Distrib. Comput. (2010) 22:363–379
DOI 10.1007/s00446-009-0088-2

Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition

Leonid Barenboim · **Michael Elkin**

Received: 16 September 2008 / Accepted: 9 August 2009 / Published online: 3 September 2009
© Springer-Verlag 2009

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 an 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.

Keywords MIS · Coloring · Arboricity · Forests-decomposition

This research has been supported by the Israeli Academy of Science, grant 483/06.

L. Barenboim (✉) · M. Elkin
Department of Computer Science,
Ben-Gurion University of the Negev,
P.O. Box 653, 84105 Beer-Sheva, Israel
e-mail: leonidba@cs.bgu.ac.il

M. Elkin
e-mail: elkinm@cs.bgu.ac.il

1 Introduction

1.1 Distributed message passing model

We study symmetry breaking problems in computer networks. The network is modeled by an undirected unweighted n-vertex graph $G = (V, E)$. The processors in the network are represented by the vertices of G. For each two vertices $u, v \in V$, there is an edge $(u, v) \in E$ if and only if the two processors corresponding to u and v in the network are connected by a communication link. The processors communicate over the edges of G.

Traditionally, symmetry breaking problems have been studied in the *synchronous* model [5,11,18,19]. In this model, the communication proceeds in discrete rounds. There is a global clock that is accessible to all the vertices which counts the rounds. In each communication round each vertex $v \in V$ can send a short message of size $O(\log n)$ bits to each of its neighbors, and these messages arrive before the next round starts. In addition, it can perform local computations based on the information from messages that it has received so far. 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 $\mathcal{A}$.

The need to break symmetry in the distributed message-passing model arises for problems that have multiple solutions. These solutions are sensitive to local perturbations, i.e., changing a (local) decision taken by one given vertex may ruin the entire (global) solution. On the other hand, from the local perspective of each given vertex at the beginning of the computation, all possible decisions appear to be equally feasible.

We focus on deterministic algorithms for the *maximal independent set* (henceforth, MIS) and *coloring* problems. These problems are among the most important problems in symmetry breaking. It has been shown that it is impossible to break symmetry using deterministic algorithms if the vertices are anonymous [12]. Consequently, we use a common assumption [5,10,18,17] that each vertex has a distinct identity number (henceforth, ID) represented by bit string of length $O(\log n)$.

1.2 MIS

A subset $I \subseteq V$ of vertices is called a Maximal Independent Set of G if

(1) I is an independent set, i.e., 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 an MIS is one of the most fundamental problems in the area of distributed algorithms. More than 20 years ago Luby [19] and Alon et al. [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 et al. [3] devised the first deterministic algorithm for this problem on general graphs, which was later improved by Panconesi and Srinivasan [22] in 1992. The latter algorithm 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 et al. [15].

Cole and Vishkin [5] presented a deterministic algorithm for computing an MIS on oriented rings and paths. The running time of the algorithm of [5] is $O(\log^* n)$.[1] Linial [18] has shown that this result is tight up to constant factors. In 1988 Goldberg et al. [11] initiated the study of the MIS problem on *sparse* graphs. They devised deterministic algorithms for the MIS problem on oriented trees and on planar graphs. The running time of their algorithm for oriented trees (respectively, planar graphs) is $O(\log^* n)$ (resp., $O(\log n)$). Their second algorithm (the one that applies to planar graphs) extends also to graphs of *bounded genus*. The results of Goldberg et al. [11] were stated in both distributed message-passing and PRAM models of computation.

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. It is equal to the minimum number of forests into which the edge set of the graph can be partitioned. 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 $\epsilon > 0$). (See Sect. 2 for a more detailed comparison between various graph families.)

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 [11,17,18] and the family of graphs with bounded growth [9,14,24]. 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 for graphs with arboricity $a = \Omega(\sqrt{\log n})$, an MIS 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.

1.3 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 [10,11,17,18,26]. 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. In other words, the vertex set has to be partitioned into color classes, such that each color class forms an independent set. 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 et al. [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 [18] extended this result to general values of Δ. Recently, Kuhn and Wattenhofer [17] presented a $(\Delta+1)$-coloring algorithm with running time $O(\Delta \log \Delta + \log^* n)$. For planar graphs,

[1] This algorithm was presented in the PRAM model.

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 significantly 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 $a(G) \le a$ 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 $q \ge 1$, 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 particular, this implies that in just $O(\log^* n)$ time one can color planar graphs (and other graphs with bounded arboricity) using $O(n^{1/\log^* n}) = n^{o(1)}$ colors. In addition, for a positive parameter t, $1 \le t \le a$, 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 $\Omega(\frac{\log n}{\log a + \log q})$ time, and thus our first tradeoff is nearly optimal.

Using these results we show a *separation* between the problems of MIS and $O(a)$-coloring, in the following sense. The lower bound mentioned above implies that it is impossible to compute an $O(a)$-coloring of graphs with bounded arboricity in sublogarithmic time. Nevertheless, we devise an MIS algorithm for this family of graphs with running time $O(\frac{\log n}{\log\log n})$. Therefore, the problem of computing $O(a)$-coloring is harder than computing an MIS on graphs with bounded arboricity.

1.4 Forests-decomposition

A *forests-decomposition* is a partition of the edge set of a graph into edge-disjoint subsets, such that each subset forms a forest. By definition, the edge set E of any graph $G = (V, E)$ of arboricity $a = a(G)$ can be decomposed into a edge-disjoint forests. Moreover, a is the minimum number of forests into which the graph edge set can be decomposed. An equivalent definition formulated by Nash-Williams [21] in 1964 states that

$$a(G) = \max\left\{\left\lceil \frac{|E(G')|}{|V(G')| - 1}\right\rceil : \ G' \subseteq G, |V(G')| \ge 2\right\}.$$

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 procedure for computing forests-decompositions. Specifically, we demonstrate that for a parameter q, $q \ge 1$, 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 $\Omega(\frac{\log n}{\log q + \log a}) - O(\log^* n)$ for this problem, demonstrating that our algorithm is near-optimal. Remarkably, all our algorithms 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.

1.5 Related work

Recently, MIS and coloring problems were studied on unit disk, unit ball, and more generally, bounded growth graphs [9,14,16,24]. Specifically, Kuhn et al. [16] 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 [14] 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 [24]. 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 a 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 by 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 [13,25]. In particular, Kothapalli et al. [13] devised a randomized algorithm that constructs an $O(\Delta)$-coloring of G in time $O(\log \Delta + \sqrt{\log n \log\log n})$ with high probability.

1.6 The structure of the paper

In Sect. 2 we present the basic notions used throughout the paper. In Sect. 3 we present our algorithms for computing forests-decomposition. In Sects. 4 and 5 we employ these algorithms to devise efficient coloring algorithms. Section 6 is devoted to the MIS problem. Finally, in Sect. 7 we present our lower bounds.

2 Preliminaries

2.1 Definitions and notation

Unless the base value is specified, all logarithms in this paper are to 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 $i = 0, 1, 2, \ldots$. Also, for a positive integer n, $\log^* n$ is defined by: $\log^* n = \min\left\{i \mid \log^{(i)} n \le 2\right\}$.

