Theoretical Computer Science 1086 (2026) 116238

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Faster set cover in the MPC model

Hongyan Ji [a], Shreyas Pai [b], Sriram V. Pemmaraju [a,*], Joshua Sobel [a]

[a] Department of Computer Science, The University of Iowa, Iowa City, IA, 52242, USA
[b] Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, Tamil Nadu, 600036, India

ARTICLE INFO

Editor: Prof. Seth Pettie
Handling Editor: Katie Harris

Keywords:
Parallel set cover
Massively parallel computing model
Approximation algorithms

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).

1. Introduction

We present fast algorithms for the SetCover problem in the Massively Parallel Computation (MPC) model [1–4]. 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 $\mathcal{M}$ 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, where $N = n + m$). 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 gives a $\ln n + O(1)$-approximation [5–7]. This algorithm is essentially optimal since there can be no $c \ln n$-approximation to SetCover for any $c < 1$ unless $P = NP$ [8]. The SetCover problem has also been extensively studied in parallel settings and $O(\log n)$-approximation algorithms are known [9,10], and these algorithms have served as the basis for *distributed* SetCover approximation algorithms [11,12]. For example, in the Congest model[1] of distributed computation, the

* Corresponding author.
E-mail addresses: hongyan-ji@uiowa.edu (H. Ji), shreyas@cse.iitm.ac.in (S. Pai), sriram-pemmaraju@uiowa.edu (S.V. Pemmaraju), joshua-sobel@uiowa.edu (J. Sobel).

[1] The Congest model [13] 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.

https://doi.org/10.1016/j.tcs.2026.116238
Received 20 August 2025; Received in revised form 23 May 2026; Accepted 4 September 2026
Available online 8 September 2026

work of Censor-Hillel and Dory [12] 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 $c \geq 1$.

There is an active stream of research that aims to design MPC algorithms that are significantly faster than their Congest counterparts for various problems [14–18]. In particular, for the Maximal Independent Set (MIS) problem, the fastest algorithm in the Congest model is a randomized algorithm with round complexity (in terms of the number of vertices, N) $O(\log N)$ [19,20], whereas in the low-memory MPC model it is $\tilde{O}(\sqrt{\log N})$[2] [15,16] 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 [12] one round at a time, taking $O(\log^2 N)$ rounds. In this paper, 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 $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets $S_i \subseteq X$ satisfying $\cup_i S_i = X$. The output of the problem is required to be the smallest subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that $\cup_{S \in \mathcal{C}} S = X$. 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 $X \cup \mathcal{S}$ 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 W of each machine and the number $|\mathcal{M}|$ of machines used are both assumed to be sublinear in 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, i.e., $|\mathcal{M}| \cdot W \geq I$.

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, we still need to ensure that the local memory parameter W is sublinear in the total input size I. Note that I can be as large as $m \cdot n$.

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

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

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

It is known that SetCover can be solved (approximately) in $O(1)$ rounds in the superlinear-memory MPC model [21] (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 [15,16,22]. 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 [23]. 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

[2] The $\tilde{O}(\cdot)$ notation hides poly $\log \log N$ factors.

[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. Assuming that the neighborhood of every node can fit in a single machine's memory, the result is that the edges incident on each node are now on the same machine 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 [2], a node-centric distribution be assumed without loss of generality.

just before iteration k in stage i. This algorithm produces an $O(\log \mathbf{s})$-approximate solution in expectation to SetCover [23]. 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.

This algorithm is closely related to the nearly-30-year-old parallel SetCover algorithm of Berger, Rompel, and Shor [10], and it is also similar in spirit to other parallel and distributed SetCover algorithms [9,11,12]. 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, Iteration 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 in a stage, sets are added with probability $2/\mathbf{t}$. Intuitively, this means that elements that belong to close to $\mathbf{t}$ sets are covered with at least a constant probability, and all elements are covered by at most $O(1)$ 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 an $O(\log \mathbf{s})$-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.

Algorithm 1: SetCover "base algorithm" from [23].

