MIS on Trees

Christoph Lenzen
Computer Engineering
and Networks Laboratory
ETH Zurich
Switzerland
lenzen@tik.ee.ethz.ch

Roger Wattenhofer
Computer Engineering
and Networks Laboratory
ETH Zurich
Switzerland
wattenhofer@tik.ee.ethz.ch

ABSTRACT

A maximal independent set on a graph is an inclusion-maximal set of mutually non-adjacent nodes. This basic symmetry breaking structure is vital for many distributed algorithms, which by now has been fueling the search for fast local algorithms to find such sets over several decades. In this paper, we present a solution with randomized running time $\mathcal{O}(\sqrt{\log n \log\log n})$ on trees, improving roughly quadratically on the state-of-the-art bound. Our algorithm is uniform and nodes need to exchange merely $\mathcal{O}(\log n)$ many bits with high probability. In contrast to previous techniques achieving sublogarithmic running times, our approach does not rely on any bound on the number of independent neighbors (possibly with regard to an orientation of the edges).

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms, trees*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms

Algorithms, Theory

Keywords

symmetry breaking, optimal bit complexity, maximal independent set, asymptotic analysis

1. INTRODUCTION & RELATED WORK

In graph theory, two nodes are independent if they are not neighbors. A set of nodes is independent if the nodes in the set are pairwise independent. And a maximal independent set (MIS) is an independent set that is not a proper subset of any other independent set; in other words, an MIS cannot be extended. Finding an MIS is a most basic form of *symmetry breaking*, and as such widely used as a building block in distributed or parallel algorithms.

PODC'11, June 6–8, 2011, San Jose, California, USA.

Not surprisingly, obtaining an MIS quickly is one of the fundamental questions in distributed and parallel computing. More surprisingly, despite all the research, the state-of-the-art algorithm was discovered in the 1980s. Several research groups [1, 6, 10] more or less concurrently presented the same simple randomized *marking* algorithm that finds an MIS in time $\mathcal{O}(\log n)$ *with high probability*[1] (w.h.p.) Deterministic algorithms tend to be much slower; Panconesi and Srinivasan [13] presented an algorithm with running time $2^{\mathcal{O}(\sqrt{\log n})}$, based on a network decomposition. In contrast, the strongest lower bound proves that $\Omega(\sqrt{\log n})$ time is required, even for randomized algorithms on line graphs [8]. Arguably, reducing this complexity gap is one of the most important open problems in distributed computing.

In the last 25 years, a lot of work went into understanding special graph classes. One line of work shows that the MIS problem can be solved in time $\Theta(\log^* n)$ on rooted trees [4], graphs of bounded degree [5], and bounded-growth graphs [14]. The upper bounds are deterministic and the matching lower bound [9] can be extended to randomized algorithms [12]. Another common technique is to compute a coloring first and subsequently gradually augment an independent set by concurrently adding all feasible nodes of a given color. Using deterministic $(\Delta+1)$-coloring algorithms with running time (essentially) linear in the maximum degree Δ [2, 7], this reduction yields $\mathcal{O}(\Delta + \log^* n)$-time solutions suitable for small-degree graphs. However, all these results work on graphs with restricted degrees, sometimes explicitly, sometimes implicitly. In rooted trees, for instance, each node just deals with one single neighbor (the parent) and in bounded-growth graphs the number of independent neighbors is bounded.

In other words, finding an MIS seems to be difficult mostly in graphs with an unbounded number of independent neighbors. Until recently, even on (non-rooted) trees no better algorithm than the randomized marking algorithms from the 1980s was known. This was changed by Barenboim and Elkin [3], who devised a deterministic algorithm with running time $o(\log n)$ for graphs with arboricity[2] $o(\sqrt{\log n})$. Their algorithm first computes a forest decomposition and subsequently efficiently colors the graph. Again, the key ingredient of the coloring step is that nodes can restrict their attention to the small number of parents they have in the de-

[1]That is, with probability $1-1/n^c$ for an arbitrary, but fixed constant c.

[2]A forest decomposition is a partitioning of the edge set into rooted forests. The arboricity of a graph is the minimum number of forests in a forest decomposition.

composition. However, even on graphs of constant arboricity (in particular trees) the improvement is marginal, i.e., their algorithm has a time complexity of $\Theta(\log n/\log\log n)$. An accompanying lower bound shows that even randomized algorithms require $\Omega(\log n/\log f)$ rounds to compute a forest decomposition into f forests, while the computed coloring uses $\Omega(f)$ colors. Hence, their technique is limited to a factor $\mathcal{O}(\log\log n)$ gain in time complexity. What is more, their algorithm is non-uniform, as it needs an upper bound of $\mathcal{O}(A)$ on the arboricity A of the graph or a polynomial upper bound on the number of nodes as input.

In this work, we present a much faster, uniform algorithm running on arbitrary forests. Our approach guarantees termination within $\mathcal{O}(\sqrt{\log n \log\log n})$ rounds w.h.p. This is achieved by fusing several techniques into a single algorithm and devising a novel analysis. Initially, we employ a recent variation by Métivier et al. [11] of the traditional randomized marking algorithm. The authors show how their algorithm can be implemented with optimal bit complexity of $\mathcal{O}(\log n)$ w.h.p. This is not an inherent property of their approach, as also the classic algorithms can be adapted to exhibit the same bit complexity; we demonstrate how to obtain the same bound for our technique. However, their algorithm appeals by its simplistic elegance: In each phase, each eligible node picks a random value and joins the independent set if its value is a local maximum. As a positive side effect of this subroutine, our algorithm will also succeed to obtain an MIS on arbitrary graphs within $\mathcal{O}(\log n)$ rounds w.h.p. However, we make use of the technique with a different goal. The logarithmic time complexity of the randomized marking algorithm follows from the argument that in expectation a constant fraction of the edges is removed in each phase. Thus, by itself, the algorithm might also on trees exhibit a logarithmic running time. The key observation we make is that on trees—up to maybe a few exceptions—nodes of high degree will not survive more than $\mathcal{O}(\sqrt{\log n \log\log n})$ phases of this algorithm. Moreover, (in essence) it can be shown that on trees the maximum node degree falls exponentially until it becomes $\mathcal{O}(\sqrt{\log n})$. From this point on, we utilize deterministic coloring [2, 7] to clear most of the remaining subgraph consisting of nodes that are still eligible to join the independent set. Afterwards, the diameter of connected components will be sufficiently small such that iteratively removing isolated nodes and leaves will complete the task quickly. Although the presented proof is tailored to trees, the main ingredient is that in trees probabilistic dependencies are scarce. Therefore, one might hope that our result can be extended to more general sparse graph classes, e.g. graphs of bounded arboricity.

