A Time-Optimal Randomized Parallel Algorithm for MIS

Mohsen Ghaffari*
ETH Zurich
ghaffari@inf.ethz.ch

Bernhard Haeupler†
Carnegie Mellon University
haeupler@cs.cmu.edu

Abstract

We present a randomized parallel algorithm, in the Exclusive-Read Exclusive-Write (EREW) PRAM model, that computes a Maximal Independent Set (MIS) in $O(\log n)$ time and using $O(m \log^2 n)$ work, with high probability. Thus, $\mathsf{MIS} \in \mathsf{RNC}^1$. This time complexity is optimal and it improves on the celebrated $O(\log^2 n)$ time algorithms of Luby [STOC'85] and Alon, Babai, and Itai [JALG'86], which had remained the state of the art for the past 35 years.

1 Introduction

MIS. The Maximal Independent Set (MIS) problem is one of the classic graph problems studied in parallel algorithms. Given an undirected graph, the objective is to select a subset S of vertices such that no two vertices $v, u \in S$ are adjacent (*independence*) and no vertex $w \notin S$ can be added to S without violating the independence condition (*maximality*). The problem admits a trivial sequential algorithm: start with $S = \emptyset$, examine the vertices one by one and add to S if possible.

PRAM. We work with the Parallel Random Access Machine (PRAM) model, which is the parallel analogue of the sequential RAM model. In this model, a (polynomially bounded) number of RAM model processors can write to and read from some shared memory. With regard to the conflicts caused by potential simultaneous read/write accesses to to the same memory cell, there are four variants to the model. The weakest variant is Exclusive-Read Exclusive-Write (EREW), in which every memory cell can be read or written by only one processor at a time (i.e., the algorithm should ensure no simultaneous read or writes). The strongest variant is Concurrent-Read Concurrent-Write (CRCW), for which multiple simultaneous writes to (and reads from) the same memory location are allowed. There are also intermediate models CREW and ERCW, defined as the name suggests. In the variants with concurrent writes, the model comes with a prescribed resolution scheme, e.g., an arbitrary write might take effect. As an analogy, EREW algorithms can be seen as Boolean circuits of bounded fan-in and fan-out, while the other variants, when turned to a circuit, would need unbounded fan-in and/or fan-out. See Section 1.4 for more on the PRAM variants.

Similar to most of the prior work on MIS, in this paper, our focus will be on the weakest, and arguably cleanest and most realistic, variant EREW. We also briefly comment on some prior work in the stronger CRCW variant.

1.1 State of the Art

The MIS problem was first pointed out by Valiant in 1982 [Val82] as an interesting problem for parallel computation (in contrast with its trivial sequential solution). It is known that computing the lexicographically-first MIS, i.e., computing the MIS resulting from the aforementioned centralized process when we examine the vertices in a fixed given order, is P-complete [Coo85, GHR+95]. Hence, the focus in parallel algorithms has been on computing an arbitrary MIS.

Wigderson and Karp [KW84] gave the first polylogarithmic time parallel algorithm for MIS: they presented an $O(\log^3 n)$-time randomized parallel algorithm in the EREW model, as well as an $O(\log^4 n)$-time deterministic EREW PRAM algorithm. Shortly after, Luby [Lub85] and independently Alon, Babai, and Itai [ABI86] presented their celebrated randomized MIS algorithm which compute an MIS in $O(\log^2 n)$ time in EREW PRAM. This algorithm has been covered in many classes and textbooks on randomized or parallel algorithms. They also explained how to derandomize this to an $O(\log^2 n)$-time deterministic algorithm (with polynomially more processors).

These two papers also observe that a modified version of these algorithms runs in $O(\log n)$ time in the more powerful CRCW variant of PRAM. This

*Supported in part by funding from the European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (grant agreement No. 853109), and the Swiss National Foundation (project grant 200021-184735).

†Supported in part by NSF grants CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808 and a Sloan Research Fellowship.

modified algorithm assumes that up to n processors can simultaneously read the same register, and more crucially, up to n processors can write to the same register simultaneously (in which case an arbitrary write takes place).

Since the work of Luby [Lub85] and Alon, Babai, Itai [ABI86], the question of obtaining an algorithm with faster than $O(\log^2 n)$ time in the weaker EREW variant (or even in the intermediate CREW and ERCW variants) remained open. It was also asked explicitly by Luby [Lub85,Lub86] whether one can "*find a faster MIS algorithm*".

1.2 Our Contribution In this paper, we settle the randomized part of the time-complexity by presenting a randomized EREW PRAM algorithm for MIS with the optimal $O(\log n)$ time complexity:

THEOREM 1.1. *There is a randomized EREW PRAM algorithm that, for any n-node m-edge graph, computes an MIS in $O(\log n)$ time and using $O(m \log^2 n)$ work, with high probability.*

We note that even "reading" or "informing" all the neighbors of a single node with polynomially large degree requires $\Omega(\log n)$ time on an EREW PRAM: Concretely, computing a logical OR of the (up to $n-1$) neighbors, to see if any of them is in the MIS, requires $\Omega(\log n)$ time, even for a CREW PRAM [CDR86]. Moreover, informing the (up to $n-1$) neighbors of whether this node is the MIS or not (or any other similar information, such as the identifier of the node or any decision taken with regard to the node) requires making $\Theta(n)$ copies of the content of a single memory cell which also requires $\Omega(\log n)$ time on an EREW PRAM [BKK94]. Hence, the $O(\log n)$ complexity of Theorem 1.1 appears[1] to be the best possible, for an EREW PRAM.

We are also hopeful that the randomized result provided in Theorem 1.1 might pave the way for a deterministic $O(\log n)$-time EREW PRAM algorithm, which would prove that $\mathsf{MIS} \in \mathsf{NC}^1$.

1.3 Our Method in a Nutshell Next, we briefly recall Luby's algorithm [Lub85] and provide some intuition for why this algorithm, and many others, face a significant barrier when trying to achieve a time complexity below $\Theta(\log^2 n)$ on an EREW PRAM. Then, we present ideas for how to break this barrier and achieve an $O(\log n)$ time complexity.

[1] A formal proof would have to deal with more nuanced details of the EREW PRAM model, similar to [BKK94].

The $\Omega(\log^2 n)$ Barrier For Luby's Algorithm. Luby's algorithm involves $\Theta(\log n)$ iterations, where each iteration has two steps: in the first step, each node draws a random number in $[0,1]$ and the node joins the MIS if its number is strictly smaller than that of all of its neighbors. In the second step, we perform a clean up by removing all nodes that joined the MIS and their neighbors. In the EREW PRAM model (or even in CREW), one node needs to determine whether the minimum of the numbers of the neighbors is less than its own or not. As the number of neighbors can be up to $n-1$, this requires $\Omega(\log n)$ time on an EREW PRAM. In fact, even computing the OR of say $\sqrt{n}$ numbers — which is a special case of computing the minimum in just binary numbers — requires $\Omega(\log n)$ time in the (more powerful) CREW PRAM model, by a result of Cook et al. [CDR86]. This explains the overall $\Theta(\log^2 n)$ complexity of Luby's algorithm on an EREW PRAM.

We remark that another variant of the algorithm [Lub85, ABI86] works in $O(\log n)$ iterations of a slightly different process: per iteration, we mark each node v, which has degree d_v, with probability $1/(2d_v)$, and we then add a marked node v to MIS if and only if no neighbor with higher degree (and ID, if there is a tie on degrees) is marked. The clean up step is as before. This variant also faces the same $\Theta(\log^2 n)$ complexity barrier as each step of checking whether any of the $\Theta(n)$ neighbors is marked or not is exactly computing the Boolean OR of up to $\Theta(n)$ variables, which, as discussed, needs $\Omega(\log n)$ time even on a CREW PRAM [CDR86].

