Improved Local Computation Algorithm for Set Cover via Sparsification

Christoph Grunau* Slobodan Mitrović[†] Ronitt Rubinfeld[‡] Ali Vakilian[§]

Abstract

We design a Local Computation Algorithm (LCA) for the set cover problem. Given a set system where each set has size at most s and each element is contained in at most t sets, the algorithm reports whether a given set is in some fixed set cover whose expected size is $O(\log s)$ times the minimum fractional set cover value. Our algorithm requires $s^{O(\log s)} t^{O(\log s \cdot (\log \log s + \log \log t))}$ queries. This result improves upon the application of the reduction of [Parnas and Ron, TCS'07] on the result of [Kuhn et al., SODA'06], which leads to a query complexity of $(st)^{O(\log s \cdot \log t)}$.

To obtain this result, we design a parallel set cover algorithm that admits an efficient simulation in the LCA model by using a *sparsification* technique introduced in [Ghaffari and Uitto, SODA'19] for the maximal independent set problem. The parallel algorithm adds a random subset of the sets to the solution in a style similar to the PRAM algorithm of [Berger et al., FOCS'89]. However, our algorithm differs in the way that it never *revokes* its decisions, which results in a fewer number of adaptive rounds. This requires a novel approximation analysis which might be of independent interest.

1 Introduction

The set cover problem is one of the classical problems in optimization and computer science. In this problem, we are given a universe of n elements $\mathcal{E}$ and a family of m sets $\mathcal{S} \subseteq 2^{\mathcal{E}}$, and our goal is to find a minimum size *set cover* of $\mathcal{E}$; i.e., a collection of sets in $\mathcal{S}$ whose union is equal to $\mathcal{E}$. The set cover problem is a well-studied problem with applications in many areas such as machine learning, data mining and operation research [19, 35, 24, 5].

A simple greedy algorithm for this problem, that repeatedly adds a set containing the largest number of yet uncovered elements to the cover, guarantees a $O(\log n)$-approximation [23, 28]. Unless $P = NP$, this approximation guarantee is within a constant factor compared to the approximation guarantee of any polynomial-time algorithm [33, 16, 1, 29, 14]. Unfortunately, this standard greedy algorithm does not scale very well for massive data sets (e.g., see Cormode et al. [12] for an experimental evaluation of the greedy algorithm on large data sets). This difficulty has led to considerable interest in designing set cover algorithms for computational models tailored to process massive amounts of data such as parallel computation [7, 8, 9], streaming [35, 27, 15, 13, 10, 20, 4, 3, 6, 21], sublinear time/query algorithms [18, 25, 22] and local computation algorithms [30, 38].

In many scenarios we are interested in designing *extremely fast* algorithms for learning only a minuscule portion of a solution, rather than computing and storing the entire one. This has led to the model of *local computation algorithms (LCAs)* introduced by Rubinfeld et al. [34] and Alon et al. [2]. A LCA for the set cover problem provides oracle access to some fixed set cover $\mathcal{C}$.[1] That is, given some arbitrary set S, the LCA needs to report whether S is part of the set cover $\mathcal{C}$ by only having primitive query access to the input set system $(\mathcal{S}, \mathcal{E})$. The only shared state across different oracle calls is a tape of random bits. The performance is measured by the approximation guarantee and the query complexity of the LCA. In the LCA model, we further assume that each set in $\mathcal{S}$ has size at most s and each element in $\mathcal{E}$ is contained in at most t sets. Section 1.1 contains additional information about the underlying computational model.

Simulating the Greedy Algorithm There are two main approaches for designing LCAs for set cover. The first one is based on simulating the greedy algorithm using a randomized ranking technique due to Nguyen and Onak [30] which was later improved by Yoshida et al. [38]. While the goal of [30, 38] is the de-

*ETH Zurich, cgrunau@student.ethz.ch

[†] CSAIL, MIT, slobo@mit.edu

[‡] CSAIL, MIT and TAU, ronitt@csail.mit.edu

[§] University of Wisconsin - Madison, vakilian@wisc.edu; This work was done when the author was a graduate student at MIT.

[1] Note that for any given input, there may be many set covers. $\mathcal{C}$ is a unique set cover that depends only on the input and the random bits used by the algorithm.

sign of constant-time algorithms for approximating the size of an optimal set cover solution on bounded degree instances, one can turn their algorithms into $O(\log s)$-approximation LCAs with a respective query complexity of $2^{O((st)^4\cdot 2^s)}$ [30] and $(st)^{O(s)}$ [38].

LCAs via Distributed Algorithms The other approach is via a generic reduction from distributed algorithms to LCAs by Parnas and Ron [32]. Kuhn et al. [26] designed a distributed algorithm that computes a $O(1)$-approximate fractional solution in $O(\log s\cdot\log t)$ many rounds. In each round, each set and each element only requires local information, i.e., a set uses only the information of the elements it contains and an element uses only the information of the sets that it is contained in. By the reduction of [32], the distributed algorithm of [26] can be transformed into an LCA with a query complexity of $(st)^{O(\log s\cdot\log t)}$. The LCA only outputs the fractional contribution of S in the corresponding fractional set cover solution for some given set S. However, via the standard randomized rounding technique and a slight increase in the query complexity by a factor of $s\cdot t$, the LCA can be turned into an LCA that outputs an $O(\log s)$-approximate *integral* solution for the set cover problem. There is no known distributed algorithm for set cover that uses fewer than $O(\log s\cdot\log t)$ iterations of computation, and hence applying known reductions to any such algorithm would not improve the query complexity implied by [26] and [32]. It is thus natural to wonder:

> *Is it possible to obtain LCAs for the set cover problem with a smaller query complexity compared to the query complexity obtained via standard reductions to distributed algorithms?*

In this paper, we answer this question affirmatively by presenting the following result.

THEOREM 1.1. *There exists an LCA with a query complexity of $s^{O(\log s)}t^{O(\log s\cdot(\log\log s+\log\log t))}$ that produces a set cover with an expected size of $O(\log s)\cdot\mathrm{OPT}$, where* OPT *denotes the value of an optimal fractional set cover solution.*

This result is proved in Section 6. In Section 1.2 we give an overview of our approach.

1.1 The Local Computation Model

We consider a graph representation of the input set system $(\mathcal{S},\mathcal{E})$ and adopt the definition of LCAs by Rubinfeld et al. [34]. In the design of LCAs, the query access model to the input instance plays an important role. Here we consider standard *neighbor* queries: "what is the i-th element in a given set S?" and "what is the i-th set containing a given element e?". In our analysis, we assume that a query to a set (resp., element) returns all elements contained in (resp., all sets containing) the required set (resp., element). This will increase the query complexity only by a factor of $O(s+t)$.

DEFINITION 1.1. (LCA FOR SET COVER) *An LCA for the set cover problem is a (randomized) algorithm that has access to neighbor queries, a tape of random bits and local working memory. Given some set S from the input set system $(\mathcal{S},\mathcal{E})$, the LCA algorithm returns whether S is part of the set cover by making queries to the input. The answer must only depend on the given set S, the input set system $(\mathcal{S},\mathcal{E})$ and the random tape. Moreover, for a fixed tape of random bits, the answer given by the LCA to all sets, must be consistent with one particular valid set cover.*

We remark that by running the described LCA on at most t sets, we can determine which set is covering a given element in the approximate set cover constructed (hypothetically) by the LCA.

The main complexity measures of the LCA for set cover are the expected size of the solution, as well as the query complexity of the LCA which is defined as the maximum number of queries that the algorithm makes to return an answer for any arbitrary element e in the input universe. Note that all the algorithms in this paper are randomized and, for any input set system, provide guarantees on the expected size of the solution, over the random tape. For simplicity, we describe all our randomized algorithms using full independence; however, they can be implemented using a seed with a polylogarithmic number of random bits by the techniques and concentration bounds in [36, 37].

1.2 Roadmap and Our Technical Contributions

The base algorithm (Section 3). The starting point of our approach is a parallel algorithm that constructs a set cover in $\log s\cdot\log t$ rounds. This algorithm, that we present in Section 3, consists of $\log s$ *stages* and each stage consists of $\log t$ *iterations*. The stages are used to process sets that have a large number of uncovered elements. That is, in stage i we consider all the sets that have at least $s/2^i$ uncovered elements. We call such sets *large*.

After the execution of stage i, a large set is either added to the set cover constructed so far, or the number of uncovered elements it contains dropped below $s/2^i$ during the execution of stage i. Iterations are used to progressively add large sets to the cover, while attempting to assure that an element is not covered by too many large sets. (We make this statement formal in Lemma 3.3.) More precisely, in the k-th iteration, each currently large set is added with probability $2^k/t$.

This step is applied for all the large sets simultaneously. Again, adding large sets with a probability of $2^k/t$ is supposed to ensure that an element contained in roughly $t/2^k$ large sets will be covered $\Theta(1)$ times at the moment it gets covered for the first time. We show that having this kind of guarantee suffices to obtain an expected $O(\log s)$-approximate minimum set cover.

The base algorithm has a parallel depth of $O(\log s \cdot \log t)$ and directly simulating it in the LCA model would result in a query complexity of $(st)^{O(\log s \cdot \log t)}$. In spirit, our parallel algorithm is similar to the algorithm developed in [7], but there is also a crucial difference. After randomly picking a family of sets in each round, the algorithm in [7] verifies whether the number of newly covered elements is large enough in comparison to the number of chosen sets. If yes, then they add the family of sets to the set cover. If not, they repeat the selection process. A random family of sets is good with a constant probability, yielding an O(log n) bound on the number of times that the algorithm needs to repeat a round. Therefore, the resulting algorithm has a larger parallel depth compared to our algorithm.

Estimating set sizes (Section 4). In Section 4 we design and analyze Algorithm 2. Compared to Algorithm 1, Algorithm 2 only estimates the number of remaining free elements of a set. This is a crucial step towards getting query-efficient LCAs. However, for reasons explained in Section 4, Algorithm 2 does not admit an efficient LCA simulation. The reason for including Algorithm 2 is twofold. First, all the later algorithms that admit an efficient LCA simulation, are basically identical to Algorithm 2, except that they ensure that certain conditions hold. Second, the approximation guarantee of Algorithm 2 can be established in more or less the same way as the approximation guarantee for Algorithm 1. In contrast, establishing the approximation guarantee for Algorithm 3 and Algorithm 4 is much harder. Therefore, it is easier to establish the approximation guarantee of Algorithm 3 and Algorithm 4 by relating them to Algorithm 2.

Sparsification of the element-neighborhoods (Sections 5 and 6). Estimating the number of uncovered elements within a given set, instead of counting them exactly, reduces the number of queries that a set has to perform. However, this optimization in terms of set-size estimates does not affect the number of direct set-queries that an element has to perform. To achieve our advertised query complexity, we also reduce the number of queries directly performed by an element. Our high-level approach for reducing the number of queries for elements follows the lines of the work [17, 11] (and in Section 5 also the work [31]). [17] design an LCA for maximal independent set (MIS) and maximal matching. These LCAs do not have a generic reduction to other models.

One of the main challenges in adapting the approach of [17] to the set cover problem is that, in the case of MIS or maximal matching it is needed to handle only one type of objects, i.e., vertices or edges. However, in the case of set cover, our algorithm handles sets and elements simultaneously.

2 Preliminaries

Notation. We will use $\mathcal{S}$ (respectively, $\mathcal{E}$) with added sub- and super-scripts to refer to subsets of $\mathcal{S}$ (respectively, $\mathcal{E}$). Given a set S at some step of an algorithm, we use $d(S)$ to denote the number of uncovered elements of S in the current set cover. When we use $\hat{d}(S)$, it refers to an estimated number of uncovered elements of S in the current set cover.

Uncovered elements are also referred to by *free.* Subscripts i, j, and k will have the following meaning: i refers to a stage, j refers to a phase, and k refers to an iteration.

Relevant Concentration Bounds. Throughout the paper, we will use the following well-known variants of Chernoff bound.

THEOREM 2.1. (CHERNOFF BOUND) *Let* $X_1, \ldots, X_k$ *be independent random variables taking values in* $[0, 1]$. *Let* $X \overset{\text{def}}{=} \sum_{i=1}^{k} X_i$ *and* $\mu \overset{\text{def}}{=} \mathbb{E}[X]$. *Then,*

(A) For any $\delta \in [0, 1]$ *it holds* $\Pr[X \le (1-\delta)\mu] \le \exp\left(-\delta^2\mu/2\right)$.

(B) For any $\delta \ge 1$ *it holds* $\Pr[X \ge (1+\delta)\mu] \le \exp(-\delta\mu/3)$.

3 The Base Algorithm

We now fully state and analyze the base algorithm briefly described in Section 1.2. This algorithm is presented as Algorithm 1. As a reminder, this algorithm constructs a set cover in $\log(s)$ many stages (Line 1 of Algorithm 1) and each stage consists of $\log(t)$ many iterations (Line 2 of Algorithm 1). Algorithm 1 maintains the invariant that no set contains more than $s/2^i$ uncovered elements at the end of stage i. In iteration k of stage i, each set containing at least $s/2^i$ uncovered elements is added to the set cover with a probability of $2^k/t$. This step is executed for all the sets simultaneously. In Section 3.1 we show that the algorithm indeed outputs a set cover, while in Section 3.2 we analyze the approximation guarantee. Our approximation analysis diverts from the prior work known to the authors and might be of independent interest to the reader.

