Sleeping is Efficient: MIS in $O(1)$-rounds Node-averaged Awake Complexity

Soumyottam Chatterjee [*]   Robert Gmyr [†]   Gopal Pandurangan [‡]

June 16, 2020

Abstract

Maximal Independent Set (MIS) is one of the fundamental problems in distributed computing. The round (time) complexity of distributed MIS has traditionally focused on the *worst-case time* for all nodes to finish. The best-known (randomized) MIS algorithms take $O(\log n)$ worst-case rounds on general graphs (where n is the number of nodes). Breaking the $O(\log n)$ worst-case bound has been a longstanding open problem, while currently the best-known lower bound is $\Omega(\sqrt{\frac{\log n}{\log \log n}})$ rounds.

Motivated by the goal to reduce *total* energy consumption in energy-constrained networks such as sensor and ad hoc wireless networks, we take an alternative approach to measuring performance. We focus on minimizing the total (or equivalently, the *average*) time for all nodes to finish. It is not clear whether the currently best-known algorithms yield constant-round (or even $o(\log n)$) node-averaged round complexity for MIS in general graphs. We posit the *sleeping model*, a generalization of the traditional model, that allows nodes to enter either "sleep" or "waking" states at any round. While waking state corresponds to the default state in the traditional model, in sleeping state a node is "offline", i.e., it does not send or receive messages (and messages sent to it are dropped as well) and does not incur any time, communication, or local computation cost. Hence, in this model, only rounds in which a node is awake are counted and we are interested in minimizing the average as well as the worst-case number of rounds a node spends in the awake state, besides the traditional worst-case round complexity (i.e., the rounds for all nodes to finish including both the awake and sleeping rounds).

Our main result is that we show that *MIS can be solved in (expected) $O(1)$ rounds under node-averaged awake complexity measure* in the sleeping model. In particular, we present a randomized distributed algorithm for MIS that has expected $O(1)$-*rounds node-averaged awake complexity* and, with high probability has $O(\log n)$-*rounds worst-case awake complexity* and $O(\log^{3.41} n)$-*rounds worst-case complexity*.

Our work is a step towards understanding the node-averaged complexity of MIS both in the traditional and sleeping models. It is also the first step in designing energy-efficient distributed algorithms for energy-constrained networks.

Keywords: Maximal Independent Set, sleeping model, node-averaged round complexity, awake complexity, resource-efficient algorithm, MIS, energy-efficient algorithm, energy-efficiency.

---

[*]Georgetown University, Washington, D.C. Email: `sc1943@georgetown.edu`. Research supported in part by NSF award CCF-1733842

[†]Microsoft, Redmond, Washington. Email: `robert.gmyr@microsoft.com`. Work done while at University of Houston.

[‡]University of Houston, Houston, Texas. Email: `gopal@cs.uh.edu`. Research supported in part by NSF grants IIS-1633720, CCF-1540512, and CCF-1717075, and by BSF grant 2016419.

1   Introduction

Computing a *maximal independent set* or *MIS* problem in a network is one of the central problems in distributed computing. About 35 years ago, Alon, Babai, and Itai [2] and Luby [20] presented a randomized distributed algorithm for MIS, running on n-node graphs in $O(\log n)$ rounds with high probability.[1] Since then the MIS problem has been studied extensively, and recently there has been some exciting progress in designing faster distributed MIS algorithms. For n-node graphs with maximum degree Δ, Ghaffari [13] presented a randomized MIS algorithm running in

$$O(\log \Delta) + 2^{O(\sqrt{\log \log n})} \text{ rounds,}$$

improving over the algorithm of Barenboim et al. [5] that runs in

$$O(\log^2 \Delta) + 2^{O(\sqrt{\log \log n})} \text{ rounds.}$$

It was further improved by Rozhon and Ghaffari to

$$O(\log \Delta + \text{poly} (\log \log n))$$

rounds [29, Corollary 3.2].

   While the above results constitute a significant improvement in our understanding of the round complexity of the MIS problem, it should be noted that in general graphs, the best-known running time is still $O(\log n)$ (even for randomized algorithms). Furthermore, there is a lower bound of

$$\Omega(\min\{\tfrac{\log \Delta}{\log \log \Delta}, \sqrt{\tfrac{\log n}{\log \log n}}\})$$

   due to Kuhn et al. [18] that also applies to randomized algorithms. Thus, for example, say, when $\Delta = \omega(2^{\sqrt{\log n}})$, it follows that one cannot hope for algorithms faster than $\sqrt{\tfrac{\log n}{\log \log n}}$ rounds. Balliu et al. showed recently that one cannot hope for algorithms that run within $o(\Delta) + O(\log^* n)$ rounds for the regimes where $\Delta << \log \log n$ (for randomized algorithms) [4, Corollary 5] and $\Delta << \log n$ (for deterministic algorithms) [4, Corollary 6].

1.1   Energy considerations, Sleeping model, and Node-averaged round complexity

It is important to note that all prior works on MIS, including the ones mentioned above, are focused on measuring the *worst-case* number of rounds for nodes to finish. In other words, the time complexity is measured as the time (number of rounds) needed for the last (slowest) node(s) to finish. As mentioned above, the best-known bound for this measure is still $O(\log n)$ for general graphs (even for randomized algorithms). In this paper, we take an alternative approach to designing MIS algorithms motivated by two main considerations.

   The first consideration is the motivation of designing *energy-efficient* algorithms for ad hoc wireless and sensor networks. In such networks, a node's energy consumption depends on the amount of time it is actively communicating with nodes; more importantly, significant energy is spent by a node even when it is just *idle*, i.e., waiting to hear from a neighbor. Experimental results show that the energy consumption in an idle state is only slightly smaller than that in a transmitting or receiving state [32, 10]. Thus, even though there might be no messages exchanged

---

[1]Throughout, we use "with high probability (whp)" to mean with probability at least $1 - n^{-\gamma}$, for some constant $\gamma > 1$.

between a node and its sender, a node might be spending quite a bit of energy if it is just waiting to receive a message.

On the other hand, the energy consumption in the "sleeping" state, i.e., when it has switched off its communication devices and is not sending, receiving or listening, is significantly less than in the transmitting/receiving/idle (listening) state (see e.g., [32, 10, 17, 30, 31]). A node may cleverly enter and exit sleeping mode to save energy during the course of an algorithm. In fact, this has been exploited by protocols to save power in ad hoc wireless networks by judiciously switching between two states — *sleeping* and *awake* — as needed (the MAC layer provides support for switching between states [32, 31, 24]).

The second consideration, motivated by the first, is saving the *total* amount of energy spent by the nodes during the course of an algorithm. Note that in sleeping mode, we assume that there is no energy spent. Thus the total energy is measured as proportional to the *total time (number of rounds)* that nodes have spent in the "awake" or "normal" mode (i.e., non-sleeping mode). In this paper, we thus focus on minimizing the total number of rounds — or equivalently the *average* number of rounds — spent by all nodes in their *awake state* during an algorithm. Our goal is to design distributed algorithms with low *node-averaged awake complexity* (see Section 1.2).

Motivated by the above considerations, we posit the *sleeping model* for distributed algorithms which is a generalization of the traditional model (a more detailed description is given in Section 1.2). In the sleeping model, a node can be in either of the two states — *sleeping* or *awake (or normal)*. While in the traditional model nodes are only in the awake state, in the sleeping model nodes have the *option* of entering sleeping state at any round as well as exiting the sleeping state and entering the awake state at a later round. In the sleeping state, a node does not send or receive messages and messages sent to it by other nodes are lost; it also does not do any local computation. If a node enters a sleeping state, then it is assumed that it does not incur any time or message cost (or other resource costs, such as energy). Some previous models (see e.g., [16] and the references therein) assumed that nodes incur little or no energy only when they are not sending/receiving messages; however, this is not true in real-world ad hoc wireless and sensor networks, where considerable energy is spent even when nodes are "idle" or "listening" for messages. The sleeping model is more realistic, since in the sleeping state nodes turn off their communication (e.g., wireless) devices fully. However, it becomes more challenging to design efficient algorithms under this model.

1.2   Model and Complexity Measures

Before we define the sleeping model, we will recall the traditional model used in distributed algorithms.

Traditional Model.   We consider the standard synchronous Congest model [28], where nodes are always "awake" from the start of the algorithm (i.e., round zero). We are given a distributed network of n nodes, modeled as an undirected graph G. Each node hosts a processor with limited initial knowledge. We assume that nodes have unique IDs,[2] and at the beginning of the computation each node is provided its ID as input. We assume that each node has ports (each port having a unique port number); each incident edge is connected to one distinct port. We also assume that nodes know n, the number of nodes in the network. Thus, a node has only *local* knowledge.

Nodes are allowed to communicate through the edges of the graph G and it is assumed that communication is synchronous and occurs in rounds. In particular, we assume that each node

---

[2]Making this assumption is not essential, but it simplifies presentation.

knows the current round number (starting from round 0). In each round, each node can perform some local computation (which finishes in the same round) including accessing a private source of randomness, and can exchange (possibly distinct) $O(\log n)$-bit messages with each of its neighboring nodes.

This model of distributed computation is called the Congest$(\log n)$ model or simply the Congest model [28]. We note that our algorithms also, obviously apply to the Local model, another well-studied model [28] where there is no restriction on the size of the messages sent per edge per round. The Local (resp. Congest) model does not put any constraint on the computational power of the nodes, but we do not abuse this aspect: our algorithms perform only light-weight computations.

Some distributed MIS algorithms (e.g., [26, 13, 5]) assume the Local model, whereas others work also in the Congest model [20, 25, 14].

Sleeping Model. We augment the traditional Congest (or Local) model by allowing nodes to enter a *sleeping state* at any round. In the sleeping model, a node can be in either of the two states before it finishes executing the algorithm (locally) — in other words, before it enters a final "termination" state. That is, any node v, can decide to *sleep* starting at any (specified) round of its choice; we assume all nodes know the correct round number whenever they are awake.[3] It can *wake up* again later at any specified round — this is the *awake* state. In the sleeping state, a node can be considered "dead" so to speak: it does not send or receive messages, nor it does any local computation. Messages sent to it by other nodes when it was sleeping are lost. However, a node can awake itself at any specified later round. Note that in the traditional model, which can be considered as a special case of the sleeping model, nodes are always in the awake state. In the sleeping model, a node can potentially conserve its resources by judiciously determining if, when, and how long to sleep.