A High-Level Intuition of Our Approach: One of the logarithmic factors in the $O(\log^2 n)$ complexity of Luby's algorithm comes from the degrees: Scanning d neighbors, and concretely computing a function such as minimum or logical OR of their values, costs $O(\log d)$ time. To circumvent this overhead, we devise an algorithm where, effectively and in a very informal sense, each iteration works on a constant-degree subgraph, which was determined before the iteration. But for this, each iterations has to do much more *preparation* work for each of the future iterations.

Our parallel algorithm builds on the distributed algorithm of Ghaffari [Gha16]. This distributed algorithm also has $O(\log n)$ iterations, similar to Luby's, and faithfully implementing it would also lead to an $O(\log^2 n)$ EREW PRAM algorithm, for the same reasons. However, with some modifications that we describe, we can ensure that the behavior of the algorithm is "smooth" in the following sense: the probability of each node v attempting to join the MIS changes by only a constant factor from each iteration to the next. Hence, in itera-

tion t, we have a prediction of the attempt probabilities of any future iteration $t' \geq t$, where the prediction is sharp up to a $2^{O(t'-t)}$ factor. This prediction allows us to gradually narrow down the set of relevant neighbors, as we get closer and closer to iteration t'.

Concretely, at the very beginning of the algorithm, we *prepare* a *set of relevant neighbors* for each future iteration. We show that these sets of relevant neighbors can be refined gradually over time to make them smaller and smaller, in the following sense: in each iteration t, the sets of any future iteration $t' \geq t+1$ have size $2^{O(t'-t)}$. This intuitive description ignores some technical aspects. Once we arrive at iteration t', the set size is constant. Moreover, this size guarantee is such that we can maintain it as t gets closer to t'. For instance, once time proceeds from iteration t to iteration $t+(t'-t)/2$, we have time proportional to $(t'-t)/2$ iterations to refine the set of $2^{O(t'-t)}$ relevant neighbors. Notice that, as a very simple example, computing the OR of $2^{O(t'-t)}$ values can be easily done in $O(t'-t)$ time, i.e., in at most $(t'-t)/2$ iterations by setting the length of each iteration a large enough constant time.

Turning the above intuitive description to a formal algorithm relies on several properties of the (modified) algorithm and its analysis. There also a number of implementation issues to make the algorithm suitable for an EREW PRAM. These are deferred to the technical section.

Finally, we note that our work is in part inspired by the ideas that were used by Ghaffari and Uitto [GU19] to provide a $\tilde{O}(\sqrt{\log n})$-round algorithm in the Massively Parallel Computation (MPC) model with sublinear memory per machine. In particular, they partitioned the $O(\log n)$ rounds of the distributed algorithm into $\tilde{O}(\sqrt{\log n})$ phases, and showed that each phase can be performed in $O(1)$ rounds of the MPC model, by gathering a certain $\tilde{O}(\sqrt{\log n})$-hop topology, which fits in the memory of one machine because one can effectively reduce the degrees in one phase to $2^{\tilde{O}(\sqrt{\log n})}$. Although details would have to be checked, we think that one could probably extend their method to the EREW PRAM model, but the resulting time complexity would be $O(\log^{1.5} n\sqrt{\log\log n})$, which is far higher than our optimal $O(\log n)$ bound. In our algorithm, we have to perform preparation work for each of the future iterations separately, all at the same time (with dependencies between them), and we make sure that once we arrive at any given iteration, each node has to examine only constant many neighbors.

1.4 Variants of PRAM and Other Related Work There is an abundance of literature on comparing the power of PRAM variants. For instance, it is well-known that a CRCW PRAM can be an $\Omega(\log n)$ factor faster than an EREW PRAM for some problems. See [Fic93] for a survey of such comparisons, and the textbook of Jaja [JaJ92] for more details on model variants. We also note that, while EREW is clearly a weaker model, there are also extensive discussions in the literature to indicate that EREW algorithms are "more practical", in the sense that they can be adapted to other more realistic models–see, e.g., the discussions in [CHL01, Section 1].

Perhaps because of this, there has been extensive work on PRAM algorithms with exclusive write, and especially EREW PRAM algorithms, for various problems, even though faster CRCW PRAM algorithms were known. Below, we briefly review the developments for another classic graph problem, connectivity.

It was long known, see e.g. [AS87, CV86], that in the CRCW model, one can solve both the connectivity problem, i.e., identifying connected components in an undirected graph, and its close relative minimum spanning tree in $O(\log n)$ time, deterministically. In contrast, when requiring the writes to be exclusive, the best known algorithm needed $O(\log^2 n)$ time [CLC82, HCS79] and this remained the state of the art for a while until the first breakthrough of Johnson and Metaxas [JM91, JM92], with an $O(\log^{1.5} n)$ time complexity for both problems. A work of Nisan, Szemeredi, and Wigderson [NSW92] implied another algorithm with the same time complexity. Shortly after, Karger, Nisan, and Parnas [KNP92] gave a randomized EREW algorithm for connectivity with $O(\log n \log\log n)$ time complexity. Chong and Lam [CL95] provided[2] a deterministic EREW PRAM algorithm with the same $O(\log n \log\log n)$ complexity for connectivity. This was later extended to an MST algorithm by Chong [Cho96]. Finally, Chong, Han, and Lam [CHL01] settled the problem by providing $O(\log n)$ time deterministic EREW algorithms for the connectivity and minimum spanning tree problems.

2 Our Algorithm

In this section, we present our algorithm that proves Theorem 1.1. We first describe a vanilla version of the algorithm, which is a modified variant of the distributed algorithm of Ghaffari [Gha16], with some additional *smoothness* properties that we will leverage in our parallel algorithm. We then intuitively discuss some key ideas in our parallel algorithm. After that, we proceed

[2] This $O(\log n \log\log n)$-time EREW algorithm later led to an $O(\log n \log\log n)$-space algorithm by Trifonov [Tri05], simultaneous with Regingold's $O(\log n)$-space connectivity breakthrough [Rei05].

to presenting the outline of our parallel algorithm, which, roughly speaking, runs this smoothed distributed algorithm, by predicting and gradually preparing for each future round. We then finish by discussing how we implement this algorithm in $O(\log n)$ time on an EREW PRAM, and its analysis.

2.1 Warm Up: A Vanilla Version of the Algorithm

We start with an empty set $S \subseteq V$ of vertices, which will gradually grow and eventually become the desired maximal independent set, while always remaining an independent set. Each node v starts with a probability parameter $p_0(v) = 2^{-\lceil \log \Delta \rceil}$. Then, the algorithm has $O(\log n)$ identical iterations, where per iteration $t \geq 1$, each node v does as follows:

- If node v is in the set S, set $p_{t+1}(v) = 1$. If node v has a neighbor in the set S, set $p_{t+1}(v) = p_t/2$. Otherwise, if $d_t(v) = \sum_{u \in N(v)} p_t(u) \geq 10$, set $p_{t+1}(v) = p_t/2$, and if $d_t(v) < 10$, set $p_{t+1}(v) = \min\{2p_t, 1/2\}$.
- If $d_t(v) < 10$, then node v attempts to join the set S by marking itself with probability $p_t(v)$. Nodes in S also mark themselves (deterministically). If $v \notin S$ is marked and has no marked neighbor in this iteration, then v joins S and also informs its neighbors that they have a neighbor in the set S.

