Open access • Proceedings Article • DOI:10.1145/1835698.1835797

Deterministic distributed vertex coloring in polylogarithmic time — Source link

Leonid Barenboim, Michael Elkin

Institutions: Ben-Gurion University of the Negev

Published on: 25 Jul 2010 - Principles of Distributed Computing

Topics: Vertex (graph theory), Deterministic algorithm and Arboricity

Deterministic Distributed Vertex Coloring in Polylogarithmic Time

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

Consider an n-vertex graph $G = (V, E)$ of maximum degree Δ, and suppose that each vertex $v \in V$ hosts a processor. The processors are allowed to communicate only with their neighbors in G. The communication is synchronous, i.e., it proceeds in discrete rounds.

In the distributed *vertex coloring* problem the objective is to color G with $\Delta + 1$, or slightly more than $\Delta + 1$, colors using as few rounds of communication as possible. (The number of rounds of communication will be henceforth referred to as *running time*.) Efficient *randomized* algorithms for this problem are known for more than twenty years [1, 19]. Specifically, these algorithms produce a $(\Delta+1)$-coloring within $O(\log n)$ time, with high probability. On the other hand, the best known *deterministic* algorithm that requires polylogarithmic time employs $O(\Delta^2)$ colors. This algorithm was devised in a seminal FOCS'87 paper by Linial [16]. Its running time is $O(\log^* n)$. In the same paper Linial asked whether one can color with significantly less than Δ^2 colors in deterministic polylogarithmic time. By now this question of Linial became one of the most central long-standing open questions in this area.

In this paper we answer this question in the affirmative, and devise a deterministic algorithm that employs $\Delta^{1+o(1)}$ colors, and runs in *polylogarithmic* time. Specifically, the running time of our algorithm is $O(f(\Delta) \log \Delta \log n)$, for an arbitrarily slow-growing function $f(\Delta) = \omega(1)$. We can also produce $O(\Delta^{1+\eta})$-coloring in $O(\log \Delta \log n)$-time, for an arbitrarily small constant $\eta > 0$, and $O(\Delta)$-coloring in $O(\Delta^{\epsilon} \log n)$ time, for an arbitrarily small constant $\epsilon > 0$. Our results are, in fact, far more general than this. In particular, for a graph of arboricity a, our algorithm produces an $O(a^{1+\eta})$-coloring, for an arbitrarily small constant $\eta > 0$, in time $O(\log a \log n)$.

*This research has been supported by the Israeli Academy of Science, grant 483/06, and by the Binational Science Foundation, grant No. 2008390.

PODC'10, July 25–28, 2010, Zurich, Switzerland.

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

Arboricity, Arbdefective-Coloring, Partial-Orientation

1. INTRODUCTION

1.1 Background and Previous Research

In the message passing model of distributed computing the network is modeled by an n-vertex undirected unweighted graph $G = (V, E)$, with each vertex hosting its own processor with a unique identity number. These numbers are assumed to belong to the range $\{1, 2, ..., n\}$. Initially, each vertex v knows only its identity number $id(v)$. The vertices communicate over the edges of E in the *synchronous manner*. Specifically, computations (or equivalently, algorithms) proceed in discrete rounds. In each round each vertex v is allowed to send a message to each of its neighbors. All messages that are sent in a certain round arrive to their destinations before the next round starts. The number of rounds that elapse from the beginning of the algorithm until its end is called the *running time* of the algorithm.

In the *vertex coloring* problem one wants to color the vertices of V in such a way that no edge becomes monochromatic. It is very easy to color a graph G of maximum degree $\Delta = \Delta(G)$ in $\Delta + 1$ colors using n rounds. Coloring it in $\Delta+1$, or slightly more than $\Delta+1$, colors far more efficiently is one of the most central and fundamental problems in distributed computing. In addition to its theoretical appeal, the problem is also very well motivated by various real-life network tasks [21].

The vertex coloring problem is also closely related to the *maximal independent set* (henceforth, MIS) problem. A subset $U \subseteq V$ is an *independent set* if there is no edge $(u, u') \in E$ with both endpoints in U. It is an MIS if for every vertex $v \in V \setminus U$, the set $U \bigcup \{v\}$ is not an independent set. A classical reduction of Linial [17] shows that given a (distributed) algorithm for computing an MIS on general graphs, one can obtain a $(\Delta + 1)$-coloring within the same time.

The (distributed) vertex coloring and MIS problems have been intensively studied since the mid-eighties. Already in

1986 Luby [19] and Alon, Babai and Itai [1] devised randomized algorithms for the MIS problem that require $O(\log n)$ time. Using Linial's reduction [17] these results imply that $(\Delta+1)$-coloring can also be computed in randomized logarithmic time. More recently, Kothapalli et al. [13] devised a randomized $O(\Delta)$-coloring algorithm that requires $O(\sqrt{\log n})$ time. On the other hand, the best known *deterministic* algorithm that requires polylogarithmic time employs $O(\Delta^2)$ colors. Specifically, its running time is $O(\log^* n)$[1]. This algorithm was devised in a seminal FOCS'87 paper of Linial [17]. In the end of this paper Linial argued that his method cannot be used to reduce the number of colors below $\binom{\Delta+2}{2}$, and asked whether this can be achieved by other means. Specifically, he wrote

"Proposition 3.4 of [EFF] shows that set systems of the type that would allow further reduction of the number of colors do not exist. Other algorithms may still be capable of coloring with fewer colors. ***It would be interesting to decide whether this quadratic bound can be improved when time bounds rise from*** $O(\log^* n)$ ***to polylog, for instance.***"

By now, almost quarter a century later, this open question of Linial became one of the most central long-standing open questions in this area.

1.2 Our Results

In this paper we answer this question in the affirmative. Specifically, for an arbitrarily small constant $\eta > 0$, our algorithm constructs an $O(\Delta^{1+\eta})$-coloring in $O(\log\Delta\log n)$ time. Moreover, we show that one can trade time for the number of colors, and devise a $\Delta^{1+o(1)}$-coloring algorithm with running time $O(f(\Delta)\log\Delta\log n)$, where $f(\Delta)=\omega(1)$ is an arbitrarily slowly-growing function of Δ. Also, our algorithm can produce an $O(\Delta)$-coloring in $O(\Delta^\epsilon\log n)$ time, for an arbitrarily small constant $\epsilon>0$. All our algorithms are deterministic.

Currently, the state-of-the-art bound for deterministic $O(\Delta^{1+\eta})$-coloring (respectively, $\Delta^{1+o(1)}$-coloring; resp., $O(\Delta)$-coloring) is $\min\{O(\Delta^{1-\eta}+\log^* n), 2^{O(\sqrt{\log n})}\}$ (respectively, $\min\{\Delta^{1-o(1)}+O(\log^* n), 2^{O(\sqrt{\log n})}\}$; resp., $\min\{O(\Delta+\log^* n), 2^{O(\sqrt{\log n})}\}$). (Algorithms that produce $O(\Delta\cdot t)$-coloring in $O(\Delta/t+\log^* n)$ time were devised in [4] and in [14]. The algorithm of [22] requires $2^{O(\sqrt{\log n})}$ time.) Our results constitute an *exponential* improvement over this state-of-the-art for large values of Δ (i.e., $\Delta=2^{\Omega(\log^\epsilon n)}$, for some constant $\epsilon>0$), and a significant improvement in the wide range of $\Delta=\log^{1+\Omega(1)} n$.

In addition, our results are, in fact, far more general than described above. Specifically, we consider graphs with bounded *arboricity* [2] rather than bounded degree. This is a much wider family of graphs that contains, in addition to graphs of bounded degree, the graphs of bounded genus, bounded tree-width, graphs that exclude a fixed minor, and many other graphs. All the results that we have stated above apply to graphs of arboricity at most a. (One just needs to replace Δ by a in the statements of all results. We remark that a graph with maximum degree Δ has arboricity at most Δ as well.) One interesting consequence of this extension is that if the arboricity a and the degree Δ of a graph are polynomially separated one from another (i.e., if there exists a constant $\nu>0$ such that $a\le\Delta^{1-\nu}$) then our algorithm constructs a $(\Delta+1)$-coloring [3] in $O(\log a\cdot\log n)=O(\log\Delta\cdot\log n)$ time.