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 ← 1 to log s do
2     for iteration k ← 1 to log t do
3         for each set S ∈ S in parallel do
4             if deg_{i,k}(S) ≥ s/2^i then
5                 add S to the set cover C with probability 2^k/t;
6             end
7         end
8     end
9 end
```

2.1. Sparsified graph exponentiation

A popular technique [15,24–26] 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 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, some randomized Congest algorithms break symmetry by only activating a randomly sampled subset of the nodes in each round, while the rest remain silent (or inactive). This naturally sparsifies the instance, and 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 [16] 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, where B is the index of the iteration just before the batch starts. 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 $\deg_{i,B}(S) \geq \mathbf{s}/2^i$. A key observation is that we can execute the exact same algorithm as Algorithm 1, except that the coins are flipped in advance, prior to the start of the batch. Specifically, for each set S that is a candidate for the batch, we toss b coins, $c_k(S)$ for $k = B+1, B+2, \ldots, B+b$ with $\text{Prob}[c_k(S) = 1] = 2^k/\mathbf{t}$, and replace Line 5 in Algorithm 1 by "if $c_k(S) = 1$, add S to the set cover." This is possible because the biases of the coins are all known in advance. 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 that the only sets that can join the set cover in this batch are those that are selected in this batch, though not all selected sets will join the set cover.

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. A key result we show is that for $b = \delta\sqrt{\log N}/4$, the $2b$-hop neighborhood of S in the subgraph induced by selected sets and *sampled* elements is small enough, i.e., has size $O(N^\delta)$, to completely fit into a single machine in the low-memory MPC model.

2.3. Main results

Let $N = m + n$. For the low-memory MPC algorithm, each batch has a fixed length consisting of $\delta\sqrt{\log N}/4$ iterations, which 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 7).

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 (see Theorem 10).

3. Related work

Harvey, Liaw, and Liu [21] present what can be viewed as an $O(1)$-round superlinear-memory MPC algorithm giving a $\mathbf{t}$-approximation algorithm for weighted SetCover. More precisely, on a graph with n^{1+c} edges, 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 [23] use their base algorithm to design algorithms in the 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 [23] and are used to reduce query complexity in LCA. However, none of the results in [23] are in the MPC model.

In a recent paper, Götte, Kolb, Scheideler, and Werthmann [27] present a SetCover approximation algorithm in the *beeping* model[4] and also present 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 [23]. It is important to note that even though our work is in the MPC model, Grunau et al. [23] is in the LCA model, and [27] 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.

As the conference version of this paper was being reviewed, we were alerted to a paper by Dhulipala, Dinitz, Łącki, and Mitrović [28], containing results that are identical to our two results. In fact, the "new, surprisingly simple" algorithm referred to in the abstract of this paper [28] seems quite similar to the "base algorithm" from [23] that we refer to. At a high level, the two papers use similar ideas: tossing coins ahead of time, element sampling for up-to-date uncovered degree estimation, and sparsified graph exponentiation, though there are some technical differences, e.g., the formula used to estimate uncovered degrees. We posit that our exposition of both the algorithm and its analysis is substantially simpler, possibly because we focus exclusively on a single approximation algorithm in the MPC model. The authors of [28] derive two algorithms, one with $(1+\epsilon)\mathbf{t}$-approximation and the other with $(1+\epsilon)H_{\mathbf{s}}$-approximation[5], and try to analyze both within a common analysis framework, which likely makes their analysis more complicated. Additionally, there is technical overhead in their exposition due to the fact that they have to set parameters in their algorithms to guarantee the $(1+\epsilon)$ multiplicative factor in their approximation, whereas our analysis uses an $O(1)$ multiplicative factor in the approximation guarantee.