Next, we provide a proof sketch to show that this algorithm computes an MIS in $O(\log n)$ iterations, with high probability. The intention is to show the key properties used in the analysis; some lower-order details are omitted here but provided later in the analysis for the actual algorithm.

THEOREM 2.1. *After $O(\log n)$ iterations, the set S computed in the above algorithm is a maximal independent set, with high probability.*

Proof. [Proof Sketch] Fix an arbitrary node v. We argue that within $T = C \log n$ iterations, where C is a sufficiently large constant, there are at least $0.01T$ iterations which are *good* for v, in the following sense: either v is in S or has a neighbor in S already or in this iteration, either (I) node v has a constant probability of joining the set S, or (II) there is a constant probability that a neighbor of v joins the set S. The guarantee for the existence of these good rounds is deterministic. Moreover, these probabilities are based only on the randomness used in that iteration, and importantly independent of what happened in the previous iterations. Therefore, within $0.01T$ good iterations, the probability of node v not having joined the set S or having a neighbor in S is at most $1/n^2$. Recall that v was an arbitrarily chosen node. Thus, by union bound, we conclude that within $O(\log n)$ iterations, with probability at least $1 - 1/n$, each node is either in S or has a neighbor in S. Therefore, set S is an MIS, with high probability.

What remains is to prove that each node v has at least $0.01T$ good iterations. Consider each iteration t in which v is not already in S or adjacent to S. We call this iteration *heavy for* v if $d_t(v) = \sum_{u \in N(v)} p_t(u) \geq 5$. In such an iteration, there are two possibilities: (A) At least $\frac{1}{10} d_t(v)$ of this summation comes from neighbors u that are not neighboring a node in S and have $d_t(u) < 10$. (B) Strictly greater than $\frac{9}{10} d_t(v)$ of this summation comes from neighbors u that either have a neighbor in S or have $d_t(u) \geq 10$. We can see that case (A) is a good iteration for node v as in such an iteration, there is a constant probability that a neighbor of v gets marked, has no marked or S neighbor, and thus joins S. If we have at least $0.01T$ heavy iterations in case (A), we have $0.01T$ good iterations for v and thus, we are done. So, let us assume otherwise.

In any heavy iteration that is in case (B), we will have the summation $d_t(v)$ change as $d_{t+1}(v) \leq \frac{9}{10} d_t(v)/2 + 2\frac{1}{10} d_t(v) \leq \frac{2}{3} d_t(v)$. This is because neighbors u that are adjacent to S or have $d_t(u) \geq 10$ set $p_{t+1}(u) = p_t(u)/2$. Thus, in any heavy iteration that is not a good iteration, we have $d_{t+1}(v) \leq (2/3) d_t(v)$. We start with $d_0(v) \leq 1$. We have at most $0.01T$ heavy iterations that are good for v, during each of which we have $d_{t+1}(v) \leq 2 d_t(v)$. Every two times of decrease by a $2/3$ factor are equivalent to a decrease by a factor of $4/9 < 1/2$, which cancels the effect of one 2 factor increase. Hence, at most $0.03T$ iterations can be iterations where $d_t(v) \geq 10$. Thus, out of T iterations, in at least $0.95T$ iterations, we have $d_t(v) < 10$.

Now, in any iteration which $d_t(v) < 10$, node v sets $p_{t+1} = \min\{2p_t(v), 1/2\}$. In all other iterations, p_t is decreased by a 2 factor. Hence, we can conclude that there are at least $0.95T - 0.05T = 0.9T$ in which $p_t(v) = 1/2$. Moreover, out of these at most $0.05T$ are iterations in which $d_t(v) \geq 10$. Hence, there are greater than $0.8T \gg 0.01T$ iterations in which $p_t(v) = 1/2$ and $d_t(v) < 10$. Any such iteration is good for v as v has a constant probability of joining set S in each such iteration. We conclude that v has at least $0.01T$ good iterations. ∎

2.2 An Intuitive Discussion of the Parallel Algorithm

Naively implementing the vanilla version of

the algorithm would require $O(\log^2 n)$ time in EREW PRAM. This is because, we have $O(\log n)$ iterations and implementing each iteration may require $O(\log n)$ time, e.g., even computing $d_t(v) = \sum_{u \in N(v)} p_t(u)$ might require summing up to $\Theta(n)$ numbers, which needs $\Theta(\log n)$ time in EREW PRAM. To reach our desired $O(\log n)$ time complexity, we will "simulate" a version of this algorithm, where we will leverage some smoothness properties of it to ensure that, implementing each iteration requires only $O(1)$ time in EREW PRAM. Let us intuitively discuss a few key ideas.

Predictions, using Smoothness of Probabilities. For any node u, so long as u is not in the set S, the value $p_t(v)$ changes in a smooth manner throughout the iterations: per iteration, it either increases or decreases by a 2 factor, or stays the same at $p_t = 1/2$. Because of this, we can have some predictions into the future, in the following format: if in iteration t node v has $d_t(v) \geq 10 \cdot 2^{|t'-t|}$, we can conclude that in future iteration $t' \geq t$, we will have $d_{t'}(v) \geq 10$. Thus, we know that the update in iteration t' will be $p_{t'+1}(v) = p_{t'}(v)/2$ and that node v will not join set S in iteration t'.

Sparse Estimates, via Sampling. The above smoothness indicates that when $d_t(v)$ is fairly large, the update of $p_{t'}(v)$ is clear for quite some time. We will discuss later how we make use of this property. On the flip side, what if $d_t(v) = \sum_{u \in N(v)} p_t(u)$ is small? Computing the actual value $d_t(v)$ still requires computing a summation over all neighbors $u \in N(v)$, which might be up to $\Theta(n)$ many neighbors and would necessitate $\Theta(\log n)$ time in EREW PRAM, per iteration. Fortunately, we do not need the precise value and having a "good estimation" of $d_t(v)$ suffices. If we sample each node u with probability $p_t(u)$, the number of sampled neighbors is an unbiased estimator of $d_t(v)$ and has the additional nice property that when $d_t(v)$ is small, the size of this sampled set is also likely to be small: in particular, the expected size of the sampled set is $d_t(v)$ and since the samplings are independent, informally, the probability of the set size being much higher than this expectation decays exponentially as a function of the expectation.

Fixing the Randomness. To leverage the sparse sampling-based estimations mentioned above, we fix the randomness of sampling and marking, as follows. At the beginning of the whole execution, each node $v \in V$ samples for each iteration $t \in [T]$, where $T = C \log n$, a random number $r_t(v) \in [0, 1]$ uniformly at random[3], and all independent of each other. We will interpret these by comparing with p-values, e.g., if $r_t(v) \leq p_t(v)$, then node v is marked in iteration t. We fix all these random values $r_t(v)$, for all nodes and all iterations, at the beginning of the execution, and from there on we deal with a deterministic procedure.

Predicting and Refining the Set of Relevant Neighbors. In order to have iterations that can be implemented in $O(1)$ time in EREW PRAM, we would like to have the property that when we arrive at some iteration t', a node needs to check only constant many (relevant) neighbors for whether they are marked or not. For this, we will build a set that contains all potentially relevant neighbors, and as we get closer and closer in time to iteration t', we gradually refine this set by removing some of its vertices. The ultimate guarantee of the refinement is that in iteration t', the set of relevant neighbors for that iteration will have size $O(1)$. But we achieve that by maintaining a more general guarantee: in each earlier iteration $t \leq t'$, the set of neighbors relevant for iteration t' will have size $2^{O(t'-t)}$.

