle 0, b\rangle$. □

Lemma 5.2. *The bit complexity of our edge-coloring algorithm is $O(\Delta + \log n)$ per edge. If initially vertices know the IDs of their neighbors, then the bit complexity is $O(\Delta + \log\log n)$ per edge.*

Proof. Exchanging initial IDs between neighbors requires $O(\log n)$ bits. Exchanging the colors during the 2-defective Δ^2-edge-coloring requires $O(\log \Delta)$ bits. The first round of Cole-Vishkin algorithm is performed based on IDs of $O(\log n)$ bits. The second round of Cole-Vishkin algorithm requires $O(\log\log n)$ bits, the third one requires $O(\log\log\log n)$ bits, and so on. The last round of Cole-Vishkin algorithm requires a constant number of bits. The exchange between neighbors of the resulting proper $O(\Delta^2)$-edge coloring of the input graph requires $O(\log \Delta)$ bits. Each of the following $O(\Delta)$ rounds requires 1 bit. □

We can also produce edge-coloring with exactly $(2\Delta - 1)$-colors as follows. Once the stage of $O(\Delta)$-edge-coloring terminates, we apply a procedure similar to One-bit AG halving reduction. (See Section 3.2.) Specifically, let k be the current number of colors, and $q = \lceil k/2\rceil$. we represent each color of an edge as an ordered pair $\langle a_e, b_e\rangle$, where $a_e \in \{0, 1\}$, $b_e \in \{0, 1, ..., q - 1\}$. Then we execute 2Δ rounds to halve the number of colors. In each round, for each edge $e = (u, v) \in E$, its endpoints u, v check whether

b_e is distinct from all $b_{e'}$ of edges e' adjacent on these endpoints. Then v notifies u whether this is the case for all edges adjacent on v. In parallel, u notifies v whether this is the case for all edges adjacent on u. If both u and v pass the check, they update the color of e to $\langle 0, b_e \rangle$. Otherwise, they update it to $\langle 1, b_e + 1 \bmod q \rangle$. Since each edge has at most $2\Delta - 1$ edges adjacent on it, within 2Δ rounds all edges $e \in E$ select a color with $a_e = 0$. Hence the number of colors is halved. Repeating this for a constant number of phases converts the $O(\Delta)$-edge-coloring into a $(2\Delta - 1)$-edge-coloring. We summarize this below.

Theorem 5.3. *We compute $(2\Delta - 1)$-edge-coloring within $O(\Delta + \log^* n)$ rounds in the CONGEST model, within $O(\Delta + \log\log n)$ rounds in the Bit-Round model with knowledge of neighbors' IDs, and within $O(\Delta + \log n)$ time in the Bit-Round model without knowledge of neighbors' IDs.*

6 3-Dimensional Additive Group Algorithm

In Section 3 we described our Additive Group (shortly AG) algorithm that starts from a proper $O(p^2)$-coloring, for some prime $p \geq 2 \cdot \Delta + 1$, and computes a proper p-coloring in $O(p)$ rounds. This algorithm can be used, of course, also for decreasing the number of colors more than quadratically. Specifically, if we have an $O(p^3)$-coloring, for some prime $p \geq 2 \cdot \Delta + 1$, we can decrease the number of colors to $O(p)$ in the following way. Partition the palette $[p^3]$ into p disjoint sub-palettes $[p^2], [p^2 + 1, 2p^2], ..., [p^3 - p^2 + 1, p^3]$, and run AG$(p)$ algorithm in each sub-palette in parallel. Within $O(p)$ rounds the number of colors reduces to $O(p^2)$, and by an additional application of AG(p), we obtain a p-coloring in overall $2 \cdot O(p) = O(p)$ rounds.

In some faulty network setting it is, however, desirable to employ algorithms that do not consist of several distinct phases, like the algorithm above. These distinct phases may pose a problem when faults are introduced, and some vertices are in one phase of the algorithm, while others are in another. We, therefore, next devise a variant of our AG algorithm that reduces the number of colors from $O(p^3)$ to $O(p)$ within $O(p)$ rounds, but it is more uniform than the above algorithm, i.e., at all times all vertices perform precisely the same step. We call this algorithm *3-dimensional AG with a parameter q*, or shortly, 3AG(q). The algorithm starts by representing colors $\psi(v) = \langle c_v, b_v, a_v \rangle$ as triples, $a_v, b_v, c_v \in Z_q$. It then runs the following iterative step for $2 \cdot p$ rounds. We will assume $p \geq 3 \cdot \Delta + 1$. All additions are in Z_p.