We also show that one can decrease the running time further almost all the way to $\log n$, while still having less than a^2 colors. Specifically, we show that in $O(\log\log a\log n)$ time one can construct an $O(a^2/\log^C a)$-coloring, for an arbitrarily large constant C. More generally, for any function $\omega(1)=f(a)=o(\log a)$, one can construct an $O(a^2/2^{f(a)})$-coloring in $O(f(a)\log n)$ time.

Our algorithms for coloring graphs of arboricity a also compare very favorably with the current state-of-the-art. Specifically, the fastest algorithm known today for $O(a)$-coloring [3] requires $O(a\log n)$ time. Our algorithm produces $O(a)$-coloring in $O(a^\epsilon\log n)$ time, for arbitrarily small constant $\epsilon>0$. The best known tradeoff between the number of colors and the running time (also due to [3]) is $O(a\cdot t)$-coloring in $O(\frac{a}{t}\log n+\log n+a)$ time. We improve this tradeoff and show that $O(a\cdot t)$-coloring can be computed in just $O((\frac{a}{t})^\epsilon\log n)$ time, for an arbitrarily small constant $\epsilon>0$. In some points on the tradeoff curve the improvement is even greater than that. For example, we compute an $O(a^{1+\eta})$-coloring, for an arbitrarily small $\eta>0$, in $O(\log a\log n)$ time, while the previous bound was $O(a^{1-\eta}\log n+a)$ time. Similarly, our $a^{1+o(1)}$-coloring algorithm requires $O(f(a)\log a\log n)$ time, for any function $f(a)=\omega(1)$, while the previous bound was $a^{1-o(1)}\cdot\log n+O(a)$ time.

Finally, our results imply improved bounds for the deterministic MIS problem on graphs of bounded arboricity a. Specifically, our algorithm produces an MIS in time $O(a+a^\epsilon\log n)$, for an arbitrarily small constant $\epsilon>0$. The previous state-of-the-art is $\min\{O(a\sqrt{\log n}+\log n),\ 2^{O(\sqrt{\log n})}\}$ due to [3, 22]. Hence our result is stronger in the wide range $\log^{1/2+\Omega(1)} n\le a\le 2^{c\sqrt{\log n}}$, for some universal constant $c>0$.

1.3 Our Techniques and Overview of the Proof

We employ a combination of a number of different existing approaches, together with a number of novel ideas. The first main building block is the machinery for constructing distributed forests decomposition, developed by us in a previous paper [3]. Specifically, it is known [3] that a graph $G=(V,E)$ of arboricity a can be efficiently decomposed into $O(a)$ edge-disjoint forests in $O(\log n)$ time. Moreover, these forests come with a *complete acyclic orientation* of the edges of E. In other words, both endpoints u and v of every edge $e=(u,v)\in E$ know the identity of the forest F to which the edge e ends up to belong, and the parent-child relation of u and v in F. In addition, this forests decomposition comes along with another useful graph decomposition, called *H-partition*. Roughly speaking, an H-partition is a decomposition of the *vertex* set V of G into $\ell=O(\log n)$ vertex sets $H_1,H_2,...,H_\ell$, such that each $G(H_i)$, $i=1,2,...,\ell$ is a graph with maximum degree $O(a)$. (See Section 2.2 for more details.) This decomposition is extremely useful, as it allows one to apply algorithms that were devised for graphs of bounded degree on graphs of bounded arboricity. We will discuss this point further below.

The second main building block is the suite of algorithms

[1] $\log^* n$ is the smallest integer t such that the t-iterated logarithm of n is no greater than 2, i.e., $\log^{(t)} n\le 2$.

[2] The *arboricity* of a graph $G=(V,E)$ is the minimal number a such that the edge set E of G can be covered with at most a edge disjoint forests.

[3] Actually, even $o(\Delta)$-coloring.

for constructing *defective colorings*, developed by us in another previous paper (in STOC'09 [4, 5]), and by Kuhn (in SPAA'09 [14]). These algorithms enable one to efficiently decompose a graph G of maximum degree Δ into $h = O(t^2)$ subgraphs $G'_1, G'_2, ..., G'_h$, of maximum degree $\Delta' = O(\Delta/t)$ each. This decomposition was used in [4, 14] for devising $(\Delta+1)$-coloring algorithms that run in $O(\Delta + \log^* n)$ time. It is a natural idea to construct this decomposition and then to recurse on each of the subgraphs. However, unfortunately, the product $h \cdot \Delta'$ may be significantly larger than Δ. Hence, in this simplistic form this approach is doomed either to have a large running time or to use prohibitively many colors.

The approach that we employ in this paper is based on *arbdefective colorings*. While defective coloring is a classical graph-theoretic notion [6, 11], arbdefective coloring is a new concept that we introduce in this paper. It generalizes the notion of defective coloring. A coloring φ is an *r-arbdefective k-coloring* if it employs k colors, and each color class induces a subgraph of *arboricity* at most r. We demonstrate that in a graph G of arboricity a, an r-arbdefective k-coloring with $r \cdot k = O(a)$ can be efficiently computed. Here the combination of parameters is significantly better than in the case of defective colorings, and consequently, recursing on each of the subgraphs gives rise to an efficient algorithm for $O(a)$-coloring of the original graph G.

A key idea of our $O(a)$-coloring algorithm is a partition of the input graph (whose arboricity is a) into k subgraphs of smaller arboricity $O(a/k)$ each, for a certain positive parameter k. Note that the product of the number of subgraphs and the arboricity in each subgraph is $O(a)$. Then each of the subgraphs is partitioned again into subgraphs of yet smaller arboricity. However, the product of the number of subgraphs and the arboricity of each subgraph still remains $O(a)$. This refinement procedure of partitioning is repeated for several phases until the arboricity of all subgraphs is sufficiently small. Consequently, we achieve an r'-arbdefective k'-coloring φ, for a sufficiently small integer r'. Then we invoke a known algorithm form [3] for coloring graphs of bounded arboricity on each of the k' subgraphs induced by the color classes of φ in parallel. Since the arboricity of each of these k' subgraphs is at most r', this step requires $O(r' \log n)$ time, and provides $O(r')$-coloring for each subgraph. It is then easy to combine these colorings into a unified $O(r' \cdot k') = O(a)$-coloring of the original graph.

Another intricate part of our argument is the routine that computes an $O(\Delta/t)$-arbdefective t-coloring of a graph G with maximum degree at most Δ. In this part of the proof we manipulate with orientations in a novel way. A *complete orientation* σ assigns a direction to each edge $e = (u, w)$ of G. Orientations play a central role in the theory of distributed graph coloring [7, 17, 13]. We introduce the notion of *partial orientations*. A partial orientation σ is allowed not to orient some edges of the graph. Specifically, we say that σ has *deficit* at most d, for some positive integer parameter d, if for every vertex v in the graph the number of edges incident to v that σ does not orient is no greater than d. Another important parameter of an orientation σ is its *length*, defined as the length of the longest path P in which all edges are oriented consistently according to σ. We demonstrate that partial orientations with appropriate deficit and length parameters can be constructed efficiently. Moreover, these orientations turn out to be extremely useful for computing arbdefective colorings. We believe that the notion of partial orientation, and our technique of constructing these orientations are of independent interest.

1.4 Related Work

There is an enormous amount of literature on distributed graph coloring. Already before the work of Linial, Cole and Vishkin [7] devised a deterministic 3-coloring algorithm with running time $O(\log^* n)$ for oriented [1] rings. In STOC'87 Goldberg and Plotkin [9] generalized the algorithm of [7] and obtained a $(\Delta+1)$-coloring algorithm with running time $2^{O(\Delta)} + O(\log^* n)$. Also, Goldberg, Plotkin, and Shannon [10] devised a $(\Delta+1)$-coloring algorithm with running time $O(\Delta \log n)$. In FOCS'89 Awerbuch, Goldberg, Luby and Plotkin [2] devised a $2^{O(\sqrt{\log n \log\log n})}$-time deterministic algorithm for the MIS, and consequently, for the $(\Delta+1)$-coloring problem. In STOC'92 Panconesi and Srinivasan [22] improved this upper bound to $2^{O(\sqrt{\log n})}$. More recently, in PODC'06 Kuhn and Wattenhofer [15] devised a $(\Delta+1)$-coloring algorithm with running time $O(\Delta \log \Delta + \log^* n)$. In STOC'09 Barenboim and Elkin [4, 5], and independently Kuhn [14] in SPAA'09, devised a $(\Delta+1)$-coloring algorithm with running time $O(\Delta + \log^* n)$.

