Distributed Deterministic Edge Coloring using Bounded Neighborhood Independence

Leonid Barenboim[*]
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel.
leonidba@cs.bgu.ac.il

Michael Elkin[**]
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel.
elkinm@cs.bgu.ac.il

ABSTRACT

We study the *edge-coloring* problem in the message-passing model of distributed computing. This is one of the most fundamental problems in this area. Currently, the best-known deterministic algorithms for $(2\Delta - 1)$-edge-coloring requires $O(\Delta) + \log^* n$ time [23], where Δ is the maximum degree of the input graph. Also, recent results of [5] for vertex-coloring imply that one can get an $O(\Delta)$-edge-coloring in $O(\Delta^{\epsilon} \cdot \log n)$ time, and an $O(\Delta^{1+\epsilon})$-edge-coloring in $O(\log \Delta \log n)$ time, for an arbitrarily small constant $\epsilon > 0$.

In this paper we devise a significantly faster deterministic edge-coloring algorithm. Specifically, our algorithm computes an $O(\Delta)$-edge-coloring in $O(\Delta^{\epsilon}) + \log^* n$ time, and an $O(\Delta^{1+\epsilon})$-edge-coloring in $O(\log \Delta) + \log^* n$ time. This result improves the state-of-the-art running time for deterministic edge-coloring with this number of colors in almost the entire range of maximum degree Δ. Moreover, it improves it exponentially in a wide range of Δ, specifically, for $2^{\Omega(\log^* n)} \leq \Delta \leq polylog(n)$. In addition, for small values of Δ (up to $\log^{1-\delta} n$, for some fixed $\delta > 0$) our deterministic algorithm outperforms all the existing *randomized* algorithms for this problem.

On our way to these results we study the *vertex-coloring* problem on graphs with bounded *neighborhood independence*. This is a large family of graphs, which strictly includes line graphs of r-hypergraphs (i.e., hypergraphs in which each hyperedge contains r or less vertices) for $r = O(1)$, and graphs of bounded growth. We devise a very fast deterministic algorithm for vertex-coloring graphs with bounded neighborhood independence. This algorithm directly gives rise to our edge-coloring algorithms, which apply to *general* graphs.

Our main technical contribution is a subroutine that computes an $O(\Delta/p)$-defective p-vertex coloring of graphs with bounded neighborhood independence in $O(p^2) + \log^* n$ time, for a parameter p, $1 \leq p \leq \Delta$. In all previous efficient distributed routines for m-defective p-coloring the product $m \cdot p$ is super-linear in Δ. In our routine this product is *linear* in Δ, and this enables us to speed up the coloring drastically.

[*]Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.
[**] Supported by the Binational Science Foundation, grant No. 2008390. Additional funding was provided by the Lynn and William Frankel Center for Computer Sciences.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Computations on Discrete Structures; G.2.2 [**Graph Theory**]: Network Problems

General Terms

Algorithms

Keywords

Legal-Coloring, Defective-Coloring, Line-Graphs

1. INTRODUCTION

1.1 Edge-Coloring

We study the *edge-coloring* problem in the *message passing model* of distributed computing. Specifically, we are given an n-vertex undirected unweighted graph $G = (V, E)$, with each vertex hosting an autonomous processor. The processors have distinct identity numbers (henceforth, Ids) from the range $\{1, 2, ..., n\}$. They communicate with each other over the edges of E. The communication occurs in discrete rounds. In each round each vertex can send a message to each of its neighbors, and these messages arrive to their destinations before the next round starts. The running time of an algorithm in this model is the number of rounds of communication that are required for the algorithm to terminate.

A legal *edge-coloring* φ of $G = (V, E)$ is a function $\varphi : E \rightarrow N$ that satisfies that for any pair of edges $e, e' \in E$ that share an endpoint (henceforth, *incident*), it holds that $\varphi(e) \neq \varphi(e')$. Denote by $\Delta = \Delta(G)$ the maximum degree of the graph G. A classical theorem of Vizing [29] shows that for any graph G, its edges can be legally colored in $(\Delta + 1)$ colors. Obviously, at least Δ colors are required.

The edge coloring problem is one of the most fundamental problems in Graph Theory and Graph Algorithms. It also has numerous applications in Computer Science, including job-shop scheduling, packet-routing, and resource allocation [16, 12]. This problem was also extensively studied in the message-passing model [10, 11, 12, 14, 23, 25]. Panconesi and Rizzi [23] showed that a $(2\Delta - 1)$-edge-coloring can be computed deterministically in $O(\Delta) + \log^* n$ time. Panconesi and Srinivasan [25] devised a randomized $(1.6\Delta +$

$O(\log^{1+\epsilon} n)$)-edge coloring algorithm that runs in polylogarithmic time, where $\epsilon > 0$ is an arbitrarily small constant. Dubhashi et. al. [11] used the Rődl nibble method to improve this to a randomized $(1+\epsilon)\Delta$-edge-coloring in time $O(\log n)$, as long as $\Delta = \omega(\log n)$. Grable and Panconesi [14] showed that if for every edge $e = (u, w)$, the degree of either u or w is sufficiently large (at least $2^{\Omega(\frac{\log n}{\log\log n})}$), then a $(1+\epsilon)\Delta$-edge-coloring, for an arbitrarily small constant $\epsilon > 0$, can be computed in $O(\log\log n)$ time by a randomized algorithm. Czygrinow et. al. [10] devised a deterministic $O(\Delta \log n)$-edge-coloring that requires $O(\log^4 n)$ time.

A more general approach to the edge-coloring and many other related problems was taken in [1, 22, 24]. These papers presented algorithms that compute a *network decomposition*, i.e., a partition of the input graph into regions of small diameter. This partition admits also additional helpful properties. This partition can then be used to compute edge-coloring, vertex-coloring, maximal independent set, and other related structures. In particular, by this technique one can get a deterministic $(2\Delta - 1)$-edge-coloring algorithm that requires $2^{O(\sqrt{\log n})}$ time [24, 1].

A (legal) vertex coloring ψ of $G = (V, E)$ is a function $\psi : V \to N$ that satisfies that for any edge $e = (u, w) \in E$, $\psi(u) \neq \psi(w)$. We refer to $\psi(u)$ as the *ψ-color of u*. By considering the line graph $L(G) = (E, \mathcal{E} = \{(e, e') \mid e \cap e' \neq \emptyset\})$, it is easy to see that any vertex-coloring algorithm that employs $f(\Delta)$ colors, for a function $f()$, translates into an edge-coloring algorithm that employs $f(2\Delta)$ colors, with essentially the same running time [1]. This observation enables one to harness many of the recent advances in vertex-coloring for obtaining significantly faster edge-coloring algorithms as well. Most relevant in this context are the results of [18, 28, 5]. Kothapalli et. al. [18] showed that an $O(\Delta)$-vertex-coloring (and, consequently, $O(\Delta)$-edge-coloring as well) can be computed in $O(\sqrt{\log n})$ rounds, by a randomized algorithm. Recently Schneider and Wattenhofer [28] devised a randomized algorithm that computes (1) a $(\Delta+1)$-vertex-(and edge-) coloring in $O(\log \Delta + \sqrt{\log n})$ time; (2) an $O(\Delta + \log n)$-coloring in $O(\log\log n)$ time; and (3) an $O(\Delta \log^{(k)} n + \log^{1+1/k} n)$-coloring in $f(k) = O(1)$ time, for some fixed function $f()$ and any positive integer k. In [5] the authors of the current paper devised a deterministic algorithm that, for an arbitrarily small constant $\epsilon > 0$, computes (1) an $O(\Delta^{1+\epsilon})$-coloring in $O(\log \Delta \log n)$ time; and (2) an $O(\Delta)$-coloring in $\Delta^{\epsilon} \log n$ time.

In the current paper we show that in the case of *edge*-coloring the factor $\log n$ can be eliminated. Specifically, we devise a deterministic algorithm that for an arbitrarily small constant $\epsilon > 0$, computes (1) an $O(\Delta^{1+\epsilon})$-edge-coloring in $O(\log \Delta) + \log^* n$ time; (2) an $O(\Delta)$-edge-coloring in $O(\Delta^{\epsilon}) + \log^* n$ time. In addition we have a tradeoff curve with a number of results along it, in which the number of colors is larger than $\Omega(\Delta)$, but smaller than $\Delta^{1+\epsilon}$.

These results compare very favorably to the state-of-the-art. We start with comparing them to deterministic algorithms. For Δ in the range $\omega(\log^* n) \leq \Delta \leq O(\log n \log\log n)$ the fastest currently known algorithm for edge-coloring with $O(\Delta^{1+\epsilon})$ or less colors is due to Panconesi and Rizzi [23]. Its running time is $O(\Delta) + \log^* n$. Our algorithm runs *exponentially* faster, in time $O(\log \Delta) + \log^* n$, but it employs more colors ($O(\Delta^{1+\epsilon})$ instead of $(2\Delta - 1)$). In addition, another variant of our algorithm employs only $O(\Delta)$ colors, and has a significantly better running time than that of the algorithm of [23], specifically, $O(\Delta^{\epsilon}) + \log^* n$. We stress that this result holds in particular for the range $\Delta = o(\log n)$. In this range, all previously known deterministic algorithms for $O(\Delta)$-edge-coloring and $O(\Delta)$-vertex-coloring on general graphs have running time $\Omega(\Delta)$. Therefore, our algorithm is the first to compute an $O(\Delta)$-edge-coloring in sublinear in Δ time on general graphs.