Algorithm 5 3AG(p)

for $v \in V$ in parallel **do**
  if $c_v \neq 0$ **then**
    if $\forall u \in \Gamma(v)$ it holds that $b_v \neq b_u$ **then**
      $\psi(v) = \langle 0, b_v, a_v \rangle$
    else
      $\psi(v) = \langle c_v, b_v + c_v, a_v \rangle$
    end if
  else
    if $\forall u \in \Gamma(v)$ it holds that $a_v \neq a_u$ **then**
      $\psi(v) = \langle 0, 0, a_v \rangle$
    else
      $\psi(v) = \langle 0, b_v, a_v + b_v \rangle$
    end if
  end if
end for

Next, we analyze the algorithm.

Lemma 6.1. *Suppose we have a proper coloring φ. Then the coloring ψ obtained after one round of 3AG(p) is proper as well.*

Proof. Denote $\varphi(v) = \langle c_v, b_v, a_v \rangle$ and consider an edge (u, v). We split the analysis into two cases, depending on whether c_v is non-zero.

Case 1: ($c_v \neq 0$). In this case our analysis splits again into two cases, depending on whether all neighbors u' of v have $b_u \neq b_v$, or not.

Case 1.1: ($\forall u' \in \Gamma(v)$, $b_{u'} \neq b_v$). Then the algorithm sets: $\psi(v) = \langle 0, b_v, a_v \rangle$.

The vertex $u \in \Gamma(v)$ (recall that we have fixed an edge (u, v)) with $\varphi(u) = \langle c_u, b_u, a_u \rangle$ could have been in one of the following cases.

Case 1.1.1: ($c_u \neq 0$). Then, if $\forall z \in \Gamma(u)$, we have $b_z \neq b_u$, then $\psi(u) = \langle 0, b_u, a_u \rangle$. But recall that $b_u \neq b_v$, and thus $\psi(u) \neq \psi(v)$ as required. Otherwise, there exists a neighbor $z \in \Gamma(u)$ whith $b_z = b_u$. Then the algorithm sets $\psi(u) = \langle c_u, b_u + c_u, a_u \rangle$ and $c_u \neq 0$. But $\psi(v) = \langle 0, b_v, a_v \rangle$, i.e., $\psi(v) \neq \psi(u)$.

Case 1.1.2: ($c_u = 0$). In this case $\varphi(u) = \langle 0, b_u, a_u \rangle$. The analysis here splits again to a number of sub-cases.

Case 1.1.2.a ($b_u = 0$). Then $\varphi(u) = \langle 0, 0, a_u \rangle$, and so $\psi(u) = \langle 0, 0, a_u \rangle$ as well. But we have for every $u' \in \Gamma(v)$, $b_{u'} \neq b_v$, and so $b_v \neq 0$. Hence $\psi(v) \neq \psi(u)$.

Case 1.1.2.b: $b_u \neq 0$, but b_u stayed as is, i.e., $\psi(u) = \langle 0, b_u, a_u + b_u \rangle$ (this means that there exists a neighbor $z \in \Gamma(u)$ with $a_z = a_u$). But then again $b_v \neq b_u$, because for every $u' \in \Gamma(v)$, $b_{u'} \neq b_v$. Hence $\psi(v) \neq \psi(u)$.

Case 1.1.2.c: $\varphi(u) = \langle 0, b_u, a_u \rangle$ and $b_u \neq 0$ and $\forall z \in \Gamma(u)$, $a_z \neq a_u$. Then $\psi(u) = \langle 0, 0, a_u \rangle$. But then,

in particular, $a_v \neq a_u$, and so $\psi(v) = \langle 0, b_v, a_v \rangle \neq \langle 0, 0, a_u \rangle = \psi(u)$, as required.