Node-averaged Round Complexity. For a distributed algorithm on a network $G = (V, E)$ in the sleeping model, we are primarily interested in the "node-averaged awake round complexity" or simply "node-averaged awake complexity". For a deterministic algorithm, for a node v, let a_v be the number of "awake" rounds needed for v to finish, i.e., a_v *only counts the number of rounds in the awake state* of v. Then we define the *node-averaged awake complexity* to be $\frac{1}{n} \sum_{v \in V} a_v$.

For a randomized algorithm, for a node v, let A_v be the random variable denoting the number of awake rounds needed for v to finish. Then let the random variable A be defined as $\frac{1}{n} \sum_{v \in V} A_v$, i.e., the average of the A_v random variables. Then the expected *node-averaged awake complexity* of the randomized algorithm is

$$\mathrm{E}[A] = \mathrm{E}[\tfrac{1}{n} \textstyle\sum_{v \in V} A_v] = \tfrac{1}{n} \textstyle\sum_{v \in V} \mathrm{E}[A_v].$$

In this paper, we are mainly focused on this measure. However, one can also study other properties of A, e.g., high probability bounds on A.

Note that analogous definitions also naturally apply to the *node-averaged round complexity*[4] in the *traditional model*, where all rounds are counted (since nodes are always awake).

---

[3] One way to implement the sleeping model is to assume that a node's local (synchronized) clock is always running; a node before it enters the sleeping state, sets an "interrupt" (alarm) to wake at a specified later round. In practice, the IEEE 802.11 MAC provides low-level support for power management and synchronizing nodes to wake up for data delivery [32, 24, 31].

[4]Note that henceforth when we don't specify "awake" in the complexity measure, it means that we are referring to the traditional model, and if we do, we are referring to the sleeping model.

Worst-case Round Complexity. We measure the "worst-case awake round complexity" (or simply the "worst-case awake complexity") in the sleeping model as the *worst-case* number of *awake* rounds (from the start) taken by a node to finish the algorithm. That is, if a_v be the number of awake rounds of v before it terminates, then the *worst-case awake complexity* is $\max_{v \in V} a_v$.

While our goal is to design distributed algorithms that are efficient with respect to *node-averaged awake complexity*, we would also like them to be efficient (as much as possible) with respect to the *worst-case awake complexity*, as well as the traditional *worst-case round complexity*, where *all rounds* (including rounds spent in sleeping state) are counted.

1.3 MIS in $O(1)$-rounds node average complexity?

In light of the difficulty in breaking the $o(\log n)$-round (traditional) worst-case barrier and the $\Omega(\min\{\frac{\log \Delta}{\log \log \Delta}, \frac{\sqrt{\log n}}{\log \log n}\})$ lower bound for worst-case round complexity, as well as motivated by energy considerations discussed above, a fundamental question that we seek to answer is this:

> *Can we design a distributed MIS algorithm that takes $O(1)$-rounds node-averaged awake complexity?*

Before we answer this question, it is worth studying the node-averaged round complexity of some well-known distributed MIS algorithms in the *traditional model*. It is not clear whether Luby's algorithms (both versions of it [27, 20]) give $O(1)$-round node-averaged complexity, or even $o(\log n)$-round node-averaged complexity. The same is the situation with the algorithms of Alon et al. [2] and Karp et al. [15] as well as known deterministic MIS algorithms [3, 26] (see also [5, 29]). The algorithm of [13] (also of Barenboim et al. [5]) does not seem to give $O(1)$ (or even $o(\log n)$) node-averaged complexity. For example, take Ghaffari's algorithm [13], which is well-suited for analyzing the node-averaged complexity since it is "node centric": for any node v, it gives a probabilistic bound on when v will finish. More precisely, it shows that for each node v, the probability that v has not finished (i.e., its status has not been determined) after $O(\log (deg(v)) + \log (\frac{1}{\epsilon}))$ rounds is at most ϵ. Using this it is easy to compute the (expected) node-averaged complexity of Ghaffari's algorithm. However, this is still only $O(\log n)$, as $\log (deg(v))$ can be $\Theta(\log n)$ for most nodes.

Recently Barenboim and Tzur [6] showed that MIS can be solved in $O(a + \log^* n)$ rounds under *node-averaged complexity* deterministically, where a is the arboricity of the graph. It is a open question whether one can design an algorithm with $O(1)$ (or even $o(\log n)$) node-averaged round complexity in the *traditional model* for general graphs (which can have arboricity as high as $\Theta(n)$). Hence a new approach is needed to show $o(\log n)$, in particular, $O(1)$ node-averaged round complexity.

1.4 Our Contributions

Our main contributions are positing the sleeping model and designing algorithms with *constant*-rounds node-averaged awake complexity algorithm for MIS in the model.

Our main result is a randomized distributed MIS algorithm in the *sleeping model* whose *(expected) node-averaged awake complexity is $O(1)$*. In particular, we present a randomized distributed algorithm (Algorithm 2) that has $O(1)$-rounds expected node-averaged awake complexity and, with high probability, has $O(\log n)$-round worst-case awake complexity, and $O(\log^{3.41} n)$ worst-case (traditional) round complexity (cf. Theorem 2). We refer to Table 1 for a comparison of the results. Please also see Theorem 1 and Theorem 2, respectively.

Our work is also a step towards understanding whether a $O(1)$-round node-averaged algorithm is possible in the traditional model (without sleeping).

	Prior MIS algorithms (e.g., Luby's [20, 2], CRT [9, 7, 12], etc.)	Our algorithms	
		Algorithm 1 (SLEEPINGMIS)	Algorithm 2 (FAST-SLEEPINGMIS)
Node-averaged awake complexity	Not applicable.	$O(1)$	$O(1)$
Worst-case awake complexity	Not applicable.	$O(\log n)$	$O(\log n)$
Worst-case round complexity	$O(\log n)$	$O(n^3)$	$O(\log^{3.41} n)$
Node-averaged round complexity	The best known bounds are $O(\log n)$.	$O(n^3)$	$O(\log^{3.41} n)$

Table 1: A summary of the different complexity measures for the various distributed MIS algorithms

1.5 Comparison with Related Work

Much of the research in design and analysis of efficient distributed algorithms and proving lower bounds of such algorithms in the last three decades have focused on the worst-case round complexity. Recently, there have been a few works that have focused on studying various fundamental distributed computing problems under a node-average round complexity (in the traditional model).

The notion of node-averaged (or vertex-averaged) complexity for the *traditional model* was proposed by Feuilloley [11] and further studied (with slight modifications) in Barenboim and Tzur [6]. The motivation for node-averaged complexity — which also applies here — is that it can better capture the performance of distributed algorithms vis-a-vis the resources expended by individual nodes [6].

In Feuilloley's notion [11], a node's running time is counted only till it outputs (or commits its output); the node may still participate later (e.g., can forward messages etc.) but the time after it decides its output is *not counted*. In other words, the node-averaged complexity is the average of the runtimes of the nodes, where a node's runtime is till it decides its output (though it may not have terminated). The average running time is the average of the running times under the above notion. This work [11] studies the average time complexity of leader election and coloring algorithms on cycles and other specific sparse graphs. For leader election, the paper shows gives an algorithm with $O(\log n)$ node-average complexity (we note that the worst-case time complexity has a lower bound of $\Omega(n)$, even for randomized algorithms [19]). For 3-coloring a cycle, the paper [11] shows that the node-averaged complexity cannot be improved over the worst-case, i.e., it is $\Theta(\log^* n)$.

Following the work of Feuilloley [11], Barenboim and Tzur [6] address several fundamental problems under node-averaged round complexity. Their notion is somewhat different from that of Feuilloley — in [6], as soon as a node decides its output, it sends its output to its neighbors and terminates (does not take any further part in the algorithm). This is arguably a more suitable version for real-world networks in light of what was discussed in the context of saving energy and other node resources. Our notion is the same as that of Barenboim and Tzur, but extended to apply to the more general sleeping model where time spent by nodes in the sleeping state (if any) is not counted.

Barenboim and Tzur show a number of results for node-coloring as well as for MIS, $(2\Delta - 1)$-edge-coloring and maximal matching. Their bounds apply to general graphs, but depend on the arboricity of the graph. In particular, for MIS, $(2\Delta - 1)$-edge-coloring and maximal matching, they show a deterministic algorithm with $O(a + \log^* n)$ node-average complexity, where a is the

arboricity of the graph (which can be $\Theta(n)$ in general).

It is still not known whether one can obtain $O(1)$ (or even $o(\log n)$) round node-averaged complexity for MIS in general graphs in the traditional model (without sleeping). In this paper, we answer this question in the affirmative in the sleeping model. Note that $(\Delta + 1)$-coloring can be solved in $O(1)$ round node-averaged complexity in general graphs by using Luby's $(\Delta + 1)$-coloring algorithm [21], e.g., see the paper of Barenboim and Tzur [6, Section 6.2]; however, this does not imply any such bound for MIS.

There is also another important distinction between the algorithms of this paper and those of two works discussed above of [11, 6]: some of the algorithms in the above works assume the Local model (unbounded messages), whereas Congest model (small-sized messages) is assumed here.

Finally, we point out that in a *dynamic* network model the work of [8] analyzed the average time complexity of algorithms using amortized analysis. This model is dynamic where nodes and edges may be added or deleted from the graph and is different compared to the static setting studied in this paper which is the case with almost all prior works on MIS mentioned here.

The work of King et al. [17] uses a sleeping model similar to ours — nodes can be in two states sleeping or awake (listening and/or sending) — but their setting is different. They present an algorithm in this model to solve a reliable broadcast problem in an energy-efficient way where nodes are awake only a fraction of the time. Another different model studied in the literature for problems such as MIS is the *beeping model* (see, e.g., [1]) where nodes can communicate (broadcast) to their neighbors by either beeping or not. Sleeping is orthogonal to beeping and one can study a model that uses both.