[4] In the beeping model nodes communicate with beeps. In the specific version of the beeping model in [27], 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.

[5] $H_{\mathbf{s}}$ is the $\mathbf{s}$*th Harmonic number* and it is known that $H_{\mathbf{s}} = \ln \mathbf{s} + O(1)$.

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, for $j = 1, 2, \ldots$, 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 $P_j + 1$ through 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* in batch j 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 independently for each iteration in batch j, each time with probability $2c \log N \cdot 2^i/\mathbf{s}$, where c is a specific constant fixed throughout the paper. For concreteness, we set $c = 200/\epsilon^2$. This value is sufficient for all Chernoff bound applications below: the proofs of Lemma 1 (via Corollary 4.6 of [29]), Lemma 3 (via a direct calculation), and Lemma 5 (via Theorem 4.4 of [29], footnote 6). This provides independent estimates of the degree of each candidate set for each iteration in batch j.

Let $A = (S_A, X_A, E_A)$ be the bipartite graph with node set $S_A \cup X_A$, where S_A is the set of selected candidate sets in batch j and X_A is the set of sampled elements in batch j. The edge set E_A of the bipartite graph consists of edges $\{S, e\}$, where $S \in S_A$, $e \in X_A$ 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 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 $\deg_{i,P_j+1}(S) \geq \mathbf{s}/2^i$ that have not been selected in prior batches become candidates for batch j;
4 Every batch j 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 $P_j + 1 \leq k \leq P_{j+1}$ and becomes selected if $c_k(S) = 1$ for some $P_j + 1 \leq k \leq P_{j+1}$; let S_A denote the set of selected sets;
5 Every element not covered in previous batches samples itself independently with probability $2c \log N \cdot 2^i/\mathbf{s}$ for each iteration k for $P_j + 1 \leq k \leq P_{j+1}$; let $X_{A,k}$ denote the set of sampled elements for iteration k, let $X_A = \cup_{k=P_j+1}^{P_{j+1}} X_{A,k}$ denote all sampled elements, and let $A = (S_A, X_A, E_A)$ denote the bipartite subgraph induced by $S_A \cup X_A$;
6 Every candidate set S selected in iteration k, $P_j + 1 \leq k \leq P_{j+1}$, computes $C(S) = X_{A,k} \cap S \cap U_{P_j+1}$, the elements in S that were uncovered at the start of the batch and sampled for iteration k;
7 For every selected candidate set S, gather the ball $B_A(S, \delta\sqrt{\log N}/2)$ at node S;
8 Using $B_A(S, \delta\sqrt{\log N}/2)$ every candidate set S selected in iteration k, $P_j + 1 \leq k \leq P_{j+1}$, computes $C_u(S) = X_{A,k} \cap S \cap U_k \subseteq C(S)$, the subset of $C(S)$ that remains uncovered at the start of iteration k;
9 Every selected set $S \in S_A$ for which

$$\widehat{\deg}_{i,k}(S) \stackrel{\text{def}}{=} \frac{|C_u(S)|}{|C(S)|} \cdot \deg_{i,P_j+1}(S) \geq \frac{\mathbf{s}}{2^i}$$

joins the set cover.

As informally described in the introduction, for a set S that is selected in iteration k, $P_j + 1 \leq k \leq P_{j+1}$ (i.e., in batch j), we use a sample of elements and define an estimator of $\deg_{i,k}(S)$, denoted $\widehat{\deg}_{i,k}(S)$, based on this sample. We then use this estimator in the comparison $\widehat{\deg}_{i,k}(S) \geq \mathbf{s}/2^i$ to determine if the selected set should be added to the set cover. To make this precise, we introduce the following notation. Let U_k denote the set of elements that are uncovered at the start of iteration k, so that $U_{k'} \subseteq U_k$ for $k' \geq k$ and $U_k \subseteq U_{P_j+1}$ for all iterations k in batch j. For each iteration k in batch j, let $X_{A,k}$ denote the set of elements sampled for use in iteration k (recall that each element in U_{P_j+1} is sampled independently with probability $2c \log N \cdot 2^i/\mathbf{s}$). For a candidate set S selected in iteration k, define:

- $C(S) = X_{A,k} \cap S \cap U_{P_j+1}$, the elements in S that were uncovered *at the start of the batch* and sampled for iteration k; and
- $C_u(S) = X_{A,k} \cap S \cap U_k \subseteq C(S)$, the subset of $C(S)$ that remains uncovered at the start of iteration k (here, the subscript u stands for "uncovered").

As our estimator we use

$$\widehat{\deg}_{i,k}(S) \stackrel{\text{def}}{=} \frac{|C_u(S)|}{|C(S)|} \cdot \deg_{i,P_j+1}(S).$$

The following lemma shows that this estimator, used in Line 9 of Algorithm 2 for $\deg_{i,k}(S)$ 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 small $(1 + O(\epsilon))$ multiplicative factor.

Lemma 1. *Suppose that S is a candidate selected in iteration k, $P_j + 1 \le k \le P_{j+1}$. Let $C(S) = X_{A,k} \cap S \cap U_{P_j+1}$ and $C_u(S) = X_{A,k} \cap S \cap U_k \subseteq C(S)$ be as defined above, and let $\widehat{\deg}_{i,k}(S) = (|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S)$ be the estimator defined above. Then, for any constant $\epsilon > 0$, whp,*

$$\widehat{\deg}_{i,k}(S) \ge \frac{\mathbf{s}}{2^i} \implies \deg_{i,k}(S) \ge \frac{\mathbf{s}}{2^i(1+\epsilon)},$$
$$\widehat{\deg}_{i,k}(S) < \frac{\mathbf{s}}{2^i} \implies \deg_{i,k}(S) < \frac{\mathbf{s}}{2^i(1-\epsilon)}.$$

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 a candidate before the start of this batch, $|U| \ge \mathbf{s}/2^i$ and therefore $\mathbb{E}[|C(S)|] = (2c \log N \cdot 2^i/\mathbf{s}) \cdot |U| \ge 2c \log N$ and by the standard multiplicative Chernoff bound[6]

$$\text{Prob}\left[||C(S)| - \mathbb{E}[|C(S)|]| \ge \frac{\epsilon}{4} \cdot \mathbb{E}[|C(S)|]\right] \le \frac{1}{N^3}. \tag{1}$$

(with $c = 200/\epsilon^2$, the actual Chernoff bound is even tighter than $1/N^3$; we state the looser $1/N^3$ for clarity). Furthermore, $\mathbb{E}[|C_u(S)|] = (2c \log N \cdot 2^i/\mathbf{s}) \cdot \deg_{i,k}(S)$. We partition the rest of the proof into two cases depending on how large $\deg_{i,k}(S)$ is relative to $\mathbf{s}/2^{i+1}$.

Case (1): $\deg_{i,k}(S) \ge \mathbf{s}/2^{i+1}$. In this case, $\mathbb{E}[|C_u(S)|] \ge c \log N$ and by Chernoff bounds (Corollary 4.6 of [29]),

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

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), and a union bound, $\text{Prob}[\mathcal{E}_1 \wedge \mathcal{E}_2] \ge 1 - 2/N^3 \ge 1 - 1/N^2$ for $N \ge 2$. Therefore, with probability at least $1 - 1/N^2$

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

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

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

Case (2): $\deg_{i,k}(S) < \mathbf{s}/2^{i+1}$. In this case, $\mathbb{E}[|C_u(S)|] < c \log N$. Therefore, by Chernoff bounds $\text{Prob}[|C_u(S)| \ge (3c/2) \log N] \le 1/N^3$. The bound in (1) also implies that $\text{Prob}[|C(S)| \le ((3c/2) \log N \cdot 2^i/s) \cdot |U|] \le 1/N^3$. Therefore, using a similar reasoning as in Case 1, with probability at least $1 - 1/N^2$, both 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 $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \ge \mathbf{s}/2^i$, then whp Case (1) holds. This is because if Case (2) were to hold, then whp $(|C_u(S)|/|C(S)|) \cdot |U| < \mathbf{s}/2^i$. Thus, whp $\deg_{i,k}(S) \ge \mathbf{s}/2^{i+1}$ and therefore $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \le (1+\epsilon) \cdot \deg_{i,k}(S)$. Hence, $\deg_{i,k}(S) \ge \mathbf{s}/2^i(1+\epsilon)$.
- If $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) < \mathbf{s}/2^i$ then either Case (1) or (2) above may hold. If Case (1) holds and $\deg_{i,k}(S) \ge \mathbf{s}/2^{i+1}$ then we have $(|C_u(S)|/|C(S)|) \cdot \deg_{i,P_j+1}(S) \ge (1-\epsilon) \cdot \deg_{i,k}(S)$ and therefore $\deg_{i,k}(S) \le \mathbf{s}/2^i(1-\epsilon)$. If Case (2) holds, then $\deg_{i,k}(S) < \mathbf{s}/2^{i+1} < \mathbf{s}/2^i(1-\epsilon)$.

□

The lemma above shows that the proposed estimator for $\deg_{i,k}(S)$ 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 $B_A(S, \delta\sqrt{\log N}/2)$.