2. MODEL & PRELIMINARIES

We employ the standard synchronous message passing model of distributed computation. The (non-faulty) system is modelled as a simple graph $G = (V, E)$, where nodes represent computational devices and edges represent bidirectional communication links. In each synchronous round, nodes may perform arbitrary finite local computations, and send (receive) a message to (from) each neighbor. The set of neighbors of a node $v \in V$ is denoted by $\mathcal{N}_v := \{w \in V \mid \{v, w\} \in E\}$. The *degree* of $v \in V$ is $\delta_v := |\mathcal{N}_v|$. Initially, each node knows its neighbors and the (local part) of the input of the problem. For deterministic MIS algorithms, the latter is typically an initial coloring of polynomially many (in $n := |V|$) colors. If randomization is permitted, nodes have access to an infinite source of unbiased and independent random bits. In this case, an initial coloring can be generated w.h.p. (without any communication) and verified in one round by exchanging the colors between neighbors. The *time* and *bit complexity* of an algorithm are the maximum number of rounds any node requires to terminate and the maximum number of bits sent over any edge, respectively.

Throughout our analysis, we will make frequent use of Chernoff type bounds. For reference, we state here the variants we use.

THEOREM 2.1. *For N independent 0–1 random variables $X_1, \ldots, X_N$, define $X := \sum_{i=1}^{N} X_i$. Then*

(i) $X \in E[X] + \mathcal{O}\left(\log n + \sqrt{E[X]\log n}\right)$ *w.h.p.*

(ii) $E[X] \in \mathcal{O}\left(\frac{1}{\sqrt{\log n}}\right) \Rightarrow X \in \mathcal{O}\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ *w.h.p.*

(iii) $P[X = 0] \leq e^{-E[X]/2}$

(iv) $E[X] \geq 8c\log n \Rightarrow X \in \Theta(E[X])$ *w.h.p.*

(v) $E[X] \in \omega(\log n) \Rightarrow X \in (1 \pm o(1))E[X]$ *w.h.p.*

3. ALGORITHM

In this section, we introduce our MIS algorithm for trees. For the sake of simplicity, we (*i*) present a non-uniform variant, (*ii*) assume that the algorithm makes use of uniformly random real numbers, and (*iii*) use a generic term of $\Theta(R)$ in the description of the algorithm. In Theorem 4.10, we will show how to remove the first two assumptions, and it will turn out that for the uniform algorithm the precise choice of the constants in the term $\Theta(R)$ is (up to a constant factor) irrelevant for the running time of the algorithm.

The algorithm seeks to perpetually increase the number of nodes in the independent set I until it finally is maximal. Whenever a node enters I, its inclusive neighborhood is removed from the graph and the algorithm proceeds on the remaining subgraph of G. It consists of three main steps, each of which employs a different technique to add nodes to I. It takes a single parameter R, which ideally is small enough to guarantee a small running time of the first two loops of the algorithm, but large enough to guarantee that the residual nodes can be dealt with quickly by the final loop of the algorithm.

We proceed by describing the parts of the algorithm in detail, whose pseudocode is given in Algorithm 1. In the first part, the following procedure is repeated $\Theta(R)$ times. Each active node draws uniformly and independently at random (u.i.r.) a number from $[0, 1] \subset \mathbb{R}$ and joins I if its value is a local maximum.[3] This and similar techniques have been known for long and ensure to reduce the number of edges in the graph exponentially w.h.p.; however, in our case we have the different goal of reducing the maximum degree in the graph rapidly. Once degrees become small, we cannot guarantee a quick decay of degrees w.h.p. anymore, therefore the employed strategy is changed.

[3]A random real number from $[0, 1]$ can be interpreted as infinite string of bits of decreasing significance. As nodes merely need to know which one of two values is larger, it is sufficient to generate and compare random bits until the first difference occurs.

Algorithm 1: Fast MIS on Trees.

```
input  : R ∈ ℕ
output: maximal independent set I
I := ∅
for i ∈ {1,...,Θ(R)} do // reduce degrees
    for v ∈ V in parallel do
        r_v := u.i.r. number from [0,1]
        if r_v > max_{w∈N_v}{r_w} then
            I := I ∪ {v}
            delete N_v ∪ {v} from G
for i ∈ {1,2} do // remove nodes of small degree
    H := subgraph of G induced by nodes of degree
    δ_v ≤ R
    (R+1)-color H
    for c ∈ {1,...,R+1} do
        for v ∈ V with color c in parallel do
            I := I ∪ {v}
            delete N_v ∪ {v} from G
while V ≠ ∅ do // clean up
    for v ∈ V in parallel do
        if δ_v = 0 then // remove isolated nodes
            I := I ∪ {v}
            delete N_v ∪ {v} from G
        if δ_v = 1 then // remove leaves
            {w} := N_v
            if δ_w ≠ 1 then // true leaf
                I := I ∪ {v}
                delete N_v ∪ {v} from G
            else // pair of degree-1 nodes
                r_v := u.i.r. number from [0,1]
                if r_v > r_w then
                    I := I ∪ {v}
                    delete N_v ∪ {v} from G
```

Consequently, the second part of the algorithm aims at dealing with small-degree nodes by means of a deterministic scheme. Though it might be the case that not all nodes of degree larger than R could be removed during the first loop, we will show that if R is large enough, most nodes will not have more than R neighbors of degree larger than R in their neighborhood. Thus, removing all nodes of degree at most R for *two* times will thin out the graph considerably. To this end, we first $(R+1)$-color the subgraph induced by all nodes of degree at most R and then iterate through the colors, adding all nodes sharing color c concurrently to I.