Algorithm 1: The base algorithm for our LCAs which runs in $O(\log s \cdot \log t)$ iterations.

```
for stage $i = 1$ to $\log s$ do
    for iteration $k = 1$ to $\log t$ do
        for each set $S$ in parallel do
            if $d(S) \geq s/2^i$ then // $d(S)$ denotes the number of free (or yet uncovered) elements in $S$
                add $S$ to $\mathcal{S}_{\mathsf{cover}}$ w.p. $\frac{2^k}{t}$.
return $\mathcal{S}_{\mathsf{cover}}$
```

3.1 Correctness It is not hard to see that Algorithm 1 indeed constructs a set cover. This stems from the fact that in iteration $k = \log t$, each set having at least $s/2^i$ uncovered elements at that point is added to the cover. A more elaborate argument is provided in the proof of the following claim.

Lemma 3.1. *For $s \geq 2$ and $t \geq 2$, Algorithm 1 returns a valid set cover.*

Proof. It suffices to show that each element is covered by some set in the solution. Consider an arbitrary element e. If e was covered before the last iteration of the last stage, then we are fine. Otherwise, there is at least one set S which contains e and was not chosen before the last iteration of the last stage. As this set has at least one free element, it is added to the set cover with a probability of $2^{\log t}/t = 1$. Hence, e will be covered by S. □

3.2 Analysis of the Approximation Guarantee In this section we analyze the approximation guarantee of Algorithm 1. Our analysis shows that the algorithm matches, up to a constant factor and in expectation, the guarantee of the best possible polynomial-time algorithm unless P = NP.

Theorem 3.1. *Let $(\mathcal{S}, \mathcal{E})$ be some set cover instance and let $\mathcal{S}_{\mathsf{cover}}$ denote the solution returned by Algorithm 1. Furthermore, let* OPT *denote the value of an optimal fractional set cover solution for $(\mathcal{S}, \mathcal{E})$. Then, $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] = O(\log s) \cdot \mathrm{OPT}$.*

To prove this claim, we need the following result that we establish in the rest of this section.

Lemma 3.2. *Let $(\mathcal{S}, \mathcal{E})$ be a set cover instance with a maximal set size of s and let $\mathcal{S}_{\mathsf{cover},1}$ denote the number of sets that are added to $\mathcal{S}_{\mathsf{cover}}$ during the first stage of Algorithm 1. Then, $\mathbb{E}[|\mathcal{S}_{\mathsf{cover},1}|] = O(n/s)$.*

Proof of. Theorem 3.1. In Lemma 3.2, we show that Algorithm 1 adds at most $O(n/s)$ many sets, in expectation, to the set cover during the first stage. Here we show that this observation suffices to prove the theorem. To see why, let $(\mathcal{S}_i, \mathcal{E}_i)$ denote the set cover instance with $\mathcal{E}_i$ being the set of all uncovered elements prior to the i-th stage and $\mathcal{S}_i = \{S \cap \mathcal{E}_i | S \in \mathcal{S}\}$. Let $n_i = |\mathcal{E}_i|$ denote the number of uncovered elements prior to the i-th stage. Note that the maximal set size in S_i is at most $s_i := s/2^{i-1}$. Let OPT_i denote the value of an optimal fractional set cover for $(\mathcal{S}_i, \mathcal{E}_i)$.

$$n_i/s_i \leq \mathrm{OPT}_i \leq \mathrm{OPT} \tag{3.1}$$

Let $\mathcal{S}_{\mathsf{cover},i}$ denote the collection of sets that Algorithm 1 picks in stage i. Note that $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover},i}|\right]$ is equal to the expected number of sets that Algorithm 1 would pick in the first stage when given $(\mathcal{S}_i, \mathcal{E}_i)$ as an input. Thus, by Eq. (3.1), $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover},i}|\right] = O(n_i/s_i) = O(\mathrm{OPT})$ which implies that

$$\begin{aligned} \mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] = \sum_i \mathbb{E}\left[|\mathcal{S}_{\mathsf{cover},i}|\right] &\leq \log(s) \cdot O(\mathrm{OPT}) \\ &= O(\log s) \cdot \mathrm{OPT}. \end{aligned}$$

□

The following is the main technical lemma that we use to prove Lemma 3.2.

Lemma 3.3. *Let $e \in \mathcal{E}$ be an arbitrary element and X_e be a random variable which is equal to 0 if e does not get covered during the first stage and otherwise is equal to the number of sets that contain e and are added to $\mathcal{S}_{\mathsf{cover}}$ in the same iteration in which e is covered for the first time. Then, $\mathbb{E}[X_e] \leq 5$.*

Let's observe what happens with e during the first stage. In each iteration, a set S is *active* if it contains e and has more than $s/2$ free elements in the beginning of the iteration. Suppose that there are $N_1 \leq t$ active sets in the first iteration. Since each active set joins the set cover independently with probability $2/t$, the number of times that e is covered in the first iteration can be described by a random variable $Y_1 \sim \mathrm{Bin}(N_1, 2/t)$. If $Y_1 > 0$, then $X_e = Y_1$. Otherwise, e is still uncovered after the first iteration. In this case, there are N_2 active sets at the beginning of the second iteration. Note that as the number of free elements in some of the N_1 active sets of the first iteration might have dropped below $s/2$, $N_1 \geq N_2$. More generally, given that e was not covered prior to the k-th iteration, there are N_k active sets and the number of times that e is covered in this iteration can be described as a random variable $Y_k \sim \mathrm{Bin}(N_k, 2^k/t)$. If $Y_k > 0$, then $X_e = Y_k$, otherwise we proceed to the next iteration. Note that for all $k \geq 1$, $N_k \geq N_{k+1}$ and N_{k+1} are random variables. To analyze the random process it is however easier to think of $N_1, \cdots, N_{\log t}$ as being fixed in advance. To that end, we think about an equivalent random process. We first choose a random vector $b_S \in \{0,1\}^{\log(t)}$ for each set S such that $\Pr[(b_S)_k = 1] = 2^k/t$. Now, in the k-th iteration we add a set S if it is an active set in the k-th iteration and $(b_S)_k = 1$. It is easy to see that this process is equivalent. Additionally, it allows us to use the principle of deferred decision making by fixing all the random vectors for those sets which do not contain e in advance. After fixing those, it is easy to see that N_k is also fixed if e is a free element in the k-th iteration.

Definition 3.1. *Let $seq = (n_1, ..., n_{\log t})$ be an integer sequence such that $t \geq n_1 \geq n_2 \geq ... \geq n_{\log t} \geq 0$. Let $Y_1, ..., Y_{\log t}$ be random variables with $Y_k \sim \mathrm{Bin}(n_k, 2^k/t)$. Let $Y_{seq} = Y_{k^*}$ with k^* being the smallest index for which $Y_{k^*} > 0$ if such a k^* exists and 0 otherwise.*

We now prove the following result that we will use to upper-bound $\mathbb{E}[X_e]$.

Lemma 3.4. *$\mathbb{E}[X_e]$ can be upper-bounded as follows*

$$\mathbb{E}[X_e] \leq \max_{t \geq n_1 \geq \ldots \geq n_{\log t} \geq 0} \mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right].$$

Proof. Let $\mathcal{N}$ denote the collection of all possible values for the random variable $(N_1, \ldots, N_{\log t})$. The proof is given by the following sequence of inequalities:

$$\begin{aligned}
&\mathbb{E}[X_e] \\
&= \sum_{(n_1,\ldots,n_{\log t}) \in \mathcal{N}} \mathbb{E}[X_e | (N_1, \ldots, N_{\log t}) = (n_1, \ldots, n_{\log t})] \\
&\quad \cdot \Pr[(N_1, \ldots, N_{\log t}) = (n_1, \ldots, n_{\log t})] \\
&= \sum_{(n_1,\ldots,n_{\log t}) \in \mathcal{N}} \mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right] \\
&\quad \cdot \Pr[(N_1, \ldots, N_{\log t}) = (n_1, \ldots, n_{\log t})] \\
&\leq \max_{(n_1,\ldots,n_{\log t}) \in \mathcal{N}} \mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right] \\
&\leq \max_{t \geq n_1 \geq \ldots \geq n_{\log t} \geq 0} \mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right]
\end{aligned}$$

□

Following Lemma 3.4, it only remains to upper-bound $\mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right]$ for any arbitrary sequence $n_1, \cdots, n_{\log t}$ such that $t \geq n_1 \geq \ldots \geq n_{\log t} \geq 0$. We will do this in two steps.

First, we will find an upper bound for $\mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right]$ and define a function f such that $f((n_1, \ldots, n_{\log t}), t)$ is equal to this upper bound. Then, we will show by induction on the sequence length that for some restrictions on the input parameters the value of the function can be bounded by a constant. The restricted input parameters still capture the parameters produced by Algorithm 1 and what we require for the analysis. We upper-bound $\mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right]$ as follows

$$\begin{aligned}
\mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right] &= \mathbb{E}[Y_1] \\
&\quad + \sum_{k=2}^{\log t} \mathbb{E}[Y_k] \cdot \Pr[Y_1 = \ldots = Y_{k-1} = 0] \\
&= n_1 \cdot \frac{2}{t} + \sum_{k=2}^{\log t} n_k \cdot \frac{2^k}{t} \prod_{j=1}^{k-1} \left(1 - 2^j/t\right)^{n_j} \\
&\leq n_1 \cdot \frac{2}{t} + \sum_{k=2}^{\log t} n_k \cdot \frac{2^k}{t} \prod_{j=1}^{k-1} e^{-n_j \cdot \frac{2^j}{t}}.
\end{aligned}$$

Definition 3.2. *Let $f((x_1, \ldots, x_l), r) := x_1 \cdot \frac{2}{r} + \sum_{k=2}^{l} x_k \cdot \frac{2^k}{r} \cdot \prod_{j=1}^{k-1} e^{-x_j \cdot \frac{2^j}{r}}$. For $l \in \mathbb{N}$, let $P(l)$ be the property that for all $(x_1, \ldots, x_l) \in \mathbb{R}_{>0}$ and $r \in \mathbb{R}_{>0}$ such that $r \geq x_1 \geq x_2 \geq \ldots \geq x_l \geq 0$, we have $f((x_1, \ldots, x_l), r) \leq 5$.*

We have that $P(\log(t))$ being true directly implies $\mathbb{E}\left[Y_{(n_1,\ldots,n_{\log t})}\right] \leq 5$, which in turn implies Lemma 3.3.

Lemma 3.5. *Let $P(l)$ be as defined in Definition 3.2. Then, $P(l)$ holds for all $l \geq 1$.*

Proof. We prove this statement by induction on l. We first show $P(1)$. Let x_1 and $r \in \mathbb{R}_{>0}$ be arbitrary such that $r \geq x_1 \geq 0$. We have $f((x_1), r) = x_1 \cdot 2/r \leq 5$.

Now, let $l \in \mathbb{N}$ be arbitrary. We assume $P(l)$ to be true and need to show $P(l+1)$. Let $r \in \mathbb{R}_{>0}$ and $x_1, \ldots, x_{l+1}$ be arbitrary such that $r \geq x_1 \geq \ldots \geq x_{l+1} \geq 0$.

$$\begin{aligned}
&f((x_1, \ldots, x_{l+1}), r) \\
&= x_1 \cdot \frac{2}{r} + \sum_{k=2}^{l+1} x_k \cdot \frac{2^k}{r} \cdot \prod_{j=1}^{k-1} e^{-x_j \cdot \frac{2^j}{r}} \\
&= x_1 \cdot \frac{2}{r} + e^{-x_1 \cdot \frac{2}{r}} \cdot \left(x_2 \cdot \frac{2^2}{r} + \sum_{k=3}^{l+1} x_k \cdot \frac{2^k}{r} \cdot \prod_{j=2}^{k-1} e^{-x_j \cdot \frac{2^j}{r}} \right) \\
&= x_1 \cdot \frac{2}{r} \\
&\quad + e^{-x_1 \cdot \frac{2}{r}} \cdot (2x_2 \cdot \frac{2}{r} + \sum_{k=2}^{l} (2x_{k+1}) \cdot \frac{2^k}{r} \cdot \prod_{j=1}^{k-1} e^{-(2x_{j+1}) \cdot \frac{2^j}{r}}) \\
&= x_1 \cdot \frac{2}{r} + e^{-x_1 \cdot \frac{2}{r}} \cdot f((2x_2, 2x_3, \ldots, 2x_{l+1}), r)
\end{aligned}$$

We would like to use the induction hypothesis to bound $f((2x_2, 2x_3, \dots, 2x_{l+1}), r)$. However, it might be the case that $2x_2 > r$. Therefore we need to do a case distinction. First, assume that $x_1 \le r/2$ which implies that $2x_2 \le r$. Thus we can indeed use the induction hypothesis:

$$f((x_1, \dots, x_{l+1}), r) \le x_1 \cdot \frac{2}{r} + e^{-x_1 \cdot \frac{2}{r}} \cdot 5 \le 5$$

Next we consider the case where $2x_2 > r$.

$$\begin{aligned} & f((x_1, \dots, x_{l+1}), r) \\ & \le x_1 \cdot \frac{2}{r} + e^{-x_1 \cdot \frac{2}{r}} \cdot f((2x_2, 2x_3, \dots, 2x_{l+1}), r) \\ & \le x_1 \cdot \frac{2}{r} + e^{-x_1 \cdot \frac{2}{r}} \cdot 2 \cdot f((x_2, x_3, \dots, x_{l+1}), r) \\ & \le x_1 \cdot \frac{2}{r} + e^{-x_1 \cdot \frac{2}{r}} \cdot 10 \le 5 \quad \triangleright \text{since } 1 \le x_1 \cdot \frac{2}{r} \le 2 \end{aligned}$$

□

We are now ready to prove Lemma 3.2.

