# Distributed Edge Coloring in Time Polylogarithmic in $\Delta$

Alkida Balliu
alkida.balliu@gssi.it
Gran Sasso Science Institute
L'Aquila, Italy

Sebastian Brandt
brandt@cispa.de
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany

Fabian Kuhn
kuhn@cs.uni-freiburg.de
University of Freiburg
Freiburg, Germany

Dennis Olivetti
dennis.olivetti@gssi.it
Gran Sasso Science Institute
L'Aquila, Italy

## ABSTRACT

We provide new deterministic algorithms for the edge coloring problem, which is one of the classic and highly studied distributed local symmetry breaking problems. As our main result, we show that a $(2\Delta-1)$-edge coloring can be computed in time $\text{poly}\log\Delta + O(\log^* n)$ in the LOCAL model. This improves a result of Balliu, Kuhn, and Olivetti [PODC '20], who gave an algorithm with a quasi-polylogarithmic dependency on $\Delta$. We further show that in the CONGEST model, an $(8+\varepsilon)\Delta$-edge coloring can be computed in $\text{poly}\log\Delta + O(\log^* n)$ rounds. The best previous $O(\Delta)$-edge coloring algorithm that can be implemented in the CONGEST model is by Barenboim and Elkin [PODC '11] and it computes a $2^{O(1/\varepsilon)}\Delta$-edge coloring in time $O(\Delta^{\varepsilon} + \log^* n)$ for any $\varepsilon \in (0, 1]$.

## CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

## KEYWORDS

edge coloring, LOCAL model, CONGEST model, token dropping

**ACM Reference Format:**
Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. 2022. Distributed Edge Coloring in Time Polylogarithmic in $\Delta$. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC '22), July 25–29, 2022, Salerno, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3519270.3538440

## 1 INTRODUCTION

In the most standard setting of distributed graph algorithms, we are given a network that is modeled as an undirected graph $G = (V, E)$. The nodes $V$ are the active entities of the network and communication happens by exchanging messages over the edges $E$ in synchronous rounds. The goal is to solve some graph problem on $G$. This distributed computation model is known as the LOCAL model if the communication between neighbors in each round is not restricted and it is known as the CONGEST model if the messages exchanged neighbors have to consist of at most $O(\log n)$ bits (where $n = |V|$) [41, 47]. Four graph problems that have received particular attention in this context are the problems of computing a maximal independent set (MIS) of $G$, a $(\Delta+1)$-vertex coloring of $G$ (where $\Delta$ is the maximum degree of $G$), a maximal matching of $G$, and a $(2\Delta-1)$-edge coloring of $G$. All four problems have in common that they can be solved by a trivial sequential greedy algorithm. The four problems can be seen as prototypical examples of distributed symmetry breaking problems and understanding the distributed complexities of them has been at the very core of the area of distributed graph algorithms, e.g. [9, 29, 41]. All four problems can be solved by quite simple and straightforward $O(\log n)$-round randomized distributed algorithms, which have been known for more than thirty years [1, 37, 41, 42]. In light of those efficient randomized algorithms, a lot of the work concentrated on developing deterministic distributed algorithms for the four problems. As the contributions of the present paper are on deterministic algorithms, we also focus on deterministic algorithms when discussing the prior work in the following.

*Deterministic Complexity as a Function of the Network Size.* In [2, 45], a tool called network decomposition was introduced as a generic technique to solve distributed graph problems. This resulted in deterministic $2^{O(\sqrt{\log n})}$-round algorithms in particular for the four problems discussed above and it left open the question of whether the problems can also be solved deterministically in polylogarithmic time and thus similarly fast as with the simple randomized algorithms of the 1980s. This was first shown for the maximal matching problem in [33, 34], where the authors gave a deterministic $O(\log^4 n)$-time algorithm. The result was later improved to the current running time of $O(\log^2\Delta\cdot\log n)$ in [22]. Much more recently, it was shown that also the $(2\Delta-1)$-edge coloring problem can also be solved in polylogarithmic time deterministically [23, 27, 35] by reducing the problem to the problem of computing a maximal matching in 3-uniform hypergraphs and by giving a polylogarithmic-time deterministic distributed algorithm for maximal matching in hypergraphs of bounded rank. The algorithm of [35] achieves a round complexity of $\tilde{O}(\log^2\Delta\cdot\log n)$ and is thus almost as fast as the maximal matching algorithm of [22]. Subsequently, in a breakthrough result, Rozhoň and Ghaffari [48] gave a polylogarithmic-time algorithm for the network decomposition problem. The current best version of this algorithm in [26] implies $O(\log^5 n)$-time deterministic distributed algorithms


*PODC '22, July 25–29, 2022, Salerno, Italy*

ACM ISBN 978-1-4503-9262-4/22/07…$15.00
https://doi.org/10.1145/3519270.3538440

for all four problems discussed above. Finally, in a recent paper [28], a direct $O(\log^2 \Delta \cdot \log n)$-round algorithm for the $(\Delta+1)$-vertex coloring (and thus also for the $(2\Delta-1)$-edge color problem) was presented.

*Deterministic Complexity as a Function of the Maximum Degree.* The optimal time complexity of a graph problem in the LOCAL model can be interpreted as the *locality* of the problem in the following sense. If there is an $R$-round deterministic LOCAL algorithm for solving a problem in a graph $G$, then every node of $G$ can compute its output as a function of its $R$-hop neighborhood in $G$ and if more than $R$ rounds are needed to solve a problem, then some node must learn something about the graph that is outside the node's $R$-hop neighborhood [41, 47]. The local neighborhood of a node is in principle independent of the size of the network. It is therefore natural to ask for the locality of graph problems not just as a function of the network size, but also as a function of more local properties. Further, local graph algorithms are particularly interesting in large networks where the node degrees might be independent or almost independent of the network size. As a result, there is an extended body or prior work that tries to understand the distributed complexity of graph problems as a function of the maximum degree $\Delta$ rather than as a function of $n$.

Note however that the distributed complexity of many problems cannot be completely independent of $n$. Linial [41] showed that even in networks of maximum degree $\Delta = 2$, computing a coloring with $O(1)$ colors (and by simple reductions computing solutions for all four classic problems discussed above) requires at least $\Omega(\log^* n)$ rounds.[1] However, in [41], it is also shown that in $O(\log^* n)$ rounds, one can compute a vertex coloring with $O(\Delta^2)$ colors for any graph $G$. Given such a coloring, one can then iterate through the color classes and obtain simple distributed $O(\Delta^2)$-round implementations of the natural sequential greedy algorithms for computing an MIS, a maximal matching, a $(\Delta+1)$-vertex coloring, or a $(2\Delta-1)$-edge coloring. All four problems can therefore be solved in $O(\Delta^2+\log^* n)$ rounds in the LOCAL model (and also in the CONGEST model). That is, we can solve the problems in time $O(f(\Delta)+\log^* n)$ for some function $f$. This keeps the dependency on $n$ as small as it can be. When establishing the local complexity of a distributed graph problem, we are interested in optimizing the function $f$ and thus the $\Delta$-dependency of this bound.

Starting from the early work on distributed graph algorithms, there is a long line of research that tries to optimize the $\Delta$ dependency of the four discussed problems [5–8, 10, 11, 24, 31, 39–41, 43, 44, 51]. As given a $C$-vertex or a $C$-edge coloring, all four problems can be solved in $C$ rounds, the primary focus was on developing efficient coloring algorithms. In a first phase, the time for computing a $(2\Delta-1)$-edge coloring [44] and for computing a $(\Delta+1)$-vertex coloring was improved to $O(\Delta+\log^* n)$ [11, 38]. As a result, we therefore also obtain $O(\Delta+\log^* n)$-round algorithms for MIS and maximal matching. In [3, 4, 15], it was shown that for MIS and maximal matching, this bound is tight, even on tree networks. More specifically, it was shown that there is no randomized MIS or maximal matching algorithm with round complexity $o\big(\Delta+\frac{\log\log n}{\log\log\log n}\big)$ and there is no deterministic such algorithm with round complexity $o\big(\Delta+\frac{\log n}{\log\log n}\big)$. While for MIS and matching, there is a linear-in-$\Delta$ lower bound, for $(\Delta+1)$-vertex coloring and $(2\Delta-1)$-edge coloring, there are in fact algorithms with a sublinear-in-$\Delta$ complexity. This was first shown by Barenboim in [6]. The best known algorithm that works for both vertex and edge coloring has a time complexity of $O(\sqrt{\Delta\log\Delta}+\log^* n)$ [10, 24, 43]. For $(2\Delta-1)$-edge coloring, it has recently been shown that we can even obtain a dependency on $\Delta$ that is subpolynomial in $\Delta$. First, Kuhn [39] showed that the problem can be solved in $2^{O(\sqrt{\log\Delta})}+O(\log^* n)$ rounds and subsequently, Balliu, Kuhn, and Olivetti [5] showed that the number of rounds can even be reduced to $\log^{O(\log\log\Delta)}\Delta+O(\log^* n)$ and thus to a quasi-polylogarithmic dependency on $\Delta$. This leaves a natural open question.

*Is it possible to solve $(2\Delta-1)$-edge coloring in time polylogarithmic in $\Delta$?*

*Our Contribution.* As our main result, we resolve the above open question and prove the following theorem.

> Theorem 1.1. *There is a deterministic* $\operatorname{poly}\log\Delta+O(\log^* n)$*-round* LOCAL *algorithm to solve the* $(2\Delta-1)$*-edge coloring problem.*

In fact, as we will prove Theorem 1.1 also for the more general $(degree+1)$-list edge coloring problem, as long as all the colors come from a domain of size at most $\operatorname{poly}(\Delta)$. In this problem, initially, every edge $e$ is given a list consisting of at least $\deg_G(e)+1$ different colors, where the degree $\deg_G(e)$ of $e$ is defined as the number of edges that are adjacent to $e$. The output must be a proper edge coloring such that every edge $e$ is colored with some color from its list. In addition, we also provide a more efficient edge coloring algorithm for the CONGEST model. The best known CONGEST algorithm for computing a $(2\Delta-1)$-edge coloring (as a function of $\Delta$) has a round complexity of $O(\Delta+\log^* n)$ [10]. In [8], it was further shown that for any $\varepsilon\in(0,1]$, a $2^{O(1/\varepsilon)}\cdot\Delta$-edge coloring can be computed in $O(\Delta^\varepsilon+\log^* n)$ rounds in the CONGEST model.[2] We improve this by showing that an $O(\Delta)$-edge coloring can be computed in polylogarithmic (in $\Delta$) time.

