Sublogarithmic Distributed MIS Algorithm for Sparse Graphs using Nash-Williams Decomposition

[Extended Abstract]

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 distributed *maximal independent set* (henceforth, MIS) problem on sparse graphs. Currently, there are known algorithms with a sublogarithmic running time for this problem on oriented trees and graphs of bounded degrees. We devise the first *sublogarithmic* algorithm for computing MIS on graphs of bounded arboricity. This is a large family of graphs that includes graphs of bounded degree, planar graphs, graphs of bounded genus, graphs of bounded treewidth, graphs that exclude a fixed minor, and many other graphs. We also devise efficient algorithms for coloring graphs from these families.

These results are achieved by the following technique that may be of independent interest. Our algorithm starts with computing a certain graph-theoretic structure, called *Nash-Williams forests-decomposition*. Then this structure is used to compute the MIS or coloring. Our results demonstrate that this methodology is very powerful.

Finally, we show nearly-tight *lower bounds* on the running time of any distributed algorithm for computing a forests-decomposition.

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

*This research has been supported by the Israeli cademy of Science, grant 483/06. Additional funding was provided by the Lynn and William Frankel Center for Computer Sciences.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.

Keywords

MIS, Coloring, Arboricity, Forests Decomposition

1. INTRODUCTION

1.1 MIS

We consider the synchronous message-passing model of distributed computing. The network is modeled by an unweighted undirected n-vertex graph $G = (V, E)$, with every vertex hosting a processor. The processors communicate over the edges of G. In each communication round each vertex $v \in V$ can send a short message to each of its neighbors, and these messages arrive before the next round starts. The processors have distinct identity numbers (henceforth, IDs) represented by bit strings of length $O(\log n)$. For an algorithm $\mathcal{A}$ in this model, the running time of $\mathcal{A}$ is the (worst-case) number of rounds of distributed communication that may occur during an execution of $\mathcal{A}$.

We focus on the *maximal independent set* (henceforth, MIS) and coloring problems. A subset $I \subseteq V$ of vertices is called an MIS of G if (1) for every pair $u, w \in U$ of neighbors, either u or w do not belong to I, and (2) for every vertex $v \in V$, either $v \in I$ or there exists a neighbor $w \in V$ of v that belongs to I.

The problem of computing MIS is one of the most fundamental problems in the area of Distributed Algorithms. More than twenty years ago Luby [18] and Alon, Babai, and Itai [1] devised two logarithmic time randomized algorithms for this problem on *general* graphs. These algorithms remain the state-of-the-art to this date. Awerbuch, Goldberg, Luby, and Plotkin [3] devised the first deterministic algorithm for this problem on general graphs, which was later improved by Panconesi and Srinivasan [20] in 1992. The latter algorithms is the state-of-the-art. Its running time is $2^{O(\sqrt{\log n})}$. The best-known lower bound for the MIS problem on general graphs, $\Omega(\sqrt{\frac{\log n}{\log \log n}})$, is due to Kuhn, Moscibroda, and Wattenhofer [14].

Cole and Vishkin [5] presented an algorithm for computing MIS on rings and oriented trees. The running time of the algorithm of [5] is $O(\log^* n)$. Linial [17] has shown that this result is tight up to constant factors. In 1988 Goldberg, Plotkin, and Shannon [11] initiated the study of the MIS problem on *sparse* graphs. They devised a deterministic algorithm for the MIS problem on *planar* graphs that requires $O(\log n)$ time. Their algorithm extends also to graphs of *bounded genus*.

In this paper we improve and generalize the result of Goldberg et al. [11] and devise a deterministic algorithm for the MIS problem on graphs of *bounded arboricity* that requires time $O(\frac{\log n}{\log\log n})$. The arboricity of a graph is a measure for its sparsity. Sparse graphs have low arboricity. The family of graphs of bounded arboricity includes not only planar graphs, graphs of bounded genus, and graphs of bounded degree, but also graphs that *exclude any fixed minor* and graphs of *bounded treewidth*. Moreover, a graph with constant arboricity may have genus $O(n)$, and may contain $K_{\sqrt{n}}$ as a minor. Consequently, the family of graphs on which our algorithm constructs MIS in sublogarithmic time is much wider than each of the families that we have listed above. Moreover, our result applies also when the arboricity $a = a(G)$ of the input graph G is super-constant (up to $a = \log^{1/2-\epsilon} n$, for any $\epsilon > 0$).

To our knowledge, prior to our work the only graph families on which there existed a sublogarithmic time algorithm for the MIS problem were the family of graphs with bounded degree [11, 16, 17] and the family of graphs with bounded growth [9]. In other words, our algorithm is the *first sublogarithmic time* (deterministic or randomized) algorithm for the MIS problem on *any graph family* other than these two families of graphs. Even for the family of unoriented trees, which is contained in the family of graphs of constant arboricity, the best previous result has running time of $O(\log n)$.

In addition, we show that an MIS on graphs of arboricity at most a can be computed deterministically in $O(a\sqrt{\log n} + a\log a)$ time. In particular, this result implies that an MIS can be computed deterministically in polylogarithmic time on graphs G with arboricity at most polylogarithmic in n. Hence we significantly extend the class of graphs on which an efficient (that is requiring a polylogarithmic time) deterministic algorithm for computing MIS is known.

1.2 Coloring

We also study the *coloring* problem. This problem is closely related to the MIS problem, and similarly to the latter problem, the coloring problem is one of the most central and most intensively studied problems in Distributed Algorithms [10, 11, 17, 16, 24]. The goal of the coloring problem is to assign colors to vertices so that for each edge e, the endpoints of e are assigned distinct colors. There is an inherent tradeoff between the running time of a distributed coloring algorithm and the number of colors it employs for coloring the underlying network.

There are efficient algorithms for coloring graphs of bounded degree. Specifically, for a positive integer parameter Δ, Goldberg, Plotkin, and Shannon [11] devised a $(\Delta+1)$-coloring algorithm with running time $O(\Delta\log n)$. Goldberg and Plotkin [10] devised an $O(\Delta^2)$-coloring algorithm with running time $O(\log^* n)$, for constant values of Δ, and Linial [17] extended this result to general values of Δ. Recently, Kuhn and Wattenhofer [16] presented a $(\Delta+1)$-coloring algorithm with running time $O(\Delta\log\Delta + \log^* n)$. For planar graphs, Goldberg et al. [11] devised a 7-coloring algorithm with running time $O(\log n)$, and a 5-coloring algorithm with running time $O(\log n\log\log n)$. (The latter algorithm assumes that a planar embedding of the input graph is known to the vertices.)

We significally extend the class of graphs families for which efficient coloring algorithms are known, and devise a $(\lfloor(2+\epsilon)\cdot a\rfloor+1)$-coloring algorithm for graphs G of bounded arboricity $a(G) \leq a$ that has running time $O(a\cdot\log n)$. (The parameter $\epsilon > 0$ can be set as an arbitrarily small positive constant.) In particular, our algorithm 7-colors any graph of arboricity at most 3 in logarithmic time, subsuming the result of Goldberg et al. [11]. Moreover, it provides an $O(1)$-coloring of any graph of constant arboricity in logarithmic time. As was discussed above, this family of graphs contains graphs of bounded degree, graphs of bounded genus, graphs that exclude any fixed minor, and many other graphs.

We also present two tradeoffs between the running time of our algorithm and the number of colors it employs. For a positive parameter $q \geq 1$, and an input graph G of arboricity $a = a(G)$, our first algorithm computes an $O(q\cdot a^2)$-coloring of the input graph in time $O(\frac{\log n}{\log q} + \log^* n)$. In addition, for a positive parameter t, $1 \leq t \leq a$, our second algorithm computes an $O(t\cdot a)$-coloring in time $O(\frac{a}{t}\cdot\log n + a\cdot\log a)$. Finally, we show that for any a and q, any algorithm for $O(q\cdot a^2)$-coloring graphs requires $\Omega(\frac{\log n}{\log a+\log q} + \log^* n)$ time, and thus our first tradeoff is nearly optimal.