Case 1.2: ($c_v \neq 0$, and there exists $u' \in \Gamma(v)$ with $b_{u'} = b_v$). Then $\psi(v) = \langle c_v, b_v + c_v, a_v \rangle$. Then if $c_u = 0$ (i.e., $\varphi(u) = \langle 0, b_u, a_u \rangle$), then in $\psi(u)$ the first coordinate is also 0 (by the rules of the algorithm), and so $\psi(v) \neq \psi(u)$.

Else we have $c_u \neq 0$. So both v and u have non-zero first coordinate, and so they do not change their third coordinate. So if $a_v \neq a_u$ then $\psi(v) \neq \psi(u)$. Otherwise ($a_v = a_u$), and so $\langle c_v, b_v \rangle \neq \langle c_u, b_u \rangle$. So if u sets $\psi(u) = \langle 0, b_u, a_u \rangle$, then $\psi(v) \neq \psi(u)$, because $c_v \neq 0$.

Else, u sets $\psi(u) = \langle c_u, b_u + c_u, a_u \rangle$, but $\langle c_v, b_v + c_v \rangle \neq \langle c_u, b_u + c_u \rangle$ because $\langle c_v, b_v \rangle \neq \langle c_u, b_u \rangle$. In either case $\psi(v) \neq \psi(u)$.

Case 2: ($c_v = 0$). If $\varphi(u) = \langle c_u, b_u, a_u \rangle$ and $c_u \neq 0$, then by symmetric argument, $\psi(v) \neq \psi(u)$. Finally, if $\varphi(v) = \langle 0, b_v, a_v \rangle$, $\varphi(u) = \langle 0, b_u, a_u \rangle$ and $\varphi(v) \neq \varphi(u)$, then by our analysis of the two-dimensional AG, we have $\psi(v) \neq \psi(u)$. □

Within the first $3 \cdot \Delta + 1$ rounds, each vertex v will have $c_v = 0$. This is because each neighbor u of v may have a conflicting b_u to the b-value b_v at most three times: once with a non-finalized b-value, once with a finalized b-value (on line 4 of the algorithm), and once with a b-value 0 (set on line 10 of the algorithm). So among $3 \cdot \Delta + 1$ first rounds, there will be a round on which for all $u \in \Gamma(v)$, $b_v \neq b_u$, and on that round v finalizes its b-value. (In fact, $2 \cdot \Delta + 2$ rounds suffice, as b_v can be zero at most once during all these rounds, assuming $p \geq 2 \cdot \Delta + 2$).

After all vertices have their $c_v = 0$, in $2 \cdot \Delta + 1$ additional rounds, by the same argument, all a_v's will be finalized.

Corollary 6.2. *The algorithm 3AG(p), starting with a proper p^3-coloring, where $p \geq 2\Delta + 2$, computes a proper p-coloring in $O(p)$ rounds.*

We next argue that one can decrease the palettes size (in both ordinary and 3-dimensional variants of the algorithm AG), at the expense of slightly increasing the running time. Consider first the ordinary (two dimensional) variant of algorithm AG, and suppose that instead of running it for $p \geq 2 \cdot \Delta + 1$ rounds, we run it for $p \geq (1 + \epsilon) \cdot \Delta$ rounds, for an arbitrary small constant $\epsilon > 0$. We will run it for $1 + \lceil \frac{1}{\epsilon} \rceil$ phases, each lasting for p rounds. (Observe, however, that vertices that run the algorithm are oblivious of the phases. They always run the same AG-iteration, on which a vertex v with $\varphi(v) = \langle b_v, a_v \rangle$ checks if it has a neighbor u with $a_v = a_u$. If it does not, it finalizes its color to $\psi(v) = \langle 0, a_v \rangle$. Otherwise it sets it to $\psi(v) = \langle b_v, a_v + b_v \rangle$.) Consider a fixed vertex v. Note that if it does not finalize its color on phase 1, it means that at least $\epsilon \cdot \Delta$ of its neighbors u have finalized their colors (and had a conflict with the color of v at least twice during the phase). Observe also that these neighbors u will be able to conflict at most once with v on each subsequent phase. Hence if v does not finalize its color for i phases,

$i = 1, 2, ..., i < 1/\epsilon$, it means that at least $i \cdot \epsilon\Delta$ among its neighbors did. Hence after $\lceil \frac{1}{\epsilon} \rceil$ phases, all neighbors of v have finalized their colors, and on the next phase v will necessarily finalize its color. The same reasoning is applicable to the 3-dimensional variant of the AG algorithm, but the number of phases grows by a factor of 2.