We build these sets of relevant neighbors as follows: For each iteration $t' \in [T]$, each node v creates a set $N_{t'}(v)$, which is a *set of potentially relevant neighbors*. At the very beginning of the algorithm, we include in $N_{t'}(v)$ any neighbor u of v such that $r_{t'}(u) \leq 2^{t'} p_0(u)$. Notice that this definitely includes any node u that would be marked in iteration t' according to the rule $r_{t'}(u) \leq p_{t'}(u)$, because we have $p_{t'}(u) \leq 2^{t'} p_0(u)$–unless u is already in the set S. Moreover, the probability of a neighbor u being included in $N_{t'}(v)$ is $\min\{1, 2^{t'} p_0(u)\}$. When we build $N'_t(v)$ according to this rule, we have two possibilities:

(I) We have $\mathbb{E}[|N_{t'}(v)|] \leq 10K \cdot 2^{3t'}$, in which case we know that $Pr[|N_{t'}(v)| \geq 10K \cdot 2^{3t'}] \leq exp(-K\Theta(2^{3t'}))$. Here, K is a sufficiently large constant.

(II) Otherwise, we have $\mathbb{E}[|N_{t'}(v)|] \geq 10K \cdot 2^{3t'}$, which implies that

$$\begin{aligned} d_{t'}(v) &= \sum_{u \in N(v)} p_{t'}(u) \geq \sum_{u \in N(v)} p_0(u) 2^{-t'} \\ &\geq \sum_{u \in N(v)} 2^{-2t'} \min\{1, p_0(u) 2^{t'}\} \\ &= 2^{-2t'} \mathbb{E}[|N_{t'}(v)|] \\ &\geq 10K \cdot 2^{t'}. \end{aligned}$$

Hence, for all iteration $t'' \in [0, 2t']$, we have $d_t(v) \geq 10$. In this case, it is clear that in any such t'', node v will not join the set S and moreover it will set $p_{t''}(v) = p_{t''-1}(v)/2$. To handle this case (II), we do not need to know the exact set $N_{t'}(v)$; it suffices

[3] A number with $O(\log n)$ bits of precision suffices, as the value $p_t(v)$ remains polynomially bounded.

to detect that $|N_{t'}(v)| \geq 10K \cdot 2^{3t'}$ which indicates that, with probability $1-exp(-K\Theta(2^{3t'}))$, we have $\mathbb{E}[|N_{t'}(v)|] \geq 10K \cdot 2^{3t'}$. We will later see how the analysis deals with the small $exp(-K\Theta(2^{3t'}))$ probably that this indication was wrong and we actually had $\mathbb{E}[|N_{t'}(v)|] < 10K \cdot 2^{3t'}$.

As time proceeds, we refine the sets $N_{t'}(v)$ of potentially relevant neighbors. Generally, in any iteration t, we can look to any future iteration $t' \geq t+1$ and refine $N_{t'}(v)$ by including in it only neighbors u for which $r_{t'}(u) \leq 2^{(t'-t)}p_t(u)$. Again, with a similar argument as above, we see that: either (I) $|N_{t'}(v)| \geq 10K \cdot 2^{3(t'-t)}$, or (II) we detect that $|N_{t'}(v)| \geq 10K \cdot 2^{3(t'-t)}$ which indicates that, with probability $1 - exp(-K\Theta(2^{3(t'-t)}))$, we have $\mathbb{E}[|N_{t'}(v)|] \geq 10 \cdot 2^{3(t'-t)}$. The former would indicate that the desired invariant about the size of $N_{t'}(v)$ is preserved. On the other hand, if the latter happens, it would imply that we can already predict the behavior of v for all iterations $t'' \in [t, t' + (t'-t)]$: in any such iteration t'', node v will not join the set S and moreover it will set $p_{t''+1}(v) = p_{t''}(v)/2$. We later see in the analysis how to take care of the small failure probability, in this inference.

Informing Neighbors in Time. When a node v joins the independent set S in some iteration t, it should inform its neighbors so that they do not join S in the future iterations. Doing this immediately can require $O(\log n)$ time in EREW PRAM, as v might have up to $\Theta(n)$ neighbors. But we can pace down this process. For any future iteration $t' \geq t+1$, all the neighbors u that might attempt to join the independent set S are in the relevant neighbors set $N_{t'}(v)$. Moreover, because of the item discussed above, we know that $|N_{t'}(v)| \leq 2^{O(t'-t)}$, as otherwise we would have inferred that $d_t(v) \geq 10$ and v would not have joined S in iteration t. We have time proportional to $t'-t$ iterations to inform those neighbors in $N_{t'}(v)$. This is sufficient to inform the at most $2^{O(t'-t)}$ neighbors of v in set $N_{t'}(v)$.

One subtlety is that we need to perform this procedure for all iterations $t' \geq t+1$, which might be up to $\Theta(\log n)$ values of t'. Doing this simultaneously for all these iterations would require the information to spread and be written in $\Theta(\log n)$ registers, and that would need $\Theta(\log \log n)$ time in EREW PRAM, for one iteration t. However, fortunately, we do not need to inform all future iterations at the same time. We can slightly delay informing the iterations that are further into the future. We will use a simple structure, which we call *binary push tree*, for spreading this information in a manner that guarantees to start informing neighbors of each iteration $t' \geq t+1$ by iteration $t + \log(t'-t)$. After that, there would still remain time equal to $(t'-t) - \log(t'-t) \geq (t'-t)/2$ iterations, which is sufficient to spread this information to all the $2^{O(t'-t)}$ neighbors in $N_{t'}(v)$.

2.3 Our Parallel Algorithm Here, we present the higher-level aspects of our parallel algorithm. The low-level implementation details for EREW PRAM are discussed in the next subsection.

At the beginning, for each v and each iteration $t' \in [C \log n]$, we draw a uniform random value $r_{t'}(v) \in [0,1]$, independent of all the others[4]. Throughout the iterations, for any future iteration t', we maintain a set $N_{t'}(v)$ of relevant neighbors of node v who might attempt to join the independent set S in iteration t'. At the beginning, we include in $N_{t'}(v)$ any neighbor u of v such that $r_{t'}(u) \leq 2^{t'}p_0(u)$. If $|N_{t'}(v)| \geq 10K2^{3t'}$, we then declare that *node v is overloaded in all iterations $t'' \in [2t']$*. In this case, we do not maintain $N_{t''}(v)$ anymore. Here, K is a sufficiently large constant, which will only affect the constant in the length of an iteration.

The algorithm performs two types of tasks, in each iteration: (I) marking/sampling nodes and updating their probabilities, (II) refining the neighborhood sets for the future iterations. We next discuss these two procedures:

Algorithm Part I, Marking and Probability Updates. Iteration t performs the marking and probability updates as follows:

- If $r_t(v) \leq p_t(v)$, then v is marked, and otherwise it is not marked. In particular, any node $v \in S$ has $p_t(v) = 1$ and is thus marked.
- If $v \in S$, we keep $p_{t+1}(v) = 1$. If v has a neighbor in S, it was declared overloaded for iteration t, or it has a marked neighbor in $N_t(v)$, then regardless of whether v was marked or not, we set $p_{t+1}(v) = p_t(v)/2$. Otherwise, we set $p_{t+1}(v) = \min\{2p_t(v), 1/2\}$.
- If we have all the following conditions satisfied, then v joins the independent set S: (A) v is marked, (B) it is not already in S and (C) it does not have a neighbor in S, (D) it was not previously declared overloaded for iteration t, and most crucially (E) it does not have a marked neighbor in $N_t(v)$. If all of these hold, then v joins S, and it it will inform any neighbor $u' \in N_{t'}(v)$ for any future iteration $t' \geq t+1$. Moreover, it will set $p_{t'}(v) = 1$ for all future iterations $t' \geq t+1$.