> Theorem 1.2. *For any constant $\varepsilon>0$, there is a deterministic* $\operatorname{poly}\log\Delta+O(\log^* n)$*-round* CONGEST *algorithm to compute an $(8+\varepsilon)$-edge coloring.*

*Further Related Work.* There is also substantial work on randomized algorithms for computing graph colorings [12, 17, 21, 32, 36, 46, 49], MIS [12, 25], and maximal matchings [12]. The best known randomized complexities are $O(\log^3\log n)$ for $(\Delta+1)$-vertex coloring and $(2\Delta-1)$-edge coloring [17, 21, 28], $O(\log\Delta+\log^5\log n)$ for MIS [25, 26], and $O(\log\Delta+\log^3\log n)$ for maximal matching [12, 22]. For the edge coloring problem, which is the main focus of the present paper, there have also been various results on finding edge colorings with less than $2\Delta-1$ colors. Note first that it is not possible

[1] Note that even in bounded-degree graphs, the local neighborhoods are not completely independent of the network size. In order to equip the network with unique identifiers, the space from which the identifiers are chosen has to grow as a function of the network size $n$.

[2] The authors of [8] do not explicitly show that their algorithm works in the CONGEST model. It is however not hard to see that it can be adapted to work in the CONGEST model.

to compute such a coloring in time $O(f(\Delta)+\log^* n)$. In [18] (based on techniques developed in [13, 16]), it was shown that for every $\Delta > 1$, every deterministic algorithm for computing a $(2\Delta-2)$-edge coloring of $\Delta$-regular trees requires at least $\Omega(\log_\Delta n)$ rounds and every randomized such algorithm requires at least $\Omega(\log_\Delta \log n)$ rounds. When computing an edge coloring with less than $2\Delta-1$ colors, the objective therefore is on the one hand to obtain a coloring that uses not many more than $\Delta$ colors and on the other hand to achieve a time complexity that gets as close as possible to the lower bounds of [18]. In recent years, different distributed algorithms that compute edge colorings with $(1+\varepsilon)\Delta$ colors (and even with $\Delta + O(1)$ colors) have been developed [18, 20, 21, 30, 50]. For example, by using the randomized algorithm of [18] together with derandomization techniques of [27, 48], one obtains deterministic poly $\log n$-time and randomized poly $\log\log n$-time algorithms for computing edge colorings with $\Delta + \tilde{O}(\sqrt{\Delta})$ colors.

## 2 MODEL AND DEFINITIONS

*Basic notions.* Leg $G = (V, E)$ be a graph. We denote with $\Delta$ the maximum degree of $G$, and with $\bar{\Delta}$ the maximum degree of the line graph of $G$, that is, the maximum number of neighboring edges of an edge. Clearly, $\bar{\Delta} \leq 2\Delta - 2$. We denote with $\deg_G(v)$ the degree of a node $v \in V$, and for an edge $e \in E$ we denote with $\deg_G(e)$ the degree of edge $e$ in the line graph of $G$, that is, for $e = \{u, v\}$, $\deg_G(e) = \deg_G(u) + \deg_G(v) - 2$. If $G$ is a directed graph, we use $\deg_G(v)$ and $\deg_G(e)$ to denote the degrees of $v$ and $e$ in the undirected version of $G$. If the graph is clear from the context, we may omit it and write $\deg(v)$ and $\deg(e)$. We assume that $\log n$ denotes $\log_2 n$.

*List Edge Coloring.* Assume that for a graph $G = (V, E)$, we are given a set $C = \{1, \ldots, |C|\}$ of colors, called the color space, and for each edge $e = \{u, v\}$, we are given a list of colors $L_e \subseteq C$. The list edge coloring asks to assign a color $c_e \in L_e$ to each edge $e$ such that edges incident to the same node are assigned different colors. In the distributed version of the problem, we assume that all nodes know $C$, both nodes of an edge know the list $L_e$ and at the end, both nodes need to output the color of $e$. The $(degree + 1)$-list edge coloring problem is a special case where $|L_e| \geq \deg_G(e) + 1$ for all $e \in E$. The standard $K$-edge coloring is a special of the list edge coloring problem in which every edge $e$ is given the set $L_e = \{1, \ldots, K\}$ as its list.

*Relaxed List Edge Coloring.* In this work, we will make use of a technique that allows us to decompose a hard list coloring instance into many easier ones. This technique has been already used in [5, 6, 24, 39]. A list edge coloring instance can be characterized by a parameter $S$, specifying how much larger the lists are as compared to the degree of the edges. A list edge coloring instance is said to have slack at least $S$ if $|L_e| > S \cdot \deg(e)$, for all $e$. We define $P(\bar{\Delta}, S, C)$ to be the family of list edge coloring instances where the graph has maximum edge degree $\bar{\Delta}$, the slack is at least $S$, and the color space has size $C$. We define $T(\bar{\Delta}, S, C)$ to be the time required to solve $P(\bar{\Delta}, S, C)$.

*Defective Coloring.* A $d$-defective $c$-coloring is an assignment of colors from $\{1, \ldots, c\}$ to the nodes, such that the maximum degree of each graph induced by nodes of the same colors is bounded by $d$. The $d$-defective $c$-edge coloring of $G$ is a $d$-defective $c$-coloring of the line graph of $G$.

LOCAL *and* CONGEST *Model.* We consider two standard models of distributed computing, the LOCAL and the CONGEST model [41, 47]. In the LOCAL model, the network is modeled as a graph $G = (V, E)$, where the nodes $V$ represent computational entities and the edges $E$ represent pairwise communication links. Communication proceeds in synchronous rounds, where in each round, each node can send (possibly different) messages to its neighbors, receive messages from the neighbors, and perform some internal computation. We do not restrict the message size or the interal computational power of the nodes.

At the beginning of the computation, each node knows a unique identifier from $\{1, \ldots, \text{poly}\, n\}$, where $n = |V|$ is known to all nodes. Further, each node knows $\Delta$, the maximum degree of $G$. At the end of a computation, each node must produce its own part of the solution (e.g., in the case of the edge coloring problem, each node $v$ must output the colors of all incident edges). The time complexity of an algorithm is the worst-case of number of rounds required to produce the solution. We may express this running time as a function of $n$ and $\Delta$. The CONGEST model is similar to the LOCAL model, with the only difference that each message must have size bounded by $O(\log n)$ bits.

## 3 ROAD MAP AND HIGH LEVEL IDEAS

In this section, we provide an overview of the main ingredients that we use to solve the edge coloring problem. While our algorithms do not work exactly as stated here, we still try to highlight the core ideas that we use. For this overview, we first sketch how to obtain an edge coloring with $O(\Delta)$ colors.

At a very high level, our algorithm uses a divide-and-conquer approach that has been used in various previous deterministic distributed coloring algorithms [5, 7, 8, 11, 38, 39]. The algorithm uses a defective coloring to divide the graph into subgraphs of smaller degree and at the same time the global space of colors is divided into the same number of parts so that the different low-degree subgraphs can use disjoint color spaces. The colorings or the different subgraphs are then colored recursively in parallel. There are different challenges that we face with this high-level approach. First, when computing a defective coloring with defect $d$, all previous algorithms use a number of colors that is at best $c \cdot \Delta/d$ for some constant $c > 1$. Because of this, the ratio between the necessary number of colors and the maximum degree grows by a factor $c$ for every recursion level. To keep the total number of colors needed in the end moderately small, this requires to keep the number of recursion levels small, which in return makes the defective coloring steps more expensive. As the main technical contribution of this paper, we obtain a new algorithm for defective edge coloring, where we can keep the number of colors and the reduction of the degree arbitrarily close to 1.

Concretely, for the special case of 2-colored bipartite graphs, we obtain a defective 2-coloring algorithm with the guarantee that the defect of every edge $e$ is only $(1/2 + \varepsilon) \cdot \deg(e)$ for an arbitrary parameter $\varepsilon > 0$ (as long as the defect of the edge does not fall below some threshold). We can compute this defective 2-coloring in time $\text{poly}(\log(\Delta)/\varepsilon)$. By choosing $\varepsilon \leq \varepsilon'/\log\Delta$ and recursively

applying the defective 2-coloring, we show that we can partition the graph into $2^k = \Delta/\text{poly}\log\Delta$ parts such that the defect of each node $v$ is at most $(1+O(\varepsilon'))\cdot \deg_G(e)/2^k$. We can thus compute a $(2+\varepsilon')\Delta$-edge coloring of a given 2-colored bipartite graph in time $\text{poly}(\log(\Delta)/\varepsilon')$. We further give a reduction that allows to use this algorithm to color general graphs with $(8+\varepsilon)\Delta$ colors in time $\text{poly}\log\Delta + O(\log^* n)$.

We next describe the high-level idea of our defective 2-coloring algorithm. Assume that we are given a bipartite graph $G = (U \cup V, E)$, where the nodes know if they are in $U$ or in $V$. For simplicity, assume further that $G$ is $\Delta$-regular. Our goal is to color each edge either red or blue such that every red edge has at most $(1+\varepsilon)\Delta$ adjacent red edges and every blue edge has at most $(1+\varepsilon)\Delta$ adjacent blue edges. To achieve this, we generalize an idea that was presented in [14]. There, the authors show how to efficiently solve a problem known as locally optimal semi-matching [19] and in particular a special case of this problem, a so-called stable edge orientation of a graph. Given an edge orientation of a graph $G$, for every node $v$, let $x_v$ denote the number of incident edges that are oriented towards $v$. An edge orientation is called stable if for every edge $e = \{u,v\}$, if $e$ is oriented from $u$ to $v$, then $x_v - x_u \leq 1$ (and otherwise $x_u - x_v \leq 1$). Note that such a stable orientation of a $\Delta$-regular bipartite graph $G = (U \cup V, E)$ directly gives a perfect defective 2-coloring. Assume that every edge that is oriented from $U$ to $V$ is colored red and every edge that is oriented from $V$ to $U$ is colored blue. If an edge $\{u,v\}$ for $u \in U$ and $v \in V$ is oriented from $u$ to $v$ and thus colored red, then the number of adjacent red edges is exactly $(x_v - 1) + (\Delta - x_u - 1) = \Delta + (x_v - x_u) - 2 \leq \Delta - 1$ (by using that $x_v - x_u \leq 1$). Note that the degree of every edge in a $\Delta$-regular graph is exactly $2(\Delta-1)$. An analogous argument works for blue edges.