Proof of. Lemma 3.2. We have $(s/2) \cdot |\mathcal{S}_{\mathsf{cover},1}| \le \sum_{e \in \mathcal{E}} X_e$ as during the first stage, Algorithm 1 only adds sets which have at least $s/2$ free elements at the beginning of the iteration. As $\mathbb{E}\left[\sum_{e \in \mathcal{E}} X_e\right] \le 5 \cdot n$, we get $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover},1}|\right] = O(n/s)$ as desired. □

4 A Generic Algorithm

In this section we present Algorithm 2. The algorithm differs from Algorithm 1 in two ways. First, it only *estimates* the number of uncovered elements contained in a given set. Second, it fixes all the randomness right at the beginning of Algorithm 2. After Line 4, the algorithm just executes a deterministic process. Both of these modifications are crucial to get efficient LCAs and will be explained in more detail in Section 4.1.

Although Algorithm 2 estimates set sizes instead of computing them exactly (as Algorithm 1 does) it is, however, not clear how to simulate Algorithm 2 in the local computation model with significantly less than $(st)^{O(\log s \cdot \log t)}$ many queries. The main reason for this is the following. The efficiency of a LCA is measured by its worst-case query complexity. In our case, this means that in order for a (randomized) LCA to have a query complexity of $T(s,t)$, each set S needs to decide after at most $T(s,t)$ queries whether it is contained in the set cover or not with high probability (in n). As we aim for a query complexity independent of n, this essentially implies that for each set S, S needs to decide after at most $T(s,t)$ queries whether it is part of the set cover or not, even if all the random bits are chosen adversarially. Although the subsequent LCAs will essentially simulate Algorithm 2, crucially, they will bound the influence that "bad randomness" can have on the query complexity by, e.g., immediately adding sets to the set cover for which the number of sampled elements is too large. However, these tests make it hard to directly establish an approximation guarantee for those algorithms. To deal with that problem, we first establish the approximation guarantee for Algorithm 2 in Section 4.2. After that, we use the approximation guarantee of Algorithm 2 to establish the approximation guarantee of the subsequent algorithms by relating them to Algorithm 2.

Algorithm 2: A generic set cover algorithm

```
1  Let $(\mathcal{S}, \mathcal{E})$ be some set cover instance.
2  for each pair of $(i,k)$ where $i \in [\log s]$ and
     $k \in [\log t]$ do
3      $B_i(S) \leftarrow$ a subset of $S$ s.t. each element is
         included indep. w.p.
         $p_i = \min(1, \log^{10} s \cdot \log^{10} t \cdot \frac{2^i}{s})$.
4      $\mathcal{S}_{i,k} \leftarrow$ a subfamily of $\mathcal{S}$ s.t. each set is
         included indep. w.p. $2^k/t$.
5  for stage $i = 1$ to $\log s$ do
6      for iteration $k = 1$ to $\log t$ do
7          $\hat{d}_{B_i}(S) \overset{\text{def}}{=} \frac{|B_i(S) \cap \mathcal{E}_{i,k}|}{p_i}$. // $\mathcal{E}_{i,k}$ denotes the
             set of free elements at the beginning of the
             current iteration
8          for each $S \in \mathcal{S}_{i,k}$ with $\hat{d}_{B_i}(S) \ge \frac{s}{2^i}$ do
9              add $S$ to $\mathcal{S}_{\mathsf{cover}}$.
10 return $\mathcal{S}_{\mathsf{cover}}$
```

4.1 Details of Algorithm 2 In this section, we further elaborate on the design decisions made in Algorithm 2. For a given set S, Algorithm 2 estimates the number of free elements within stage i by counting the number of uncovered elements in $B_i(S)$. The set $B_i(S)$ is a random subset of S such that each element in S is contained in $B_i(S)$ with a probability of p_i. Note that if $p_i = 1$, then $\hat{d}_{B_i}(S)$ is equal to the actual number of free elements of S at the beginning of iteration k in stage i. Otherwise, if $p_i < 1$, $\hat{d}_{B_i}(S)$ estimates the number of free elements to be at least $s/2^i$ if the number of uncovered elements in $B_i(S)$ is at least $\log^{10} s \cdot \log^{10} t$ and otherwise $\hat{d}_{B_i}(S)$ estimates that S has fewer than $s/2^i$ free elements. Note that if we would not reuse the samples within the same stage, then $\hat{d}_{B_i}(S)$ would be an unbiased estimator of $d(S)$. However, reusing the samples will be helpful in the analysis as we get the following monotonicity property within stage i: S does not get added to the set cover in stage i once the estimated number of uncovered elements in S is smaller than $s/2^i$. This fact will turn out to be very useful in order to establish the approximation guarantee of Algorithm 2.

Note that we only need to decide on $B_i(S)$ at the beginning of stage i and on $\mathcal{S}_{i,k}$ at the beginning of iteration k in stage i. The reason why we have decided to fix all the $B_i(S)$'s and all the $\mathcal{S}_{i,k}$'s at the beginning is to make the connection of Algorithm 2 to the subsequent algorithms more obvious.

4.2 Approximation Analysis of Algorithm 2

In this section we prove the following approximation guarantee for Algorithm 2.

THEOREM 4.1. *Let $\mathcal{S}_{\mathsf{cover}}$ be the solution returned by Algorithm 2. Then, $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] = O(\log s) \cdot \mathrm{OPT}$.*

Now we give a roadmap of our analysis. In Line 4, we include each set S in $\mathcal{S}_{i,k}$ with a probability of $2^k/t$. If S is not contained in $\mathcal{S}_{i,k}$, then S will not be added to the set cover in iteration k of stage i. If S is contained in $\mathcal{S}_{i,k}$, then S will be added to the set cover in iteration k of stage i if the estimated number of free elements in S at the beginning of iteration k in stage i is at least $s/2^i$. In this section we show that $\mathbb{E}\left[\mathcal{S}_{\mathsf{cover}}\right] = O(\log s) \cdot \mathrm{OPT}$. To that end, we upper-bound the expected size of

- $\mathcal{S}_{\mathsf{big},i}$: Sets which contain at least $4 \cdot \frac{s}{2^i}$ free elements at the beginning of stage i.
- $\mathcal{S}_{\mathsf{small}}$: Sets which contained fewer than $\frac{1}{2} \cdot \frac{s}{2^i}$ free elements in the iteration they got added to the set cover
- $\mathcal{S}_{\mathsf{normal},i}$: Sets which contained between $\frac{1}{2} \cdot \frac{s}{2^i}$ and $4 \cdot \frac{s}{2^i}$ free elements in the iteration they got added to the set cover

Bounding $\mathbb{E}\left[\mathcal{S}_{\mathsf{big},i}\right]$ and $\mathbb{E}\left[\mathcal{S}_{\mathsf{small}}\right]$ is straightforward. Let n_i denote the number of free elements at the beginning of stage i and let $s_i := s/2^i$. We show that $\mathbb{E}\left[\mathcal{S}_{\mathsf{normal},i}\right] = O(n_i/s_i)$ in a similar way as in the approximation analysis of Algorithm 1 by relying on the monotonicity property of the estimates established in the previous subsection. However, unlike Algorithm 1, the bound $\mathrm{OPT} = \Omega(n_i/s_i)$ does not generally hold in Algorithm 2, as we are loosing the guarantee that all sets contain no more than $2 \cdot s_i$ uncovered elements at the beginning of stage i. Nevertheless, we can still relate $\mathbb{E}\left[\mathcal{S}_{\mathsf{normal},i}\right]$ to OPT by using the bound on $\mathbb{E}\left[\mathcal{S}_{\mathsf{big},i}\right]$.

LEMMA 4.1. *Let S be some arbitrary set and i be some arbitrary stage. The probability that S gets added to the set cover in some arbitrary iteration of stage i such that S had fewer than $\frac{1}{2} \cdot s/2^i$ many free elements at the beginning of the iteration in which it was added is at most $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$.*

LEMMA 4.2. *Let S be some arbitrary set and i be some arbitrary stage. The probability that S has more than $2 \cdot s/2^i$ many free elements at the end of stage i is at most $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$.*

Proof of. Lemma 4.1 and Lemma 4.2. Note that we can assume that $p_i < 1$ as otherwise the estimated number of free elements of a set as computed in Line 7 is equal to the actual number of free elements of the set. In this case, both Lemma 4.1 and Lemma 4.2 trivially hold as the bad events cannot happen.

Next, note that the set of random bits used in Algorithm 2 are set right at the beginning. After Line 4 the algorithm basically executes a deterministic process. By the principle of deferred decision making, assume that we fix all the random bits except for those ones determining $B_i(S)$. Let k be the first iteration for which S is contained in $\mathcal{S}_{i,k}$. Since $p_{i,\log t} = 1$, there always exists such an iteration.

Since the algorithm uses the same set of elements for estimation within one stage, if in iteration k, the number of free elements in S is estimated to be less than $s/2^i$, then S will not be added to the set cover within stage i in any subsequent iteration. Furthermore, if the number of free elements in S is estimated to be at least $s/2^i$ in iteration k, then S will be added to the set cover and after that the estimated number of free elements will always be 0.

This implies that in order to prove Lemma 4.2, we can assume that S has less than $\frac{1}{2} \cdot \frac{s}{2^i}$ many free elements at the beginning of iteration k in stage i and we only need to show that the probability that the estimated number of free elements in S is at least $s/2^i$ is $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. Note that the first condition implies that the expected number of free elements in $B_i(S)$ is at most $\frac{1}{2} \cdot \log^{10} s \cdot \log^{10} t$. We only estimate S to have a size of at least $s/2^i$ if $B_i(S)$ contains at least $\log^{10} s \cdot \log^{10} t$ many free elements. A Chernoff bound (Theorem 2.1-B) implies that this happens with a probability of $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$.

Similarly, to prove Lemma 4.1, we can assume that S has at least $2 \cdot s/2^i$ many free elements at the beginning of iteration k in stage i and we need to bound the probability that the estimated number of free elements of S is less than $s/2^i$. A Chernoff bound (Theorem 2.1-A) implies that this happens with a probability of $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. ∎

LEMMA 4.3. *Let $\mathcal{S}_{\mathsf{small}}$ be the family of sets containing those sets which were added to the set cover in some arbitrary stage i despite having fewer than $\frac{1}{2} \cdot \frac{s}{2^i}$ many free elements at the beginning of the iteration in which they were added. Then,* $\mathbb{E}\left[|\mathcal{S}_{\mathsf{small}}|\right] = O(\mathrm{OPT})$.

Proof. By Lemma 4.1 and a union bound over the $\log s$ many stages, any set S is in $\mathcal{S}_{\mathsf{small}}$ with a probability of $\log s \cdot O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. The lemma follows as there are at most $m \leq n \cdot t$ many sets and $\mathrm{OPT} \geq n/s$. □

LEMMA 4.4. *Consider some arbitrary stage i. Let e be an arbitrary element and $X_e^{(i)}$ the random variable which equals 0 if e does not get covered for the first time during stage i, and otherwise is equal to the number of sets that contain e and were added to the set cover in the same iteration in which e was covered for the first time. Then,* $\mathbb{E}\left[X_e^{(i)}\right] \leq 5$. *Furthermore, this bound even holds if we fix all the randomness needed for all stages up to stage i in some arbitrary way.*

Proof. Let $\mathcal{S}_e$ be the family of sets that contain e. We show this lemma by using the principle of deferred decision making. Suppose that we have fixed all the randomness used in stage i other than the sets in $\mathcal{S}_e$ at the beginning: e.g. it is not yet determined whether $S \in \mathcal{S}_{i,k}$ or not for $S \in \mathcal{S}_e$. We will fix those decisions only in the iteration for which we need them. Let $N_{i,k}$ denote the random variable (with respect to the randomness which we have not fixed) that is equal to the number of sets in $\mathcal{S}_e$ whose estimated size is at least $s/2^i$ at the beginning of iteration k in stage i. As we always use the same elements to estimate the size of a particular set within stage i, we have $t \geq N_{i,1} \geq N_{i,2} \geq ... \geq N_{i,\log t}$. As we have fixed all the randomness needed to execute all the stages up to stage i, $N_{i,1}$ is just equal to some number $n_{i,1}$. Now, each of those $n_{i,1}$ sets is considered in the first iteration in stage i with probability $\frac{2}{t}$. Thus, the number of sets which cover e in the first iteration is a random variable $Y_1 \sim \mathrm{Bin}(n_{i,1}, p_{i,1})$. If $Y_1 > 0$, then $X_e^{(i)} = Y_1$. Otherwise, none of the sets in $\mathcal{S}_e$ is added to the set cover. Given that information, $N_{i,2}$ is equal to some number $n_{i,2}$. Then, the number of sets containing e in the second iteration is a random variable $Y_2 \sim \mathrm{Bin}(n_2, p_{i,2})$ and so on. Note that this process is completely analogous to the process in the proof of Lemma 3.3. Thus, we can conclude that $\mathbb{E}\left[X_e^{(i)}\right] \leq 5$. □

LEMMA 4.5. *Let $\mathcal{S}_{\mathsf{big},i}$ be a family of sets that contains those sets which have more than $4 \cdot \frac{s}{2^i}$ many free elements at the beginning of stage i. Then,* $\mathbb{E}\left[|\mathcal{S}_{\mathsf{big},i}|\right] \leq \frac{n}{t \cdot s^3}$ *and* $\Pr[|\mathcal{S}_{\mathsf{big},i}| > \frac{n}{s^2}] \leq \frac{1}{st}$.

