Faster Set Cover in the MPC Model

Hongyan Ji
The University of Iowa
Iowa City, United States
hongyan-ji@uiowa.edu

Shreyas Pai
Indian Institute of Technology Madras
Chennai, India
shreyas@cse.iitm.ac.in

Sriram Pemmaraju
The University of Iowa
Iowa City, United States
sriram-pemmaraju@uiowa.edu

Joshua Sobel
The University of Iowa
Iowa City, United States
joshua-sobel@uiowa.edu

Abstract

The Massively Parallel Computation (MPC) model is a popular abstraction for large-scale distributed computing. The Set Cover problem is a classical combinatorial optimization problem that has a wide variety of applications and generalizes many well-studied problems such as vertex cover, edge cover, and minimum dominating set. For a Set Cover instance with a ground set of n elements and a collection of m sets, we present two $O(\log n)$-approximation algorithms in the MPC model. Our algorithms run in $\tilde{O}(\log N)$-rounds in the linear-memory MPC model and in $\tilde{O}(\log^{1.5} N)$ rounds in the sublinear-memory MPC model, where $N = n + m$. These are the first $O(\log n)$-approximation algorithms for Set Cover in this setting that run in $o(\log^2 N)$ rounds. Our results are obtained by repurposing the sparsified graph exponentiation technique that has been successful for the maximal independent set problem in the MPC model and applying it to a simple, distributed Set Cover algorithm by Grunau, Mitrović, Rubinfeld, and Vakilian (SODA 2020).

CCS Concepts

• **Theory of computation → Massively parallel algorithms**; **Distributed algorithms**.

Keywords

Parallel Set Cover, Massively Parallel Computing Model, Approximation

ACM Reference Format:
Hongyan Ji, Shreyas Pai, Sriram Pemmaraju, and Joshua Sobel. 2025. Faster Set Cover in the MPC Model. In *26th International Conference on Distributed Computing and Networking (ICDCN 2025), January 04–07, 2025, Hyderabad, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3700838.3700861

1 Introduction

We present fast algorithms for the SetCover problem in the Massively Parallel Computation (MPC) model [12, 16, 24, 31]. In the SetCover problem we are given a ground set of n elements and a collection of m sets that jointly cover all elements. The goal is to find the smallest subcollection of sets that still cover all elements. SetCover is a classical combinatorial optimization problem that is a generalization of many well-studied problems like vertex cover, edge cover, and minimum dominating set. The MPC model is defined by a set of machines connected via an all-to-all communication network, with each machine having at most W words of local memory (one word is $O(\log N)$ bits). Communication and computation in this model are synchronous. At the beginning of the algorithm, the input is distributed across the machines. During a round, each machine receives up to W words from all the other machines, performs local computation, and sends up to W words to all the other machines to guide the computation in the next round. Note that in this model the bandwidth constraint is not associated with individual edges, but is associated with each machine.

A natural, greedy sequential algorithm provides a $\ln n + O(1)$-approximation [5, 23, 26]. This algorithm is optimal since there can be no $c \ln n$-approximation to SetCover for any $c < 1$ unless $P = NP$ [11]. The SetCover problem has also been extensively studied in parallel settings and $O(\log n)$-approximation algorithms are known [3, 30], and these algorithms have served as the basis for *distributed* SetCover approximation algorithms [4, 22]. For example, in the Congest model[1] of distributed computation, the work of Censor-Hillel and Dory [4] implies an algorithm for the SetCover problem that guarantees an approximation ratio of $O(\log n)$, and takes $O(\log^2 N)$ rounds, where $N = n+m$. This Congest algorithm succeeds with high probability (whp), i.e., with probability at least $1 - 1/N^c$ for a constant $c > 1$.

There is an active stream of research that aims to design MPC algorithms that are significantly faster than their Congest counterparts for various problems [7, 8, 14, 15, 25]. In particular for the Maximal Independent Set (MIS) problem, the best-known round complexity (in terms of the number of vertices, N) in the Congest model is $O(\log N)$ [1, 27], whereas in low-memory MPC model it is $\tilde{O}(\sqrt{\log N})$[2] [15, 25] and in the linear-memory MPC model it is $O(\log \log N)$ [14]. But, as far as we know, SetCover seems to have resisted such improvements thus far and the fastest known MPC algorithm simply simulates the fastest Congest algorithm [4] one round at a time, taking $O(\log^2 N)$ rounds. In this paper,

[1]The Congest model [29] is a classical distributed model where the input is a graph $G = (V, E)$, which also serves as the communication network. Nodes in the graph are processors with unique IDs from a space whose size is polynomial in $|V|$. In each round, each node can send an $O(\log |V|)$-bit message to each of its neighbors.
[2]The $\tilde{O}(\cdot)$ notation hides poly log log N factors.

ICDCN 2025, January 04–07, 2025, Hyderabad, India
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1062-9/25/01
https://doi.org/10.1145/3700838.3700861

motivated by the speedups obtained for MIS, we show that there are substantially faster MPC algorithms for SetCover. Specifically, we prove the following theorem.

Theorem (Informal Main Result). *We are given a SetCover instance with a ground set of n elements and a collection of m sets. Let $N = m + n$. We can compute whp an $O(\log n)$-approximate solution to SetCover that runs in*

- *$\tilde{O}(\log^{1.5} N)$ rounds in the low-memory MPC model and*
- *$\tilde{O}(\log N)$ rounds in the linear-memory MPC model.*

1.1 SetCover in the MPC Model

The input to the SetCover problem is a ground set X, $|X| = n$ and a collection $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ of subsets $S_i \subseteq X$ satisfying $\cup_i S_i = X$. The output of the problem is required to be the smallest subcollection $C \subseteq \mathcal{S}$ such that $\cup_{S \in C} S = X$. For a SetCover instance, we use $\mathbf{s}$ to denote the size of the largest set and $\mathbf{t}$ to denote the largest number of sets an element appears in. A SetCover instance can be represented as a bipartite graph $G = (X, \mathcal{S}, E)$ with node set $X \cup S$ of size $N = m + n$, and the edge set E consisting of edges $\{S, x\}$, where $x \in X$, $S \in \mathcal{S}$, and $x \in S$.

In the MPC model, the local memory of each machine W and the number of machines $|\mathcal{M}|$ used are assumed to be sublinear to the input size I. Specifically, $W = I^{\alpha}$ and $|\mathcal{M}| = I^{\beta}$, where $0 < \alpha < 1$, and $0 < \beta < 1$ are constants. The number of machines $|\mathcal{M}|$ is assumed to be large enough so that we have enough memory to store the entire input, that is $|\mathcal{M}| \cdot W \geq I$. When the input is an N-node graph, the difficulty of designing MPC algorithms seems to be tied to how the memory per machine W relates to N. For this reason, researchers have separately studied the following three memory-size regimes. Note that in all three regimes, it is still true that the local memory parameter W is sublinear in the total input size I.

Low-Memory MPC: $W = \Theta(N^{\delta})$ local memory per machine for $0 < \delta < 1$.

Linear-Memory MPC: $W = \Theta(N)$ local memory per machine.

Superlinear-Memory MPC: $\Theta(N^{1+\delta})$ local memory per machine for $0 < \delta < 1$.

It is known that SetCover can be solved (approximately) in $O(1)$ rounds in the superlinear-memory MPC model [20] (see details below in Section 3). Our paper focuses on the two more challenging memory regimes, low-memory MPC and linear-memory MPC.