1.3 Forests-Decomposition

It is well-known [19] that the edge set E of any graph $G = (V, E)$ of arboricity $a = a(G)$ can be decomposed into a edge-disjoint forests. We refer to this decomposition as the *forests-decomposition*. This fundamental theorem has many applications in graph theory and combinatorics (see [4], and the references therein). However, so far there was no efficient distributed algorithm known for computing such a decomposition. A key ingredient in most of our algorithms for the MIS and coloring problems is an efficient subroutine for computing forests-decompositions. We demonstrate that for a parameter q, $q \geq 1$, a forests-decomposition into $O(q\cdot a)$ forests of a graph with arboricity a can be computed (distributedly) in time $O(\frac{\log n}{\log q})$. We also show a lower bound of $\Omega(\frac{\log n}{\log q+\log a} - \log^* n)$ for this problem, demonstrating that our algorithm is near-optimal. Remarkably, all our algorithms in this paper can be applied even when vertices do not know the arboricity of the underlying graph.

It is plausible that our algorithm for computing forests-decompositions will be useful for other applications. Hence we believe that this result is of independent interest.

1.4 Related Work

Recently, MIS and coloring problems were studied on unit disk, unit ball, and more generally, bounded growth graphs [15, 13, 9, 22]. Specifically, Kuhn et al. [15] devised a deterministic algorithm with running time $O(\log^* n)$ for these problems on unit ball graphs whose underlying metric is doubling. This result was extended in [13] to a more general family of bounded growth graphs at the expense of increasing the running time to $O(\log\Delta\cdot\log^* n)$. Gfeller and Vicari devised a randomized algorithm for computing MIS in $O(\log\log n\cdot\log^* n)$ time on bounded growth graphs [9]. Finally, Schneider and Wattenhofer improved this result and devised a deterministic algorithm with running time $O(\log^* n)$ for the MIS problem on graphs of bounded growth [22]. We remark that the family of graphs of bounded growth and of bounded arboricity are incomparable, i.e., there are graphs of bounded growth that have large arboricity and vice versa.

Our algorithm for computing forests-decomposition is closely related to one of the coloring algorithms from Goldberg et al. [11]. However, the latter algorithm does not

explicitly compute a forests-decomposition. An algorithm that does compute a forests-decomposition explicitly in the PRAM model of parallel computing was devised by Arikati et al. [2]. This algorithm computes an *unoriented* forests-decomposition, i.e., the computed trees are unrooted and, consequently, there is no child-parent relationship between neighboring vertices. Once the decomposition has been computed, each forest is oriented separately. However, this technique does not guarantee an *acyclic* orientation which is required for our coloring algorithms. Moreover, the algorithm of [2] starts with computing a constant approximation on the graph arboricity. While this can be accomplished efficiently in the PRAM model, it is not hard to see that in the distributed model computing such an estimate requires $\Omega(n)$ time. Consequently, the technique of Arikati et al. is inapplicable in the distributed setting.

Finally, a number of recent papers considered the effect of "sense of direction" or "orientation" on distributed computation [23, 12]. In particular, Kothapalli et al. [12] showed that if the graph G is provided with an orientation that satisfies a certain helpful property then an $O(\Delta)$-coloring of G can be constructed in $O(\log\Delta + \sqrt{\log n \log\log n})$ time with high probability. However, unfortunately, it is currently not known whether such an orientation can be constructed as efficiently from scratch.

1.5 The structure of the paper

In Section 3 we present our algorithm for computing a forests-decomposition. In Section 4 we present an $O(a)$-coloring algorithm with running time $O(a\cdot\log n)$. In Section 5 we demonstrate that if the algorithm is allowed to use more than $O(a)$ colors, then coloring can be accomplished much faster. In Section 6 we utilize the results of Section 5 to devise our sublogarithmic algorithm for the MIS problem, and present our lower bounds.

Due to space limitations, in this extended abstract starting from Section 4 we assume that the value $a = a(G)$ of the arboricity of the input graph is known to the vertices before the computation starts. However, all our algorithms can be extended to the scenario when the arboricity is unknown to the vertices using techniques similar to those presented in Section 3.2. The time complexity of the extended algorithms grows only by constant factors.

Also, some proofs are omited from this extended abstract.

2. PRELIMINARIES

2.1 Definitions and Notation

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 a positive integer n, $\log^{(0)} n = n$, and $\log^{(i+1)} n = \log(\log^{(i)} n)$, for every $i = 0, 1, 2, ...$ Also, for a positive integer n, $\log^* n$ is defined by: $\log^* n = \min\left\{i \mid \log^{(i)} n \leq 2\right\}$.

The *degree* of a vertex v in a graph $G = (V, E)$, denoted *deg(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. For a subset $U \subseteq V$, the degree of v with respect to U, denoted *deg(v,U)*, is the number of neighbors of v in U. The maximum degree of a vertex in G, denoted $\Delta(G)$, is defined by $\Delta(G) = \max_{v\in V} deg(v)$. 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 *out-degree* of a vertex v in a directed graph G is the number of edges connected to v that are oriented outwards of v.

A *directed cycle* in a directed graph G is a cycle whose edges are oriented consistently, e.g., each vertex in the cycle is connected to one outgoing edge and one incoming edge of the cycle.

An *orientation* of (the edges of) an undirected graph G is an assignment μ of direction to each edge of G. An orientation is *acyclic* if the resulting directed graph $\hat{G}$ contains no directed cycles. An *out-degree* of a vertex v in G *with respect to an orientation* μ, or shortly, *μ-out-degree*, is the out-degree of v in $\hat{G}$. An outgoing edge of v in $\hat{G}$ is called an *outgoing edge with respect to* μ, or shortly, a *μ-outgoing edge*.

The *arboricity* of a graph $G = (V, E)$ is defined by: $a(G) = \max\left\{\left\lceil \frac{|E(G')|}{|V(G')|-1}\right\rceil \middle| G' \subseteq G, |V(G')| \geq 2\right\}$.

If the graph G can be understood from the context, we use the notation Δ (respectively, a) as a shortcut for $\Delta(G)$ (resp., $a(G)$).

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*

Some of our algorithms use as a black-box an algorithm due to Kuhn and Wattenhofer [16] that $(\Delta + 1)$-colors any graph G with maximum degree Δ in time $O(\Delta \cdot \log\Delta + \log^* n)$. We will refer to this algorithm as *KW Coloring Algorithm*.

2.2 Graph Parameters and Classes

It follows from the definition of arboricity that $a(G) \leq \Delta(G)$, and thus, the family of graphs with bounded arboricity contains the family of graphs with bounded degree.

For a graph G of bounded genus g, by Euler formula, $|E| \leq 3|V| - 6 + 6\cdot g$, and thus, $a(G) = O(1 + g/|V|)$. Hence the family of graphs of genus $g = O(n)$ is contained in the family of graphs of bounded arboricity.

Another important graph family that is contained in the family of graphs with bounded arboricity is the family of graphs that exclude a fixed minor. Given an edge $e = (x, y)$ of a graph G, the graph G/e is obtained from G by contracting the edge e, that is, by identifying the vertices x and y, and removing all self-loops. In addition, for each pair of nodes u and w for which the resulting graph has now more than one edge, we replace all these edges with a single edge (u, w). A graph H' that is obtained from G by a sequence of edge contractions is called a *contraction* of G, and a subgraph H of a contraction of G is called a *minor* of G. For a fixed graph H, a graph family $\mathcal{G}$ is said to *exclude a minor* H if for every graph $G \in \mathcal{G}$, H is not a minor of G.

It is well-known (see, e.g., [6] Theorem 7) that for any fixed graph H there exists a number a_H such that every graph G that excludes minor H has arboricity at most a_H. Consequently, the family of graphs that exclude a fixed minor is contained in the family of graphs with bounded arboricity.

The same is true for graphs with bounded *treewidth*. For a positive integer parameter k, we say that a vertex v is a *k-simplicial* vertex if the set of its neighbors forms a clique of size k, K_k. A *k-tree* is a graph G that is either isomorphic to K_k, or G has a k-simplicial vertex v and the graph $G \setminus v$ obtained by removing v from G is a k-tree. A *treewidth* of a graph G is the minimum k such that G is a spanning

subgraph of a k-tree. It is well-known that a graph with treewidth k has arboricity at most k. (See, e.g., [7] Theorem 2). Consequently, the family of graphs of bounded treewidth is contained in the famiy of graphs of bounded arboricity.

3. FORESTS-DECOMPOSITION

In this section we present a distributed algorithm that computes a forests-decomposition with $(2+\epsilon)\cdot a$ forests, for an arbitrarily small constant parameter $\epsilon > 0$. This algorithm is a basic building block in most of our coloring algorithms. Some of them invoke this algorithm directly. Other algorithms do not employ it as a black-box, but rather use the partition of the vertex set V produced by this algorithm.