2   Challenges and High-level overview

Before we go to an overview of our algorithm, we discuss some of the challenges in obtaining an $O(1)$-round node-averaged complexity in the traditional model. As mentioned in Section 1.3, either prior distributed MIS algorithms have $\Theta(\log n)$ node-average complexity or it is not clear if their node-averaged complexity is (even) $o(\log n)$ for arbitrary graphs. A straightforward way to show constant node-averaged round complexity is to argue that a constant fraction (on expectation, for randomized algorithms) of the nodes finish in every round; this can be shown to imply $O(1)$ node-averaged complexity. Indeed, this is the reason why Luby's (randomized) algorithm for $(\Delta + 1)$-coloring [21] gives $O(1)$ node-averaged complexity (see [6, Section 6.2]). It is not clear whether this kind of property can be shown for existing MIS algorithms (see Section 1.3).

Our main contribution is to show how one can design an algorithm with $O(1)$-rounds node-averaged *awake* complexity for MIS in the sleeping model while still having small worst-case running times (both in the sleeping and traditional models). This is non-trivial since it is not obvious how to take advantage of the sleeping model properly. A key difficulty in showing an $o(\log n)$ node-averaged awake complexity for the MIS problem in the sleeping model is that messages sent to a sleeping node are simply ignored; they are not received at a later point. Hence a sleeping node is unable to know the status of its neighbors (even after it is awake, since the neighbors could be then sleeping or even finished). Another difficulty is that it is not clear when to wake up a sleeping node; and when it wakes up, its neighbors might be sleeping and it won't know their status. It could be costly (in terms of node-averaged awake complexity) to keep all of its neighbors awake for many rounds. Hence a new approach is needed to get constant node-averaged awake complexity.

The high-level idea of our (randomized) algorithm is quite simple and can be explained by a simple recursive procedure (see Figure 1). Consider a graph $G = (V, E)$. Every node flips a fair coin. If the coin comes up heads, the node falls asleep. Otherwise, it stays awake. Let S be the
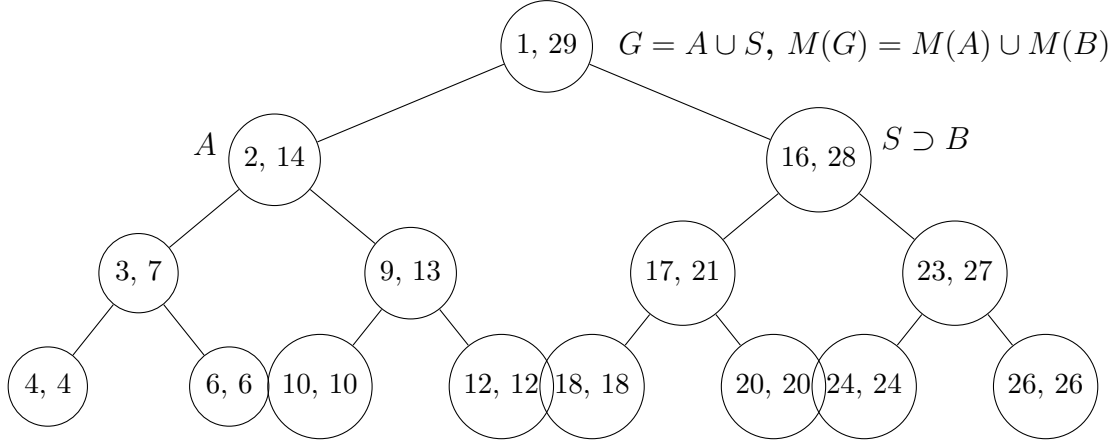
Figure 1: A sample recursion tree consisting of four levels; each tree vertex is labeled with two numbers — the first of which denotes the time when the vertex is reached for the first time, while the second number denotes the time when computation finishes at that vertex.

set of sleeping nodes and A be the set of awake nodes. The procedure is invoked recursively on the subgraph $G[A]$ induced by A to compute an MIS $M(A)$ of that subgraph. The recursion bottoms out when the procedure is invoked on an empty subgraph or a subgraph containing only a single node. In the latter case, the node joins the MIS.

Once $M(A)$ is determined, the nodes in S wake up at an appropriate time — *that is synchronized to the time when the recursive call on $G[A]$ returns* – and every node in $M(A)$ informs its neighbors that its in the MIS. At this point, the status (i.e., whether a node is in the MIS or not) of the nodes in $M(A)$ and their neighbors (including those in S) is fixed, so all of these nodes terminate.

It remains to fix the status of the remaining nodes, which form a subset $B \subseteq S$. To do so, we recursively invoke the procedure on $G[B]$, which gives us an MIS $M(B)$ of that subgraph. The overall MIS is then given by $M(A) \cup M(B)$. The recursion bottoms out when the procedure is invoked on an empty subgraph or a subgraph containing only a single node. In the latter case, the node joins the MIS.

The main observation for the analysis of this procedure is the following. By definition, $M(A)$ dominates A and therefore all nodes in A terminate after the first recursive call is finished. On top of that, the nodes in $M(A)$ might also dominate some of the nodes in S. In fact, one can show that on *on expectation at least a 1/4-fraction* of the nodes in S have a neighbor in $M(A)$ and hence will be *eliminated* when the recursive call finishes. This is shown in the key technical lemma called the *Pruning Lemma* (see Lemma 3).

A main challenge in proving Lemma 3 is that A is fixed by sampling (coin tosses) and the MIS of A does not depend on S, the set of sleeping nodes. Yet we would like to show that a constant fraction of nodes in S have a neighbor in the MIS of A, i.e., $M(A)$. Note that given A, $M(A)$ can possibly be such that the number of neighbors in S can be very small; this will not eliminate many nodes in S. We avoid this by *coupling the process of sampling with the process of finding an MIS* and show that, despite choosing A first (by random sampling), the MIS computed on A will eliminate a constant fraction of S. However, choosing A first introduces dependencies which makes it non-trivial to prove the Pruning Lemma; we overcome this by using the *principle of deferred decisions* (cf. proof of Lemma 3).

The Pruning Lemma (see Lemma 3) guarantees that a constant fraction of nodes in the graph terminate *without being included* in either of the two recursive calls. So the status of these nodes is fixed by being awake for only a constant number of rounds (only three rounds). As a consequence, the two recursive calls together only operate on at most $\left(\frac{3}{4}\right)$-fraction of the given nodes on expectation. This saving propagates down the tree of recursive calls such that at level i of the tree the overall number of nodes on which the calls at that level operate is at most $\left(\frac{3}{4}\right)^i \cdot n$. It is not hard to see that the number of rounds per vertex required by the procedure outside of the recursive calls is constant. Therefore, the overall expected vertex-average complexity of the procedure in the sleeping model is: $\frac{1}{n} \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i n = O(1)$.

The above algorithm (cf. Section 3) has *constant* round node-averaged complexity and $O(\log n)$-rounds worst-case awake complexity, but it has polynomial worst-case complexity. We then show that our MIS algorithm can be combined with a variant of Luby's algorithm so that the worst-case (traditional) complexity can be improved to polylogarithmic ($O(\log^{3.41} n)$) rounds, while still having $O(1)$ rounds node-averaged complexity and $O(\log n)$ worst-case awake complexity.

3   The Sleeping MIS Algorithm

We consider the algorithm given in Figure 1. To compute an MIS for a given graph $G = (V, E)$, each node v in G calls the function SLEEPINGMIS at the same time. The function SLEEPINGMIS is called with the function parameter $K = 3 \log n$, where n is the network size. We show in the analysis (see Lemma 1 and Theorem 1) that our algorithm is correct with high probability.

After initializing some variables, the function calls the recursive function SLEEPINGMISRE-CURSIVE. This function computes an MIS on the subgraph induced by the set of nodes that call it. In the initial call, all nodes in G call the function. In later calls, however, the function operates on proper subgraphs of G. Each (non-trivial) call of SLEEPINGMISRECURSIVE partitions the set of nodes participating in the call into two subsets. The function uses a recursive call to compute an MIS on the subgraph induced by the first set, updates the nodes in the second set about the result of the first recursive call, and finally uses a second recursive call to finish the computation of the MIS. SLEEPINGMISRECURSIVE takes an integer parameter k. This parameter starts with $k = K = \lceil 3 \log n \rceil$ in the initial call. The function parameter k is then decremented from one level of the recursion to the next until the recursion base with $k = 0$ is reached.

During the algorithm, each node v in G stores a variable v.inMIS. Initially, this variable is set to *unknown* to signify that it has not yet been determined whether v is in the MIS or not. Over the course of the algorithm the variable is set to *true* or *false*. Once v.inMIS has been set to one of these values, it is never changed again. The MIS M computed by the algorithm is given by the set of nodes v with v.inMIS = true after termination.

Consider a call of the function SLEEPINGMISRECURSIVE by a node set $U \subseteq V$ and with a parameter k. The goal of such a call is to compute an MIS in the induced subgraph $G[U]$. The function consists of six parts as indicated by the comments on the right side in Algorithm 1. The first part (Lines 9 – 12) is the *base case* of the recursion. Once the base case of the recursion is reached it holds, with high probability, that $|U| \leq 1$. Therefore, the function has to compute an MIS on a graph consisting of at most one node. Such a node simply joins the MIS.