For $\Delta = \Omega(\log n \log\log n)$ the fastest known algorithm for edge-coloring with $O(\Delta^{1+\epsilon})$ colors is due to [5]. Its running time is $O(\log \Delta \log n)$, instead of $O(\log \Delta) + \log^* n$ for our new algorithm. Note that as long as Δ is at most polylogarithmic in n, the new running time is $O(\log\log n)$ instead of $O(\log n \log\log n)$ of [5], i.e., our improvement in this range is exponential as well. To summarize, our algorithm improves the state-of-the-art running time for deterministic algorithms in almost the entire range of the maximum degree Δ, i.e., for $\Delta = \omega(\log^* n)$, and it improves it exponentially for $2^{\Omega(\log^* n)} \leq \Delta \leq O(\log^k n)$, for an arbitrarily large constant k. We remark that even when Δ is greater than polylogarithmic in n, the running time $O(\log \Delta + \log^* n)$ of our $O(\Delta^{1+\epsilon})$-coloring algorithm is at least *quadratically* smaller than the previous state-of-the-art $O(\log \Delta \log n)$ due to [5]. In other words, the factor of $\log n$ is a huge factor in this context, and eliminating it results in major improvements. See Table 1 below for a concise comparison of previous and new deterministic results.

Next, we compare the running time and the number of colors of our *deterministic* algorithm with the state-of-the-art with respect to *randomized* algorithms. For $\Delta = \Omega(\log n)$ the recent randomized algorithm of Schneider and Wattenhofer [28] outperforms our algorithm. However, for $\Delta \leq \log^{1-\delta} n$, for an arbitrarily small constant $\delta > 0$, the algorithm of [28] either employs $\Omega(\log n)$ colors (i.e., more than $\Delta^{1+\epsilon}$ for an arbitrarily small $\epsilon > 0$), or its running time is $\Omega(\sqrt{\log n})$. (Note, however, that the randomized algorithm of [28] solves a generally harder vertex-coloring problem, rather than edge-coloring.) Hence in the range $\omega(\log^* n) \leq \Delta \leq \log^{1-\delta} n$, for some fixed constant $\delta > 0$, our deterministic algorithm outperforms all previous algorithms, deterministic and randomized. Moreover, in the range $2^{\Omega(\log^* n)} \leq \Delta \leq \log^{1-\delta} n$ our algorithm is *exponentially faster* than the previous ones. Indeed, for $\Delta \leq \sqrt{\log n}$ the best previous algorithm that achieves $O(\Delta^{1+\epsilon})$ or less colors is due to [23], whose running time is $O(\Delta) + \log^* n$. On the other hand, the running time of our algorithm is $O(\log \Delta) + \log^* n$. For $\sqrt{\log n} \leq \Delta \leq \log^{1-\delta} n$ the best previous algorithms that achieve that many colors are due to [28, 18], and their running time is $O(\sqrt{\log n})$. Our algorithm requires in this range just $O(\log \Delta) + \log^* n = O(\log\log n)$ time. (On the other hand, the variant of the algorithm of [28] that runs in $O(\sqrt{\log n})$ time employs just $(2\Delta - 1)$ colors, as opposed to $\Delta^{1+\epsilon}$ colors that are employed by our algorithm. The algorithm of [23] also employs only $(2\Delta - 1)$ colors.) See Table 2 below for a concise comparison.

Observe also that the $\log^* n$ term in the running time of our algorithms is optimal up to a factor of 2, in view of the lower bounds of [21]. Specifically, Linial's lower bound [21] implies that $f(\Delta)$-edge-coloring, for any fixed function $f()$, requires at least $\frac{1}{2} \log^* n$ time. Moreover, there is a variant of our algorithm that achieves the precisely optimal

[1]As long as one allows arbitrarily large messages.

Range of Δ	$\omega(\log^* n) = \Delta = o(\log n \log\log n)$	$\Omega(\log n \log\log n) = \Delta$
Previous	$(2\Delta - 1)$ colors, $O(\Delta) + \log^* n$ time [23]	$O(\Delta)$ colors, $O(\Delta^\epsilon \log n)$-time [5]
		$O(\Delta^{1+\epsilon})$ colors, $O(\log \Delta \log n)$-time [5]
New	$O(\Delta)$ colors, $O(\Delta^\epsilon) + \log^* n$ time	$O(\Delta)$ colors, $O(\Delta^\epsilon) + \log^* n$ time
	$O(\Delta^{1+\epsilon})$ colors, $O(\log \Delta) + \log^* n$ time	$O(\Delta^{1+\epsilon})$ colors, $O(\log \Delta) + \log^* n$ time

Table 1: ***A concise comparison of previous state-of-the-art edge-coloring*** **deterministic** ***algorithms with our new algorithms.***

Range of Δ	$\omega(\log^* n) = \Delta = O(\sqrt{\log n})$	$\Omega(\sqrt{\log n}) = \Delta \leq \log^{1-\delta} n$
Previous	$(2\Delta - 1)$ colors, $O(\Delta) + \log^* n$ time [23]	$(2\Delta - 1)$ colors, $O(\sqrt{\log n})$ time [28]
New (Deter.)	$O(\Delta^{1+\epsilon})$ colors, $O(\log \Delta) + \log^* n$ time	$O(\Delta^{1+\epsilon})$ colors, $O(\log\log n)$ time

Table 2: ***A concise comparison of previous state-of-the-art edge-coloring*** **randomized and deterministic** ***algorithms with our new*** **deterministic** ***algorithm. The algorithm of [23] is deterministic. The algorithm of [28] is randomized.***

additive term of $\frac{1}{2}\log^* n$, while achieving almost the same dependence on Δ. By "almost" the same dependence on Δ we mean that it achieves (for an arbitrarily small constant $\epsilon > 0$), (1) an $O(\Delta)$-edge-coloring in $O(\Delta^\epsilon) + \frac{1}{2}\log^* n$ time, and (2) an $O(\Delta^{1+\epsilon})$-edge-coloring in $O(\log \Delta \cdot \frac{\log^* \Delta}{\log(\log^* \Delta)}) + \frac{1}{2}\log^* n$ time. In other words, item (1) is the same as that cited above, except that $\log^* n$ is replaced by $\frac{1}{2}\log^* n$, and in item (2) there is also a tiny slack factor of $\frac{\log^* \Delta}{\log(\log^* \Delta)}$.

1.2 Bounded Neighborhood Independence

Our results for edge-coloring follow from far more general results that we describe below. *Neighborhood independence* $I(G)$ of a graph $G = (V, E)$ is the maximum number of independent [1] neighbors of a single vertex $v \in V$. The family of graphs with constant neighborhood independence (henceforth, *bounded neighborhood independence*) is a very general family of graphs. Indeed, for any graph G, the neighborhood independence of its line graph $L(G)$ is at most 2. Moreover, for an r-hypergraph $\mathcal{H}$ (i.e., a hypergraph in which every hyperedge contains at most r vertices), $I(L(\mathcal{H})) \leq r$.

Another important family of graphs which is subsumed by the family of graphs with bounded neighborhood independence is the family of graphs of *bounded growth.* A graph $G = (V, E)$ is said to be of bounded growth if there exists a function $f()$ such that for any $r = 1, 2, ...,$ the number of independent vertices at distance at most r from any given vertex is at most $f(r)$. Distributed algorithms for vertex-coloring and computing a maximal independent set on graphs from this family is a subject of intensive recent research [17, 13, 27]. The crowning result of this effort is the deterministic algorithm of [27] that computes a maximal independent set and a $(\Delta + 1)$-vertex-coloring for graphs from this family in optimal time $O(\log^* n)$. Note, however, that a graph G with a constant neighborhood independence may contain an arbitrarily large independent set U whose all vertices are at distance at most 2 from some given vertex v in G. Thus, graphs with bounded neighborhood independence may have unbounded growth. (Consider, for example, a graph H that is obtained by connecting each vertex of an $n/2$-vertex clique with a distinct isolated vertex. Each vertex in H has at most 2 independent neighbors. However, each vertex v in the clique has at least $n/2 = \Omega(\Delta)$ independent vertices in $\Gamma_2(v)$, and so the graph H is not a graph of bounded growth.)

Yet another family of graphs which is subsumed by the family of graphs of bounded independence is the family of *claw-free* graphs. A graph is *claw-free* if it excludes $K_{1,3}$ as an induced subgraph. (In fact, for any $r = 2, 3, ...,$ the family of graphs with independence at most r is precisely the family of graphs that exclude induced $K_{1,r+1}$.) The family of claw-free graphs attracted enormous attention in Structural Graph Theory. See, e.g., the series of papers by Chudnovsky and Seymour, starting with [7].

In this paper we devise a vertex-coloring algorithm for graphs of bounded neighborhood independence that computes (for an arbitrarily small constant $\epsilon > 0$) (1) an $O(\Delta)$-vertex-coloring in $O(\Delta^\epsilon) + \frac{1}{2}\log^* n$ time, and (2) an $O(\Delta^{1+\epsilon})$-vertex-coloring in $O(\log \Delta \cdot \frac{\log^* \Delta}{\log(\log^* \Delta)}) + \frac{1}{2}\log^* n$ time. Modulo some subtleties, these results imply our main results about edge-coloring described in Section 1.1. In addition, they apply to line graphs of r-hypergraphs for any constant r, to claw-free graphs, to graphs of bounded growth, and to many other graphs.