The *degree* of a vertex v in an undirected 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. The *neighborhood* $\Gamma(v)$ of v is the set of neighbors of v. 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 notation $V(G')$ and $E(G')$ is used to denote the vertex set V' of G', and the edge set E' of G', respectively. The *out-degree* of a vertex v in a directed graph $\hat{G}$ is the number of edges connected to v that are oriented outwards of v. A *directed cycle* in a directed graph $\hat{G}$ is a cycle whose edges are oriented consistently. In a simple directed cycle, each vertex in the cycle is adjacent 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* $a(G)$ of a graph $G = (V, E)$ is the minimum number of forests into which E can be partitioned. An equivalent definition due to Nash-Williams [21] is given by:

$$a(G) = \max\left\{\left\lceil \frac{|E(G')|}{|V(G')| - 1} \right\rceil : G' \subseteq G, |V(G')| \ge 2\right\}.$$

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 $\varphi(v) \ne \varphi(u)$ 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 [17] 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 $a(G) \le \Delta(G)$, and thus, the family of graphs with bounded arboricity contains the family of graphs with bounded degree.

For an integer parameter $g > 0$, a graph G is said to have *orientable* (respectively, *non-orientable*) *genus* g if g is the smallest number such that G can be drown on an orientable (resp. non-orientable) surface of genus g in such a way that no two edges of G intersect. Intuitively, an orientable (resp., non-orientable) surface of genus g can be thought of as a sphere with g "handles" (resp, "cross-caps"). The orientable and non-orientable genuses of a given graph are always within a factor of 2 of each other. See [20] for further details. Henceforth we use the notion *genus* as a shortcut for *orientable genus*.

For a graph $G = (V, E)$ of bounded genus g, by Euler formula, $|E| \le 3|V| - 6 + 6 \cdot g$. In addition, for any subgraph $G' = (V', E')$ of G, the genus of G' is at most g. Therefore, for any subgraph G' of G, it follows that

$$|E'| \le \min\left\{3|V'| - 6 + 6 \cdot g,\ |V'| \cdot (|V'| - 1)\right\}.$$

Thus,

$$a(G) \le \max\left\{\left\lceil \min\left\{\frac{3|V'| + 6 \cdot g}{|V'| - 1}, \frac{|V'| \cdot (|V'| - 1)}{|V'| - 1}\right\}\right\rceil \right.$$
$$\left. : V' \subseteq V,\ |V'| \ge 2\right\} = O(\sqrt{g}).$$

Hence the family of graphs of bounded genus 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 family 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 Sect. 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 Sect. 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

In the first step, our algorithm for computing a forests-decomposition, henceforth called 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 $H_1, H_2, \ldots, H_\ell$ that satisfy that every vertex $v \in H_i, i \in \{1, 2, \ldots, \ell\}$, has at most $(2+\epsilon)\cdot a$ neighbors in the vertex set $\cup_{j=i}^{\ell} H_j$, i.e., $deg(v, \cup_{j=i}^{\ell} H_j) \le (2+\epsilon)\cdot a$. 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 $i = 1, 2, \ldots, \ell$, 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 $v \in H_i$, $i \in \{1, 2, \ldots, \ell\}$, has at most $(2+\epsilon)\cdot a$ neighbors in $\bigcup_{j=i}^{\ell} H_j$.

```
Initially all vertices are active.
1: for round $i = 1, 2, \ldots, \ell$ do
2:   if $v$ is active and has at most $(2+\epsilon)\cdot a$ 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 each vertex in the network becomes inactive during the execution, and joins one of the sets $H_1, H_2, \ldots, H_\ell$.

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 $2|E| = \sum_{v\in V} deg(v) > ((2+\epsilon)\cdot a)\cdot|V|\cdot\frac{2}{2+\epsilon} = 2\cdot a\cdot|V| \ge 2\cdot\frac{|E|}{|V|-1}\cdot|V| > 2|E|$. 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. For ϵ in the range $0 < \epsilon \le 2$, it holds that $\log_{(2+\epsilon)/2} n \le \frac{2}{\epsilon}\log n$. For $\epsilon > 2$ we note that $\log_{(2+\epsilon)/2} n = O(\frac{\log n}{\log \epsilon})$. Hence, we have proved the following lemma.

Lemma 3.3 *For a graph G with $a(G) = a$, and a parameter $\epsilon > 0$, Procedure Partition(a,ϵ) produces an H-partition*

$\mathcal{H} = H_1, H_2, \ldots\ldots, H_\ell$ of size $\ell \le \lfloor \log_{(2+\epsilon)/2} n \rfloor$. It does so within at most ℓ rounds. Whenever $\epsilon \le 2$, ℓ is at most $\lfloor \frac{2}{\epsilon} \log n \rfloor$. In the complementary range, $\ell = O(\frac{\log n}{\log e})$.

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

Lemma 3.4 *The H-partition $\mathcal{H} = \{H_1, H_2, \ldots, H_\ell\}$ 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, $H_{j+1}, \ldots, H_\ell$ was added to its set on round j or later. Therefore, at the end of round $j-1$ all its neighbors in $H_j \cup H_{j+1} \cup \cdots \cup H_\ell$ 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 $H_j \cup H_{j+1} \cup \cdots \cup H_\ell$ 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 $\epsilon > 0$, Procedure Partition(a, ϵ) computes an H-partition of size ℓ with degree at most $(2+\epsilon) \cdot a$. The running time of the procedure is $O(\log n)$ whenever $\epsilon \le 2$, and it is $O(\frac{\log n}{\log \epsilon})$ for $\epsilon > 2$.*

For convenience, we will denote the second parameter of Procedure Partition by ϵ whenever $\epsilon \le 2$, and by q whenever it is greater than 2.

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, $i \ne j$, 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 $\{1, 2, \ldots, \lfloor (2+\epsilon) \cdot a \rfloor\}$. 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($a$, $\epsilon$)
2: $\mu$ := Orientation()
3: Assign a distinct label to each $\mu$-outgoing edge of $v$ from the set
   $\{1, 2, \ldots, \lfloor (2+\epsilon) \cdot a \rfloor\}$
```

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 3.6–3.8 constitute the proof of correctness of the algorithm for computing forests-decomposition.

Lemma 3.6 *The orientation μ formed by the algorithm is consistent.*

Proof For an edge $e = (u, v)$, if u orients e towards v then either the index of v is greater than the index of u, or they have the same index but $ID(u) < ID(v)$. In both cases v orients e towards v as well. □

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

Proof We show that there are no directed cycles with respect to μ. Let C be a cycle of G. Let v be a vertex in C such that the H-index i of v (that is, the index i s.t. $v \in H_i$) is the smallest index of a vertex in C, and such that $ID(v)$ is the smallest identity number in $H_i \cap C$. Let u, w denote the two neighbors of v in C. Obviously, both edges (v, u) and (v, w) are oriented outwards of v, and thus, the μ-out-degree of u in the cycle is 2. Hence C is not a directed cycle with respect to μ. Consequently, the orientation μ is acyclic. □

For each $i = 1, 2, \ldots, \ell$, 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 statement follows:

Lemma 3.8 *Each vertex has μ-out-degree at most $(2+\epsilon) \cdot a$.*

Proof Let v be a vertex of G. Let j be the H-index of v. Each outgoing edge of v is connected to a vertex with an H-index that is greater or equal to j. Hence by Lemma 3.4, v has at most $(2+\epsilon) \cdot a$ outgoing edges. □

By Lemma 3.8, once the orientation μ is formed, each vertex can assign distinct labels to its outgoing edges from the range $1, 2, \ldots, \lfloor (2+\epsilon) \cdot a \rfloor$. The next lemma shows that the undirected graph induced by the set of edges labeled with the label i does not contain cycles.