[4] As mentioned before, $O(\log n)$ bits of precision suffice.

Algorithm Part II, Neighborhood Set Refinement. Once we have finishing performing these updates for iteration t, we also refine the relevant neighborhood sets $N_t(v)$ of the future iterations t', as follows: At the time of iteration t, we include neighbor u in $N_{t'}(v)$ only if $r_{t'}(u) \leq 2^{(t'-t)}p_t(u)$. Notice that, if $u \notin N_{t'}(v)$, we know that u would not be marked in iteration t', because then $r_{t'}(u) > 2^{(t'-t)}p_t(u) \geq p_{t'}(u)$. The algorithm works to maintain the invariant that, in each iteration t, for each future iteration $t' \geq t+1$, we have $|N_{t'}(v)| \leq 10K2^{3(t'-t)}$. This will not be performed immediately in the time of one iteration and relies on lower level implementation in EREW PRAM, as will be discussed in the next subsection. If after finishing the updates of some iteration t, we realize that this invariant is broken for $N_{t'}(v)$ of some future iteration t', we then declare that *node v is overloaded in all iterations $t'' \in [t, t' + (t'-t)]$*. In this case, we do not maintain $N_{t''}(v)$ anymore. Recall from the description in Part I that in this case, in iteration t'', node v does not join set S and it will set $p_{t''+1}(v) = p_{t''}(v)/2$.

2.4 Implementation Details Here, we present the implementation details of the above algorithm, and in particular we show that we can implement each iteration in $O(1)$ time in the EREW PRAM model, and we perform at most $O(m \log n)$ work overall.

2.4.1 Pushing Information to the Relevant Neighbors of Future Iterations Creating Multiple Copies for Different Future Iterations. Consider an iteration t and a node v. After performing this iteration, we may learn that its probability $p_t(v)$ should increase by a 2 factor (or remain at $1/2$) or decrease by a 2 factor. Moreover, node v might have joined the independent set S in this iteration. These information about v, which can be described in $O(1)$ bits, should be sent to all neighbors u that might attempt to enter the independent set S in the future. Concretely, for any future iteration $t' \geq t+1$, any node $u \in N_{t'}(v)$ should be informed. For this, we need to create many copies of the information, one for each iteration $t' \geq t+1$. As discussed above, a flexibility in the problem is that we have more time to inform later iterations.

We create a *binary push tree*: in iteration t, one processor creates the first copy of the information, then in iteration $t+1$, we have two processors each of which creates a new copy (one of them by reading the first copy, generated in the previous iteration), and in general, in iteration $t+i$, we have i processors that create 2^i new copies. We use the k^{th} copy used to start informing the neighbors in $N_{t+k}(v)$. Hence, for any future iteration $t' \geq t+1$, the process of informing $N_{t'}(k)$ will start by iteration $t + \log(t'-t)$. Finally, observe that we perform at most $O(\log n)$ work, in this binary push tree, for a fixed iteration t informing the future iterations. Hence, overall all nodes v and iterations t, we perform at most $O(n \log^2 n)$ work.

Pushing the Information to the Neighbors in Future Iterations. Now that we have the copies for different future iterations t' ready, we have to send the information to the relevant neighbors $N_{t'}(v)$. We can now treat each t' separately. For each iteration t', we spread the information to the $2^{O(t'-t)}$ neighbors in $N_{t'}(v)$, by iteration $t+\log(t'-t)+(t'-t)/10$. Notice that here we use the invariant that $|N_{t'}(v)| \leq 10K2^{3(t'-t)}$, which allows us to spread the information in time $\log(|N_{t'}(v)|) \leq (\log K) + 3(t'-t) + 4$ in EREW PRAM, which is less than $(t'-t)/10$ iterations. Here, we assume that we have fixed the length of each iteration to be a large enough constant time, e.g., greater than $\log K + 40$. We have $|N_{t'}(v)| = 10K2^{3(t'-t)}$ because otherwise, node v would have been declared overloaded for iteration t and we would have already known to set $p_{t+1}(v) = p_t(v)/2$, without v attempting to join the independent set S in iteration t. As such, if $|N_{t'}(v)| > 10K2^{3(t'-t)}$, we do not need to deliver the update to the neighbors in $N_{t'}(v)$ and any such neighbor (that does not hear about an increase of $p_t(v)$ in iteration t) can safely assume that (I) node v did not join S in iteration t and (II) the probability update of v in iteration t was a decrease as $p_{t+1}(v) = p_t(v)/2$.

Finally, observe that in the above push process, for each node v, each current iteration t, and each future iteration t', we perform work at most proportional to the degree of v in the base graph, i.e., we inform at most all the neighbors of v. Hence, overall, we perform at most $O(m \log^2 n)$ work.

2.4.2 Pulling Information from the Relevant Neighbors of the Previous Iterations We have to regularly update the sets $N_{t'}(v)$ of relevant neighbors in the future iterations t'. For each fixed t', we perform this update in exponentially spaced apart times, that is, a new update starts when we reach iteration $t' - 2^i$ for some $i \geq 0$. We call this level i of the update for iteration t'. Notice that we go from level $O(\log \log n)$ to level 0. Of course, pushing and pulling the updates takes time, and the result does not arrive immediately. For our algorithm, it suffices to ensure that, by each previous iteration $t' - 2^i$, we have updated $N_{t'}(v)$ in a way that it incorporates all the probability updates of neighbors up to iteration $s = t' - 2 \cdot 2^i$. This way, if after having incorporated these updates, we

have $|N_{t'}(v)| > 10K2^{3(s-t')} = 10K2^{6\cdot 2^i}$, then we can declare iteration t' overloaded, as well as all iterations in $[t'-2^i, t'+(s-t')]$.

Updates of Level i. Inductively, we can assume that before iteration $t = t' - 2\cdot 2^i$, we have finished the level $i+1$ update for $N_{t'}(v)$ which means we have incorporated all the updates up to iteration $t'-4\cdot 2^i$. At this point, either the set $N_{t'}(v)$ has size at most $10K2^{12\cdot 2^i}$ or we have declared v overloaded for iteration t' and thus do not need to maintain $N_{t'}(v)$ anymore. In the former case, we now should perform the level i update and incorporate all the updates of iterations $[t'-4\cdot 2^i, t'-2\cdot 2^i]$.

First, we wait for iteration $t'-2\cdot 2^i+i$ so that all the updates of iterations $[t'-4\cdot 2^i, t'-2\cdot 2^i]$ by any of the neighbors have been pushed forward and prepared as a separate copy for the update of $N_{t'}(v)$, as discussed in the previous subsubsection. Then, for each node $u \in N_{t'}(v)$, starting in iteration $t'-2\cdot 2^i+i$, we use $O(2^i)$ processors who aggregate the number of probability increase or decreases in the probability of node u, during the iterations $[t'-4\cdot 2^i, t'-2\cdot 2^i]$. Since these are $O(2^i)$ numbers and already have been prepared for us in allocated registers, thanks to the push operation discussed above, this is possible in i iterations.