Yet, a small fraction of the nodes may still remain in the graph. In order to deal with these nodes, we repeat the step of removing all leaves and isolated nodes from the forest until all nodes have terminated.

As evident from the description of the algorithm, iterations of the first and third loop can be implemented within a constant number of synchronous distributed rounds. The second loop requires $\mathcal{O}(R)$ time plus the number of rounds needed to color the respective subgraph; for this problem deterministic distributed algorithms taking $\mathcal{O}(R + \log^* n)$ time are known [2, 7]. In Theorem 4.9 we will show that for some $R \in \mathcal{O}(\sqrt{\log n \log\log n})$, the third loop of the algorithm will complete in $\mathcal{O}(R)$ rounds w.h.p. Thus, for an appropriate value of R, the algorithm computes an MIS within $\mathcal{O}(\sqrt{\log n \log\log n})$ rounds.

4. ANALYSIS

For the purpose of our analysis, we will make use of the notion of a *rooted* tree. A tree becomes rooted by choosing a node $r \in V$ as root. The parent $p \in \mathcal{N}_v$ of a node $V \setminus \{r\}$ then is its neighbor which is closer to the root, while its children $\mathcal{C}_v := \mathcal{N}_v \setminus \{p\}$ are the remaining neighbors. In all lemmas, w.l.o.g. we take for granted that G is a rooted tree. Note that this assumption is introduced to simplify the presentation. To execute the algorithm, nodes do not require knowledge of an orientation of the edges and the proof trivially generalizes to forests.

The lion's share of the argumentation will focus on the first loop of Algorithm 1. We will call an iteration of this loop a *phase*. By $\delta_v(i)$ we denote the degree of node v at the beginning of phase i in the subgraph of G induced by the nodes that have not been deleted yet; similarly, $\mathcal{N}_v(i)$ and $\mathcal{C}_v(i)$ are the sets of neighbors and children of v still active at the beginning of phase i, respectively.

We start our analysis with the observation that, in any phase, a high-degree node without many high-degree children is likely to be deleted in that phase, *independently* of the behaviour of its parent.

LEMMA 4.1. *If at the beginning of phase i it holds for a node v that half of its children have a degree of at most $\delta_v(i)/16 \ln \delta_v(i)$, then v is deleted with probability at least $1 - 5/\delta_v(i)$ in that phase, independently of the random number of its parent.*

PROOF. Observe that the probability that v survives phase i is increasing in the degree $\delta_w(i)$ of any child $w \in \mathcal{C}_v(i)$ of v. Thus, w.l.o.g., we may assume that all children of v of degree at most $\delta := \delta_v(i)/16 \ln \delta_v(i)$ have exactly that degree.

Consider such a child w. With probability $1/\delta$, its random value $r_w(i)$ is larger than all its children's. Denote by X the random variable counting the number of children $w \in \mathcal{C}_v(i)$ of degree δ satisfying that

$$\forall u \in \mathcal{C}_w(i) : \; r_w(i) > r_u(i). \tag{1}$$

Thus, for the random variable X counting the number of such nodes it holds that

$$E[X] = \sum_{\substack{w \in \mathcal{C}_v(i) \\ \delta_w(i) = \delta}} \frac{1}{\delta} \geq 8 \ln \delta_v(i).$$

Since the random choices are independent, applying Chernoff's bound yields that

$$P\left[X < \frac{E[X]}{2}\right] < e^{-E[X]/8} \leq \frac{1}{\delta_v(i)}.$$

Node v is removed unless the event $\mathcal{E}$ that $r_v(i) < r_w(i)$ for all the children $w \in \mathcal{C}_v(i)$ of degree δ satisfying (1) occurs. If $\mathcal{E}$ happens, this implies that $r_v(i)$ is also smaller than all random values of children of such w, i.e., $r_v(i)$ is smaller than $\delta\, E[X]/2 \geq \delta_v(i)/4$ other independent random values. Since the event that $X \geq E[X]/2$ depends only on the order of the involved random values, we infer that $P[\mathcal{E} \mid X \geq E[X]/2] < 4/\delta_v(i)$. We conclude that v is deleted with probability at

least

$$\begin{aligned} & P\left[X \geq \frac{E[X]}{2}\right] P\left[\bar{\mathcal{E}} \,\middle|\, X \geq \frac{E[X]}{2}\right] \\ & > \left(1 - \frac{1}{\delta_v(i)}\right)\left(1 - \frac{4}{\delta_v(i)}\right) \\ & > 1 - \frac{5}{\delta_v(i)} \end{aligned}$$

as claimed. Since we reasoned about whether children of v join the independent set only, this bound is independent of the behaviour of v's parent. □

Applied inductively, this result implies that in order to maintain a high degree for a considerable number of phases, a node must be the root of a large subtree. This concept is formalized by the following definition.