Lemma 3.9 *For each label i, the set of edges labeled by i forms a forest.*

Proof By Lemma 3.7, each cycle in the original undirected graph G has a vertex with two μ-outgoing edges on this cycle. Suppose for contradiction that there is a cycle C in G 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 that are oriented outwards of v by μ. Thus, the algorithm labeled the edges e_1, e_2 with different labels, a contradiction. □

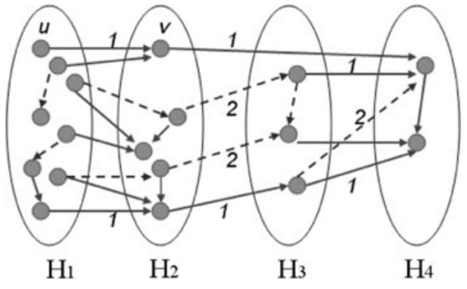

Fig. 1 Forests-decomposition. (Some vertices and edges are omitted from the figure for clarity.) An outgoing edge from a vertex u to a vertex v labeled with a label i means that v is the parent of u in a tree of the ith forest F_i. *Solid edges* represent edges with the label '1'. *Dashed edges* represent edges with the label '2'

Consider a directed subgraph induced by all edges labeled by the same label i. By the previous Lemma, this subgraph is a forest. We refer to this forest as F_i. The labeling step assigned distinct colors to each outgoing edge of each vertex of G. Consequently, each vertex in F_i has at most one outgoing edge connecting it with another vertex in F_i. Hence, the orientation μ inside F_i determines the parent-child relationship in the following way. If a vertex v has no outgoing edges in F_i, it is a root in the forest F_i. Otherwise, it has exactly one outgoing edge in F_i that connects it with its parent. All the other edges adjacent to v in F_i connect it with its children. To summarize, in each forest F_i, μ orients the edges "bottom-up", from children to their parents.

We summarize this section with the following corollary.

Corollary 3.10 *For a graph G with arboricity $a = a(G)$, and a positive parameter ϵ, Procedure Forests-Decomposition(a, ϵ) partitions the edge set of G into $\lfloor (2+\epsilon)\cdot a \rfloor$ forests. The running time of the procedure is $O(\log n)$ whenever $\epsilon \le 2$, and $O(\frac{\log n}{\log \epsilon})$ for $\epsilon > 2$. Moreover, as a result of its execution each vertex v knows the label and the orientation of every edge (v, u) adjacent to v.*

See Fig. 1 for an illustration.

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 $\lfloor \frac{2}{\epsilon} \log n \rfloor$, and the arboricity is needed to compute the degree threshold $(2+\epsilon)\cdot a$.) 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. Specifically, we devise algorithms for the scenario when a is not known, but n is known. We extend it to even more general scenario in which both a and n are unknown, but the vertices are provided with a polynomial estimate of n.

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 $i = 0, 1, \dots, \lceil \log n \rceil$. 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. In every round, for all i, 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 $a(G) \le 2^i \le n$, the vertex v is i-inactive, i.e., $A_v[i] = 0$. (Since when Procedure Partition is invoked with the parameters $a' \ge a(G)$, and ϵ, all vertices become inactive during its execution. See Lemmas 3.1–3.3).

In round $k+1$ each vertex v joins a set H_{ind}, $1 \le ind \le \ell$, for $\ell = \log((2+\epsilon)\cdot 2\cdot a)\cdot \frac{2}{\epsilon}\log n = O(\log a \log n)$. (Here, ℓ is an upper bound on the number of sets H created by the procedure.) 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 $q \in \{1, 2, \dots, \ell\}$ 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. Note that in step 12 only $O(\log n)$ bits are sent in each message. Next, we show that the degree of the H-partition $\mathcal{H} = \{H_1, H_2, \dots, H_\ell\}$ 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*.

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 $i = 0, 1, \ldots, \lceil \log n \rceil$.
set $k = \lfloor \frac{2}{\epsilon} \log n \rfloor$
1: for round $i := 1, 2, \ldots, k$ do
2:   for $j := 0, 1, \ldots, \lceil \log n \rceil$ in parallel do
3:     $Sum[j] := \sum_{u \in \Gamma(v)} A_u[j]$   /* $Sum[j]$ is the number of
       $j$-active neighbors of $v$ */
4:   end for
5:   for $j := 0, 1, \ldots, \lceil \log n \rceil$ in parallel do
6:     if $A_v[j] = 1$ and $Sum[j] \le (2 + \epsilon) \cdot 2^j$ then
7:       /* if $v$ has $\le (2 + \epsilon) \cdot 2^j$ $j$-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: $m := \min\{i \mid A_v[i] = 0, i = 0, 1, \ldots, \lceil \log n \rceil\}$
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
```

Lemma 3.11 *For an index $m = 1, 2, \ldots, \lceil \log n \rceil$, a set $H_{ind} \in \mathcal{C}_m$, and a vertex $v \in H_{ind}$,*

(1) $deg(v, \cup_{j=ind}^{\ell} H_j) \le (2 + \epsilon) \cdot 2 \cdot a(G)$.
(2) $deg(v, \cup \{H_j \mid H_j \in \mathcal{C}_m, j \ge ind\}) \le (2 + \epsilon) \cdot 2^m$.

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

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

because $1 \le R_w[q], R_v[m] \le \lfloor \frac{2}{\epsilon} \log n \rfloor$. 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 $(2 + \epsilon) \cdot 2^m$ neighbors that became m-inactive after v did, or in the same round as v did, 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 $deg(v, \cup \{H_j | H_j \in \mathcal{C}_m, j \ge ind\}) \le deg(v, \cup_{j=ind(v)}^{\ell} H_j) \le (2 + \epsilon) \cdot 2^m$, proving the second assertion of the lemma.

Observe that for all i such that $a(G) \le 2^i \le n$, at the end of the execution $A_v[i] = 0$ holds. Recall also that m is the smallest index such that $A_v[m] = 0$ at the end of the execution. Hence, it follows that $2^m \le 2 \cdot a(G)$. 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 $v \in H_t$, $t = 1, 2, \ldots, \ell$, $deg(v, \cup_{j=t}^{\ell} H_j) \le (2 + \epsilon) \cdot 2^m \le (2 + \epsilon) \cdot 2 \cdot a(G)$, proving the first assertion. □

Next, we build upon Procedure General-Partition to devise an algorithm (Procedure General-Forests- Decomposition) for computing the forests-decomposition in the scenario 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, in the Labeling step each vertex that has δ outgoing edges with respect to μ assigns distinct labels to its outgoing edges from the set $\{1, 2, \ldots, \delta\}$. By the same argument as in the proof of Lemma 3.7, the orientation μ is an acyclic orientation. Also, by Lemma 3.8, 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 3.12 *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 $2 \cdot a(G)$ as one wishes. Let ϵ', $\epsilon' > 0$, denote an arbitrarily small positive constant. The only modification is that in Procedure General-Partition we run $\lceil \log_{1+\epsilon'} n \rceil + 1$ executions of Procedure Partition in parallel, with values of the threshold parameter $a = (1+\epsilon')^i$, for $i = 0, 1, \ldots, \lceil \log_{1+\epsilon'} n \rceil$, instead of $\lceil \log n \rceil + 1$ executions with values $a = 2^i$, $i = 0, 1, \ldots, \lceil \log_2 n \rceil + 1$. This way the number of forests in the resulting decomposition is at most $(1 + \epsilon')(2 + \epsilon) \cdot a(G)$. The time complexity remains the same, up to a constant factor. The size of the H-partition, as well as the maximum message size, grows also only 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 $a(G) \cdot (2 + q)$.

Finally, consider the scenario in which neither the graph arboricity nor the number of vertices n is known to the vertices in advance, but instead vertices are provided with a polynomial estimate N of n. (Specifically, there exists a universal constant c, $c > 1$, such that $n^{1/c} \le N \le n^c$.) In this scenario we replace all occurrences of n in Procedure General-Partition with N^c. It is easy to see that precisely the same analysis applies, but the size ℓ of the H-partition becomes at most $\lfloor \frac{2}{\epsilon} \log N^c \cdot \log(2a) \rfloor \le \lfloor c^2 \cdot \frac{2}{\epsilon} \log n \cdot \log(2a) \rfloor$, and the running time of the algorithm grows by at most a factor c^2 as well. To summarize, Corollary 3.12 holds up to constant factors in this more general scenario as well.

4 $O(a)$-Coloring

In this section we employ the Forests-Decomposition algorithm described in Sect. 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 $O(a \cdot \log n)$. In Sect. 4.1 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. In Sect. 4.2 the algorithm is extended to the scenarios in which the arboricity or the number of vertices is not known in advance. The number of colors in the extended algorithm is not affected, and the running time grows only by a constant factor.

4.1 Known arboricity

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 $1, 2, , \dots, A+1$.

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 $\ell \leq \left\lfloor \frac{2}{\epsilon} \log n \right\rfloor$ 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 (see Sect. 2.1). By Lemma 3.4, for all i, $i = 1, 2, \dots, \left\lfloor \frac{2}{\epsilon} \log n \right\rfloor$, 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 $H_1, H_2, \dots, H_\ell$ that satisfies the above properties, with each set H_i being $(A+1)$-colored legally. Procedure Recolor merges these $(A+1)$ colorings of $H_1, H_2, \dots, H_\ell$ into a unified legal $(A+1)$-coloring of the entire vertex set $V = \cup_{i=1}^{\ell} H_i$.

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

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 $\{1, 2, \dots, A+1\}$ with respect to W for every vertex $v \in H_{\ell-i}$ such that $\varphi(v) = 1$.

Observe that for every vertex $v \in H_{\ell-i}$, $deg(v, W) \leq deg(v, \cup_{j=\ell-i}^{\ell} H_j) \leq A$, and thus, there necessarily exists an admissible color for v with respect to W in the set $\{1, 2, \dots, A+1\}$. 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 $H_{\ell-i}^2 = \{v \in H_{\ell-i} \mid \varphi(v) = 2\}$, $H_{\ell-i}^3 = \{v \in H_{\ell-i} \mid \varphi(v) = 3\}, \dots, H_{\ell-i}^{A+1} = \{v \in H_{\ell-i} \mid \varphi(v) = A+1\}$. Later we argue that the resulting $(A+1)$-coloring of $\cup_{j=\ell-i}^{\ell} H_j$ is legal.

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

Algorithm 4 Procedure Arb-Color(a,ϵ)

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

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

