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 $O(n \log n)$ worst-case rounds on general graphs (where $n$ is the number of nodes). Breaking the $O(n \log n)$ worst-case bound has been a longstanding open problem, while currently the best-known lower bound is $\Omega(\sqrt{\log n \log \log n})$ 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 $o(n \log n)$ 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 $1$ has $O(\log n)$-rounds worst-case awake complexity and $O(\log^3 n)$-rounds worst-case complexity.

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

CCS CONCEPTS
• Theory of computation → Distributed algorithms.

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

ACM Reference Format:
Soumyottam Chatterjee, Robert Gmyr, and Gopal Pandurangan. 2020. Sleeping is Efficient: MIS in $O(1)$-rounds Node-averaged Awake Complexity. In ACM Symposium on Principles of Distributed Computing (PODC ’20), August 3–7, 2020, Virtual Event, Italy. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3382734.3405718

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 [18] presented a randomized distributed algorithm for MIS, running on $n$-node graphs in $O(\log n)$ rounds with high probability. 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 $\Delta$, Ghaffari [12] presented a randomized MIS algorithm running in $O(\log \Delta + 2^{O(\sqrt{\log \log n})})$ rounds, improving over the algorithm of Barenboim et al. [5] that runs in $O(\log^2 \Delta + 2^{O(\sqrt{\log \log n})})$ rounds. It was further improved by Rozhno and Ghaffari to $O(\log \Delta + \text{poly}(\log n))$ rounds [26, 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 $\Omega(\min\{\frac{\log \Delta}{\log \log \Delta}, \sqrt{\frac{n}{\log \log n}}\})$ due to Kuhn et al. [17] that also applies to randomized algorithms. Thus, for

$1$Throughout, we use “with high probability (w.h.p)” to mean with probability at least $1 - \gamma^T$, for some constant $\gamma > 1$. 

99
example, say, when $\Delta = \Theta(2\sqrt{\log n})$, it follows that one cannot hope for algorithms faster than $\Omega(\frac{\log n}{\log\log n})$ rounds. Balliu et al. showed recently that one cannot hope for algorithms that run within $o(\Delta) + O(\log^2 n)$ 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 awake 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 [9, 29]. Thus, even though there might be no messages exchanged 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., [9, 16, 27–29]). 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 [21, 28, 29]).

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., [15] 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 [25], 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, 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 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 [25]. We note that our algorithms also, obviously apply to the Local model, another well-studied model [25] where there is no restriction on the size of the messages sent per edge per round.

Some distributed MIS algorithms (e.g., [5, 12, 23]) assume the Local model, whereas others work also in the Congest model [13, 18, 22].

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

\[^3\text{Making this assumption is not essential, but it simplifies presentation.}\]
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 does it do 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 \(\frac{1}{V} \sum_{v \in V} a_v\).

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 \(\frac{1}{V} \sum_{v \in V} A_v\), i.e., the average of the \(A_v\) random variables. Then the expected node-averaged awake complexity of the randomized algorithm is

\[
E[A] = E\left[\frac{1}{V} \sum_{v \in V} A_v\right] = \frac{1}{V} \sum_{v \in V} E[A_v].
\]

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).

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 \(\Omega(\min\{\log \Delta, \frac{\log \log n}{\log \log \log n}\})\) 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 [18, 24]) 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. [14] as well as known deterministic MIS algorithms [3, 23] (see also [5, 26]). The algorithm of [12] (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 [12], 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 \(\epsilon\). 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 \((a + \log^* n)\) rounds under node-averaged complexity deterministically, where \(a\) is the arboricity of the graph. It is an 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 \(O(\log^{1.41} n)\) 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 a step towards understanding whether a \(O(1)\)-round node-averaged algorithm is possible in the traditional model (without sleeping).

1.5 Comparison with Related Work

The notion of node-averaged (or vertex-averaged) complexity for the traditional model was proposed by Feuilloley [10] 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].

\(^3\) 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 [21, 28, 29].

\(^4\) 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.
In Feuilloley’s notion [10], 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 [10] studies the average time complexity of leader election and coloring algorithms on cycles and other specific sparse graphs.

Following the work of Feuilloley [10], 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.

There is also another important distinction between the algorithms of this paper and those of two works discussed above of [6, 10]: some of the algorithms in the above works assume the Local model (unbounded messages), whereas Congest model (small-sized messages) is assumed here.