The second part (Lines 13–16) detects isolated nodes and adds them to the MIS. We refer to this part as the *first isolated node detection*. Note that on the top level of the recursion, which operates on the entire graph G, the nodes do not have to communicate in order to determine whether a node is isolated or not. On lower levels of the recursion, however, the function generally operates on a proper subgraph $G[U]$ of G. The given instructions make sure that a node correctly determines its

Algorithm 1 The "Sleeping MIS algorithm" executed by a node v

1: **function** SLEEPINGMIS(K)
2:     $v.\text{inMIS} \leftarrow$ unknown
3:     **for** i **in** $1, \ldots, K$ **do**
4:         $v.X_i \leftarrow$ random bit that is 1 with probability $\frac{1}{2}$
5:     **end for**
6:     SLEEPINGMISRECURSIVE(K)   ▷ K is the recursion depth.
7: **end function**

8: **function** SLEEPINGMISRECURSIVE(k)
9:     **if** $k = 0$ **then**   ▷ base case
10:         $v.\text{inMIS} \leftarrow$ true
11:         **return**
12:     **end if**

13:     send message to every neighbor   ▷ first isolated node detection, 1 round
14:     **if** v receives no message **then**
15:         $v.\text{inMIS} \leftarrow$ true
16:     **end if**

17:     **if** $v.\text{inMIS} =$ unkown **and** $v.X_k = 1$ **then**   ▷ left recursion
18:         SLEEPINGMISRECURSIVE($k - 1$)
19:     **else**
20:         sleep for $T(k - 1) = 3(2^{k-1} - 1)$ rounds ▷ $T(k-1)$ is the duration of the recursive call
    in Line 18; it is computed in Section 4.3.
21:     **end if**

22:     send value of $v.\text{inMIS}$ to every neighbor   ▷ synchronization step, 1 round
23:     **if** $v.\text{inMIS} =$ unknown **and** v receives message from neighbor w with $w.\text{inMIS} =$ true **then**
24:         $v.\text{inMIS} \leftarrow$ false   ▷ elimination
25:     **end if**

26:     send value of $v.\text{inMIS}$ to every neighbor   ▷ second isolated node detection, 1 round
27:     **if** $v.\text{inMIS} =$ unknown **and** v only receives messages from neighbors w with $w.\text{inMIS} =$ false
    then
28:         $v.\text{inMIS} \leftarrow$ true
29:     **end if**

30:     **if** $v.\text{inMIS} =$ unknown **then**   ▷ right recursion
31:         SLEEPINGMISRECURSIVE($k - 1$)
32:     **else**
33:         sleep for $T(k - 1) = 3(2^{k-1} - 1)$ rounds ▷ $T(k-1)$ is the duration of the recursive call
    in Line 31
34:     **end if**
35: **end function**

neighborhood in $G[U]$.

The third part (Lines 17–21) uses a recursive call to compute a partial MIS. We call this part the *left recursion* due to the intuition of organizing the recursive calls of the function into a binary tree that is traversed in a left-to-right order (see Figure 1). Every non-isolated node $v \in U$ with $v.X_k = 1$ participates in this recursive call, where $v.X_k$ is a variable that is set to a random bit during initialization. The recursive call computes an MIS in the subgraph induced by these nodes. In doing so, it fixes the value of $v.$inMIS for every node $v \in U$ with $v.X_k = 1$. The nodes $v \in U$ with $v.X_k = 0$ and all isolated nodes in U sleep for the duration of the left recursive call, see Line 20. Thereby, all nodes in U start executing the next part of the function (which begins in Line 22) at the same time.

The purpose of the fourth and fifth part of the function is to update the nodes $v \in U$ with $v.X_k = 0$ about the decisions made in the left recursive call. We call the fourth part (Lines 22–25) the *elimination step*. In this part, every node $v \in U$ with $v.$inMIS = unkown checks whether it has a neighbor in $G[U]$ that is in the MIS. If that is the case, v sets $v.$inMIS to false. The fifth part (Lines 26 – 29) is the *second isolated node detection*. In this part, a node $v \in U$ checks whether the variable $w.$inMIS is false for every neighbor w of v in $G[U]$. If so, v sets $v.$inMIS to true.

The sixth and final part of the function (Lines 30–34) uses a second recursive call to complete the computation of the MIS in $G[U]$. We refer to this part as the *right recursion*. As in the left recursion, only a subset of the nodes participates in the recursive call while the other nodes sleep. Specifically, every node $v \in U$ for which the value of $v.$inMIS is still *unknown* participates in the right recursive call.

One important technical issue is *synchronization*. The sleeping nodes wake up at the appropriately synchronized round to synchronize with the wake up nodes (at the end of their recursive calls — (Lines 18 and 31) so that messages can be exchanged between neighbors. A node that calls SLEEPINGMISRECURSIVE(k) will wake up at a later round, which is given by the function $T(k)$. $T(k)$ is the (*worst-case*) time taken to complete this recursive call. The function $T(k) = 3(2^k - 1)$ as computed in the proof of Lemma 10.

4   Analysis

We now turn to the analysis of Algorithm 1.

4.1   Correctness

Lemma 1. *On an n-node graph G, the set M computed by the algorithm* SLEEPINGMIS(K), *with $K = \lceil 3 \log n \rceil$, is an MIS with high probability.*

Proof. We show the statement by induction. We use the following induction hypothesis: If all nodes in a set $U \subseteq V$ simultaneously call the function SLEEPINGMIS with parameter k then

1. all nodes in U return from the call in the same round,

2. after the call, the variable inMIS is true or false for all nodes in U, and

3. the set of nodes in U with inMIS = true is an MIS of the subgraph induced by U.

We begin by showing the induction step, i.e., we show that the above statement holds for node set U and parameter $k > 0$ under the assumption that it holds for any subset of U and parameter $k - 1$.

For Condition 1 of the statement observe that outside of the recursive parts, all nodes perform the same instructions and, therefore, spend the same number of rounds. In the recursive parts, the nodes that participate in the recursive calls all return in the same round according to the induction hypothesis, and the nodes that do not participate in the recursive call sleep for the exact number of rounds required for the recursive call. Therefore, Condition 1 holds.

To see that Condition 2 holds, observe that in Line 30 every node in U with inMIS = unknown performs a recursive call so that the value of inMIS is set to true or false for all of these nodes according to the induction hypothesis.

It remains to show that Condition 3 holds, i.e., the algorithm actually computes an MIS $M_{G[U]}$ on the subgraph $G[U]$ induced by U. Let $I \subseteq U$ be the set of isolated nodes in $G[U]$. The algorithm explicitly takes care of these nodes during the (first) isolated node detection (Lines 13 - 16). The remaining nodes are partitioned into two sets A and B where A is the set of nodes that participate in the first recursive call, and B is the set of remaining non-isolated nodes.

Finally, let $C \subseteq B$ be the set of nodes that participate in the second recursive call. To show that $M_{G[U]}$ is an MIS we first show that $M_{G[U]}$ is a *dominating set* in $G[U]$ and then show that $M_{G[U]}$ is an *independent set* in $G[U]$.

To show that $M_{G[U]}$ is a dominating set in $G[U]$, we show that for every node $v \in U$ either v is in $M_{G[U]}$ or a neighbor of v is in $M_{G[U]}$. If $v \in I$ then v joins the MIS during the first isolated node detection. If $v \in A$ then, according to the induction hypothesis, either v joins the MIS during the first recursive call or some neighbor u of v that is also in A joins the MIS.

If $v \in B$ we have to distinguish between two subcases:

If $v \notin C$ then we have two possibilities: (i) v.inMIS was set to false during the elimination step (Lines 22 – 25) and, therefore, v must have a neighbor u that is in A and joined the MIS; (ii) v.inMIS was set to true during the second isolated node detection (Lines 26–29), and therefore is in the MIS (and dominated).

Finally, the case $v \in C$ (Lines 30 – 31) is analogous to $v \in A$.

To show that $M_{G[U]}$ is an independent set in $G[U]$, we prove that for any edge $\{u, v\}$ in $G[U]$ if $u \in M_{G[U]}$ then $v \notin M_{G[U]}$. If $u, v \in A$ then the statement must hold according to the induction hypothesis. If $u \in A$ and $v \in B$ then v.inMIS is set to false during the elimination step (Lines 22 -25) and the statement holds. If $u \in B$ and $v \in A$ then u.inMIS was not set to false (i.e., set to true) during second isolated node detection (Lines 26 - 29) and, therefore, we must have v.inMIS = false.

For the case $u, v \in B$ we have to consider two subcases:

If $u \notin C$ or $v \notin C$ then the statement holds due to second isolated node detection step (Lines 26 - 29).

Otherwise (i.e..both u and v are in C), the statement holds by the induction hypothesis.

Finally, we consider the base case of the induction which corresponds to the base case of the recursion where $k = 0$. Recall that at the top level of the recursion, the function SLEEPINGMIS-RECURSIVE is called with the parameter k set to $\lceil 3 \log n \rceil$, and then k is decremented from one level of the recursion to the next. Consider the recursion tree corresponding to the execution of the algorithm. Each tree node corresponds to a call of the function SLEEPINGMISRECURSIVE by a subset of the nodes in G. The root of the tree corresponds to the initial call made by all nodes in G. At an internal tree node, the probability that a non-isolated node in G participates in the first recursive call is $1/2$, and the probability that a node in G participates in the second recursive call is at most $1/2$. An isolated node does not participate in any recursive calls. Therefore, the probability that a node in G participates in a specific call of SLEEPINGMISRECURSIVE corresponding to a leaf of the recursion tree is at most $2^{-k} = O(2^{-3 \log n}) = O(n^{-3})$. So for any pair of nodes u, v in G the probability that both nodes participate in the same recursive call at the bottom level of the recursion is at most $O(n^{-3})$. Applying the union bound over all pairs of nodes implies that, with

11

high probability, at most one node participates in any given recursive call at the bottom level of the recursion. Therefore, if a node v reaches the base case of the recursion then the function SLEEP-INGMISRECURSIVE operates on a subgraph of G containing only v, w.h.p. The node v simply joins the MIS (see Lines 9–12). It is easy to check that, thereby, all three conditions of induction hypothesis are satisfied. $\square$

4.2 Time (or Round) Complexity

The intuition behind the node-averaged running time analysis is that in each recursive call of SLEEPINGMISRECURSIVE, a constant fraction (on expectation) of the nodes calling the function participate in neither the left nor the right recursive call. Thereby, these nodes sleep for almost the entire duration of the call. This effect propagates through the levels of recursion and ultimately leads to a low node-averaged awake complexity.