1.3 Our Techniques

In the heart of our algorithms lie improved algorithms for computing *defective colorings.* For a non-negative integer m and positive integer χ, an *m-defective χ-vertex-coloring* φ of a graph $G = (V, E)$ is a function $\varphi : V \to \{1, 2, ..., \chi\}$ that satisfies that for every vertex $v \in V$, it has at most m neighbors colored by $\varphi(v)$. The parameter m is called the *defect* of the coloring. Defective coloring was introduced by Cowen et. al [8] and by Harary and Jones [15]. It was extensively studied from a graph-theoretic perspective [2, 9]. Recently defective coloring was discovered to be very useful in the context of distributed graph coloring [4, 19]. Specifically, the state-of-the-art $(\Delta + 1)$-vertex-coloring algorithms for general graphs [4, 19] are based on subroutines for computing defective coloring. For a parameter p, these subroutines compute an $O(\Delta/p)$-defective p^2-coloring. (In [4] the running time of such a subroutine is $O(p^2) + \frac{1}{2}\log^* n$, and in [19] it is $O(\log^* \Delta) + \frac{1}{2}\log^* n$.) It was observed in [4] that one could have devised significantly faster coloring algorithms if there were an efficient (distributed) routine for computing an

[1] Two vertices u, w are *independent* in G if $(u, w) \notin E$.

m-defective χ-coloring with a *linear* in Δ product of m and χ. (The current state-of-the-art [4, 19], has $m = O(\Delta/p)$, $\chi = p^2$, i.e., $m \cdot \chi = O(\Delta \cdot p)$ instead of the desired $O(\Delta)$.)

In this paper we show that if one restricts his attention to the family of bounded neighborhood independence graphs, then this goal can be achieved. Specifically, we devise an algorithm that computes an $O(\Delta/p)$-defective p-vertex-coloring of a given graph of bounded neighborhood independence in $O(p^2) + \frac{1}{2}\log^* n$ time. As a result we obtain a bunch of drastically faster algorithms for vertex-coloring these graphs, and, consequently, for edge-coloring general graphs.

Whether it is possible to devise an efficient $O(\Delta/p)$-defective p-vertex-coloring algorithm for general graphs remains a challenging open question. Recently in [5] the authors of the current paper were able to circumvent this question by the means of *arbdefective coloring*. Note that an $O(\Delta/p)$-defective p-vertex-coloring can be seen as a partition of the vertex set into p subsets, each inducing a subgraph of maximum degree at most $O(\Delta/p)$. In [5] the authors showed that the vertex set of a graph of arboricity [1] a can be efficiently partitioned into p subsets, each inducing a subgraph of arboricity $O(a/p)$. This partition is then employed in [5] to devise a suite of efficient algorithms for vertex-coloring general graphs. In particular, using this technique [5] devised an $O(\Delta^{1+\epsilon})$-vertex coloring algorithm in $O(\log\Delta \log n)$ time, for an arbitrarily small constant $\epsilon > 0$.

Note, however, that the factor of $\log n$ in the running time of the algorithms of [5] is inherent, because these algorithms rely heavily on the notion of arboricity, and more specifically, on the machinery of forest-decompositions developed in [3] for working with graphs of bounded arboricity. On the other hand, a lower bound shown in [3] stipulates that computing a forests-decomposition requires $\Omega(\frac{\log n}{\log a})$ time, where a is the arboricity. Consequently, the factor of $\log n$ in the running time is unavoidable [2] using the approach of [5]. In the current paper we pursue a different line of attack. Specifically, we devise improved algorithms for *defective* coloring, rather than circumventing it and going through *arbdefective* coloring.

1.4 Structure of the Paper

In Section 2 we describe the definitions and notation employed in our algorithms. In Section 3 we devise defective vertex-coloring algorithms for graphs with bounded neighborhood independence. In Section 4 we devise legal vertex-coloring algorithms for this family of graphs. In Section 5 we devise legal edge-coloring algorithms for general graphs.

2. PRELIMINARIES

Unless the base value is specified, all logarithms in this paper are of base 2. For a non-negative integer i, the *iterative log-function* $\log^{(i)}(\cdot)$ is defined as follows. For an integer $n > 0$, $\log^{(0)} n = n$, and $\log^{(i+1)} n = \log(\log^{(i)} n)$, for every $i = 0, 1, 2, \ldots$. Also, $\log^* n$ is defined by: $\log^* n = \min\left\{i \mid \log^{(i)} n \le 2\right\}$.

[1]The *arboricity* of a graph $G = (V, E)$ is $a(G) = \max\{\left\lceil \frac{|E(U)|}{|U|-1} \right\rceil : U \subseteq V, |U| \ge 2\}$.

[2]In fact, the algorithm of [5] computes an $O(a^{1+\epsilon})$-coloring in time $O(\log a \log n)$ for graphs of arboricity a. An $O(\Delta^{1+\epsilon})$-coloring in $O(\log\Delta \log n)$ time is a direct corollary of this result. On the other hand, it is known [3] that $O(a^{1+\epsilon})$-coloring requires $\Omega(\frac{\log n}{\log a})$ time.

The *degree* of a vertex v in a graph $G = (V, E)$, denoted $\deg(v) = \deg_G(v)$, is the number of edges incident to v. A vertex u such that $(u, v) \in E$ is called a *neighbor* of v in G. The *neighborhood* $\Gamma(v) = \Gamma_G(v)$ of v is the set of neighbors of v. The maximum degree of a vertex in G, denoted $\Delta(G)$, is defined by $\Delta = \Delta(G) = \max_{v\in V} \deg(v)$. The graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$, denoted $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. The notation $V(G')$ and $E(G')$ is used to denote the vertex set V' of G', and the edge set E' of G', respectively.

The *line graph* $L(G) = (V'', E'')$ of a graph $G = (V, E)$ is a graph in which V'' contains a vertex v_e for each edge $e \in E$, and an edge $(v_e, w_{e'})$ if and only if the edges e and e' of E share a common endpoint. We say that a vertex $v_e \in V''$ and an edge $e \in E$ *correspond* to each other.

The *out-degree* of a vertex v in a directed graph $\hat{G}$ is the number of edges incident to v that are oriented outwards of v. An *orientation* σ of (the edge set of) a graph is an assignment of direction to each edge $(u, v) \in E$, either towards u or towards v. An edge (u, v) that is oriented towards v is denoted by $\langle u, v\rangle$. The *out-degree* of an orientation σ of a graph G is the maximum out-degree of a vertex in G with respect to σ. In a given orientation, each neighbor u of v that is connected to v by an edge oriented towards u is called a *parent* of v. In this case we say that v is a *child* of u.

For a graph $G = (V, E)$, a set of vertices $U \subseteq V$ is called an *independent set* if for every pair of vertices $v, w \in U$ it holds that $(v, w) \notin E$.

The minimum number of colors that can be used in a legal vertex-coloring of a graph G is called *the chromatic number* of G, denoted $\chi(G)$.

Next, we state a number of known results that will be used in our algorithms.

Lemma 2.1. *(1) [21] A legal $O(\Delta^2)$-vertex-coloring can be computed in $\log^* n$ time.*
(2) [4, 19] A legal $(\Delta + 1)$-vertex-coloring can be computed in $O(\Delta) + \log^* n$ time.
(3) [19] A $\lfloor \Delta/p \rfloor$-defective $O(p^2)$-vertex-coloring can be computed in $O(\log^* n)$ time.

3. DEFECTIVE COLORING

In this section we present a defective vertex coloring algorithm for graphs with bounded neighborhood independence. We begin with a formal definition of this family of graphs.

Definition 3.1. Graphs with neighborhood independence bounded by c.
For a graph $G = (V, E)$ and a vertex $v \in V$, the neighborhood independence *of v, denoted $I(v)$, is the size of maximum-size independent subset $U \subseteq \Gamma(v)$.*
The neighborhood independence *of a graph G is defined as $I(G) = \max_{v\in V}\{I(v)\}$. For a positive parameter c, a graph $G = (V, E)$ is said to have* neighborhood independence bounded by c *if $I(G) \le c$.*

Let c be a fixed positive constant, and p be a parameter such that $1 \le p \le \Delta$. We devise a procedure, called *Procedure Defective-Color*, that computes an $O(\Delta/p)$-defective p-coloring on graphs with neighborhood independence bounded by c. This coloring is achieved by first computing a defective $O(p^2)$-coloring, and then reducing the number of colors to p, using special properties of graphs with bounded neighborhood independence. Procedure Defective-Color receives as input a graph G with neighborhood independence bounded by c, a positive parameter b, and the parameter Λ which serves as an upper bound on the maximum degree of the input graph. The parameter b satisfies that $b \geq 1$, $b \cdot p \leq \Lambda$. This parameter controls the tradeoff between the defect of the resulting coloring and the running time of the procedure. Specifically, the defect behaves as $\frac{\Lambda}{p}(1 + O(1/b))$, and the running time is at most $O(b^2 \cdot p^2 + \log^* n)$. We assume that all vertices know the value of c before the computation starts.