In [14], the authors introduce a tool that they call the *token dropping game* and which is used in particular to compute stable edge orientations. The token dropping game works as follows. We are given a directed graph where each node holds either 0 or 1 tokens. The goal is to move tokens around and reach a "stable" solution, that is a solution in which tokens cannot move. A token can move from node $u$ to node $v$ if $u$ has a token, $v$ has no tokens, and the edge $(u,v)$ exists. Every time a token passes through an edge, the edge is deleted. In [14], it is shown that the token dropping game can be solved in time $O(L\cdot\Delta^2)$ if the directed graph of the game has no cycles and the longest directed path is of length $L$. It is shown that this algorithm can be used to compute a stable edge orientation and thus a perfect defective 2-coloring in $\Delta$-regular 2-colored bipartite graphs in $O(\Delta^4)$ rounds.

We cannot directly apply the algorithm of [14] for several reasons. First, a time complexity of $O(\Delta^4)$ is way too slow for us as we aim for algorithms that run in time polylogarithmic in $\Delta$. Second, we need to deal with non-regular graphs. Even if at the beginning the graph is regular, after a single application of defective 2-coloring, we might end up with two very non-regular graphs.[3] In non-regular graphs, a stable orientation does not lead to a good defective 2-coloring. The second problem can be solved relatively easily. For each edge $e = \{u,v\}$, one can just define a different threshold $\eta_e$ such that if $e$ is oriented from $u$ to $v$, it must hold that $x_v - x_u \leq \eta_e + 1$ and if $e$ is oriented from $u$ to $v$, it must hold that $x_u - x_v \leq -\eta_e + 1$. If the thresholds $\eta_e$ are chosen in the right way, we can still get a perfect defective 2-coloring from such an edge orientation and one can also still use the reduction in [14] to the token dropping game to compute such an edge orientation. Reducing the time complexity is more challenging. For this, we relax the requirement on the orientation. For $\Delta$-regular graphs, we now want to require that if an edge $\{u,v\}$ is oriented from $u$ to $v$, then $x_v - x_u \leq \varepsilon\Delta$ and otherwise $x_u - x_v \leq \varepsilon\Delta$. Such an orientation can be computed fast if we have a fast algorithm to a generalized token dropping game, where each node can have up to $O(\varepsilon\Delta)$ tokens. For more details of how to define the generalized token dropping game, we refer to Section 4.

As long as the directed graph of the token dropping game has no cycles and only short directed paths, it is in fact possible to adapt the token dropping algorithm and the analysis of [14] (in a non-trivial way) to obtain a $\text{poly}(\log(\Delta)/\varepsilon)$-round algorithm to compute a defective 2-coloring in a 2-colored bipartite graph $G$ such that the defect of any edge $e$ of $G$ is at most $(1+\varepsilon)\deg_G(e)$ and this is sufficient to obtain an $(8+\varepsilon)\Delta$-edge coloring algorithm with a time complexity of $\text{poly}\log\Delta + O(\log^* n)$. In order to obtain a $(2\Delta-1)$-coloring, there is however still one major challenge remaining. When recursively using defective colorings, we always have to slightly relax the coloring problem along the way and end up with more that $2\Delta-1$ colors. We can use such a more relaxed edge coloring to compute a $(2\Delta-1)$-edge coloring or even a $(degree+1)$-edge coloring if we solve the more general list edge coloring [6, 24]. In this case, one however also has to use a generalized version of defective coloring [5, 39] (for a definition of what we need, see Section 5). Even for this generalized defective 2-coloring variant, one can define appropriate conditions for the required edge orientation and one can use our generalized token dropping game to compute such an edge orientation. However, in this case, the resulting token dropping graph can have cycles and therefore, even the adapted variant of the token dropping algorithm of [14] does not work. In Section 4, we therefore design a completely new token dropping algorithm, which also works in general directed graphs (for the relaxed token dropping game that we are using).

The remainder of the paper is organized as follows. In Section 4, we introduce and solve the generalized token dropping game. In Section 5, we then introduce our generalized defective 2-coloring problem and we show how to solve it by using our token dropping algorithm of Section 4. Finally, in Section 6 and Section 7, we prove our main results. We start in Section 6 with the $O(\Delta)$-edge coloring algorithm for the CONGEST model, which is conceptually simpler. Then, in Section 7, we present our algorithm for solving the $(\deg(e)+1)$-list edge coloring problem in the LOCAL model.

## 4 THE GENERALIZED TOKEN DROPPING GAME

A key technical tool that we use is a generalization of the token dropping game of [14]. This game is defined on a directed graph $G = (V,E)$, and it has an integer parameter $k \geq 1$. Initially, each node $v \in V$ receives at most $k$ tokens as input. Each edge $e \in E$ can be either active or passive. Initially, all edges are active, and

[3] Note that even if the new edge degrees are at most $\Delta - 1$, the node degrees can still take arbitrary values between 0 and $\Delta$.

as the game progresses, edges can become passive. In a sequential execution, the game proceeds in steps, where in each step, on some active edge $(u, v) \in E$ on which $u$ has at least 1 token and $v$ has less than $k$ tokens, one token can be moved from $u$ to $v$. After moving the token, the edge $(u, v)$ becomes passive. Passive edges cannot become active again. That is, over each edge of $G$, at most one token can be moved, and an edge is passive if and only if a token was moved over the edge. Let $\tau(v)$ be the number of tokens at a node $v$ at a given time. When the game ends, it must hold that $\tau(v) \leq k$ for every $v \in V$ and

$$\forall e = (u, v) \in E : e \text{ active} \implies \tau(u) \leq \tau(v) + \sigma(e), \tag{1}$$

where $\sigma(e) \geq 0$ is a value that specifies how much slack we tolerate on a given edge $e$. In the original token dropping game introduced in [14], $k = 1$ and $\sigma(e) = 0$ for all edges. In [14], it was further assumed that the graph $G$ is organized in layers and that all edges are oriented from higher to lower layers (therefore the name token dropping). We here generalize the game by allowing general diriected graphs, larger values of $k$, and by tolerating some slack on the active edges.

## 4.1 Distributed Token Dropping Algorithms

In the distributed version of the (generalized) token dropping game the goal is to run an execution of the game, where tokens are moved in parallel, but which is still equivalent to the sequential definition of the game. That is, over each edge, at most one token can be moved and at all times, and every node has a set of at most $k$ tokens.

We next describe a distributed algorithm to solve the generalized token dropping game with $\sigma(e) = \varepsilon \cdot \deg_G(e)$ for some given parameter $\varepsilon \in (0, 1/2]$, that is, by allowing a slack proportional to the edge degree. The algorithm has an integer parameter $\delta > 0$, and we will see that $\delta$ can be used to control the trade-off between the round complexity and the slack of the algorithm. The smaller $\delta$ is chosen, the smaller $\varepsilon$ can be chosen, however the time of the algorithm depends linearly on $1/\delta$.

Throughout the algorithm, some tokens are active and some tokens are passive. Initially, all tokens are active, and once a token becomes passive, it remains passive and cannot be moved anymore. The algorithm operates in synchronous phases. We use $x_v(t)$ and $y_v(t)$ to denote the number of active and passive tokens of node $v$ at time $t$, i.e., at the end of phase $t$. The value $x_v(0)$ and $y_v(0)$ denote the number of tokens of $v$ at the beginning. At all times $t \geq 0$, the algorithm always guarantees that $x_v(t) + y_v(t) \leq k$ for every node $v \in V$. We further have $y_v(0) = 0$ for all $v \in V$. For each node, we further define an integer parameter $\alpha_v \geq 1$ that controls how much slack node $v$ is willing to tolerate on its edges. We will see that if we want to tolerate slack $\sigma_e$ on an edge $\{u, v\}$, then we in particular have to choose $\alpha_u, \alpha_v \leq c\sigma_e$ for a sufficiently small constant $c$. The value of $\alpha_v$ controls how much slack node $v$ is willing to tolerate on its edges. In the end, the slack on each edge $(u, v)$ that is still active has to be $O(\alpha_u + \alpha_v)$. The algorithm is run for $\lfloor \frac{k}{\delta} \rfloor - 1$ phases. In each phase $t \geq 1$, the algorithm does the following steps.

(1) Define $A(t) \subseteq V$ as the set of nodes $v \in V$ with $x_v(t-1) \geq \alpha_v + \delta$. We call $A(t)$ the active nodes in phase $t$ and only nodes in $A(t)$ will be able to move tokens to other nodes in phase $t$.
(2) Each node $v \in A(t)$ sets $x'_v(t) := x_v(t-1) - \delta$ and $y'_v(t) := y_v(t-1) + \delta$. All other nodes $v \in V \setminus A(t)$ set $x'_v(t) := x_v(t-1)$ and $y'_v(t) := y_v(t-1)$.
(3) For a node $v \in V$, let $S(v) \subset A(t)$ be the set of nodes $u \in A(t)$ such that there is an active edge $(u, v)$ from $u$ to $v$ ($S(v)$ are the nodes that can potentially send a token to $v$ in phase $t$).
(4) If $x'_v(t) \leq k - t\delta - \alpha_v$, $v$ sends a token proposal to the $\min\left\{|S(v)|, k - t\delta - x'_v(t)\right\}$ nodes $w \in S(v)$, where priority is given to nodes $w \in S(v)$ of smaller $\deg_G(w)/\alpha_w$ value.
(5) For each node $v \in V$, let $p_v(t)$ be the number of token proposals that node $v$ receives in phase $t$ and let $q_v(t) := \min\left\{p_v(t), x'_v(t)\right\}$. Node $v$ accepts $q_v(t)$ of the proposals and sends a token to the respective outneighbors. The edges over which a token is sent become passive.
(6) For each node $v \in V$, let $r_v(t)$ be the number of tokens that $v$ receives in phase $t$. The number of active tokens at the end of phase $t$ of each node $v$ is set to $x_v(t) := x'_v(t) + r_v(t) - q_v(t)$, i.e., $x_v(t)$ is $x'_v(t)$ plus the number of received tokens and minus the number of sent tokens.