The work of King et al. [16] 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.

|                      | Prior MIS algorithms (e.g., Luby’s [2, 18], CRT [7, 8, 11], etc.) | Our algorithms |
|----------------------|---------------------------------------------------------------|----------------|
| Node-averaged awake complexity                             | Not applicable.                                                | \(O(1)\)       |
| Worst-case awake complexity                                | Not applicable.                                                | \(O(\log n)\)  |
| Worst-case round complexity                                | \(O(\log n)\)                                                  | \(O(\log n)\)  |
| Node-averaged round complexity                              | The best known bounds are \(O(\log n)\).                        | \(O(\log n)\), \(O(\log^{3.41} n)\) |

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

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-averaged 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 (in 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 [19] 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).

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. 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 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 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 $\left(\frac{3}{4}\right)^i$ of the 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 $\left(\frac{3}{4}\right)\cdot n$. 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:

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

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 ($O(\log^{341} n)$) 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 $\text{SleepingMIS}$ at the same time. The function $\text{SleepingMIS}$ 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 $\text{SleepingMISRecursive}$. 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 $\text{SleepingMISRecursive}$ 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. $\text{SleepingMISRecursive}$ takes an integer parameter $k$. This parameter starts with $k = K = \lfloor 3\log n \rfloor$ 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.

#### Algorithm 1 The “Sleeping MIS algorithm” executed by a node $v$

```plaintext
1. function SleepingMIS(K)
2.   v.inMIS = unknown
3.   for i in 1, ..., K do
4.       v.X_i = random bit that is 1 with probability \( \frac{1}{2} \)
5.       SleepingMISRecursive(K) \( \rightarrow \) K is the recursion depth.
6.   function SleepingMISRecursive(k)
7.     if k = 0 then \( \rightarrow \) base case
8.       v.inMIS = true
9.       return
10.    if v receives no message then
11.       v.inMIS = true
12.     if v.inMIS = unknown and v.X_k = 1 then \( \rightarrow \) left recursion
13.     SleepingMISRecursive(k - 1)
14.    else
15.       sleep for \( T(k - 1) = 3(2^{k-1} - 1) \) rounds \( \rightarrow T(k - 1) \) is the duration of the recursive call in Line 14.
16.     send value of v.inMIS to every neighbor \( \rightarrow \) synchronization step, 1 round
17.     if v.inMIS = unknown and v receives message from neighbor w with w.inMIS = true then
18.       v.inMIS = false \( \rightarrow \) elimination
19.       send value of v.inMIS to every neighbor \( \rightarrow \) second isolated node detection, 1 round
20.     if v.inMIS = unknown and v only receives messages from neighbors w with w.inMIS = false then
21.       v.inMIS = true
22.    else
23.       sleep for \( T(k - 1) = 3(2^{k-1} - 1) \) rounds \( \rightarrow T(k - 1) \) is the duration of the recursive call in Line 24
24.       SleepingMISRecursive(k - 1)
25.     end if
26. end if
27. end for
28. end if
29. v.inMIS = true
30. return
31. end function
```

### Algorithm 2 SleepingMISRecursive(k)

```plaintext
1. function SleepingMISRecursive(k)
2.   if k = 0 then \( \rightarrow \) base case
3.     v.inMIS = true
4.     return
5.   if v receives no message then
6.     v.inMIS = true
7.     if v.inMIS = unknown and v.X_k = 1 then \( \rightarrow \) left recursion
8.     SleepingMISRecursive(k - 1)
9.   else
10.    sleep for \( T(k - 1) = 3(2^{k-1} - 1) \) rounds \( \rightarrow T(k - 1) \) is the duration of the recursive call in Line 14.
11.    send value of v.inMIS to every neighbor \( \rightarrow \) synchronization step, 1 round
12.    if v.inMIS = unknown and v receives message from neighbor w with w.inMIS = true then
13.      v.inMIS = false \( \rightarrow \) elimination
14.      send value of v.inMIS to every neighbor \( \rightarrow \) second isolated node detection, 1 round
15.    if v.inMIS = unknown and v only receives messages from neighbors w with w.inMIS = false then
16.      v.inMIS = true
17.      if v.inMIS = unknown then \( \rightarrow \) right recursion
18.      SleepingMISRecursive(k - 1)
19.      else
20.       sleep for \( T(k - 1) = 3(2^{k-1} - 1) \) rounds \( \rightarrow T(k - 1) \) is the duration of the recursive call in Line 24
21. end if
22. end if
23. end if
24. end function
```
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 7 – 9) 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 10–12) 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 neighborhood in $G[U]$.

The third part (Lines 13–16) 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. 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 16. Thereby, all nodes in $U$ start executing the next part of the function (which begins in Line 17) 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 17–19) the elimination step. In this part, every node $v \in U$ with $v.\inMIS = unknown$ 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 20–22) 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 23–26) 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) so that messages can be exchanged between neighbors.