Proof. Consider an arbitrary set S. For $i = 1$, S is never contained in $\mathcal{S}_{\mathsf{big},i}$. For $i > 1$, S is only in $\mathcal{S}_{\mathsf{big},i}$ if S has at least $2 \cdot (s/2^{i-1})$ many free elements at the end of stage $i-1$. By Lemma 4.2, this only happens with a probability of $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. As we have at most $t \cdot n$ many sets, $\mathbb{E}\left[|\mathcal{S}_{\mathsf{big},i}|\right] \leq \frac{n}{t \cdot s^3}$. Hence, a Markov bound implies that $\Pr[|\mathcal{S}_{\mathsf{big},i}| > \frac{n}{s^2}] \leq \frac{1}{st}$. □

LEMMA 4.6. *Consider the beginning of stage i. Let $\mathcal{S}_{\mathsf{normal},i}$ be the family of sets consisting of those sets which are added to the set cover in some arbitrary stage i and additionally, at the beginning of the iteration in which they are added to the set cover, contain at least $\frac{1}{2} \cdot (s/2^i)$ and at most $4 \cdot (s/2^i)$ many free elements. Suppose that the randomness needed for all stages up to i are fixed in advance. Let n_i denote the number of free elements at the beginning of stage i and $s_i := s/2^i$. Then,* $\mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}|\right] = O(n_i/s_i)$.

Proof. Let $\mathcal{E}_i$ denote the set of uncovered elements at the beginning of stage i. As all sets in $\mathcal{S}_{\mathsf{normal},i}$ have at least $\frac{1}{2} \cdot (s/2^i)$ many free elements at the beginning of the iteration in which they are added to the set cover,

$$|\mathcal{S}_{\mathsf{normal},i}| \leq \frac{1}{\frac{1}{2} \cdot s_i} \cdot \sum_{e \in \mathcal{E}_i} X_e^{(i)}$$

Thus, as $|\mathcal{E}_i| = n_i$ and $\mathbb{E}\left[X_e^{(i)}\right] \leq 5$,

$$\mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}|\right] \leq \frac{5 \cdot n_i}{\frac{1}{2} \cdot s_i} = O(n_i/s_i)$$

□

LEMMA 4.7. *Consider the beginning of stage i. Let $\mathcal{S}_{\mathsf{normal},i}$ be the family of sets that are added to the set cover in stage i and additionally, at the beginning of the iteration in which they are added to the set cover, contain at least $\frac{1}{2} \cdot (s/2^i)$ and at most $4 \cdot (s/2^i)$ many free elements. Then,* $\mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}|\right] = O(\mathrm{OPT})$.

Proof. Assume that we are at the beginning of stage i. As before, we denote with n_i the number of free elements at the beginning of stage i and $s_i := s/2^i$. As a reminder, $\mathcal{S}_{\mathsf{big},i}$ was defined as the sets which have more than $4 \cdot s/2^i$ many free elements at the beginning of stage i. We have the following lower bound for OPT

$$\mathrm{OPT} \geq \frac{n_i - s \cdot |\mathcal{S}_{\mathsf{big},i}|}{4 \cdot s_i}.$$

This holds because after adding all sets in $\mathcal{S}_{\mathsf{big},i}$ to the set cover, at least $n_i - s \cdot |\mathcal{S}_{\mathsf{big},i}|$ free elements remain and the maximal number of free elements any set contains is at most $4 \cdot s_i$. Assume that $|\mathcal{S}_{\mathsf{big},i}| \leq \frac{n}{s^2}$. Then

$$\text{OPT} \geq \frac{n_i - s \cdot |\mathcal{S}_{\mathsf{big},i}|}{4 \cdot s_i} \geq \Omega(n_i/s_i) - \frac{n}{s} \geq \Omega(n_i/s_i) - \text{OPT}.$$

Hence, in this case $\text{OPT} = \Omega(n_i/s_i) \geq \Omega(\mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}|\right])$. (Note that $|\mathcal{S}_{\mathsf{big},i}|$ only depends on the randomness needed for all stages prior to stage i and therefore we can apply the previous lemma)
By Lemma 4.5, $\Pr[|\mathcal{S}_{\mathsf{big},i}| > \frac{n}{s^2}] \leq \frac{1}{s \cdot t}$. Therefore

$$\begin{aligned}\mathbb{E}\left[\mathcal{S}_{\mathsf{normal},i}\right] &= \mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}| \mid |\mathcal{S}_{\mathsf{big},i}| > \frac{n}{s^2}\right] \\ &\quad \cdot \Pr\left[|\mathcal{S}_{\mathsf{big},i}| > \frac{n}{s^2}\right] \\ &\quad + \mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}| \mid |\mathcal{S}_{\mathsf{big},i}| \leq \frac{n}{s^2}\right] \\ &\quad \cdot \Pr\left[|\mathcal{S}_{\mathsf{big},i}| \leq \frac{n}{s^2}\right] \\ &\leq n \cdot t \cdot \frac{1}{s \cdot t} + \mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}| \mid |\mathcal{S}_{\mathsf{big},i}| \leq \frac{n}{s^2}\right] \\ &\leq O(n/s) + O(\text{OPT}) = O(\text{OPT}).\end{aligned}$$

□

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

Proof of. Theorem 4.1. First, observe that

$$\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] \leq \mathbb{E}\left[|\mathcal{S}_{\mathsf{small}}|\right] + \sum_{i=1}^{\log s} \mathbb{E}\left[|\mathcal{S}_{\mathsf{big},i}|\right] + \mathbb{E}\left[|\mathcal{S}_{\mathsf{normal},i}|\right].$$

Then, from Lemmas 4.3, 4.5 and 4.7 we conclude

$$\begin{aligned}\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] &\leq O(n/s) + \sum_{i=1}^{\log s} \frac{n}{t \cdot s^3} + O(\text{OPT}) \\ &\leq O(\log s) \cdot \text{OPT}.\end{aligned}$$

□

4.2.1 Bounding the Probability of "Bad" Events

In this section we introduce the notion of *bad elements* and *bad sets*. Bounding the probability of some element or set being *bad* will play a central role in relating all the subsequent algorithms to Algorithm 2. The connection we get is the following: Consider that we are executing Algorithm 2 and some subsequent algorithm with the same randomness. Let S be some arbitrary set and assume that there does not exist a bad element or a bad set in the $(c \cdot \log s \cdot \log t)$-hop neighborhood of S for some large enough constant c. Then, S is either added to the set cover in both algorithms or in none of them. As the probability is fairly large that there does not exist such a bad element or bad set in the neighborhood of S, by the approximation analysis of Algorithm 2, we conclude that the subsequent algorithm returns a set cover with an expected size of $O(\log s) \cdot \text{OPT}$.

An element e is a *bad element* iff there exist $i \in [\log s]$ and $1 \leq k_1 \leq k_2 \leq \log t$ such that the following conditions hold:

1. e is uncovered at the beginning of iteration k_1 of stage i.

2. e is contained in more than $\log^{20} s \cdot \log^{20} t \cdot 2^{k_2 - k_1}$ sets in $\mathcal{S}_{i,k_2}$ with an estimated size of at least $s/2^i$ at the beginning of iteration k_1 of stage i.

LEMMA 4.8. *For any element e, $\Pr[e \text{ is bad}] = O\left(\frac{1}{(st)^{\log^4 s \cdot \log^4 t}}\right)$.*

Proof. First consider the case $k_1 = 1$. The expected number of sets in $\mathcal{S}_{i,k_2}$ which contain e is at most $t \cdot \frac{2^{k_2}}{t} = 2^{k_2}$. Hence, an application of Chernoff bound (Theorem 2.1-B) implies that the probability, that the number of sets in $\mathcal{S}_{i,k_2}$ exceeds $\log^{20} s \cdot \log^{20} t \cdot 2^{k_2 - k_1}$, is at most $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$.
Next, consider the case $k_1 > 1$ and we consider two subcases for this case.

- **There are at least $\log^{10} s \cdot \log^{10} t \cdot \frac{t}{2^{k_1 - 1}}$ many sets that contain e and have estimated size of at least $s/2^i$ at the beginning of iteration $k_1 - 1$ in stage i.** Then, the probability that e remains uncovered at the beginning of iteration k_1 is at most

$$\left(1 - \frac{2^{k_1 - 1}}{t}\right)^{\log^{10} s \cdot \log^{10} t \cdot \frac{t}{2^{k_1 - 1}}} = O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right).$$

- **There are less than $\log^{10} s \cdot \log^{10} t \cdot \frac{t}{2^{k_1 - 1}}$ many sets that contain e and have estimated size of at least $s/2^i$ at the beginning of iteration $k_1 - 1$ in stage i.** This implies that the expected number of sets in S_{i,k_2} that contain e and have estimated size of at least $s/2^i$ at the beginning of iteration k_1 of stage i is at most $\log^{10} s \cdot \log^{10} t \cdot \frac{t}{2^{k_1 - 1}} \cdot \frac{2^{k_2}}{t}$. Hence, an application of Chernoff bound (Theorem 2.1-B) implies that the number of sets in $\mathcal{S}_{i,k_2}$ that contain e and have an estimated size of at least $s/2^i$ at the beginning of iteration k_1 in stage i exceeds $\log^{20} s \cdot \log^{20} t \cdot 2^{k_2 - k_1}$ with a probability of at most $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. The lemma follows by a union bound over the at most $O(\log s \cdot \log^2 t)$ many choices for i, k_1 and k_2.

□

A set S is a *bad set* iff there exist $1 \leq i_1 \leq i_2 \leq \log s$ such that the number of free elements in $B_{i_2}(S)$ at the beginning of stage i_1 exceeds $\log^{20} s \cdot \log^{20} t \cdot 2^{i_2 - i_1}$.

Lemma 4.9. *For any set S,* $\Pr[S \textit{ is bad }] = O\left(\frac{1}{(st)^{\log^4 s \cdot \log^4 t}}\right)$.

Proof. We first consider the case that S has more than $2 \cdot s/2^{i_1 - 1}$ many free elements at the beginning of stage i_1. Lemma 4.2 implies that this happens with a probability of $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. Next, suppose that S has less than $2 \cdot s/2^{i-1}$ free elements at the beginning of stage i_1. This implies that the expected number of free elements in $B_{i_2}(S)$ is at most $2 \cdot s/2^{i_1 - 1} \cdot p_{i_2} \leq 4 \cdot \log^{10} s \cdot \log^{10} t \cdot 2^{i_2 - i_1}$. Hence, an application of Chernoff bound (Theorem 2.1-B) implies that the number of free elements in $B_{i_2}(S)$ is more than $\log^{20} s \cdot \log^{20} t \cdot 2^{i_2 - i_1}$ with a probability of at most $O\left(\frac{1}{(st)^{\log^5 s \cdot \log^5 t}}\right)$. The lemma follows by a union bound over the $O(\log^2(s))$ many possible choices for i_1 and i_2. □

Consider the bipartite graph G induced by the set system $(\mathcal{S}, \mathcal{E})$. A set S (resp., An element e) has a *bad neighborhood* iff there exists a bad element (resp., a bad set) in the 10-hop neighborhood of S (resp., e) in G.

Lemma 4.10. *The probability that an arbitrary set or element has a bad neighborhood is at most* $O\left(\frac{1}{(st)^{\log^3 s \cdot \log^3 t}}\right)$.

Proof. The lemma directly follows by Lemma 4.8 and Lemma 4.9 together with a union bound over the at most poly(st) many sets and elements in the 10-hop neighborhood of S. □

5 Beyond the Parnas-Ron Paradigm

By applying the standard reduction of Parnas and Ron [32] for transforming distributed algorithms into LCAs, Algorithm 1 and Algorithm 2 can be simulated in the local computation model with a query complexity of $(st)^{O(\log s \log t)}$. In this section, we present an LCA that can be implemented with a query complexity of $(st)^{O\left(\log s \sqrt{\log t}\right)}$.

Overview of Our Approach We now discuss our general ideas for obtaining a query-efficient LCA simulation. These ideas will be applied to the algorithms developed in this section and in Section 6. The main purpose of the discussion that follows is to illustrate a way to design LCAs for the set cover so that it suffices to access only a "small" number of sets and elements in the direct neighborhood of another set/element.

LCA from the point of view of a set. Our LCA simulations test whether a given set S is in the cover or not by recursively testing whether S has been added to the cover in any of the rounds, starting from round $\log s \cdot \log t$. To be more detailed, consider a round r of stage i. To decide whether S has been added to the set cover by the end of round r in Algorithm 2, we perform two steps:

(1) We test whether S is added to the cover during the first $r - 1$ rounds of the algorithm.

(2) If S has not been added to the set cover yet, we count the number of yet uncovered elements in $B_i(S)$ at the beginning of round r and proceed based on the count.

To perform step (1), it suffices to perform the same two steps for only round $r - 1$. For step (2), as we already know the outcome of step (1), we expect that S does not have more than $10 \cdot s/2^i$ many free elements at the beginning of stage i. As each element of S is contained in $B_i(S)$ with a probability of $\min\left(1, \log^{10} s \cdot \log^{10} t \cdot \frac{2^i}{s}\right)$, we furthermore expect that $B_i(S)$ does not have more than poly$(\log s \log t)$ many free elements at the beginning of stage i, and therefore at the beginning of round r. Intuitively, the sooner we detect those uncovered elements, the more efficient LCA we are likely to obtain. The reason is that testing whether an element is free or not requires performing additional recursive tests. Since in general $B_i(S)$ may contain a lot of elements that are covered before stage i, these recursive tests for each covered element can be very expensive query-wise. To alleviate that, informally speaking, we remove all previously covered elements at the beginning of a stage i (see Algorithm 4) or at the beginning of some phase j that consists of multiple iterations (see Algorithm 3). Therefore, within a phase j (or stage i), S only needs to check for all elements in this "sparsified" version of $B_i(S)$, which only includes the free elements in $B_i(S)$, whether they are covered or not at the beginning of round r. We expect this "sparsified" version of $B_i(S)$ to only consist of $O(\text{poly}(\log s \log t))$ many elements.