Second, we have to examine each node $u \in N_{t'}(v)$ to see whether it should be included in $N_{t'}(v)$, according to the refinement rule for $t = t'-2\cdot 2^i$, which means we include only those neighbors for which $r_t(u) \le 2^{t'-t}p_{t(u)}$. This can be performed separately for each of the neighbors. This way, we can determine which nodes should be removed from $N_{t'}(v)$, in $O(1)$ time.

Then, we use $2^i/3$ iterations to filter the neighbors that should be removed and reduce the size of $N_{t'}(v)$. Since the set $N_{t'}(v)$ that we work with has size only $10K2^{12\cdot 2^i}$, we can perform this operation of removing the vertices in $2^i/3$ iterations (each having $O(1)$ time in EREW PRAM). At the end, by iteration $t'-2\cdot 2^i + i + 2^i/3$, we have kept only relevant neighbors, with the guarantee that (I) either $N_{t'}(v) \le 10K2^{6\cdot 2^i}$ or (II) node v is declared as overloaded for iteration t'. In the latter case, we also declare v overloaded for all iterations in $[t'-2^i, t'+2\cdot 2^i]$, and this can be done using i extra iterations. Hence, by $t'-2\cdot 2^i+i+2^i/3+i \le t'-2^i$, we are done with processing the refinement of the set $N_{t'}(v)$ in a way that incorporates all the updates of the neighbors in iterations $[t'-4\cdot 2^i, t'-2\cdot 2^i]$.

Finally, observe that in the above pull process, for each node v and each future iteration t', we perform work at most proportional to the degree of v in the base graph for each iteration in the level. Hence, over all nodes, all future iterations and all current iterations, this is at most $O(m\log^2 n)$ work.

2.5 Analysis Proof Outline. For the analysis, we focus on one node v and trace the changes in $p_t(v)$ as well as $d_t(v) = \sum_{u\in N(v)} d_t(u)$. We argue that, except for a very small fraction of the iterations, all of the changes should be as we expect from the vanilla version of the algorithm. Thus, we can argue, similar to how we did for the vanilla version of the algorithm, that each node v will have a constant fraction of iterations that are good for it. Again, we call an iteration good for a node if there is a constant probability[5] that this nodes joins or has a neighbor join the independent set S (unless already that has happened in the previous iterations). Since we have $T = C\log n$ iterations, with a desirably large constant C, having a constant fraction that are good implies that v is in S or has a neighbor in S, with high probability, and a union bound over all nodes v completes the proof.

To formalize this outline, we next define some types of iterations in which the changes are not according to what we expect, and we bound the number of such iterations.

Definition 1 — Iterations Wrongly Declared as Overloaded. Consider an iteration t and a future iteration $t' \ge t+1$. Recall that if $|N_{t'}(v)| \ge 10K2^{3(t'-t)}$, we declare node v overloaded in iteration t', and in fact all iterations $t'' \in [t, t'+(t'-t)]$. This is because, we estimate that we must have $d_t(v) \ge 102^{3(t'-t)}$ which would indicate that $d_{t''}(v) \ge 10$. We emphasize that here $d_t(v) = \sum_{u\in N(v)} p_t(u)$, where the summation is over all the neighbors. If in fact $d_t(v) \le 102^{3(t'-t)}$ and still we got $|N_{t'}(v)| \ge 10K2^{3(t'-t)}$, we say that we *wrongly declared each iterations* $t'' \in [t, t'+(t'-t)]$ *overloaded* for node v.

Next, we show that the probability of each iteration being wrongly declared overloaded for node v, is at most some constant ε, which can be made desirably small by making the constant K large enough, and that overall, with high probability, there are at most εT iterations that are wrongly declared overloaded. Much of our analysis will rely on this second statement, which we prove first:

LEMMA 2.1. *The number of iterations that are wrongly declared overloaded for a node v is at most εT, w.h.p. Here, $\varepsilon > 0$ is a constant that can be made desirably small by making the constant K large enough*[6].

[5] Throughout the analysis, we prioritize simplicity and readability, and thus we make no attempt at optimizing the constants. We think that a more careful (but also more tedious) analysis should be able to provide much smaller constants.

[6] We note that it suffices if K grows linearly in $\log 1/\varepsilon$, and

Proof. Consider a future iteration t' and the related set $N_{t'}(v)$ of relevant neighbors. Recall that we process $N_{t'}(v)$ in levels, starting from level $O(\log\log n)$ and ending in level 0, where level i considers the size of the set according to probabilities $p_t(u)$ of neighbors $u \in N(v)$ at time $t = t' - 2 \cdot 2^i$. Then, we declare v overloaded if $|N_{t'}(v)| \geq 10K2^{6\cdot 2^i}$. If that happens, all iterations $t'' \in [t' - 2^i, t' + 2 \cdot 2^i]$ are declared overloaded. These iterations were *wrongly* declared overloaded if in iteration t, we had $d_t(v) \leq 102^{6\cdot 2^i}$ and still the sampled set $N_{t'}(v)$ had size greater than $10K2^{6\cdot 2^i}$. If that happened, we wrongly declared some $y = 3 \cdot 2^i$ iterations overloaded. Since different nodes are sampled independently for inclusion in $N_{t'}(v)$, by Chernoff bound, the probability of that event is at most $exp(-\Theta(K2^{6(2^i)})) \ll \frac{\varepsilon}{10}2^{-y}$.

For each $t' \in [0, T]$, let $X_{t'}$ be the total number of iterations that are wrongly declared overloaded because of examining $|N_{t'}(v)|$ in different levels. Define $X = \sum_{t'\in[T]} X_{t'}$. Notice that $\mathbb{E}[X'_t] \leq \sum_{y\geq 1} \frac{\varepsilon}{10} y 2^{-y} \leq \varepsilon/5$. Hence, $\mathbb{E}[X] \leq \frac{\varepsilon}{5}T$. We next argue that X is also well-concentrated around this mean, and with high probability, we have $X \leq \varepsilon T$.

Notice that for each $t' \in [0, T]$, we have $\Pr[X'_t \geq z] \leq \sum_{y\geq z} \frac{\varepsilon}{10}2^{-y} \leq \frac{2\varepsilon}{10}2^{-z}$. Moreover, $X = \sum_{t'\in[T]} X_{t'}$ is a summation of independent random variables $X_{t'}$, since the randomness used for different sets $N_{t'}$ is independent. Furthermore, these random variables have an exponentially decaying tail as described by $\Pr[X'_t \geq z] \leq \frac{\varepsilon}{5}2^{-z}$. To formalize the above concentration claim, we use a basic stochastic domination argument, see e.g., [Doe18, Section 1.8]. For each $t' \in [0, T]$, define a new random variable $Y_{t'}$ as follows. First, toss a coin with tail probability $\frac{4\varepsilon}{10}$ and if it comes out head, set $Y_{t'} = 0$. If the coin is head, then set $Y_{t'}$ equal to a geometric random variable with parameter 1/2. The coins of different variables $Y_{t'}$ are independent. For any $z \geq 1$, we have $Pr[Y_{t'} = z] = \frac{4\varepsilon}{10}2^{-z}$. This also implies that for any $z \geq 1$, we have $Pr[Y_{t'} \geq z] \geq \frac{4\varepsilon}{10}2^{-z} > \Pr[X_{t'} \geq z]$. Hence, the random variable $Y_{t'}$ stochastically dominates the random variable $X_{t'}$, and the random variable $Y = \sum_{t'} Y_{t'}$ stochastically dominates the random variable $X = \sum_{t'} X_{t'}$.