4 ANALYSIS

We now turn to the analysis of Algorithm 1.

Lemma 1. It holds with high probability that on an $n$-node graph $G$, the set $M$ computed by the algorithm $SleepingMIS$ is an MIS.

4.1 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. $E[|L| + |U|] \leq \frac{|U|}{4}$.

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, $v$ participates 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). $E[|R| + |U|] \leq \frac{|U|}{4}$.

Together, Lemmas 2 and 3 imply that the expected number of nodes that participate in either of the recursive calls is $E[|L| + |R| + |U|] \leq \frac{1}{4} |U|$.

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 $r_k(v) = (-1)$ 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 $u \in U$ is such that every $w \in N_G[U \setminus \{v\}]$ with $r_k(w) > r_k(v)$ sets $w.\inMIS$ to false at some point during the algorithm, then $v$ joins the MIS.
Proof Sketch. The main idea is to use induction on \( k \) and then do a case-by-case analysis depending on the value of the variable \( v.X_k \).

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. \( \square \)

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 \( v_i \) joins the MIS. Since \( v_i.X_k = 1 \), \( v_i \) must set \( v_i.inMIS \) to true in the first isolated node detection or the left recursive call. Therefore, Statement 1 holds.

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 \( v_i.X_k = 1 \). 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. \( \square \)

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 \( \Pr[v \in R] \leq 1/4 \), 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
\[
\Pr \left( v \in R \right) = \Pr \left( v \in R \mid v.\mathcal{X}_k = 1 \right) \cdot \Pr \left( v.\mathcal{X}_k = 1 \right) + \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \right) \cdot \Pr \left( v.\mathcal{X}_k = 0 \right).
\]
By definition, it holds \( \Pr \left( v \in R \mid v.\mathcal{X}_k = 1 \right) = 0 \) and \( \Pr \left( v.\mathcal{X}_k = 0 \right) = 1/2 \). Therefore, we have
\[
\Pr \left( v \in R \right) = \frac{1}{2} \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \right).
\]

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
\[
\Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \right) = \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \land E \right) \cdot \Pr \left( v.\mathcal{X}_k = 0 \land E \right) + \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \land \lnot E \right) \cdot \Pr \left( v.\mathcal{X}_k = 0 \land \lnot E \right).
\]

We first determine the probability \( \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \land E \right) \). If \( v.\mathcal{X}_k = 0 \), \( v \) 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 \( w.\text{inMIS} \) to false before the second isolated node detection. Hence, \( v \) sets \( v.\text{inMIS} \) to true during the second isolated node detection. This implies that \( \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \land E \right) = 0 \).

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

By combining the equations above, we get \( \Pr \left( v \in R \mid v.\mathcal{X}_k = 0 \right) \leq 1/2 \) and, therefore, \( \Pr \left( v \in R \right) \leq 1/4 \).

**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 \( 0 \leq i \leq K \cdot E \left[ Z_{K-i} \right] \) \( \leq \left( \frac{3}{4} \right)^i \cdot n \).

**Proof.** We show the lemma by induction on \( i \). For the induction base \( i = 0 \), 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 consider the equation holds for an \( i \) such that \( 0 \leq i < K \) 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
\[
Z_{K-i} = \sum_{j=1}^{m} |U_j|.
\]
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
\[
Z_{K-(i+1)} = \sum_{j=1}^{m} (|L_j| + |R_j|).
\]
This implies that
\[
E[ Z_{K-(i+1)} | U_1, \ldots, U_m ] = \sum_{j=1}^{m} E[ |L_j| + |R_j| | U_j ] \leq \frac{3}{4} \sum_{j=1}^{m} |U_j| = \frac{3}{4} \cdot Z_{K-i},
\]
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
\[
E[Z_{K-(i+1)}] \leq \frac{3}{4} \cdot E[Z_{K-i}] \leq \left( \frac{3}{4} \right)^{i+1} \cdot n.
\]

**□**

**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 \( \frac{E[C]}{n} \).

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 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
\[
E[C] = O(1) \cdot \sum_{k=0}^{K} E[Z_k] = O(1) \cdot \sum_{k=0}^{K} E[Z_{K-k}]
\]
\[
\leq O(1) \cdot \sum_{k=0}^{K} \left( \frac{3}{4} \right)^k \cdot n \leq O(n) \cdot \sum_{i=0}^{\infty} \left( \frac{3}{4} \right)^i = O(n).
\]
Therefore, the expected node-average awake complexity of the algorithm is \( \frac{E[C]}{n} = O(1) \).
Combining the results corresponding to the different complexity measures 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:

- \(O(1)\) 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).