[6] Throughout this paper, we use the following Chernoff bounds for sums of independent Bernoulli random variables, as stated in Theorem 4.4 and Corollary 4.6 of [29]. Let $Y = \sum_{i=1}^n X_i$ where each $X_i \in \{0,1\}$ is independent with $\text{Prob}[X_i = 1] = p_i$, and let $\mu = \mathbb{E}[Y]$. Then: (i) (Two-sided, Corollary 4.6) For $0 < \delta < 1$, $\text{Prob}[|Y - \mu| \ge \delta\mu] \le 2\exp(-\delta^2\mu/3)$. (ii) (Upper tail, Theorem 4.4) For $0 < \delta \le 1$, $\text{Prob}[Y \ge (1+\delta)\mu] \le \exp(-\delta^2\mu/3)$. (iii) (Large deviation, Theorem 4.4) For any $R \ge 6\mu$, $\text{Prob}[Y \ge R] \le 2^{-R}$.

Lemma 2. *Suppose that a candidate node S is selected in iteration ℓ, $P_j + 1 \le \ell \le P_{j+1}$. Further suppose that the machine hosting set S knows the subgraph $B_A(S, \delta\sqrt{\log N}/2)$. Then, the machine can determine whp via local computation the set $C_u(S) = X_{A,\ell} \cap S \cap U_\ell$ of sampled elements in S that remain uncovered at the start of iteration ℓ.*

Proof. Let M_S denote the machine hosting set S. We will show by induction that for any h, $1 \le h \le \ell$, 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 $k \le h - 1$. 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 $k \le h - 1$, 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 $(|C_u(Y)|/|C(Y)|) \cdot \deg_{i,k}(Y) \ge \mathbf{s}/2^i$ 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 $k' \le h$. All neighbors of x that are selected candidates are in $B_A(S, 2(\ell - h))$. Therefore, for each $k \le h - 1$, 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 $k' \le h$, 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 5) to show that every $\delta\sqrt{\log N}/2$-neighborhood in the active graph has size $O(N^{\delta/2})$.

Lemma 3. *At the start of batch j in stage i, whp every not-yet-covered element x has at most $3\mathbf{t} \cdot \ln N/2^{P_j}$ candidate neighbors.*

Proof. For $j = 1$ (the first batch of a stage), we have $P_1 = 0$ and the bound becomes $3\mathbf{t} \cdot \ln N$, which holds trivially since every element belongs to at most $\mathbf{t}$ sets (and $3 \ln N \ge 1$ for the regimes of interest).

For $j \ge 2$: consider an element $x \in X$ that is not yet covered at the start of batch j. Suppose for contradiction that x has more than $3\mathbf{t} \cdot \ln N/2^{P_j}$ neighbors that are candidates for batch j. Every such candidate S satisfies $\deg_{i,P_j+1}(S) \ge \mathbf{s}/2^i$. Since uncovered degrees can only decrease over iterations, $\deg_{i,P_{j-1}+1}(S) \ge \deg_{i,P_j+1}(S) \ge \mathbf{s}/2^i$, so S was also a candidate at the start of batch $j - 1$.

Note that for set S to be a candidate in batch j it is also required that S not be selected in any prior batches. Now, in batch $j - 1$, each such candidate S is selected with probability at least $2^{P_j}/\mathbf{t}$ (from the coin flip $c_{P_j}(S)$ in the last iteration of batch $j - 1$ alone). Since x has more than $3\mathbf{t} \cdot \ln N/2^{P_j}$ such candidates, the probability that none is selected in batch $j - 1$ is at most

$$\left(1 - \frac{2^{P_j}}{\mathbf{t}}\right)^{3\mathbf{t} \ln N/2^{P_j}} \le \exp(-3 \ln N) = \frac{1}{N^3}.$$

By the union bound over all elements, with probability at least $1 - 1/N^2$, every element $x \in X$ not-yet-covered at the start of batch j has at most $3\mathbf{t} \cdot \ln N/2^{P_j}$ candidate neighbors. □

The next lemma states a basic invariant maintained by the algorithm across stages, which we will use in the proof of Lemma 5.

Lemma 4. *At the beginning of stage i, every set not currently in the set cover contains at most $\mathbf{s}/2^{i-1}$ uncovered elements.*

This follows because any set with more uncovered 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.

Lemma 5. *For any node x in the active graph A of batch j in stage i, the number of edges in the subgraph $B_A(x, \delta\sqrt{\log N}/2)$ is bounded by $O(N^{\delta/2})$ whp.*