In Section 3.1 we present a simpler variant of our algorithm for computing a forests decomposition that applies to the scenario in which both the number of vertices n and the arboricity $a = a(G)$ of the input graph are known to all vertices before the computation starts. In Section 3.2 we extend the algorithm to scenarios in which one of those parameters is not known in the beginning of the computation.

3.1 Known arboricity

On the first step, our algorithm for computing a forests-decomposition, henceforth Procedure *Forests-Decomposition*, invokes the partitioning subroutine called Procedure *Partition*. Both Procedure Forests-Decomposition and Procedure Partition accept as input two parameters. The first parameter is the arboricity of the input graph, and the second parameter ϵ is a positive real number. Procedure *Partition* partitions the vertex set of the graph into $\ell = \lfloor \frac{2}{\epsilon}\log n \rfloor$ disjoint subsets $H_1, H_2, ..., H_\ell$ that satisfy that every vertex $v \in H_i$, $i \in \{1, 2, ..., \ell\}$, has at most $(2+\epsilon)\cdot a$ neighbors in the vertex set $\bigcup_{j=i}^{\ell} H_j$, i.e., $deg(v, \bigcup_{j=i}^{\ell} H_j) \leq (2+\epsilon)\cdot a$. We will henceforth refer to partitions that satisfy this property as *H-partitions* with *degree* at most $(2+\epsilon)\cdot a$ and *size* $\ell = O(\log n)$.

During the execution of this procedure each vertex in V is either active or inactive. Initially, all the vertices are active. For every $i = 1, 2, ..., \ell$, on the ith round each active vertex with at most $(2+\epsilon)\cdot a$ active neighbors joins the set H_i and becomes inactive. The pseudo-code of Procedure Partition is presented below. In all our algorithms the presented pseudo-code is for a given vertex v, and it is executed in parallel by all vertices in the network.

Algorithm 1 Procedure Partition(a,ϵ): partitions the vertices into $\ell = \lfloor \frac{2}{\epsilon}\log n \rfloor$ sets such that every vertex $v \in H_i$, $i \in \{1, 2, ..., \ell\}$, has at most $(2+\epsilon)\cdot a$ neighbors in $\bigcup_{j=i}^{\ell} H_j$.

Initially all vertices are active.

```
1: for round i = 1, 2, ..., ℓ do
2:    if v is active and has at most (2+ε)·a active neighbors
      then
3:       make v inactive
4:       add v to H_i
5:       send the messages 'inactive' and 'v joined H_i' to all
         the neighbors
6:    end if
7:    for each received 'inactive' message do
8:       mark the sender neighbor as inactive
9:    end for
10: end for
```

The next lemma shows that if all the vertices execute this procedure with $\ell = \lfloor \frac{2}{\epsilon}\log n \rfloor$, then each vertex in the network becomes inactive during the execution, and joins one of the sets $H_1, H_2, ..., H_\ell$.

Lemma 3.1. *A graph $G = (V, E)$ with arboricity $a(G)$ has at least $\frac{\epsilon}{2+\epsilon}\cdot|V|$ vertices with degree $(2+\epsilon)\cdot a$ or less.*

PROOF. Suppose for contradiction that there are more than $\frac{2}{2+\epsilon}\cdot|V|$ vertices with degree greater than $(2+\epsilon)\cdot a$. It follows that
$2|E| = \sum_{v\in V} deg(v) > ((2+\epsilon)\cdot a)\cdot|V|\cdot\frac{2}{2+\epsilon} = 2\cdot a\cdot|V|$
$\geq 2\cdot\frac{|E|}{|V|-1}\cdot|V| > 2|E|$. This is a contradiction. □

By the definition of arboricity, the subgraph induced by any subset of V of active vertices has arboricity at most a.

Lemma 3.2. *For any subgraph G' of G, the arboricity of G' is at most the arboricity of G.*

By Lemmas 3.1-3.2, on each round at least $(\frac{\epsilon}{2+\epsilon})$-fraction of the active vertices become inactive, and so after $\log_{(2+\epsilon)/2} n$ rounds all vertices become inactive. Since $\log_{(2+\epsilon)/2} n \leq \frac{2}{\epsilon}\log n$ for ϵ, $0 < \epsilon \leq 2$, we have proved the following lemma.

Lemma 3.3. *For a graph G with $a(G) = a$, and a parameter ϵ, $0 < \epsilon \leq 2$, Procedure Partition(a,ϵ) produces an H-partition $\mathcal{H} = H_1, H_2, ..., H_\ell$ of size $\ell \leq \left\lfloor \log_{(2+\epsilon)/2} n \right\rfloor \leq \lfloor \frac{2}{\epsilon}\log n \rfloor$.*

The next lemma shows that the H-partition $\mathcal{H}$ has a small degree.

Lemma 3.4. *The H-partition $\mathcal{H} = \{H_1, H_2, ..., H_\ell\}$, $\ell \leq \lfloor \frac{2}{\epsilon}\log n \rfloor$, has degree at most $(2+\epsilon)\cdot a$.*

PROOF. The vertex v was added to H_j on round number j. Every neighbor of v that belongs to one of the sets $H_j, H_{j+1}, ..., H_\ell$ was added to its set on round j or later. Therefore, at the end of round $j-1$ all its neighbors in $H_j \cup H_{j+1} \cup ... \cup H_\ell$ were active. The vertex v has been added because the number of its active neighbors was at most $(2+\epsilon)\cdot a$. Thus the number of the neighbors of v in $H_j \cup H_{j+1} \cup ... \cup H_\ell$ is at most $(2+\epsilon)\cdot a$. □

We summarize the properties of Procedure Partition in the following theorem.

Theorem 3.5. *For a graph G with arboricity $a(G) = a$, and a parameter ϵ, $0 < \epsilon \leq 2$, Procedure Partition(a, ϵ) computes an H-partition of size $\ell \leq \lfloor \frac{2}{\epsilon}\log n \rfloor$ with degree at most $(2+\epsilon)\cdot a$. The running time of the procedure is $O(\log n)$.*

We will also use this procedure with second parameter $q \geq 2$. (For convenience, we call this parameter ϵ when it is at most 2, and q when it is larger than 2.) Observe that Lemma 3.1 is applicable for all values of the second parameter. The number of rounds required to make all vertices inactive is at most $\log_{\frac{2+q}{2}} n = O(\frac{\log n}{\log q})$, and thus, for $q > 2$, we set $\ell = \left\lfloor \log_{\frac{2+q}{2}} n \right\rfloor$. Hence the resulting H-partition has size $O(\frac{\log n}{\log q})$ as well. On the other hand, by Lemma 3.4, the degree of the H-partition is at most $(2+q)\cdot a$.

Corollary 3.6. *For a graph G with arboricity $a(G) = a$, and a parameter q, $q > 2$, Procedure Partition(a,q) computes an H-partition of size $O(\frac{\log n}{\log q})$ with degree at most $(2+q)\cdot a$. The running time of the procedure is $O(\frac{\log n}{\log q})$.*

On the next step Procedure Forests-Decomposition orients the edges of the graph as follows. For each edge $e = (u, v)$, if the endpoints u, v are in different sets $H_i, H_j, i \neq j$, then the edge is oriented towards the vertex in the set with a greater index. Otherwise, if $i = j$, the edge e is oriented towards the vertex with a greater ID among the two vertices u and v. The orientation μ produced by this step is acyclic. By Lemma 3.4, each vertex has μ-out-degree at most $(2+\epsilon) \cdot a$. This step is called Procedure *Orientation*.

Finally, on the last step Procedure Forests-Decomposition partitions the edge set of the graph into forests. Each vertex is in charge for its outgoing edges, and it assigns each outgoing edge a different label from the set $\{1, 2, ..., \lfloor(2+\epsilon) \cdot a\rfloor\}$. This step will be henceforth referred as the *labeling step*. It will later be shown that for each index i, the set of edges labeled by i forms a forest.

Algorithm 2 Forests-Decomposition(a,ϵ): partition the edge set into $\lfloor(2+\epsilon) \cdot a\rfloor$ forests.

1: Invoke Procedure Partition(a, ϵ)
2: μ := Orientation()
3: Assign a distinct label to each μ-outgoing edge of v from the set $\{1, 2, ..., \lfloor(2+\epsilon) \cdot a\rfloor\}$

The time complexity of Procedure Partition is $O(\log n)$, and the steps 2 and 3 of Procedure Forests-Decomposition, orienting and labeling the edges, require $O(1)$ rounds each. Hence the overall time complexity of the forests- decomposition algorithm is $O(\log n)$.