LCA from the point of view of an element. In the following, assume that round r corresponds to iteration k in stage i. Furthermore, assume that we merge iterations into multiple phases, with iteration k being part of phase j. In order to test whether an element e is covered for the first time in round r of Algorithm 2, our LCA simulation performs the following test:

- Given that e was not yet covered, does *any* of the sets e is contained in get added to the set cover in round r?

To answer this question efficiently, ideally, e should invoke tests on a small number of sets. We develop the following two ideas to reduce the number of sets that the above question should be answered for:

(i) As in Algorithm 2, we sample all sets that are potentially considered in stage i of iteration k beforehand. This sample is denoted by $\mathcal{S}_{i,k}$.

(ii) At the beginning of phase j, we remove all sets from $\mathcal{S}_{i,k}$ that have an estimated set size smaller than $s/2^i$.

These ideas have the following positive consequence. Assume that each phase has length T and that round r corresponds to the ℓ-th iteration within phase j. If e is still uncovered at the beginning of phase j, then we expect e to be contained in fewer than $10 \cdot \frac{t}{2^{j \cdot T}}$ many "large" sets, as each "large" set is added to the set cover with a probability of $\frac{2^{j \cdot T - 1}}{t}$ in iteration $j \cdot T - 1$. Each of those large sets is contained in $\mathcal{S}_{i,k}$ with a probability of $\frac{2^{j \cdot T + l}}{t}$. Hence, we expect that the number of sets containing e and belonging to the sparsified version of $\mathcal{S}_{i,k}$ is at most $2^l \leq 2^T$, which for our choice of T is significantly less than t.

Handling expectations. In the previous paragraphs, most of the statements we provided on the sparsified neighborhoods of elements and sets were stated only in expectation. However, as noted in Section 4, the query complexity still needs to be guaranteed even if the random bits are chosen adversarially. To guarantee this, we immediately add a set to the set cover if the sparsified version of $B_i(S)$ contains much more elements than expected at the beginning of a phase (Algorithm 3) or stage (Algorithm 5). Likewise, we "pretend" that an element is covered if e is contained in much more sets than expected in the sparsified version of $\mathcal{S}_{i,k}$. This is also the only difference to Algorithm 2. As e reports to be covered, it might happen that e is still uncovered at the end of the algorithm. In that case, we simply add one set to the set cover that contains e. In this way, bad randomness can only affect the approximation guarantee, but not the query complexity of the LCA. In this way, intuitively, while not completely true, one can think about executing one phase in a sparsified set system where the maximal set size is $O(\text{poly}(\log s \log t))$ and the maximal number of sets any element is contained in is $2^{O(T)}$. Note that there is a trade-off for the phase length T. With increasing T, the sparsified versions are becoming less and less sparse. Likewise, in order to produce the illusion of a sparsified set system, we need to query the set system corresponding to the previous phase poly(st) many times for each query that we receive for the sparsified set system. We balance this tradeoff by setting $T = \sqrt{\log t}$.

5.1 Details about Algorithm 3 In this section we design Algorithm 3. Compared to Algorithm 2, Algorithm 3 processes iterations in groups of T. Each group is called a *phase*. Algorithm 3 also samples sets $B_i(S)$, for each stage i and each set S. At the beginning of phase j, we compute the set $B_{i,j}(S)$ by removing all elements in $B_i(S)$ that are covered at the beginning of phase j. The LCA can efficiently estimate the number of free elements within phase j by only counting the number of free elements in $B_{i,j}(S)$. Furthermore, $\hat{\mathcal{S}}_{i,k}$ denotes the family of sets obtained by removing all the sets from $\mathcal{S}_{i,k}$ that have an estimated number of free elements smaller than $s/2^i$ at the beginning of the respective phase.

5.2 Simulation of Algorithm 3 in LCA In the rest of this section we describe the LCA simulation of Algorithm 3 and analyze its query complexity. Our final goal is to provide oracle access to the set cover produced by Algorithm 3. We call this oracle $\mathcal{O}^{\mathsf{set}}$. That is, $\mathcal{O}^{\mathsf{set}}(S)$ answers if S is part of the set cover or not. It is convenient to define intermediate oracles for the analysis. Namely, $\mathcal{O}^{\mathsf{set}}_i(S)$ outputs if S is part of the set cover at the end of the i-th stage, $\mathcal{O}^{\mathsf{set}}_{i,j}(S)$ outputs if S is part of the set cover at the end of phase (i,j) and $\mathcal{O}^{\mathsf{set}}_{i,j,\ell}(S)$ outputs if S is part of the set cover at the end of iteration (i,j,ℓ). We remark that $\mathcal{O}^{\mathsf{set}}_{i,j,0}(S)$ is defined to output whether S is added to the set cover at the beginning of iteration $(i,j,1)$. Note that the oracle $\mathcal{O}^{\mathsf{set}}_{i,j,T}$ is always equal to the oracle $\mathcal{O}^{\mathsf{set}}_{i,j}$ and $\mathcal{O}^{\mathsf{set}}_{i,T-1}$ is always equal to $\mathcal{O}^{\mathsf{set}}_i$. However, $\mathcal{O}^{\mathsf{set}}_{\log(s)}$ is not equal to $\mathcal{O}^{\mathsf{set}}$ because of Line 20. We also define oracles $\mathcal{O}^{\mathsf{el}}_i(e)$, $\mathcal{O}^{\mathsf{el}}_{i,j}(e)$ and $\mathcal{O}^{\mathsf{el}}_{i,j,\ell}(e)$ which answer if e was covered at the end of stage i, phase (i,j) or iteration (i,j,ℓ), respectively. Note that they also answer that e is covered even if e only pretends that it is covered. We also remark that $\mathcal{O}^{\mathsf{el}}_{i,j,0}(e)$ is defined to output whether e is covered by any of the sets at the beginning of iteration $(i,j,1)$.

Description of $\mathcal{O}^{\mathsf{set}}_{i,j}$ and $\mathcal{O}^{\mathsf{el}}_{i,j}$ given query access to $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$. In this section, we assume to have query access to $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$ which respectively denote the collection of unselected sets and uncovered elements at beginning of phase (i,j). Given that, we describe how to answer oracles $\mathcal{O}^{\mathsf{set}}_{i,j}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i,j}(e)$ for some set S or element e by querying $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$ at most poly(st) times. We first note that it takes $O(1)$ many queries to decide for a set S and some $k \in [\log t]$ if $S \in \mathcal{S}_{i,k}$: S is contained in $\mathcal{S}_{i,k}$ with a probability of $\frac{2^k}{t}$.

Furthermore, for a given set S, we can get all elements in $B_{i,j}(S)$ with $O(s)$ many queries to $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$ and for a given element $e \in \mathcal{E}_{i,j}$ and some $k \in \{j \cdot T + 1, ..., (j+1) \cdot T\}$, we can get all the sets

Algorithm 3: A variant of Algorithm 2 that can be simulated by an LCA with $(st)^{O(\log s\sqrt{\log t})}$ queries.

```
1  $T \overset{\text{def}}{=} \sqrt{\log(t)}$
2  for each pair of $(i,k)$ where $i \in [\log s]$ and $k \in [\log t]$ do
3      $B_i(S)$ and $\mathcal{S}_{i,k}$ are generated in the exact same way as in Algorithm 2
4  for $i = 1$ to $\log s$ do
5      for phase $j = 0$ to $T-1$ do
6          for each set $S \in \mathcal{S}$ do
7              $B_{i,j}(S) \leftarrow B_i(S) \cap \mathcal{E}_{i,j}$ // $\mathcal{E}_{i,j}$ denotes the set of free elements at the beginning of the phase
8              if $|B_{i,j}(S)| \geq \log^{20} s \cdot \log^{20} t$ then
                // whether $S$ is a bad set
9                  add $S$ to $\mathcal{S}_{\mathsf{cover}}$.
10         for each $k \in \{j \cdot T+1, j \cdot T+2, ..., (j+1) \cdot T\}$ do
11             $\widehat{\mathcal{S}}_{i,k} \leftarrow \{S \mid S \in \mathcal{S}_{i,k}, \hat{d}_{B_i}(S) \geq s/2^i\}$
12             for each $e \in \mathcal{E}_{i,j}$ in parallel do
13                 if $d_{\widehat{\mathcal{S}}_{i,k}}(e) \geq \log^{20} s \cdot \log^{20} t \cdot 2^T$ then // whether $e$ is a bad element
14                     pretend that $e$ is covered. // $e$ will be covered in Line 20.
15         for iteration $\ell$ = 1 to $T$ do
16             for each $S \in \widehat{\mathcal{S}}_{i,j \cdot T+\ell}$ in parallel do
17                 if $\hat{d}_{B_i}(S) \geq s/2^i$ then
18                     add $S$ to $\mathcal{S}_{\mathsf{cover}}$.
19 for each free element in parallel $e$ do // Handling bad elements
20     add the set with smallest ID which contains $e$ to $\mathcal{S}_{\mathsf{cover}}$.
```

in $\hat{\mathcal{S}}_{i,k}$ that contain e with $O(st)$ many queries to $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$.

To answer $\mathcal{O}^{\mathsf{set}}_{i,j,\ell}(S)$, we first check whether S was already added to the set cover at the beginning of iteration (i,j,ℓ) or not:

1. **$\ell = 0$:** we simply check whether $S \notin \mathcal{S}_{i,j}$
2. **$\ell > 0$:** we recursively invoke $\mathcal{O}^{\mathsf{set}}_{i,j,\ell-1}(S)$.

If S was indeed added, then we can immediately answer $\mathcal{O}^{\mathsf{set}}_{i,j,\ell}(S)$. Otherwise, we need to figure out whether S gets added in iteration (i,j,ℓ) or not. To that end, we again consider two cases:

1. **$\ell = 0$:** S gets added to the set cover iff $|B_{i,j}(S)| \geq \log^{20} s \cdot \log^{20} t$.
2. **$\ell > 0$:** we first check whether $S \in \mathcal{S}_{i,j \cdot T+\ell}$. If $S \in \mathcal{S}_{i,j \cdot T+\ell}$, then for each element $e \in B_{i,j}(S)$, we invoke the oracle $\mathcal{O}^{\mathsf{el}}_{i,j,\ell-1}(e)$. If the estimated number of free elements in S is at least $s/2^i$, then S gets added to the set cover in iteration (i,j,ℓ), otherwise not. This step requires at most $\log^{20} s \cdot \log^{20} t$ many recursive calls to the oracle $\mathcal{O}^{\mathsf{el}}_{i,j,\ell-1}$.

Next, we describe how to answer $\mathcal{O}^{\mathsf{el}}_{i,j,\ell}(e)$. To do so, we first check whether e was already covered (or pretends to be covered) at the beginning of iteration (i,j,ℓ).

1. **$\ell = 0$:** we simply check whether $e \notin \mathcal{E}_{i,j}$,
2. **$\ell > 0$:** we recursively invoke $\mathcal{O}^{\mathsf{el}}_{i,j,\ell-1}(e)$.

If e is not covered, then we proceed as follows:

1. **$\ell = 0$:** we first invoke $\mathcal{O}^{\mathsf{set}}_{i,j,0}$ for all sets which contain e. If none of them is added to the set cover by iteration $(i,j,0)$, then we check if e pretends to be covered in Line 14 by checking for each $k \in \{j \cdot T+1, ..., (j+1) \cdot T\}$ if e is contained in more than $\log^{20} s \cdot \log^{20} t \cdot 2^T$ many sets in $\hat{\mathcal{S}}_{i,k}$.
2. **$\ell > 0$:** Otherwise, for all sets in $\hat{\mathcal{S}}_{i,j \cdot T+\ell}$ containing e, we invoke the oracle $\mathcal{O}^{\mathsf{set}}_{i,j,\ell}$ to check if any of them is added to the set cover by the end of iteration (i,j,ℓ). If that's the case, then e is covered after iteration (i,j,ℓ). This step requires at most $\log^{20} s \cdot \log^{20} t \cdot 2^T$ recursive calls to $\mathcal{O}^{\mathsf{set}}_{i,j,\ell-1}$ oracle.