Proof. By Lemma 3 each not-yet-covered element has at most $3\mathbf{t} \ln N/2^{P_j}$ neighboring candidates with high probability. Furthermore, each candidate set is selected with probability at most $\sum_{k=P_j+1}^{P_{j+1}} 2^k/\mathbf{t} \le 2 \cdot 2^{P_{j+1}}/\mathbf{t}$, by a union bound over the $\delta\sqrt{\log N}/4$ biased coins. Thus, in expectation, a not-yet-covered element has at most $6 \ln N \cdot 2^{\delta\sqrt{\log N}/4}$ selected candidate neighbors. By Chernoff bounds, (Theorem 4.4(3) of [29], applied with $R = (6 \log N) \cdot \mu$ where $\mu = 6 \ln N \cdot 2^{\delta\sqrt{\log N}/4}$ so $R \ge 6\mu$), a not-yet-covered element has more than $(6 \log N) \cdot 6 \ln N \cdot 2^{\delta\sqrt{\log N}/4}$ selected candidate neighbors with probability at most $2^{-6 \log N} = 1/N^6$. Thus, whp, every not-yet-covered element has at most $(6 \log N) \cdot 6 \ln N \cdot 2^{\delta\sqrt{\log N}/4} = O(\log^2 N \cdot 2^{\delta\sqrt{\log N}/4})$ selected candidate neighbors.

On the other hand, every not-yet-covered element e is sampled independently with probability $2c \log N \cdot 2^i/\mathbf{s}$ in each iteration in Stage i (with $c = 200/\epsilon^2$).

By Lemma 4, the expected number of sampled not-yet-covered elements in any candidate set in this batch is $4c \log N$. By Chernoff bounds (Theorem 4.4(3) of [29], applied with $R = 24c \log N = 6\mu$), the probability that a set not in the current set cover contains more than $24c \log N$ sampled uncovered elements is at most $2^{-24c \log N} = 1/N^{24c}$, which is whp. 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 $24c \log^{1.5} N = O(\log^{1.5} N)$ 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 5 ensures that the local neighborhoods we need to gather in the active graph are small enough to fit in a single machine's memory. The next lemma shows that these neighborhoods can be efficiently gathered using graph exponentiation: by repeatedly doubling the known neighborhood radius, we can collect $\delta\sqrt{\log N}/2$-neighborhoods in just $O(\log\log N)$ rounds.

Lemma 6. *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 5, 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 an $O(N^{\delta})$ 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.

Since the goal is to gather balls of radius $\delta\sqrt{\log N}/2$, and each communication increases the radius by a factor of 2, this process completes in $\log(\delta\sqrt{\log N}/2) = O(\log\log N)$ rounds. □

With all the pieces in place — Lemma 1 provides a good estimator for the uncovered degree using sampled elements, Lemma 5 bounds the size of local neighborhoods in the active graph, and Lemma 6 shows how to gather these neighborhoods efficiently — we can now state and prove our main result for the low-memory MPC model. The key idea is that each batch of $\delta\sqrt{\log N}/4$ iterations can be simulated in $O(\log\log N)$ rounds, leading to a round complexity that improves upon the $O(\log^2 N)$-round simulation of the CONGEST algorithm.

Theorem 7. *There exists 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 6. Lines 7-9 only require local computation. Hence, Algorithm 2, 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 [23]. Algorithm 2 simulates Algorithm 1 faithfully except 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 k in Stage i if $\deg_{i,k}(S) \ge \mathbf{s}/2^i$. But, as per Lemma 1, S may be classified as a candidate even if $\deg_{i,k}(S)$ 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 classified as a non-candidate even if $\deg_{i,k}(S)$ 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 $(1 + O(\epsilon))$ factor.

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. Near-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 7.

Theorem 8. *There exists 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 5 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 $\deg(S) \geq \mathbf{s}^{\delta/2}$. The first $(1-\delta/2)\log \mathbf{s}$ 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 at 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 $\log \mathbf{t} - 3\log\log N$. Each candidate set 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 $A = (S_A, X_A, E_A)$ be the active graph for stage i and batch j which is the bipartite graph with node set $S_A \cup X_A$, where 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 $S \in S_A, e \in X_A$, and $e \in S$.

Lemma 9. *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 set 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 $\mathbb{E}[|S_A|] \leq 2N/\log^3 N$. Since the candidates select themselves independently, we can use Chernoff bound to say that whp, $|S_A| \leq 2cN/\log^3 N$.