Lemmas 3.7-3.9 constitute the proof of correctness of the algorithm for computing forests-decomposition.

Lemma 3.7. *The orientation μ formed by the algorithm is consistent.*

Lemma 3.8. *The orientation μ formed by the algorithm is acyclic.*

The correctness of the last lemma follows from the fact that the orientation μ guarantees that each cycle contains a vertex with two outgoing edges.

For each $i = 1, 2, ..., \ell$, consider the graph $G_i = G(H_i)$ induced by the set H_i. Lemma 3.4 implies that the maximum degree $\Delta(G_i)$ of a vertex in G_i is at most $(2+\epsilon) \cdot a$. Moreover, a stronger result follows:

Lemma 3.9. *Each vertex has μ-out-degree at most $(2+\epsilon)\cdot a$.*

By Lemma 3.9, once the orientation μ is formed, each vertex can assign distinct labels to its outgoing edges from the range $1, 2, ..., \lfloor(2+\epsilon) \cdot a\rfloor$. The next lemma shows that the undirected graph induced by the set of edges labeled with the label i does not contain cycles.

Lemma 3.10. *For each label i, the set of edges labeled by i forms a forest.*

Proof. By Lemma 3.8 , each cycle of G has a vertex with two outgoing edges on this cycle. Suppose for contradiction that there is a cycle C with all edges labeled by the same label i. There exists a vertex v in this cycle and two edges e_1, e_2 adjacent to v oriented outwards of v. Thus, the algorithm labeled the edges e_1, e_2 with different labels, contradiction. □

We summarize this section with the following corollary.

Corollary 3.11. *For a graph G with arboricity $a = a(G)$, and a parameter ϵ, $0 < \epsilon \leq 2$,*
Procedure Forests-Decomposition(a, ϵ) partitions the edge set of G into $\lfloor(2+\epsilon) \cdot a\rfloor$ forests in $O(\log n)$ rounds. Moreover, as a result of its execution each vertex v knows the label and the orientation of every edge (v, u) adjacent to v.

Similarly to Procedure Partition, Procedure Forests- Decomposition can be invoked with second parameter $q > 2$. Lemmas 3.7 - 3.10 stay unchanged, and thus we obtain the following corollary.

Corollary 3.12. *For a graph G with $a(G) = a$, and a parameter q, $q > 2$, Procedure Forests-Decomposition(a, q) partitions the edge set of G into at most $(2+q) \cdot a$ forests within time $O(\frac{\log n}{\log q})$.*

An outgoing edge from a vertex u to a vertex v labeled with a label i means that v is the parent of u in a tree of the ith forest F_i.

3.2 General scenarios

For Algorithm 2 to work properly, each vertex needs to know the number of vertices n, and the arboricity of the graph a at the beginning of the computation. (The number of vertices is required for the vertices to be able to compute the value $\lfloor\frac{2}{\epsilon}\log n\rfloor$.) In this section we extend the algorithm to apply to the scenario in which one of these parameters is not known to the vertices when the computation starts.

If only the number of vertices n is known, we compute a $2\cdot(2+\epsilon)\cdot a$ forests-decomposition without a priori knowledge of a in $O(\log n)$ rounds by the following algorithm. First, we extend Procedure Partition to this scenario. The generalized procedure is called *Procedure General-Partition*.

Procedure General-Partition invokes a procedure similar to Procedure Partition(a,ϵ) for $\lceil\log n\rceil + 1$ times in parallel. The ith invocation of this procedure accepts as input $a = 2^i$, for $i = 0, 1, ..., \lceil\log n\rceil$. Each vertex v maintains a boolean activity array A_v, and round numbers array R_v. The entry $A_v[i]$ is equal to 1 if v is currently active in the invocation of Procedure Partition with the parameter $a = 2^i$. Henceforth we say that v is *i-active* (respectively, *i-inactive*) if it is active (resp., inactive) with respect to invocation i. Initially, all vertices are i-active in all invocations, i.e., $A_v[i] = 1$ for all i. On every round, each i-active vertex v that has at most $(2+\epsilon) \cdot 2^i$ i-active neighbors becomes i-inactive, and the value of $A_v[i]$ is set to 0. In addition, the round number in which it became i-inactive is recorded in $R_v[i]$. We remark that some vertices may stay i-active in some invocation i during the entire execution of the algorithm. However, by a previous argument, when the process stops after $k = \lfloor\frac{2}{\epsilon}\log n\rfloor$ rounds, for all vertices v in the graph G and for all indices i such that $a(G) \leq 2^i \leq n$, the vertex v is i-inactive, i.e., $A_v[i] = 0$.

On round $k+1$ each vertex v joins a set H_{ind}, $1 \leq ind \leq \ell$, for $\ell = O(\log a \log n)$. The index ind of the set H_{ind} depends on the invocation with the smallest index m in which v became m-inactive, and on the number of the round $R_v[m]$ in which it became m-inactive. Note that v had at most $(2+\epsilon) \cdot 2^m$ m-active neighbors when it became m-inactive. All other neighbors w became m-inactive before v did. The index ind of v is computed by the formula $ind = ind(v) = m \cdot \lfloor\frac{2}{\epsilon}\log n\rfloor + R_v[m]$.

For the index ind as above, the set H_{ind} is said to *belong to the class m*. The collection of sets H_{ind} that belong to the

class m is denoted $\mathcal{C}_m$. Observe that there may exist indices $q \in \{1, 2, ..., \ell\}$ for which no vertex v satisfies $ind(v) = q$. The corresponding sets H_q are set as empty, i.e, $H_q := \emptyset$.

The pseudo-code of the algorithm is provided below.

Algorithm 3 General-Partition(ϵ)

Each vertex maintains an activity array A_v of size $\lceil \log n \rceil + 1$. Initially for each vertex v, $A_v[i] = 1$ for $i = 0, 1, ..., \lceil \log n \rceil$. set $k = \lfloor \frac{2}{\epsilon} \log n \rfloor$

1: **for** round $i := 1, 2, ..., k$ **do**
2: **for** $j := 0, 1, ..., \lceil \log n \rceil$ in parallel **do**
3: $Sum[j] := \sum_{u \in adj(v)} A_u[j]$ /* sum of arrays of neighbors of v */
4: **end for**
5: **for** $j := 0, 1, ..., \lceil \log n \rceil$ in parallel **do**
6: **if** $A_v[j] = 1$ and $Sum[j] \leq (2 + \epsilon) \cdot 2^j$ **then**
7: /* if v has $\leq (2 + \epsilon) \cdot 2^j$ active neighbors in the invocation j */
8: $A_v[j] := 0$
9: $R_v[j] := i$
10: **end if**
11: **end for**
12: send the array A_v to all the neighbors
13: **end for**
14: $m := \min \{i \mid A_v[i] = 0, i = 0, 1, ..., \lceil \log n \rceil\}$
15: ind $:= m \cdot k + R_v[m]$
16: join the set H_{ind}
17: send the message 'v joined H_{ind}' to all neighbors

Note that on step 11 only $O(\log n)$ bits are sent in each message.

Next, we show that the degree of the H-partition $\mathcal{H} = \{H_1, H_2, ..., H_\ell\}$ is $O(a(G))$.

Recall that for a vertex v and a set H_i, we say that the set H_i is the *set of* v if $v \in H_i$.

The index i as above is called the *H-index of* v.

Lemma 3.13. *For an index $m = 1, 2, ..., \lceil \log n \rceil$, a set $H_{ind} \in \mathcal{C}_m$, and a vertex $v \in H_{ind}$,*
(1) $deg(v, \bigcup_{j=ind}^{\ell} H_j) \leq (2 + \epsilon) \cdot 2 \cdot a(G)$.
(2) $deg(v, \bigcup \{H_j \mid H_j \in \mathcal{C}_m, j \geq ind\}) \leq (2 + \epsilon) \cdot 2^m$.