To formalize this intuition, consider the execution of SLEEPINGMISRECURSIVE by a set of nodes $U \subseteq V$ and with parameter $k \geq 1$. Let $L \subseteq U$ be the set of nodes that participate in the left recursive call, and let $R \subseteq U$ be the set of nodes that participate in the right recursive call. The following lemma bounds the number of nodes participating in the left recursive call.

Lemma 2. $E[\, |L| \mid U \,] \leq \frac{|U|}{2}$.

Proof. Consider a node $v \in U$. If v is isolated in $G[U]$, it joins the MIS in the first isolated node detection and, thereby, does not participate in the left recursive call. Otherwise, v partipates in the left recursive call if and only if $v.X_k = 1$, which holds with probability $1/2$. The lemma follows by the linearity of expectation. $\square$

The next is a key lemma that establishes a bound for the right recursive call. The proof of this lemma requires a series of definitions and auxiliary lemmas.

Lemma 3 (Pruning Lemma). $E[\, |R| \mid U \,] \leq \frac{|U|}{4}$.

Together, Lemmas 2 and 3 imply that the expected number of nodes that participate in either of the recursive calls is

$$\mathrm{E}[\, |L| + |R| \mid U \,] \leq \tfrac{3}{4} \cdot |U|.$$

Thereby, on expectation, at least one-fourth of the nodes in U do not participate in either of the recursive calls.

We now establish the definitions and lemmas necessary to prove Lemma 3. We recall from the algorithm that $K = \lceil 3 \log n \rceil$ is the maximum recursion depth.

Definition 1. *For any k such that $0 \leq k \leq K$ and any node $v \in V$ we define the k-rank of v as the sequence*

$$r_k(v) = (v.X_k, v.X_{k-1}, \ldots, v.X_1, -1).$$

For two nodes v, w we write $r_k(v) < r_k(w)$ if and only if $r_k(v)$ is lexicographically strictly less than $r_k(w)$. (Note that $r_0(v) = (-1)$, is simply a sentinel value for the base case.)

Let $N_H(v)$ denotes the neighborhood of v in the (sub)graph H. The following lemma provides us with a condition under which a node joins the MIS based on ranks.

Lemma 4. *Consider a call of SLEEPINGMISRECURSIVE by a node set U with parameter k during the execution of the algorithm. If $v \in U$ is such that every $w \in N_{G[U]}(v)$ with $r_k(w) > r_k(v)$ sets $w.inMIS$ to false at some point during the algorithm, then v joins the MIS.*

12

Proof. We prove the lemma by *induction* on k.

The *induction base* with $k = 0$ corresponds to the base case of the recursion and holds trivially because if $k = 0$, then SLEEPINGMISRECURSIVE sets v to be in MIS (Lines $9 - 12$) regardless of rank value.

For the *induction step* we assume that the statement holds for $k - 1$ and show that it holds for k. Consider a call of SLEEPINGMISRECURSIVE by a node set U with parameter k, and let $v \in U$ be such that every $w \in N_{G[U]}(v)$ with $r_k(w) > r_k(v)$ sets w.inMIS to false at some point during the algorithm. If v is isolated in $G[U]$ then v joins the MIS in the first isolated node detection. Otherwise, we distinguish two cases based on the value of $v.X_k$.

If $v.X_k = 1$ then v participates in the left recursive call with parameter $k - 1$. Let A be the set of v's neighbors in $G[U]$ that also participate in the left recursive call. We show that each $w \in A$ with $r_{k-1}(w) > r_{k-1}(v)$ sets w.inMIS to false at some point.

Consider a node $w \in A$. If $r_k(w) \leq r_k(v)$ then it holds $r_{k-1}(w) \leq r_{k-1}(v)$ since $w.X_k = v.X_k = 1$. If $r_k(w) > r_k(v)$ then w sets w.inMIS to false at some point by definition. Thereby, the induction hypothesis implies that v joins the MIS.

If $v.X_k = 0$ then v sleeps during the left recursive call. A node $w \in N_{G[U]}(v)$ with $r_k(w) \leq r_k(v)$ must have $w.X_k = 0$. Thereby, w also sleeps during the left recursive call, and it cannot cause v to set v.inMIS to false in the synchronization step. A node $w \in N_{G[U]}(v)$ with $r_k(w) > r_k(v)$ never sets w.inMIS to true by definition. Thereby, such a node also cannot cause v to set v.inMIS to false in the synchronization step. If after the synchronization step it holds w.inMIS = false for every $w \in N_{G[U]}(v)$, then v joins the MIS in the second isolated node detection.

Otherwise, v participates in the right recursive call. Let A be the set of v's neighbors in $G[U]$ that also participate in the right recursive call. We show that each $w \in A$ with $r_{k-1}(w) > r_{k-1}(v)$ sets w.inMIS to false at some point.

Consider a node $w \in A$. If $r_k(w) \leq r_k(v)$ then it holds $r_{k-1}(w) \leq r_{k-1}(v)$ since $w.X_k = v.X_k = 0$. If $r_k(w) > r_k(v)$ then w sets w.inMIS to false at some point by definition. Thereby, the induction hypothesis implies that v joins the MIS. □

Definition 2. *Consider a call of* SLEEPINGMISRECURSIVE *by a node set $U \subseteq V$ with parameter k during the execution of the algorithm. Let $(v_1, \ldots, v_{|U|})$ be the sequence of the nodes in U sorted by lexicographically decreasing $(k-1)$-rank. We refer to this sequence as the* evaluation sequence.

Remark 1. Note that the order of the nodes in the evaluation sequence is defined based on the $(k-1)$-rank of the nodes and not the k-rank. Intuitively, the $(k-1)$-rank of a node is the sequence of random decisions used in *future* recursions. The following lemma shows that the evaluation sequence is well defined.

Lemma 5. *For every call of* SLEEPINGMISRECURSIVE *by a node set U with parameter k during the execution of the algorithm and for any $v, w \in U$, it holds with high probability that $r_{k-1}(v) \neq r_{k-1}(w)$.*

Proof. For two nodes $v, w \in V$ to participate in the same call with node set U and parameter k, we must have $v.X_i = w.X_i$ for all i such that $k < i \leq K$. For both nodes to have the same $(k-1)$-rank, we must have $v.X_i = w.X_i$ for all i such that $1 \leq i < k$. Therefore, we have

$$\Pr[\, v, w \in U \text{ and } r_{k-1}(v) = r_{k-1}(w) \,] = 2^{-K-1} \leq 2n^{-3}.$$

The statement holds by applying the union bound over all pairs of nodes. □

Using the principle of "deferred decisions". The proof of Lemma 3 is based on the *principle of deferred decisions*[5]: We still focus on a call of SLEEPINGMISRECURSIVE by a node set U with parameter k during the execution of the algorithm. We assume for the sake of analysis that the values of the variables X_k are not fixed at the beginning of the algorithm. However, the remaining X_i for $i \neq k$ are fixed, which means that the set U, the $(k-1)$-ranks of the nodes, and the evaluation sequence $(v_1, \ldots, v_{|U|})$ have all been determined. We then fix the values of the X_k using the following deferred decision process.

Deferred Decision Process

1. Let v_i be the node in the evaluation sequence with the smallest i such that $v_i.X_k$ has not yet been fixed. We fix the value of $v_i.X_k$ by flipping a fair coin and we say that v_i is *sequence-fixed*.

2. If $v_i.X_k = 1$, we also fix the value of $w.X_k$ for each neighbor $w \in N_{G[U]}(v_i)$ for which $w.X_k$ has not yet been fixed, again by flipping independent and fair coins. We say that such a neighbor w is *neighbor-fixed*.

3. We repeat these steps until the variable X_k is fixed for all nodes in the evaluation sequence.

Note that the above process does *not* change the behavior of the algorithm. It merely serves as a useful tool that allows us to analyze the algorithm as demonstrated in the following lemma.

Lemma 6. *The following statements hold for every node $v \in U$.*

1. *If v is sequence-fixed and $v.X_k = 1$, then v sets $v.inMIS$ to true before the synchronization step.*

2. *If v is neighbor-fixed, then v sets $v.inMIS$ to false before the second isolated node detection.*

Proof. We show the lemma by complete induction on the evaluation sequence $(v_1, \ldots, v_{|U|})$. The node v_1 is sequence-fixed by definition. If $v_1.X_k = 0$, none of the statements apply and the lemma holds. If $v_1.X_k = 1$, we have to show that Statement 1 holds. By definition of the evaluation sequence we have $r_{k-1}(v_1) > r_{k-1}(v_j)$ for all $j > 1$. Together with $v_1.X_k = 1$ this implies that $r_k(v_1) > r_k(v_j)$ for all $j > 1$. Therefore, Lemma 4 implies that v_1 joins the MIS. Since $v_1.X_k = 1$, v_1 must set $v_1.inMIS$ to true in the first isolated node detection or the left recursive call. Thereby, Statement 1 and the induction base hold.

For the induction step we consider an index $i \geq 2$ and show that the lemma holds for v_i under the assumption that it holds for all v_j with $j < i$. We first show that Statement 1 holds. Suppose that v_i is sequence-fixed and $v_i.X_k = 1$. We have to show that v_i sets $v_i.inMIS$ to true before the synchronization step. Consider a node $w \in N_{G[U]}(v_i)$. If $r_{k-1}(w) \leq r_{k-1}(v_i)$ then we must also have $r_k(w) \leq r_k(v_i)$ because $v_i.X_k = 1$. If $r_{k-1}(w) > r_{k-1}(v_i)$ then the lemma holds for w according to the induction hypothesis. We distinguish two cases. If w is sequence-fixed then we must have $w.X_k = 0$ because otherwise v_i would be neighbor-fixed, which is a contradiction. Therefore, we have $r_k(w) \leq r_k(v_i)$ in this case. If w is neighbor-fixed then w sets $w.inMIS$ to false before the second isolated node detection according to Statement 2. In summary, we have that for every node $w \in N_{G[U]}(v_i)$ it holds $r_k(w) \leq r_k(v_i)$ or w sets $w.inMIS$ to false at some point during the algorithm. Thereby, Lemma 4 implies that v_1 joins the MIS. Since $v_1.X_k = 1$, v_1 must set $v_1.inMIS$ to true in the first isolated node detection or the left recursive call. Therefore, Statement 1 holds.