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 has at most $O(\log N\log \mathbf{t}) = O(\log^2 N)$ sampled neighbors. Therefore, $|E_A| \leq |S_A|\cdot O(\log^2 N) \leq O(N/\log N)$ whp, and the lemma holds. □

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

Theorem 10. *There exists 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 7, the use of the estimators of Lemma 1 only worsens the expected $O(\log \mathbf{s})$ approximation factor of the base Algorithm 1 by a small $(1+O(\epsilon))$ multiplicative factor, where ϵ can be made 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 $O(\log\log N)$ iterations 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. We note that the same results were independently obtained at the same time by Dhulipala, Dinitz, Łącki, and Mitrović [28]. 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 [16], combined with element sampling to have an up-to-date estimate of degrees. Several natural questions arise from this work. First, is it possible to design even faster MPC algorithms for SetCover? For example, is an $\tilde{O}(\log N)$-round MPC algorithm possible in the sublinear-memory MPC model? What about an $o(\log N)$-round MPC algorithm in the linear-memory MPC model? Some impressive progress has been made on derandomizing MPC algorithms for MIS [30] and coloring [17]. 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 also be a natural follow-up to this work.

CRediT authorship contribution statement

Hongyan Ji: Writing – review & editing, Writing – original draft; **Shreyas Pai:** Writing – review & editing, Writing – original draft; **Sriram V. Pemmaraju:** Writing – review & editing, Writing – original draft; **Joshua Sobel:** Writing – review & editing, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Hongyan Ji, Sriram V. Pemmaraju, and Joshua Sobel were partially supported during this project by National Science Foundation grants 1955939 and 2402835.

References

[1] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, Z. Svitkina, On distributing symmetric streaming computations, ACM Trans. Algorithms 6(4), 2010. https://doi.org/10.1145/1824777.1824786
[2] M.T. Goodrich, N. Sitchinava, Q. Zhang, Sorting, searching, and simulation in the mapreduce framework, in: Proceedings of the 22nd International Conference on Algorithms and Computation, ISAAC'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 374–383. https://doi.org/10.1007/978-3-642-25591-5_39
[3] H. Karloff, S. Suri, S. Vassilvitskii, 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, 2010, pp. 938–948.
[4] G. Yaroslavtsev, A. Vadapalli, Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p distances, in: J. Dy, A Krause, (Eds.), Proceedings of the 35th International Conference on Machine Learning, 80 of *Proceedings of Machine Learning Research*, PMLR, US, 2018, pp. 5600–5609. https://proceedings.mlr.press/v80/yaroslavtsev18a.html
[5] V. Chvatal, A greedy heuristic for the set-covering problem, Math. Oper. Res. 4 (3) (1979) 233–235. https://doi.org/10.1287/moor.4.3.233
[6] D.S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Syst. Sci. 9 (3) (1974) 256–278. https://doi.org/10.1016/S0022-0000(74)80044-9
[7] L. Lovász, On the ratio of optimal integral and fractional covers, Discrete Math. 13 (4) (1975) 383–390. https://doi.org/10.1016/0012-365X(75)90058-8
[8] I. Dinur, D. Steurer, Analytical approach to parallel repetition, in: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, STOC '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 624–633. https://doi.org/10.1145/2591796.2591884
[9] S. Rajagopalan, V.V. Vazirani, Primal-dual RNC approximation algorithms for set cover and covering integer programs, SIAM J. Comput. 28 (2) (1999) 525–540. https://doi.org/10.1137/S0097539793260763
[10] B. Berger, J. Rompel, P.W. Shor, Efficient NC algorithms for set cover with applications to learning and geometry, in: Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, Academic Press, Inc., USA, 1994, pp. 454–477.
[11] L. Jia, R. Rajaraman, T. Suel, 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
[12] K. Censor-Hillel, M. Dory, Distributed spanner approximation, SIAM J. Comput. 50 (3) (2021) 1103–1147. https://doi.org/10.1137/20M1312630