DEFINITION 4.2 (DELAY TREES). *A* delay tree of depth $d \in \mathbb{N}_0$ *rooted at node v is defined recursively as follows. For $d = 0$, the tree consists of v only. For $d > 0$, node v satisfies at least one of the following criteria:*

(i) At least $\delta_v(d+1)/4$ children $w \in \mathcal{C}_v$ are roots of delay trees of depth d with $\delta_w(d) \geq \delta_v(d+1)/16 \ln \delta_v(d+1)$.

(ii) Node v is the root of a delay tree of depth $d-1$ and it holds that $\delta_v(d) \geq \delta_v(d+1)^2/(81 \ln \delta_v(d+1))$.

In order to bound the number of phases for which a node can have a significant chance to retain a large degree, we bound the depth of delay trees rooted at high-degree nodes.

LEMMA 4.3. *Assume that $R \geq 2\sqrt{\ln n \ln\ln n}$ and also that $\delta_v(d) \geq e^R$. Then for a delay tree of depth $d-1$ rooted at v it holds that $d \in \mathcal{O}(\sqrt{\log n / \log\log n})$.*

PROOF. Assume w.l.o.g. that $d > 1$. Denote by $s_i(\delta)$, where $i \in \{0, \ldots, d-1\}$ and $\delta \in \mathbb{N}$, the minimal number of leaves in a delay tree of depth i rooted at some node w satisfying $\delta_w(i+1) = \delta$.

We claim that for any δ and $i \leq \ln \delta/(2\ln(81 \ln \delta))$, it holds that

$$s_i(\delta) \geq \prod_{j=1}^{i} \frac{\delta}{(81 \ln \delta)^{j-1}},$$

which we will show by induction. This is trivially true for $i = 1$, hence we need to perform the induction step only.

Observe that because any node is a delay tree of depth zero, the number of leaves in a delay tree of depth one equals the degree of the root. Moreover, as δ is sufficiently large, we have for any $x' \geq \delta/(81 \ln \delta)^i$ that the derivative of the function $x/(81 \ln x)$ at x' is at least one. Hence, for any $C \geq 1$, we have that

$$s_i(C\delta') \geq C s_i(\delta')$$

because the minimal number of leaves s_i in the tree must grow at least linearly in the argument no matter which of the two possible conditions in the recursive definition of a delay tree is satisfied.

Consequently, for any $i \in \{2, \ldots, \lfloor \ln \delta/(2\ln(81 \ln \delta))\rfloor\}$, the assumption that the claim is true for $i-1$ and the recursive definition of delay trees yield that

$$\begin{aligned} s_i(\delta) & \geq \min\left\{\frac{\delta}{4} s_{i-1}\left(\frac{\delta}{16 \ln \delta}\right), s_{i-1}\left(\frac{\delta^2}{81 \ln \delta}\right)\right\} \\ & \geq \delta\, s_{i-1}\left(\frac{\delta}{81 \ln \delta}\right) \\ & > \delta \prod_{j=1}^{i-1} \frac{\delta}{(81 \ln \delta)^j} \\ & = \prod_{j=1}^{i} \frac{\delta}{(81 \ln \delta)^{j-1}}. \end{aligned}$$

Thus the induction step succeeds, showing the claim.

Now assume that v is the root of a delay tree of depth $d-1$. As $\delta_v(d) \geq e^R$, we may insert any $i \in \{1, \ldots, \min\{d-1, \lfloor R/(2\ln 81R)\rfloor\}\}$ into the previous claim. Supposing for contradiction that $d-1 \geq \lfloor R/(2\ln 81R)\rfloor$, it follows that the graph contains at least

$$\begin{aligned} & \prod_{j=1}^{\lfloor R/(2\ln 81R)\rfloor} \frac{e^R}{(81R)^{j-1}} \\ & > \prod_{j=1}^{\lfloor R/(2\ln 81R)\rfloor} e^{R/2} \\ & \in e^{R^2/((4+o(1))\ln R)} \\ & \subseteq n^{2-o(1)} \end{aligned}$$

nodes. On the other hand, if $d-1 < \lfloor R/(2\ln 81R)\rfloor$, we get that the graph contains at least $e^{(d-1)R/2}$ nodes, implying that $d \in \mathcal{O}(\sqrt{\ln n/\ln\ln n})$ as claimed. □

With this statement at hand, we infer that for some $R \in \Theta(\sqrt{\log n \log\log n})$, it is unlikely that a node has degree e^R or larger for R phases.

LEMMA 4.4. *Suppose that $R \geq 2\sqrt{\ln n \ln\ln n}$. Then, for any node $v \in V$ and some number $r \in \mathcal{O}(\sqrt{\log n/\log\log n})$, it holds with probability at least $1 - 6e^{-R}$ that $\delta_v(r+1) < e^R$. This statement holds independently of the behaviour of v's parent.*

PROOF. Assume that $\delta_v(r) \geq e^R$. According to Lemma 4.1, node v is removed with probability at least $1 - 5/\delta_v(r)$ in phase r unless half of its children have at least degree $\delta_v(r)/16 \ln \delta_v(r)$.