The procedure starts with computing a $\lfloor \Lambda/(b \cdot p) \rfloor$-defective $O((b \cdot p)^2)$-coloring φ of G using Lemma 2.1(3). The coloring φ is employed for computing another defective coloring ψ of the vertices of G. The recoloring step spends one round for each φ-color class. Specifically, each vertex $v \in V$ computes $\psi(v)$ as follows. The vertex v waits for each neighbor u of v with $\varphi(u) < \varphi(v)$ to select a color $\psi(u)$. Once v receives a message from each such neighbor u with its color $\psi(u)$, it sets $\psi(v)$ to be a value from the range $\{1, 2, .., p\}$ that is used by the minimum number of neighbors u with $\varphi(u) < \varphi(v)$. Once v selects its color $\psi(v)$, it sends it to all its neighbors. This completes the description of the algorithm.

We need the following piece of notation. For a vertex v and an index $k \in \{1, 2, ..., p\}$, let $N_v(k) = |\{u \in \Gamma(v) \mid \psi(u) = k, \varphi(u) < \varphi(v)\}|$ denote the number of neighbors u of v that have smaller φ-color than v has, and whose ψ-color was set to k. Next, we provide the pseudocode of Procedure Defective-Color.

Algorithm 1 Procedure Defective-Color(G, b, p, Λ)

```
An algorithm for each vertex $v \in V$
 1: $\varphi(v)$ := compute $\lfloor \Lambda/(b \cdot p) \rfloor$-defective $O((b \cdot p)^2)$-coloring
    using Lemma 2.1(3)
 2: send $\varphi(v)$ to all neighbors
 3: $\psi(v) := 0$
 4: while $\psi(v) = 0$, in each round do
 5:    if $v$ received $\psi(u)$ for each neighbor $u$ of $v$ with
       $\varphi(u) < \varphi(v)$ then
 6:       $m := \min\{N_v(k) \mid k \in \{1, 2, ..., p\}\}$
 7:       $\psi(v)$ := a color $k \in \{1, 2, ..., p\}$ such that $N_v(k) = m$
 8:       send $\psi(v)$ to all neighbors
 9:    end if
10: end while
```

Observe that a vertex v waits only for neighbors with smaller φ-color before selecting $\psi(v)$. Consequently, it selects the color $\psi(v)$ after at most $\varphi(v)$ rounds from the time when step 2 of Algorithm 1 was executed. This fact is stated in the following lemma.

Lemma 3.2. *Let φ be the coloring computed in the first step of Algorithm 1. Let R be the round in which step 2 is executed. A vertex v selects a color $\psi(v) \neq 0$ in round $R + \varphi(v)$ or earlier.*

Proof. Let $\ell = O((b \cdot p)^2)$ be the number of colors employed by φ. The lemma is proved by induction on the number of rounds. We prove that once a round $i = R + 1, R + 2, \ldots, R + \ell$ is completed, all vertices v with $\varphi(v) \leq i$ have already selected the color $\psi(v)$. For the base case, observe that the vertices v with $\varphi(v) = 1$ have no neighbors with smaller φ-color. Therefore they select a ψ-color in round $R + 1$, immediately after receiving the φ-colors of their neighbors in round R. For the induction step, suppose that in round $i - 1$ all vertices v with $\varphi(v) \leq i - 1$ have already selected the color $\psi(v)$. Hence, each vertex u with $\varphi(u) \leq i$ receives the color ψ for each neighbor with smaller φ-color before round i. Consequently, the vertices u select a ψ-color in round i or earlier. □

Since the coloring φ employs $\ell = O((b \cdot p)^2)$ colors, it follows that all vertices select a ψ-color at most ℓ rounds after the computation of the defective coloring in step 1 of Algorithm 1. By Lemma 2.1(3), step 1 requires $O(\log^* n)$ rounds. The overall running time of Algorithm 1 is given in the following corollary.

Corollary 3.3. *The running time of Procedure Defective-Color is $O((b \cdot p)^2 + \log^* n)$.*

In what follows we prove the correctness of Procedure Defective-Color. By the Pigeonhole principle, the number of neighbors u of a vertex v such that $\varphi(u) < \varphi(v)$ and $\psi(u) = \psi(v)$ is at most Λ/p. (Otherwise, there are more than (Λ/p) neighbors of v that are colored by a ψ-color i, for each $i = 1, 2, ..., p$. Hence, v has more than Λ neighbors, a contradiction.) In addition, since φ is a $\lfloor \Lambda/(b \cdot p) \rfloor$-defective coloring, there are at most $\lfloor \Lambda/(b \cdot p) \rfloor$ neighbors u of v that have the same φ-color as v has, i.e., satisfy $\varphi(u) = \varphi(v)$. These neighbors may also end up selecting the same ψ-color that v selects. Hence the number of neighbors u of v that satisfy $\varphi(u) \leq \varphi(v)$ and $\psi(u) = \psi(v)$ is at most $\Lambda/p + \Lambda/(b \cdot p)$. In order to prove that ψ is an $O(\Lambda/p)$-defective coloring we also prove a somewhat surprising claim regarding the other neighbors of v. Specifically, we prove that the number of neighbors u of a vertex v such that $\varphi(u) > \varphi(v)$ and $\psi(u) = \psi(v)$ is $O(\Lambda/p)$ as well. Consequently, ψ is an $O(\Lambda/p)$-defective p-coloring.

For $i = 1, 2, ..., p$, let G_i be the subgraph induced by the ψ-color class i, i.e., by the vertex set $\{v \in V \mid \psi(v) = i\}$. As a first step we show that the chromatic number $\chi(G_i)$ of G_i is at most $(\Lambda/(b \cdot p) + \Lambda/p) + 1$. (See Lemma 3.5.) We prove this claim by presenting an acyclic orientation of G_i with out-degree at most $(\Lambda/(b \cdot p) + \Lambda/p)$. Since a graph with acyclic orientation with out-degree d is legally $(d+1)$-colorable (see Lemma 3.4), the claim follows. Then we show that bounded chromatic number in conjunction with bounded neighborhood independence imply a bounded degree.

Lemma 3.4. *A graph G with an acyclic orientation μ with out-degree d satisfies that $\chi(G) \leq d + 1$.*

Lemma 3.4 is widely known. Its correctness follows from the fact that if a graph G has an acyclic orientation with out-degree d, then the degeneracy [1] of G is at most d. Therefore, it is $(d + 1)$-colorable.

Lemma 3.5. *For $i = 1, 2, ..., p$, it holds that $\chi(G_i) \leq (\Lambda/(b \cdot p) + \Lambda/p) + 1$.*

Proof. Let μ_i be the following orientation of G_i. For each edge $e = (u, v) \in E(G_i)$, orient the edge towards the endpoint that is colored by a smaller φ-color. If $\varphi(u) = \varphi(v)$, then orient e towards the endpoint with smaller Id among u, v. Each vertex v in G_i has at most Λ/p neighbors u in

[1]A graph G has *degeneracy* at most d if any subgraph of G contains a vertex with degree at most d.

G_i with smaller φ-colors. (This is because $\psi(u) = \psi(v)$ and $\varphi(u) < \varphi(v)$.) In addition, each vertex v in G_i has at most $\Lambda/(b \cdot p)$ neighbors u in G_i with $\varphi(v) = \varphi(u)$. Consequently, μ_i has out-degree at most $(\Lambda/(b \cdot p) + \Lambda/p)$.

Next, we prove that the orientation μ_i is acyclic. Let C be a cycle of G_i. Let v be a vertex on the cycle C with the largest φ-color. If there are several vertices that satisfy this condition, let v be the vertex with the greatest Id. Let u and w be the neighbors of v in C. Since $\varphi(u) < \varphi(v)$ or ($\varphi(u) = \varphi(v)$ and $Id(u) < Id(v)$), the edge (v, u) is oriented by μ_i towards u. Similarly, the edge (v, w) is oriented towards w. Therefore, C is not an oriented cycle. Consequently, μ_i is acyclic. Since the out-degree of μ_i is at most $(\Lambda/(b \cdot p) + \Lambda/p)$, Lemma 3.4 implies that $\chi(G_i) \leq (\Lambda/(b \cdot p) + \Lambda/p) + 1$. $\square$

The next lemma shows that the family of graphs with bounded neighborhood independence is closed under taking vertex-induced subgraphs.

Lemma 3.6. *For a positive integer c, let $G = (V, E)$ be a graph with neighborhood independence at most c. The subgraph induced by a subset $U \subseteq V$ also has neighborhood independence at most by c.*

Proof. Let $\mathcal{G} = G(U)$ be the subgraph induced by U. For a vertex $u \in U$, $\Gamma_{\mathcal{G}}(u)$ is the neighborhood of u in $\mathcal{G}$, and $\Gamma_G(u)$ is the neighborhood of u in G. Suppose for contradiction that there exists a vertex $u \in U$ such that there is an independent set $W \subseteq \Gamma_{\mathcal{G}}(u)$ with cardinality $|W| > c$. For a pair of vertices $v, w \in W$, it holds that $(v, w) \notin E$. In addition, $\Gamma_{\mathcal{G}}(u) \subseteq \Gamma_G(u)$. Therefore, $W \subseteq \Gamma_G(u)$ is an independent set with more than c vertices, and it is contained in the neighborhood $\Gamma_G(u)$ of the vertex u. This is a contradiction. $\square$

We employ Lemmas 3.4 - 3.6 to prove the correctness of Procedure Defective-Color.