In the MPC model, the standard assumption is that the input graph edges are distributed arbitrarily among the machines. But it is convenient to design algorithms assuming that the input is distributed in a *node-centric* manner[3]. That is for each node v in the input graph, there is a machine M_v that hosts it and M_v knows all the neighbors of v and the machines that host these neighbors. A machine can host multiple vertices as long as the local memory constraint is not violated.

An issue that immediately arises in the low-memory MPC model, is that the degree of a node itself can be too large to fit in a single machine. We deal with this issue in a standard way [2, 15, 25]. We first replace the edges incident on a node v with $\deg(v) > N^{\delta}$ by a virtual, $N^{O(\delta)}$-ary balanced tree of depth $O(1/\delta)$. The root of this tree is v and its leaves are the neighbors of v. Each intermediate node is a virtual node with degree $N^{O(\delta)}$ and thus the neighborhood of every virtual node can fit in a machine. Communication between v and its neighbors occurs via this tree and takes $O(1/\delta)$ rounds, instead of just 1 round.

2 Technical Contributions

Our starting point is a simple SetCover algorithm by Grunau, Mitrović, Rubinfeld, and Vakilian [19]. For our purpose, it is best to view this as a distributed Congest model algorithm. In the pseudocode (Algorithm 1) below, $\mathbf{s}$ is the size of the largest set in $\mathcal{S}$ and $\mathbf{t}$ is the maximum number of sets an element appears in. We use $\deg_{i,k}(S)$ to refer to the number of uncovered elements in S just before iteration k in stage i. This algorithm produces an $O(\log \mathbf{s})$-approximate solution in expectation to SetCover [19]. This algorithm can be implemented "as is" in the linear-memory MPC model in $O(\log \mathbf{s} \log \mathbf{t})$ rounds and in the low-memory MPC model in $O\left(\frac{1}{\delta} \log \mathbf{s} \log \mathbf{t}\right)$ rounds.

While this algorithm is closely related to the nearly-30-year-old parallel SetCover algorithm of Berger, Rompel, and Shor [3], it is similar in spirit to other parallel and distributed SetCover algorithms [4, 22, 30]. All of these algorithms emulate the standard, sequential greedy SetCover algorithm, but relax the sequential nature of the greedy algorithm by processing not just one "best" set, but multiple "good enough" sets in parallel. Specifically, all the sets S for which $\deg_{i,k}(S) \geq \mathbf{s}/2^i$ (in Line 4 Algorithm 1) are good enough and they are candidates for being added to the set cover. For example, in Stage 1, all sets with cardinality at least $\mathbf{s}/2$ are considered good enough. However, adding all of these good enough sets to the set cover can lead to a very large solution and damage the approximation factor. So sets are added in $\log \mathbf{t}$ iterations in a randomized fashion with geometrically increasing probability. For example, in iteration 1, sets are added with probability $2/\mathbf{t}$. Intuitively, this means that elements that belong to close to $\mathbf{t}$ sets are covered, but only by $O(1)$ sets. This allows the algorithm to double the set selection probability to $4/\mathbf{t}$ in the next iteration and so on. Adding candidate sets to the set cover in such a way that no element is covered by more than $O(1)$ sets leads to a $O(\log \mathbf{s})$-approximation. The $\log \mathbf{t}$ iterations in each stage, essentially do this type of local symmetry breaking in parallel. Our work in this paper is motivated by the fact that for MIS, which is a classical local symmetry breaking problem, there are techniques that lead to fast MPC algorithms.

2.1 Sparsified Graph Exponentiation

A popular technique [13, 15, 21, 28] for speeding up the simulation of Congest algorithms in "all-to-all" communication models is *graph exponentiation.* The main idea of this technique is that, if every node v knows the state of the k-neighborhood around v, then by

[3]Given an arbitrary distribution of edges across machines we can convert it to a node-centric distribution as follows: (1) for each edge $\{u, v\}$ create two ordered tuples (u, v) and (v, u) and (2) sort the list of tuples by node ID and machine ID so that earliest tuples in the ordering appear on machines with smallest ID. The result is that the edges incident on each node are now on the same or two machines with adjacent IDs. It is easy to go from this intermediate distribution to a node-centric distribution by having machines with adjacent IDs coordinate with each other. Since we can do sorting in MPC in constant rounds [16], a node-centric distribution be assumed without loss of generality.

Algorithm 1: SetCover "base algorithm" from [19]

```
input  : Ground set $X$, $|X| = n$, collection of sets
         $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$
output : A set cover $C \subseteq \mathcal{S}$
1 for stage $i \leftarrow 1$ to $\log \mathbf{s}$ do
2     for iteration $k \leftarrow 1$ to $\log \mathbf{t}$ do
3         for each set $S \in \mathcal{S}$ in parallel do
4             if $\deg_{i,k}(S) \geq \mathbf{s}/2^i$ then
5                 add $S$ to the set cover $C$ with probability
                  $2^k/\mathbf{t}$;
6             end
7         end
8     end
9 end
```

exchanging this information with all nodes, it is possible to learn the state of the $2k$-neighborhood around each node. If the information exchange in each iteration can be done in $O(1)$ rounds, then nodes can learn the state of their ℓ-neighborhood in $O(\log \ell)$ rounds. The main obstacle to obtaining this exponential speedup is that the k-neighborhoods around nodes may be so large that exchanging them may take too many rounds. However, many Congest algorithms are randomized and there is a natural *sparsification* that occurs, i.e., in each round only a randomly sampled subset of the nodes are active, and the rest are silent. This implies that the k-neighborhoods that are exchanged only need to involve sparse subgraphs induced by the sampled nodes. This *sparsified graph exponentiation* technique has been applied to the MIS problem in the low-memory MPC model [15]. Furthermore, a simulation theorem that allows us to apply sparsified graph exponentiation as a black box was presented in [25] and applied to the 2-ruling set problem and MIS. In this paper, we show that the sparsified graph exponentiation technique can be applied to Algorithm 1 to obtain fast simulations in both the low-memory and the linear-memory MPC models.

2.2 Sparsified Graph Exponentiation Applied to SetCover

Our overall plan is to partition the $\log \mathbf{t}$ iterations in a stage in Algorithm 1 into batches and simulate each batch fast, using sparsified graph exponentiation. Consider a batch with b iterations, $k = B+1, B+2, \ldots, B+b$ in Stage i. At the start of this batch, it is possible to identify sets that will be "active" in this batch. Call a set S a *candidate* for this batch if $\deg_{i,B}(S) \geq \mathbf{s}/2^i$. For each candidate set S, we toss b coins $c_k(S)$ for $k = B+1, B+2, \ldots, B+b$ with $\text{Prob}[c_k(S) = 1] = 2^k/\mathbf{t}$. Call a set S *selected* in this batch if it is a candidate for this batch and $c_k(S) = 1$ for some $B+1 \leq k \leq B+b$. It is clear from Lines 4-5 in Algorithm 1 that the only sets that can join the set cover in this batch are those that are selected in this batch.