```
1: $W := \emptyset$
2: for $i := \ell - 1$ downto 1 do
3:   for round $k := 1$ to $A+1$ do
4:     for each vertex $v$ in $H_i$ such that $\varphi_i(v) = k$ (in parallel) do
5:       recolor $v$ with a color from $\{1, 2, \dots, A+1\}$ 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 an index i, $i = 1, 2, \dots, \ell$, 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 $1, 2, \dots, A+1$ with respect to $\cup_{j=i}^{\ell} H_j$.*

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, $i = 1, 2, \ldots, \ell$, 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, $i = 1, 2, \ldots, \ell$. We prove by induction on i that Procedure Recolor produces a legal $(A+1)$-coloring for the graph induced by $H_{\ell-i} \cup H_{\ell-i+1} \cup \cdots \cup H_\ell$.

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 Ψ denote the $(A+1)$-coloring of the graph $G(\cup_{j=\ell-i+1}^{\ell} H_j)$ produced by the first $i-1$ iterations of Procedure Recolor. By the induction hypothesis, Ψ 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 Ψ' for $G(\cup_{j=\ell-i}^{\ell} H_j)$. Consider two neighboring vertices u, v in $\cup_{j=\ell-i}^{\ell} H_j$. If they both belong to $\cup_{j=\ell-i+1}^{\ell} H_j$ then their colors do not change during the ith iteration, and so $\Psi'(u) = \Psi(u) \neq \Psi(v) = \Psi'(v)$, as required. If they both belong to $H_{\ell-i}$ then $\varphi_{\ell-i}(u) \neq \varphi_{\ell-i}(v)$. In other words, in this case u and v were colored differently before the ith iteration has started. Hence u and v select their respective colors $\Psi'(u)$ and $\Psi'(v)$ 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 $\Psi'(v)$ with respect to W and $u \in W$ is a neighbor of v, it follows that $\Psi'(u) \neq \Psi'(v)$.

Finally, we are left with the case that one of the vertices u and v belongs to $H_{\ell-i}$, and the other to $\cup_{j=\ell-i+1}^{\ell} H_j$. Suppose without loss of generality that $u \in H_{\ell-i}$ and $v \in \cup_{j=\ell-i+1}^{\ell} H_j$. In this case the color of v does not change on the ith iteration, i.e., $\Psi'(v) = \Psi(v)$. When the vertex u sets its color $\Psi'(u)$ it selects an admissible color. Since v is a neighbor of u, it follows that $\Psi'(u) \neq \Psi'(v)$, 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 3.10, 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 $G_1, G_2, \ldots, G_\ell$. Since the graphs $G_1, G_2, \ldots, G_\ell$ are colored in parallel, step 2 of Algorithm 4 requires $O(\Delta(G_i) \cdot \log \Delta(G_i) + \log^* n)$ rounds. (Recall that the time complexity of the KW coloring algorithm is $O(\Delta \log \Delta + \log^* n)$.) In addition, the maximum degree of G_i is at most A for every index $i = 1, 2, \ldots, \ell$, 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 $\epsilon = q > 2$ as second parameter results in inferior bounds than those given by Theorem 4.4. Specifically, the running time becomes $O(q \cdot a \cdot \frac{\log n}{\log q})$, and the number of used colors increases to $O(q \cdot a)$.

4.2 General scenarios

In this section we extend Procedure Arb-Color described in Sect. 4.1 to the scenario when the value of the graph arboricity is not known to the vertices before the beginning of the computation. They are, however, assumed to know the number of vertices n, or at least a polynomial estimate of n.

The running time of our algorithm, Procedure *General-Arb-Color*, for $O(a)$-coloring in this scenario is still $O(a \log n)$. Similarly to Procedure Arb-Color, Procedure General-Arb-Color consists of three steps. In the first step Procedure General-Forests-Decomposition is used instead of Procedure Forests-Decomposition. (See Sect. 3.2.) This procedure returns an H-partition $H_1, H_2, \ldots, H_\ell$ with $\ell = O(\log a \log n)$ and degree at most $2A$.

In the second step subgraphs $G_i = G(H_i)$, $i = 1, 2, \ldots, \ell$, are colored in parallel using the KW coloring algorithm. To properly run the KW coloring algorithm, vertices need to know an upper bound on the maximum degree of the underlying graph. (The KW algorithm invokes Linial's algorithm [18] as a subroutine, and Linial's algorithm needs this information.) Let m be the index of the class such that $H_i \in \mathcal{C}_m$. By Lemma 3.11, $\Delta(G_i) \leq (2+\epsilon) \cdot 2^m$. Since all vertices of H_i know their class index m, it follows that they can compute the upper bound $\Delta_m = (2+\epsilon) \cdot 2^m$, and employ it for running the KW coloring algorithm. The resulting coloring will use at most $\Delta_m + 1 = ((2+\epsilon) \cdot 2^m + 1)$ colors.