[13] D. Peleg, Distributed Computing: A Locality-Sensitive Approach, Society for Industrial and Applied Mathematics, 3600 University City Science Center Philadelphia, PA, United States, 2000.
[14] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrović, R. Rubinfeld, Improved massively parallel computation algorithms for MIS, matching, and vertex cover, in: Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), PODC '18, ACM, New York, NY, USA, 2018, pp. 129–138.
[15] M. Ghaffari, J. Uitto, Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation, in: T.M. Chan, (Eds.), Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, SIAM, Philadelphia, PA 19104, 2019, pp. 1636–1653. https://doi.org/10.1137/1.9781611975482.99
[16] K. Kothapalli, S. Pai, S.V. Pemmaraju, Sample-and-gather: fast ruling set algorithms in the low-memory MPC model, in: N. Saxena, S. Simon (Eds.), 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, 182 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Oktavie-Allee, 66687 Wadern, Germany, 2020, pp. 28:1–28:18. https://doi.org/10.4230/LIPIcs.FSTTCS.2020.28
[17] A. Czumaj, P. Davies, M. Parter, Simple, deterministic, constant-round coloring in congested clique and MPC, SIAM J. Comput. 50 (5) (2021) 1603–1626. https://doi.org/10.1137/20M1366502
[18] A. Czumaj, J. Ł ącki, A. Mądry, S. Mitrović, K. Onak, P. Sankowski, Round compression for parallel matching algorithms, in: Proceedings of the Annual ACM Symposium on Theory of Computing, 14, 2018, pp. 471–484.
[19] M. Luby, A simple parallel algorithm for the maximal independent set problem, SIAM J. Comput. 15 (4) (1986) 1036–1053. https://doi.org/10.1137/0215074

[20] N. Alon, L. Babai, A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, J. Algorithms7 (4), (1986) 567–583. https://doi.org/10.1016/0196-6774(86)90019-2
[21] N.J.A. Harvey, C. Liaw, P. Liu, Greedy and local ratio algorithms in the MapReduce model, in: C. Scheideler, J.T. Fineman, (Eds.), Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, ACM, Vienna, Austria, July 16-18, 2018, 2018, pp. 43–52. https://doi.org/10.1145/3210377.3210386
[22] S. Behnezhad, S. Brandt, M. Derakhshan, M. Fischer, M. Hajiaghayi, R.M. Karp, J. Uitto, Massively parallel computation of matching and MIS in sparse graphs, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 481–490. https://doi.org/10.1145/3293611.3331609
[23] C. Grunau, S. Mitrović, R. Rubinfeld, A. Vakilian, Improved local computation algorithm for set cover via sparsification, ACM Trans. Algorithms 17 (3) (2021). https://doi.org/10.1145/3470770
[24] M. Ghaffari, Distributed MIS via all-to-all communication, in: Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 141–149. https://doi.org/10.1145/3087801.3087830
[25] J.W. Hegeman, S.V. Pemmaraju, V. Sardeshmukh, Near-constant-time distributed algorithms on a congested clique, in: F. Kuhn, (Eds.), Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings, 8784 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2014, pp. 514–530. https://doi.org/10.1007/978-3-662-45174-8_35
[26] M. Parter, E. Yogev, Congested clique algorithms for graph spanners, in: U. Schmid, J. Widder (Eds.), 32nd International Symposium on Distributed Computing, DISC 2018, 121 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, New Orleans, LA, USA, 2018, pp. 40:1–40:18. https://doi.org/10.4230/LIPIcs.DISC.2018.40
[27] T. Götte, C. Kolb, J. Scheideler, J. Werthmann, 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, Springer-Verlag, Berlin, Heidelberg, 2021, pp. 94–110. https://doi.org/10.1007/978-3-030-89240-1_7
[28] L. Dhulipala, M. Dinitz, J. Łącki, S. Mitrović, Parallel set cover and hypergraph matching via uniform random sampling, in: D. Alistarh, (Eds.), 38th International Symposium on Distributed Computing (DISC 2024), 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2024, pp. 19:1–19:23. https://doi.org/10.4230/LIPIcs.DISC.2024.19
[29] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis, Cambridge University Press, 2017.
[30] A. Czumaj, P. Davies, M. Parter, Graph sparsification for derandomizing massively parallel computation with low space, in: Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Association for Computing Machinery, New York, NY, USA, 2020, pp. 175–185. https://doi.org/10.1145/3350755.3400282