---

[5]For an introduction to the principle see e.g., [22].

Finally, we show that Statement 2 holds. Suppose that v_i is neighbor-fixed. We have to show that v_i sets v_i.inMIS to false before the second isolated node detection. Since v_i is neighbor-fixed, it must have a neighbor $w \in N_{G[U]}(v_i)$ such that $r_{k-1}(w) > r_{k-1}(v_i)$, w is sequence-fixed, and $w.X_k = 1$. Since $r_{k-1}(w) > r_{k-1}(v_i)$ the lemma holds for w according to the induction hypothesis. Because w is sequence-fixed and $w.X_k = 1$, Statement 1 implies that w sets w.inMIS to true before the synchronization step. Since the algorithm is correct (see Lemma 1), this implies that v_i must eventually set v_i.inMIS to false. If $v_i.X_k = 0$ then v_i must set v_i.inMIS to false in the left recursive call. Otherwise, v_i learns during the synchronization step that w set w.inMIS to true and, therefore, v_i sets v_i.inMIS to false in the synchronization step. Thereby, Statement 2 holds. $\qquad\square$

We are finally ready to prove Lemma 3.

Proof of Lemma 3. Consider a call of SLEEPINGMISRECURSIVE by a node set U with parameter k during the execution of the algorithm. We show that for each $v \in U$ it holds $\Pr[\, v \in R \,] \leq 1/4$. The linearity of expectation then implies $\mathrm{E}[\, |R| \mid U \,] \leq |U|/4$, so the lemma holds.

If v is isolated in $G[U]$, then v joins the MIS in the first isolated node detection and we have $\Pr[\, v \in R \,] = 0$. Thereby, the statement holds for isolated nodes. If v is not isolated, we can apply the *law of total probability* to obtain

$$\Pr[\, v \in R \,] = \Pr[\, v \in R \mid v.X_k = 1 \,] \cdot \Pr[\, v.X_k = 1 \,]$$
$$+ \Pr[\, v \in R \mid v.X_k = 0 \,] \cdot \Pr[\, v.X_k = 0 \,].$$

By definition, it holds $\Pr[\, v \in R \mid v.X_k = 1 \,] = 0$ and $\Pr[\, v.X_k = 0 \,] = 1/2$. Therefore, we have

$$\Pr[\, v \in R \,] = \frac{1}{2} \cdot \Pr[\, v \in R \mid v.X_k = 0 \,].$$

We define E as the event that all neighbors in $N_{G[U]}(v)$ are neighbor-fixed. Recall that v is not isolated. We apply the *law of total probability* again to obtain

$$\Pr[\, v \in R \mid v.X_k = 0 \,]$$
$$= \Pr[\, v \in R \mid v.X_k = 0 \wedge E \,] \cdot \Pr[\, v.X_k = 0 \wedge E \,]$$
$$+ \Pr[\, v \in R \mid v.X_k = 0 \wedge \overline{E} \,] \cdot \Pr[\, v.X_k = 0 \wedge \overline{E} \,].$$

We first determine the probability $\Pr[\, v \in R \mid v.X_k = 0 \wedge E \,]$. If $v.X_k = 0$, v cannot cause any of its neighbors to be neighbor-fixed. Hence, if the event E occurs then every node in $N_{G[U]}(v)$ must have been neighbor-fixed by some other node. In this case, Lemma 6 implies that every node $w \in N_{G[U]}(v)$ sets w.inMIS to false before the second isolated node detection. Hence, v sets v.inMIS to true during the second isolated node detection. This implies that $\Pr[\, v \in R \mid v.X_k = 0 \wedge E \,] = 0$.

Next, we establish an upper bound on $\Pr[\, v \in R \mid v.X_k = 0 \wedge \overline{E} \,]$. If $\overline{E}$ occurs, then there is a node $w \in N_{G[U]}(v)$ that is sequence-fixed. With probability $1/2$, we have $w.X_k = 1$. In this case, Lemma 6 implies that w sets w.inMIS to true before the synchronization step. Thereby, v sets v.inMIS to false during the synchronization step. This implies $\Pr[\, v \in R \mid v.X_k = 0 \wedge \overline{E} \,] \leq 1/2$.

By combining the equations above, we get $\Pr[\, v \in R \mid v.X_k = 0 \,] \leq 1/2$ and, therefore, $\Pr[\, v \in R \,] \leq 1/4$. $\qquad\square$

Lemma 7. *Consider all calls of* SLEEPINGMISRECURSIVE *with the same parameter k. Let U_j be the set of nodes participating in the j-th of these calls. Note that all the U_j are pairwise disjoint by definition. We define*

$$Z_k = \sum_j |U_j|$$

to be the number of nodes participating in the recursive calls with parameter k. Then the following holds for all i such that $0 \le i \le K$:

$$E[Z_{K-i}] \le \left(\tfrac{3}{4}\right)^i \cdot n.$$

Proof. We show the lemma by induction on i. For the induction base consider $i = 0$. This case corresponds to the initial call of the recursive function by all nodes. Thus, we have $Z_K = n$ by definition, which shows the induction base.

For the induction step we assume that the equation holds for an i such that $0 \le i < K$ and show that it also holds for $i + 1$. Consider the calls of SLEEPINGMISRECURSIVE with parameter $k = K - i$ during the algorithm. Let m be the number of these calls, and let U_j be the set of nodes participating in the j-th call. By definition, we have

$$Z_{K-i} = \sum_{j=1}^{m} |U_j|. \tag{1}$$

Each of the m calls creates a left and right recursive call at the next level of recursion. For the call j-th call let $L_j, R_j \subseteq U_j$ be the node sets that participate in the left and right recursive call, respectively. By definition, we have

$$Z_{K-(i+1)} = \sum_{j=1}^{m} \left(|L_j| + |R_j| \right).$$

This implies that

$$
\begin{aligned}
\mathrm{E}[\, Z_{K-(i+1)} \,|\, U_1, \ldots, U_m \,] &= \mathrm{E}\left[\sum_{j=1}^{m} \left(|L_j| + |R_j| \right) \,\Bigg|\, U_1, \ldots, U_m \right] \\
&= \sum_{j=1}^{m} \mathrm{E}\left[\, |L_j| + |R_j| \,\big|\, U_j \,\right] \\
&\le \frac{3}{4} \sum_{j=1}^{m} |U_j| = \frac{3}{4} \cdot Z_{K-i},
\end{aligned}
$$

where the inequality follows from Lemmas 2 and 3, and the last equality follows from Equation 1. Taking the expectation on both sides of this equation and applying the induction hypothesis gives us

$$\mathrm{E}[Z_{K-(i+1)}] \le \tfrac{3}{4} \cdot \mathrm{E}[Z_{K-i}] \le \left(\tfrac{3}{4}\right)^{i+1} \cdot n.$$

$\square$

Lemma 8. *The expected node-averaged awake complexity of Algorithm 1 in the sleeping model is $O(1)$.*

Proof. For each node $v \in V$, we define a cost $C(v)$ which is the number of rounds v is awake during the execution of the algorithm. We define $C = \sum_{v \in V} C(v)$ to be the total cost of the algorithm. The expected node-average complexity is then $\frac{\mathrm{E}[C]}{n}$.

The initialization at the beginning of the algorithm incurs a cost of $O(1)$ for each node. To compute the cost of the recursive part of the algorithm, we use the following observation: Consider

16

a node v and a call of SLEEPINGMISRECURSIVE that v participates in. If we ignore the cost of the recursive calls, the participation of v in the call incurs a cost of $O(1)$. This implies that we can compute the total cost of the recursive part by computing the number of nodes that participate in the recursive calls at the different levels of the recursion. Recall that we defined the number of nodes participating in the calls with common parameter k as Z_k. Formally, we have

$$C = O(1) \cdot \sum_{k=0}^{K} Z_k.$$

Taking the expectation on both sides and applying Lemma 7 gives us

$$\mathrm{E}[C] = O(1) \cdot \sum_{k=0}^{K} \mathrm{E}[Z_k] = O(1) \cdot \sum_{i=0}^{K} \mathrm{E}[Z_{K-i}]$$

$$\leq O(1) \cdot \sum_{i=0}^{K} \left(\frac{3}{4}\right)^i \cdot n \leq O(n) \cdot \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i = O(n).$$

Therefore, the expected node-average awake complexity of the algorithm is $\frac{\mathrm{E}[C]}{n} = O(1)$. □

4.3   Analysis of the worst-case awake complexity of Algorithm 1

Lemma 9. *The worst-case awake round complexity of Algorithm 1 is $O(\log n)$.*

Proof. The worst-case awake round complexity can be seen to be proportional to the depth of the recursion tree of the Algorithm SLEEPINGMISRECURSIVE(k). We note that the activation of nodes proceeds in a depth-first, left-to-right fashion. Also, each node is awake only a constant number of rounds for possibly every level of the recursion tree. Thus, any node can be awake at most a number of rounds proportional to the depth of the recursion, which is $O(\log n)$ rounds. □

Lemma 10. *The worst-case round complexity of Algorithm 1 is $O(n^3)$.*

Proof. The worst-case round complexity of Algorithm SLEEPINGMISRECURSIVE(k) can be computed recursively. Let $T(k)$ be the worst-case number of rounds taken by SLEEPINGMISRECURSIVE(k). Then

$$T(k) \leq 2T(k-1) + 3 \text{ for } k > 0,$$

since SLEEPINGMISRECURSIVE(k) calls SLEEPINGMISRECURSIVE$(k-1)$ (at most) two times and takes 3 extra rounds. The base case is $T(0) = 0$. The above recurrence gives

$$T(k) \leq 3(2^k - 1).$$

The depth of the tree is $3 \log n$ (see Lines 6 and 9 of the pseudocode of Algorithm 1; also see Figure 2), and hence plugging this value of k gives the worst-case round complexity as $O(n^3)$. □

As an immediate corollary to Lemma 10, we have

Lemma 11. *The node-averaged round complexity of Algorithm 1 is $O(n^3)$.*

Combining Lemmas 8, 9, 10, and 11 yields the main result of this section.