Theorem 3.7. *Suppose that Procedure Defective-Color is invoked on a graph G with maximum degree Δ and with neighborhood independence bounded by a positive constant c. Suppose also that it receives as input three integer parameters $b \geq 1, p \geq 1$, $\Lambda \geq 1$, such that $b \cdot p \leq \Lambda$, and $\Lambda \geq \Delta$. Then Procedure Defective-Color computes a $((\Lambda/(b \cdot p) + \Lambda/p) \cdot c + c)$-defective p-coloring.*

Proof. Recall that G_i is the graph induced by vertices with ψ-color i returned by Procedure Defective-Color, for $i = 1, 2, ..., p$. By Lemma 3.6, since G_i is a subgraph of G, the neighborhood independence of G_i is bounded by c. We prove that the maximum degree of G_i, for $i = 1, 2, ..., p$, is at most $(\Lambda/(b \cdot p) + \Lambda/p) \cdot c + c$. Suppose for contradiction that there is a vertex $v \in G_i$ such that $\deg_{G_i}(v) > (\Lambda/(b \cdot p) + \Lambda/p) \cdot c + c$. Let φ' be a legal coloring of G_i that employs the minimum number of colors. Each color class of φ' is an independent set. Therefore, for a positive integer q, the number of neighbors u of v such that $\varphi'(u) = q$ is at most c. Consequently, the number of different colors employed for coloring the set $\Gamma_{G_i}(v)$ of neighbors of v in G_i is at least $\left\lceil \deg_{G_i}(v)/c \right\rceil > (\Lambda/(b \cdot p) + \Lambda/p) + 1$. However, by Lemma 3.5, $\chi(G_i) \leq (\Lambda/(b \cdot p) + \Lambda/p) + 1$, contradiction. $\square$

We summarize this section with the following corollary.

Corollary 3.8. *For an integer parameter p, $1 \leq p \leq \Lambda$, $\Lambda \geq \Delta$, and a constant $\epsilon > 0$, a $((c + \epsilon) \cdot \frac{\Delta}{p} + c)$-defective p-coloring of a graph G with $I(G) \leq c$ can be computed in $O(p^2 + \log^* n)$ time.*

Proof. Set $b < \frac{1}{\epsilon}$. Now the corollary follows from Corollary 3.3 and Theorem 3.7. $\square$

Observe that for graphs with bounded neighborhood independence, the product of the defect $O(\Delta/p)$ and the number of colors p in the coloring produced by Corollary 3.8 is $O(\Delta)$. This is in sharp contrast to the current state-of-the-art for distributed defective coloring in *general* graphs [4, 19], which is $O(\Delta/p)$-defective p^2-coloring. On the other hand, the latter coloring can be computed faster, specifically, within $O(\log^* n)$ time [19].

4. LEGAL COLORING GRAPHS WITH BOUNDED NEIGHBORHOOD INDEPENDENCE

In this section we employ the defective coloring algorithm from the previous section for legal vertex coloring of graphs with neighborhood independence at most $c = O(1)$. Fix an arbitrarily small constant $\epsilon > 0$. Once a $(c + \epsilon) \cdot \frac{\Delta}{p} + c = O(\Delta/p)$-defective p-coloring ψ of a graph G is computed, it constitutes a vertex partition $V_1, V_2, ..., V_p$, in which V_i is the set of vertices with ψ-color i, for $i = 1, 2, ..., p$. In other words, $V = \bigcup_{i=1}^{n} V_i$, and for a pair of distinct indices $i, j \in \{1, 2, ..., p\}$, $i \neq j$, $V_i \cap V_j = \emptyset$. The subgraph G_i induced by V_i has maximum degree $O(\Delta/p)$, since each vertex in G has at most $O(\Delta/p)$ neighbors with the same ψ-color. Therefore, one can legally color the subgraphs $G_1, G_2, ..., G_p$ employing $O(\Delta/p)$ colors for each subgraph, using Lemma 2.1(2). These colorings, denoted $\varphi_1, \varphi_2, ..., \varphi_p$, are computed in parallel on the subgraphs $G_1, G_2, ..., G_p$. Let $m = O(\Delta/p)$ denote the maximum number of colors employed by φ_i, for $i = 1, 2, .., p$. Next, the colorings are combined into a unified legal coloring φ of G as follows. Observe that each vertex $v \in V$ belongs exactly to one subgraph G_j among $G_1, G_2, ..., G_p$. We set $\varphi(v) = \varphi_j(v) + (j - 1) \cdot m$. (The color $\varphi(v)$ can also be thought of as a pair $(j, \varphi_j(v))$, where $v \in V_j$.)

The coloring φ is a legal coloring of G, since for any pair of vertices $u, v \in G$, if they belong to the same subgraph G_i, $i \in \{1, 2, ..., p\}$, then $\varphi_i(v) \neq \varphi_i(u)$, and, therefore, $\varphi(v) \neq \varphi(u)$. Otherwise, u belongs to G_i, and v belong to G_j, for some $1 \leq i \neq j \leq p$. Since $|(j - 1) \cdot m - (i - 1) \cdot m| \geq m$, and $|\varphi_j(u) - \varphi_i(v)| \leq m - 1$, in this case also it holds that $\varphi(v) \neq \varphi(u)$.

The running time required for computing φ is the running time of computing a legal coloring of a graph with degree $\left\lfloor (c + \epsilon) \cdot \frac{\Delta}{p} + c \right\rfloor = O(\Delta/p)$. Hence, by Lemma 2.1(2), the running time of computing φ from a given $O(\Delta/p)$-defective p-coloring ψ is $O(\Delta/p + \log^* n)$. By Corollary 3.8, the running time of computing ψ is $O(p^2 + \log^* n)$. Therefore, the overall running time of computing φ from scratch is $O(\Delta/p + p^2 + \log^* n)$. This running time is optimized by setting $p = \left\lfloor \Delta^{1/3} \right\rfloor$, resulting in overall $O(\Delta^{2/3} + \log^* n)$ time. The number of colors employed by the resulting legal coloring is at most $((c+\epsilon) \cdot \frac{\Delta}{p} + c + 1) \cdot p \leq (c + \epsilon') \cdot \Delta$, for any constant ϵ', $\epsilon' > \epsilon$. (Note that $c = O(1)$ and $\Delta/p = \omega(1)$). We summarize this result in the following lemma.

Lemma 4.1. *For any constant $\epsilon' > 0$, a legal $((c + \epsilon') \cdot \Delta)$-coloring of graphs with neighborhood independence at most c can be computed in $O(\Delta^{2/3} + \log^* n)$ time.*

Next, we present a significantly faster $O(\Delta)$-coloring procedure for the family of graphs with neighborhood independence bounded by c, for a positive constant c. The procedure is called *Procedure Legal-Color*. During its execution defective colorings are computed several times. In the first phase of the procedure a defective coloring of the input graph is computed. This coloring forms a partition of the original graph into vertex-disjoint subgraphs, each with maximum degree smaller than Δ. Then the procedure is invoked recursively on these subgraphs in parallel. This invocation partitions each subgraph into more subgraphs with yet smaller maximum degrees. This process repeats itself until the maximum degrees of all subgraphs are sufficiently small. Then, legal colorings of these subgraphs are computed in parallel, and merged into a unified legal coloring of the input graph. Even though Procedure Defective-Color is invoked many times by Procedure Legal-Color, the running time of Procedure Legal-Color is much smaller than the time given in Lemma 4.1. The improvement in time is achieved by selecting different parameters in the defective colorings computations, making the invocations significantly faster than a single invocation with the parameter $p = \left\lfloor \Delta^{1/3} \right\rfloor$.

In particular, in the algorithm that was described above we partitioned the vertex set of the graph into p disjoint subsets with maximum degree $\Delta' \leq (c+\epsilon')\frac{\Delta}{p}$ each. In each of the p subgraphs one can invoke recursively the algorithm from Lemma 4.1. It will produce a $((c+\epsilon')^2 \cdot \frac{\Delta}{p})$-coloring of each of the subgraphs, i.e., a $((c+\epsilon')^2 \cdot \Delta)$-coloring of the original graph. The running time becomes $O((\frac{\Delta}{p})^{2/3} + p^2 + \log^* n)$. By setting $p = \Delta^{1/4}$, we achieve the running time $O(\Delta^{1/2} + \log^* n)$. Generally, suppose for a constant $i = 1, 2, ...$ that we have a $((c+\epsilon')^i \cdot \Delta)$-coloring algorithm with running time $O(\Delta^{\frac{2}{2+i}} + \log^* n)$. Then the above argument converts it into a $((c+\epsilon')^{i+1} \cdot \Delta)$-coloring algorithm with running time $O(\Delta^{\frac{2}{2+(i+1)}} + \log^* n)$. To summarize:

Theorem 4.2. *For any constant $i = 1, 2, ...$, and any arbitrarily small constant $\eta > 0$, a $((c^i + \eta) \cdot \Delta)$-coloring of a graph G with maximum degree Δ and neighborhood independence at most c can be computed in $O(\Delta^{\frac{2}{2+i}} + \log^* n)$ time.*

In what follows we extend this argument to the case of superconstant i. Procedure Legal-Color receives as input a graph G, positive integer parameters b, p such that $p > 4c$, $1 \leq b \cdot p \leq \Delta$, a parameter λ such that $2c < \lambda \leq \Delta$, and a parameter Λ. The parameter Λ represents an upper bound on the maximum degree of the input graph. Initially, Λ is set to Δ. Later as the procedure is invoked recursively, this parameter is set to smaller values, and it keeps decreasing as the recursion proceeds. The threshold parameter λ determines the termination condition of the recursion. Specifically, if $\Lambda \leq \lambda$ then a legal $(\Lambda+1)$-coloring of G is computed directly using Lemma 2.1(2). Otherwise, an $O(\Delta/p)$-defective p-coloring ψ of G is computed, producing the subgraphs $G_1, G_2, ..., G_p$ induced by the ψ-color classes $1, 2, ..., p$, respectively. Let $\lambda' = O(\Delta/p)$ denote the defect of the coloring ψ. Next, Procedure Legal-Color is invoked on each of these subgraphs recursively, with the degree parameter λ'. All other parameters, that is, b, p, and λ, do not change throughout the recursion.