Another difficulty arises on the third (recoloring) step. In Procedure Arb-Color the vertices of $H_{\ell-1}$ recolor themselves in the beginning, then the vertices of $H_{\ell-2}$ do, etc. However, if the arboricity a is not known to the vertices, a vertex v has no way to compute $\ell = O(\log a \log n)$ and consequently, it cannot deduce the index of the round in which v should recolor itself.

To overcome this difficulty, we modify the algorithm as follows. Consider a vertex $v \in H_i$, and let φ_i be the coloring of H_i computed on step 2. The vertex v will recolor itself once it learns that every neighbor u of v that belongs to H_j for $j > i$ recolored itself, and moreover, that every vertex $w \in H_i$ with $\varphi_i(w) > \varphi_i(v)$ did so as well. This completes the description of the modified algorithm.

Observe that this rule guarantees that when a vertex $v \in H_i$ recolors itself, all its neighbors that have already been recolored belong to $\cup_{j=i}^{\ell} H_j$. Consequently, by Lemma 3.11, at this point v has at most $2A$ recolored neighbors. Thus there necessarily exists an admissible color from $\{1, 2, \ldots, 2A+1\}$ for v with respect to the set of already recolored vertices.

Lemma 4.5 *After each recoloring step the set W of already recolored vertices is legally $(2A + 1)$-colored.*

Proof The proof is by induction on the number of recoloring steps.

At the beginning, before the first recoloring step, $W = \emptyset$, and the statement holds vacuously.

Consider some recoloring step. We only need to show that no two neighboring vertices v and u recolor themselves on this step. Suppose without loss of generality that the H-index i_v of v is no greater than the H-index i_u of u, i.e., $i_v \le i_u$. If $i_v < i_u$, and u recolors itself on this step, then v does not recolor itself as it has an unrecolored neighbor with a larger H-index.

If $i = i_v = i_u$ then suppose without loss of generality that $\varphi_i(v) < \varphi_i(u)$. (Recall that φ_i is a legal coloring of H_i computed at step 2.) Then, again, if u recolors itself on this step then v does not do so, and we are done. □

Next, we analyze the running time of the algorithm.

Lemma 4.6 *The running time is $O(a \log n)$.*

Proof By Corollary 3.10, the first step, Procedure General-Forests-Decomposition, requires $O(\log n)$ time. To analyze the running time of the second step, recall that no vertex joins a set H_i of class m with $m > \lfloor \log 2a \rfloor$. (See the proof of Lemma 3.11.) Coloring the graph $G_i = G(H_i)$ using the KW coloring algorithm with an upper bound Δ_m on the maximum degree requires $O(\Delta_m \cdot \log \Delta_m + \log^* n)$ time. Since $\Delta_m = (2 + \epsilon) \cdot 2^m$, the overall running time of the second step is $O(a \cdot \log a + \log^* n)$.

Finally, the running time T of the recoloring step satisfies $T \le \sum_{m=0}^{\lfloor \log 2a \rfloor} T_m$, where for each index $m = 0, 1, \ldots, \lfloor \log 2a \rfloor$, T_m is the time required to recolor all sets H_i of class m. Recall that each set H_i of class m is $(\Delta_m + 1)$-colored, for $\Delta_m = \lfloor (2 + \epsilon) \cdot 2^m \rfloor$, and that there are at most $O(\log n)$ sets in each class. Hence, for each $m = 0, 1, \ldots \lfloor \log 2a \rfloor$, $T_m \le O(\log n) \cdot (\Delta_m + 1) = O(\log n) \cdot 2^m$. Hence, $T \le \sum_{m=0}^{\lfloor \log 2a \rfloor} T_m = O(a \cdot \log n)$. □

Similarly to Sect. 3.2, it is easy to see that the number of employed colors can be reduced to $\lfloor (2 + \epsilon) \cdot a \rfloor$ for an arbitrarily small constant $\epsilon > 0$.

Finally, consider the most general scenario in which the vertices provided only with a polynomial estimate N of the number of vertices, $n^{1/c} \le N \le n^c$ for a universal constant $c > 1$. In this case we replace all occurrences of n in Procedure General-Arb-Color by N^c. It is easy to verify that the modified algorithm is correct, and that the running time grows only by a constant factor.

We summarize this section with the following corollary.

Corollary 4.7 *Procedure General-Arb-Color produces an $O(a)$-coloring in $O(a \log n)$ time.*

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, $1 \le t \le a$, 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 $O(\frac{\log n}{\log q} + \log^* n)$. In Sect. 5.1 we assume that the arboricity and the number of vertices are known in advance. In Sect. 5.2 we extend those algorithms to general scenarios.

5.1 Known arboricity

5.1.1 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 in step 3 it invokes Procedure Tradeoff-Recolor instead of Procedure Recolor.

Similarly to Procedure Recolor, Procedure Tradeoff-Recolor accepts as input the H-partition $\mathcal{H} = \{H_1, H_2, \ldots, H_\ell\}$ of the graph G computed by Procedure Forests-Decomposition in 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, $i = 1, 2, \ldots, \ell - 1$, 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 $C_1, C_2, \ldots, C_{A+1}$ of $H_{\ell-i}$ into $p = \lceil \frac{A+1}{t} \rceil$ disjoint subsets $S_1, S_2, \ldots, S_p$. Each subset S_j, $j = 1, 2, \ldots, p$, contains the color classes C_r with indices $r \in I_j = \{(j - 1)t + 1, (j - 1) t + 2, \ldots, \min\{j \cdot t, p\}\}$.

The ith iteration of Procedure Tradeoff-Recolor continues for p rounds. In round j, $j = 1, 2, \ldots, p$, 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 $\{\mathcal{P}_r \mid r \in I_j\}$, $\mathcal{P}_r = \{(A+1)(r-1-(j-1)t)+1, (A+1)(r-1-(j-1)t)+2, \ldots, (A+1)(r-1-(j-1)t)+(A+1)\}$.

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

Consider a vertex $v \in C_r$, $r \in I_j$. In 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 $deg(v, W) \leq deg(v, \cup_{j=\ell-i}^{\ell} H_j) \leq A$, 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 the 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 comes at 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 $O(\frac{a}{t} \cdot \log n + a \log a)$.*

5.1.2 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 [18], relies on the following combinatorial result by Erdős et al. [8]. (The proof can also be found in [18]).

Theorem 5.2 [8,18] *For positive integers n' and r, $n' > r$, there exists a family $\hat{Q} = \hat{Q}(n', r)$ of n' subsets of $\{1, 2, \ldots, \lceil 5r^2 \cdot \log n' \rceil\}$ that satisfies that for every $r+1$ sets $Q_0, Q_1, \ldots, Q_r \in \hat{Q}$, $Q_0 \nsubseteq \cup_{i=1}^{r} Q_i$.*

Our algorithm consists of two steps. In the first step it constructs a forests-decomposition $\mathcal{F} = \{F_1, F_2, \ldots, F_A\}$ 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 3.10, 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, $i \in \{1, 2, \ldots, A\}$, 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 all parents of v.