Proof. For a neighbor w that became m-inactive before v did, the number of the invocation with the smallest index q in which w becomes q-inactive at some point during the execution is less or equal to m. If $q < m$ then

$$ind(w) = q \cdot \left\lfloor \frac{2}{\epsilon} \log n \right\rfloor + R_w[q] \leq ind(v) = m \cdot \left\lfloor \frac{2}{\epsilon} \log n \right\rfloor + R_v[m],$$

because $1 \leq R_w[q], R_v[m] \leq \lfloor \frac{2}{\epsilon} \log n \rfloor$. If $q = m$ then $R_w[m] < R_v[m]$, since w became m-inactive before v did. Hence in this case as well, $ind(w) < ind(v)$. Since v has at most $(2+\epsilon) \cdot 2^m$ other neighbors, the number of neighbors of v with an H-index greater or equal than the H-index ind of v is at most $(2+\epsilon) \cdot 2^m$. Hence $deg(v, \bigcup \{H_j \mid H_j \in \mathcal{C}_m, j \geq ind\}) \leq deg(v, \bigcup_{j=ind(v)}^{\ell} H_j) \leq (2+\epsilon) \cdot 2^m$, proving the second assertion of the lemma. Since for all i such that $a(G) \leq 2^i \leq n$, at the end of the execution $A_v[i] = 0$ holds, it follows that $2^m \leq 2 \cdot a(G)$. Therefore, the overall number ℓ of sets H_i is at most $\log(2 \cdot a(G)) \cdot \lfloor \frac{2}{\epsilon} \log n \rfloor = O(\log a(G) \cdot \log n)$. Also, it follows that for a vertex $v \in H_t$, $t = 1, 2, ..., \ell$, $deg(v, \bigcup_{j=t}^{\ell} H_j) \leq (2 + \epsilon) \cdot 2^m \leq (2 + \epsilon) \cdot 2 \cdot a(G)$, proving the first assertion. □

The algorithm for computing the forests-decomposition when only n is known, Procedure General-Forests- Decomposition, starts with invoking Procedure General-Partition with input ϵ. Step 2, Procedure Orientation, is executed exactly as in Algorithm 2, and produces an orientation μ of the graph. Finally, on the Labeling step each vertex that has δ outgoing edges with respect to μ assigns distinct labels to its outgoing edges from the set $\{1, 2, ..., \delta\}$. By the same argument as in the proof of Lemma 3.8, the orientation μ is an acyclic orientation. Also, by Lemma 3.9, for every vertex v the μ-out-degree of v is at most $2 \cdot (2 + \epsilon) \cdot a(G)$.

The properties of Procedure General-Forests- Decomposition are summarized in the following corollary.

Corollary 3.14. *Procedure General-Forests-Decomposition(ϵ) computes a forests-decomposition of the input graph $G = (V, E)$ into $O(a(G))$ forests. In addition, the procedure produces an H-partition of size $O(\log a \log n)$ and degree at most $(2 + \epsilon) \cdot 2 \cdot a(G)$. The running time of the procedure is $O(\log n)$.*

We remark that the upper bound on the degree of the H-partition can be made as close to $(2+\epsilon) \cdot a(G)$ as one wishes, at the expense of increasing its size and the running time of the procedure by a constant factor.

One can also run Procedure General-Forests-Decomposition with an input parameter q, $q > 2$. By the same considerations, this way we obtain a forests-decomposition into at most $a(G) \cdot (2+q)$ forests in time $O(\frac{\log n}{\log q})$, and an H-partition of size $O(\log a \cdot \frac{\log n}{\log q})$ and degree at most $a(G) \cdot (2 + q)$.

4. $O(A)$-COLORING

In this section we employ the Forests-Decomposition algorithm described in Section 3 to devise an efficient algorithm, called Procedure *Arb-Color*, that colors the input graph G of arboricity $a = a(G)$ in $(\lfloor (2 + \epsilon) \cdot a \rfloor + 1)$-colors, for an arbitrarily small parameter $\epsilon > 0$. The running time of the algorithm is $O(a \cdot \log n)$. We present the algorithm for the scenario when all vertices know both the number of vertices n, and the arboricity at the beginning of computation. The algorithm can be extended to the scenarios in which the arboricity or the number of vertices is not known in advance, using standard synchronization techniques in Procedure Arb-Color on top of Procedure General-Forests-Decomposition. The number of colors in the extended algorithm is not affected, and the running time grows only by a constant factor.

Set $A = \lfloor (2 + \epsilon) \cdot a \rfloor$. We say that a color c is *admissible for a vertex v with respect to the vertex set H* if each neighbor of v in $G(H)$ has a color different from c. We will prove that whenever a vertex is required to choose an admissible color by the algorithm, there is at least one such a color in the range $1, 2, , ..., A + 1$.

The algorithm starts by executing Procedure Forests- Decomposition with the input parameter $a = a(G)$. This invocation returns an H-partition of G of size $\ell \leq \lfloor \frac{2}{\epsilon} \log n \rfloor$ and degree at most A. Then, for each index i, the graph $G_i = G(H_i)$ induced by the set H_i is colored using the KW $(\Delta + 1)$-coloring algorithm. By Lemma 3.4, for all i, $i = 1, 2, ..., \lfloor \frac{2}{\epsilon} \log n \rfloor$, the subgraph G_i satisfies $\Delta(G_i) \leq A$. Hence the algorithm colors each graph G_i with at most $A + 1$ colors. The resulting coloring is not necessarily a legal coloring for the entire network G. We then convert

it into a legal $(A+1)$-coloring for G using the subroutine *Recolor*. The latter subroutine accepts as input the H-partition $H_1, H_2, ..., H_\ell$ that satisfies the above properties, with each set H_i being $(A+1)$-colored legally. Procedure Recolor merges these $(A+1)$ colorings of $H_1, H_2, ..., H_\ell$ into a unified legal $(A+1)$-coloring of the entire vertex set $V = \bigcup_{i=1}^{\ell} H_i$.

The vertices of the set H_ℓ retain their colors. Vertices of the sets $H_1, H_2, ..., H_{\ell-1}$ are recolored iteratively. On the first iteration vertices of the set $H_{\ell-1}$ are recolored, and in the end of this iteration the set $H_{\ell-1} \cup H_\ell$ is $(A+1)$-colored legally. More generally, for $i = 1, 2, ..., \ell-1$, before the iteration i starts the set $\bigcup_{j=\ell-i+1}^{\ell} H_j$ is $(A+1)$-colored legally. On iteration i vertices of the set $H_{\ell-i}$ are recolored, and in the end of this iteration the set $\bigcup_{j=\ell-i}^{\ell} H_j$ is $(A+1)$-colored legally. The algorithm maintains also an auxiliary set W of vertices that were already recolored. Before the iteration i starts, $W = \bigcup_{j=\ell-i+1}^{\ell} H_j$, and during the iteration i it holds that $\bigcup_{j=\ell-i+1}^{\ell} H_j \subseteq W \subseteq \bigcup_{j=\ell-i}^{\ell} H_j$.

To recolor the set $H_{\ell-i}$ (on the ith iteration of Procedure Recolor), the algorithm uses the $(A+1)$-coloring φ of $H_{\ell-i}$ that was computed on step 2. Specifically, the algorithm recolors one color class of $H_{\ell-i}$ at a time. It starts with finding (in parallel) an admissible color from the set $\{1, 2, ..., A+1\}$ with respect to W for every vertex $v \in H_{\ell-i}$ such that $\varphi(v) = 1$.

Observe that for every vertex $v \in H_{\ell-i}$, $deg(v, W) \leq deg(v, \bigcup_{j=\ell-i}^{\ell} H_j) \leq A$, and thus, there necessarily exists an admissible color for v with respect to W in the set $\{1, 2, ..., A+1\}$. In addition, since φ is a legal coloring of $H_{\ell-i}$, it follows that the vertex set $H_{\ell-i}^1 = \{v \in H_{\ell-i} \mid \varphi(v) = 1\}$ is an independent set, and thus vertices of $H_{\ell-i}^1$ can be recolored in parallel. Once $H_{\ell-i}^1$ is recolored, the algorithm proceeds to recoloring $H_{\ell-i}^2 = \{v \in H_{\ell-i} \mid \varphi(v) = 2\}$, $H_{\ell-i}^3 = \{v \in H_{\ell-i} \mid \varphi(v) = 3\}$,..., $H_{\ell-i}^{A+1} = \{v \in H_{\ell-i} \mid \varphi(v) = A+1\}$. Later we argue that the resulting $(A+1)$-coloring of $\bigcup_{j=\ell-i}^{\ell} H_j$ is legal.

The pseudo-code of Procedure Arb-Color is provided below.