We start by proving that the maximum number of active tokens decreases after each phase, and that the total number of tokens at each node never exceeds $k$.

Lemma 4.1. *For all $v \in V$ and for all $t \geq 0$, if $\delta \leq \alpha_v$, it holds that $x_v(t) \leq \max\{2\alpha_v, k - t \cdot \delta\}$ and $y_v(t) \leq k - x_v(t)$.*

Proof. We prove the upper bound on $x_v(t)$ by induction on $t$. For $t = 0$, we have $x_v(0) \leq k$ and therefore clearly $x_v(0) \leq \max\{\alpha_v, k - 0 \cdot \delta\}$. For the induction step, let us assume that $t \geq 1$ and let us therefore focus on what happens in phase $t$. Note that in step 4 of phase $t$ the above algorithm, each node $v$ sends at most $k - t\delta - x'_v(t)$ proposals and it therefore receives at most that many new tokens. If $x'_v(t) \leq k - t\delta$, we therefore also have $x_v(t) \leq x'_v(t) + (k - t\delta - x'_v(t)) = k - t\delta$. In this case, we have proven the required upper bound on $x_v(t)$. Let us therefore assume that $x'_v(t) > k - t\delta$. In this case, $v$ does not send any proposals and we therefore know that $x_v(t) \leq x'_v(t)$. By the induction hypothesis, we also know that $x_v(t-1) \leq \max\{2\alpha_v, k - (t-1) \cdot \delta\}$. Since we know that $x'_v(t) \leq x_v(t-1)$, then, whenever $x_v(t-1) \leq 2\alpha_v$, we now also have $x_v(t) \leq x'_v(t) \leq 2\alpha_v$. Let us therefore further assume that $2\alpha_v < x_v(t-1) \leq k - (t-1)\delta$. Because $\delta \leq \alpha_v$, in this case, we have $x_v(t-1) > 2\alpha_v \geq \alpha_v + \delta$ and therefore $v \in A(t)$. However, we then set $x'_v(t) := x_v(t-1) - \delta \leq k - t\delta$, and we therefore also have $x_v(t) \leq k - t\delta$. The proves the upper bound on $x_v(t)$.

It remains to show that $y_v(t) \leq k - x_v(t)$. We again apply induction on $t$. Note that we have $y_v(0) = 0$ and $x_v(0) \leq k$ and therefore the bound certainly holds for $t = 0$. For every $t \geq 1$, we either have $y_v(t) = y_v(t-1)$ and $v \notin A(t)$ or $y_v(t) = y_v(t-1) + \delta$ and $v \in A(t)$. We therefore in particular always have $y_v(t) \leq y_v(t-1) + \delta$ and thus $y_v(t) \leq t\delta$. If $x_v(t) \leq k - t\delta$, we thus have $x_v(t) + y_v(t) \leq k$ as required. Let us therefore assume that $x_v(t) > k - t\delta$. Above, we showed that if $x'_v(t) \leq k - t\delta$, it follows that $x_v(t) \leq k - t\delta$. By the contrapositive, if $x_v(t) > k - t\delta$, we therefore know that also $x'_v(t) > k - t\delta$ and thus that $v$ does not sent any proposals in step 4 of the algorithm. In this case, we therefore know that $x_v(t) \leq x'_v(t)$. If $v \notin A(t)$, we have $y_v(t) = y_v(t-1)$ and $x'_v(t) = x_v(t-1)$ and if $v \in A(t)$, we have $y_v(t) = y_v(t-1) + \delta$

and $x'_v(t) = x_v(t-1) - \delta$. In both cases, $x_v(t) \le x'_v(t)$ and the induction hypothesis $x_v(t-1) + y_v(t-1) \le k$ directly imply that $x_v(t) + y_v(t) \le k$. □

We now prove that, for each active edge, the number of passive tokens of each endpoint cannot differ by much.

Lemma 4.2. *For every edge $e = (u,v) \in E$, at the end of each phase $t \le k/\delta - 1$ in the above algorithm, $e$ is passive or we have*

$$y_u(t) - y_v(t) \le 2\alpha_v + \left(\frac{\deg_G(u)\deg_G(v)}{\alpha_u \cdot \alpha_v} + \frac{\deg_G(u)}{\alpha_u} + \frac{\deg_G(v)}{\alpha_v}\right)\cdot\delta.$$

Proof. In each phase $t$, for every node $w \in V$, we have $y_w(t) = y_w(t-1) + \delta$ if $w \in A(t)$ and $y_w(t) = y_w(t-1)$ otherwise. To upper bound $y_u(t) - y_v(t)$ while $e = (u,v)$ is active, we can therefore count the number of phases in which $u \in A(t)$ and $v \notin A(t)$ and where no token is passed over the edge. For $u$ to be in $A(t)$, we must have $x_u(t-1) \ge \alpha_u + \delta$ and for $v$ to not be in $A(t)$, we must have $x_v(t-1) < \alpha_v + \delta$.

As long as $t$ is not too large, $x_v(t-1) < \alpha_v + \delta$ implies that $v$ has the capacity to receive tokens in phase $t$ and it can therefore send proposals to active in-neighbors. To count the number of phases in which $u \in A(t)$ and $v \notin A(t)$, we therefore make a case distinction depending on the value of $t$. We first assume that $t \le (k - 2\alpha_v)/\delta - 1$. We can rewrite this as $\alpha_v + \delta \le k - t\delta - \alpha_v$ and because we know that $x_v(t-1) \le \alpha_v + \delta$, this implies that $x_v(t-1) \le k - t\delta - \alpha_v$. Because $v \notin A(t)$, we also know that $x'_v(t) = x_v(t-1)$ and therefore $x'_v(t) \le k - t\delta - \alpha_v$. This is the condition that is needed in step 4 of the above algorithm for $v$ to send token proposals over its incoming active edges. Whenever $v$ sends a token proposal to an in-neighbor $w$, either $w$ sends a token to $v$ and afterwards the edge $(w,v)$ becomes passive, or $w$ sends at least $\alpha_w$ tokens to other out-neighbors and therefore at least $\alpha_w$ out-edges of $w$ become passive. To each in-neighbor $w \ne u$, $v$ can therefore send proposals at most $\lceil \deg_G(w)/\alpha_w \rceil$ times. Note that in each phase $t$, $v$ only sends proposals (in step 4) if $x'_v(t) \le k - t\delta - \alpha_v$ and it sends proposals either to all nodes in $S(v)$ (i.e., to all active neighbors with an active edge to $v$) or it sends proposals to $k - t\delta - x'_v(t) \ge \alpha_v$ nodes in $S(v)$. Hence, in each phase $t$ in which $v$ sends proposals and in which $v$ does not send a proposal to $u$ although $u \in A(t)$ and $(u,v)$ is still active, there must be at least $\alpha_v$ neighbors $w \in S(v) \setminus \{u\}$ to which $v$ sends a proposal. Note that for each such neighbor $w$, we have $\deg_G(w)/\alpha_w \le \deg_G(u)/\alpha_u$ (because in step 4, proposals are sent to active in-neighbors of smallest $\deg_G(w)/\alpha_w$ ratio first). The number of such phases $t$ for $t \le (k - 2\alpha_v)/\delta - 1$ in which $u \in A(t)$ and $v \notin A(t)$, in which $(u,v)$ is active and $v$ does not send a proposal to $u$ can therefore be upper bounded by

$$\left\lfloor \frac{(\deg_G(v) - 1)\cdot\left\lceil \frac{\deg_G(u)}{\alpha_u}\right\rceil}{\alpha_v}\right\rfloor \le \left(\frac{\deg_G(v)-1}{\alpha_v}\right)\cdot\left(\frac{\deg_G(u)}{\alpha_u}+1\right) \le \frac{\deg_G(u)\cdot\deg_G(v)}{\alpha_u\cdot\alpha_v} + \frac{\deg_G(v)}{\alpha_v}. \quad (2)$$

Further, $v$ can send at most $\lfloor \deg_G(u)/\alpha_u \rfloor$ proposals to $u$ without receiving a token from $u$ (and thus such that $(u,v)$ remains active). The total number of phases $t$ for $t \le k/\delta - 1$ in which $u \in A(t)$ and $v \notin A(t)$ and in which $(u,v)$ remains active can consequently be upper bounded by

$$\frac{2\alpha_v}{\delta} + \frac{\deg_G(u)\cdot\deg_G(v)}{\alpha_u\cdot\alpha_v} + \frac{\deg_G(v)}{\alpha_v} + \frac{\deg_G(u)}{\alpha_u}.$$

In each of those phases, $y_u(t) - y_v(t)$ increases by $\delta$, which directly implies the claim of the lemma. □

We are now ready to prove that our algorithm solves the generalized token dropping game. The following theorem follows almost directly from Lemma 4.1, for space reasons, the proof appears in the full version of this paper.

Theorem 4.3. *At the end of the above algorithm, for every $v \in V$, let $\tau(v)$ be the number of tokens at node $v$. If for all $v \in V$, $\alpha_v \ge \delta$, the above algorithm has a time complexity of $O(k/\delta)$ and at the end of the algorithm, for every node $v \in V$, we have $\tau(v) \le k$ and for every edge $(u,v)$, either $(u,v)$ is passive or*

$$\tau(u) - \tau(v) \le 2(\alpha_u + \alpha_v) + \left(\frac{\deg_G(u)\cdot\deg_G(v)}{\alpha_u\cdot\alpha_v} + \frac{\deg_G(u)}{\alpha_u} + \frac{\deg_G(v)}{\alpha_v}\right)\cdot\delta.$$

## 5 GENERALIZED DEFECTIVE 2-EDGE COLORING

At the core, our edge coloring algorithms are based on the following simple idea. We partition the space of possible colors into two parts and each edge commits to choosing a color from one of the two parts. The two parts can then be solved recursively. Because edges that pick colors from disjoint color spaces cannot conflict with each other, the two parts can be colored recursively in parallel. The task of splitting the set of edges into two parts can be formulated as a defective edge coloring problem as follows.