Suppose the latter is the case and that w is such a child. Using Lemma 4.1 again, we see that in phase $r-1$, when $\delta_w(r-1) \geq \delta_w(r) \geq \delta_v(r)/16 \ln \delta_v(r)$, w is removed with probability $1 - 5/\delta_w(r-1)$ if it does not have $\delta_w(r-1)/2$ children of degree at least $\delta_w(r-1)/16 \ln \delta_w(r-1)$. Thus, the expected number of such nodes w that do not themselves have many high-degree children in phase $r-1$ but survive until phase r is bounded by

$$\frac{5\delta_v(r-1)}{\delta_w(r-1)} \leq \frac{80\delta_v(r-1)\ln \delta_v(r)}{\delta_v(r)}.$$

Since Lemma 4.1 states that the probability bound for a node $w \in \mathcal{C}_v(r-1)$ to be removed holds independently of v's actions, we can apply Chernoff's bound in order to see that

$$\frac{(80 + 1/2)\delta_v(r-1)\ln \delta_v(r)}{\delta_v(r)} + \mathcal{O}(\log n)$$

of these nodes remain active at the beginning of phase r w.h.p. If this number is not smaller than $\delta_v(r)/4 \in \omega(\log n)$, it holds that $\delta_v(r-1) \in \delta_v(r)^2/((80+1/2+o(1))\ln\delta_v(r))$. Otherwise, at least $\delta_v(r)/2 - \delta_v(r)/4 = \delta_v(r)/4$ children $w \in \mathcal{C}_v(r-1)$ have degree $\delta_w(r-1) \geq \delta_v(r)/16\ln\delta_v(r)$. In both cases, v meets one of the conditions in the recursive definition of a delay tree. Repeating this reasoning inductively for all $r \in \mathcal{O}(\sqrt{\log n/\log\log n})$ rounds (where we may choose the constants in the $\mathcal{O}$-term to be arbitrarily large), we construct a delay tree of depth at least r w.h.p.

However, Lemma 4.3 states that $r \in \mathcal{O}(\sqrt{\log n/\log\log n})$ provided that $\delta_v(r) \geq e^R$. Therefore, for an appropriate choice of constants, we conclude that w.h.p. the event $\mathcal{E}$ that both half of the nodes in $\mathcal{C}_v(r)$ have degree at least $\delta_v(r)/16\ln\delta_v(r)$ *and* $\delta_v(r) \geq e^R$ does not occur. If $\mathcal{E}$ does not happen, but $\delta_v(r) \geq e^R$, Lemma 4.1 gives that v is deleted in phase r with probability at least $1-5e^{-R}$.

Thus, the total probability that v is removed or has sufficiently small degree at the beginning of phase $r+1$ is bounded by

$$\begin{aligned}
&P\left[\bar{\mathcal{E}}\right] \cdot P\left[v \text{ deleted in phase } r \,\middle|\, \bar{\mathcal{E}} \text{ and } \delta_v(r) \geq e^R\right] \\
&> \left(1-\frac{1}{n}\right)\left(1-\frac{5}{e^R}\right) \\
&> 1-6e^{-R},
\end{aligned}$$

where we used that $R < \ln n$ because $\delta_v \leq n-1$. Since all used statements hold independently of v's parent's actions during the course of the algorithm, this concludes the proof. □

For convenience reasons, we rephrase the previous lemma in a slightly different way.

COROLLARY 4.5. *Provided that $R \geq 2\sqrt{\log n \log\log n}$, for any node $v \in V$ it holds with probability $1-e^{-\omega(R)}$ that $\delta_v(R) < e^R$. This bound holds independently of the actions of a constant number of v's neighbors.*

PROOF. Observe that $R \in \omega(r)$, where r and R are as in Lemma 4.4. The lemma states that after r rounds, v retains $\delta_v(r+1) \geq e^R$ with probability at most $6e^{-R}$. As the algorithm behaves identically on the remaining subgraph, applying the lemma repeatedly we see that $\delta_v(R) < e^R$ with probability $1-e^{-\omega(R)}$. Ignoring a constant number of v's neighbors does not change the asymptotic bounds. □

Since we strive for a sublogarithmic value of R, the above probability bound does not ensure that all nodes will have degree smaller than e^R after R phases w.h.p. However, on paths of length at least $\sqrt{\ln n}$, at least one of the nodes will satisfy this criterion w.h.p. Moreover, nodes of degree smaller than e^R will have left a few high-degree neighbors only, which do not interfere with our forthcoming reasoning.

LEMMA 4.6. *Assume that $R \geq 2\sqrt{\log n \log\log n}$. Given a path $P = (v_0, \dots, v_k)$, define for $i \in \{0, \dots, k\}$ that C_i is the connected component of G containing v_i after removing the edges of P. If $\delta_{v_i}(R) < e^R$, denote by $\bar{C}_i$ the connected (sub)component of C_i consisting of nodes w of degree $\delta_w(R) < e^R$ that contains v_i. Then, with probability $1-e^{-\omega(\sqrt{\ln n})}$, we have that*

(i) $\delta_{v_i}(R) < e^R$ and

(ii) nodes in $\bar{C}_i$ have at most $\sqrt{\ln n}$ neighbors w of degree $\delta_w(R) \geq e^R$.

This probability bound holds independently of anything that happens outside C_i.

PROOF. Corollary 4.5 directly yields Statement (*i*). For the second statement, let u be any node of degree $\delta_u(R) < e^R$. According to Corollary 4.5, all nodes $w \in \mathcal{C}_u(R)$ have $\delta_w(R) < e^R$ with independently bounded probability $1-e^{-\omega(R)}$. In other words, the random variable counting the number of such nodes having degree $\delta_w(R) \geq e^R$ is stochastically dominated from below by the sum of $\delta_u(R)$ independent Bernoulli variables attaining the value 1 with probability $e^{-\omega(R)} \subset e^{-\omega(\sqrt{\ln n})}$. Applying Chernoff's bound, we conclude that w.h.p. no more than $\sqrt{\ln n}$ of u's neighbors have too large degrees. By means of the union bound, we thus obtain that with probability $1-e^{-\omega(\sqrt{\ln n})}$, both statements are true. □

Having dealt with nodes of degree e^R and larger, we need to show that we can get rid of the remaining nodes sufficiently fast. As a first step, we show a result along the lines of Lemma 4.1, trading in a weaker probability bound for a stronger bound on children's degrees.