The second (coloring) step of the algorithm proceeds iteratively. Initially, each vertex v uses its distinct identity number ID(v) as its color. In 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 [18], 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 $F_1, F_2, \ldots, F_A$ 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 $Q_{\varphi(v)} \setminus \cup \{Q_{\varphi(u)} \mid u \in \Pi(v)\}$. By Theorem 5.2, the set $Q_{\varphi(v)} \setminus \cup \{Q_{\varphi(u)} \mid u \in \Pi(v)\}$ is not empty. Moreover, by Theorem 5.2, $\varphi'(v) \in \{1, 2, \ldots, \lceil 5A^2 \cdot \log p \rceil\}$, 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 [8,18]*: There exists a collection $\mathcal{T}$ of $O(A^2 \log A)$ subsets of $\{1, 2, \ldots, O(A^2)\}$ such that for every $A+1$ subsets $T_0, T_1, \ldots, T_A \in \mathcal{T}$, $T_0 \nsubseteq \cup_{i=1}^{A} T_i$.*

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 $F_1, F_2, \ldots, F_A$. 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 $\mathcal{F} = \{F_1, F_2, \ldots, F_A\}$ is a partition of the edge set E into disjoint forests, there exist an index $i \in \{1, 2 \ldots A\}$ 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 3.10, 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 3.10, 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 $O(\frac{\log n}{\log q} + \log^* n)$.*

In particular, by substituting $q = n^{1/\log^* n}$, we get an $O(n^{1/\log^* n})$-coloring of graphs with bounded arboricity in just $O(\log^* n)$ time.

5.2 General scenarios

5.2.1 Procedure tradeoff-color

It is easy to verify that the same considerations that were applied to Procedure Arb-Color in Sect. 4.2 can be applied to Procedure Tradeoff-Color. Consequently, Theorem 5.1 is applicable in the more general scenarios as well. In particular, it is applicable if the vertices are provided with only a polynomial estimate N on the number of vertices n instead of the number of vertices n itself, and are not provided with the arboricity a of the input graph.

5.2.2 Procedure Arb-Linial

Observe that Procedure Arb-Linial is applicable if instead of the arboricity a the vertices are provided with an upper bound $a' > a$ on the arboricity. However, in this case the argument of Theorem 5.5 guarantees that the algorithm computes a legal coloring that uses at most $O((a')^2)$ colors, and not necessarily $O(a^2)$.

We adapt Procedure Arb-Linial to the scenario when the arboricity a is not known to the vertices before the computation starts. Let c, $c > 0$, be the universal constant hidden by the O-notation in the expression "$O(a^2)$-coloring" in Theorem 5.5. (In other words, the implied coloring uses at most $c \cdot a^2$ colors.)

To adapt the first (forests-decomposition) step, we employ Procedure General-Forests-Decomposition instead of Procedure Forests-Decomposition. A difficulty arises, however, in the second (coloring) step, when the vertices need to compute the set system $\hat{Q}(p, A)$ without knowing the value of A. We show that this difficulty can be overcome without increasing the running time of the algorithm, if we allow the algorithm to use messages of a slightly larger size, specifically, $O(\log^2 n)$. In addition, we show that using messages of size at most $O(\log n)$, one can still complete the coloring step within only a slightly larger running time of $O(\log n \cdot \log^* n)$.

To implement the coloring step without knowing A within the same running time by using larger messages we replace the coloring step of Procedure Arb-Linial with $\lceil \log n \rceil + 1$ invocations $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{\lceil \log n \rceil + 1}$ of the original coloring step, with each invocation $\mathcal{A}_i$ using the arboricity parameter $A_i = 2^i$, for $i = 0, 1, \ldots, \lceil \log n \rceil + 1$. These invocations are executed in parallel and employ distinct palettes $\mathcal{P}_i, i = 0, 1, \ldots, \lceil \log n \rceil + 1$. Specifically, the invocation $\mathcal{A}_0$ ends up coloring the graph with colors from the palette $\mathcal{P}_0 = \{1, 2, \ldots, c \cdot (A_0)^2\}$, the invocation $\mathcal{A}_1$ employs the palette $\mathcal{P}_1 = \{|\mathcal{P}_0| + 1, |\mathcal{P}_0| + 2, \ldots, |\mathcal{P}_0| + c \cdot (A_1)^2\}$, etc.

Observe that for indices i such that $A_i \geq A$, the invocation $\mathcal{A}_i$ succeeds to construct a legal $(c \cdot (A_i)^2)$-coloring of the graph in time $O(\log n)$. For an index i such that $A_i < A$, there might be a situation that there is no admissible color for some vertex v in the invocation $\mathcal{A}_i$. To handle such situations we add to the algorithm a "failure" rule. Specifically, if a vertex v recognizes that it has no admissible color in an invocation $\mathcal{A}_i$, for some i, it selects an arbitrary color γ from the palette $\mathcal{P}_i$, and sets its color $\varphi_i(v)$ in the invocation $\mathcal{A}_i$ to be equal to γ.

Finally, once all the invocations have terminated, and a vertex v and all its neighbors have selected their colors in each invocation, v selects its final color $\varphi(v)$ as follows. $\varphi(v)$ is equal to $\varphi_m(v)$ for the smallest index m of an invocation $\mathcal{A}_m$ in which for every neighbor u of v in G, $\varphi_m(v) \neq \varphi_m(u)$ holds. This completes the description of the algorithm. The index $m = m(v)$ as above will be henceforth referred to as the *index* of v.

Next, we show that the algorithm is correct.

Lemma 5.7 *The coloring φ is legal.*

Proof The invocation with index $j = \lceil \log n \rceil$ satisfies $2^j \geq n \geq a(G)$. Hence the coloring φ_j that it produces is legal. Consequently, for each vertex v there is an invocation with index $m \leq j$ such that $\varphi_m(v) \neq \varphi_m(u)$, for each neighbor u of v.

Consider a pair $u, w \in V$ of neighboring vertices. Let m (respectively, m') be the index such that $\varphi(u) = \varphi_m(u)$ (resp., $\varphi(w) = \varphi_{m'}(w)$). First, consider the case that $m = m'$. It follows that, $\varphi_m(u) \neq \varphi_m(w)$ because the invocation of index $m = m(u)$ satisfies $\varphi_m(u) \neq \varphi_m(z)$ for every neighbor z of u. Hence $\varphi(u) \neq \varphi(w)$. Suppose now that $m \neq m'$. However, then $\varphi(u) = \varphi_m(u) \in \mathcal{P}_m$, $\varphi(w) = \varphi_{m'}(w) \in \mathcal{P}_{m'}$, and the palettes $\mathcal{P}_m$ and $\mathcal{P}_{m'}$ are disjoint. Thus, $\varphi(u) \neq \varphi(w)$, completing the proof. □

Observe that for for every vertex v, the index $m = m(v) \leq \log(2A)$. It follows that the overall number of colors used by the algorithm is at most

$$\sum_{i=0}^{\lfloor \log(2A) \rfloor} c \cdot (2^i)^2 = O(A^2) = O(a^2). \tag{1}$$

The running time is $O(\log n)$. Note, however, that running $\lceil \log n \rceil + 1$ invocations $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\lceil \log n \rceil}$ in parallel requires sending $\lceil \log n \rceil + 1$ messages of size $O(\log n)$ each on each round, over each edge of the network. In other words, the messages sent by this algorithm may be of size $O(\log^2 n)$.

Alternatively, one may opt to execute the $\lceil \log n \rceil + 1$ invocations $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\lceil \log n \rceil}$ one after another instead of executing them in parallel. Since each execution requires $O(\log^* n)$ time, the overall running time increases only slightly to $O(\log n \cdot \log^* n)$.

We summarize this argument in the following theorem.