Whether a selected set S is added to the set cover in this batch can be determined by examining the $2b$-hop neighborhood of S in the subgraph induced by selected sets and all elements. We could get a fast simulation of Algorithm 1 if we can use the graph exponentiation technique to gather this subgraph at S, for every selected set S in $O(\log b)$ rounds. Unfortunately, this graph is too large because while we have sparsified the sets (by keeping only selected sets), we have not sparsified the elements. To explain our idea of element sparsification, we first describe the role of elements in the sparsified graph. Consider a candidate set S for which $c_k(S) = 1$ for some $B+1 \leq k \leq B+b$, but $c_{k'}(S) = 0$ for all $B+1 \leq k' < k$. In other words, k denotes the first iteration in this batch in which S is selected. Even though S is selected in iteration k, whether it should be added to the set cover depends on whether S is still a candidate just before iteration k, i.e., if $\deg_{i,k}(S) \geq \mathbf{s}/2^i$. To check this S needs to know the *number* of uncovered elements it contains (though not necessarily which elements are uncovered). In order to sparsify elements, we use the insight that it is enough for S to have a good enough estimate of the number of uncovered elements it contains. It turns out that it is enough to randomly sample a small number of elements so that there are roughly $\Theta(\log N)$ sampled elements for each iteration in the neighborhood of each candidate set.

2.3 Main Results

Let $N = m + n$. For the low-memory MPC algorithm, each batch has a fixed length consisting of $\Theta(\delta\sqrt{\log N}/4)$ iterations, that can be simulated using sparsified graph exponentiation in $O(\log \log N)$ rounds whp. This leads to a low-memory MPC model algorithm that runs in

$$\tilde{O}\left(\frac{1}{\delta^2}\frac{\log \mathbf{s} \log \mathbf{t}}{\sqrt{\log N}}\right) = \tilde{O}\left(\frac{1}{\delta^2}\min\{\sqrt{\log \mathbf{s}}\log \mathbf{t}, \log \mathbf{s}\sqrt{\log \mathbf{t}}\}\right)$$

rounds (see Theorem 4.6). In the linear-memory MPC algorithm, we simulate the first $\log \mathbf{t} - O(\log \log N)$ iterations of each stage i by using the sparsification technique developed in the low-memory MPC algorithm. The interesting point here is that we do not need sparsified graph exponentiation because the active graph induced by the selected candidate sets and sampled elements in the first $\log \mathbf{t} - O(\log \log N)$ iterations has $O(N)$ size and can be sent to a single machine for local processing. Then the remaining $O(\log \log N)$ iterations can be executed one by one. Our linear-memory MPC algorithm has an overall running time of $O(\log \mathbf{s} \cdot \log \log N)$ rounds whp in the linear-memory MPC model (see Theorem 5.2).

3 Related Work

Harvey, Liaw, and Liu [20] present, what can be viewed as a $O(1)$-round superlinear-memory MPC algorithm giving a $\mathbf{t}$-approximation algorithm for weighted SetCover. More precisely, their algorithm runs in $O((c/\mu)^2)$ rounds, where the space per machine is assumed to be $\Theta(\mathbf{t} \cdot n^{1+\mu})$.

It is also worth mentioning that Grunau, Mitrović, Rubinfeld, and Vakilian [19] use their base algorithm to design algorithms in Local Computation Algorithms (LCA) model with small query complexity. In the LCA model one can perform queries on the input and learn about a small portion of the output. For example, for SetCover in the LCA model, we want to perform as few queries as possible to find out if a set S belongs to the approximate set cover solution. Some of the techniques used in the present paper (e.g., sparsified graph exponentiation) for SetCover in the MPC model may be implicit in [19] and are used to reduce query complexity in LCA. However, none of the results in [19] are in the MPC model.

In a recent paper, Götte, Kolb, Scheideler, and Werthmann [17] present a SetCover approximation algorithm in the *beeping* model[4] and also presents a variant that has low message complexity in the Congest model. Even though this algorithm has many features specific to the beeping model, at its core it is similar to the base algorithm in [19]. It is important to note that even though our work is in the MPC model, [19] is in the LCA model, and [17] is in the beeping and Congest models, the common goal in all of these is to reduce the message complexity of the algorithms without worsening the quality of the approximation.

Note added to the camera-ready version: As the current paper was being reviewed, we were alerted to a paper by Dhulipala, Dinitz, Łącki, and Mitrović posted on arxiv [10] (and to appear in DISC 2024 [9]), one of whose results is very similar to our result. In fact, the "new, surprisingly simple" algorithm referred to in the abstract of this paper [10] seems quite similar to the "base algorithm" from [19] that we refer to. We note that [10] uses this algorithm as a starting point for deriving multiple algorithms (with different approximation factors), not just in the MPC model, but also in the PRAM model.

4 Low-Memory MPC Algorithm

Fix an arbitrary stage i. The $\log \mathbf{t}$ iterations in the stage are partitioned into *batches*, each batch consisting of $\delta\sqrt{\log N}/4$ iterations. To be precise, let $P_j = \delta\sqrt{\log N}/4 \cdot (j-1)$ denote batch boundaries. Then batch j, for $j = 1, 2, \ldots$ consists of iterations $k = P_j + 1$ through $k = P_{j+1}$. Recall that a set S is called a *candidate* for batch j if $\deg_{i,P_j+1}(S) \geq \mathbf{s}/2^i$. Also recall that each candidate S tosses $\delta\sqrt{\log N}/4$ coins $c_k(S)$ for $P_j + 1 \leq k \leq P_{j+1}$ with $\text{Prob}[c_k(S) = 1] = 2^k/\mathbf{t}$ to mimic the random selection of candidates in Line 5 in Algorithm 1. A candidate set S is said to be *selected* if $c_k(S) = 1$ for some $P_j + 1 \leq k \leq P_{j+1}$. A candidate set S is said to be *selected in iteration* ℓ, $P_j+1 \leq \ell \leq P_{j+1}$ if $c_\ell(S) = 1$ and $c_k(S) = 0$ for all $P_j + 1 \leq k < \ell$. As mentioned in the previous section, we just need enough not-yet-covered elements to participate in the algorithm in batch j to obtain an up-to-date estimate of the number of not-yet-covered elements in each selected candidate set. For this purpose, we sample each element that is not covered by the start of batch j with probability $2c\log N \cdot 2^i/\mathbf{s}$ independently for each iteration in batch j, for some constant c that will be fixed later. This is done so that we can get independent estimates of the degree of each iteration.

Let $A = (\mathcal{S}_A, X_A, E_A)$ be the bipartite graph with node set $\mathcal{S}_A \cup X_A$, where $\mathcal{S}_A$ is the set of selected candidate sets in batch j and X_A is the set of sampled elements in batch j and the edge set E_A consists of edges $\{S, e\}$, where $S \in \mathcal{S}_A$, $e \in X_A$ and $e \in S$. We call A the *active graph* for batch j because it includes all sets and elements that might be active in batch j. Let $B_A(v, r)$ be the subgraph of A induced by nodes that are at most r hops from $v \in S_A \cup X_A$. Each node w in $B_A(v, r)$ representing a set is labeled with the following information: (a) $\deg_{i,P_j+1}(w)$ and (b) the coin toss sequence $(c_{P_j+1}(w), c_{P_j+2}(w), \ldots, c_{P_{j+1}}(w))$. The algorithm that simulates batch j in stage i is described in Algorithm 2.