Corollary 6.3. *Given a proper $O(p^3)$-coloring, for some $p \geq (1+\epsilon) \cdot \Delta$, for some $\epsilon > 0$, running 3AG(p) for $O(\frac{1}{\epsilon} \cdot p)$ rounds produce a proper p-coloring.*

7 Conclusion and Future Work

The coloring technique based on Additive groups gives rise to general graph coloring. It allowed us to break a 25 years old heuristic barrier in terms or running time. Our technique also incurs a very low bit complexity of just 1 bit per round. We believe that it may turn out to be the most efficient possible technique for $O(\Delta)$ proper coloring in terms of network usage.
Additive groups technique has a several variants. It can be more specific or more generic and can also compute arbdefective coloring very efficiently. With those features it can also be used as a black box for solving other problems in graph theory as described in this work.

This work presented the begining of usage of algebric techniques that might be useful for some more applications. An example is distance-2 coloring techniques using minimal network usage. There could be also an improvment in running time for coloring graphs with bounded arboricity. It may be also possible to discover a better running time rulling sets algorithms.

References

[1] R. Baker, G. Harman, and J. Pintz. The difference between consecutive primes, II. In *Proceedings of the London Mathematical Society*, 83 (3): 532–562, 2001.

[2] L. Barenboim. Deterministic $(\Delta + 1)$-Coloring in Sublinear (in Δ) Time in Static, Dynamic and Faulty Networks. *Journal of the ACM*, 63(5) : 47, 2016.

[3] A. Balliu, S. Brandt, J. Hirvonen, D. Olivetti, M. Rabie And J. Suomela. Lower bounds for maximal matchings and maximal independent sets. *Arxiv*, 2019.

[4] L. Barenboim, and M. Elkin. Distributed $(\Delta + 1)$- coloring in linear (in Δ) time. In *Proc. of the 41st ACM Symp. on Theory of Computing*, pp. 111-120, 2009.

[5] L. Barenboim, and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proc. 29th ACM Symp. on Principles of Distributed Computing*, pages 410-419, 2010.

[6] L. Barenboim, and M. Elkin. Distributed deterministic edge coloring using bounded neighborhood independence. In *Proc. of the 30th ACM Symp. on Principles of Distributed Computing*, pages 129 - 138, 2011.

[7] L. Barenboim, and M. Elkin. Distributed Graph Coloring: Fundamentals and Recent Developments. *Morgan and Claypool*, 2013.

[8] L. Barenboim, M. Elkin, and F. Kuhn. Distributed (Delta+1)-Coloring in Linear (in Delta) Time. *SIAM Journal on Computing*, 43(1): 72-95, 2014.

[9] L. Barenboim, M. Elkin, T. Maimon. Deterministic Distributed $(\Delta + o(\Delta))$-Edge-Coloring, and Vertex-Coloring of Graphs with Bounded Diversity. In *Proc. of the 36th ACM Symp. on Principles of Distributed Computing*, pages 175-184, 2017.

[10] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. In *Proc. of the 53rd Annual Symp. on Foundations of Computer Science*, pages 321-330, 2012.

[11] L. Barenboim, M. Elkin U. Goldenberg. Locally-Iterative Distributed $(\Delta + 1)$-Coloring below Szegedy-Vishwanathan Barrier, and Applications to Self-Stabilization and to Restricted-Bandwidth Models *https://arxiv.org/pdf/1712.00285.pdf*

[12] R. Cole, and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[13] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of the ACM*, 17 (11): 643–644, 1974.

[14] S. Dolev. Self-Stabilization. *MIT Press*, 2000.

[15] S. Dolev, and T. Herman. Superstabilizing Protocols for Dynamic Distributed Systems. *Chicago J. Theor. Comput. Sci.* 1997.