Let $Q_{set}(\ell)$ and $Q_{el}(\ell)$ be an upper bound on the total number of queries to $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$ that we need in order to answer $\mathcal{O}^{\mathsf{set}}_{i,j,\ell}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i,j,\ell}(e)$ respectively. From the description, we derive the following recursions:

$$\begin{aligned}
Q_{set}(0) &= O(\mathrm{poly}(st)) \\
Q_{set}(\ell) &= Q_{set}(\ell-1) + \log^{20} s \cdot \log^{20} t \cdot Q_{el}(\ell-1) + O(s) \\
&\leq \mathrm{poly}(\log s) \cdot 2^{O(T)} \cdot (Q_{set}(\ell-1) + Q_{el}(\ell-1)) \\
&\quad + O(st) \text{ for } \ell > 0 \\
Q_{el}(0) &= O(\mathrm{poly}(st)) \\
Q_{el}(\ell) &= Q_{el}(\ell-1) + \log^{20} s \cdot \log^{20} t \cdot 2^T \cdot Q_{set}(\ell) + O(st) \\
&\leq \mathrm{poly}(\log s) \cdot 2^{O(T)} \cdot (Q_{set}(\ell-1) + Q_{el}(\ell-1)) \\
&\quad + O(st) \text{ for } \ell > 0
\end{aligned}$$

Therefore, we get $Q_{set}(\ell) = Q_{el}(\ell) = O(\mathrm{poly}(st)) \cdot (\mathrm{poly}(\log s) \cdot 2^{O(T)})^{\ell}$. This implies $Q_{set}(T) = Q_{el}(T) = O(\mathrm{poly}(st))$. Recall that we have $\mathcal{O}^{\mathsf{set}}_{i,j,T} = \mathcal{O}^{\mathsf{set}}_{i,j}$ and $\mathcal{O}^{\mathsf{el}}_{i,j,T} = \mathcal{O}^{\mathsf{el}}_{i,j}$. Thus, we can answer both $\mathcal{O}^{\mathsf{set}}_{i,j}$ and $\mathcal{O}^{\mathsf{el}}_{i,j}$ with $O(\mathrm{poly}(st))$ many queries.

Description of $\mathcal{O}^{\mathsf{set}}_{i,j}$ and $\mathcal{O}^{\mathsf{el}}_{i,j}$ given query access to $\mathcal{O}^{\mathsf{set}}_{i,j-1}$ and $\mathcal{O}^{\mathsf{el}}_{i,j-1}$ $(j > 1)$. It is easy to verify that we can give query access to $(\mathcal{S}_{i,j}, \mathcal{E}_{i,j})$ by using $O(st)$ many queries to $\mathcal{O}^{\mathsf{set}}_{i,j-1}$, $\mathcal{O}^{\mathsf{el}}_{i,j-1}$ and the original set cover instance. Together with the previous paragraph, this implies that we can answer $\mathcal{O}^{\mathsf{set}}_{i,j}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i,j}(e)$ with $O(\text{poly}(st))$ many oracle calls to $\mathcal{O}^{\mathsf{set}}_{i,j-1}$, $\mathcal{O}^{\mathsf{el}}_{i,j-1}$ and the original set cover instance.

Description of $\mathcal{O}^{\mathsf{set}}_{i}$ and $\mathcal{O}^{\mathsf{el}}_{i}$ given query access to $(\mathcal{S}_{i,0}, \mathcal{E}_{i,0})$. The query complexity analysis of the previous case together with a simple induction argument implies that we can answer $\mathcal{O}^{\mathsf{set}}_{i,j}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i,j}(e)$ with $(st)^{O(j)}$ many queries to $(\mathcal{S}_{i,0}, \mathcal{E}_{i,0})$. In particular, this implies that we can answer the oracles corresponding to the last phase in stage i, namely $\mathcal{O}^{\mathsf{set}}_{i,T-1}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i,T-1}(e)$, with $(st)^{O(\sqrt{\log(t)})}$ many queries to $(\mathcal{S}_{i,0}, \mathcal{E}_{i,0})$ and the original set cover instance. Remember that $\mathcal{O}^{\mathsf{set}}_{i,T-1}(S) = \mathcal{O}^{\mathsf{set}}_{i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i,T-1}(e) = \mathcal{O}^{\mathsf{el}}_{i}(e)$. This implies that we can answer $\mathcal{O}^{\mathsf{set}}_{i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i}(e)$ with $(st)^{O(\sqrt{\log(t)})}$ many queries to $(\mathcal{S}_{i,0}, \mathcal{E}_{i,0})$ and the original set cover instance.

Description of $\mathcal{O}^{\mathsf{set}}_{i}$ and $\mathcal{O}^{\mathsf{el}}_{i}$ given query access to $\mathcal{O}^{\mathsf{set}}_{i-1}$ and $\mathcal{O}^{\mathsf{el}}_{i-1}$ $(i > 1)$. It is easy to verify that we can give query access to $(\mathcal{S}_{i,0}, \mathcal{E}_{i,0})$ by using $O(st)$ many queries to $\mathcal{O}^{\mathsf{set}}_{i-1}$, $\mathcal{O}^{\mathsf{el}}_{i-1}$ and the original set cover instance. Together with the query complexity analysis of the previous paragraph, this implies that we can answer $\mathcal{O}^{\mathsf{set}}_{i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i}(e)$ with $(st)^{O(\sqrt{\log(t)})}$ many oracle calls to $\mathcal{O}^{\mathsf{set}}_{i-1}$ and $\mathcal{O}^{\mathsf{el}}_{i-1}$ and the original set cover instance.

Description of $\mathcal{O}^{\mathsf{set}}$ given query access to $(\mathcal{S}, \mathcal{E})$. Note that $(\mathcal{S}_{1,0}, \mathcal{E}_{1,0}) = (\mathcal{S}, \mathcal{E})$. Hence, a simple induction argument implies that we can answer $\mathcal{O}^{\mathsf{set}}_{i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{i}(e)$ with $(st)^{O(i\cdot\sqrt{\log(t)})}$ many queries to $(\mathcal{S}, \mathcal{E})$. In particular, this implies that we can answer $\mathcal{O}^{\mathsf{set}}_{\log(s)}(S)$ with $(st)^{O(\log(s)\cdot\sqrt{\log(t)})}$ many queries to $(\mathcal{S}, \mathcal{E})$. Now, it is straightforward to implement $\mathcal{O}^{\mathsf{set}}(S)$ with $O(st)$ many calls to $\mathcal{O}^{\mathsf{set}}_{\log(s)}$ and $(\mathcal{S}, \mathcal{E})$. This implies that the total query complexity is $(st)^{O(\log(s)\cdot\sqrt{\log(t)})}$.

5.3 Approximation Proof

In this section we analyze the approximation guarantee of Algorithm 3 and prove the following result.

THEOREM 5.1. *Let $\mathcal{S}_{\mathsf{cover}}$ be the solution returned by Algorithm 3. Then,* $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] = O(\log s) \cdot \text{OPT}$.

To prove Theorem 5.1, we highly reuse the statements proved for Algorithm 2. In particular, Theorem 4.1 shows that Algorithm 2 in expectation returns a $O(\log s)$-approximate solution. In this section, we will analyze the similarity between Algorithm 3 and Algorithm 2, and show that from the point of view of most of the elements and sets these two algorithms are identical. Then we apply Theorem 4.1 to prove Theorem 5.1.

We now briefly describe the notation and the setup that is used for proving the required lemmas. First, we randomly choose sets $B_i(S)$ and families $\mathcal{S}_{i,k}$ as described in Line 3 and Line 4 of Algorithm 2. After that we execute Algorithm 2 and Algorithm 3 "simultaneously" with the same $B_i(S)$'s and $\mathcal{S}_{i,k}$'s. For $r \in \{0, 1, \ldots, \log s \cdot \log t\}$, some set S is considered to have the same state after r rounds in both algorithms iff S was either added to the set cover in both algorithms or in none of them after r rounds. Similarly, an element e is considered to be in the same state in both algorithms after r rounds iff e is still free after r rounds in both algorithms or e is covered in both algorithms after r rounds. We say that the T-hop neighborhood of some set S (resp., element e) is in the same state after r rounds in both algorithms iff all the sets and all the elements in the T-hop neighborhood of S (resp., e) have the same state in both algorithms after r rounds.

LEMMA 5.1. *Consider an arbitrary round $r < \log s \cdot \log t$. For a set S (an element e, respectively) let $EQ^{(r)}_{S,10}$ ($EQ^{(r)}_{e,10}$, respectively) denote the event that the 10-hop neighborhood of S (e, respectively) in Algorithm 2 and Algorithm 3 is in the same state after r rounds. Furthermore, let $DIF^{(r+1)}_S$ ($DIF^{(r+1)}_e$, respectively) denote the event that S (e, respectively) is in a different state in Algorithm 2 compared to Algorithm 3 after $r+1$ rounds. Assume that $\Pr\left[EQ^{(r)}_{S,10}\right] \geq \frac{1}{2}$ and $\Pr\left[EQ^{(r)}_{e,10}\right] \geq \frac{1}{2}$. Then,*

$$\Pr\left[DIF^{(r+1)}_S \mid EQ^{(r)}_{S,10}\right] = O\left(\frac{1}{(st)^{\log^3 s\cdot\log^3 t}}\right), \textit{ and}$$

$$\Pr\left[DIF^{(r+1)}_e \mid EQ^{(r)}_{e,10}\right] = O\left(\frac{1}{(st)^{\log^3 s\cdot\log^3 t}}\right).$$

Proof. We provide the proof for S, but the same argument works for e. Assume that the 10-hop neighborhood of S is in the same state after r rounds in both algorithms. In order for S to be in a different state after $r+1$ rounds, either

(A) there exists a set S' in the 5-hop neighborhood of S such that S' is added to the set cover in Line 9 of Algorithm 3 during round $r+1$; or

(B) there exists an element e' in the 5-hop neighborhood of S such that e' pretends to be covered in Line 14 of Algorithm 3 during round $r+1$.

We consider these two cases independently.

Case A. Let i be the stage that r is part of. This implies that $B_i(S')$ contains more than $\log^{20} s \cdot \log^{20} t$ many free elements at the beginning of stage i in Algorithm 3. As the 5-hop neighborhood of S' is contained in the 10-hop neighborhood of S, this implies that $B_i(S')$ also contains more than $\log^{20} s \cdot \log^{20} t$ many elements in Algorithm 2. This implies that S' is a bad set in round r.

Case B. Observe that an element can start pretending to be covered from round $r+1$ onwards only if $r+1$ corresponds to the first iteration of some phase (i,j). Hence, assume that $r+1$ corresponds to the first iteration of phase j in stage i of Algorithm 3 and to iteration $k_1 = j \cdot T + 1$ in stage i of Algorithm 2. As e' pretends to be covered, e' was still uncovered at the beginning of iteration k_1. Therefore, there exists $k_2 \in \{j \cdot T + 1, \ldots, (j+1) \cdot T\}$ such that $\mathcal{S}_{i,k_2}$ had more than $\log^{20} s \cdot \log^{20} t \cdot 2^T \geq \log^{20} s \cdot \log^{20} t \cdot 2^{k_2 - k_1}$ many sets that contain e' with an estimated set size of at least $s/2^i$ at the beginning of iteration k_1 in stage i (see Line 13 of Algorithm 3). This implies that e' is a bad element in Algorithm 2.

Combining the two cases. Both cases imply that S has a bad neighborhood in Algorithm 2. Therefore, from Lemma 4.10 we conclude

$$\begin{aligned}
&\Pr\left[DIF_S^{(r+1)} \mid EQ_{S,10}^{(r)}\right] \\
&\leq \Pr\left[\text{“}S\text{ has a bad neighborhood”} \mid EQ_{S,10}^{(r)}\right] \\
&\leq \frac{\Pr\left[\text{“}S\text{ has a bad neighborhood”}\right]}{\Pr\left[EQ_{S,10}^{(r)}\right]} \\
&\leq 2 \cdot \Pr\left[\text{“}S\text{ has a bad neighborhood”}\right] \\
&\leq 2 \cdot O\left(\frac{1}{(st)^{\log^3 s \cdot \log^3 t}}\right).
\end{aligned}$$

In the same way we derive $\Pr\left[DIF_e^{(r+1)} \mid EQ_{e,10}^{(r)}\right] = O\left(\frac{1}{(st)^{\log^3 s \cdot \log^3 t}}\right)$. □

Lemma 5.2. *Consider a round $r \in \{0, \ldots, \log s \cdot \log t\}$. Define $T_r := 10 \cdot (\log s \cdot \log t + 1 - r)$. Let p_r be equal to the maximum probability, over all sets S and all elements e, that the T_r-hop neighborhood of S or e looks different in Algorithm 2 compared to Algorithm 3 after r rounds. Then, it holds*

$$p_r \leq r \cdot c \cdot \frac{1}{(st)^{\log s \cdot \log t}}, \tag{5.2}$$

for an absolute constant c.

Proof. We proof the lemma by induction over r. The base case $r = 0$ holds trivially. Let $r \in \{0, \ldots, \log s \cdot \log t - 1\}$ be an arbitrary round. Assuming that Eq. (5.2) holds for r, we will show that Eq. (5.2) holds for $r+1$ as well.