Another related thread of study is the theory of distributed graph decompositions. (See the book of Peleg [21] for an excellent in-depth survey of this topic.) In particular, Awerbuch et al. [2] and Panconesi and Srinivasan [22] showed that any n-vertex graph G can be efficiently decomposed into disjoint regions of diameter $2^{O(\sqrt{\log n})}$, so that the super-graph induced by contracting each region into a super-vertex has arboricity $2^{O(\sqrt{\log n})}$. (See also [18].) Note, however, that these decompositions are inherently different from the ones that we develop, in a number of ways.

We remark that the algorithmic scheme of [2, 18, 22] that utilizes graph decompositions for computing colorings stipulates that on each round only a small portion of all vertices (specifically, vertices that belong to regions of a given color) are active. This approach is inherently suboptimal, as it does not exploit the network parallelism to the fullest possible extent. In our approach, on the contrary, once the original graph is decomposed into subgraphs the algorithm recurses *in parallel on all subgraphs*. In this way all vertices are active at (almost) [2] all times. This extensive utilization of parallelism is the key to the drastically improved running time of our algorithms.

1.5 Structure of the Paper

The remainder of the paper is organized as follows. Section 2 contains basic definitions and notation, and statements of a few known results about forests-decomposition. Section 3 describes the algorithms for computing arbdefective colorings. Section 4 contains various efficient algorithms that produce legal coloring. Section 5 presents even faster algorithms that, however, employ more colors.

2. PRELIMINARIES

2.1 Definitions and Notation

Unless the base value is specified, all logarithms in this paper are to base 2.

[1] In an *oriented* ring each vertex v knows which of its two neighbors is located in the clockwise direction from v, and which is located in the counter-clockwise direction.

[2] In some branches the recursion may proceed faster than in others. This may result in some vertices becoming idle sooner than other vertices.

The *out-degree* of a vertex v in a directed graph is the number of edges incident to v that are oriented out 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. A *partial orientation* is an orientation of a subset $E' \subseteq E$. Edges in $E \setminus E'$ have no orientation. The *length* of a vertex v with respect to an orientation σ, denoted $len(v) = len_\sigma(v)$, is the length ℓ of the longest directed path $< v = v_0, v_1, ..., v_\ell >$ that emanates from v, where all edges (v_i, v_{i+1}), for $i = 0, 1, ...\ell - 1$, are oriented by σ towards v_{i+1}. The *length* of a (partial) orientation σ, denoted $len(\sigma)$, is the maximum length of a vertex v with respect to the orientation. The *deficit* of a vertex v with respect to a partial orientation σ is the number of edges e that are unoriented by σ, and incident to v. The *deficit of* σ is the maximum deficit of a vertex $v \in V$ with respect to σ. 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.

A coloring $\varphi : V \to \mathbb{N}$ that satisfies $\varphi(v) \neq \varphi(u)$ for each edge $(u,v) \in E$ is called a *legal coloring*. The minimum number of colors that can be used in a legal coloring of a graph G is called *the chromatic number* of G. It is denoted $\chi(G)$.

An *m-defective p-coloring* of a graph G is a coloring of the vertices of G using p colors, such that each vertex has at most m neighbors colored by its color. Each color class in the m-defective coloring induces a graph of maximum degree m. It is known that for any positive integer parameter p, an $\lfloor \Delta/p \rfloor$-defective $O(p^2)$-coloring can be efficiently computed distributively [4, 5, 14].

Lemma 2.1. *[14] A $\lfloor \Delta/p \rfloor$-defective $O(p^2)$-coloring can be computed in $O(\log^* n)$ time.*

We conclude this section by defining the notion of arbdefective coloring. This notion generalizes the notion of defective coloring.

Definition 2.1. An *r-arbdefective k-coloring* is a coloring with k colors such that all the vertices colored by the same color i, $1 \leq i \leq k$, induce a subgraph of arboricity at most r.

2.2 Forests-Decomposition

A *k-forests-decomposition* is a partition of the edge set of the graph into k subsets, such that each subset forms a forest. Efficient distributed algorithms for computing $O(a)$-forests decompositions have been devised recently in [3]. Several results from [3] are used in the current paper. They are summarized in the following lemmas.

Lemma 2.2. *[3] (1) For any graph G, a legal $(\lfloor (2+\epsilon) \cdot a \rfloor + 1)$-coloring of G can be computed in $O(a \log n)$ time, for an arbitrarily small positive constant ϵ.*

(2) For any graph G, an $O(a)$-forests-decomposition can be computed in $O(\log n)$ time.

Moreover, the algorithm in [3] for computing forests- decompositions produces a vertex partition with a certain helpful property, called an H-partition. An *H-partition* is a partition of V into subsets $H_1, H_2, ..., H_\ell$, $\ell = O(\log n)$, such that each vertex in H_i, $1 \leq i \leq \ell$, has at most $O(a)$ neighbors in $\bigcup_{j=i}^{\ell} H_j$. The *degree of the H-partition* is the maximum number of neighbors of a vertex $v \in H_i$ in $\bigcup_{j=i}^{\ell} H_j$ for $1 \leq i \leq \ell$. For a vertex $v \in V$, the *H-index* of v is the index i, $1 \leq i \leq \ell$, such that $v \in H_i$.

Lemma 2.3. *[3] For any graph G, an H-partition of the vertex set of G can be computed in $O(\log n)$ time. The degree of the computed H-partition is $\lfloor (2+\epsilon) \cdot a \rfloor$, for an arbitrarily small positive constant ϵ.*

The H-partition is used to compute an acyclic orientation such that each vertex has out-degree $O(a)$.

Lemma 2.4. *[3] For any graph G, an acyclic complete orientation with out-degree $O(a)$ can be computed in $O(\log n)$ time.*

Finally, the relationship between arboricity and acyclic orientation is given in the following lemma.

Lemma 2.5. *[3, 8] If there exists an acyclic complete orientation of G with out-degree k, then $a(G) \leq k$.*

3. SMALL ARBORICITY DECOMPOSITION

We begin with presenting a simple algorithm that computes an $O(a/k)$-arbdefective k-coloring for any integer parameter $k > 0$. (In other words, it computes a vertex decomposition into k subgraphs such that each subgraph has arboricity $O(a/k)$.) The running time of our first algorithm is $O(a \log n)$. Later, we present an improved version of the algorithm with a significantly faster running time.

Suppose that we are given an acyclic complete orientation of the edge set of G, such that each vertex has at most m outgoing edges, for a positive parameter m. The following procedure, called *Procedure Simple-Arbdefective* accepts as input such an orientation and a positive integer parameter k. During its execution, each vertex computes its color in the following way. The vertex waits for all its parents to select their colors. (Recall that a parent of a vertex v is a neighbor u of v connected by an edge $\langle v, u \rangle$ that is oriented towards u.) Once the vertex receives a message from each of its parents containing their selections, it selects a color from the range $\{1, 2, ..., k\}$ that is used by the minimum number of parents. (In particular, if there is a color in this range that is not used by any parent, then such a color is selected.) Then it sends its selection to all its neighbors. This completes the description of the procedure.

Let c be the color that a vertex v has selected. Since v has at most m parents, by the Pigeonhole Principle, the number of parents colored by the color c is at most $\lfloor m/k \rfloor$. For $c = 1, 2, ..., k$, consider the subgraph G_c induced by all the vertices that have selected the color c. For each edge e in G_c, orient e in the same way it is oriented in the original graph G. The orientation in G_c is therefore acyclic, and each vertex in G_c has out-degree at most $\lfloor m/k \rfloor$. Thus, the arboricity of G_c is at most $\lfloor m/k \rfloor$ (See Lemma 2.5). Hence Procedure Simple-Arbdefective has produced an $\lfloor m/k \rfloor$-arbdefective k-coloring.