[16] D. Dubhashi, D. Grable, and A. Panconesi. Nearly-optimal distributed edge-colouring via the nibble method. *Theoretical Computer Science, a special issue for the best papers of ESA95*, 203(2):225–251, 1998.

[17] M. Elkin, S. Pettie, and H. Su. $(2\Delta - 1)$-Edge-Coloring is Much Easier than Maximal Matching in the Distributed Setting. In *Proc. of the 26th ACM-SIAM Symp. on Discrete Algorithms*, pages 355-370, 2015.

[18] M. Fischer, M. Ghaffari, and F. Kuhn . Deterministic Distributed Edge Coloring via Hypergraph Maximal Matching. To appear in *58th Annual Symp on Foundations of Computer Science*, 2017.

[19] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local Conflict Coloring. In *Proc. of the 57th Annual Symp. on Foundations of Computer Science*, pages 625 - 634, 2016.

[20] D. Grable, and A. Panconesi. Nearly optimal distributed edge colouring in O(log log n) rounds. *Random Structures and Algorithms*, 10(3): 385-405, 1997.

[21] A. Goldberg, and S. Plotkin. Parallel $(\Delta + 1)$-Coloring of Constant-Degree Graphs. *Inf. Process. Lett.* 25(4): 241-245, 1987.

[22] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

[23] N. Guellati, and H. Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *Journal of Parallel and Distributed Computing*, 70(4): 406-415, 2010.

[24] G. Hardy, and E. Wright. An introduction to the theory of numbers. *Oxford university press*, 5th edition, 1980.

[25] D. Hefetz, F. Kuhn, Y. Maus, and A. Steger. Polynomial Lower Bound for Distributed Graph Coloring in a Weak LOCAL Model. In *Proc. of the 30th International Symp. on DiStributed Computing*, pages 99 - 113, 2016.

[26] T. Herman. Self-stabilization bibliography: Access guide. *Chicago Journal of Theoretical Computer Science*, Working Paper WP-1, 2002.

[27] S.C. Hsu, S.T. Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters*, 43 (2):7781, 1992 .

[28] M. Ikeda, S. Kamei, and H. Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. *In Proc. 3rd International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2002.

[29] A. Kosowski, L. Kuszner. Self-stabilizing algorithms for graph coloring with improved performance guarantees. In *Proc. 8th International Conference on Artificial Intelligence and Soft Computing*, pages 1150-1159 2006.

[30] K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. In *Proc. of the 20th International Parallel and Distributed Processing Symp.*, 2006.

[31] F. Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proc. of the 21st ACM Symp. on Parallel Algorithms and Architectures*, pages 138–144, 2009.

[32] F. Kuhn, and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th ACM Symp. Principles of Distributed Computing*, pp. 7–15, 2006.

[33] N. Linial. Distributive graph algorithms: Global solutions from local data In *Proc. 28th Symp. on Foundation of Computer Science*, pp. 331–335, 1987.

[34] L. Lovasz. On decompositions of graphs. *Studia Sci. Math. Hungar.*, 1:237–238, 1966.

[35] Y. Métivier, J. M. Robson, N. Saheb-Djahromi and A. Zemmari An optimal bit complexity randomized distributed MIS algorithm. *Distrib. Comput.*, 23:331–340, 2011.

[36] M. Naor, and L. Stockmeyer. What can be computed locally? In *Proc. 25th ACM Symp. on Theory of Computing*, pages 184-193, 1993.

[37] S. Pai, G. Pandurangan, S. Pemmaraju, T. Riaz, and P. Robinson. Symmetry Breaking in the Congest Model: Time- and Message-Efficient Algorithms for Ruling Sets. https://arxiv.org/abs/1705.07861

[38] A. Panconesi, and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.

[39] A. Panconesi, and A. Srinivasan. Randomized Distributed Edge Coloring via an Extension of the Chernoff-Hoeffding Bounds. *SIAM Journal on Computing*, 26(2):350-368, 1997.

[40] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[41] S. Sur, and P.K. Srimani. A self-stabilizing algorithm for coloring bipartite graphs. *Information Sciences*, 69, pages 219-227, 1993 .