*Definition 5.1 (Generalized Defective 2-Edge Coloring).* Given $\varepsilon \ge 0$ and $\beta \ge 0$, a graph $G = (V,E)$, and parameters $\lambda_e \in [0,1]$ for all edges $e \in E$, a generalized $(1+\varepsilon,\beta)$-relaxed defective 2-edge coloring of $G$ is an assignment of colors red and blue to the edges $e \in E$ such that for every edge $e \in E$:

- If $e$ is colored red, the number of neighboring red edges is $\le (1+\varepsilon)\cdot\lambda_e\cdot\deg_G(e) + \lambda_e\beta$.
- If $e$ is colored blue, the number of neighboring blue edges is $\le (1+\varepsilon)\cdot(1-\lambda_e)\cdot\deg_G(e) + (1-\lambda_e)\beta$.

We will next show how we can solve a given generalized defective 2-edge coloring instance in two-colored bipartite graphs by using the token dropping game of Section 4. We next show how to transform the generalized defective 2-coloring problem to make it more directly amenable to applying the token dropping game. We first define the notion of generalized balanced edge orientations. For convenience, we only give a definition for bipartite graphs.

*Definition 5.2 (Generalized Balanced Edge Orientation).* Assume that we are given values $\varepsilon \ge 0$ and $\beta \ge 0$, a bipartite graph $G = (U \dot\cup V, E)$, parameters $\eta_e \in \mathbb{R}$ for all edges $e = (u,v) \in E$, $u \in U$, $v \in V$, and an orientation on the edges of $G$. For each node $w \in U \cup V$, let $x_w$ be the number of edges of $w$ that are oriented towards $w$. The orientation is called a *generalized $(\varepsilon,\beta)$-balanced edge orientation* of $G$ if the following holds. For every edge $(u,v) \in E$,

(I) If $e$ is oriented from $u$ to $v$, then $x_v - x_u \leq \eta_e + 1 + \frac{\varepsilon}{2} \cdot \deg_G(e) + \beta$.
(II) If $e$ is oriented from $v$ to $u$, then $x_u - x_v \leq -\eta_e + 1 + \frac{\varepsilon}{2} \cdot \deg_G(e) + \beta$.

The following lemma follows almost directly from the above definitions. The proof appears in the full version of this paper.

Lemma 5.3. *Assume that we are given a bipartite graph $G = (U \dot\cup V, E)$, a parameter $\varepsilon \geq 0$, and parameters $\lambda_e \in [0,1]$ for all $e \in E$. For every edge $e = (u,v)$, $u \in U$, $v \in V$, we define*

$$\eta_e := 1 - 2\lambda_e - (1-\lambda_e) \cdot \deg_G(u) + \lambda_e \cdot \deg_G(v) + \varepsilon \cdot \left(\lambda_e - \frac{1}{2}\right) \cdot \deg_G(e) + (2\lambda_e - 1)\beta \tag{3}$$

*A generalized $(\varepsilon, \beta)$-balanced edge orientation of $G$ w.r.t. the edge parameters $\eta_e$ can be turned into a solution to the given generalized $(1+\varepsilon, 2\beta)$-relaxed defective 2-edge coloring w.r.t. the original edge parameters $\lambda_e$ by coloring edges red that are oriented from $U$ to $V$ and by coloring edges blue that are oriented from $V$ to $U$.*

We next show how to compute a generalized balanced edge orientation (as given by Definition 5.2) and thus in combination with Lemma 5.3 a generalized defective 2-edge coloring. Assume that we have a bipartite 2-colored graph $G = (U \cup V, E)$ with edge parameters $\eta_e \in \mathbb{R}$. We will compute a generalized balanced edge orientation with parameter $\eta_e$ by reducing it to a sequence of instances of the token dropping game. More concreteley, the algorithm has a parameter $\nu > 0$ and it runs in a sequence of phases $\phi = 1, 2, 3, \ldots, O\big(\frac{\log \Delta}{\nu}\big)$. At the start, all edges of $G$ are unoriented and in each phase, a subset of the unoriented edges become oriented. We define $F_\phi$ as the set of edges that get oriented in phase $\phi$, and $F_{<\phi}$ as the set of edges that get oriented before phase $\phi$. By writing $\deg_{F_{<\phi}}(v)$ we refer to the degree of node $v$ in the graph induced by the edges $F_{<\phi}$. In each phase, we use one instance of the token dropping game to make sure that the set of all oriented edges satisfies inequalities (I) and (II) of Definition 5.2 (for an appropriate value of $\varepsilon$). For an edge $e \in E$ and a phase $\phi \geq 0$, we use $d(e,\phi)$ to denote the number of unoriented neighboring edges of $e$ at the end of phase $\phi$. For convenience, we also use $d(e,0) = \deg_G(e)$ to denote the number of unoriented neighboring edges of $e$ at the start. We further define $\bar\Delta := 2\Delta - 2$ as an upper bound on the maximum edge degree in $G$. We further set the parameter $\nu$ such that

$$0 < \nu \leq \frac{1}{8}. \tag{4}$$

For every node $v$ and every phase $\phi \geq 1$, let $x_v(\phi)$ denote the number of edges that are oriented towards $v$ at the end of phase $\phi$. For convenience, we also define $x_v(0) = 0$ as at the beginning all edges are unoriented and therefore no edges are oriented towards $v$. For each node $v$, we further define

$$d^-_\phi(v) := \min_{e \in F_{<\phi} : v \in e} \deg_G(e) \qquad \text{and}$$
$$\alpha_v(\phi) := \max\left\{1, \frac{1}{4} \cdot \frac{\nu^2}{\ln \bar\Delta} \cdot (d^-_\phi(v) + 1)\right\}. \tag{5}$$

We assume that the nodes of $G$ know if they are in $U$ or in $V$ (i.e., we assume that the bipartite graph $G$ is equipped with a 2-vertex coloring). The algorithm in phase $\phi \geq 1$ works as follows:

(1) Let $E_\phi \subseteq E$ be the edges $e \in E$ that are still unoriented at the beginning of phase $\phi$ and for which $d(e, \phi-1) > (1-\nu)^\phi \bar\Delta$.
(2) For every edge $e = (u,v) \in E_\phi$ with $u \in U$ and $v \in V$, $e$ sends a proposal to $v$ if $x_v - x_u \leq \eta_e$ and it sends a proposal to $u$ otherwise.
(3) We set $k_\phi := \lceil \nu(1-\nu)^{\phi-1}\bar\Delta \rceil$. Every node $u \in U \cup V$ that receives at most $k_\phi$ proposals from its edges in $E_\phi$ accepts all those proposals and every node $u \in U \cup V$ that receives more that $k_\phi$ proposals from its edges in $E_\phi$ accepts an arbitrary subset of $k_\phi$ of them.
(4) Let $F_\phi \subseteq E_\phi$ be the set of edges for which the proposal gets accepted. The edges $e \in F_\phi$ will be the ones that get newly oriented in phase $\phi$. For each edge $(u,v) \in F_\phi$, the edge is oriented from $u$ to $v$ if $(u,v)$'s proposal was sent to and accepted by $v$ and the edge is oriented from $v$ to $u$ otherwise.
(5) Let $F_{<\phi} := \bigcup_{i=1}^{\phi-1} F_\phi$ be the set of edges that get oriented before phase $\phi$. We define a subset $F'_{<\phi} \subseteq F_{<\phi}$ of those edges as follows. An edge $e = (u,v) \in F_{<\phi}$ ($u \in U$, $v \in V$) is included in $F'_{<\phi}$ if either $e$ is oriented from $u$ to $v$ and $x_v(\phi-1) - x_u(\phi-1) > \eta_e$ or if $e$ is oriented from $v$ to $u$ and $x_u(\phi-1) - x_v(\phi-1) > -\eta_e$.
(6) We now run an instance of the token dropping game on the graph $G_\phi = (U \cup V, F'_{<\phi})$ (i.e., on the subgraph of $G$ induced by the edges in $F'_{<\phi}$), where each edge in $F'_{<\phi}$ is directed in the opposite direction of its current orientation. Each node $u \in U \cup V$ uses the parameter $\alpha_u(\phi)$ as fixed in Equation (5). Further, the initial number of tokens of each node $u \in U \cup V$ is equal to the number of proposals from its edges in $E_\phi$, $u$ has accepted in the above step 3. Finally, the parameter $\delta_\phi$ is set to
$$\delta_\phi := \max\left\{1, \left\lfloor \frac{1}{16} \cdot \frac{\nu^6}{\ln^3 \bar\Delta} \cdot (1-\nu)^{\phi-1}\bar\Delta \right\rfloor\right\}. \tag{6}$$
(7) To conclude phase $\phi$, we now update the orientation of the edges in $F_{<\phi}$ as follows. We switch the orientation of each edge over which a token is moved in the above token dropping game instance of step 6. All other edges in $F_{<\phi}$ keep their orientations.

We first show that the maximum edge degree of the unoriented part of $G$ decreases exponentially as a function of the number of phases. The proof of Lemma 5.4 is simple and appears in the full version of this paper.