For technical convenience, Procedure Legal-Color returns not only the resulting coloring φ, but also an upper bound ϑ on the number of colors that this coloring employs. On the bottom level of the recursion, i.e., when $\Lambda \leq \lambda$, the number of colors that is used by φ is at most $\Lambda + 1$. In this case the algorithm (see line 3) sets $\vartheta = \Lambda + 1$. In the more general case when Λ is greater than the threshold value λ, the algorithm invokes itself recursively on each of the subgraphs $G_1, G_2, ..., G_p$. These recursive invocations return the pairs $(\varphi_1, \vartheta_1), (\varphi_2, \vartheta_2), ..., (\varphi_p, \vartheta_p)$, where for each $i = 1, 2, ..., p$, φ_i is a ϑ_i-coloring of G_i. These colorings are merged into a unified ϑ-coloring φ of the entire graph G, with $\vartheta = \Sigma_{i=1}^{p} \vartheta_i$. It will be shown later that, in fact, $\vartheta_i = \vartheta_j$ for every $i, j \in \{1, 2, ..., p\}$. (In other words, the *upper bound* on the number of colors employed by the coloring φ_i that the algorithm returns is equal to the *upper bound* that it returns for the coloring φ_j. This does not necessarily mean that the two coloring use exactly the same number of colors.) This implies that one can actually set $\vartheta = \vartheta' \cdot p$, where $\vartheta' = \vartheta_i$ for some $i \in \{1, 2, ..., p\}$, as the algorithm indeed does in line 11.

Moreover, to obtain the unified legal coloring φ the algorithm just adds to the color $\varphi_i(v)$ of a vertex $v \in V_i = V(G_i)$ the value $(i-1) \cdot \vartheta_i$. Since $\vartheta_i = \vartheta'$ for every $i \in \{1, 2, ..., p\}$, it follows that this way vertices of V_i end up being φ-colored by a color from the palette $\{(i-1)\vartheta'+1, (i-1)\vartheta'+2, ..., i \cdot \vartheta'\}$. Consequently, for two vertices u, w, $u \in V_i$, $w \in V_j$, $i \neq j$, their φ-colors belong to disjoint palettes, and are, thus, different. Below we provide the pseudocode of Procedure Legal-Color.

Algorithm 2 Procedure Legal-Color($G, b, p, \lambda, \Lambda$)

```
An algorithm for each vertex $v \in V$.
1: if ($\Lambda \leq \lambda$) then
2:     $\varphi :=$ a legal $(\Lambda+1)$-coloring of $G$ using Lemma 2.1(2)
3:     $\vartheta := \Lambda + 1$   /* number of colors employed by $\varphi$ */
4: else
5:     $\psi :=$ Defective-Color($G, b, p, \Lambda$)
       /* for $i = 1, 2, ..., p$, let $G_i$ be the graph induced by
       vertices of $\psi$-color $i$ */
6:     $\Lambda' := \lfloor (\Lambda/(b \cdot p) + \Lambda/p) \cdot c + c \rfloor$   /* defect parameter
       of $\psi$ */
7:     for $i = 1, 2, ..., p$, in parallel do
8:         $(\varphi_i, \vartheta') :=$ Legal-Color($G_i, b, p, \lambda, \Lambda'$)
           /* recursive invocation that computes $\vartheta'$-coloring
           $\varphi_i$ of $G_i$ */
9:         if $v \in V(G_i)$ then
10:            $\varphi(v) := \varphi_i(v) + (i-1) \cdot \vartheta'$
11:            $\vartheta := \vartheta' \cdot p$
12:        end if
13:    end for
14: end if
15: return $(\varphi, \vartheta)$   /* return the legal coloring and the num-
    ber of employed colors */
```

Let ϵ be an arbitrarily small positive constant. We execute Procedure Legal-Color on the input graph G of maximum degree Δ and neighborhood independence bounded by c, with the parameters $b = \left\lceil \Delta^{\epsilon/6} \right\rceil, p = \left\lceil \Delta^{\epsilon/3} \right\rceil, \lambda = \lceil \Delta^{\epsilon} \rceil, \Lambda = \Delta$. The running time analysis and the correctness proof of this invocation are provided in the next Lemmas.

Lemma 4.3. *Procedure Legal-Color invoked with the parameters $b = \left\lceil \Delta^{\epsilon/6} \right\rceil, p = \left\lceil \Delta^{\epsilon/3} \right\rceil, \lambda = \lceil \Delta^{\epsilon} \rceil, \Lambda = \Delta$ terminates in $O(\Delta^{\epsilon} + \log^* n)$ time.*

Proof. In each recursion level it holds that

$$\Lambda' = \lfloor (\Lambda/(b \cdot p) + \Lambda/p) \cdot c + c \rfloor \leq$$

$$(\Lambda/\Delta^{\epsilon/2} + \Lambda/\Delta^{\epsilon/3}) \cdot c + c \leq 3 \cdot c \cdot \Lambda/\Delta^{\epsilon/3}. \qquad (1)$$

Therefore, for a sufficiently large Δ, the number of recursion levels r satisfies

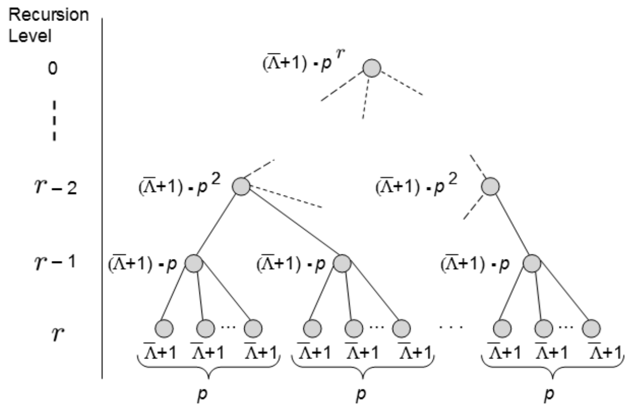
$$r \leq \log_{\Delta^{\epsilon/3}/3c} \Delta = \frac{\log \Delta}{\log(\Delta^{\epsilon/3}/3c)} = O(1). \qquad (2)$$

(Recall that both ϵ and c are constants.)

By Lemma 3.3, each recursion level, except for the last one, requires $O(\Delta^{\epsilon} + \log^* n)$ time. (This running time is dominated by the time required for executing Procedure Defective-Color in line 5 of Algorithm 2.) By Lemma 2.1(2), the last recursion level, in which $\Lambda \leq \lambda$, requires $O(\lambda + \log^* n) = O(\Delta^{\epsilon} + \log^* n)$ time. Therefore, the overall running time is also $O(\Delta^{\epsilon} + \log^* n)$. □

We remark that equation (1) in the proof of Lemma 4.3 holds for any (not necessarily constant) ϵ, $0 < \epsilon < 1$. On the other hand, for constant $\epsilon > 0$ this equation can be refined.