Algorithm 2: Fast simulation of batch j in stage i in the low-memory MPC model

```
1 Every set that joined the set cover in the previous batch
    informs all of its elements;
2 Every element that got covered in the previous batch
    informs all sets it belongs to;
3 Every set $S$ computes $\deg_{i,P_j+1}(S)$ and sets $S$ with
    $\deg_{i,P_j+1}(S) \geq \mathbf{s}/2^i$ become candidates;
4 Every candidate set $S$ tosses $\delta\sqrt{\log N}/4$ biased coins
    $(c_{P_j+1}(S), c_{P_j+2}(S), \ldots, c_{P_{j+1}}(S))$ where
    $\text{Prob}[c_k(S) = 1] = 2^k/\mathbf{t}$ for $P_j + 1 \leq k \leq P_{j+1}$;
5 Every element not covered in previous batches samples
    itself independently with probability $2c\log N \cdot 2^i/\mathbf{s}$ in
    group $k$ for $P_j + 1 \leq k \leq P_{j+1}$ ;
6 For every selected candidate set $S$, gather $B_A(S, \delta\sqrt{\log N}/2)$
    at node $S$;
7 Using $B_A(S, \delta\sqrt{\log N}/2)$ every selected candidate set $S$
    computes $C(S)$, the subset of sampled elements in $S$;
8 Using $B_A(S, \delta\sqrt{\log N}/2)$ every candidate set $S$ selected in
    iteration $\ell$, $P_j + 1 \leq \ell \leq P_{j+1}$, computes $C_u(S)$, the subset
    of sampled neighbors in group $\ell$ not-yet-covered after $\ell - 1$
    iterations;
9 Every candidate $S$ for which
        $$\frac{|C_u(S)|}{|C(S)|} \cdot \deg_{i,P_j+1}(S) \geq \frac{\mathbf{s}}{2^i}$$
    joins the set cover.
```

As informally described in the introduction, for a set S that is selected in iteration ℓ, $P_j + 1 \leq \ell \leq P_{j+1}$ (i.e., in batch j), we use a sample of elements to estimate $\deg_{i,\ell}(S)$ and determine if $\deg_{i,\ell}(S) \geq \mathbf{s}/2^i$. The following lemma shows that the estimator used in Line 9 of Algorithm 2 for $\deg_{i,\ell}(S)$ is good enough. Specifically, set S may be incorrectly classified as a candidate when the number of not-yet-covered elements in S is very close, i.e., within a $\left[\frac{1}{(1+\epsilon)}, \frac{1}{(1-\epsilon)}\right]$ factor of the Stage i threshold of $\mathbf{s}/2^i$ for an arbitrarily small constant ϵ. However, this only worsens the approximation factor of the algorithm by a factor that depends on ϵ.

Lemma 4.1. *Suppose that S is a candidate selected in iteration ℓ, $P_j + 1 \leq \ell \leq P_{j+1}$. Let $C(S)$ be the subset of S of sampled elements in batch j. Let $C_u(S) \subseteq C(S)$ be the set of sampled neighbors of u that remain uncovered at the end of iteration $\ell - 1$. Then, for any constant $\epsilon > 0$, whp,*

$$\left(\frac{|C_u(S)|}{|C(S)|}\right) \cdot \deg_{i,P_j+1}(S) \geq \frac{\mathbf{s}}{2^i} \implies \deg_{i,\ell}(S) \geq \frac{\mathbf{s}}{2^i(1+\epsilon)},$$
$$\left(\frac{|C_u(S)|}{|C(S)|}\right) \cdot \deg_{i,P_j+1}(S) < \frac{\mathbf{s}}{2^i} \implies \deg_{i,\ell}(S) < \frac{\mathbf{s}}{2^i(1-\epsilon)}.$$

Proof. Let U denote the set of not-yet-covered elements in S just before the start of the batch. Thus, $\deg_{i,P_j+1}(S) = |U|$. Since S is

[4] In the beeping model nodes communicate with beeps. In the specific version of the beeping model in [17], in each round a node can either *listen* or *beep*. If, in a round, a node v is listening and some subset of its neighbors beep, then v hears a beep.

a candidate before the start of this batch, $|U| \geq \mathbf{s}/2^i$ and therefore $\mathbb{E}[|C(S)|] = (2c \log N \cdot 2^i/\mathbf{s}) \cdot |U| \geq 2c \log N$ and by Chernoff bounds

$$\text{Prob}\left[||C(S)| - \mathbb{E}[|C(S)|]| \geq \frac{\epsilon}{4} \cdot \mathbb{E}[|C(S)|]\right] \leq \frac{1}{N^3} \qquad (1)$$

for large enough constant c. Furthermore, $\mathbb{E}[|C_u(S)|] = (2c \log N \cdot 2^i/\mathbf{s}) \cdot \deg_{i,\ell}(S)$. We partition the rest of the proof into two cases depending on how large $\deg_{i,\ell}(S)$ is relative to $\mathbf{s}/2^{i+1}$.

Case (1): $\deg_{i,\ell}(S) \geq \mathbf{s}/2^{i+1}$. In this case, $\mathbb{E}[|C_u(S)|] \geq c \log N$ and by Chernoff bounds, for large enough constant c,

$$\text{Prob}\left[||C_u(S)| - \mathbb{E}[|C_u(S)|]| \geq \frac{\epsilon}{4} \cdot \mathbb{E}[|C_u(S)|]\right] \leq \frac{1}{N^3}. \qquad (2)$$

Let $\mathcal{E}_1$ denote the event $||C(S)| - \mathbb{E}[|C(S)|]| < \frac{\epsilon}{4} \cdot \mathbb{E}[|C(S)|]$ and similarly let $\mathcal{E}_2$ denote the event $||C_u(S)| - \mathbb{E}[|C_u(S)|]| < \frac{\epsilon}{4} \cdot \mathbb{E}[|C_u(S)|]$. Then, by bounds in (1) and (2), $\text{Prob}[\mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1 - 2/N^3 \geq 1 - 1/N^2$ for $N \geq 2$. Therefore, with probability at least $1 - 1/N^2$

$$\left(\frac{1-\epsilon/4}{1+\epsilon/4}\right) \cdot \frac{\deg_{i,\ell}(S)}{|U|} \leq \frac{|C_u(S)|}{|C(S)|} \leq \left(\frac{1+\epsilon/4}{1-\epsilon/4}\right) \cdot \frac{\deg_{i,\ell}(S)}{|U|}. \qquad (3)$$

For $\epsilon \leq 1$, the inequalities in (3) imply that with probability at least $1 - 1/N^2$,

$$(1-\epsilon) \cdot \deg_{i,\ell}(S) \leq \frac{|C_u(S)|}{|C(S)|} \cdot |U| \leq (1+\epsilon) \cdot \deg_{i,\ell}(S).$$

Case (2): $\deg_{i,\ell}(S) < \mathbf{s}/2^{i+1}$. In this case, $\mathbb{E}[|C_u(S)|] < c \log N$. Therefore, by Chernoff bounds $\text{Prob}[|C_u(S)| \geq (3c/2) \log N] \leq 1/N^3$. The bound in (1) also implies that $\text{Prob}[|C(S)| \leq ((3c/2) \log N \cdot 2^i/s) \cdot |U|] \leq 1/N^3$. Therefore, using a similar reasoning as in Case 1, with probability at least $1 - 1/N^2$, both the events $|C_u(S)| < (3c/2) \log N$ and $|C(S)| > ((3c/2) \log N \cdot 2^i/s) \cdot |U|$ occur. This implies that with probability at least $1 - 1/N^2$,

$$\frac{|C_u(S)|}{|C(S)|} \cdot |U| < \frac{(3c/2) \log N}{((3c/2) \log N \cdot 2^i/s) \cdot |U|} \cdot |U| = \frac{s}{2^i}.$$

To finish the proof we separately consider the two parts in the statement of the lemma.