Lemma 5.4. *At the end of phase $\phi \geq 1$ of the above algorithm, we have $d(e,\phi) \leq (1-\nu)^\phi \bar\Delta$ for every edge $e \in E \setminus F_{\leq\phi}$, that is, for every edge that is still unoriented after phase $\phi$.*

To analyze the quality of the produced edge orientation, we define the following quantities for every edge $e \in E$.

$$k_e := \left\lceil \frac{\nu}{1-\nu} \cdot \deg_G(e) \right\rceil \qquad \text{and} \qquad \xi_e := \frac{5}{2} \cdot \frac{\nu}{\ln \bar\Delta} \cdot k_e + 28 \cdot \frac{\ln^2 \bar\Delta}{\nu^4}. \tag{7}$$

Lemma 5.5. *After $\phi$ phases of the above algorithm, for every edge $e = (u,v) \in E$ ($u \in U$, $v \in V$), it holds that either:*

- *$e$ is unoriented, or*

- *$e$ is oriented from $u$ to $v$ and $x_v(\phi) - x_u(\phi) \leq \eta_e + k_e + \phi \cdot \xi_e$, or*
- *$e$ is oriented from $v$ to $u$ and $x_u(\phi) - x_v(\phi) \leq -\eta_e + k_e + \phi \cdot \xi_e$.*

Proof. We prove the lemma by induction on the number of phase $\phi$. At the beginning, all edges are unoriented and therefore clearly for $\phi = 0$, the claim of the lemma holds. For the induction step, assume that $\phi \geq 1$. Consider some edge $(u, v) \in E$ with $u \in U$ and $v \in V$ and assume that $e$ is oriented at the end of phase $\phi$ (as otherwise, the claim of the lemma clearly holds for $e$). We first show that for every node $v \in U \cup V$, we have

$$x_v(\phi - 1) \leq x_v(\phi) \leq x_v(\phi - 1) + k_\phi. \tag{8}$$

To see why Equation (8) holds, first consider the number of incoming edges at node $v$ in the middle of phase $\phi$, right before the we run the token dropping instance of phase $\phi$ in step 6 of the above algorithm. As the orientation of the edges that have been oriented prior to phase $\phi$ can only be changed during token dropping game, the number of incoming edges of $v$ at this point is equal to $x_v(\phi-1)$ plus the number of newly oriented edges of $v$ that are oriented towards $v$. Let us use $y_v$ to denote this number of those newly oriented edges towards $v$. Note that $y_v$ is exactly equal to the number of proposals $v$ accepts in step 3 of the above algorithm. Note also that the initial number of tokens of node $v$ in the token dropping game of phase $\phi$ is equal to $y_v$. When running token dropping, whenever moving a token from a node $u$ to a node $u'$, an edge that was previously oriented from $u'$ to $u$ is reoriented from $u$ to $u'$. Hence, for every token that node $v$ receives in the token dropping game, the total number of edges oriented to $v$ increases by 1 and for every token that $v$ moves to a neighbor, the total number of edges oriented to $v$ decreases by 1. By induction on the steps of the token dropping game, the total number of incoming edges at node $v$ at the end of phase $\phi$ is therefore exactly $x_v(\phi - 1)$ plus the number of tokens at node $v$ at the end of the token dropping game instance of phase $\phi$. Because by definition of the token dropping game, the number of tokens at each node is always in $\{0, \ldots, k_\phi\}$, Equation (8) follows.

Recall that if an edge $e$ gets oriented in a phase $\phi'$, we have $\deg_G(e) \geq d(e, \phi' - 1) \geq (1 - \nu)^{\phi'} \bar{\Delta}$. For every $e$ that gets oriented in or before phase $\phi$, we therefore have $\deg_G(e) \geq (1 - \nu)^{\phi} \bar{\Delta}$. This means that for every edge $e$ that is oriented by the end of phase $\phi$, we in particular have

$$k_e = \left\lceil \frac{\nu}{1 - \nu} \cdot \deg_G(e) \right\rceil \geq \left\lceil \nu (1 - \nu)^{\phi - 1} \bar{\Delta} \right\rceil. \tag{9}$$

For the induction step of our main induction (on $\phi$), we distinguish 4 cases:

*Edge $e$ gets newly oriented in phase $\phi$:* Edge $e$ gets oriented from $u$ to $v$ in phase $\phi$ if $x_v(\phi - 1) - x_u(\phi - 1) \leq \eta_e$ and it gets oriented from $v$ to $u$ otherwise (i.e., if $x_u(\phi - 1) - x_v(\phi - 1) < -\eta_e$). Hence, if $e$ is oriented from $u$ to $v$, Equation (8) implies that $x_v(\phi) - x_u(\phi) \leq \eta_e + k_\phi$ and if $e$ is oriented from $v$ to $e$, Equation (8) implies that $x_u(\phi) - x_v(\phi) \leq -\eta_e + k_\phi$. In both cases, the claim of the lemma follows together with

$$k_\phi = \left\lceil \nu \cdot (1 - \nu)^{\phi - 1} \bar{\Delta} \right\rceil \overset{(9)}{\leq} k_e.$$

*Edge $e$ does not participate in the token dropping game of phase $\phi$:* For the remaining cases, we assume that $e$ was first oriented prior to phase $\phi$. Since $e$ does not to participate in the token dropping in phase $\phi$, the orientation of $e$ does not change in phase $\phi$. For $e$ not to participate in the token dropping in phase $\phi$, we then also must have that either $e$ is oriented from $u$ to $v$ and $x_v(\phi - 1) - x_u(\phi - 1) \leq \eta_e$ or $e$ is oriented from $v$ to $u$ and $x_u(\phi - 1) - x_v(\phi - 1) \leq -\eta_e$. In both cases, the claim of the lemma follows directly by combining this with Equations (8) and (9).

*A token is moved over $e$ in the token dropping game of phase $\phi$:* If prior to running the token dropping game of phase $\phi$, $e$ is oriented from $u$ to $v$, then $x_v(\phi-1) - x_u(\phi-1) > \eta_e$ and otherwise $x_u(\phi - 1) - x_v(\phi - 1) > -\eta_e$. Since a token is moved over $e$ in the token dropping game of phase $\phi$, the orientation of $e$ is switched in phase $\phi$. Hence, if before running token dropping in phase $\phi$, the edge $e$ is oriented from $u$ to $v$, at the end of the phase, $e$ is oriented from $v$ to $u$ and

$$x_u(\phi) - x_v(\phi) \overset{(8),(9)}{\leq} \underbrace{x_u(\phi - 1) - x_v(\phi - 1)}_{x_v(\phi-1) - x_u(\phi-1) > \eta_e} + k_e < -\eta_e + k_e.$$

If before the token dropping instance of phase $\phi$, $e$ is oriented from $v$ to $u$, at the end of the phase, the edge is oriented from $u$ to $v$ and

$$x_v(\phi) - x_u(\phi) \overset{(8),(9)}{\leq} \underbrace{x_v(\phi - 1) - x_u(\phi - 1)}_{x_u(\phi-1) - x_v(\phi-1) > -\eta_e} + k_e < \eta_e + k_e.$$

The claim of the lemma therefore also follows in this case.

*No token is moved over $e$ in the token dropping game of phase $\phi$:* The last case to consider is the case where $e$ participates in the token dropping game, but where no token is moved over the edge. First assume that $e$ is oriented from $u$ to $v$, that is, the edge is directed from $v$ to $u$ in the token dropping game. Let $\tau_u$ and $\tau_v$ denote the number of tokens at node $u$ and $v$ at the end of the token dropping game instance of phase $\phi$. Observe that

$$\begin{aligned}
\alpha_v(\phi) &= \max\left\{1, \frac{1}{4} \cdot \frac{\nu^2}{\ln \bar{\Delta}} \cdot (d^-_\phi(v) + 1)\right\} \\
&\geq \max\left\{1, \frac{1}{4} \cdot \frac{\nu^2}{\ln \bar{\Delta}} \cdot (1 - \nu)^{\phi - 1} \bar{\Delta}\right\} \\
&\geq \max\left\{1, \left\lfloor \frac{1}{16} \cdot \frac{\nu^6}{\ln^3 \bar{\Delta}} \cdot (1 - \nu)^{\phi - 1} \bar{\Delta} \right\rfloor\right\} = \delta_\phi,
\end{aligned}$$

where the first inequality holds because, for every $e$ that gets oriented before phase $\phi$, we have $\deg_G(e) \geq (1 - \nu)^{\phi - 1} \bar{\Delta}$. Hence, $\alpha_v(\phi) \geq \delta_\phi$, and since no token is moved from $v$ to $u$ in the token

dropping game, Theorem 4.3 implies that

$$\begin{aligned}\tau_v - \tau_u &\le 2(\alpha_u(\phi) + \alpha_v(\phi)) \\ &\quad + \left(\frac{\deg_{F_{<\phi}}(u)\deg_{F_{<\phi}}(v)}{\alpha_u(\phi)\cdot\alpha_v(\phi)} + \frac{\deg_{F_{<\phi}}(u)}{\alpha_u(\phi)} + \frac{\deg_{F_{<\phi}}(v)}{\alpha_v(\phi)}\right)\delta_\phi \\ &\le \max\left\{4, \frac{\nu^2}{2\ln\bar{\Delta}}\cdot(d^-_\phi(u) + d^-_\phi(v) + 2)\right\} \\ &\quad + \left(\frac{16\ln^2\bar{\Delta}}{\nu^4} + \frac{8\ln\bar{\Delta}}{\nu^2}\right)\delta_\phi \\ &\le \frac{\nu^2}{\ln\bar{\Delta}}\cdot\deg_G(e) + 4 + \left(\frac{\nu^2}{\ln\bar{\Delta}} + \frac{\nu^4}{2\ln^2\bar{\Delta}}\right)\cdot(1-\nu)^{\phi-1}\bar{\Delta} \\ &\quad + \frac{16\ln^2\bar{\Delta}}{\nu^4} + \frac{8\ln\bar{\Delta}}{\nu^2} \\ &\le \frac{5\nu^2}{2\ln\bar{\Delta}}\cdot\deg_G(e) + 28\cdot\frac{\ln^2\bar{\Delta}}{\nu^4}\end{aligned}$$

The second inequality follows from plugging in the definition of $\alpha_u(\phi)$ and $\alpha_v(\phi)$ (Equation (5)) and the fact that for all nodes $v \in V$, $\deg_{F_{<\phi}}(v) \le d^-_\phi(v) + 1$. The third inequality follows from using that $\nu^2/\ln\bar{\Delta} \le 1$ and from the definition of $\delta_\phi$ (Equation (6)). In the fourth inequality, we use that since edge $e$ was first oriented in a phase before $\phi$, we must have $\deg_G(e) \ge (1-\nu)^{\phi-1}\bar{\Delta}$ and again by using that $\nu^2/\ln\bar{\Delta} \le 1$.