Now, let us examine $Y = \sum_{t'\in T} Y_t$. Among these T random variables, each had a tail coin (i.e., thus allowing the variable to assume non-zero values) independently with probability $\frac{4\varepsilon}{10}$. Hence, by Chernoff bound, the number of the tails is at most $5\varepsilon T/10$, with probability at least $1 - exp(-\varepsilon T/10) \leq 1 - 1/^{10}$. The inequality holds as we set the constant C in the definition $T = C \log n$ large enough, e.g., $C \geq 200/\varepsilon$. Thus, with high probability, Y is a summation of at most $\frac{5\varepsilon}{10}T$ geometric random variables, each with parameter 1/2. Hence, by applying the variant of Chernoff bound for summation of geometric random variables [Doe18, Theorem 1.10.32], we have $Pr[Y \geq \varepsilon T] \leq exp(-\varepsilon T/10) \leq 1/n^{10}$. That is, with high probability, we have $Y \leq \varepsilon T$. Since Y stochastically dominates X, we conclude that the number X of wrongly declared overloaded iterations is at most εT, with high probability. ∎

that making K larger increases only the constant in the time-length of an iteration (which should grow proportional to $\log K$), in terms of EREW model time.

LEMMA 2.2. *The probability of each given iteration t' being wrongly declared overloaded for v is at most ε.*

Proof. Iteration t' might be wrongly declared overloaded when considering the size of $N_{t'}(v)$ in each previous iteration $t \leq t'$. But the probability of this wrong declaration in iteration t is at most $exp(-\Theta(K)2^{3(t'-t)}) \ll \frac{\varepsilon}{10}2^{-(t'-t)}$. Even a union bound over these upper bounds for iterations $t \leq t'$, which are decaying as geometric series as $t' - t$ increases, is sufficient to conclude that the overall probability of wrongly declaring t' overloaded is at most ε. ∎

Definition 2 — Heavy, Light, and Good Iterations. We say iteration t is *heavy* for node v if $d_t(v) \geq 10$. If $d_t(v) \leq 0.01$, we call this a *light* iteration for v. Notice that when the summation is in $(0.1, 10)$, the iteration is neither light nor heavy.

We call iteration t *good for node v* if (I) this is not a heavy iteration for v, it has not been wrongly declared overloaded, and we have $p_t(v) > 0.01$, or (II) if this is not a light iteration for v, it has not been wrongly declared overloaded, and in the summation $d_t(v) = \sum_{u\in N(v)} p_t(u)$, at least $0.01d_t(v)$ is contributed by neighbors u for which this is not a heavy iteration, do not have a neighbor in S, and who were not wrongly declared overloaded.

We next see that in each of these the two kinds of good iterations, there is a constant probability of v joining or having a neighbor join S.

LEMMA 2.3. *In each type (I) good iteration, node v has a constant probability of joining the independent set S.*

Proof. The probability that v is marked in this iteration is at least 0.01. Independent of that, the probability that no neighbor v is marked is at least $\prod_{u\in N(v)}(1 - p_t(u)) \geq 4^{-\sum_{u\in N(v)}(1-p_t(u)} = 4^{-10}$. Hence, v joins the independent set S with a constant probability. ∎

LEMMA 2.4. *In each type (II) good iteration, there is a constant probability that a neighbor of node v joins the independent set S.*

Proof. First, let us ignore that we are in an iteration that is not wrongly declared overloaded, as being wrongly declared overloaded has probability at most ε, by Lemma 2.2, which can be made arbitrarily small. Scan the set U of neighbors u that have $d_{t-1}(u) \leq 10$ and are not adjacent to S one by one, until we find one that is marked. We succeed to find one marked neighbor with probability at least $1 - \prod_{u \in U}(1 - p_t(u)) \geq 1 - e^{-\sum_{u \in U} p_t(u)} \geq 1 - e^{-0.5} > 0.4$ Now, since the event that v is wrongly declared overloaded has probability at most ε, the probability that we find a marked neighbor even conditioned on that the iteration is not wrongly declared overloaded is at least $0.4 - \varepsilon \geq 0.3$. Once we have found the first such marked neighbor $u \in U$, the probability that no neighbor of u is marked is at least $\prod_{w \in N(u)}(1 - p_t(w)) \geq 4^{-\sum_{w \in N(u) p_t(w)}} \geq 4^{-10} = 2^{-20}$. Hence, the probability that node v has one of its neighbors join S in this iteration is at least $0.3 \cdot 2^{-20}$, which is a constant[7]. ∎

Definition 3 — Wrong Moves. Consider an iteration t that is not good for v. First, suppose that this is not a light iteration for v. We call this iteration a *wrong up-move* for v if $d_{t+1}(v) > 0.6 d_t(v)$. On the other hand, suppose that this is a light iteration for v. We call this iteration a *wrong down-move* for v if node v has at least one marked neighbor in $N_t(v)$ and thus sets $p_{t+1}(v) = p_t(v)/2$. We next see that the probability of each wrong move is also fairly small.

LEMMA 2.5. *For any iteration that is not good for v and which is not light for v, the probability of a wrong up-move for v is at most* 0.01.

Proof. Consider v and the set U of its neighbors for which this iteration is heavy, they have a neighbor in S, or they were wrongly declared overloaded. Notice that since iteration t is not good for v, in the summation $d_t(v) = \sum_{u \in N(v)} p_t(u)$, at least $0.99 d_t(v)$ is contributed by neighbors $u \in U$. Consider a neighbor $w \in U$. Node w increases its probability if it was not declared overloaded in iteration t, it does not have a neighbor in S, and no neighbor of w is marked. Since this iteration is heavy for w, in the latter case, we have $d_t(w) \geq 10$. In that case, the probability that no neighbor of w is marked is at most $\prod_{w' \in N(w)}(1 - p_t(w')) \leq e^{-\sum_{w' \in N(w)} p_w} \leq e^{-10}$. Thus, over all neighbors in U, we expect at most a e^{-10} fraction of the summation $\sum_{w \in U} p_t(w)$ to increase (by a 2 factor at most). By Markov's inequality, the probability that more than 0.01 of $\sum_{w \in U} p_t(w)$ increases is less than 0.01. The contribution to the summation that comes from neighbors in $N(v) \setminus U$ is at most $0.01 d_t(v)$ and that can grow by at most a 2 factor. All the rest of the contribution in U decrease by a 2 factor. Hence, we can conclude that with probability at least 0.99, we can bound $d_{t+1}(v) \leq 2 \times 0.02 d_t(v) + d_t(v)/2 < 0.6 d_t(v)$. ∎

LEMMA 2.6. *For any iteration that is not good for v and is light for v, the probability of a wrong down-move for v is at most* 0.02.

Proof. Recall that this is a wrong down-move for v if, (I) although this is a light iteration for v and we have $d_t(v) \leq 0.01$, at least one neighbor of v is marked and thus we set $p_{t+1}(v) = p_t(v)/2$, or (II) this iteration is declared overloaded for v. The probability of the former is at most $\sum_{u \in N(v)} p_t(u) \leq 0.01$, by a simple union bound. The probability of the latter is at most $\varepsilon \ll 0.01$. Hence, the overall probability of a wrong down-move is at most 0.02. ∎

We use these two lemmas to conclude that, overall, the number of wrong up or down moves is at most $0.03T$, with high probability. Notice that wrong moves deal only with iterations that are not good. In particular, when bounding the number of wrong moves, we have not revealed the randomness used in good iterations.