Theorem 5.8 *Procedure Arb-Linial is applicable in the scenario when the vertices do not know the arboricity a of the graph before the algorithm starts. In this scenario the procedure can **either** compute an $O(a^2)$-coloring of the graph in time $O(\log n)$ using messages of size $O(\log^2 n)$ **or** compute an $O(a^2)$-coloring in time $O(\log n \cdot \log^* n)$ using messages of size $O(\log n)$.*

The algorithm extends also in a straightforward way to the scenario in which instead of the number of vertices n, the vertices are provided with a polynomial estimate N of n.

Similarly, Procedure Tradeoff-Arb-Linial can also be adapted to the scenario when the arboricity a is unknown to the vertices before the computation starts, but n is known. If we allow messages of size $O(\log^2 n)$, by the same considerations as with Procedure Arb-Linial, Procedure Tradeoff-Arb-Linial computes $O(a^2 \cdot q^2)$-coloring in time $O(\frac{\log n}{\log q})$. However, if message size is restricted to $O(\log n)$, then sequential executions $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\lceil \log n \rceil}$ require $O(\log n \cdot \log^* n)$ time. Consequently, a straight-forward adaptation of Procedure Tradeoff-Arb-Linial to this scenario results in $O(a^2 \cdot q^2)$-coloring within time $O(\log n \cdot \log^* n)$. This, however, is inferior to the results in Theorem 5.8.

To provide a meaningful tradeoff for this scenario too, we modify the algorithm in the following way. Set $\delta > 0$ to be an arbitrarily small real constant, and let $t = \log^{\delta/2} n$ be a parameter. Set $A'_i = t^i$, for $i = 0, 1, \ldots, \left\lceil \frac{\log n}{\log t} \right\rceil + 1$. (This is instead of setting $A_i = 2^i$.) Let $\mathcal{A}'_i$ be an invocation of Procedure Tradeoff-Arb-Linial with the arboricity parameter $A'_i, i = 0, 1, \ldots, \left\lceil \frac{\log n}{\log t} \right\rceil + 1$. Invocations $\mathcal{A}'_i$ are now executed one after another, and require overall $O(\frac{\log n}{\log \log n} \cdot \log^* n)$ time. The computation of forests-decomposition requires an additional $O(\frac{\log n}{\log q})$ time, and thus the total running time of Procedure Tradeoff-Arb-Linial in this scenario is $O(\frac{\log n}{\log \log n} \cdot \log^* n + \frac{\log n}{\log q})$.

The overall number of colors used by the algorithm is (see (1)) at most $C = \sum_{i=0}^{\lfloor \log_t (t \cdot A) \rfloor} c(t^i)^2$. Since $A = a \cdot (2 + q)$, it follows that $C = O((t \cdot A)^2) = O(a^2 \cdot q^2 \cdot \log^{\delta} n)$. We summarize the properties of Procedure Tradeoff-Arb-Linial in this scenario by the following theorem.

Theorem 5.9 *Let $q > 2$, $\delta > 0$, be a pair of positive real numbers. In the scenario when the vertices do not know the arboricity a of the graph before the algorithm starts, Procedure Tradeoff-Arb-Linial invoked with the parameter q can **either** compute an $O(a^2 \cdot q^2)$-coloring of the graph in time $O(\frac{\log n}{\log q} + \log^* n)$ using messages of size $O(\log^2 n)$, **or** compute an $O(a^2 \cdot q^2 \cdot \log^{\delta} n)$-coloring in time $O(\frac{\log n}{\log \log n} \cdot \log^* n + \frac{\log n}{\log q})$ using messages of size $O(\log n)$.*

6 MIS algorithms

6.1 Known arboricity

In this section we capitalize on the results of Sect. 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 (see, e.g., [23], chapter 7), described below for the sake of completeness.

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 [23] 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 $U_1, U_2, \ldots, U_p$ be the disjoint color classes, with all vertices of U_i being colored by i, for $i = 1, 2, \ldots, p$. Initialize the independent set W as U_1. The reduction algorithm proceeds iteratively, taking care of one of the color classes $U_2, U_3, \ldots, U_p$ on each of the $p-1$ iterations. For iteration $i = 1, 2, \ldots, p-1$, before the iteration i starts, an independent set $W \subseteq \cup_{j=1}^{i} U_j$ is maintained. In 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 proof can be found in [23, Chap. 7].)

Lemma 6.2 *W is a maximal independent set.*

The next theorem follows directly from Corollary 6.1.

Theorem 6.3 *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 $t = \sqrt{\log n}$.

Theorem 6.4 *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 $O(a \cdot \sqrt{\log n} + a \log a)$.*

In particular, Theorem 6.4 implies that an MIS can be computed deterministically in polylogarithmic time on graphs with polylogarithmic arboricity.

6.2 General scenarios

Consider the scenario that vertices know neither a nor an upper bound a' on a, but we are allowed to use messages of size $O(\log^2 n)$. In this case, as we have shown in Sect. 5, Procedure Tradeoff-Arb-Linial is applicable, and it provides the same tradeoffs up to constant factors. Hence the arguments presented in this section are applicable as well. Consequently, Theorem 6.4 extends to the scenario when the vertices do not know the value of a, but are allowed to use messages of size $O(\log^2 n)$.

Finally, Theorem 5.9 provides us with only slightly inferior tradeoff for the case that vertices do know the arboricity and message size is limited to $O(\log n)$. An analogous calculation shows that in this case, as long as $a = \log^{1/2-\delta} n$ (for an arbitrarily small $\delta > 0$) the algorithm computes an MIS in time $O(\frac{\log n}{\log\log n} \cdot \log^* n)$.

7 Lower bounds

In this section we build upon a fundamental result of Linial [18] and show nearly tight lower bounds on the running time required for coloring and for computing a forests-decomposition. For graphs of constant arboricity our upper and lower bounds match up to constant factors.

Theorem 7.1 [18] *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 of radius r which has running time at most $\frac{2}{3}r$ uses at least $\frac{1}{2}\sqrt{d}$ colors. In other words, any algorithm that uses less than $\frac{1}{2}\sqrt{d}$ colors has running time greater than $\frac{2}{3}r$.*

Observe that $r \geq \frac{1}{2}\frac{\log n}{\log(d-1)}$. 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. By Theorem 7.1, q-coloring the $5q^2$-regular tree requires at least $\frac{2}{3}r = \Omega(\frac{\log n}{\log q})$ rounds. Since a $5q^2$-regular tree is a planar graph, it follows that $t(q) = \Omega(\frac{\log n}{\log q})$. We conclude that q-coloring planar graphs requires $\Omega(\frac{\log n}{\log q})$ time, which matches our upper bound up to constant factors, as long as $q = O(n^{1/\log^* n})$. (See Corollary 5.6.)

For a positive integer parameter $a = o(n^{1/4})$ it is possible to construct an n-vertex graph with arboricity a. Let G be such a graph. Let H be a $(a^4 \cdot q^2)$-regular tree with $\Theta(n)$ vertices, for some parameter $q = o(\sqrt{n}/a^2)$. Let J be the graph obtained by connecting the root of H with one of the vertices of G. Since $O(a^2 \cdot q)$-coloring of H requires

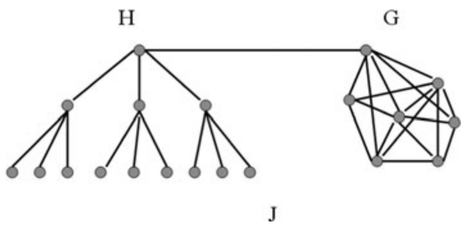

Fig. 2 The graph J is constructed by connecting the two graphs G and H

$\Omega(\log_{a^2 q} n) = \Omega(\frac{\log n}{\log q + \log a})$ time (by Theorem 7.1), and H is an induced subgraph of J, this lower bound applies also for $O(a^2 \cdot q)$-coloring of J. Since the arboricity of J is equal to the arboricity of G, the next result follows.