**4.2 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).

**Proof sketch:**

To prove Theorem 2, we first show the correctness of the algorithm by using Corollary 1. The correctness of the algorithm follows from the analysis of Fischer and Noever [11]. 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 1. Note that we slightly modify Greedy-MIS as follows:

**Algorithm 2** The Fast-SleepingMISRecursive(k) algorithm.

1. Proceed along the same lines as in Algorithm SleepingMISRecursive(k) until the recursion tree reaches depth \(\ell \cdot \log \log n\), where

\[
\ell \defeq \log_2 \left(\frac{4}{3}\right)^{-1}
\]

2. As shown in the analysis, the above value of \(\ell\) is chosen so that it optimizes both the node-averaged awake complexity and the worst-case round complexity.

Using Corollary 1, the correctness of the Fast-SleepingMIS algorithm can be argued in a similar way as was done in the case of Algorithm SleepingMIS. The main idea is to analyze the recursion tree (see Figure 1) to prove the various clauses of 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.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their many insightful comments.

REFERENCES

[1] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haerupler, and Fabian Kuhn. 2013. Beeping a maximal independent set. Distributed Computing 26, 4 (2013), 195–208. https://doi.org/10.1007/s00444-012-0175-7
[2] Noga Alon, László Babai, and Alon Itai. 1986. A fast and simple randomized parallel algorithm for the maximal independent set problem. Journal of Algorithms 7, 4 (1986), 567–583.
[3] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. 1989. Network decomposition and locality in distributed computation (FOCS). 364–369.
[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Fabian Kuhn. 2013. Beeping a maximal independent set. Distributed Computing: A Locality-sensitive Approach. Society for Industrial and Applied Mathematics.
[5] David Peleg. 2000. Distributed Network Algorithms. Prentice Hall PTR, USA. 9781107154889 (the base case for Algorithm 2)
[6] Valerie King, Cynthia A. Phillips, Jared Saia, and Maxwell Young. 2011. Sleeping on the Job: Energy-Efficient and Robust Broadcast for Radio Networks. Algorithmica 61, 3 (2011), 518–554.
[7] Fabian Kuhn, Thomas Mosbruger, and Roger Wattenhofer. 2016. Local Computation: Lower and Upper Bounds. Journal of the ACM 63, 2, Article Article 17 (March 2016), 44 pages.
[8] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. 15, 4 (1986), 1036–1053.
[9] Michael Luby. 1993. Removing randomness in parallel computation without a processor penalty. J. Comput. System Sci. 47, 2 (1993), 250–286.
[10] Michael Mitzenmacher and Eli Upfal. 2017. Probability and Computing: Randomized Algorithms and Probabilistic Analysis (2nd ed.). Cambridge University Press, Cambridge CB2 8BS, United Kingdom. http://www.cambridge.org/9781107154889
[11] Chebiryam Siva Ram Murthy and Balakrishnan Manoj. 2004. Ad Hoc Wireless Networks: Architectures and Protocols. Prentice Hall PTR, USA.
[12] Maleq Khan, Gopal Pandurangan, and V. S. Anil Kumar. 2009. Distributed Algorithms for Constructing Approximate Minimum Spanning Trees in Wireless Sensor Networks. IEEE Transactions on Parallel and Distributed Systems 20, 1 (January 2009), 124–139.
[13] Richard M. Karp and Avi Wigderson. 1985. A Fast Parallel Algorithm for the Maximal Independent Set Problem. Journal of the ACM 32, 4 (October 1985), 762–773.
[14] Mohsen Ghaffari. 2016. An Improved Distributed Algorithm for Maximal Independent Set (SODA). USA, 270–277.
[15] Mohsen Ghaffari. 2019. Distributed Maximal Independent Set Using Small Messages (SODA). USA, 805–820.
[16] Manuela Fischer and Andreas Noever. 2018. Tight Analysis of Parallel Randomized Greedy MIS (SODA). USA, 2152–2160.
[17] David Peleg. 2000. Distributed Network Algorithms. Prentice Hall PTR, USA. 9781107154889 (the base case for Algorithm 2)
[18] Ou Yang and Wendi Heinzelman. 2013. An Adaptive Sensor Sleeping Solution Based on Sleeping Multipath Routing and Duty-Cycled MAC Protocols. ACM Transactions on Sensor Networks 10, 1, Article Article 10 (December 2013), 30 pages.
[19] Rong Zheng and Robin Kravets. 2005. On-demand power management for ad hoc networks. Ad Hoc Networks 3, 1 (2005), 51–68.