The induction step in the induction over the number of phases and thus the claim of the lemma now follows directly from the definitions of $k_e$ and of $\xi_e$ (Equation (7)). If the edge $e$ is oriented from $v$ to $u$, then the statement holds for the same reasons. □

The following theorem can now be proven by Lemma 5.4 and by plugging together Lemma 5.5, Equation (7), and Theorem 4.3. The proof appears in the full version of this paper.

Theorem 5.6. *Assume that we are given a bipartite graph $G = (U \dot\cup V, E)$, a value $\varepsilon > 0$, and edge parameters $\eta_e \in \mathbb{R}$ for all edges $e \in E$. There exists a constant $C > 0$ such that if $\varepsilon \le 1$, there is an $O\big(\frac{\log^4\Delta}{\varepsilon^6}\big)$-round distributed algorithm to compute a generalized $(\varepsilon,\beta)$-balanced orientation of $G$ w.r.t. the edge parameters $\eta_e$, where $\beta = C\cdot\frac{\ln^3\bar{\Delta}}{\varepsilon^5}$.*

By combining Theorem 5.6 and Lemma 5.3, we obtain the following.

Corollary 5.7. *Let $\varepsilon \le 1$. The generalized $(1+\varepsilon,\beta)$-relaxed defective 2-edge coloring can be solved deterministically in $O\big(\frac{\log^4\Delta}{\varepsilon^6}\big)$ rounds in the* CONGEST *model, for $\beta = O\big(\frac{\log^3\Delta}{\varepsilon^5}\big)$.*

## 6 $O(\Delta)$-EDGE COLORING IN THE CONGEST MODEL

In this section, we present an algorithm for solving the $O(\Delta)$-edge coloring problem in the CONGEST model. We start by showing how to solve the problem on bipartite 2-colored graphs. We will later show how to remove this restriction.

Lemma 6.1. *The $(2+\varepsilon)\Delta$-edge coloring problem can be solved in $O\big(\frac{\log^{11}\Delta}{\varepsilon^6}\big)$ deterministic rounds in the* CONGEST *model in bipartite 2-colored graphs, for any $1 \ge \varepsilon > 0$.*

Proof Sketch. The high-level idea of the proof is the following. In Corollary 5.7, we show that a $(1+\varepsilon',\beta)$-relaxed defective 2-edge coloring can be solved in $\mathrm{poly}(\log(\Delta)/\varepsilon')$ time in the CONGEST model (for $\beta = O\big(\frac{\log^3\Delta}{\varepsilon'^5}\big)$). As long as the maximum edge degree $\bar{\Delta}$ is sufficiently larger than $\beta/\varepsilon'$, we can therefore compute a 2-defective edge coloring for which the maximum defect is only by a $(1+\varepsilon')$ factor larger than $\bar{\Delta}/2$. Choosing $\varepsilon' \le c\cdot\varepsilon/\log\Delta$ for a sufficiently small constant $c$ and an integer $k \ge 1$, as long as $k \gg \beta/\varepsilon'$, we can therefore recursively compute a defective $2^k$-edge coloring such that the maximum edge defect is at most $(1+\varepsilon/2)\bar{\Delta}/2^k$. The required $(2+\varepsilon)\Delta$-coloring of the given bipartite graph is then obtained by using $2^k$ disjoint color ranges for each of the $2^k$ graphs of maximum degree $(1+\varepsilon/2)\bar{\Delta}/2^k$ that we get from the recursive defective coloring. A full proof of the claim appears on the full version of this paper. □

In order to solve the problem in general graphs, we make use of the following lemma, that follows directly from results presented in [11] (a proof is also shown in the full version of this paper).

Lemma 6.2. *The $(\varepsilon\Delta + \lfloor\Delta/2\rfloor)$-defective vertex 4-coloring problem, given an $O(\Delta^2)$-vertex coloring, can be solved in $O(1/\varepsilon^2)$ rounds in the* CONGEST *model.*

We are now ready to present our CONGEST algorithm for $O(\Delta)$-edge coloring on general graphs. Theorem 1.2 follows directly from the following theorem.

Theorem 6.3. *The $(8+\varepsilon)\Delta$-edge coloring problem can be solved in $O\big(\frac{\log^{12}\Delta}{\varepsilon^6} + \log^* n\big)$ deterministic rounds in the* CONGEST *model, for any small enough constant $\varepsilon > 0$.*

Proof Sketch. We start by computing an initial $O(\Delta^2)$-vertex coloring, which can be done in $O(\log^* n)$ rounds. Let $\varepsilon_1$ be a parameter that we choose appropriately We apply Lemma 6.2 with parameter $\varepsilon_1$ for 4-coloring the nodes of the graph with colors in $\{1,2,3,4\}$. Then, let $G_1$ be the graph induced by edges $\{u,v\}$ satisfying that the color of $u$ is either 1 or 2, and the color of $v$ is either 3 or 4. This graph is clearly bipartite, and nodes know their side of the bipartition. Hence, we can apply Lemma 6.1 to color the edges of this graph by using at most $(2+\varepsilon_2)\Delta$ colors, for some appropriate parameter $\varepsilon_2$. We then do the same in the bipartite graph induced by the edges that go from colors $\{1,3\}$ to colors $\{2,4\}$. For the edges that are colored so far, we have now used $(4+2\varepsilon_2)\Delta$ colors. All the remaining uncolored edges are now between nodes of the same color and hence the degree of them is at most $(1/2+\varepsilon_1)\Delta$ and thus close to $\Delta/2$. To color the rest of the graph, we now recurse. As the maximum degree (almost) halves in each step, the total number of colors will be (essentially) twice the number of colors we have used so far and thus $(8+O(\varepsilon_1))\Delta$. A full proof appears in the full version of this paper. □

## 7 $(2\Delta-1)$-EDGE COLORING IN THE LOCAL MODEL

In this section, we give an overview of our algorithm that computes a (*degree*+1)-list edge coloring and thus as a special case a $(2\Delta-1)$-edge coloring. For the detailed formal arguments, we refer to the full version of this paper.

*Coloring 2-Colored Bipartite Graphs.* We first again consider the case of computing a somewhat relaxed coloring of a 2-colored bipartite graph $G = (U \cup V, E)$. For the high level description here, assume that $G$ has maximum degree $\Delta$ and that every edge $e \in E$ has a list $L_e$ of size $|L_e| \geq 2\deg_G(e)$. Assume further that $L_e \subseteq \{1, \dots, C\}$, i.e., all colors come from a global space of $C$ colors. If all nodes have access to the same colors (as in the algorithm of Section 6), we can use defective 2-colorings to split the graph into two parts such that the maximum edge degree in each part is approximately halved compared to the original maximum degree. However, in the case of list coloring, we cannot do this because in a local distributed way, we cannot split the colors into two parts such that each node can keep half of its colors. We instead have to adapt a method that was introduced in [39] and also used in [5]. First, the global space of colors $\{1, \dots, C\}$ is split into two approximately equal parts. For this, let us call the colors $\{1, \dots, \lfloor C/2 \rfloor\}$ red and the remaining colors blue. We want to color the edges red and blue such that afterwards, the red edges $e$ only keep the red colors in their list $L_e$ and the blue edges only keep their blue colors. In this way, the two parts are independent of each other and can be colored in parallel. Note however that because the lists $L_e$ are arbitrary subsets of $\{1, \dots, C\}$, the list $L_e$ of an edge $e$ can consist of an arbitrary division into red and blue colors. For an edge $e \in E$, let $\lambda_e$ be the fraction of red colors in its list, i.e., $|L_e \cap \{1, \dots, \lfloor C/2 \rfloor\}| = \lambda_e |L_e|$. If $e$ chooses to be red, its list shrinks by a factor $\lambda_e$ and we therefore also want to shrink $e$'s degree by at least (approximately) a factor $\lambda_e$ and if $e$ chooses to be blue, the $e$'s degree has to shrink by at least (approximately) a factor $1 - \lambda_e$. If the degree of $e$ is sufficiently large, we can achieve this by computing a generalized defective 2-edge coloring as defined in Definition 5.1 and we can use Corollary 5.7 to compute such a 2-coloring efficiently. The goal therefore is to recursively split the global color space into two parts and always use Corollary 5.7 to split the edges such that the degree-to-list size ratio grows by at most a factor $1 + o(1)$. This essentially works as in Section 6. There is only one additional small issue that we have to take care of. Since the ratio $\lambda_e$ can be an arbitrary value between 0 and 1, we have no control over the minimum and maximum edge degree or list size in the graph. Corollary 5.7 however only gives good guarantees for edges that have a sufficiently large degree. As soon as an edge $e$ has degree at most some poly $\log \Delta$, we therefore do not further split the color space of $e$ recursively. Edge $e$ then waits until all neighboring edges that are split further have been colored and afterwards, $e$ only has a small uncolored degree and can therefore be colored greedily by a standard edge coloring algorithm.

*Degree+1 List Edge Coloring on General Graphs.* Given a (*degree*+1)-list coloring instance of an arbitrary graph $G$, we start by computing a poly($\Delta$)-vertex coloring, which can be done in $O(\log^* n)$ rounds.

Then, we compute a defective $O(1)$-vertex coloring of $G$, where each node has defect at most $\Delta/c$ for a sufficiently large constant $c$. By using an algorithm from [11], this can be done in $O(\log^* \Delta)$ time by exploiting the precomputed poly($\Delta$)-vertex coloring.

We then sequentially iterate through all possible pairs of colors $(a, b)$, and we consider the graph induced by edges with one endpoint of color $a$ and the other endpoint of color $b$. This graph is clearly bipartite, but we cannot directly apply the previously described algorithm, because the lists of the edges may be too small, compared to their degree. In order to apply the algorithm anyways, we use a method that has already been used in [5, 24, 39] and that allows us to use the previously described algorithm to *partially* color the graph.