Lemma 4.4. *For any constant $\eta > 0$, for a sufficiently large Δ it holds that $\Lambda' \leq c \cdot (1 + \eta) \cdot \Lambda/\Delta^{\epsilon/3}$.*



Recursion Level
0
r − 2
r − 1
r
$(\bar{\Lambda}+1) \cdot p^r$
$(\bar{\Lambda}+1) \cdot p^2$
$(\bar{\Lambda}+1) \cdot p^2$
$(\bar{\Lambda}+1) \cdot p$
$(\bar{\Lambda}+1) \cdot p$
$(\bar{\Lambda}+1) \cdot p$
$\bar{\Lambda}+1$
p
p
p

Fig. 1. *The recursion tree. For each node α, the value that appears near α in the figure represents the number of colors that are employed by the recursive invocation that corresponds to α.*

Lemma 4.5. *Procedure Legal-Color invoked with the parameters $b = \left\lceil \Delta^{\epsilon/6} \right\rceil, p = \left\lceil \Delta^{\epsilon/3} \right\rceil, \lambda = \lceil \Delta^{\epsilon} \rceil, \Lambda = \Delta$ computes a legal $O(\Delta)$-coloring.*

Proof. Consider the recursion tree τ of Procedure Legal-Color invoked with the parameters $(G, b, p, \lambda, \Lambda)$, set as above. (See Figure 1.) With each node α of τ we associate the value Λ_α of the parameter Λ in the recursive invocation corresponding to α. By induction on the level i in the tree τ it can be shown that all nodes α of τ that have level i have the same value $\Lambda_\alpha = \Lambda^{(i)}$. The choice of parameters also guarantees that $\Lambda = \Lambda^{(0)} > \Lambda^{(1)} > \Lambda^{(2)} > ....$ Let r be the smallest index such that $\Lambda^{(r)} \leq \lambda$. Denote $\Lambda^{(r)} = \hat{\Lambda}$. Observe that nodes α of level r in τ are the leaves of τ, and every leaf of τ has level r. Hence all leaves of τ have the same level. Moreover, it follows that in each of theses leaf-invocations α of τ, the corresponding subgraph $G[\alpha]$ is colored with at most $\Lambda^{(r)} + 1$ colors, and that each of these invocations returns $\vartheta^{(r)} = \Lambda^{(r)} + 1$. Hence, by induction on j, $j = 1, 2, ..., r$, for any two invocations α and β of the j'th level in the recursion tree τ, these two invocations return the same values of ϑ. (The induction base is the case $j = r$, and in the step we proceed from $j + 1$ to j, for $j \in \{2, 3, ..., r\}$.)

For $j \in \{1, 2, ..., r\}$, denote by $\vartheta^{(j)}$ the parameter ϑ returned by some invocation α of level j. Next, we argue, again by the induction on i with the base case $j = r$, that the ϑ-coloring φ returned by the invocation α is legal. The base case follows from Lemma 2.1(2). For the induction step let $G[\alpha]$ denote the subgraph of G on which the invocation of Procedure Legal-Color that corresponds to the node α is invoked. Also, denote by $G_1[\alpha], G_2[\alpha], ..., G_p[\alpha]$ the p subgraphs of $G[\alpha]$ that correspond to the p children of α. We remark that, in general, line 5 of Procedure Legal-Color may partition the graph $G[\alpha]$ into less than p subgraphs. In this case, however, the routine leaves $\vartheta^{(j+1)}$ empty color classes for each of the "missing" subgraphs. It does so by setting $\vartheta^{(j)} = p \cdot \vartheta^{(j+1)}$, even though it really needs a smaller palette. These redundant colors help to maintain uniform bounds on the number of colors used to color subgraphs of the same recursion level.

For each $i \in \{1, 2, ..., p\}$, vertices of $G_i[\alpha]$ are colored by colors from the palette $\{(i-1) \cdot \vartheta^{(j+1)} + 1, (i-1) \cdot \vartheta^{(j+1)} + 2, i \cdot \vartheta^{(j+1)}\}$. Thus, for a pair of vertices u, w, $u \in G_i[\alpha]$, $w \in G_{i'}[\alpha]$, $i \neq i'$, $i, i' \in \{1, 2, ..., p\}$, the invocation α colors them by distinct colors. (Denote these colors by $\varphi[\alpha](u)$ and $\varphi[\alpha](w)$, respectively. Then $\varphi[\alpha](u) \neq \varphi[\alpha](w)$.) Consider also the case that $u, w \in G_i[\alpha]$ belong to the same subgraph $G_i[\alpha]$ of G_i, and $(u, w) \in E$. By the induction hypothesis, the coloring $\varphi_i[\alpha]$ of $G_i[\alpha]$ returned by the i'th child invocation of α is legal. Hence $\varphi_i[\alpha](u) \neq \varphi_i[\alpha](w)$. Recall that $\varphi[\alpha](u) = \varphi_i[\alpha](u) + (i-1) \cdot \vartheta^{(i+1)}$ and $\varphi[\alpha](w) = \varphi_i[\alpha](w) + (i-1) \cdot \vartheta^{(i+1)}$, and so $\varphi[\alpha](u) \neq \varphi[\alpha](w)$, as required.

Finally, we provide an estimate on the number of colors $\vartheta = \vartheta^{(0)}$ employed by the coloring φ returned by the root invocation Legal-Color$(G, b, p, \lambda, \Lambda)$ of the recursion tree τ. We have already shown that $\vartheta^{(r)} = \Lambda^{(r)} = \hat{\Lambda} + 1$. Also, for any level j, $0 \leq j \leq r-1$, $\vartheta^{(j)} = p \cdot \vartheta^{(j+1)}$. Hence the overall number of colors satisfies $\vartheta^{(0)} \leq \vartheta^{(r)} \cdot p^r = (\hat{\Lambda} + 1) \cdot p^r$. By equation (2) in proof of Lemma 4.3, it holds that $r \leq \log(\Delta/\hat{\Lambda}) / \log(\Delta^{\epsilon/3}/3c) = O(1)$. (This is true for a constant $\epsilon > 0$.) Finally, for a sufficiently large Δ,

$$(\hat{\Lambda} + 1) \cdot p^r$$

$$\leq (\hat{\Lambda} + 1) \cdot \left\lceil \Delta^{\epsilon/3} \right\rceil^r \leq (\hat{\Lambda} + 1) \cdot (3c + 1)^r \cdot (\frac{\Delta^{\epsilon/3}}{3c})^r$$

$$\leq (\hat{\Lambda} + 1) \cdot (3c + 1)^r \cdot (\frac{\Delta^{\epsilon/3}}{3c})^{\frac{\log(\Delta/\hat{\Lambda})}{\log(\Delta^{\epsilon/3}/3c)}}$$

$$= (3c + 1)^r \cdot (\hat{\Lambda} + 1) \cdot (\Delta/\hat{\Lambda}) = O(\Delta),$$

since $c, r = O(1)$, and $\frac{\hat{\Lambda}+1}{\hat{\Lambda}} \leq 2$. Hence $\vartheta^{(0)} = O(\Delta)$, completing the proof. □

It follows that for any constant $\epsilon > 0$, our algorithm computes an $O(\Delta)$-coloring of graphs with bounded neighborhood independence in time $O(\Delta^\epsilon + \log^* n)$. See also Theorem 4.2. Next, we show that one can compute a legal coloring of G much faster, at the expense of increasing the number of employed colors. To this end, one needs only to select different parameters for the invocation of Procedure Legal-Color.

Theorem 4.6. *For an arbitrarily small positive constant η, our algorithm computes an $O(\Delta^{1+\eta})$-coloring of graphs with bounded neighborhood independence, in time $O(\log \Delta \log^* n)$.*

PROOF. Let $t > 2$ be an arbitrarily large constant. Set $\lambda = (3c+1)^{6t}, b = \lambda^{1/3} = (3c+1)^{2t}, p = \lambda^{1/6} = (3c+1)^t, \Lambda = \Delta$. In each recursion level it holds that

$$\begin{aligned}\Lambda' &= \lfloor (\Lambda/(b \cdot p) + \Lambda/p) \cdot c + c \rfloor \\ &\le (\Lambda/(3c+1)^{3t} + \Lambda/(3c+1)^t) \cdot c + c \le 3 \cdot c \cdot \Lambda/(3c+1)^t.\end{aligned}$$

Therefore, the number of recursion levels in this case is at most $r = \log_{(3c+1)^{t-1}} \Delta$. By Corollary 3.3, invoking Procedure Defective-Color on each level requires $O((3c+1)^{6t} + \log^* n) = O(\log^* n)$ time. Other steps of Algorithm 2, except for the recursive invocation step, are executed locally, and require zero time. Hence each recursion level (except for the bottom level) requires $O(\log^* n)$ time. On the bottom level of the recursion we invoke a $(\Lambda + 1)$-coloring algorithm from [4] on subgraphs of maximum degree at most Λ. (See Lemma 2.1(2).) This algorithm requires $O(\Lambda + \log^* n)$ time. Note that at the bottom line of the recursion $\Lambda \le \lambda$, and $\lambda = (3c+1)^t = O(1)$ is a constant. Hence this running time is $O(\log^* n)$ as well. We conclude that the overall running time is $O(\log_{(3c+1)^{t-1}} \Delta \cdot \log^* n) = O(\log \Delta \log^* n)$.

The inductive proof of Lemma 4.5 is applicable as is also for the new selection of parameters. Hence, the produced coloring is legal. The number of colors in this case is at most

$$\begin{aligned}(\lambda + 1) \cdot p^r &= ((3c+1)^{6t} + 1) \cdot (3c+1)^{(t-1)\cdot r} \cdot (3c+1)^r \\ &\le ((3c+1)^{6t} + 1) \cdot \Delta \cdot \Delta^{1/(t-1)} = O(\Delta^{1+1/(t-1)}).\end{aligned}$$

Given a constant $\eta > 0$, set t to be sufficiently large so that $\frac{1}{t-1} < \eta$. Hence the number of colors is $O(\Delta^{1+\eta})$. □

We remark that our algorithms can be modified to guarantee running time of $O(\log \Delta \cdot \frac{\log^* \Delta}{\log(\log^* \Delta)}) + \frac{1}{2} \log^* n$, while maintaining the same bound of $O(\Delta^{1+\epsilon})$ on the number of colors. Also, a $\Delta^{1+o(1)}$-coloring can be computed in $O((\log \Delta)^{1+\epsilon}) + \frac{1}{2} \log^* n$ time. See the full version of this paper [6]. Consequently, we achieve the following results.