Next, we consider a more general scenario in which instead of accepting as input a complete acyclic orientation we are given a partial acyclic orientation. Specifically, the orientation that Procedure Simple-Arbdefective accepts as input has out-degree at most m, and deficit at most τ. (Observe that Procedure Simple-Arbdefective is applicable as is with a partial orientation as input. If a partial orientation is given as input instead of a complete orientation, a vertex still waits for its parents before selecting a color. Again, the

selected color is the color used by the minimum number of neighbors. However, the number of parents may be smaller than in a complete orientation.) Once the procedure is invoked on such an orientation and a parameter k as input, a coloring with k colors is produced. Consider the graph G_c induced by all the vertices that are colored by the color c, $1 \leq c \leq k$. Each edge in G_c is oriented in the same way as in G. Each vertex in G_c has at most $\lfloor m/k \rfloor$ parents, and at most τ unoriented edges connected to it in G_c. The following lemma states that it is possible to orient all unoriented edges of G_c to achieve a complete acyclic orientation.

Lemma 3.1. *Any acyclic partial orientation σ of a graph $G = (V, E)$ can be transformed into a complete acyclic orientation by adding orientation to unoriented edges.*

PROOF. Let $\hat{E}$ be the set of all edges oriented by σ. Since σ is acyclic, the graph $\hat{G} = (V, \hat{E})$ is a directed acyclic graph. Perform a topological sort of $\hat{G}$ such that for any edge $\langle u, v \rangle$ that is oriented towards v, the vertex v is placed after u. Orient each unoriented edge (w, z) in G towards the endpoint that appears later in the topological sorting of $\hat{G}$. It is easy to see that the resulting orientation is a complete acyclic orientation of G. □

Once the unoriented edges of G_c are oriented as in the proof of Lemma 3.1, each vertex v in G_c has an out-degree at most $\tau + \lfloor m/k \rfloor$. (Recall that v had at most τ unoriented edges incident to it.) Hence, by Lemma 2.5, the arboricity of G_c is at most $\tau + \lfloor m/k \rfloor$. The next Theorem summarizes the properties of Procedure Simple-Arbdefective.

Theorem 3.2. *Suppose that Procedure Simple-Arbdefective is invoked with the following two input arguments:*
(1) An acyclic (partial) orientation of length ℓ, out-degree at most m, and deficit at most τ.
(2) An integer parameter $k > 0$.
Then Procedure Simple-Arbdefective produces a $(\tau + \lfloor m/k \rfloor)$-arbdefective k-coloring in $O(\ell)$ time.

PROOF. By Lemmas 2.5, 3.1, the arboricity of G_c, for $1 \leq k \leq c$, is at most $\tau + \lfloor m/k \rfloor$. Hence, Procedure Simple-Arbdefective produces a $(\tau + \lfloor m/k \rfloor)$-arbdefective k-coloring.

Next, we analyze the running time of Procedure Simple-Arbdefective. We prove by induction on i that after i rounds, all vertices $v \in V$ with $len(v) \leq i$, have selected their colors.

Base ($i = 0$)**:** The vertices v with $len(v) = 0$ are the vertices that have no outgoing edges. Such vertices select an arbitrary color from the range $\{1, 2, ..., k\}$ immediately after the algorithm starts, requiring no communication whatsoever.

Induction step: Assume that after $i - 1$ rounds, all vertices $v \in V$ with $len(v) \leq i - 1$, have selected their colors. Let u be a vertex with $len(u) \leq i$. Then, for each parent w of u, it holds that $len(u) \leq i - 1$. Consequently, by the induction hypothesis, all parents of u select their color after at most $i - 1$ rounds. Therefore, the vertex u is aware of the selection of all its parents on round i or before. Hence, after i rounds the vertex u necessarily selects a color. This completes the inductive proof.

If Procedure Simple-Arbdefective accepts as input an acyclic orientation of length ℓ, then all directed paths are of length at most ℓ. Consequently, all vertices select their color after at most ℓ rounds. □

For Procedure Simple-Arbdefective to be useful, we need to compute partial acyclic orientations with small length and out-degree. Next, we devise efficient algorithms for computing appropriate acyclic orientations. First, we devise a distributed algorithm that receives as input an undirected graph G, and computes a complete acyclic orientation such that each vertex has out-degree $O(a)$. Observe that in a distributed computation of an orientation, each vertex has to compute only the orientation of edges incident to it, as long as the global solution formed by this computation is correct. The algorithm we devise is called *Procedure Complete-Orientation*.

Procedure Complete-Orientation consists of three steps. First, an H-partition of the input graph G is computed. (See Section 2.2.) As a consequence, the vertex set of G is partitioned into $\ell' = O(\log n)$ subsets $H_1, H_2, ..., H_{\ell'}$, such that each vertex in H_i, $1 \leq i \leq \ell'$, has $O(a)$ neighbors in $\bigcup_{j=i}^{\ell'} H_j$. Next, each subgraph induced by a set H_i is colored legally using $O(a)$ colors. Finally, an orientation is computed as follows. Consider an edge (u, v) such that $u \in H_i$ and $v \in H_j$ for some $1 \leq i, j \leq \ell'$. If $i < j$, orient the edge towards v. If $j < i$ orient the edge towards u. Otherwise $i = j$. In this case the vertices u and v have different colors. Orient the edge towards the vertex that is colored with a greater color. This completes the description of the procedure. We summarize the properties of Procedure Complete-Orientation in the following lemma.

Lemma 3.3. *The running time of Procedure Complete-Orientation is $O(a + \log n)$. It produces a complete acyclic orientation with out-degree $\lfloor (2 + \epsilon) \cdot a \rfloor$ for an arbitrarily small constant $\epsilon > 0$, and length $O(a \log n)$.*

PROOF. By Theorem 2.3, the first step, in which the H-partition is computed, requires $O(\log n)$ time. The second step consists of coloring graphs of maximum degree $O(a)$. All the colorings are performed in parallel in $O(a + \log^* n)$ time using the algorithm from [4]. The orientation step requires a single round in which vertices learn the colors and the H-indices of their neighbors. To summarize, the total running time is $O(a + \log n)$.

Next, we show that the out-degree of each vertex is $O(a)$. Consider a vertex $v \in H_i$. Each outgoing edge of v is connected to a vertex in a set H_j such that $j \geq i$. By Lemma 2.3, v has at most $\lfloor (2 + \epsilon) \cdot a \rfloor$ neighbors in $\bigcup_{j=i}^{\ell'} H_j$. Therefore, the out-degree of v is $\lfloor (2 + \epsilon) \cdot a \rfloor$. Next, we show that the length of the orientation is $O(a \log n)$. Consider a subgraph G_i induced by a set H_i, $1 \leq i \leq \ell'$. Each edge in G_i is oriented towards a vertex with a greater color among its two endpoints. Hence, a certain color appears in any directed path at most once. Consequently, the length of the longest directed path in G_i is less than the number of colors used for coloring G_i. Since G_i is colored using $O(a)$ colors, the length of the longest directed path in G_i is $O(a)$. Consider an edge (u, v) such that $u \in H_i, v \in H_j, i < j$. The edge (u, v) is oriented towards the set with the greater index H_j. Therefore, each directed path in G has at most $\ell' - 1$ edges whose endpoints belong to different H-sets. Inside any path, each two edges whose endpoints belong to different H-sets are separated be $O(a)$ consequent edges whose endpoints belong to the same H-set. Therefore, the length of any directed path is $O(a \cdot \ell') = O(a \log n)$. □

Theorem 3.2 and Lemma 3.3 imply the following corollary.

Corollary 3.4. *For an integer $k > 0$, an $O(a/k)$-arbdefective k-coloring can be computed in $O(a \log n)$ time.*

The running time of Procedure Simple-Arbdefective is proportional to the length of the acyclic orientation that is given to the procedure as part of its input. Hence, to improve its running time, we have to compute a much shorter orientation. However, the shortest complete acyclic orientation of a graph G is of length at least $\chi(G)-1$. (Since a complete acyclic orientation of length ℓ allows one to color the graph legally with $\ell+1$ colors.) There exist graphs for which $\chi(G) = \Omega(a)$. For example, the chromatic number of a complete graph on n vertices is n, and its arboricity is $\lceil n/2 \rceil$. Consequently, an acyclic complete orientation of length $o(a)$ does not always exist. We overcome this difficulty by computing a partial acyclic orientation instead. This partial orientation is significantly shorter, and its deficit is sufficiently small. Moreover, we show that a partial orientation can be computed considerably faster than a complete orientation. Also, in the computation of a partial orientation it is no longer required that the H-sets are legally colored, which is the case in Procedure Complete-Orientation. Instead it suffices to color the H-sets with a defective coloring, and this can be done far more efficiently. (See Lemma 2.1.)