Like this, we can reduce the uncolored degree of each edge by a constant factor even in a (*degree* + 1)-list coloring instance, and repeating $O(\log \Delta)$ times allows to color all the edges and to fully solve the given (*degree* + 1)-list coloring problem.

## REFERENCES


[1] Noga Alon, László Babai, and Alon Itai. 1986. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms* 7, 4 (1986), 567–583. https://doi.org/10.1016/0196-6774(86)90019-2

[2] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. 1989. Network Decomposition and Locality in Distributed Computation. In *Proc. 30th Symp. on Foundations of Computer Science (FOCS)*. 364–369. https://doi.org/10.1109/SFCS.1989.63504

[3] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2019. Lower Bounds for Maximal Matchings and Maximal Independent Sets. In *Proc. 60th IEEE Symp. on Foundations of Computer Science (FOCS)*. 481–497.

[4] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. 2021. Distributed $\Delta$-Coloring Plays Hide-and-Seek. *arXiv preprint arXiv:2110.00643* (2021).

[5] Alkida Balliu, Fabian Kuhn, and Dennis Olivetti. 2020. Distributed Edge Coloring in Time Quasi-Polylogarithmic in Delta. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*. 289–298.

[6] Leonid Barenboim. 2016. Deterministic ($\Delta$+1)-Coloring in Sublinear (in $\Delta$) Time in Static, Dynamic, and Faulty Networks. *Journal of ACM* 63, 5 (2016), 1–22. https://doi.org/10.1145/2979675

[7] Leonid Barenboim and Michael Elkin. 2010. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Comput.* 22 (2010), 363–379. https://doi.org/10.1007/s00446-009-0088-2

[8] Leonid Barenboim and Michael Elkin. 2011. Deterministic Distributed Vertex Coloring in Polylogarithmic Time. *Journal of ACM* 58 (2011), 23:1–23:25. https://doi.org/10.1145/2027216.2027221

[9] Leonid Barenboim and Michael Elkin. 2013. *Distributed Graph Coloring: Fundamentals and Recent Developments.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00520ED1V01Y201307DCT011

[10] Leonid Barenboim, Michael Elkin, and Uri Goldenberg. 2018. Locally-Iterative Distributed ($\Delta$ + 1)-Coloring below Szegedy-Vishwanathan Barrier, and Applications to Self-Stabilization and to Restricted-Bandwidth Models. In *Proc. 37th ACM Symp. on Principles of Distributed Computing (PODC)*. 437–446.

[11] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. 2014. Distributed ($\Delta$+1)-Coloring in Linear (in $\Delta$) Time. *SIAM J. Comput.* 43, 1 (2014), 72–95. https://doi.org/10.1137/12088848X

[12] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The Locality of Distributed Symmetry Breaking. *J. ACM* 63, 3 (2016), 1–45. https://doi.org/10.1145/2903137

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

[14] Sebastian Brandt, Barbara Keller, Joel Rybicki, Jukka Suomela, and Jara Uitto. 2021. Efficient Load-Balancing through Distributed Token Dropping. In *Proc. 33rd ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*. 129–139.

[15] Sebastian Brandt and Dennis Olivetti. 2020. Truly Tight-in-$\Delta$ Bounds for Bipartite Maximal Matching and Variants. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*. 69–78.

[16] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2019. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. *SIAM J. Comput.* 48, 1 (2019), 122–143. https://doi.org/10.1137/17M1117537

[17] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. 2018. An optimal distributed ($\Delta$+1)-coloring algorithm?. In *Proc. 50th ACM Symp. on Theory of Computing, (STOC)*. 445–456. https://doi.org/10.1145/3188745.3188964

[18] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. 2020. Distributed Edge Coloring and a Special Case of the Constructive Lovász Local Lemma. *ACM Transactions on Algorithms* 16, 1 (2020), 8:1–8:51. https://doi.org/10.1145/3365004

[19] Andrzej Czygrinow, Michal Hanckowiak, Edyta Szymanska, and Wojciech Wawrzyniak. 2016. On the distributed complexity of the semi-matching problem. *J. Comput. Syst. Sci.* 82, 8 (2016), 1251–1267.

[20] Devdatt P. Dubhashi, David A. Grable, and Alessandro Panconesi. 1998. Near-Optimal, Distributed Edge Colouring via the Nibble Method. *Theor. Comput. Sci.*

203, 2 (1998), 225–251.
[21] Michael Elkin, Seth Pettie, and Hsin-Hao Su. 2015. (2Δ - l)-Edge-Coloring is Much Easier than Maximal Matching in the Distributed Setting. In *Proc. 26thACM-SIAM Symp. on Discrete Algorithms (SODA)*. 355–370. https://doi.org/10.1137/1.9781611973730.26
[22] Manuela Fischer. 2017. Improved Deterministic Distributed Matching via Rounding. In *Proc. 31st Symp. on Distributed Computing (DISC)*. 17:1–17:15.
[23] Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. 2017. Deterministic Distributed Edge-Coloring via Hypergraph Maximal Matching. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*. 180–191. https://doi.org/10.1109/FOCS.2017.25
[24] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. 2016. Local Conflict Coloring. In *Proc. 57th IEEE Symp. on Foundations of Computer Science (FOCS)*. 625–634.
[25] Mohsen Ghaffari. 2016. An Improved Distributed Algorithm for Maximal Independent Set. In *Proc. 27th ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 270–277. https://doi.org/10.1137/1.9781611974331.ch20
[26] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhon. 2021. Improved Deterministic Network Decomposition. In *Proc. 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2904–2923. https://doi.org/10.1137/1.9781611976465.173
[27] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. 2018. On Derandomizing Local Distributed Algorithms. In *Proc. 59th Symp. on Foundations of Computer Science (FOCS)*. 662–673.
[28] Mohsen Ghaffari and Fabian Kuhn. 2021. Deterministic Distributed Vertex Coloring: Simpler, Faster, and without Network Decomposition. In *Proc. 62nd IEEE Symp. on Foundations of Computing (FOCS)*.
[29] M. Ghaffari, F. Kuhn, and Y. Maus. 2017. On the Complexity of Local Distributed Graph Problems. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*. 784–797.
[30] Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. 2018. Deterministic distributed edge-coloring with fewer colors. In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*. 418–430.
[31] A.V. Goldberg, S.A. Plotkin, and G.E. Shannon. 1988. Parallel Symmetry-Breaking in Sparse Graphs. *SIAM Journal on Discrete Mathematics* 1, 4 (1988), 434–446.
[32] Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan. 2021. Near-Optimal Distributed Degree+1 Coloring. *CoRR* abs/2112.00604 (2021).
[33] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. 1998. On the Distributed Complexity of Computing Maximal Matchings. In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 219–225.
[34] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. 1999. A Faster Distributed Algorithm for Computing Maximal Matchings Deterministically. In *Proc. 18th ACM Symp. on Principles of Distributed Computing (PODC)*. 219–228.
[35] David G. Harris. 2020. Distributed Local Approximation Algorithms for Maximum Matching in Graphs and Hypergraphs. *SIAM J. Comput.* 49, 4 (2020), 711–746. https://doi.org/10.1137/19M1279241
[36] David G. Harris, Johannes Schneider, and Hsin-Hao Su. 2018. Distributed (Δ +1)-Coloring in Sublogarithmic Rounds. *J. ACM* 65, 4 (2018), 19:1–19:21.
[37] Amos Israeli and A. Itai. 1986. A fast and simple randomized parallel algorithm for maximal matching. *Inform. Process. Lett.* 22, 2 (1986), 77–80. https://doi.org/10.1016/0020-0190(86)90144-4
[38] Fabian Kuhn. 2009. Local Weak Coloring Algorithms and Implications on Deterministic Symmetry Breaking. In *Proc. 21st ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*.
[39] Fabian Kuhn. 2020. Faster Deterministic Distributed Coloring Through Recursive List Coloring. In *Proc. 32st ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 1244–1259.
[40] Fabian Kuhn and Roger Wattenhofer. 2006. On the complexity of distributed graph coloring. In *Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC)*. 7–15.
[41] Nathan Linial. 1987. Distributive graph algorithms – Global solutions from local data. In *Proc. 28th Symp. on Foundations of Computer Science (FOCS 1987)*. IEEE, 331–335. https://doi.org/10.1109/SFCS.1987.20
[42] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053. https://doi.org/10.1137/0215074
[43] Yannic Maus and Tigran Tonoyan. 2020. Local Conflict Coloring Revisited: Linial for Lists. *CoRR* abs/2007.15251 (2020).
[44] Alessandro Panconesi and Romeo Rizzi. 2001. Some simple distributed algorithms for sparse networks. *Distributed Computing* 14, 2 (2001), 97–100.
[45] Alessandro Panconesi and Aravind Srinivasan. 1996. On the Complexity of Distributed Network Decomposition. *Journal of Algorithms* 20, 2 (1996), 356–374. https://doi.org/10.1006/jagm.1996.0017
[46] Alessandro Panconesi and Aravind Srinivasan. 1997. Randomized Distributed Edge Coloring via an Extension of the Chernoff-Hoeffding Bounds. *SIAM J. Comput.* 26, 2 (1997), 350–368. https://doi.org/10.1137/S0097539793250767
[47] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719772
[48] Václav Rozhoň and Mohsen Ghaffari. 2020. Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization. In *Proc. 52nd ACM Symp. on Theory of Computing (STOC)*. 350–363.
[49] Johannes Schneider and Roger Wattenhofer. 2010. A new technique for distributed symmetry breaking. In *Proc. 29th ACM Symp. on Principles of Distributed Computing (PODC)*. 257–266. https://doi.org/10.1145/1835698.1835760
[50] Hsin-Hao Su and Hoa T. Vu. 2019. Towards the locality of Vizing's theorem. In *Proc. 51st ACM Symp. on Theory of Computing (STOC)*. 355–364. https://doi.org/10.1145/3313276.3316393
[51] Mario Szegedy and Sundar Vishwanathan. 1993. Locality based graph coloring. In *Proc. 25th ACM Symp. on Theory of Computing (STOC)*. 201–207.