- If $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \geq \mathbf{s}/2^i$, then with probability at least $1 - 1/N^2$ Case (1) holds. This is because if Case (2) were to hold, then with probability at least $1 - 1/N^2$, $(|C_u(S)|/|C(S)|) \cdot |U| < \mathbf{s}/2^i$. Thus, whp $\deg_{i,\ell}(S) \geq \mathbf{s}/2^{i+1}$ and therefore $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \leq (1+\epsilon) \cdot \deg_{i,\ell}(S)$. Hence, $\deg_{i,\ell}(S) \geq \mathbf{s}/2(1+\epsilon)$.
- If $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \geq \mathbf{s}/2^i$ then either Case (1) or (2) above may hold. If Case (1) holds and $\deg_{i,\ell}(S) \geq \mathbf{s}/2^{i+1}$ then $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \geq (1-\epsilon) \cdot \deg_{i,\ell}(S)$ and therefore $\deg_{i,\ell}(S) \leq \mathbf{s}/2^i(1-\epsilon)$. If Case (2) holds, then $\deg_{i,\ell}(S) < \mathbf{s}/2^{i+1} < \mathbf{s}/2^i(1-\epsilon)$.

□

The lemma above shows that the proposed estimator for $\deg_{i,\ell}(S)$ is good enough. To compute this estimator, a selected candidate set S needs to know the quantities $|C_u(S)|$, $|C(S)|$, and $\deg_{i,P_j+1}(S)$. The latter two quantities are available at the start of the batch just by communicating with neighbors. The following lemma shows that $C_u(S)$ can be computed via local computation on $B_A(S, \delta\sqrt{\log N}/2)$.

Lemma 4.2. *Suppose that a candidate node S is selected in iteration ℓ, $P_j + 1 \leq \ell \leq P_{j+1}$. Further suppose that machine hosting set S knows $B_A(S, \delta\sqrt{\log N}/2)$. Then, the machine can determine whp via local computation the set $C_u(S)$ of sampled neighbors who remain uncovered after $\ell - 1$ iterations.*

Proof. Let M_S denote the machine hosting set S. We will show by induction that for any h, $1 \leq h \leq \ell$, machine M_S can perform local computation on $B_A(S, \delta\sqrt{\log N}/2)$ to determine for any selected candidate set Y in $B_A(S, 2(\ell - h))$ whp, the set $C_u(Y)$ of sampled neighbors of Y who remain uncovered after k iterations for any $k \leq h - 1$. Plugging $h = \ell$ into this claim leads to the lemma.
Base case: $h = 1$. Consider a selected candidate set $Y \in B_A(S, 2(\ell - 1))$. Let $C(Y)$ denote the set of sampled neighbors of Y. After $h - 1 = 0$ iterations all nodes in $C(Y)$ remain uncovered. Since all elements in $C(Y)$ belong to $B_A(S, 2(\ell-1)+1) \subseteq B_A(S, \delta\sqrt{\log N}/2)$, machine M_S can compute the set $C_u(Y) = C(Y)$ from $B_A(S, \delta\sqrt{\log N}/2)$ using the first group of sampled elements.
Inductive step: We assume the inductive claim is true after iteration h. In other words, for any selected candidate set $Y \in B_A(S, 2(\ell - h))$ and $k \leq h - 1$, machine M_S can compute the set $C_u(Y)$ of sampled neighbors of Y who remain uncovered after k iterations via local computation on $B_A(S, \delta\sqrt{\log N}/2)$ using the first $h - 1$ groups of sampled elements. Therefore, for each such Y and k, machine M_S knows if $(|C_u(Y)|/|C(Y)|) \cdot \deg_{i,k}(Y) \geq \mathbf{s}/2^i$ and if Y is selected in iteration $k+1$. This implies that for each such Y and k, machine M_S knows if Y joins the set cover in iteration $k+1$. Now consider any sampled element $x \in B_A(S, 2(\ell-h)-1)$ belonging to a group $k' \leq h$. All neighbors of x that are selected candidates are in $B_A(S, 2(\ell-h))$. Therefore, for each $k \leq h-1$, machine M_S can figure out via local computation on $B_A(S, \delta\sqrt{\log N}/2)$ if x is covered after $k' = k+1$ iterations have been completed. This implies that for any selected candidate set $Y \in B_A(S, 2(\ell-h)-2) = B_A(S, 2(\ell-(h+1)))$ and any $k' \leq h$, machine M_S can compute $C_u(Y)$, the set of sampled neighbors in group k' that remain uncovered after k' iterations. This establishes the inductive hypothesis after iteration $h+1$. □

In the next lemma, we show an upper bound on the maximum degree of the active graph. Subsequently, we use this (in Lemma 4.4) to show that every $\delta\sqrt{\log N}/2$-neighborhood in the active graph has size $O(N^{\delta/2})$.

Lemma 4.3. *At the start of batch j, whp every not-yet-covered element x has at most $\mathbf{t} \cdot \ln N/2^{P_j}$ candidate neighbors.*

Proof. Consider an element $x \in X$ that is not yet covered at the start of batch j. Suppose x has more than $\mathbf{t} \cdot \ln N/2^{P_j}$ candidate neighbors at the start of batch j. Each of these candidates is selected with probability $2^{P_j+1}/\mathbf{t}$ in batch $j-1$. Then the probability that none of these candidates are selected in batch j is

$$\left(1 - \frac{2^{P_j+1}}{\mathbf{t}}\right)^{\mathbf{t}\ln N/2^{P_j}} \leq \exp(2\ln N) = \frac{1}{N^2}.$$

Thus with probability at least $1 - 1/N^2$ such an element x is covered in batch j. By the union bound, with probability at least $1 - 1/N$

every element $x \in X$ not-yet-covered at the start of batch j has at most $\mathbf{t} \cdot \ln N / 2^{P_{j+1}}$ candidate neighbors. □

Lemma 4.4. *For any node $x \in A$, the number of edges in the ball $B_A(x, \delta\sqrt{\log N}/2)$ is bounded by $O(N^{\delta/2})$ whp.*

Proof. By Lemma 4.3 each not-yet-covered element has at most $\mathbf{t} \ln N/2^{P_j}$ neighboring candidates with high probability. Furthermore, each candidate set is selected with probability at most $2 \cdot 2^{P_{j+1}}/\mathbf{t}$. Thus, in expectation, a not-yet-covered element has at most $2 \ln N \cdot 2^{\delta\sqrt{\log N}/4}$ selected candidate neighbors. By Chernoff bounds, a not-yet-covered node has more than $(6 \log N) 2 \ln N \cdot 2^{\delta\sqrt{\log N}/4}$ with probability at most $2^{-6\log N} = 1/N^6$. Thus, whp, every not-yet-covered element has at most $(6\log N)2\ln N \cdot 2^{\delta\sqrt{\log N}/4} = O(\log^2 N \cdot 2^{\delta\sqrt{\log N}/4})$ selected candidate neighbors.

On the other hand, every not-yet-covered element e is sampled with probability $2c \log N \cdot 2^i/\mathbf{s}$. At the beginning of stage i the number of uncovered elements of any set not currently in the set cover is at most $\mathbf{s}/2^{i-1}$, since any set with more not-yet-covered elements would have been a candidate in the previous stage and was either added to the set cover or now has fewer than $\mathbf{s}/2^{i-1}$ uncovered elements. Thus, the expected number of sampled not-yet-covered elements in any candidate set in this batch is $4c \log N$. By Chernoff bounds, the probability that a set not in the current set cover contains more than $24c \log N$ sampled uncovered elements is less than $2^{-24c\log N} = 1/N^{24c}$. Thus whp every set not in the current set cover has at most $24c \log N$ sampled uncovered elements, from one group. And since we independently sample a group of elements for each iteration in the batch, whp every set not in the current set cover has at most $24c \log^{1.5} N$ sampled uncovered elements overall.