[42] M. Szegedy, and S. Vishwanathan. Locality based graph coloring. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 201-207, 1993.

[43] V. Vizing. On an estimate of the chromatic class of a p-graph. *Metody Diskret. Analiz*, 3: 25-30, 1964.

1 תקציר

בעבודת תזה זו שעשיתי בהדרכתו של דוקטור לאוניד ברנבוים ובתמיכה רבה של פרופסור מיכאל אלקין מאוניברסיטת בן גוריון אני חוקר את בעית צביעת הגרפים בעזרת $\Delta + 1$ צבעים. המודל המתמטי שהתבססתי עליו הוא מודל צביעה מבוזר בו זמני הריצה נמדדים בסיבובים . בכל סיבוב כל קודקוד יכול לשלוח הודעה אחת (בגודל קבוע או לא מוגבל) לכל אחד משכניו בגרף. ,תוצאות העבודה משפרת את התאוריה שנכתבה בשנת 1993 על ידי זגדי ווישונטאן שהניחו שכל אלגוריתם איטרטיבי מקומי (תכונה של אלגוריתם אשר שומר על אינוריאנטה ומתכנס לתשובה נכונה). דורש $\Omega(\Delta \log \Delta + \log^* n)$ סיבובים על מנת להגיע לצביע תקינה בעלת $\Delta + 1$ צבעים, עם סייג, שקיימת צביעה מאד מיוחדת שיכולה לשפר את יעילות הצביעה. עד המאמר הנ"ל לא הצליחו למצוא צביעת קודקודים דטרמיניסטית שמשפרת את יעילות הצביעה אך במאמר הנוכחי הוכחתי לא רק שניתן לצבוע בצביעה תקינה בעזרת $\Delta + 1$ צבעים בסיבוכיות של $O(\Delta)$ סיבובים. אלא גם בעזרת הודעות של ביט אחת בכל סיבוב. התוצאות משפרות תוצאות ידועות רבות בינהם

* שיפור זמן הריצה בצביעת קודקודים ללא הגבלות.

בעזרת הטכניקות שאתאר ניתן לצבוע גרף כללי ב $(O\Delta)$ צביעה בזמן $O\sqrt{\Delta} + \log^* n$ סיבובים

* שיפור תוצאות של צביעה פגומה עם אוריאנטציה.

בעזרת הטכניקות שפיתחתי ניתן לפתור את בעיית צביעה פגומה בתוצאה של $O(\sqrt{\Delta})$ צביעה $O(\sqrt{\Delta})$ פגומה בזמן $O\sqrt{\Delta} + \log^* n$ סיבובים.

* שיפור בנצילות הרשת באלגוריתמים לצביעת צלעות.

הטכניקות מאפשרות צביעת צלעות ב $(2\Delta - 1)$ צבעים בזמן $O(\Delta + \log n)$ סיבובים בעזרת שימוש בביט אחד בכל סיבוב.

* איפשרו פיתוח של אלגוריתמים מתכנסים עצמית.

פיתחתי אלגוריתמים מתכנסים עצמית לצביעת צלעות ב $(2\Delta - 1)$, קבוצה בלתי תלויה מקסימלית, לבעיית השידוך המקסימלי, לצביעת צלעות ב $(2\Delta - 1)$ צבעים ולקבוצה בלתי תלויה מקסימלית כאשר כולם בזמן ריצה מבוזר של $O(\Delta + \log^* n)$ סיבובים.

2 תוכן עניינים

האוניברסיטה הפתוחה
המחלקה למתמטיקה ומדעי המחשב

צביעה מבוזרת מקומית איטרטיבית בעזרת $\Delta + 1$ צבעים מתחת לחסם זגדי ווישונטאן ושימושים באלגוריתמים מתכנסים עצמית וברשתות בעלות רוחב פס מוגבל

מאת
אורי גולדנברג

התזה מוגשת כחלק מדרישות
תואר "מוסמך מדעים" M.Sc
במדעי המחשב
המחלקה למתמטיקה ולמדעי המחשב

התזה הוכנה בהדרכתו של
פרופ' לאוניד ברנבוים

21 בדצמבר 2020