Algorithm 4 Procedure Arb-Color(a,ϵ)

1: $\mathcal{H} = (H_1, H_2 ..., H_\ell)$:= Forests-Decomposition(a,ϵ).
2: In parallel, color each graph G_i, $i = 1, 2, ..., \ell$, with $A+1$ colors using the KW coloring algorithm. Denote the resulting colorings φ_i, $i = 1, 2, ..., \ell$.
3: Recolor($\mathcal{H}$).

Algorithm 5 Procedure Recolor ($\mathcal{H} = (H_1, H_2, ..., H_\ell)$)

1: $W := \emptyset$
2: **for** $i := \ell - 1$ downto 1 **do**
3: **for** round k from 1 to $A+1$ **do**
4: **for** each vertex v in H_i such that $\varphi_i(v) = k$ **do**
5: recolor v with a color from $\{1, 2, ..., A+1\}$ that is admissible with respect to W
6: $W := W \cup \{v\}$
7: **end for**
8: **end for**
9: **end for**

The next corollary follows directly from Lemma 3.4.

Corollary 4.1. *For and index i, $i = 1, 2, ..., \ell$, and any coloring of the vertices of G, legal or illegal, each vertex v that belongs to H_i has an admissible color in the range $1, 2, ..., A+1$ with respect to $\bigcup_{j=i}^{\ell} H_j$.*

The correctness of Procedure Arb-Color is proven in the next theorem.

Theorem 4.2. *Procedure Arb-Color produces a legal $(A+1)$-coloring.*

Proof. Step 1 of Algorithm 4 divides the vertex set V of the graph G into $\ell = O(\log n)$ sets H_i. By Lemma 3.4, for each index i, $i = 1, 2, ..., \ell$, the maximum degree of $G_i = G(H_i)$ is at most A. Step 2 of Algorithm 4 produces a legal $(A+1)$-coloring for each G_i, $i = 1, 2, ..., \ell$.
We prove by induction on i that Procedure Recolor produces a legal $(A+1)$-coloring for the graph induced by $H_{\ell-i} \cup H_{\ell-i+1} \cup ... \cup H_\ell$.
Base ($i = 0$)**:** Step 2 of Procedure Arb-Color produces a legal coloring for H_ℓ. This coloring does not change in step 3. Therefore, when the algorithm terminates, G_ℓ is $(A+1)$-colored legally.
Induction step: Let φ denote the $(A+1)$-coloring of the graph $G(\bigcup_{j=\ell-i+1}^{\ell} H_j)$ produced by the first $i-1$ iterations of Procedure Recolor. By the induction hypothesis, φ is a legal $(A+1)$-coloring. Also, let $\varphi_{\ell-i}$ denote the legal $(A+1)$-coloring of $G_{\ell-i}$ produced on step 2 of Procedure Arb-Color.

We argue that the ith iteration produces a legal $(A+1)$-coloring φ' for $G(\bigcup_{j=\ell-i}^{\ell} H_j)$. Consider two neighboring vertices u, v in $\bigcup_{j=\ell-i}^{\ell} H_j$. If they both belong to $\bigcup_{j=\ell-i+1}^{\ell} H_j$ then their colors do not change during the ith iteration, and so $\varphi'(u) = \varphi(u) \neq \varphi(v) = \varphi'(v)$, as required. If they both belong to $H_{\ell-i}$ then $\varphi_{\ell-i}(u) \neq \varphi_{\ell-i}(v)$. In other words, u and v were colored differently before the ith iteration has began. Hence u and v select their respective colors $\varphi'(u)$ and $\varphi'(v)$ on different rounds of the ith iteration. Suppose without loss of generality that v selects a color after u does so. Since v selects an admissible color $\varphi'(v)$ with respect to W and $u \in W$ is a neighbor of v, it follows that $\varphi'(u) \neq \varphi'(v)$.

Finally, we are left with the case that one of the vertices u and v belongs to $H_{\ell-i}$, and the other to $\bigcup_{j=\ell-i+1}^{\ell} H_j$. Suppose without loss of generality that $u \in H_{\ell-i}$ and $v \in \bigcup_{j=\ell-i+1}^{\ell} H_j$. In this case the color of v does not change on the ith iteration, i.e., $\varphi'(v) = \varphi(v)$. When the vertex u sets its color $\varphi'(u)$ it selects an admissible color. Since v is a neighbor of u, it follows that $\varphi'(u) \neq \varphi'(v)$, and we are done. □

Recall that $A = O(a)$, and thus, Procedure Arb-Color produces an $O(a)$-coloring of the input graph.

Lemma 4.3. *The time complexity of Procedure Arb-Color is $O(a \log n)$.*

Proof. By Corollary 3.11, Procedure Forests- Decomposition requires $O(\log n)$ rounds. Let i be the index such that the maximum degree of G_i is the largest among $G_1, G_2, ..., G_\ell$. Since the graphs $G_1, G_2, ..., G_\ell$ are colored in parallel, step 2 of Algorithm 4 requires $O(\Delta(G_i) \cdot \log \Delta(G_i) + \log^* n)$ rounds. In addition, the maximum degree of G_i is at most A for every index $i = 1, 2, ..., \ell$, and $A = O(a)$. Hence, it follows that the time complexity of step 2 is $O(a \log a + \log^* n)$. Step 3 of Algorithm 4, Procedure Recolor, invokes $\ell - 1 = O(\log n)$ iterations, each running for $A+1$ rounds. Hence this step requires $O(a \log n)$ rounds. □

We summarize this section with the following theorem.

Theorem 4.4. *For a graph G with arboricity $a = a(G)$, and a positive parameter ϵ, $0 < \epsilon \leq 2$, Procedure Arb-Color(a, ϵ) computes an $O(a)$ coloring of G in time $O(a \log n)$.*

We remark that invoking Procedure Arb-Color with $q > 2$ as second parameter results in inferior results than those given by Theorem 4.4.

5. FASTER COLORING

In this section we present two algorithms. Both algorithms exhibit tradeoffs between the running time and the number of colors that they employ. For a positive parameter t, $1 \leq t \leq a$, our first algorithm, Procedure *Tradeoff-Color*, computes an $O(t \cdot a)$-coloring in time $O(\frac{a}{t} \cdot \log n + a \log a)$. Our second algorithm, Procedure *Tradeoff-Arb-Linial*, achieves an $O(q \cdot a^2)$-coloring within time $O(\frac{\log n}{\log q} + \log^* n)$. Again, we assume that the arboricity and the number of vertices are known in advance, but it is possible to extend those algorithms to general scenarios.

5.1 Procedure Tradeoff-Color

Similarly to Procedure Arb-Color (Algorithm 4), Procedure Tradeoff-Color consists of three steps. Moreover, steps 1 and 2 are exactly the same as in Procedure Arb-Color, and the only difference is that on step 3 it invokes Procedure Tradeoff-Recolor instead of Procedure Recolor.

Similarly to Procedure Recolor, Procedure Tradeoff-Recolor accepts as input the H-partition $\mathcal{H} = \{H_1, H_2, ..., H_\ell\}$ of the graph G computed by Procedure Forests-Decomposition on step 1. Both Procedure Recolor and Procedure Tradeoff-Recolor proceed iteratively, and in both procedures vertices of the set H_ℓ retain their colors, and on iteration i, $i = 1, 2, ..., \ell - 1$, vertices of the set $H_{\ell-i}$ are recolored. The difference between the two procedures is that while in Procedure Recolor each color class of $H_{\ell-i}$ is recolored in a separate round, Procedure Tradeoff-Recolor recolors roughly t color classes of $H_{\ell-i}$ on the same round. Specifically, Procedure Tradeoff-Color groups the $(A+1)$ color classes $C_1, C_2, ..., C_{A+1}$ of $H_{\ell-i}$ into $p = \lceil \frac{A+1}{t} \rceil$ disjoint subsets $S_1, S_2, ..., S_p$. Each subset S_j, $j = 1, 2, ..., p$, contains the color classes C_r with indices
$r \in I_j = \{(j-1)t + 1, (j-1)t + 2, ..., \min\{j \cdot t, p\}\}$.

The ith iteration of Procedure Tradeoff-Recolor continues for p rounds. On round j, $j = 1, 2, ..., p$, vertices of color classes C_r, $r \in I_j$, are recolored in parallel. To guarantee that no pair of neighboring vertices $u \in C_r$, $w \in C'_r$, $r \neq r'$, $r, r' \in I_j$, will select the same color, the color classes $\{C_r \mid r \in I_j\}$ are assigned disjoint palettes $\{\mathcal{P}_r \mid r \in I_j\}$,
$\mathcal{P}_r = \{(A+1)(r-1-(j-1)t) + 1,$
$(A+1)(r-1-(j-1)t)+2, ..., (A+1)(r-1-(j-1)t)+(A+1)\ \}$.