LEMMA 2.7. *In the course of $T = C \log n$ iterations, we have at most $0.03T$ wrong moves for node v, with high probability. Moreover, this guarantee holds independent of the randomness used by v and its neighbors in the good iterations.*

Proof. Follows from the Chernoff bound since the randomness of iterations that are not declared overloaded are independent, and at most εT iterations are wrongly declared overloaded. ∎

Wrap Up of the Analysis. Having all these lemmas, we are now ready to wrap up the proof and show that any node v, unless it has already joined the independent set S or has a neighbor in S, with have at least $0.05T$ good iterations rounds, with high probability.

LEMMA 2.8. *After T iterations, the computed set S is a maximal independent set, with high probability.*

[7] As noted before, this analysis prioritizes simplicity and does not attempt to optimize the constants. We think that a more careful (but also more tedious) analysis should be able to provide significantly better constants.

Proof. Fix a node v. We show that within T iterations, with probability at least $1 - 1/n^2$, node v is either in S or has a neighbor in S. The lemma then follows from a union bound over all vertices.

We know that, with high probability, we have at most $0.03T$ wrong moves for v, and at most εT iterations that are wrongly declared overloaded for v. Having this, we use an argument similar to the vanilla version and conclude that we have at least $0.03T$ good iterations.

Suppose not. First, consider all iterations that are not light for v. By definition of a wrong move-up, each such iteration is either good, a wrong up-move or, it implies that $d_{t+1}(v) \leq 0.6 d_t(v)$. Among these not-light iterations, at most $0.03T$ are (type II) good iterations, at most $0.03T$ are wrong up-moves, and at most $\varepsilon T < 0.01T$ are wrongly declared overloaded. So, among the not-light iterations, with the exception of less than $0.07T$ iterations, the rest lead to a 0.6 factor decrease in $d_t(v)$. We start with $d_0(v) = 1$, and every two iterations of decrease by 0.6 factor cancels the effect of one increase iteration (at most a 2 factor increase). Hence, we cannot have more than $0.24T$ iterations that are not light for v.

Now consider the light iterations for v. With the exception of at most $0.03T$ wrong down-moves, at most $\varepsilon T < 0.01T$ iterations that are wrongly declared overloaded for v, and at most $0.03T$ good iterations, all other light iterations make v set $p_{t+1}(v) = \min\{2p_t(v), 1/2\}$. Hence, we have less than $0.07T$ light iterations during which $p_t(v)$ decreases. We start with $p_t(v) = 2^{-\lceil \log \Delta \rceil}$, and every 2-factor increase cancels a 2-factor decrease. In not-light iterations, $p_t(v)$ may decrease, but each time by a 2-factor, and we only have at most $0.24T$ not-light iterations. Thus, there are at most $2(0.24 + 0.07)T + \log \Delta \leq 0.63T$ light iterations where $p_t < 1/2$ or v is wrongly declared overloaded.

Since the number of not-light iterations is at most $0.24T$ and the number of light iterations where $p_t(v) < 1/2$ or v is declared overloaded is at most $0.63T$, at least $0.03T$ iterations remain, which are not heavy and not wrongly declared overloaded, and where $p_t(v) = 1/2$. These are good iterations for v, of type (I). Hence, we have at least $0.03T$ good iterations for v.

The above argument for the number of good iterations does not reveal anything about the randomness of good iterations (in our accounting, we always admitted good iterations to behave opposite to our argument, and thus the argument holds even if the randomness for those iterations is fixed in an adversarial manner). In each good iteration, there is a constant probability of having v join S or have a neighbor join S. Therefore, after $0.03T$ good iterations, with probability $1 - 1/n^2$, node v is either in S or has a neighbor in S. ∎

References

[ABI86] Noga Alon, László 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.

[AS87] Baruch Awerbuch and Yossi Shiloach. New connectivity and msf algorithms for shuffle-exchange network and pram. *IEEE Trans. on Computers*, 36(10):1258–1263, 1987.

[BKK94] Paul Beame, Mirosław Kutyłowwski, and Marcin Kik. Information broadcasting by exclusive-read prams. *Parallel Processing Letters*, 4(01n02):159–169, 1994.

[CDR86] Stephen Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing (SICOMP)*, 15(1):87–97, 1986.

[CHL01] Ka Wong Chong, Yijie Han, and Tak Wah Lam. Concurrent threads and optimal parallel minimum spanning trees algorithm. *Journal of the ACM (JACM)*, 48(2):297–323, 2001.

[Cho96] Ka Wong Chong. Finding minimum spanning trees on the erew pram. In *International Computer Symposium*, pages 7–14, 1996.

[CL95] Ka Wong Chong and Tak Wah Lam. Finding connected components in $O(\log n \log \log n)$ time on the erew pram. *Journal of Algorithms*, 18(3):378–402, 1995.

[CLC82] Francis Y Chin, John Lam, and I-Ngo Chen. Efficient parallel algorithms for some graph problems. *Communications of the ACM*, 25(9):659–665, 1982.

[Coo85] Stephen Cook. A taxonomy of problems with fast parallel algorithms. *Information and control*, 64(1-3):2–22, 1985.

[CV86] Richard Cole and Uzi Vishkin. Approximate and exact parallel scheduling with applications to list, tree and graph problems. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 478–491. IEEE, 1986.

[Doe18] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. *arXiv preprint arXiv:1801.06733*, 2018.

[Fic93] Faith E Fich. The complexity of computation on the parallel random access machine. In John H. Reif, editor, *Synthesis of Parallel Algorithms*, chapter 20, pages 843–899. Morgan Kaufmann, 1993.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *ACM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016.

[GHR+95] Raymond Greenlaw, H James Hoover, Walter L Ruzzo, et al. *Limits to parallel computation: P-completeness theory*. Oxford University Press on Demand, 1995.

[GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively

parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1636–1653. SIAM, 2019.

[HCS79] Daniel S. Hirschberg, Ashok K. Chandra, and Dilip V. Sarwate. Computing connected components on parallel computers. *Communications of the ACM*, 22(8):461–464, 1979.

[JaJ92] Joseph JaJa. *An introduction to parallel algorithms*. Reading, MA: Addison-Wesley, 1992.

[JM91] Donald B Johnson and Panagiotis Takis Metaxas. Connected components in $O(\log^{3/2}|V|)$ parallel time for the crew pram. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 688–697, 1991.

[JM92] Donald B Johnson and Panagiotis Metaxas. A parallel algorithm for computing minimum spanning trees. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 363–372, 1992.

[KNP92] David R Karger, Noam Nisan, and Michal Parnas. Fast connected components algorithms for the erew pram. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 373–381, 1992.

[KW84] Richard M Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing (STOC)*, pages 266–272, 1984.

[Lub85] M Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC)*, pages 1–10, 1985.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing (SICOMP)*, 15(4):1036–1053, 1986.

[NSW92] N Nisan, E Szemeredi, and A Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 24–29, 1992.

[Rei05] Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 376–385, 2005.

[Tri05] Vladimir Trifonov. An $O(\log n \log\log n)$ space algorithm for undirected st-connectivity. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 626–633, 2005.

[Val82] Leslie G. Valiant. Parallel computation. In *the 7th IBM Symposium on Mathematical Foundations of Computer Science*, 1982.