Theorem 1. *There is a randomized Monte Carlo,[6] distributed MIS algorithm — described in Algorithm 1 — that is correct with high probability and has the following performance measures:*

---

[6]We recall that a Monte Carlo randomized algorithm is one that may sometimes produce an incorrect solution. In contrast, Las Vegas algorithms are randomized algorithms that always produce the correct solution. Please refer to [23, Section 1.2] for a detailed discussion on these two classes of randomized algorithms.

- $O(1)$ *node-averaged awake complexity (on expectation),*

- $O(\log n)$ *worst-case awake complexity (always),*

- $O(n^3)$ *node-averaged round complexity (always), and*

- $O(n^3)$ *worst-case round complexity (always).*

4.4   Improving the Worst-case Round Complexity

We now show how to reduce the worst-case round complexity significantly to polylogarithmic rounds while still keeping the node-averaged awake complexity to be constant and the worst-case awake complexity to be $O(\log n)$ rounds. In particular, we show the following theorem, which is the main result of this section.

Theorem 2. *There is a randomized, Monte-Carlo distributed MIS algorithm — described in Algorithm 2 — that is correct with high probability and has the following performance measures:*

- $O(1)$ *node-averaged awake complexity (on expectation),*

- $O(\log n)$ *worst-case awake complexity (with high probability),*

- $O(\log^{3.41} n)$ *node-averaged round complexity (with high probability), and*

- $O(\log^{3.41} n)$ *worst-case round complexity (with high probability).*

In the rest of the section, we prove Theorem 2.

Modifications to Algorithm SleepingMISRecursive(k).   The main idea is to truncate the recursion tree of Algorithm SLEEPINGMISRECURSIVE(k) earlier (see Figure 2) and use the *parallel/distributed randomized greedy Maximal Independent Set (MIS) algorithm* (described below) to solve the base cases.

The parallel/distributed randomized greedy Maximal Independent Set (MIS) algorithm [9, 7, 12].   The parallel/distributed randomized greedy MIS algorithm works as follows[12]. An *order* (also called *ranking*) of the vertices is chosen uniformly at random. Then, in each round, all vertices having the highest rank among their respective neighbors are added to the independent set and removed from the graph along with their neighbors. This process continues iteratively until the resulting graph is empty.

   The parallel/distributed randomized greedy MIS algorithm was first introduced by Coppersmith et al. [9]. They used this algorithm to find an MIS for $G(n, p)$ random graphs and showed that it runs in $O(\log^2 n)$ *expected* rounds and that this holds true for all values of $p \in [0, 1]$. Blelloch et al. [7] extended this result to general (arbitrary) graphs and showed an $O(\log^2 n)$ run-time with high probability. Finally, Fischer and Noever [12] improved the analysis further and showed that the parallel/distributed randomized greedy MIS algorithm ran in $O(\log n)$ rounds with high probability and also that this bound was tight. Fischer and Noever's work showed that the parallel/distributed randomized greedy MIS algorithm was (asymptotically) as fast as the famous algorithm by Luby [20, 2], where the random ranking of the nodes is chosen afresh for each iteration.

   It is a well-known fact (see, e.g., [7]) that the parallel/distributed randomized greedy MIS algorithm always produces what is known as the *lexicographically first* MIS [9], i.e., the same MIS output by the sequential greedy algorithm. In other words, it always produces the same result
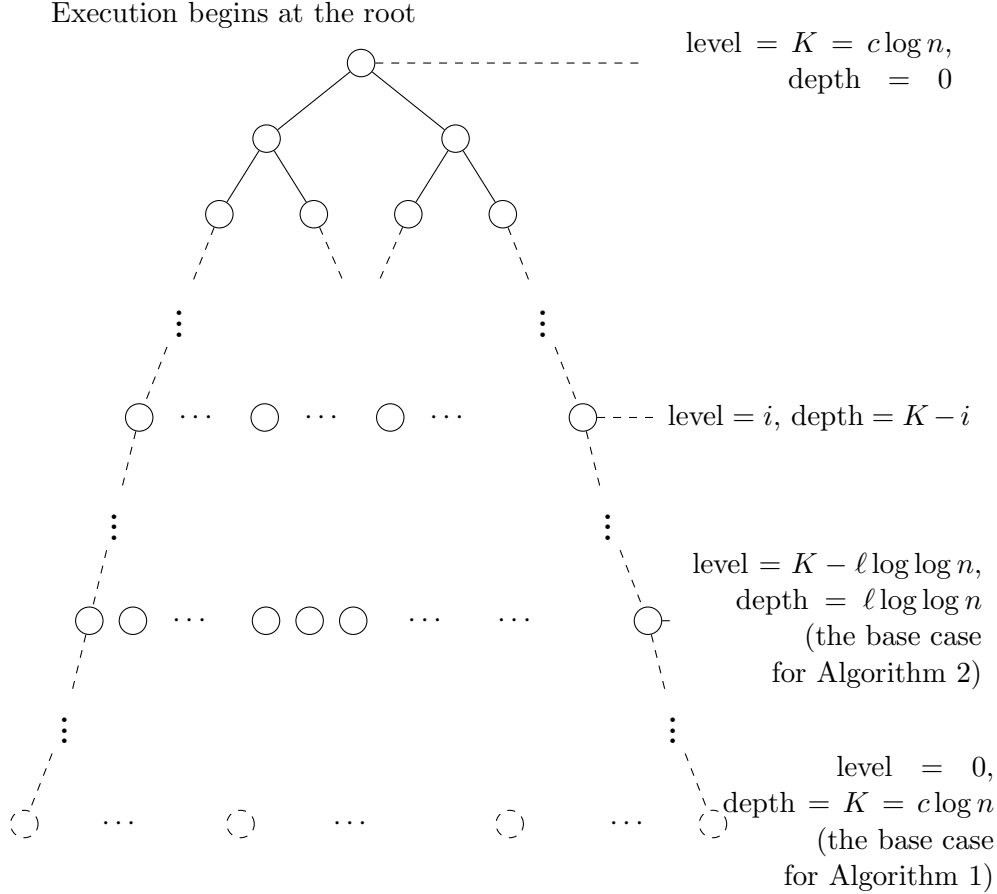
Figure 2: The recursion tree for our algorithms (Algorithms 1 and 2, respectively)

once an ordering of the vertices is fixed. We observe that Algorithm SLEEPINGMISRECURSIVE(k) produces a lexicographically first MIS as well. This follows as an immediate corollary to Lemma 4. Formally,

Corollary 1. *Algorithm* SLEEPINGMISRECURSIVE*(k) and the parallel/distributed randomized greedy MIS algorithm produce the same MIS.*

This means that at any given level i, the recursion tree corresponding to Algorithm SLEEPING-MISRECURSIVE(k) (see Figure 2) would look exactly the same — no matter which of Algorithm SLEEPINGMISRECURSIVE(k) or the parallel/distributed randomized greedy MIS algorithm was used to solve the subsequently lower levels. In particular, Lemma 7 would still hold if the ith level (for any $i \in [1, K]$) of the recursion tree was considered as the base case (instead of the 0th level as in Algorithm SLEEPINGMISRECURSIVE(k)) and the parallel/distributed randomized greedy MIS algorithm was used to solve each of the base cases. We note that for synchronization at higher levels of the recursion, we will require that the greedy algorithm runs for (exactly) $c \log n$ rounds for some large (but fixed) constant $c > 0$. The constant c is chosen such that the greedy algorithm finishes with high probability and the existence of this constant follows from the analysis of Fischer and Noever [12]. However, this makes the algorithm Monte-Carlo, since there is a small probability

that some base case might not finish within the required time which might affect the correctness of the overall algorithm.

We are now ready to present our modified algorithm which we call FAST-SLEEPINGMISRECURSIVE(k). The recursion tree for this algorithm is shown in Figure 2. Note that we slightly modify the parallel/distributed randomized greedy MIS algorithm as follows:

Let's call the main algorithm that invokes FAST-SLEEPINGMISRECURSIVE as FAST-SLEEPINGMIS (analogous to SLEEPINGMIS) and it is now invoked with $\ell \cdot \log \log n$, where

$$\ell \stackrel{\text{def}}{=} (\log_2 (\frac{4}{3}))^{-1} \tag{2}$$

Using Corollary 1, the correctness of this algorithm (FAST-SLEEPINGMIS) can be argued in a similar way as was done in the case of Algorithm SLEEPINGMIS (see Section 4.1). We now proceed to analyze the time complexity of Algorithm 2.

Lemma 12. *The expected node-average awake complexity of* FAST-SLEEPINGMIS*(K) with* $K = \ell \cdot \log \log n$ *(where* ℓ *is defined as in Equation 2) in the sleeping model is* $O(1)$.

Proof Sketch. As in the proof of Lemma 8, we argue that the cost of recursion is $O(1)$ per vertex, on expectation. In addition to the recursive calls, since the base cases here are non-trivial, each of them takes substantially more time, so we need to account for those too.

As has been argued before, Lemma 7 guarantees that the total (expected) number of network nodes participating at depth $\ell \cdot \log \log n$

$$= (\frac{3}{4})^{\ell \cdot \log \log n} \cdot n$$
$$= (\frac{1}{2})^{\log \log n} \cdot n \qquad \text{(substituting the value of } \ell \text{ from Equation 2)}$$
$$= \frac{n}{\log n}$$

By virtue of [12, Theorem 1.1], the parallel/distributed randomized greedy MIS algorithm takes — with high probability — $O(\log n)$ rounds to solve each instance of these base cases, i.e., recursion tree-nodes at depth $\ell \cdot \log \log n$. Thus each node participating at this level (depth $\ell \cdot \log \log n$ of the recursion tree) stays awake for $O(\log n)$ time.