In other words, the color class $C_{(j-1)t+1}$ is assigned the palette $\mathcal{P}_{(j-1)t+1} = \{1, 2, ..., A+1\}$, the color class $C_{(j-1)t+2}$ is assigned the palette $\mathcal{P}_{(j-1)t+2} = \{(A+1)+1, (A+1)+2,$ $..., 2(A+1)\ \}$, etc.

Consider a vertex $v \in C_r$, $r \in I_j$. On round j of the ith iteration the vertex v selects an admissible color from its palette $\mathcal{P}_r$ with respect to the set W of already recolored vertices. This completes the description of Procedure Tradeoff-Recolor.

Since each palette $\mathcal{P}_r$ contains $(A+1)$ colors, and $deg(v, W) \leq deg(v, \bigcup_{j=\ell-i}^{\ell} H_j) \leq A$, it follows that there necessarily exists an admissible color for v with respect to W. An inductive argument similar to the one employed in the proof of Theorem 4.2 shows that Procedure Tradeoff-Color produces a legal coloring.

For an upper bound on running time, observe that Procedure Tradeoff-Recolor runs for $O(\log n)$ iterations, and each iteration requires $\lceil \frac{A+1}{t} \rceil = O(\frac{a}{t})$ rounds. Hence the running time of Procedure Tradeoff-Recolor is $O(\frac{a}{t} \log n)$. The running time of step 1 of Procedure Tradeoff-Color, that is, of Procedure Forests-Decomposition, is $O(\log n)$. Finally, step 2 of Procedure Tradeoff-Color (see step 2 of Procedure Arb-Color, Algorithm 4) requires $O(a \log a + \log^* n)$ rounds. Hence the overall running time of Procedure Tradeoff-Color is $O(\frac{a}{t} \cdot \log n + a \log a)$.

However, the improved running time of Procedure Tradeoff-Color has a price. Specifically, since we used t disjoint palettes of size $A + 1$ each, the number of colors that were used is $t \cdot (A + 1) = O(t \cdot a)$. We summarize the properties of Procedure Tradeoff-Color in the following theorem.

Theorem 5.1. *For a positive parameter t, $1 \leq t \leq a$, Procedure Tradeoff-Color produces a legal $O(a \cdot t)$-coloring of the input graph in time $O(\frac{a}{t} \cdot \log n + a \log a)$.*

5.2 Procedures Arb-Linial and Tradeoff-Arb-Linial

Observe that by substituting $t = a$ in Theorem 5.1 we obtain an $O(a^2)$-coloring algorithm with running time $O(\log n + a \log a)$. Next, we present another $O(a^2)$-coloring algorithm that has an even better running time of $O(\log n)$. The improved algorithm, Procedure *Arb-Linial*, like the algorithm of Linial [17], relies on the following combinatorial result by Erdős et al. [8]. (The proof can also be found in [17]).

Theorem 5.2. *For positive integers n' and r, $n' > r$, there exists a family $\hat{Q} = \hat{Q}(n', r)$ of n' subsets of $\{1, 2, ..., \lceil 5r^2 \cdot \log n' \rceil\}$ that satisfies that for every $r+1$ sets $Q_0, Q_1, ..., Q_r \in \hat{Q}$, $Q_0 \notin \bigcup_{i=1}^{r} Q_i$.*

Our algorithm consists of two steps. On the first step it constructs a forests-decomposition $\mathcal{F} = \{F_1, F_2, ..., F_A\}$ of the input graph G, and on the second step it uses $\mathcal{F}$ for computing the $O(a^2)$-coloring of G. The first (forests-decomposition) step entails an invocation of Procedure Forests-Decomposition with the input parameter $a = a(G)$ and ϵ, $0 < \epsilon \leq 2$. By Corollary 3.11, this invocation produces a forests-decomposition F with $A \leq \lfloor (2+\epsilon) \cdot a \rfloor$ forests. For a vertex v and a forest F_i, $i \in \{1, 2, ..., A\}$, such that $v \in V(F_i)$ and such that v has a parent in F_i, let $\pi_i(v)$ denote the parent of v in F_i. Finally, let $\Pi(v)$ denote the set of neighbors u of v such that $u = \pi_i(v)$ for some index i, $i \in \{1, 2, ..., A\}$.

The second (coloring) step of the algorithm proceeds iteratively. Initially, each vertex v uses its distinct identity number $\mathrm{ID}(v)$ as its color. On each round vertices recolor themselves while maintaining the legality of the coloring. The number of colors used by these coloring is gradually reduced from n to $O(a^2)$. Similarly to the algorithm of Linial [17], the reduction in the number of colors consists of two phases. The first phase continues for $O(\log^* n)$ rounds, and reduces the number of colors from n to $O(a^2 \log a)$. The second phase lasts for just one single round, and reduces the number of colors to $O(a^2)$.

In each round of the coloring step each vertex v sends its current color to all its neighbors. Fix a round R and a vertex v, and let $\varphi(v)$ and $\{\varphi(u) \mid u \in \Pi(v)\}$ be the colors of v and the colors of its parents in forests $F_1, F_2, ..., F_A$

in the beginning of round R, respectively. Also, let p be the current upper bound on the number of colors employed by the algorithm. (Initially, $p = n$.) Based on the colors $\varphi(v)$ and $\{\varphi(u) \mid u \in \Pi(v)\}$, and on parameters p and A, the vertex v computes the set system $\hat{Q}(p, A)$ whose existence is guaranteed by Theorem 5.2. This computation is performed by v locally, involving no communication whatsoever. Then the vertex v selects an arbitrary new color $\varphi'(v)$ from $Q_{\varphi(v)} \setminus \bigcup \{Q_{\varphi(u)} \mid u \in \Pi(v)\}$. By Theorem 5.2, the set $Q_{\varphi(v)} \setminus \bigcup \{Q_{\varphi(u)} \mid u \in \Pi(v)\}$ is not empty. Moreover, by Theorem 5.2, $\varphi'(v) \in \{1, 2, ..., \lceil 5A^2 \cdot \log p \rceil\}$, and thus, the vertex v updates its upper bound on the number of employed colors from p to $\lceil 5A^2 \cdot \log p \rceil$.

After $O(\log^* n)$ rounds the number of colors reduces to $O(A^2 \log A)$. Employing another related set system $\mathcal{T}$ exactly in the same way as described above, our algorithm reduces the number of colors to $O(A^2)$. The required set system $\mathcal{T}$ is given by the following theorem.

Theorem 5.3. *[17],[8]: There exists a collection $\mathcal{T}$ of $O(A^2 \log A)$ subsets of $\{1, 2, ..., O(A^2)\}$ such that for every $A + 1$ subsets $T_0, T_1, ..., T_A \in \mathcal{T}$, $T_0 \notin \bigcup_{i=1}^{A} T_i$.*

We remark that Procedure Arb-Linial is essentially a composition of Linial $O(\Delta^2)$-coloring algorithm with our algorithm for computing forests-decomposition. The main difference of the coloring step of Procedure Arb-Linial from the original Linial coloring algorithm is that in Procedure Arb-Linial each vertex considers only the colors of its *parents* in forests $F_1, F_2, ..., F_A$. On the other hand, in the algorithm of Linial each vertex considers the colors of *all its neighbors*.

The next lemma shows that all colorings produced throughout the algorithm are legal.

Lemma 5.4. *Suppose that the coloring φ is legal. Then the coloring φ' is legal as well.*

PROOF. Consider an edge $e = (u, v) \in E$. Since $F = \{F_1, F_2, ..., F_A\}$ is a partition of the edge set E into disjoint forests, there exist an index $i \in \{1, 2 ...A\}$ such that $e \in E(F_i)$. Suppose without loss of of generality that u is the parent of v. Then $u \in \Pi(v)$, and consequently $\varphi'(v) \in Q_{\varphi(v)} \setminus Q_{\varphi(u)}$. On the other hand, $\varphi'(u) \in Q_{\varphi(u)}$ and so $\varphi'(v) \neq \varphi'(u)$, as required. □

By Corollary 3.11, the running time of Procedure Forests-Decomposition is $O(\log n)$. The coloring step of Procedure Arb-Linial requires $O(\log^* n)$ time. Hence the overall running time of the algorithm Arb-Linial is $O(\log n)$. Observe that $A = O(a)$, and thus, the resulting coloring is an $O(a^2)$ coloring. To summarize, we have proved the following theorem.