The pseudocode of the algorithm for computing short acyclic orientations, called *Procedure Partial-Orientation* is given below. It receives as input a graph G and a positive integer parameter t. It computes an orientation with out-degree $\lfloor (2+\epsilon)\cdot a \rfloor$, and deficit at most $\lfloor a/t \rfloor$. Procedure Partial-Orientation is similar to Procedure Complete-Orientation, except step 2, in which an $\lfloor a/t \rfloor$-defective $O(t^2)$-coloring is computed instead of a legal $O(a)$-coloring.

Algorithm 1 Procedure Partial-Orientation(G, t)

1: H_1,H_2,...,$H_{\ell'}$:= an H-partition of G.
2: **for** $i = 1, 2..., \ell'$ in parallel **do**
3: compute an $\lfloor a/t \rfloor$-defective $O(t^2)$-coloring φ of $G(H_i)$.
4: **end for**
5: **for** each edge $e = (u, v)$ in E in parallel **do**
6: **if** u and v belong to different H-sets **then**
7: orient e towards the set with greater index.
8: **else**
9: **if** u and v have different colors **then**
10: orient e towards the vertex with greater color among u, v.
11: **end if**
12: **end if**
13: **end for**

The dominant term in the running time of Procedure Partial-Orientation is the computation of the H-partition that requires $O(\log n)$ time. The other steps are significantly faster, since computing defective colorings in lines 2-4 of the procedure requires $O(\log^* n)$ time, and the orientation step (lines 5-13) requires only $O(1)$ time. Therefore, the running time of Procedure Partial-Orientation is $O(\log n)$. Another important property of Procedure Partial-Orientation is that the length of the produced orientation is bounded. Consider a directed path in a subgraph $G(H_i)$, $1 \leq i \leq \ell'$. The length of this path is smaller than the number of colors used in the defective coloring of $G(H_i)$, which is $O(t^2)$. Now consider a directed path in the graph G with respect to the orientation produced by Procedure Partial-Orientation. The path contains $O(\log n)$ edges that cross between different H-sets. Between any pair of such edges in the path there are $O(t^2)$ consequent edges whose endpoints belong to the same H-set. Hence, the length of a directed path in G is $O(t^2 \log n)$. (See Figure 1 below.)

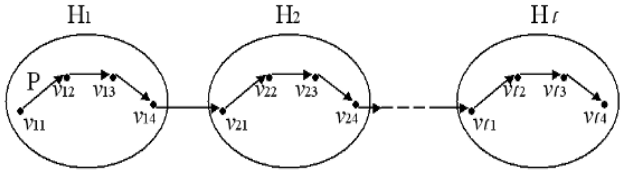

Fig. 1. *A directed path $P = (v_{11}, v_{12}, ..., v_{\ell 4})$ with respect to the orientation produced by Algorithm 1. In this example each H_i is colored with 4 colors. For all $i \in \{1, 2, ..., \ell\}$, $j \in \{1, 2, 3, 4\}$, v_{ij} is colored by j. P contains at most $\ell - 1 = O(\log n)$ edges that cross between H_i's.*

The properties of Procedure Partial-Orientation are summarized in the next theorem.

Theorem 3.5. *Let ϵ be an arbitrarily small positive constant. Procedure Partial-Orientation invoked on a graph G and an integer parameter $t > 0$ produces an acyclic orientation of out-degree $\lfloor (2+\epsilon)\cdot a \rfloor$, length $O(t^2 \cdot \log n)$, and deficit at most $\lfloor a/t \rfloor$. The running time of the procedure is $O(\log n)$.*

We conclude this section with an efficient procedure for computing an arbdefective coloring. The procedure, called *Procedure Arbdefective-Coloring*, receives as input a graph G and two positive integer parameters k and t. First, it invokes Procedure Partial-Orientation on G and t. Then it employs the produced orientation and the parameter k as an input for Procedure Simple-Arbdefective, which is invoked once Procedure Partial-Orientation terminates. This completes the description of the procedure. By Theorem 3.2, the procedure produces an $\lfloor a/t + (2+\epsilon)\cdot a/k \rfloor$-arbdefective k-coloring. The properties of Procedure Arbdefective-Coloring are summarized in the next corollary. The corollary follows directly from Theorems 3.2 and 3.5.

Corollary 3.6. *Procedure Arbdefective-Coloring invoked on a graph G and two positive integer parameters k and t computes an $\lfloor a/t + (2+\epsilon)\cdot a/k \rfloor$-arbdefective k-coloring in time $O(t^2 \log n)$.*

We will invoke Procedure Arbdefective-Coloring with $t = k$. In this case it returns a $\lfloor (3+\epsilon)\cdot a/t \rfloor$-arbdefective t-coloring in $O(t^2 \log n)$ time. Observe that this t-coloring can also be viewed as a decomposition of the original graph G into t subgraphs $G'_1, G'_2, ..., G'_t$, each of arboricity at most $\lfloor (3+\epsilon)\cdot a/t \rfloor$.

4. FAST LEGAL COLORING

In this section we employ the procedures presented in the previous section to devise efficient algorithms that produce legal colorings (i.e., colorings with no defect). Our algorithms rely on the following key properties of arbdefective coloring. Consider a b-arbdefective k-coloring for some positive integer parameters b and k. For $1 \leq i \leq k$, let G_i denote the subgraph induced by all vertices colored with the color i. For all $1 \leq i \leq k$, it holds that $a(G_i) \leq b$. Therefore, by Lemma 2.2, each subgraph G_i can be efficiently legally

colored using $\lfloor(2+\epsilon)\cdot b\rfloor+1$ colors. If each subgraph is assigned a distinct color palette of size $\lfloor(2+\epsilon)\cdot b\rfloor+1$, then the parallel legal coloring of all subgraphs results in a legal $O(b\cdot k)$-coloring of the entire graph G. Observe that once the arbdefective coloring is computed, each vertex $v\in G_i$ communicates only with its neighbors in the same subgraph G_i. Once the legal colorings of all subgraphs are computed, the color of v is different not only from all its neighbors in G_i, but from all its neighbors in G, as we shortly prove.

Our goal is to efficiently compute an $O(a)$-coloring of the graph G. Therefore, we employ Corollary 3.6 with appropriate parameters to guarantee that $b\cdot k=O(a)$. First, we present an $O(a)$-coloring algorithm with running time $O(a^{2/3}\log n)$ that involves a single invocation of Procedure Arbdefective-Coloring. Then, we show a more complex algorithm that achieves running time $O(a^{\mu}\log n)$ for an arbitrarily small positive constant μ.

In our first algorithm we invoke Procedure Arbdefective-Coloring on a graph G with the input parameters $k=t=\left\lceil a^{1/3}\right\rceil$. By Corollary 3.6, as a result of this invocation we achieve a $\left\lfloor(3+\epsilon)\cdot a^{2/3}\right\rfloor$-arbdefective $\left\lceil a^{1/3}\right\rceil$-coloring of G. For $1\leq i\leq k$, let G_i denote the subgraph induced by all vertices colored with the color i. The arboricity of G_i is at most $(3+\epsilon)\cdot a^{2/3}$. For $1\leq i\leq k$, in parallel, color G_i with $\gamma=\left\lfloor(2+\epsilon)(3+\epsilon)\cdot a^{2/3}\right\rfloor+1$ colors. (See Lemma 2.2). Let ψ_i, $1\leq i\leq k$, denote the resulting colorings. For each index i, ψ_i is a legal coloring of G_i. However, for a pair of neighboring vertices $v\in G_i$, $w\in G_j$, $i\neq j$, it may happen that $\psi_i(v)=\psi_j(w)$. Finally, each vertex $v\in G_i$ selects a new color φ that is computed by $\varphi(v)=(i-1)\cdot\gamma+\psi_i(v)$. Intuitively, the color $\varphi(v)$ can be seen as an ordered pair $<i,\psi_i(v)>$. This completes the description of the algorithm. Its correctness and running time are summarized below.