Combining the previous two results, the degree of the active graph is bounded by $\max\{O(\log^2 N \cdot 2^{\delta\sqrt{\log N}/4}), O(\log^{1.5} N)\} = O(\log^2 N \cdot 2^{\delta\sqrt{\log N}/4})$ whp. Next, we can bound the number of edges in $B_A(x, \delta\sqrt{\log N}/2)$ by $(C \log^2 N \cdot 2^{\delta\sqrt{\log N}/4})^{\delta\sqrt{\log N}/2}$ for some constant C. This quantity is bounded above by $O(N^{\delta/2})$. □

Lemma 4.5. *Suppose that the total memory, in all machines, is $\Omega(N^{1+\delta})$. Then, for every set $S \in \mathcal{S}_A$, the subgraph $B_A(S, \delta\sqrt{\log N}/2)$ can be gathered at S (in parallel) in $O(\log\log N)$ rounds.*

Proof. Suppose the total memory is $\Omega(N^{1+\delta})$. In that case, there are $\Omega(N)$ machines (since per machine memory is $O(N^\delta)$) and therefore we can assume without loss of generality that every machine hosts $O(1)$ sets or elements. For a node v in the active graph A, let M_v denote the machine hosting v. Suppose that each machine M_v knows $B_A(v, p)$ for some $0 \le p \le \delta\sqrt{\log N}$. Each machine M_v then sends $B_A(v, p)$ to machine M_u for every node u in $B_A(v, p)$. After this communication is completed, each machine M_v can construct $B_A(v, 2p)$ from the information it has received because $B_A(v, 2p)$ is contained in the union of $B_A(u, p)$ for all u in $B_A(v, p)$.

We now argue that this communication can be performed in $O(1)$ rounds. First, note that by Lemma 4.4, the size of $B_A(v, p)$ is bounded above by $O(N^{\delta/2})$. This also means that $B_A(v, p)$ contains $O(N^{\delta/2})$ nodes. Therefore, M_v needs to send a total of $O(N^{\delta/2}) \times O(N^{\delta/2}) = O(N^\delta)$ words. A symmetric argument shows a $O(N^\delta)$ bound on the number of words M_v receives. Since $O(N^\delta)$ words can be sent and received in each communication round by a machine this communication can be completed in $O(1)$ rounds. □

Theorem 4.6. *There is a low-memory MPC algorithm that runs in $\tilde{O}\left(\frac{1}{\delta^2} \log \mathbf{s} \log \mathbf{t}/\sqrt{\log N}\right)$ rounds and produces an $O(\log \mathbf{s})$-approximate set cover whp. This algorithm needs $O(I + N^{1+\delta})$ total memory.*

Proof. The purpose of Line 1 in Algorithm 2 is for every element to know if it is covered by a set added to the set cover in the previous batch. This boolean information can be aggregated up the depth-$O(1/\delta)$-tree representing the neighborhood of each element. Hence, Line 1 can be implemented in $O(1/\delta)$ rounds. The purpose of Line 2 in Algorithm 2 is for every set S not in the set cover to know $\deg_{i,P_j+1}(S)$, i.e., the number of elements in S not-yet-covered at the end of the previous batch. This count can be aggregated up the depth-$O(1/\delta)$-tree representing the neighborhood of each set. Hence, Line 2 can be implemented in $O(1/\delta)$ rounds. Lines 3-5 involve local computation only. Line 6 is the graph exponentiation step, which takes $O(\log\log N)$ rounds in the low-memory MPC model according to Lemma 4.5. Lines 7-9 only require local computation. Hence, Algorithm 1, which simulates a batch of iterations, can be implemented in $O(1/\delta + \log\log N)$ rounds. Each batch consists of $\delta\sqrt{\log N}/4$ iterations and thus there are $4 \log \mathbf{t}/\delta\sqrt{\log N}$ batches in a stage. This implies a total of $O\left(\frac{1}{\delta^2} \log \mathbf{t} \log\log N/\sqrt{\log N}\right)$ rounds per stage and thus a total of $\tilde{O}\left(\frac{1}{\delta^2} \log \mathbf{s} \log \mathbf{t}/\sqrt{\log N}\right)$ rounds.

The fact that Algorithm 1 is an $O(\log s)$-approximation (in expectation) is shown in [18, 19]. Algorithm 2 simulates Algorithm 1 faithfully expects that a set S may be incorrectly classified as a candidate when the number of not-yet-covered elements in S is very close to the Stage i threshold of $\mathbf{s}/2^i$. Specifically, in Algorithm 1 a set S is a candidate in iteration ℓ in Stage i if $\deg_{i,\ell}(S) \ge \mathbf{s}/2^i$. But, as per Lemma 4.1, S may be classified as a candidate even if $\deg_{i,\ell}(S)$ is slightly smaller, i.e., in the range $\left[\frac{\mathbf{s}}{2^i(1+\epsilon)}, \frac{\mathbf{s}}{2^i}\right)$. Similarly, S may be may be classified as a non-candidate even if $\deg_{i,\ell}(S)$ is slightly larger, i.e., in the range $\left[\frac{\mathbf{s}}{2^i}, \frac{\mathbf{s}}{2^i(1-\epsilon)}\right)$. This approximation candidate-estimation only worsens the approximation factor by a factor that depends on ϵ, which we can take to be an arbitrarily small constant.

The $O(\log \mathbf{s})$-approximation factor holds only in expectation. We can make the $O(\log \mathbf{s})$-approximation hold whp (with a larger constant) using the standard technique of running $O(\log N)$ copies of the algorithm and returning the smallest solution. Standard Chernoff bounds show that the minimum of these solutions has an approximation ratio $O(\log \mathbf{s})$ whp. This increases our memory usage (both local and total) by a factor of $\log N$ but this can be absorbed in the big-Oh notation by reducing δ to 0.9δ since $\log N = O(N^{0.1\delta})$. This only increases the round complexity by a constant factor. □

4.1 Linear Total Memory

In this section, we describe how to modify Algorithm 2 to get linear total memory. This comes at the cost of a slightly higher round complexity compared to Theorem 4.6.

Theorem 4.7. *There is a low-memory MPC algorithm that runs in $\tilde{O}\left(\frac{1}{\delta^2}\sqrt{\log \mathbf{s}}\log \mathbf{t}\right)$ rounds and produces an $O(\log \mathbf{s})$-approximate set cover whp. This algorithm needs $\tilde{O}(I)$ total memory.*