Theorem 5.5. *Procedure Arb-Linial computes a legal $O(a^2)$-coloring in $O(\log n)$ time.*

Next, we present a variant of Procedure Arb-Linial, Procedure *Tradeoff-Arb-Linial*, that provides a tradeoff between the number of colors and the running time. Procedure Tradeoff-Arb-Linial accepts as input $a = a(G)$, and a parameter q, $q > 2$. On its first step it invokes Procedure Forests-Decomposition with the same pair of parameters a and q. By Corollary 3.12, this procedure partitions the edge set of G into at most $(2 + q) \cdot a$ forests, and it does so within time $O(\frac{\log n}{\log q})$. The second recoloring step of Procedure Tradeoff-Arb-Linial is very similar to that of Procedure Arb-Linial. The only difference is that the value of A is now $(2+q) \cdot a$ and not $\lfloor (2 + \epsilon) \cdot a \rfloor$ as it was in Procedure Arb-Linial. By the same argument, Procedure Tradeoff-Arb-Linial computes an $O(A^2 \cdot q^2) = O(a^2 \cdot q^2)$-coloring within time $O(\frac{\log n}{\log q} + \log^* n)$.

Finally, set $q' = q^2$. We get an $O(a^2 \cdot q')$-coloring within time $O(\frac{\log n}{\log q} + \log^* n)$.

Corollary 5.6. *For an n-vertex graph G with arboricity a and a parameter q, $2 < q \leq O(\sqrt{\frac{n}{a}})$, Procedure Tradeoff-Arb-Linial invoked with parameters a and q computes $O(a^2 \cdot q)$-coloring in time $O(\frac{\log n}{\log q} + \log^* n)$.*

6. MIS ALGORITHMS AND LOWER BOUNDS

In this section we capitalize on the results of Section 5, and present an algorithm that computes an MIS in graphs with bounded arboricity in sublogarithmic time. The algorithm employs a standard reduction from MIS to coloring [21], described below.

First, observe that by Corollary 5.6, for any graph with arboricity $o(\sqrt{\log n})$, a legal $o(\log n)$-coloring can be found in $o(\log n)$ time. Then a standard technique [21] that reduces the number of colors, one color per round, can be used to achieve $(\Delta + 1)$-coloring in additional $o(\log n)$ rounds. We summarize this fact in the following corollary.

Corollary 6.1. *For a graph G with arboricity $a(G) = o(\sqrt{\log n})$, both $(\Delta+1)$-coloring and $o(\log n)$-coloring can be found in sublogarithmic time.*

Suppose we are given a legal p-coloring of the graph, for some positive integer p. Let $U_1, U_2, ..., U_p$ be the disjoint color classes, with all vertices of U_i being colored by i, for $i = 1, 2, ..., p$. Initialize the independent set W as U_1. The reduction algorithm proceeds iteratively, taking care of one of the color classes $U_2, U_3, ..., U_p$ on each of the $p - 1$ iterations. For iteration $i = 1, 2, ..., p - 1$, before the iteration i starts, an independent set $W \subseteq \bigcup_{j=1}^{i} U_j$ is maintained. On iteration i each vertex $v \in U_{i+1}$ checks in parallel whether it has a neighbor $w \in W$. If it has, it decides not to join W. Otherwise it joins W. Obviously, the algorithm requires $(p - 1)$ rounds, and produces an MIS.

The next theorem follows directly from Corollary 6.1.

Theorem 6.2. *Consider an n-vertex graph G with arboricity $a(G) = o(\sqrt{\log n})$. Procedure Tradeoff-Arb-Linial combined with the standard reduction from an MIS to coloring, computes an MIS of G in time $o(\log n)$. Moreover, whenever $a = O(\log^{1/2-\delta} n)$, for some constant δ, $0 < \delta < 1/2$, the same algorithm runs in time $O(\frac{\log n}{\log \log n})$.*

Whenever $a = \Omega(\sqrt{\log n})$ we use the same reduction in conjunction with Procedure Tradeoff-Color. The running time of the resulting algorithm for computing MIS becomes $O(\frac{a}{t} \cdot \log n + a \log a + a \cdot t)$. This expression is optimized by setting $t = \min\{\sqrt{\log n},\ a\}$.

Theorem 6.3. *Consider an n-vertex graph G with arboricity $a(G) = \Omega(\sqrt{\log n})$. Procedure Tradeoff-Color invoked with parameters a and $t = \sqrt{\log n}$, combined with the standard reduction from MIS to coloring, computes an MIS of G in time $O(a \cdot \sqrt{\log n} + a \log a)$.*

Our lower bounds rely on the following result of Linial [17].

Theorem 6.4. *[17] For a pair of positive integer numbers n and d, $n-1 \geq d$, any distributed algorithm for coloring the d-regular n-vertex tree which has running time at most $O(\log_d n)$ uses at least $\Omega(\sqrt{d})$ colors.*

Consider the family $\mathcal{G}$ of planar graphs. Suppose that a correct algorithm for q-coloring $\mathcal{G}$ requires at least $t(q)$ time in the worst-case. Since unoriented $O(q^2)$-regular tree is a planar graph, it follows that $t(q) = \Omega(\frac{\log n}{\log q})$. By another lower bound of Linial [17], 3-coloring an oriented path requires $\Omega(\log^* n)$ time. We conclude that q-coloring planar graphs requires $\Omega(\frac{\log n}{\log q} + \log^* n)$ time, which matches our upper bound up to constant factors. (See Corollary 5.6.) The next result follows directly.

Corollary 6.5. *For graphs with arboricity a, and a parameter $q = O(\sqrt{n}/a^2)$, $O(a^2 \cdot q)$-coloring requires $\Omega(\frac{\log n}{\log a + \log q} + \log^* n)$ time.*

Next, we turn to forests-decomposition lower bounds. Once an $O(a \cdot q)$-forests decomposition is computed, it is possible to compute $O(a^2 \cdot q^2)$-coloring by step 2 of procedure Arb-Linial, in additional $O(\log^* n)$ time. Since $O(a^2 \cdot q^2)$-coloring requires $\Omega(\frac{\log n}{\log a + \log q})$ time, the following result is achieved.

Theorem 6.6. *For an n-vertex graph of arboricity a, and a parameter q, $q \geq 1$, $q = O(n^{1/4}/a)$, computing an $O(q \cdot a)$-forests-decomposition requires $\Omega(\frac{\log n}{\log q + \log a} - \log^* n)$ time.*

7. CONCLUSIONS

In this paper we have presented efficient deterministic MIS and coloring algorithms for the family of graphs with arboricity polylogarithmic in n. Although this is a wide and important family of graphs, the question regarding the existence of efficient deterministic algorithms for yet wider families remains open. In particular, it is currently not clear whether it is possible to extend our results to graphs with arboricity at most $2^{\log^\epsilon n}$ for some constant $\epsilon > 0$. Also, we have devised a sublogarithmic time MIS algorithm on graphs with arboricity at most $o(\sqrt{\log n})$. It would be interesting to extend this result to graphs with arboricity $o(\log n)$.

8. 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] S. R. Arikati, A. Maheshwari, and C. Zaroliagis. Efficient Computation of Implicit Representations of Sparse Graphs. *Discrete Applied Mathematics*, 78(1-3):1-16, 1997.

[3] 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.

[4] B. Bollobas. *Extremal Graph Theory.* Academic Press, 1978.

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

[6] N. Deo and B. Litow. A structural approach to graph compression. In *Proc. of the MFCS Workshop on Communications*, pages 91–100, 1998.

[7] V. Dujmovic, and D. R. Wood. Graph Treewidth and Geometric Thickness Parameters. *Discrete and Computational Geometry*, 37(4):641–670, 2007.

[8] P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51:79–89, 1985.

[9] 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

[10] 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.

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

[12] 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.

[13] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast Deterministic Distributed Maximal Independent Set Computation on Growth-Bounded Graphs. In *19th International Symposium on Distributed Computing*, 2005.

[14] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be ccomputed locally! In P*roc. of the 23rd ACM Symp. on Principles of Distributed Computing*, pages 300-309, 2004.

[15] 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*, 2005.

[16] 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.

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

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

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

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

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

[22] 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*, 2008.

[23] G. Singh. Efficient leader election using sense of direction. *Distributed Computing*, 10(3):159Ű-165, 1997.

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