Lemma 4.1. *φ is a legal $O(a)$-coloring of G computed in $O(a^{2/3}\log n)$ time.*

Proof. First, we prove that φ is a legal $O(a)$-coloring. Observe that for each vertex v, it holds that $1\leq\varphi(v)\leq k\cdot\gamma$. Since $k=\left\lceil a^{1/3}\right\rceil$, and $\gamma=\left\lfloor(2+\epsilon)(3+\epsilon)\cdot a^{2/3}\right\rfloor+1$, it follows that $\varphi(v)=O(a)$, and consequently φ is an $O(a)$-coloring. It is left to show that φ is a legal coloring. Consider an edge (u,v) in G, such that $u\in G_i$, $v\in G_j$. If $i=j$ then $\psi(u)\neq\psi(v)$ and hence also $\varphi(u)\neq\varphi(v)$. Otherwise $i\neq j$, and again $\varphi(u)\neq\varphi(v)$.

Next, we prove that φ is computed in $O(a^{2/3}\log n)$ time. By Corollary 3.6, the invocation of Procedure Arbdefective-Coloring requires $O(a^{2/3}\log n)$ time. It produces k subgraphs $G_1,G_2,...,G_k$, each with arboricity at most $\left\lfloor(3+\epsilon)\cdot a^{2/3}\right\rfloor$. By Lemma 2.2, coloring all subgraphs G_i, for $1\leq i\leq k$ in parallel, requires $O(a^{2/3}\log n)$ time as well. The computation of the final coloring φ is performed locally, requiring no additional communication. Therefore, the overall running time is $O(a^{2/3}\log n)$. □

Lemma 4.1 shows that this algorithm is already a significant improvement over the best previously known algorithm for $O(a)$-coloring, whose results are summarized in Lemma 2.2. Nevertheless, the running time can be improved further by invoking Procedure Arbdefective-Coloring several times. Since Procedure Arbdefective-Coloring produces subgraphs of smaller arboricity comparing to the input graph, it can be invoked again on the subgraphs, producing a refined decomposition, in which each subgraph has even smaller arboricity. For example, invoking the procedure on a graph G with the parameters $k=t=\left\lceil a^{1/6}\right\rceil$, results in an $O(a^{5/6})$-arbdefective $O(a^{1/6})$-coloring. Invoking the Procedure Arbdefective-Coloring with the same parameters again on all the $O(a^{1/6})$ subgraphs induced by the initial arbdefective coloring results in an $O(a^{4/6})$-arbdefective $O(a^{1/6})$-coloring of each subgraph. If distinct palettes are used for each subgraph, the entire graph is now colored with an $O(a^{2/3})$-arbdefective $O(a^{1/3})$-coloring. The running time of this computation is $O(a^{1/3}\log n)$. This computation is much faster than a single invocation of Procedure Arbdefective-Coloring with the parameters $k=t=\left\lceil a^{1/3}\right\rceil$ that yields the same results. However, to obtain a legal coloring of the original graph G, each subgraph still has to be colored legally. For the entire computation to be efficient, the arboricity of all subgraphs has to be as small as possible. Therefore we need to invoke Procedure Arbdefective-Coloring more times to achieve an $o(a^{2/3})$-arbdefective coloring. Indeed, applying Procedure Arbdefective-Coloring on each of the $O(a^{1/3})$ subgraphs produces an $O(\sqrt{a})$-arbdefective $O(\sqrt{a})$-coloring. This, in turn, directly gives rise to an $O(a)$-coloring within $O(\sqrt{a}\cdot\log n)$ time.

We employ this idea in the following Procedure called *Procedure Legal-Coloring.*

Algorithm 2 Procedure Legal-Coloring(G,p)

1: $G_1:=G$
2: $\alpha:=a(G_1)$
3: $\mathcal{G}:=\{G_1\}$ /* The set of subgraphs */
4: **while** $\alpha>p$ **do**
5: $\hat{\mathcal{G}}:=\emptyset$ /* Temp. var. for storing refinements of $\mathcal{G}$ */
6: **for** each $G_i\in\mathcal{G}$ in parallel **do**
7: $G'_1,G'_2,...,G'_p:=$ Arbdefective-Coloring(G_i , $k:=p$, $t:=p$) /* G'_j is the subgraph induced by all the vertices that are assigned the color j by the arbdefective coloring */
8: **for** $j:=1,2,...,p$ in parallel **do**
9: $z:=(i-1)\cdot p+j$ /* Computing a unique index for each subgraph */
10: $\hat{G}_z:=G'_j$
11: $\hat{\mathcal{G}}:=\hat{\mathcal{G}}\cup\{\hat{G}_z\}$
12: **end for**
13: **end for**
14: $\mathcal{G}:=\hat{\mathcal{G}}$
15: $\alpha:=\lfloor\alpha/p+(2+\epsilon)\cdot\alpha/p\rfloor$ /* The new upper bound for the arboricity of each of the subgraphs */
16: **end while**
17: $A:=\lfloor(2+\epsilon)\alpha\rfloor+1$
18: **for** each $G_i\in\mathcal{G}$ in parallel **do**
19: color G_i legally using the palette $\{(i-1)\cdot A+1,(i-1)\cdot A+2,...,i\cdot A\}$ /* Using Lemma 2.2 */
20: **end for**

The procedure receives as input a graph G and a positive integer parameter p. It proceeds in phases. In the first phase Procedure Arbdefective-Coloring is invoked on the input graph G with the parameters $k:=p$ and $t:=p$. Con-

sequently, a decomposition into p subgraphs is produced, in which each subgraph has arboricity $O(a/p)$. In each of the following phases Procedure Arbdefective-Coloring is invoked in parallel on all subgraphs in the decomposition that was created in the previous phase. As a result, a refinement of the decomposition is produced, i.e., each subgraph is partitioned into p subgraphs of smaller arboricity. Consequently, after each phase, the number of subgraphs in G grows by a factor of p, but the arboricity of each subgraph decreases by a factor of $\Theta(p)$. Hence, the product of the number of subgraphs and the arboricity of subgraphs remains $O(a)$ after each phase. (As long as the number of phases is constant.) Once the arboricities of all subgraphs become small enough, Lemma 2.2 is used for a fast parallel coloring of all the subgraphs, resulting in a unified legal $O(a)$-coloring of the input graph.

Let μ be an arbitrarily small positive constant. We show that invoking Procedure Legal-Coloring on G with the input parameter $p := \left\lfloor a^{\mu/2} \right\rfloor$ results in an $O(a)$-coloring in $O(a^{\mu} \log n)$ time. The following lemma constitutes the proof of correctness of the algorithm.

We assume without loss of generality that the arboricity a is sufficiently large to guarantee that $p \geq 16$. (Otherwise, it holds that $a \leq 17^{2/\mu}$, i.e., the arboricity is bounded by a constant. In this case, by Lemma 2.2, one can directly compute an $O(1)$-coloring in $O(\log n)$ time).

Let α_i and $\mathcal{G}_i$ denote the values of the variables α and $\mathcal{G}$, respectively, in the end of iteration i of the while-loop of Algorithm 2 (lines 4-16).

Lemma 4.2. (1) *(Invariant for line 16 of Algorithm 2) In the end of iteration i of the while-loop, $i = 1, 2, ...$, each graph in the collection $\mathcal{G}_i$ has arboricity at most α_i.*
(2) *The while-loop runs for a constant number of iterations.*
(3) *For i =1,2,..., after i iterations, it holds that $\alpha_i \cdot |\mathcal{G}_i| \leq (3+\epsilon)^i \cdot a$.*

PROOF. **The proof of (1):** The proof is by induction on the number of iterations. For the base case, observe that after the first iteration, $\mathcal{G}$ contains p subgraphs produced by Procedure Arbdefective-Coloring. By Corollary 3.6, the arboricity of each subgraph is at most $\lfloor a/t + (2+\epsilon) \cdot a/k \rfloor = \lfloor a/p + (2+\epsilon) \cdot a/p \rfloor = \alpha_1$.

For the inductive step, consider an iteration i. By the induction hypothesis, each subgraph in $\mathcal{G}_{i-1}$ has arboricity at most α_{i-1}. During iteration i, Procedure Arbdefective-Coloring is invoked on all subgraphs in $\mathcal{G}_{i-1}$. Consequently, $\mathcal{G}_i$ contains new subgraphs, each with arboricity at most $\lfloor \alpha_{i-1}/p + (2+\epsilon) \cdot \alpha_{i-1}/p \rfloor$, which is exactly the value α_i of α in the end of iteration i. (See line 15.)