Proof. To use less total memory, we modify Algorithm 2 so that each batch in stage i executes $R = \delta\sqrt{\log \mathbf{s}}/4$ iterations. Therefore, the size of the active graph in Lemma 4.4 is bounded above by $O(\mathbf{s}^{\delta/2})$ whp. In stage i, only candidates with degree in range $[\mathbf{s}/2^i, \mathbf{s}/2^{i-1})$ are active. Therefore until stage $(1-\delta/2)\log \mathbf{s}$ we can charge the active graph collected at an active set S to unique input graph edges for each batch since $\deg(S) \geq \mathbf{s}^{\delta/2}$. These stages are simulated in $O\left(\frac{1}{\delta^2}\log \mathbf{t}\sqrt{\log \mathbf{s}}\right)$ rounds. Now we want to simulate the remaining $(\delta/2)\log \mathbf{s}$ stages. We shrink the batch size to $R/2$ and this allows us to simulate an additional $(\delta/4)\log \mathbf{s}$ stages while still charging the active graph (of size $O(\mathbf{s}^{\delta/4})$) to unique input edges for each batch. We keep halving the batch size at most $O(\log\log \mathbf{s})$ times where we simulate $(\delta/2^{t+1})\log \mathbf{s}$ stages with batch size $R/2^t$ until we simulate all stages. This simulation runs in $\tilde{O}\left(\frac{1}{\delta^2}\sum_{t=1}^{\log\log \mathbf{s}}\delta 2^t \log \mathbf{s}\log \mathbf{t}/2^{t+1}R\right) = \tilde{O}\left(\frac{1}{\delta^2}\log \mathbf{t}\sqrt{\log \mathbf{s}}\right)$ rounds. Each input graph edge is charged a most $O(\log \mathbf{s}\log \mathbf{t})$ times so the total memory used is $\tilde{O}(I)$. □

5 Exponentially Faster Simulation in Linear-Memory MPC

In this section we will show that in the linear-memory MPC model the simulation from the previous section can be substantially simplified and accelerated, leading to an algorithm that runs $O(\log \mathbf{s} \cdot \log\log N)$ rounds whp. Our main insight is that the active graph induced by a large number of initial iterations in a stage has size $O(N)$. This implies that these initial iterations can be simulated in constant rounds in the linear-memory MPC model.

In particular, for any stage i consider iterations 1 to $\log \mathbf{t} - 2\log\log N$. Each candidate has a probability $\sum_1^{\log \mathbf{t}-3\log\log N} 2^k/\mathbf{t} \leq 2 \cdot 2^{\log \mathbf{t}-3\log\log N}/\mathbf{t} \leq 2/(\log^3 N)$ of being selected. In addition to selecting candidates, each element that is not covered by the start of batch j in stage i is sampled independently in $\log \mathbf{t} - 3\log\log N$ groups, each with probability $2c\log N \cdot 2^i/\mathbf{s}$, for some constant c. Let $A = (\mathcal{S}_A, X_A, E_A)$ be the active graph for stage i and batch j which is the bipartite graph with node set $\mathcal{S}_A \cup X_A$, where $\mathcal{S}_A$ is the set of selected candidate sets and X_A is the set of sampled elements and the edge set E_A consists of edges $\{S, e\}$, where $S \in \mathcal{S}_A$, $e \in X_A$, and $e \in S$.

Lemma 5.1. *The active graph A for iterations 1 to $\log \mathbf{t} - 3\log\log N$ of stage i has $O(N/\log N)$ edges whp.*

Proof. Each candidate will select itself in the first $\log \mathbf{t} - 3\log\log N$ iterations with probability at most $2/\log^3 N$. Therefore, the expected number of selected candidates that is $\mathbb{E}[|\mathcal{S}_A|] \leq 2N/\log^2 N$. Since the candidates select themselves independently, we can use Chernoff bound to say that whp, $|\mathcal{S}_A| \leq 2cN/\log^2 N$.

Now since we are in stage i, each set has degree at most $\mathbf{s}/2^i$ and each element samples itself independently, in $\log \mathbf{t} - 3\log\log N$ groups, each with probability $2c\log N \cdot 2^i/\mathbf{s}$. Therefore, using Chernoff bound, we can say that whp, every candidate at most $O(\log N \log \mathbf{t}) = O(\log^2 N)$ sampled neighbors. Therefore, $|E_A| \leq |\mathcal{S}_A| \cdot O(\log^2 N) \leq O(N/\log N)$ whp, and the lemma holds. □

We are now ready to prove the main result of this section.

Theorem 5.2. *There is a linear-memory MPC algorithm that runs in $O(\log \mathbf{s} \cdot \log\log N)$ rounds and produces an $O(\log \mathbf{s})$-approximate set cover whp. This algorithm uses a total of $\tilde{O}(I)$ memory.*

Proof. We simulate iterations 1 to $\log \mathbf{t} - 3\log\log N$ of stage i by having each machine compute the local part of the active graph A and send it to a single leader machine. Then the leader machine can simulate the $\log \mathbf{t} - 3\log\log N$ iterations locally using Algorithm 2 and send to each set and element its local outcome. This requires $O(1)$ rounds.

The remaining $3\log\log N$ iterations can be simulated one by one, and then we move on to stage $i+1$. Hence stage i can be simulated in $O(\log\log N)$ rounds. Therefore, the overall round complexity is $O(\log \mathbf{s} \cdot \log\log N)$.

Similar to the proof of Theorem 4.6, the use of the estimators of Lemma 4.1 only worsens the expected $O(\log \mathbf{s})$ approximation factor of the base Algorithm 1 by a factor that depends on ϵ, which we can make an arbitrarily small constant.

The $O(\log \mathbf{s})$-approximation factor we get only holds for the expected cost returned by the algorithm. We can make the $O(\log \mathbf{s})$-approximation hold whp (with a larger constant) using the standard technique of running $O(\log N)$ copies of the algorithm and returning the smallest solution. For each copy, the active graph in stage i has size $O(N/\log N)$ whp, and we assign $O(\log N)$ different leader machines that gather each active graph, simulate all iterations and send the outcome to each set and element. Therefore at each stage, each machine sends and receives at most $O((N/\log N) \cdot \log N) = O(N)$ messages, and we have no overhead in round complexity.

We can also simulate the remaining $O(\log\log N)$ for each copy in parallel by noting that a faithful simulation of Algorithm 1 only requires each set added to the cover to send *one bit* to each element it covers and each uncovered element sends *one bit* to its neighboring sets, so it can update the $\deg_{i,k}(\cdot)$ value for the next iteration k. Since each set an element sends only one bit, we can fit the messages of the $O(\log N)$ copies in $O(1)$ words of local memory by having an implicit ordering over all the copies. Therefore, the local memory usage for each machine is still $O(N)$ words.

Once each set knows which of the $O(\log N)$ set covers it belongs to, we can aggregate this information into one machine that can compute all $O(\log N)$ costs. Each set sends a bit string of size $O(\log N)$ to this machine denoting all the copies in which it belongs to the set cover solution. Therefore we have $O(\log N)$ solutions generated independently, each with expected approximation ratio $O(\log \mathbf{s})$. Standard Chernoff bounds show that the minimum of these solutions has an approximation ratio $O(\log \mathbf{s})$ whp.

Note that for the entire simulation we need only $O(\log N)$ additional machines and since $I = \Omega(N)$, the additional $O(N\log N)$ total memory is absorbed into the $\tilde{O}(\cdot)$ notation. □

6 Conclusion and Future Work

This paper presents the first $\tilde{O}(\log N)$-round and $\tilde{O}(\log^{1.5} N)$-round algorithms in the linear-memory and low-memory MPC models respectively for the classical SetCover problem. These algorithms are obtained by using the technique of sparsified graph exponentiation, which was previously used to obtain faster MPC algorithms for MIS [15] and 2-ruling sets [25].

Several natural questions arise from this work.

First, is it possible to design $o(\log N)$-round MPC algorithms for SetCover, at least in the linear-memory MPC model? Some impressive progress has been made on derandomizing MPC algorithms for MIS [6] and coloring [7]. Can these techniques be extended to yield fast, deterministic algorithms for SetCover? In combinatorial optimization, researchers have considered many generalizations of SetCover such as partial SetCover, capacitated SetCover, multi-SetCover, etc. Designing fast MPC algorithms for these generalizations would be a natural follow-up to this work.