Theorem 4.7. *For any constant $\epsilon > 0$, and a graph G with bounded neighborhood independence: **(1)** an $O(\Delta)$-coloring of G can be computed in $O(\Delta^\epsilon) + \frac{1}{2} \log^* n$ time,*
***(2)** an $O(\Delta^{1+\epsilon})$-coloring of G can be computed in $O(\log \Delta \frac{\log^* \Delta}{\log(\log^* \Delta)}) + \frac{1}{2} \log^* n$ time,*
***(3)** a $\Delta^{1+o(1)}$-coloring of G can be computed in $O((\log \Delta)^{1+\epsilon}) + \frac{1}{2} \log^* n$ time.*

5. LEGAL EDGE COLORING IN GENERAL GRAPHS

In this section we show that the techniques described in Sections 3 and 4 can be used to devise very efficient edge coloring algorithms for *general* graphs. First, observe that every line graph is claw-free, hence its neighborhood independence is at most 2

Lemma 5.1. *For a graph $G = (V, E)$, the line graph $L(G)$ has neighborhood independence bounded by 2.*

Observe also that Lemma 5.1 extends directly to line graphs of general r-hypergraphs. Specifically, for any hypergraph $\mathcal{H}$, the neighborhood independence of the line graph $L(\mathcal{H})$ is at most r. It follows that our results for graphs of bounded neighborhood independence (Theorem 4.7) apply to line graphs of r-hypergraphs, for any constant positive integer r.

Observe that, by definition, for any graph G and positive integer k, a legal k-coloring of *vertices* of $L(G)$ is a legal k-coloring of *edges* of G, and vice versa. Note also that for an edge $e = (u, w)$ in G, the number of edges incident to it is $(\deg(u) - 1) + (\deg(w) - 1)$. Hence the maximum degree $\Delta(L(G))$ of the line graph $L(G)$ satisfies $\Delta(L(G)) \le 2(\Delta - 1)$, where $\Delta = \Delta(G)$. Consequently, if we are given a line graph $L(G)$ of a graph G with $\Delta(G) = \Delta$, our algorithm can compute an $O(\Delta(L(G))) = O(\Delta)$-vertex-coloring of $L(G)$ in $O(\Delta^\epsilon) + \frac{1}{2} \log^* n$ time, for any constant $\epsilon > 0$. Similarly, one can also compute $O(\Delta^{1+\eta})$-vertex-coloring (respectively, $\Delta^{1+o(1)}$-vertex-coloring) of $L(G)$ in $O(\log \Delta \frac{\log^* \Delta}{\log(\log^* \Delta)}) + \frac{1}{2} \log^* n$ (resp., $(\log \Delta)^{1+\zeta} + \frac{1}{2} \log^* n$) time, for $\eta, \zeta > 0$ being arbitrarily small positive constants. These vertex colorings give rise directly to edge coloring of G with the same number of colors.

On the other hand, in the distributed edge-coloring problem we are given as input the graph G, rather than its line graph $L(G)$. Nevertheless, one can simulate the distributed computation of an algorithm on $L(G)$ using the network $G = (V, E)$. To this end, each vertex of $L(G)$ is simulated by one endpoint of an appropriate edge in G. Consequently, a message sent over an edge of $L(G)$, will be sent over at most two (adjacent) edges in the simulation on G.

Lemma 5.2. *Any algorithm with running time T for the line graph $L(G)$ of the input graph G, can be simulated by G, and requires at most $2T + O(1)$ time.*

We apply Lemma 5.2 in conjunction with our results for vertex-coloring of $L(G)$, and obtain the following theorem.

Theorem 5.3. *For a graph $G = (V, E)$ with maximum degree Δ, and positive arbitrarily small constants $\epsilon, \eta, \zeta > 0$, our algorithm computes: **(1)** $O(\Delta)$-edge-coloring of G in $O(\Delta^\epsilon) + \frac{1}{2} \log^* n$ time,*
***(2)** $O(\Delta^{1+\eta})$-edge-coloring of G in $O(\log \Delta \frac{\log^* \Delta}{\log(\log^* \Delta)}) + \frac{1}{2} \log^* n$ time,*
***(3)** $\Delta^{1+o(1)}$-edge-coloring of G in $O((\log \Delta)^{1+\zeta}) + \frac{1}{2} \log^* n$ time.*

We remark that the simulation of a vertex-coloring algorithm on a line graph increases the size of messages by a factor of Δ. In the full version of this paper [6] we address this issue, and show that the result stated in Theorem 5.3 can be achieved with messages of size $O(\log n)$.

By Theorem 4.2 (using $c = 2$) we also get the following corollary.

Corollary 5.4. *For any constant $i = 1, 2, ...$, and any arbitrarily small $\epsilon > 0$, a $((2^i + \epsilon) \cdot \Delta)$-edge-coloring of a graph with maximum degree Δ can be computed in $O(\Delta^{\frac{2}{2+i}} + \log^* n)$ time.*

6. CONCLUSION AND OPEN QUESTIONS

We showed that an $O(\Delta)$-edge-coloring can be computed in $O(\Delta^{\epsilon} + \log^* n)$ time, for an arbitrarily small $\epsilon > 0$. Specifically, a $((4 + \epsilon)\Delta)$-edge-coloring can be computed in $O(\Delta^{2/3} + \log^* n)$ time, a $((8 + \epsilon)\Delta)$-edge-coloring can be computed in $O(\Delta^{1/2} + \log^* n)$ time, etc'. Improving this tradeoff is an interesting open problem. Another challenging problem is to obtain an $O(\Delta)$-edge-coloring algorithm that requires polylogarithmic time. Our algorithm constructs a $\Delta^{1+o(1)}$-edge-coloring in polylogarithmic time.

Acknowledgements

The authors thank Alessandro Panconesi for sending them several papers and addressing their questions.

7. REFERENCES

[1] B. Awerbuch, A. V. Goldberg, M. Luby, and S. Plotkin. Network decomposition and locality in distributed computation. In *Proc. of the 30th Symposium on Foundations of Computer Science*, pages 364–369, 1989.

[2] J. Andrews, and M. Jacobson. On a generalization of a chromatic number. *Congressus Numer*, 47:33-48, 1985.

[3] L. Barenboim, and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. In *Proc. of the 27th ACM Symp. on Principles of Distributed Computing*, pages 25–34, 2008.

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

[5] L. Barenboim, and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proc. of the 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. *http://arXiv.org/abs/1010.2454*, 2010.

[7] M. Chudnovsky, and P. Seymour. The structure of claw-free graphs. *Surveys in Combinatorics 2005, London Math Soc. Lecture Note Series*, 327:153-171, 2005.

[8] L. Cowen, R. Cowen, and D. Woodall. Defective colorings of graphs in surfaces: partitions into subgraphs of bounded valence. *Journal of Grah Theory*, 10:187–195, 1986.

[9] L. Cowen, W. Goddard, and C. Jesurum. Coloring with defect. In *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms, New Orleans, Louisiana, USA*, pages 548–557, January 1997.

[10] A. Czygrinow, M. Hanckowiak, and M. Karonski. Distributed $O(\Delta \log n)$-edge-coloring algorithm. In *Proc. of the 9th Annual European Symposium on Algorithms*, pages 345-355, 2001.

[11] D. Dubhashi, D. Grable, and A. Panconesi. Nearly-optimal, distributed edge-colouring via the nibble method. *Theoretical Computer Science*, 203:225–251, 1998.

[12] D. Durand, R. Jain, and D. Tseytlin. Applying randomized edge coloring algorithms to distributed communication: an experimental study. In *Proc. of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 264–274, 1995.

[13] B. Gfeller, and E. Vicari. A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In *Proc. of the 26th ACM Symp. on Principles of Distributed Computing*, pages 53–60, 2007.

[14] D. Grable, and A. Panconesi. Nearly optimal distributed edge coloring in $O(\log \log n)$ rounds *Random Structures and Algorithms*, 10(3):385–405, 1998.

[15] F. Harary, and K. Jones. Conditional colorability II: Bipartite variations. *Congressus Numer*, 50:205-218, 1985.

[16] R. Jain, K. Somalwar, J. Werth, and J. C. Browne. Scheduling parallel I/O operations in multiple bus systems. *ELSEVIER Journal of Parallel and Distributed Computing*, 16(4):352-362, 1992.

[17] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the Locality of Bounded Growth. In*Proc. of the 24rd ACM Symp. on Principles of Distributed Computing*, pages 60–68, 2005.

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

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

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

[21] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[22] N. Linial and M. Saks. Low diameter graph decomposition. *Combinatorica* 13: 441 - 454, 1993.

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

[24] A. Panconesi, and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):581-Ű592, 1995.

[25] A. Panconesi, and A. Srinivasan. Distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. on Computing*, 26(2):350–368, 1997.

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

[27] J. Schneider, and R. Wattenhofer. A log-star distributed Maximal Independent Set algorithm for Growth Bounded Graphs. In *Proc. of the 27th ACM Symp. on Principles of Distributed Computing*, pages 35–44, 2008.

[28] J. Schneider, and R. Wattenhofer. A new technique for distributed symmetry breaking. To appear in *Proc. of the 29th ACM Symp. on Principles of Distributed Computing*, 2010.

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