The proof of (2): In each iteration the variable α is decreased by a factor of at least $b = p/(3+\epsilon)$. Hence, the number of iterations is at most $\log_b a$. For any $0 < \epsilon < 1/2$, and a sufficiently large a, it holds that

$$\log_b a = \frac{\log a}{\log(p/(3+\epsilon))} \leq \frac{\log a}{\log(\frac{1}{4}a^{\mu/2})} = \frac{2/\mu \cdot \log a^{\mu/2}}{\log a^{\mu/2} - 2} \leq 4/\mu.$$

The proof of (3): The correctness of the lemma follows directly from the fact that in each iteration the number $|\mathcal{G}|$ of subgraphs grows by a factor of p, and the arboricity of each subgraph decreases by a factor of at least $p/(3+\epsilon)$. □

The next theorem follows from Lemma 4.2.

Theorem 4.3. *Invoking Procedure Legal-Coloring on a graph G with arboricity a with the parameter $p = \left\lceil a^{\mu/2} \right\rceil$ for a positive constant $\mu < 1$, computes a legal $O(a)$-coloring of G within $O(a^{\mu} \cdot \log n)$ time.*

PROOF. We first prove that the coloring is legal. Observe that the selection of unique indices in line 9 guarantees that any two distinct subgraphs that were added to the same set $\hat{\mathcal{G}}$ are colored using distinct palettes. In addition, in each iteration each vertex belongs to exactly one subgraph in $\mathcal{G}$. Consequently, once the while-loop terminates, each vertex v belongs to exactly one subgraph in $\mathcal{G}$. Let $G_i \in \mathcal{G}$ be the subgraph that contains v. Let α' denote the value of α on line 17 of Algorithm 2. As we have seen, the arboricity of G_i is at most α'. Hence, G_i is colored legally using a unique palette containing $A = \lfloor (2+\epsilon)\alpha' + 1 \rfloor$ colors. Consequently, the color of v is different from the colors of all its neighbors, not only in G_i, but in the entire graph G.

Now we analyze the number of colors in the coloring. By Lemma 4.2, the number of colors employed is $A \cdot |\mathcal{G}| = (\lfloor (2+\epsilon)\alpha' \rfloor + 1) \cdot |\mathcal{G}| \leq (3+\epsilon)^c \cdot a$, for some explicit constant c. (For a sufficiently large a, the appropriate constant is $c = 4/\mu + 1$.) Hence, the number of employed colors is $O(a)$.

Next, we analyze the running time of Procedure Legal-Coloring. By Lemma 4.2(2), during the execution of Procedure Legal-Coloring, the Procedure Arbdefective-Coloring is invoked for a constant number of times. Note also that each time it is invoked with the same values of the parameters $t = k = p = \left\lfloor a^{\mu/2} \right\rfloor$. Hence, by Corollary 3.6, executing the while-loop requires $O(t^2 \log n) = O(a^{\mu} \log n)$ time. By Lemma 2.2, the additional time required for coloring all the subgraphs in step 19 of Algorithm 2 is $O(p \log n) = O(a^{\mu/2} \log n)$. (By the termination condition of the while-loop (line 4), once the algorithm reaches line 19, it holds that $\alpha \leq p$.) Therefore, the total running time is $O(a^{\mu} \log n)$. □

Theorem 4.3 implies that for the family of graphs with polylogarithmic arboricity in n, an $O(a)$-coloring can be computed in time $O((\log n)^{1+\mu'})$, for an arbitrarily small positive constant μ'. In the case of graphs with superlogarithmic arboricity, we can achieve even better results than those that are given in Theorem 4.3. In this case we execute Procedure Legal-Coloring with the parameter $p = \left\lfloor \frac{a^{\mu'}}{\log n} \right\rfloor$. Since a is superlogarithmic in n, and $\mu' > 0$ is a constant, it holds that $p > a^{\mu'/2}$, for a sufficiently large n. Therefore, Procedure Legal-Coloring executes its loop a constant number of times. Consequently, the number of colors employed is still $O(a)$. The running time is the sum of running time of Procedure Arbdefective-Color and the running time of computing legal colorings of graphs of arboricity at most p, which is $O(\frac{a^{2\mu'}}{\log^2 n} \cdot \log n + \frac{a^{\mu'}}{\log n} \cdot \log n) = O(a^{2\mu'})$. If we set $\mu' = \mu/2$, the running time becomes $O(a^{\mu})$. We summarize this result in the following corollary.

Corollary 4.4. *Let μ be an arbitrarily small constant. For any graph G, a legal $O(a)$-coloring of G can be computed in time $O(a^{\mu} + (\log n)^{1+\mu})$.*

Next, we demonstrate that one can trade the number of colors for time. Specifically, we show that if one is allowed to use slightly more than $O(a)$ colors, the running time can be

bounded by *polylog*(n), *for all values of* a. To this end, we select the parameter p to be polylogarithmic in a. With this value of p the running time $O(p \log n)$ of the coloring step in line 19 of Algorithm 2 becomes polylogarithmic. Moreover, setting the parameters t and k to be polylogarithmic in a results in a polylogarithmic running time of Procedure Arbdefective-Coloring. The number of executions of an iteration of the while-loop is $O(\log_p a)$. Consequently, the total running time is also polylogarithmic. However, the number of iterations becomes superconstant. Hence the number of colors grows beyond $O(a)$. The specific parameters we select are $p = k = t = f(a)^{1/2}$, for an arbitrarily slow-growing function $f(a) = \omega(1)$. The results of invoking Procedure Legal-Coloring with these parameters are given below.

Theorem 4.5. *Invoking Procedure Legal-Coloring with the parameter* $p = f(a)^{1/2}$, $f(a) = \omega(1)$ *as above, requires* $O(f(a) \log a \log n)$ *time. The resulting coloring employs* $a^{1+o(1)}$ *colors.*

Proof. Set $b = p/(3+\epsilon)$. The number of iterations is at most $\log_b a = O(\frac{\log a}{\log f(a)})$. Each iteration requires $O(p^2 \log n) = O(f(a) \log n)$ time. Hence the running time is $\log_b a \cdot O(f(a) \log n) = O(f(a) \log a \log n)$. By Lemma 4.2(3), the total number of employed colors is at most $a \cdot (3+\epsilon)^{O(\log a / \log f(a))} = a^{1+O(1/\log f(a))} = a^{1+o(1)}$. □

More generally, as evident from the above analysis, the running time of our algorithm is $O(p^2 \log_p a \log n)$, and the number of colors used is $2^{O(\log_p a)} \cdot a$. Another noticeable point on the tradeoff curve is on the opposite end of the spectrum, i.e., $p = C$, for some large constant C. Here the tradeoff gives rise to $a^{1+O(1/\log C)}$-coloring in $O(\log a \log n)$ time.

Corollary 4.6. *For an arbitrarily small constant* $\eta > 0$, *Procedure Legal-Coloring invoked with* $p = 2^{O(1/\eta)}$ *produces an* $O(a^{1+\eta})$*-coloring in* $O(\log a \log n)$ *time.*

Corollary 4.6 implies that any graph G for which there exists a constant $\nu > 0$ such that $a \leq \Delta^{1-\nu}$ can be colored with $o(\Delta)$ colors in $O(\log a \log n)$ time. This goal is achieved by computing an $O(a^{1+\nu})$-coloring of the input graph G. Since $a^{1+\nu} \leq \Delta^{1-\nu^2}$, this is an $o(\Delta)$-coloring of G. Therefore, our results give rise to deterministic polylogarithmic $(\Delta+1)$-coloring algorithm for a very wide family of graphs. This fact is summarized in the following corollary.

Corollary 4.7. *For the family of graphs with arboricity* $a \leq \Delta^{1-\nu}$, *for an arbitrarily small constant* ν, *one can compute* $(\Delta+1)$*-coloring in* $O(\log a \log n)$ *time.*

5. EVEN FASTER COLORING

In this section we show that one can decrease the running time of the coloring procedure almost all the way to $\log n$, at the expense of increasing the number of colors. (The number of colors still stays $o(a^2)$, but it grows significantly beyond $a^{1+\eta}$.) In addition, we show that for any t, $1 \leq t \leq a$, and any constant $\epsilon > 0$, one can compute $O(a \cdot t)$-coloring in $O((\frac{a}{t})^{\epsilon} \cdot \log n)$ time.