Thus the total (total over all nodes) *expected* awake time at the base level (i.e., at depth $\ell \cdot \log \log n$ of the recursion tree) is equal to

$$= \text{number of network nodes participating at that level}$$
$$\times \text{awake-time of each of those participating nodes}$$
$$= \frac{n}{\log n} \times O(\log n) = O(n).$$

Dividing by n we get the additional awake complexity per node and that turns out to be $\frac{O(n)}{n} = O(1)$.

Thus the total *expected* node-average awake complexity of FAST-SLEEPINGMIS(k) is

$$= \text{node-average awake complexity contributed by the recursion calls}$$
$$+ \text{node average awake complexity contributed by the solutions to the base cases}$$
$$= O(1) + O(1) = O(1).$$

$\square$

Algorithm 2 The "Fast Sleeping MIS algorithm" executed by a node v

---

1: **function** SLEEPINGMIS($\ell \cdot \log \log n$)   ▷ ℓ is defined as $\ell \overset{\text{def}}{=} (\log_2 (\frac{4}{3}))^{-1}$; see Equation 2
2:     $v.\text{inMIS} \leftarrow$ unknown
3:     **for** i **in** $1, \ldots, \ell \cdot \log \log n$ **do**
4:         $v.X_i \leftarrow$ random bit that is 1 with probability $\frac{1}{2}$
5:     **end for**
6:     SLEEPINGMISRECURSIVE($\ell \cdot \log \log n$)   ▷ $\ell \cdot \log \log n$ is the recursion depth.
7: **end function**

8: **function** FAST-SLEEPINGMISRECURSIVE(k)
9:     **if** $k = 0$ **then**   ▷ base case
10:         DISTRIBUTEDGREEDYMIS   ▷ Use the parallel/distributed randomized greedy MIS
    algorithm (see [9, Section 2]) to solve the base cases. (For synchronization at higher levels of
    the recursion, we require that the greedy algorithm runs for (exactly) $c \log n$ rounds for some
    large (but fixed) constant $c > 0$.)
11:         **return**
12:     **end if**

13:     send message to every neighbor   ▷ first isolated node detection, 1 round
14:     **if** v receives no message **then**
15:         $v.\text{inMIS} \leftarrow$ true
16:     **end if**

17:     **if** $v.\text{inMIS} =$ unkown **and** $v.X_k = 1$ **then**   ▷ left recursion
18:         SLEEPINGMISRECURSIVE($k-1$)
19:     **else**
20:         sleep for a suitable number of rounds to achieve synchronization
21:     **end if**

22:     send value of $v.\text{inMIS}$ to every neighbor   ▷ synchronization step, 1 round
23:     **if** $v.\text{inMIS} =$ unknown **and** v receives message from neighbor w with $w.\text{inMIS} =$ true **then**
24:         $v.\text{inMIS} \leftarrow$ false   ▷ elimination
25:     **end if**

26:     send value of $v.\text{inMIS}$ to every neighbor   ▷ second isolated node detection, 1 round
27:     **if** $v.\text{inMIS} =$ unknown **and** v only receives messages from neighbors w with $w.\text{inMIS} =$ false
    then
28:         $v.\text{inMIS} \leftarrow$ true
29:     **end if**

30:     **if** $v.\text{inMIS} =$ unknown **then**   ▷ right recursion
31:         SLEEPINGMISRECURSIVE($k-1$)
32:     **else**
33:         sleep for a suitable number of rounds to achieve synchronization
34:     **end if**
35: **end function**

Lemma 13. *It holds with high probability that the worst-case round complexity of* FAST-SLEEPINGMIS*(K)* *with*

$$K = \ell \cdot \log \log n$$

is

$$O(\log^{\ell+1}(n)) = O(\log^{3.41}(n)).$$

Proof. First we observe that the number of leaves of the recursion tree = recursion tree-nodes at depth $\ell \cdot \log \log n$ is $2^{\ell \cdot \log \log n} = (\log n)^{\ell}$.

Next we compute the total time taken for executing one branch of the recursion tree — from the root to the leaf. That is given by

    time taken to traverse one branch of the recursion tree from the root to the leaf

    + time taken to solve the leaf, i.e., the base case of the recursion

    = depth of the recursion tree $\times O(1) + O(\log n)$ (by [12, Theorem 1.1])

The first term of the summation above equals $\ell \cdot \log \log n \times O(1) = O(\log \log n)$ and this holds deterministically. The second term of the summation above equals $O(\log n)$ and this holds with high probability (thanks to [12, Theorem 1.1]).

Substituting the values in the above equation, we get that the total time taken for executing one branch of the recursion tree — from the root to the leaf — is $O(\log \log n) + O(\log n) = O(\log n)$, and this fact holds with high probability.

Hence it holds with high probability that the total time taken for executing the entire recursion tree, i.e., for executing the algorithm

= time taken for executing one branch of the recursion tree $\times$ number of leaves of the recursion tree

$= O(\log n) \times (\log n)^{\ell} = O(\log^{\ell+1}(n))$

$= O(\log^{3.41}(n)).$ (substituting the value of ℓ from Equation 2)

$\square$

As an immediate corollary to Lemma 13, we have

Lemma 14. *It holds with high probability that the node-averaged round complexity of* FAST-SLEEPINGMIS*(K)* *with* $K = \ell \cdot \log \log n$ *is* $O(\log^{\ell+1}(n)) = O(\log^{3.41}(n))$.

Next we argue about the worst-case awake round complexity of FAST-SLEEPINGMIS(K), with $K = \ell \cdot \log \log n$. As argued earlier, the worst-case awake round complexity is proportional to the depth of the recursion tree of the Algorithm FAST-SLEEPINGMISRECURSIVE(K) plus the worst-case time taken by a node at the last level. The latter is $O(\log n)$ rounds w.h.p. [12, Theorem 1.1], since they execute the randomized greedy algorithm with random ranks. Thus the overall worst-case awake round complexity is the depth of the recursion which is $O(\log \log n)$ plus $O(\log n)$ for the last level. Hence we can state the following.

Lemma 15. *It holds with high probability that the worst-case awake round complexity of* FAST-SLEEPINGMIS*(K)* *with* $K = \ell \cdot \log \log n$ *is* $O(\log n)$.

Combining Lemmas 12, 13, 15, and 14 immediately gives rise to Theorem 2.

5   Conclusion

In this paper, we posit the sleeping model, and show that the fundamental MIS problem on general graphs can be solved in $O(1)$ rounds under node-averaged awake complexity, i.e., the average number of awake (non-sleeping) rounds taken by all nodes is $O(1)$. This is the first such result that we are aware of where we can obtain a constant round bound for node-averaged complexity for MIS *either* in the sleeping model or in the traditional model. An important open question is whether a similar bound can be shown in the traditional model; note that such a bound can be shown for $(\Delta + 1)$-coloring for general graphs (see Section 1.5). The sleeping model which is an extension of the traditional model can prove useful in designing distributed algorithms for various problems that are efficient with respect to the node-averaged measure; that would have crucial implications for saving nodes' resources (energy, etc.) in resource-constrained networks.

Acknowledgement

References

[1] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, 2013.

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

[3] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science*, FOCS '89, pages 364–369, October 1989.

[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikal Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science*, FOCS '19, pages 481–497, November 2019.

[5] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM*, 63(3), June 2016.

[6] Leonid Barenboim and Yaniv Tzur. Distributed symmetry-breaking with improved vertex-averaged complexity. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, ICDCN 19, pages 31–40, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA 12, pages 308–317, New York, NY, USA, 2012. Association for Computing Machinery.

[8] Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC 16, pages 217–226, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Don Coppersmith, Prabhakar Raghavan, and Martin Tompa. Parallel graph algorithms that are efficient on average. *Information and Computation*, 81(3):318–333, 1989.

[10] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1548–1557 vol.3, April 2001.

[11] Laurent Feuilloley. How long it takes for an ordinary node with an ordinary id to output? *Theoretical Computer Science*, 811:42–55, 2020. Special issue on Structural Information and Communication Complexity.

[12] Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 18, pages 2152–2160, USA, 2018. Society for Industrial and Applied Mathematics.

[13] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 16, pages 270–277, USA, 2016. Society for Industrial and Applied Mathematics.

[14] Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 19, pages 805–820, USA, 2019. Society for Industrial and Applied Mathematics.

[15] Richard M. Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM*, 32(4):762–773, October 1985.

[16] Maleq Khan, Gopal Pandurangan, and V. S. Anil Kumar. Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(1):124–139, January 2009.

[17] Valerie King, Cynthia A. Phillips, Jared Saia, and Maxwell Young. Sleeping on the job: Energy-efficient and robust broadcast for radio networks. *Algorithmica*, 61(3):518–554, 2011.

[18] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM*, 63(2), March 2016.

[19] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *Journal of the ACM*, 62(1):7:1–7:27, 2015. Invited paper from *ACM PODC* 2013.

[20] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[21] Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[22] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, USA, 2005.

[23] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[24] Chebiyyam Siva Ram Murthy and Balakrishnan Manoj. *Ad Hoc Wireless Networks: Architectures and Protocols*. Prentice Hall PTR, USA, 2004.

[25] Shreyas Pai, Gopal Pandurangan, Sriram V. Pemmaraju, Talal Riaz, and Peter Robinson. Symmetry breaking in the congest model: Time- and message-efficient algorithms for ruling sets. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 38:1–38:16, 2017.

[26] Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC 92, pages 581–592, New York, NY, USA, 1992. Association for Computing Machinery.

[27] Gopal Pandurangan. Distributed network algorithms. https://sites.google.com/site/gopalpandurangan/dna.

[28] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[29] Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. STOC, 2020.

[30] Qin Wang, Mark Hempstead, and Woodward Yang. A realistic power consumption model for wireless sensor network devices. In *The Third Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, volume 1 of *SECON '06*, pages 286–295, September 2006.

[31] Ou Yang and Wendi Heinzelman. An adaptive sensor sleeping solution based on sleeping multipath routing and duty-cycled mac protocols. *ACM Transactions on Sensor Networks*, 10(1), December 2013.

[32] Rong Zheng and Robin Kravets. On-demand power management for ad hoc networks. *Ad Hoc Networks*, 3(1):51–68, 2005.