Set Cover in Sub-linear Time

Set Cover in Sub-linear Time*

Piotr Indyk † Sepideh Mahabadi † Ronitt Rubinfeld ‡ Ali Vakilian †
Anak Yodpinyanee †

Abstract

We study the classic set cover problem from the perspective of sub-linear algorithms. Given access to a collection of m sets over n elements in the query model, we show that sub-linear algorithms derived from existing techniques have almost tight query complexities.

On one hand, first we show an adaptation of the streaming algorithm presented in [17] to the sub-linear query model, that returns an α-approximate cover using $\widetilde{O}(m(n/k)^{1/(\alpha-1)} + nk)$ queries to the input, where k denotes the value of a minimum set cover. We then complement this upper bound by proving that for lower values of k, the required number of queries is $\widetilde{\Omega}(m(n/k)^{1/(2\alpha)})$, even for estimating the optimal cover size. Moreover, we prove that even checking whether a given collection of sets covers all the elements would require $\Omega(nk)$ queries. These two lower bounds provide strong evidence that the upper bound is almost tight for certain values of the parameter k.

On the other hand, we show that this bound is not optimal for larger values of the parameter k, as there exists a $(1+\varepsilon)$-approximation algorithm with $\widetilde{O}(mn/k\varepsilon^2)$ queries. We show that this bound is essentially tight for sufficiently small constant ε, by establishing a lower bound of $\widetilde{\Omega}(mn/k)$ query complexity.

Our lower-bound results follow by carefully designing two distributions of instances that are hard to distinguish. In particular, our first lower bound involves a probabilistic construction of a certain set system with a minimum set cover of size αk, with the key property that a small number of "almost uniformly distributed" modifications can reduce the minimum set cover size down to k. Thus, these modifications are not detectable unless a large number of queries are asked. We believe that our probabilistic construction technique might find applications to lower bounds for other combinatorial optimization problems.

*This work was supported by the NSF grants, including No. CCF-1650733, CCF-1733808, CCF-1420692, IIS-1741137, and the Simons Investigator award.

†CSAIL, MIT, {indyk, mahabadi, vakilian, anak}@mit.edu

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

1 Introduction

Set Cover is a classic combinatorial optimization problem, in which we are given a set (universe) of n elements $\mathcal{U} = \{e_1, \cdots, e_n\}$ and a collection of m sets $\mathcal{F} = \{S_1, \cdots, S_m\}$. The goal is to find a *set cover* of $\mathcal{U}$, i.e., a collection of sets in $\mathcal{F}$ whose union is $\mathcal{U}$, of minimum size. Set Cover is a well-studied problem with applications in operations research [16], information retrieval and data mining [32], learning theory [19], web host analysis [9], and many others. Recently, this problem and other related coverage problems have gained a lot of attention in the context of massive data sets, e.g., streaming model [32, 12, 10, 17, 7, 3, 24, 2, 5, 18] or map reduce model [22, 25, 4].

Although the problem of finding an optimal solution is **NP**-complete, a natural greedy algorithm which iteratively picks the "best" remaining set (the set that covers the most number of uncovered elements) is widely used. The algorithm finds a solution of size at most $k \ln n$ where k is the optimum cover size, and can be implemented to run in time linear in the input size. However, the input size itself could be as large as $\Theta(mn)$, so for large data sets even reading the input might be infeasible.

This raises a natural question: *is it possible to solve minimum set cover in sub-linear time?* This question was previously addressed in [28, 33], who showed that one can design constant running-time algorithms by simulating the greedy algorithm, under the assumption that the sets are of constant size and each element occurs in a constant number of sets. However, those constant-time algorithms have a few drawbacks: they only provide a mixed multiplicative/additive guarantee (the output cover size is guaranteed to be at most $k \cdot \ln n + \epsilon n$), the dependence of their running times on the maximum set size is exponential, and they only output the (approximate) minimum set cover size, not the cover itself. From a different perspective, [20] (building on [15]) showed that an $O(1)$-approximate solution to the *fractional* version of the problem can be found in $\widetilde{O}(mk^2+nk^2)$ time[1]. Combining this algorithm with the

[1]The method can be further improved to $\widetilde{O}(m+nk)$ (N. Young,

randomized rounding yields an $O(\log n)$-approximate solution to Set Cover with the same complexity.

In this paper we initiate a systematic study of the complexity of sub-linear time algorithms for set cover with multiplicative approximation guarantees. Our upper bounds complement the aforementioned result of [20] by presenting algorithms which are fast when k is *large*, as well as algorithms that provide more accurate solutions (even with a constant-factor approximation guarantee) that use a sub-linear number of *queries*[2]. Equally importantly, we establish nearly matching lower bounds, some of which even hold for estimating the optimal cover size. Our algorithmic results and lower bounds are presented in Table 1.