LEMMA 4.7. *Given a constant $\beta > 0$, assume that in phase i for a node v we have that at least $e^{-\beta}\delta_v(i)$ of its children have degree at most $e^\beta\delta_v(i)$. Then v is deleted with at least constant probability in that phase, regardless of the random value of its parent.*

PROOF. As in Lemma 4.1, we may w.l.o.g. assume that all children with degree at most $\delta := e^\beta\delta_v(i)$ have exactly that degree. For the random variable X counting the number of nodes $w \in \mathcal{C}_v(i)$ of degree δ that satisfy Condition (1) we get that

$$E[X] \geq \sum_{\substack{w \in \mathcal{C}_v(i) \\ \delta_w(i)=\delta}} \frac{1}{\delta} > e^{-2\beta}.$$

Since the random choices are independent, applying Chernoff's bound yields that

$$P[X=0] \leq e^{-E[X]/2} < e^{-e^{-2\beta}/2} =: \gamma.$$

Provided that $X > 0$, there is at least one child $w \in \mathcal{C}_v$ of v that joins the set in phase i unless $r_v(i) > r_w(i)$. Since $r_w(i)$ is already larger than all of its neighbors' random values (except maybe v), the respective conditional probability certainly does not exceed 1/2, i.e.,

$$P[v \text{ is deleted in phase } i \,|\, X>0] \cdot P[X>0] \geq \frac{1-\gamma}{2}.$$

Since we reasoned about whether children of v join the independent set only, this bound is independent of the behaviour of v's parent. □

We cannot guarantee that the maximum degree in the subgraph formed by the active nodes drops quickly. However, we can show that for all but a negligible fraction of the nodes this is the case.

LEMMA 4.8. *Denote by $H = (V_H, E_H)$ a subgraph of G still present in phase R in which all node degrees are smaller than e^R and for any node there are no more than $\mathcal{O}(\sqrt{\log n})$*

neighbors outside H still active in phase R. If $R \geq R(n) \in \mathcal{O}(\sqrt{\log n \log\log n})$, it holds that all nodes from H are deleted after the second for-loop of the algorithm w.h.p.

PROOF. For the sake of simplicity, we consider the special case that no edges to nodes outside H exist first. We claim that for a constant $\alpha \in \mathbb{N}$ and all $i, j \in \mathbb{N}_0$ such that $i > j$ and $e^{R-j} \geq 8ec \ln n$, it holds that

$$\max_{v \in V} \left\{ \left| \left\{ w \in \mathcal{C}_v(R+\alpha i) \,|\, \delta_w(R+\alpha i) > e^{R-j} \right\} \right| \right\} \leq \max\left\{ e^{R-2i+j}, 8c \ln n \right\}.$$

w.h.p. For $i = 1$ we have $j = 0$, i.e., the statement holds by definition because degrees in H are bounded by e^R. Assume the claim is established for some value of $i \geq 1$.

Consider a node $w \in H$ of degree $\delta_w(R+\alpha i) > e^{R-j} \geq 8ec \ln n$ for some $j \leq i$. By induction hypothesis, w.h.p. the number of children of w having degree larger than $e^{R-(j-1)}$ in phase $R+\alpha i$ (and thus also subsequent phases) is bounded by $\max\{e^{R-(2i-(j-1))}, 8c \ln n\} \leq e^{R-(j+1)}$, i.e., at least a fraction of $1 - 1/e$ of w's neighbors has degree at most factor e larger than $\delta_w(R+\alpha i)$ in phase $R + \alpha i$. According to Lemma 4.7, this implies that w is removed with constant probability in phase $R + \alpha i$. Moreover, as long as w keeps such a high degree, also in subsequent phases there is at least a constant probability that w is removed. This constant probability bound holds independently from previous phases (conditional to the event that w retains degree larger than e^{R-j}). Furthermore, due to the lemma, it applies to all children w of a node $v \in H$ independently. Hence, applying Chernoff's bound, we get that in all phases $k \in \{\alpha i, \alpha i + 1, \ldots, \alpha(i+1)-1\}$, the number $|\{w \in \mathcal{C}_v(k) \,|\, \delta_w(k) > e^{R-j}\}|$ drops by a constant factor w.h.p. (unless this number is already smaller than $8c \ln n$). Consequently, if the constant α is sufficiently large, the induction step succeeds, completing the induction.

Recapitulated, after in total $\mathcal{O}(R)$ phases, no node in H will have more than $\mathcal{O}(\log n)$ neighbors of degree larger than $\mathcal{O}(\log n)$. The previous argument can be extended to reduce degrees even further. The difficulty arising is that once the expected number of high-degree nodes removed from the respective neighborhoods becomes smaller than $\Omega(\log n)$, Chernoff's bound does no longer guarantee that a constant fraction of high-degree neighbors is deleted in each phase. However, as used before, for critical nodes the applied probability bounds hold in each phase independently of previous rounds. Thus, instead of choosing α as a constant, we simply increase α with i.

Formally, if j_0 is the greatest index such that $e^{R-j_0} \geq 8ec \ln n$, we define

$$\alpha(i) := \begin{cases} \alpha & \text{if } i \leq j_0 \\ \lceil \alpha e^{i-j_0} \rceil & \text{otherwise.} \end{cases}$$

This way, the factor e loss in size of expected values (weakening the outcome of Chernoff's bound) is compensated for by increasing the number of considered phases by factor e (which due to independence appears in the exponent of the bound) in each step. Hence, within

$$\begin{aligned} &\sum_{i=\lceil \ln \sqrt{\log n} \rceil}^{R} \alpha(i) \\ &\in \mathcal{O}\left(R + \sum_{i=\lceil \ln \sqrt{\log n} \rceil}^{\lceil \ln(8ec \ln n) \rceil} \frac{\ln n}{e^i} \right) \\ &= \mathcal{O}\left(R + \frac{\ln n}{\sqrt{\log n}} \right) \\ &= \mathcal{O}(R) \end{aligned}$$

many phases w.h.p., no node in H will have left more than $\mathcal{O}(\sqrt{\log n})$ neighbors of degree larger than $\mathcal{O}(\sqrt{\log n})$. Assuming that constants are chosen appropriately, this is the case after the first for-loop of the algorithm.