Let S be a set. Assume that the T_{r+1}-hop neighborhood of S looks different in Algorithm 2 compared to Algorithm 3 after $r+1$ rounds. This could happen for one of the two following reasons. First, the T_r-hop neighborhood of S looked different after r rounds in Algorithm 2 compared to Algorithm 3, which happens with a probability of at most p_r. Second, there exists a set S' (an element e', respectively) in the T_{r+1}-hop neighborhood of S such that the 10-hop neighborhood of S' (e', respectively) looks the same after r rounds, but S' (e', respectively) is not in the same state after $r+1$ rounds in the two different algorithms. Conditioned that the first case does not happen, Lemma 5.1 together with a union bound over at most $(st)^{O(\log s \cdot \log t)}$ many sets and elements in the T_{r+1}-hop neighborhood of S implies that this case happens with a probability of at most $c \cdot \frac{1}{(st)^{\log s \cdot \log t}}$, where c is an absolute constant. This implies that

$$\begin{aligned}
p_{r+1} &\leq p_r + c \cdot \frac{1}{(st)^{\log s \cdot \log t}} \\
&\leq r \cdot c \cdot \frac{1}{(st)^{\log s \cdot \log t}} + c \cdot \frac{1}{(st)^{\log s \cdot \log t}} \\
&= (r+1) \cdot c \cdot \frac{1}{(st)^{\log s \cdot \log t}},
\end{aligned}$$

as desired. □

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

Proof of. Theorem 5.1. Let p_S (p'_S, respectively) be the probability that S is added to the set cover constructed by Algorithm 2 (Algorithm 3, respectively). Lemma 5.2 implies that the $T_{\log(s) \cdot \log(t)}$-hop neighborhood of S is in a different state in Algorithm 3 compared to Algorithm 2 with a probability at most $p_{\log s \log t} \leq \frac{1}{s \cdot t}$. In particular, this implies that $|p_S - p'_S| \leq \frac{1}{s \cdot t}$. Together with Theorem 4.1, this implies that the expected size of $\mathcal{S}_{\mathsf{cover}}$ is at most $O(\log s) \cdot \mathrm{OPT} + \frac{n \cdot t}{t \cdot s}$. This implies $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] = O(\log s) \cdot \mathrm{OPT}$ as desired. □

6 Repeated Sparsification

We now discuss how to reduce the query complexity from $(st)^{O(\log s \sqrt{\log t})}$ to $s^{O(\log s)} t^{O(\log s \cdot (\log \log s + \log \log t))}$, and hence prove Theorem 1.1. We will first present the high-level ideas of our approach and convey the main intuition behind our algorithm, that we present as Algorithms 4 to 6.

Estimating set sizes will be still done in the same way as in Algorithm 2. Thus, our main focus lies on improving the sparsification of the element side. For the sake of cleaner presentation, we first review how sparsification works in Algorithm 3. Let i be a stage. Consider the moment when Algorithm 3 tests whether an element e is covered or not in the iteration $\log t$ of stage i. First, note that the family $\mathcal{S}_{i,\log t}$ contains all the sets in the set cover instance. Hence, e might be contained in up to t sets in $\mathcal{S}_{i,\log t}$. However, recall that we sparsify $\mathcal{S}_{i,\log t}$ at the beginning of the last phase by removing all the sets which have a small estimated number of free elements. The resulting family of sets is called $\hat{\mathcal{S}}_{i,\log t}$ and e only needs to check for all sets in $\hat{\mathcal{S}}_{i,\log t}$ containing e whether they get added to cover in iteration $\log t$ of stage i.

However, to obtain the relevant sets in $\hat{\mathcal{S}}_{i,\log t}$, we need to estimate the degree of t sets at the beginning of the last phase. This can be query-wise wasteful, in a sense that invoking oracles that answer queries about the last iteration/phase is significantly more expensive than invoking oracles which answer queries about iterations that happen early in the computation. Motivated by this observation, our goal now is to sparsify $\mathcal{S}_{i,\log t}$ rather "earlier" than "later". We achieve that in multiple steps. More concretely, we first estimate the degree of all the sets that contain e in $\mathcal{S}_{i,\log t}$ after $\log(t)/2$ iterations. Given that e is still uncovered after iteration $\log(t)/2$, we expect no more than $\sqrt{t}$ sets to have a large degree. For all of the at most $\sqrt{t}$ sets, we now estimate the degree after $\frac{3}{4}\log t$ iterations. If e is still uncovered after iteration $\frac{3}{4}\log t$, we expect no more than $t^{1/4}$ of those sets to have a large degree. This process is repeated until we expect no more than $\mathrm{poly}(\log t)$ of the sets to have a large degree. At that point, e is contained in a small number of sets that have to be tested, and we simply test for each of those sets whether they get added to the set cover during iteration $\log(t)$ of stage i or not.

Now we describe the structure of Algorithm 4. In order to execute stage i, Algorithm 4 calls the recursive procedure Algorithm 5. The top-level call of Algorithm 5 executes $\log(t)$ iterations. As the input, Algorithm 5 receives the families of sets $\hat{\mathcal{S}}_1, \dots, \hat{\mathcal{S}}_{\log t}$, which contain all the sets that will potentially be added to the set cover in iterations 1 to $\log t$. Given that, Algorithm 5 calls itself recursively to execute the first $\log(t)/2$ iterations. It passes down the families of sets $\hat{\mathcal{S}}_1, \dots, \hat{\mathcal{S}}_{\log(t)/2}$ that contain all the sets that potentially get added to the set cover between iterations 1 and $\log(t)/2$. Note that we expect e to be contained in no more than $\sqrt{t}$ sets of $\hat{\mathcal{S}}_{\log(t)/2}$. After executing the first $\log(t)/2$ iterations, we recursively call Algorithm 5 to execute the remaining $\log(t)/2$ iterations. However, we do not simply pass down the recursion families of sets $\hat{\mathcal{S}}_{\log(t)/2+1}, \dots, \hat{\mathcal{S}}_{\log(t)}$. Instead, we first sparsify the families of sets by removing all the sets with an estimated degree less than $s/2^i$. Then, the resulting families of sets $\mathcal{S}'_{\log(t)/2+1}, \dots, \mathcal{S}'_{\log(t)}$ get passed down to the second recursive call. Furthermore, for a free element e, we do not expect e to be contained in more than $\sqrt{t}$ sets in any of the families of sets that get passed to the second recursive call. If we detect any such element e', as before, we mark e' as pretending to be covered.

Algorithm 4: A variant of Algorithm 2 that can be simulated by an LCA with $s^{O(\log s)} t^{O(\log s \cdot (\log\log s + \log\log t))}$ queries.

```
1  for each pair of $(i,k)$ where $i \in [\log s]$ and $k \in [\log t]$ do
2      $B_i(S)$ and $\mathcal{S}_{i,k}$ are generated in the exact same way as in Algorithm 2
3  for stage $i$ = 1 to $\log s$ do
4      for each set $S \in \mathcal{S}$ do
5          $\hat{B}_i(S) \leftarrow B_i(S) \cap \mathcal{E}_i$ // $\mathcal{E}_i$ denotes the set of free elements in the beginning of the stage
6          if $|\hat{B}_i(S)| \geq \log^{20} s \cdot \log^{20} t$ then // test whether $S$ is a bad set
7              add $S$ to $\mathcal{S}_{\text{cover}}$.
8      for each $k \in [\log(t)]$ do
9          $\hat{\mathcal{S}}_{i,k} \leftarrow \{S | S \in \mathcal{S}_{i,k}, \hat{d}_{B_i}(S) \geq s/2^i\}$
10         for each $e \in \mathcal{E}_i$ in parallel do
11             if $d_{\hat{\mathcal{S}}_{i,k}}(e) \geq \log^{20} s \cdot \log^{20} t \cdot 2^T$ then // test whether $e$ is a bad element
12                 pretend that $e$ is covered.// $e$ will be covered in Line 15.
13     $\hat{\mathcal{E}}_i \leftarrow$ all elements in $\mathcal{E}_i$ which are uncovered (and don't pretend to be covered)
       RecursiveSplitting$(1, \log t, i, (\hat{\mathcal{S}}_{i,1}, \hat{\mathcal{S}}_{i,2}, ..., \hat{\mathcal{S}}_{i,\log t}), \hat{\mathcal{E}}_i)$
14 for each free element $e$ do // Handling bad elements
15     add the set with smallest ID which contains $e$ to $\mathcal{S}_{\text{cover}}$.
```

6.1 Establishing an Invariant

Invariant 6.1. *We define the following invariant for* RecursiveSplitting$(k', R, i, \hat{\mathcal{S}}_{[k',k'+R-1]}, \hat{\mathcal{E}}_i)$*:*

(1) For $l \in \{0, ..., R-1\}$, each element e in $\hat{\mathcal{E}}_i$ is contained in at most $\log^{20} s \cdot \log^{20} t \cdot 2^{l+1}$ many sets of $\hat{\mathcal{S}}_{k'+l}$.

This invariant directly follows from Line 12 of Algorithm 4 and Line 9 of Algorithm 5 together with a simple induction argument. In particular, Invariant 6.1 implies that each uncovered element is contained in at most $\mathrm{poly}(\log s) \cdot 2^{O(R)}$ many sets in $\hat{\mathcal{S}}_k$ for any $k \in \{k', ..., k' + R - 1\}$.

Algorithm 5: RecursiveSplitting($k', R, i, \hat{\mathcal{S}}_{[k',k'+R-1]} = (\hat{\mathcal{S}}_{k'}, \hat{\mathcal{S}}_{k'+1}, \dots, \hat{\mathcal{S}}_{k'+R-1}), \hat{\mathcal{E}}_i$)

```
Input:
   Simulates iterations $k'$ to $k' + R - 1$ of the $i$-th iteration
   For $k \in \{k', ..., k' + R - 1\}$, $\hat{\mathcal{S}}_k$ consists of sets that will be added to the set cover as long as the estimated number of free elements is at least $s/2^i$ in the beginning of the $k$-th iteration of stage $i$.
   $\hat{\mathcal{E}}_i$ consists of all elements which are uncovered (and don't pretend to be covered) at the beginning of iteration $k'$
1  if $R \le \log\log t$ then
2      BaseCase($k', R, i, (\hat{\mathcal{S}}_{k'}, \dots, \hat{\mathcal{S}}_{k'+R-1})$)
3      return
4  RecursiveSplitting($k', R/2, i, (\hat{\mathcal{S}}_{k'}, \hat{\mathcal{S}}_{k'+1}, \dots, \hat{\mathcal{S}}_{k'+R/2-1}), \hat{\mathcal{E}}_i$)
5  for each $k \in \{k' + R/2, \dots, k' + R - 1\}$ do
6      $\mathcal{S}'_k \leftarrow \{S | S \in \hat{\mathcal{S}}_k, \hat{d}_{B_i}(S) \ge s/2^i\}$
7      for each uncovered element $e$ in parallel do
8          if $d_{\mathcal{S}'_k}(e) \ge \log^{20} s \cdot \log^{20} t \cdot 2^{k-(k'+R/2)}$ then // only happens if $e$ is a bad element
9              pretend that $e$ is covered.
10 $\mathcal{E}'_i \leftarrow$ all uncovered elements in $\hat{\mathcal{E}}_i$
11 RecursiveSplitting($k' + R/2, R/2, i, (\mathcal{S}'_{k'+R/2}, \mathcal{S}'_{k'+R/2+1}, \dots, \mathcal{S}'_{k'+R-1}), \mathcal{E}'_i$)
```

6.2 LCA Simulation of Algorithm 5 and Algorithm 6

In this section we describe the LCA implementation of Algorithm 5 and Algorithm 6. To simplify notation, we sometimes treat $\hat{\mathcal{S}}_{[k',k'+R-1]}$ as a set consisting of all sets S which are contained in $\hat{\mathcal{S}}_k$ for $k \in \{k', ..., k' + R - 1\}$.

For the sake of analysis, we define two oracles $\mathcal{O}^{\mathsf{set}}_{k',R,i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}(e)$. $\mathcal{O}^{\mathsf{set}}_{k',R,i}$ takes as an input a set S and returns whether S gets added to the set cover during the execution of RecursiveSplitting($k', R, i, \hat{\mathcal{S}}_{[k',k'+R-1]}, \hat{\mathcal{E}}_i$). Similarly, $\mathcal{O}^{\mathsf{el}}_{k',R,i}$ takes as an input an arbitrary element e and returns whether e gets covered for the first time (or pretends to be covered) during the execution of RecursiveSplitting($k', R, i, \hat{\mathcal{S}}_{[k',k'+R-1]}, \hat{\mathcal{E}}_i$). In the following two paragraphs, we show how to implement the two oracles by having query access to the following type of queries for some arbitrary element e or some arbitrary set S:

(1) For some $k \in \{k', \dots, k' + R - 1\}$, is S contained in $\hat{\mathcal{S}}_k$?

(2) For some $k \in \{k', \dots, k' + R - 1\}$ and some $e \in \hat{\mathcal{E}}_i$, give me all sets in $\hat{\mathcal{S}}_k$ which contain e.

(3) Is e contained in $\hat{\mathcal{E}}_i$?

(4) Give me all elements contained in $\hat{B}_i(S)$.

We then derive a bound for the number of query accesses the algorithm needs.

Base Case: Description of $\mathcal{O}^{\mathsf{set}}_{k',R,i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}(e)$ ($R \le \log\log t$) As $R \le \log\log t$, we only need to focus on Algorithm 6. In the following, we assume that $S \in \hat{\mathcal{S}}_{[k',k'+R-1]}$ and $e \in \hat{\mathcal{E}}_i$, as these two cases are easy to check and $S \notin \hat{\mathcal{S}}_{[k',k'+R-1]}$ implies that S won't be added to the set cover and $e \notin \hat{\mathcal{E}}_i$ implies that e was already previously covered (or pretended to be covered). Now, consider the directed bipartite graph $G = (\hat{\mathcal{S}}_{[k',k'+R-1]} \cup \hat{\mathcal{E}}_i, E_1 \cup E_2)$ with $(S, e) \in E_1$ iff $e \in \hat{B}_i(S)$ and $(e, S) \in E_2$ iff $e \in \hat{\mathcal{S}}_{[k',k'+R-1]}$. Note that in order to decide for some set $S \in \hat{\mathcal{S}}_{[k',k'+R-1]}$ if it gets added to the set cover, or some element $e \in \hat{\mathcal{E}}_i$ whether e gets covered, it is sufficient to gather all the information within the $O(\log\log t)$-hop neighborhood of S or e, respectively. Note that the maximal out-degree of some set in G is $\log^{20} s \cdot \log^{20} t$ and Invariant 6.1 ensures that the maximal out-degree of an element is $\mathrm{poly}(\log s) \cdot 2^{O(R)}$. Thus, the maximal out-degree of G is $O(\mathrm{poly}(\log s \cdot \log t))$. We can therefore gather all the relevant information with $O(\log\log t) \cdot (\log s \cdot \log t)^{O(\log\log t)}$ many queries. This implies that we can answer $\mathcal{O}^{\mathsf{set}}_{k',R,i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}(e)$ with $(\log s \cdot \log t)^{O(\log\log t)}$ many queries.