References

[1] Noga Alon, László Babai, and Alon Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *J. Algorithms* 7, 4 (dec 1986), 567–583. https://doi.org/10.1016/0196-6774(86)90019-2

[2] Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. 2019. Massively Parallel Computation of Matching and MIS in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (Toronto ON, Canada) *(PODC '19)*. Association for Computing Machinery, New York, NY, USA, 481–490. https://doi.org/10.1145/3293611.3331609

[3] Bonnie Berger, John Rompel, and Peter W. Shor. 1994. Efficient NC Algorithms for Set Cover with Applications to Learning and Geometry. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science* (Research Triangle Park, North Carolina, USA). Academic Press, Inc., USA, 454–477.

[4] Keren Censor-Hillel and Michal Dory. 2021. Distributed Spanner Approximation. *SIAM J. Comput.* 50, 3 (2021), 1103–1147. https://doi.org/10.1137/20M1312630 arXiv:https://doi.org/10.1137/20M1312630

[5] V. Chvatal. 1979. A Greedy Heuristic for the Set-Covering Problem. *Math. Oper. Res.* 4, 3 (aug 1979), 233–235. https://doi.org/10.1287/moor.4.3.233

[6] Artur Czumaj, Peter Davies, and Merav Parter. 2020. *Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space.* Association for Computing Machinery, New York, NY, USA, 175–185. https://doi.org/10.1145/3350755.3400282

[7] Artur Czumaj, Peter Davies, and Merav Parter. 2021. Simple, Deterministic, Constant-Round Coloring in Congested Clique and MPC. *SIAM J. Comput.* 50, 5 (2021), 1603–1626. https://doi.org/10.1137/20M1366502 arXiv:https://doi.org/10.1137/20M1366502

[8] Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round compression for parallel matching algorithms. *Proceedings of the Annual ACM Symposium on Theory of Computing* 14, 1 (2018), 471–484.

[9] Laxman Dhulipala, Michael Dinitz, Jakub Łącki, and Slobodan Mitrović. 2024. Parallel Set Cover and Hypergraph Matching via Uniform Random Sampling. In *38th International Symposium on Distributed Computing (DISC 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 319)*, Dan Alistarh (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 19:1–19:23. https://doi.org/10.4230/LIPIcs.DISC.2024.19

[10] Laxman Dhulipala, Michael Dinitz, Jakub Łącki, and Slobodan Mitrović. 2024. Parallel Set Cover and Hypergraph Matching via Uniform Random Sampling. arXiv:2408.13362 [cs.DS] https://arxiv.org/abs/2408.13362

[11] Irit Dinur and David Steurer. 2014. Analytical Approach to Parallel Repetition. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing* (New York, New York) *(STOC '14)*. Association for Computing Machinery, New York, NY, USA, 624–633. https://doi.org/10.1145/2591796.2591884

[12] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. 2010. On Distributing Symmetric Streaming Computations. *ACM Trans. Algorithms* 6, 4, Article 66 (sep 2010), 19 pages. https://doi.org/10.1145/1824777.1824786

[13] Mohsen Ghaffari. 2017. Distributed MIS via All-to-All Communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (Washington, DC, USA) *(PODC '17)*. Association for Computing Machinery, New York, NY, USA, 141–149. https://doi.org/10.1145/3087801.3087830

[14] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. Association for Computing Machinery, New York, NY, USA, 129–138.

[15] Mohsen Ghaffari and Jara Uitto. 2019. 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 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, Philadelphia, PA 19104, 1636–1653. https://doi.org/10.1137/1.9781611975482.99

[16] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, Searching, and Simulation in the Mapreduce Framework. In *Proceedings of the 22nd International Conference on Algorithms and Computation* (Yokohama, Japan) *(ISAAC'11)*. Springer-Verlag, Berlin, Heidelberg, 374–383. https://doi.org/10.1007/978-3-642-25591-5_39

[17] Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. 2021. Beep-And-Sleep: Message and Energy Efficient Set Cover. In *Algorithms for Sensor Systems: 17th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2021, Lisbon, Portugal, September 9–10, 2021, Proceedings* (Lisbon, Portugal). Springer-Verlag, Berlin, Heidelberg, 94–110. https://doi.org/10.1007/978-3-030-89240-1_7

[18] Christoph Grunau, Slobodan Mitrovic, Ronitt Rubinfeld, and Ali Vakilian. 2019. Improved Local Computation Algorithm for Set Cover via Sparsification. *CoRR* abs/1910.14154 (2019), 2993–3011. arXiv:1910.14154 http://arxiv.org/abs/1910.14154

[19] Christoph Grunau, Slobodan Mitrović, Ronitt Rubinfeld, and Ali Vakilian. 2020. Improved Local Computation Algorithm for Set Cover via Sparsification. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 3600 University City Science Center Philadelphia, PA, United States, 2993–3011. https://doi.org/10.1137/1.9781611975994.181

[20] Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. 2018. Greedy and Local Ratio Algorithms in the MapReduce Model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, Christian Scheideler and Jeremy T. Fineman (Eds.). ACM, New York, NY, USA, 43–52. https://doi.org/10.1145/3210377.3210386

[21] James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. 2014. Near-Constant-Time Distributed Algorithms on a Congested Clique. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8784)*, Fabian Kuhn (Ed.). Springer, Berlin, Germany, 514–530. https://doi.org/10.1007/978-3-662-45174-8_35

[22] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. 2002. An efficient distributed algorithm for constructing small dominating sets. *Distributed Comput.* 15, 4 (2002), 193–205. https://doi.org/10.1007/s00446-002-0078-0

[23] David S. Johnson. 1974. Approximation Algorithms for Combinatorial Problems. *J. Comput. Syst. Sci.* 9, 3 (dec 1974), 256–278. https://doi.org/10.1016/S0022-0000(74)80044-9

[24] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, Society for Industrial and Applied Mathematics, USA, 938–948.

[25] Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju. 2020. Sample-And-Gather: Fast Ruling Set Algorithms in the Low-Memory MPC Model. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference) (LIPIcs, Vol. 182)*, Nitin Saxena and Sunil Simon (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Oktavie-Allee, 66687 Wadern, Germany, 28:1–28:18. https://doi.org/10.4230/LIPIcs.FSTTCS.2020.28

[26] L. Lovász. 1975. On the Ratio of Optimal Integral and Fractional Covers. *Discrete Math.* 13, 4 (jan 1975), 383–390. https://doi.org/10.1016/0012-365X(75)90058-8

[27] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053. https://doi.org/10.1137/0215074 arXiv:https://doi.org/10.1137/0215074

[28] Merav Parter and Eylon Yogev. 2018. Congested Clique Algorithms for Graph Spanners. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018 (LIPIcs, Vol. 121)*, Ulrich Schmid and Josef Widder (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, USA, 40:1–40:18. https://doi.org/10.4230/LIPIcs.DISC.2018.40

[29] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 3600 University City Science Center Philadelphia, PA, United States.

[30] Sridhar Rajagopalan and Vijay V. Vazirani. 1999. Primal-Dual RNC Approximation Algorithms for Set Cover and Covering Integer Programs. *SIAM J. Comput.* 28, 2 (feb 1999), 525–540. https://doi.org/10.1137/S0097539793260763

[31] Grigory Yaroslavtsev and Adithya Vadapalli. 2018. Massively Parallel Algorithms and Hardness for Single-Linkage Clustering under ℓ_p Distances. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, US, 5600–5609. https://proceedings.mlr.press/v80/yaroslavtsev18a.html