Recall that in the second loop the algorithm removes all nodes of degree at most R in each iteration. Thus, degrees in H will drop to $\mathcal{O}(\sqrt{\log n})$ in the first iteration of the loop, and subsequently all remaining nodes from H will be removed in the second iteration. Hence, indeed all nodes from H are deleted at the end of the second for-loop w.h.p., as claimed.

Finally, recall that no node has more than $\mathcal{O}(\sqrt{\log n})$ edges to nodes outside H. Choosing constants properly, these edges contribute only a negligible fraction to nodes' degrees even once they reach $\mathcal{O}(\sqrt{\log n})$. Thus, the asymptotic statement obtained by the above reasoning holds true also if we consider a subgraph H where nodes have $\mathcal{O}(\sqrt{\log n})$ edges leaving the subgraph, concluding the proof. □

We are now in the position to prove our bound on the running time of Algorithm 1.

THEOREM 4.9. *Assume that G is a forest and the coloring steps of Algorithm 1 are performed by a subroutine running for $\mathcal{O}(R + \log^* n)$ rounds. Then the algorithm eventually terminates and outputs a maximal independent set. Furthermore, if $R \geq R(n) \in \mathcal{O}(\sqrt{\log n \log\log n})$, Algorithm 1 terminates w.h.p. within $\mathcal{O}(R)$ rounds.*

PROOF. Correctness is obvious because (*i*) adjacent nodes can never join I concurrently, (*ii*) all neighbors of nodes that enter I are immediately deleted, (*iii*) no nodes from $V \setminus \bigcup_{v \in I}(\mathcal{N}_v \cup \{v\})$ get deleted, and (*iv*) the algorithm does not terminate until $V = \emptyset$. The algorithm will eventually terminate, as in each iteration of the third loop all leaves and isolated nodes are deleted and any forest contains either of the two.

Regarding the running time, assume that the value $R \in \mathcal{O}(\sqrt{\log n \log\log n})$ is sufficiently large, root the tree at a node v_0, and consider any path $P = (v_0, \ldots, v_k)$ of length $k \geq \sqrt{\ln n}$. Denote by C_i, $i \in \{0, \ldots, k\}$, the connected component of G containing v_i after removing the edges of P and—provided that $\delta_{v_i}(R) < e^R$—by $\bar{C}_i$ the connected (sub)component of C_i that contains v_i and consists of nodes w of degree $\delta_w(R) < e^R$ (as in Lemma 4.6). Then, by Lemma 4.6, with probability independently lower bounded by $1 - e^{-\omega(\sqrt{\ln n})}$, we have that

(*i*) $\delta_{v_i}(R) < e^R$ and

(*ii*) nodes in $\bar{C}_i$ have at most $\sqrt{\ln n}$ neighbors w of degree $\delta_w(R) \geq e^R$.

Hence, each of the $\bar{C}_i$ satisfies with a probability that is independently lower bounded by $1-e^{-\omega(\sqrt{\ln n})}$ the prerequisites of Lemma 4.8, implying that w.h.p. all nodes in $\bar{C}_i$ are deleted by the end of the second for-loop. Since Property (i) implies that $\bar{C}_i$ exists, we conclude that independently of all $v_j \neq v_i$, node v_i is *not* deleted until the end of the second loop with probability $e^{-\omega(\sqrt{\ln n})}$. Thus, the probability that no node from P gets deleted is at most

$$\left(e^{-\omega(\sqrt{\ln n})}\right)^k \subseteq e^{-\omega(\ln n)} = n^{-\omega(1)}.$$

In other words, when the second loop is completed, w.h.p. no path of length $k \geq \sqrt{\ln n}$ exists in the remaining graph, implying that w.h.p. any of its components has diameter at most $\sqrt{\ln n}$. Consequently, it will take at most $\sqrt{\ln n}$ iterations of the third loop until all nodes have been deleted.

Summing up the running times for executing the three loops of the algorithm, we get that it terminates within $\mathcal{O}(R + (R + \log^* n) + \sqrt{\ln n}) = \mathcal{O}(R)$ rounds w.h.p. □

We complete our analysis by deducing a uniform algorithm that features the claimed bounds on time and bit complexity.

THEOREM 4.10. *There exists a uniform MIS algorithm that terminates w.h.p. within $\mathcal{O}(\log n)$ rounds on general graphs and $\mathcal{O}(\sqrt{\log n \log\log n})$ rounds on forests. It can be implemented with a bit complexity of $\mathcal{O}(\log n)$ w.h.p.*

PROOF. Instead of running Algorithm 1 directly, we wrap it into an outer loop trying to guess a good value for R (i.e., $R(n) \leq R \in \mathcal{O}(R(n))$, where $R(n)$ as in Theorem 4.9). Furthermore, we restrict the number of iterations of the third loop to R, i.e., the algorithm will terminate after $\mathcal{O}(R)$ steps, however, potentially without producing an MIS.[4] Starting e.g. from two, with each call R is doubled. Once R reaches $R(n)$, according to Theorem 4.9 the algorithm outputs an MIS w.h.p. provided that G is a forest. Otherwise, R continues to grow until it becomes logarithmic in n. At this point, the analysis of the algorithm of Métivier et al. [11] applies to the first loop of the algorithm, showing that it terminates and return an MIS w.h.p. Hence, as the running time of each iteration of the outer loop is (essentially) linear in R and R grows exponentially, the overall running time of the algorithm is $\mathcal{O}(\sqrt{\log n \log\log n})$ on forests and $\mathcal{O}(\log n)$ on arbitrary graphs w.h.p.