See Fig. 2 for an illustration.

Corollary 7.2 *For $a = o(n^{1/4})$, and a parameter $q = o(\sqrt{n}/a^2)$, $O(a^2 \cdot q)$-coloring n-vertex graphs with arboricity a requires $\Omega(\frac{\log n}{\log a + \log q})$ time.*

Next, we turn to lower bounds for the problem of computing forests-decompositions. For a parameter q, $q \geq 1$, let $t(q)$ be the best possible running time of a correct algorithm for computing an $O(q \cdot a)$-forests-decomposition on graphs with a fixed arboricity a. Given an $O(q \cdot a)$-forests-decomposition, step 2 of Procedure Arb-Linial computes an $O(q^2 \cdot a^2)$-coloring within additional time $O(\log^* n)$. Hence the resulting algorithm $\mathcal{A}$ computes an $O(q^2 \cdot a^2)$-coloring within time $t(q) + O(\log^* n)$.

Let d be a positive integer value to be determined later, and consider an execution of the algorithm $\mathcal{A}$ on a a graph obtained from connecting the d-regular n-vertex tree with an arbitrary $\Theta(n)$ vertex graph of arboricity a, as earlier. By Theorem 7.1, if $t(q) + O(\log^* n) \leq \frac{1}{3} \frac{\log n}{\log(d-1)}$, then $O(q^2 \cdot a^2) \geq \frac{1}{2}\sqrt{d}$.

Set $d = c \cdot q^4 \cdot a^4$, for a sufficiently large constant c so that the inequality $O(q^2 \cdot a^2) \geq \frac{1}{2}\sqrt{d}$ will not hold. Then $t(q) = \Omega(\frac{\log n}{\log d}) - O(\log^* n) = \Omega(\frac{\log n}{\log q + \log a}) - O(\log^* n)$.

Theorem 7.3 *For $a = o(n^{1/4})$, and a parameter q, $q = o(n^{1/4}/a)$, computing an $O(q \cdot a)$-forests- decomposition for n-vertex graph with arboricity a requires $\Omega(\frac{\log n}{\log q + \log a}) - O(\log^* n)$ time.*

8 Conclusions and open questions

In this paper we have presented efficient deterministic MIS and coloring algorithms for the family of graphs with arboricity at most 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 $o(\sqrt{\log n})$. It would be interesting to extend this result to graphs with arboricity $o(\log n)$.

The lower bounds that we have presented are tight for the problems of $O(a)$-forests-decomposition and $O(a^2)$-coloring. However, it might be possible to improve the $O(a)$-coloring algorithm. This appears to be a challenging task. The lower bound $\Omega(\sqrt{\frac{\log n}{\log \log n}})$ of [15] for the MIS problem is applicable for general graphs. However, on sparse graphs it might be possible to develop more efficient algorithms. Currently, even on unoriented trees the best known algorithm has running time $O(\frac{\log n}{\log \log n})$ (shown in this paper). Finding better solution or a strong lower bound would help to classify the complexity of the distributed MIS problem.

Also, we have shown that one can color graphs with bounded arboricity in $n^{o(1)}$ colors in just $O(\log^* n)$ time. Improving the bound on the number of colors is an interesting venue for research.

Finally, we have shown that the problem of computing an $O(a)$-coloring on sparse graphs is harder than computing an MIS on such graphs. Our lower bound for computing an $O(a)$-coloring is logarithmic, while the MIS algorithm has sublogarithmic running time. This fact is quite surprising, since there is no evidence for such a separation between the $(\Delta + 1)$-coloring and the MIS problems.

References

1. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms **7**(4), 567–583 (1986)
2. Arikati, S.R., Maheshwari, A., Zaroliagis, C.: Efficient computation of implicit representations of sparse graphs. Discrete Appl. Math. **78**(1–3), 1–16 (1997)
3. Awerbuch, B., Goldberg, A.V., Luby, M., Plotkin, S.: Network decomposition and locality in distributed computation. In: Proceedings of the 30th Symposium on Foundations of Computer Science, pp. 364–369 (1989)
4. Bollobas, B.: Extremal Graph Theory. Academic Press, New York (1978)
5. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. Control **70**(1), 32–53 (1986)
6. Deo, N., Litow, B.: A structural approach to graph compression. In: Proceedings of the MFCS Workshop on Communications, pp. 91–100 (1998)
7. Dujmovic, V., Wood, D.R.: Graph treewidth and geometric thickness parameters. Discrete Comput. Geom. **37**(4), 641–670 (2007)
8. Erdős, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of r others. Isr. J. Math. **51**, 79–89 (1985)
9. Gfeller, B., Vicari, E.: A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In: Proceedings of the 26th ACM Symposium on Principles of Distributed Computing, pp. 53–60 (2007)
10. Goldberg, A., Plotkin, S.: Efficient parallel algorithms for $(\Delta + 1)$-coloring and maximal independent set problem. In: Proceedings of

the 19th ACM Symposium on Theory of Computing, pp. 315–324 (1987)

11. Goldberg, A., Plotkin, S., Shannon, G.: Parallel symmetry-breaking in sparse graphs. SIAM J. Discrete Math. **1**(4), 434–446 (1988)
12. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Inf. Comput. **88**(1), 60–87 (1990)
13. Kothapalli, K., Scheideler, C., Onus, M., Schindelhauer, C.: Distributed coloring in $\tilde{O}(\sqrt{\log n})$ bit rounds. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium, p. 24 (2006)
14. Kuhn, F., Moscibroda, T., Nieberg, T., Wattenhofer, R.: Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: Proceedings of the 19th International Symposium on Distributed Computing, pp. 273–287 (2005)
15. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing, pp. 300–309 (2004)
16. Kuhn, F., Moscibroda, T., Wattenhofer, R.: On the locality of bounded growth. In: Proceedings of the 24rd ACM Symposium on Principles of Distributed Computing, pp. 60–68 (2005)
17. Kuhn, F., Wattenhofer, R.: On the complexity of distributed graph coloring. In: Proceedings of the 25th ACM Symposium on Principles of Distributed Computing, pp. 7–15 (2006)
18. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
19. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**, 1036–1053 (1986)
20. Mohar, B.: Graph minors and graphs on surfaces. In: Hirschfeld, J.W.P. (ed.) Surveys in Combinatorics. London Mathematical Society Lecture Note Series 288. Cambridge University Press, Cambridge, pp. 145–163 (2001)
21. Nash-Williams, C.: Decompositions of finite graphs into forests. J. Lond. Math. **39**, 12 (1964)
22. Panconesi, A., Srinivasan, A.: On the complexity of distributed network decomposition. J. Algorithms **20**(2), 581–592 (1995)
23. Peleg, D.: Distributed computing: a locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications (2000)
24. Schneider, J., Wattenhofer, R.: A log-star distributed maximal independent set algorithm for growth bounded graphs. In: Proceedings of the 27th ACM Symposium on Principles of Distributed Computing, pp. 35–44 (2008)
25. Singh, G.: Efficient leader election using sense of direction. Distrib. Comput. **10**(3), 159–165 (1997)
26. Szegedy, M., Vishwanathan, S.: Locality based graph coloring. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 201–207 (1993)