Algorithm 6: BaseCase($k', R, i, (\hat{\mathcal{S}}_{k'}, \dots, \hat{\mathcal{S}}_{k'+R-1})$)

```
1  for iteration $k = k'$ to $k' + R - 1$ do
2      for all sets $S$ in parallel do
3          if $S \in \hat{\mathcal{S}}_k$ is not in the cover and $\hat{d}_{B_i}(S) \ge s/2^i$ then // $B_i$ is computed in Line 5
4              Add $S$ to the set cover
```

Recursive Case: Description of $\mathcal{O}^{\mathsf{set}}_{k',R,i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}(e)$ ($R > \log\log t$) Let $Q(R)$ denote the query complexity to answer $\mathcal{O}^{\mathsf{set}}_{k',R,i}(S)$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}(e)$, respectively. For $R > \log\log t$, we will establish that

$$Q(R) \leq \operatorname{poly}(\log S) \cdot 2^{O(R)} \cdot Q(R/2)^2$$

For some arbitrary set S, we first check if S gets added to the set cover during the first recursive call by invoking $\mathcal{O}^{\mathsf{set}}_{k',R/2,i}(S)$. Note that $\mathcal{O}^{\mathsf{set}}_{k',R/2,i}(S)$ assumes query access to 4 different types of queries. However, those queries are basically the same queries (slightly more restricted) as the ones we assume to have to implement oracles $\mathcal{O}^{\mathsf{set}}_{k',R,i}$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}$. Hence, we can answer $\mathcal{O}^{\mathsf{set}}_{k',R/2,i}(S)$ with $Q(R/2)$ many queries. Similarly, we can decide with $Q(R/2)$ many queries if e gets covered during the first recursive call.

Note that the oracles corresponding to the second recursive call assume to have access to the following queries:

(1) For some $k \in \{k'+R/2, ..., k'+R-1\}$, is S contained in $\mathcal{S}'_k$?

(2) For some $k \in \{k'+R/2, ..., k'+R-1\}$ and some $e \in \mathcal{E}'_i$, give me all sets in $\mathcal{S}'_k$ which contain e.

(3) Is e contained in $\mathcal{E}'_i$?

(4) Give me all elements contained in $\hat{B}_i(S)$.

To decide if S is contained in S'_k, we first check if S is contained in $\hat{\mathcal{S}}_k$. If yes, then we check for each element e in $\hat{B}_i(S)$ if e is still uncovered after the first recursive call. Given this information, we can decide if S is contained in S'_k. Hence, it takes $O(\operatorname{poly}(\log s)\operatorname{poly}(\log t)) \cdot Q(R/2)$ many queries to decide whether S is contained in S'_k.

In order to return all the sets in S'_k which contain e (We only assume $e \in \hat{\mathcal{E}}_i$ instead of $e \in \mathcal{E}'_i$), we first get all of the at most $\operatorname{poly}(\log s) \cdot 2^{O(R)}$ many sets in $\hat{\mathcal{S}}_k$ which contain e. This takes one query. Then, we check for each of them whether it is also contained in S'_k. Hence, we can answer this query with $\operatorname{poly}(\log s) \cdot 2^{O(R)} \cdot Q(R/2)$ many queries.

To decide if $e \in \mathcal{E}'_i$, we first check if $e \in \hat{\mathcal{E}}_i$. If yes, then we check if e gets covered during the first recursive call. If not, then we check if e pretends to be covered by checking for each of the at most $\operatorname{poly}(\log s) \cdot 2^{O(R)}$ many sets in $\hat{\mathcal{S}}_{[k',k'+R-1]}$ which contain e whether they get added to one or more of the S'_k. Thus, answering if $e \in \mathcal{E}'_i$ takes $\operatorname{poly}(\log s) \cdot 2^{O(R)} \cdot O(\operatorname{poly}(\log s)\operatorname{poly}(\log t)) \cdot Q(R/2)$ many queries.

The last query does not change across different recursive calls. Hence, we can answer each of those 4 queries with at most $\operatorname{poly}(\log s) \cdot 2^{O(R)}$ many queries. Hence, we need $\operatorname{poly}(\log s) \cdot 2^{O(R)} \cdot Q(R/2)^2$ many queries to decide if a set is added to the set cover during the second recursive call or some element gets covered during the second recursive call. Thus, we can answer the oracles $\mathcal{O}^{\mathsf{set}}_{k',R,i}$ and $\mathcal{O}^{\mathsf{el}}_{k',R,i}$ with $\operatorname{poly}(\log s) \cdot 2^{O(R)} \cdot Q(R/2)^2$ many queries.

6.2.1 Bounding the Recursion

By the previous two subsections, there exist two constants c_1 and c_2 such that we can upper bound the query complexity as follows

$$(6.3) \quad Q(R) \leq \begin{cases} (\log s \log t)^{c_1 \cdot \log\log t}, & R \leq \log\log t \\ (\log s \cdot 2^R)^{c_2} Q(R/2)^2, & R > \log\log t \end{cases}$$

We will now show by induction that there exists some constant c such that for $R \geq \frac{1}{2} \cdot \log\log t$, we have:

$$Q(R) \leq \left((\log s) \cdot 2^R\right)^{c \cdot (R-1)}$$

Note that this implies $Q(\log t) \leq (\log s)^{O(\log t)} \cdot t^{O(\log\log t)} = t^{O(\log\log t + \log\log s)}$. For c large enough, it is easy to see that it holds for $\frac{1}{2}\log\log t \leq R \leq \log\log t$. Now, consider some $R > \log\log(t)$. We get

$$\begin{aligned} Q(R) &\leq (\log s \cdot 2^R)^{c_2} \cdot (Q(R/2))^2 \\ &\leq (\log s \cdot 2^R)^{c_2} \cdot \left(\left(\log s \cdot 2^{R/2}\right)^{c \cdot (R/2-1)}\right)^2 \\ &\leq (\log s \cdot 2^R)^{c_2} \cdot \left(\log s \cdot 2^{R/2}\right)^{c \cdot R - 2c} \\ &\leq \left(\log s \cdot 2^R\right)^{c \cdot (R-1)} \end{aligned}$$

as desired.

6.3 LCA implementation and query complexity of Algorithm 4

For some stage i, we need to answer $Q(\log(t))$ many of the following queries:

(1) For some $k \in \{1, ..., \log t\}$, is S contained in $\hat{\mathcal{S}}_{i,k}$?

(2) For some $k \in \{1, ..., \log t\}$ and some $e \in \mathcal{E}_i$, give me all sets in $\hat{\mathcal{S}}_{i,k}$ which contain e.

(3) Is e contained in $\hat{\mathcal{E}}_i$?

(4) Give me all elements contained in $\hat{B}_i(S)$.

Each one of those can be answered with $O(st)$ many queries to $(\mathcal{S}_i, \mathcal{E}_i)$ and the original set cover instance. Given that, we can "glue" different stages together in the same way as in Algorithm 3. Thus, the query complexity of Algorithm 4 is $(\operatorname{poly}(st) \cdot Q(\log t))^{\log s}$. Plugging in $t^{O(\log t \log t + \log\log s)}$ for $Q(\log t)$ leads to a query complexity of $t^{O(\log s(\log\log s + \log\log t))} \cdot s^{O(\log s)} = s^{O(\log t(\log\log s + \log\log t) + \log s)}$.

6.4 Approximation Proof

THEOREM 6.1. *Let $\mathcal{S}_{\mathsf{cover}}$ be the solution returned by Algorithm 4. Then,* $\mathbb{E}\left[|\mathcal{S}_{\mathsf{cover}}|\right] = O(\log s) \cdot \mathrm{OPT}$.

Proof. The proof is completely analogous to the proof of Theorem 5.1. The only thing to show is the following: For some set S (element e), given that the 10-hop neighborhood of S looks the same after r rounds in both algorithms, S can only be in a different state after round $r+1$ in Algorithm 2 compared to Algorithm 4 if S has a bad neighborhood in Algorithm 2. This can be seen as follows: S can only be in a different state after $r+1$ rounds if either some element e' in the 5-hop neighborhood of S pretends to be covered during round $r+1$ or some set S' in the 5-hop neighborhood of S gets added to the set cover in Line 7 of Algorithm 4. This would however imply that S has a bad neighborhood in Algorithm 2 as desired. □

Acknowledgment

The authors would like to thank Mohsen Ghaffari for his helpful comments in the various stages of this project. S. Mitrović was supported by the Swiss NSF grant P2ELP2_181772, and MIT-IBM Watson AI Lab and Research Collaboration Agreement No. W1771646. R. Rubinfeld was supported by the MIT-IBM Watson AI Lab and Research Collaboration Agreement No. W1771646 and the NSF awards CCF-1740751, CCF-1733808, and IIS-1741137. A. Vakilian was supported by the NSF grant CCF-1535851.

References

[1] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algo.*, 2(2):153–177, 2006.

[2] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1132–1139. Society for Industrial and Applied Mathematics, 2012.

[3] S. Assadi. Tight Space-Approximation Tradeoff for the Multi-Pass Streaming Set Cover Problem. In *Proc. 36th ACM Sympos. on Principles of Database Systems* (PODS), pages 321–335, 2017.

[4] S. Assadi, S. Khanna, and Y. Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *Proc. 48th Annu. ACM Sympos. Theory Comput.* (STOC), pages 698–711, 2016.

[5] N. Bansal, A. Caprara, and M. Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM Journal on Computing*, 39(4):1256–1278, 2009.

[6] M. Bateni, H. Esfandiari, and V. S. Mirrokni. Almost optimal streaming algorithms for coverage problems. In *Proc. 29th ACM Sympos. Parallel Alg. Arch.* (SPAA), pages 13–23, 2017.

[7] B. Berger, J. Rompel, and P. W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. *Journal of Computer and System Sciences*, 49(3):454–477, 1994.

[8] G. E. Blelloch, R. Peng, and K. Tangwongsan. Linear-work greedy parallel approximate set cover and variants. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 23–32. ACM, 2011.

[9] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan. Parallel and i/o efficient set covering algorithms. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 82–90. ACM, 2012.

[10] A. Chakrabarti and A. Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *Proc. 27th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1365–1373, 2016.

[11] Y.-J. Chang, M. Fischer, M. Ghaffari, J. Uitto, and Y. Zheng. The complexity of (δ+ 1) coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 471–480, 2019.

[12] G. Cormode, H. J. Karloff, and A. Wirth. Set cover algorithms for very large datasets. In *Proc. 19th ACM Conf. Info. Know. Manag.* (CIKM), pages 479–488, 2010.

[13] E. D. Demaine, P. Indyk, S. Mahabadi, and A. Vakilian. On streaming and communication complexity of the set cover problem. In *Proc. 28th Int. Symp. Dist. Comp.* (DISC), volume 8784, pages 484–498, 2014.

[14] I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *Proc. 46th Annu. ACM Sympos. Theory Comput.* (STOC), pages 624–633, 2014.

[15] Y. Emek and A. Rosén. Semi-streaming set cover. In *Proc. 41st Int. Colloq. Automata Lang. Prog.* (ICALP), volume 8572 of *Lect. Notes in Comp. Sci.*, pages 453–464, 2014.

[16] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[17] M. Ghaffari and J. Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1636–1653, 2019.

[18] M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.

[19] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering

problem. *Euro. J. Oper. Res.*, 101(1):81–92, 1997.

[20] S. Har-Peled, P. Indyk, S. Mahabadi, and A. Vakilian. Towards tight bounds for the streaming set cover problem. In *Proc. 35th ACM Sympos. on Principles of Database Systems* (PODS), pages 371–383, 2016.

[21] P. Indyk, S. Mahabadi, R. Rubinfeld, J. Ullman, A. Vakilian, and A. Yodpinyanee. Fractional set cover in the streaming model. In *20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problem (APPROX 2017)*, pages 198–217, 2017.

[22] P. Indyk, S. Mahabadi, R. Rubinfeld, A. Vakilian, and A. Yodpinyanee. Set cover in sub-linear time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2467–2486, 2018.

[23] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[24] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory.* MIT press, 1994.

[25] C. Koufogiannakis and N. E. Young. A nearly linear-time PTAS for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, 2014.

[26] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 980–989. Society for Industrial and Applied Mathematics, 2006.

[27] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. In *Proc. 25th ACM Sympos. Parallel Alg. Arch.* (SPAA), pages 1–10, 2013.

[28] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

[29] D. Moshkovitz. The projection games conjecture and the NP-hardness of $\ln n$-approximating set-cover. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 276–287. 2012.

[30] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336. IEEE, 2008.

[31] K. Onak. Round compression for parallel graph algorithms in strongly sublinear space. *arXiv preprint arXiv:1807.08745*, 2018.

[32] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

[33] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th Annu. ACM Sympos. Theory Comput.* (STOC), 1997.

[34] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. *arXiv preprint arXiv:1104.1377*, 2011.

[35] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proc. SIAM Int. Conf. Data Mining* (SDM), pages 697–708, 2009.

[36] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[37] S. P. Vadhan et al. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[38] Y. Yoshida, M. Yamamoto, and H. Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing*, 41(4):1074–1093, 2012.