Regarding the bit complexity, consider the first and third loop of Algorithm 1 first. In each iteration, a constant number of bits for state updates (entering MIS, being deleted without joining MIS, etc.) needs to be communicated as well as a random number that has to be compared to each neighbor's random number. However, in most cases exchanging a small number of leading bits is sufficient to break symmetry. Overall, as shown by Métivier et al. [11], this can be accomplished with a bit complexity of $\mathcal{O}(\log n)$ w.h.p. Essentially, for every round their algorithm generates a random value and transfers the necessary number of leading bits to compare these numbers to each neighbor only. Using Chernoff's bound, comparing $\mathcal{O}(\log n)$ random numbers between neighbors thus requires $\mathcal{O}(\log n)$ exchanged bits w.h.p., as in expectation each comparison requires to examine a constant number of bits. Thus, if nodes do not wait for a phase to complete, but rather continue to exchange random bits for future comparisons in a stream-like fashion, the bit complexity becomes $\mathcal{O}(\log n)$.

However, in order to avoid increasing the (sublogarithmic) time complexity of the algorithm on trees, more caution is required. Observe that in each iteration of the outer loop, we know that $\Theta(R)$ many random values need to be compared to execute the respective call of Algorithm 1 correctly. Thus, nodes may exchange the leading bits of these $\Theta(R)$ many random values concurrently, without risking to increase the asymptotic bit complexity. Afterwards, for the fraction of the values for which the comparison remains unknown, nodes send the second and the third bit to their neighbors simultaneously. In subsequent rounds, we double the number of sent bits per number repeatedly. Note that for each single value, this way the number of sent bits is at most doubled, thus the probabilistic upper bound on the total number transmitted bits increases at most by a factor of two. Moreover, after $\log\log n$ rounds, $\log n$ bits of each single value will be compared in a single round, thus at the latest after $\log\log n + \mathcal{O}(1)$ rounds all comparisons are completed w.h.p. Employing this scheme, the total time complexity of all executions of the first and third loop of Algorithm 1 is (in a forest) bounded by

$$\begin{aligned}
&\mathcal{O}\left(\sum_{i=1}^{\lceil \log R(n) \rceil} 2^i + \log\log n\right) \\
&\subseteq \mathcal{O}(R(n) + \log R(n) \log\log n) \\
&= \mathcal{O}\left(\sqrt{\log n \log\log n}\right)
\end{aligned}$$

w.h.p.

It remains to show that the second loop of Algorithm 1 does not require the exchange of too many bits. The number of transmitted bits to execute this loop is determined by the number of bits sent by the employed coloring algorithm. Barenboim and Elkin [2] and Kuhn [7] independently provided deterministic coloring algorithms with running time $\mathcal{O}(R + \log^* n)$. These algorithms start from an initial coloring with a number of colors that is polynomial in n (typically one assumes identifiers of size $\mathcal{O}(\log n)$), which can be obtained w.h.p. by choosing a random color from the range $\{1, \ldots, n^{\mathcal{O}(1)}\}$. Exchanging these colors (which also permits to verify that the random choices indeed resulted in a proper coloring) thus costs $\mathcal{O}(\log n)$ bits.[5] However, as the maximum degree of the considered subgraphs is $R+1$, which is in $\mathcal{O}(\log n)$ w.h.p., subsequent rounds of the algorithms deal with colors that are of (poly)logarithmic size in n. As exchanging coloring information is the dominant term contributing to message size in both algorithms, the overall bit complexity of all executions of the second loop of Algorithm 1 can be kept as low as $\mathcal{O}(\log n + R(n)\log\log n) = \mathcal{O}(\log n)$. □

5. REFERENCES

[1] N. Alon, L. Babai, and A. Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal

[4] There is no need to start all over again; one can build on the IS of previous iterations, although this does not change the asymptotic bounds.

[5] To derive a uniform solution, one again falls back to doubling the size of the bit string of the chosen color until the coloring is locally feasible.

Independent Set Problem. *Journal of Algorithms*, 7(4):567 – 583, 1986.

[2] L. Barenboim and M. Elkin. Distributed $(\Delta + 1)$-Coloring in Linear (in Δ) Time. In *Proc. 41st annual ACM symposium on Theory of computing (STOC)*, pages 111–120, 2009.

[3] L. Barenboim and M. Elkin. Sublogarithmic Distributed MIS algorithm for Sparse Graphs using Nash-Williams Decomposition. *Distributed Computing*, 22(5–6):363–379, 2009.

[4] R. Cole and U. Vishkin. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Information and Control*, 70(1):32–53, 1986.

[5] A. Goldberg, S. Plotkin, and G. Shannon. Parallel Symmetry-Breaking in Sparse Graphs. In *In Proc. 19th Annual ACM Conference on Theory of Computing (STOC)*, pages 315–324, 1987.

[6] A. Israeli and A. Itai. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22(2):77 – 80, 1986.

[7] F. Kuhn. Weak Graph Coloring: Distributed Algorithms and Applications. In *In Proc. 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2009.

[8] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local Computation: Lower and Upper Bounds. *Computing Research Repository*, abs/1011.5470, 2010.

[9] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[10] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1055, 1986.

[11] Y. Métivier, J. M. Robson, N. Saheb Djahromi, and A. Zemmari. An optimal bit complexity randomised distributed MIS algorithm. In *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 323–337, 2009.

[12] M. Naor. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991.

[13] A. Panconesi and A. Srinivasan. On the Complexity of Distributed Network Decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

[14] J. Schneider and R. Wattenhofer. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Proc. of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2008.