We start with extending an algorithm from [14] to graphs of bounded arboricity. Specifically, Kuhn [14] devised an algorithm that works on an n-vertex graph G of maximum degree Δ, and for an integer parameter t, $1 \leq t \leq \Delta$, it constructs an $O(t^2)$-coloring, (Δ/t)-defective in $O(\log^* n)$ time. (His technique is based on that of Linial [17].) We show that if a graph G has arboricity at most a then an (a/t)-arbdefective $O(t^2)$-coloring can be computed in $O(\log n)$ time.

The first step of our algorithm is to construct an orientation σ of out-degree at most A, $A = (2+\epsilon) \cdot a$, for some constant $\epsilon > 0$. To this end we employ an algorithm from [3]. This algorithm requires $O(\log n)$ time, and it is the most time-consuming step of the algorithm. The second step uses this orientation to execute an algorithm that is analogous to the one of [14].

Next, we describe this algorithm. Set $d = \Delta/t$. Suppose that we start with a d'-arbdefective M-coloring of G, for some possibly very large M, and $0 \leq d' \leq d$. Consider a pair of sets $\mathcal{A}$ and $\mathcal{B}$ that will be determined later, and let $F(\mathcal{A}, \mathcal{B})$ denote the collection of all functions from $\mathcal{A}$ to $\mathcal{B}$. Consider also a mapping $\Psi : [M] \to F(\mathcal{A}, \mathcal{B})$ that associates a function $\varphi_\chi \in F(\mathcal{A}, \mathcal{B})$ (i.e, $\varphi_\chi : \mathcal{A} \to \mathcal{B}$) with each color $\chi \in [M]$. Our coloring algorithm is based on a recoloring subroutine, Procedure Arb-Recolor. This procedure is described in Algorithm 3. The procedure accepts as input the original color $\chi \in [M]$ of the vertex v that executes the procedure, the $\delta \leq A$ colors of the parents of v according to the orientation computed on the first step of the algorithm, and the defect parameter d.

Algorithm 3 Procedure Arb-Recolor

Input: A color $\chi \in [M]$, parent colors $y_1, y_2, ..., y_\delta \in [M]$, parameter d.

1: find $\alpha \in \mathcal{A}$ such that
$$|\{i \in [\delta] : \varphi_\chi(\alpha) = \varphi_{y_i}(\alpha)\}| \leq d. \quad (*)$$
2: return (color := $(\alpha, \varphi_\chi(\alpha))$).

The following lemma summarizes the properties of Procedure Arb-Recolor. (The lemma and its proof are analogous to Lemma 4.1 in [14]).

Lemma 5.1. *For a value* $k > 0$, *suppose that the functions* $\{\varphi_x : x \in [M]\}$ *satisfy that for any two distinct colors* $x, y \in [M]$, *there are at most* k *values* $\alpha \in \mathcal{A}$ *for which* $\varphi_x(\alpha) = \varphi_y(\alpha)$. *Suppose also that* $|\mathcal{A}| > k \cdot \frac{A-d'}{d-d'+1}$. *Then procedure Arb-Recolor computes a* d*-arbdefective* $(|\mathcal{A}| \cdot |\mathcal{B}|)$*-coloring* χ'.

By Lemma 4.3 [14], the collection of functions $\{\varphi_x : x \in [M]\}$ with the property required by the statement of Lemma 5.1 exists if $|\mathcal{B}| \geq \frac{\mathcal{A}}{2 \cdot \ln M}$ and $k = \lfloor 2e \cdot \ln M \rfloor$. For Lemma 5.1 to hold, $|\mathcal{A}|$ should be greater than $k \cdot \frac{A-d'}{d-d'+1}$. Hence the number of colors used by χ' is $|\mathcal{A}| \cdot |\mathcal{B}| = k^2 \frac{(A-d')^2}{(d-d'+1)2 \ln M} = O(\log M) \cdot \frac{(A-d')^2}{d-d'+1}$.

By using $O(\log^* M)$ iterations of Procedure Arb-Recolor with intermediate values of defect parameter that are specified in the proof of Theorem 4.9 of [14] we obtain an $O(A^2/d^2)$-coloring with arbdefect at most d. (The proof for this statement is identical to the proof of Theorem 4.9 of [14].) Henceforth, we refer to this algorithm that invokes Procedure Arb-Recolor $O(\log^* M)$ times by *Algorithm Arb-Kuhn*. Since each invocation of procedure Arb-Recolor requires $O(1)$ time, the overall running time of Algorithm Arb-Kuhn is $O(\log^* M) = O(\log^* n)$. (For $M = n$ we start with a trivial legal n-coloring that uses each vertex Id as its color.) As $d = A/t$, we obtain an A/t-arbdefective $O(t^2)$-coloring within this running time.

Next, we argue that using Algorithm Arb-Kuhn in conjunction with our algorithm enables one to trade between the running time and number of colors. Specifically, set $d = f(a)$ to be some growing function of the arboricity a, i.e., $f(a) = \omega(1)$. Invoke Algorithm Arb-Kuhn. We obtain a decomposition of the original graph G into $O(A^2/f(a)^2) = O(a^2/f(a)^2)$ subgraphs of arboricity at most $\alpha = f(a)$ each. Invoke on each of these subgraphs in parallel our algorithm that for n-vertex graphs with arboricity α computes an $O(\alpha^{1+\eta})$-coloring in $O(\log \alpha \log n)$ time, for an arbitrarily small constant $\eta > 0$. Use distinct palettes of size $O(\alpha^{1+\eta})$ for each of the $O(a^2/\alpha^2)$ subgraphs to get a unified $O(a^2/\alpha^{1-\eta})$-coloring of the entire graph G. The running time of this algorithm is $O(\log^* n + \log \alpha \log n) = O(\log f(a) \cdot \log n)$. We set $g(a) = f(a)^{1-\eta}$ and obtain the next Theorem.

Theorem 5.2. *For an arbitrarily small constant $\eta > 0$, and any function $\omega(1) = g(a) = O(a^{1-\eta})$, our algorithm computes an $O(a^2/g(a))$-coloring, in time $O(\log g(a) \cdot \log n)$.*

In particular, by setting $g(a) = 2^{\log^\zeta a}$ for some $\zeta > 0$, one can have here an $(a^2/2^{\Omega(\log^\zeta a)})$-coloring within $O(\log^\zeta a \log n)$ time. Also, with $g(a) = \log^c a$, for an arbitrarily large constant $c > 0$, one gets an $O(a^2/\log^c a)$-coloring in time $O(\log \log a \log n)$.

Finally, we show that this technique can be used to obtain a tradeoff between the running time and the number of colors. This new tradeoff improves the previous tradeoff (due to [3]) for *all* values of the parameters. Specifically, we have shown that $O(a/t)$-arbdefective $O(t^2)$-coloring can be computed in $O(\log n)$ time. In other words, a graph G of arboricity a can be decomposed into $O(t^2)$ subgraphs of arboricity $\alpha = O(a/t)$ each, in $O(\log n)$ time. By Corollary 4.3, by invoking Procedure Legal-Coloring in parallel on all these subgraphs we obtain an $O(\alpha)$-coloring of each of them. The running time of this step is $O((\frac{a}{t})^\mu \cdot \log n)$, for an arbitrarily small constant $\mu > 0$. Using disjoint palettes for each of the subgraphs we merge these colorings into a unified $O(\alpha \cdot t^2) = O(a \cdot t)$-coloring of the original graph G. The last step (the merging) requires no communication. Hence, the total running time of the algorithm is $O((\frac{a}{t})^\mu \cdot \log n)$.

Theorem 5.3. *For any parameter t, $1 \leq t \leq a$, and a constant $\mu > 0$, an $O(a/t)$-coloring of a graph of arboricity a can be computed in $O((\frac{a}{t})^\mu \cdot \log n)$ time.*

6. REFERENCES

[1] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[2] 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.

[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. Distributed $(\Delta + 1)$-coloring in linear (in Δ) time. *http://arXiv.org/abs/0812.1379v2*, 2008.

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

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

[8] D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4):207–211, 1994.

[9] A. Goldberg, and S. Plotkin. Efficient parallel algorithms for $(\Delta + 1)$- coloring and maximal independent set problem. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 315–324, 1987.

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

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

[12] Ö. Johansson. Simple distributed $(\Delta + 1)$-coloring of graphs. *Information Processing Letters*, 70(5):229–232, 1999.

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

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

[15] 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.

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

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

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

[19] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036-1053, 1986.

[20] C. Nash-Williams. Decompositions of finite graphs into forests. *J. London Math*, 39:12, 1964.

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

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