Data access model. As in the prior work [28, 33] on Set Cover, our algorithms and lower bounds assume that the input can be accessed via the adjacency-list oracle.[3] More precisely, the algorithm has access to the following two oracles:

1. **EltOf:** Given a set S_i and an index j, the oracle returns the j^{th} element of S_i. If $j > |S_i|$, $\perp$ is returned.
2. **SetOf:** Given an element e_i and an index j, the oracle returns the j^{th} set containing e_i. If e_i appears in less than j sets, $\perp$ is returned.

This is a natural model, providing a "two-way" connection between the sets and the elements. Furthermore, for some graph problems modeled by Set Cover (such as Dominating Set or Vertex Cover), such oracles are essentially equivalent to the aforementioned incident-list model studied in sub-linear graph algorithms. We also note that the other popular access model employing the *membership oracle*, where we can query whether an element e is contained in a set S, is not suitable for Set Cover, as it can be easily seen that even checking whether a feasible cover exists requires $\Omega(mn)$ time.

1.1 Overview of our results. In this paper we present algorithms and lower bounds for the Set Cover problem. The results are summarized in Table 1. The **NP**-hardness of this problem (or even its $o(\log n)$-approximate version [13, 31, 1, 26, 11]) precludes the existence of highly accurate algorithms with fast running times, while (as we show) it is still possible to design algorithms with sub-linear query complexities and low approximation factors. The lower bound proofs hold for the running time of any algorithm approximation set cover assuming the defined data access model.

We present two algorithms with sub-linear number of queries. First, we show that the streaming algorithm presented in [17] can be adapted so that it returns an $O(\alpha)$-approximate cover using $\widetilde{O}(m(n/k)^{1/(\alpha-1)} + nk)$ queries, which could be *quadratically* smaller than mn. Second, we present a simple algorithm which is tailored to the case when the value of k is large. This algorithm computes an $O(\log n)$-approximate cover in $\widetilde{O}(mn/k)$ *time* (not just query complexity). Hence, by combining it with the algorithm of [20], we get an $O(\log n)$-approximation algorithm that runs in time $\widetilde{O}(m + n\sqrt{m})$.

We complement the first result by proving that for low values of k, the required number of queries is $\widetilde{\Omega}(m(n/k)^{1/(2\alpha)})$ even for estimating the size of the optimal cover. This shows that the first algorithm is essentially optimal for the values of k where the first term in the runtime bound dominates. Moreover, we prove that even the Cover Verification problem, which is checking whether a given collection of k sets covers all the elements, would require $\Omega(nk)$ queries. This provides strong evidence that the term nk in the first algorithm is unavoidable. Lastly, we complement the second algorithm, by showing a lower bound of $\widetilde{\Omega}(mn/k)$ if the approximation ratio is a small constant.

1.2 Related work. Sublinear algorithms for Set Cover under the oracle model have been previously studied as an estimation problem; the goal is only to approximate the size of the minimum set cover rather than constructing one. Nguyen and Onak [28] consider Set Cover under the oracle model we employ in this paper, in a specific setting where both the maximum cardinality of sets in $\mathcal{F}$, and the maximum number of occurrences of an element over all sets, are bounded by some constants s and t; this allows algorithms whose time and query complexities are constant, $(2^{(st)^4}/\varepsilon)^{O(2^s)}$, containing no dependency on n or m. They provide an algorithm for estimating the size of the minimum set cover when, unlike our work, allowing both $\ln s$ multiplicative and εn additive errors. Their result has been subsequently improved to $(st)^{O(s)}/\varepsilon^2$ by Yoshida et al. [33]. Additionally, the results of Kuhn et al. [21] on general packing/covering LPs in the distributed $\mathcal{LOCAL}$ model, together with the reduction method of Parnas and Ron [30], implies estimating set cover size to within a $O(\ln s)$-multiplicative factor (with εn additive error), can be performed in $(st)^{O(\log s \log t)}/\varepsilon^4$ time/query complexities.

Set Cover can also be considered as a generalization of the Vertex Cover problem. The estimation variant

personal communication).

[2]Note that polynomial *time* algorithm with sub-logarithmic approximation algorithms are unlikely to exist.

[3]In the context of graph problems, this model is also known as the *incidence-list model*, and has been studied extensively, see e.g., [8, 14, 6].

Problem	Approximation	Constraints	Query Complexity	Section
Set Cover	$\alpha\rho+\varepsilon$	$\alpha \geq 2$	$\widetilde{O}(\frac{1}{\varepsilon}(m(\frac{n}{k})^{\frac{1}{\alpha-1}}+nk))$	4.2
	$\rho+\varepsilon$	-	$\widetilde{O}(\frac{mn}{k\varepsilon^2})$	4.3
	α	$k<(\frac{n}{\log m})^{\frac{1}{4\alpha+1}}$	$\widetilde{\Omega}(m(\frac{n}{k})^{1/(2\alpha)})$	A
	α	$\alpha \leq 1.01$ $k=O(\frac{n}{\log m})$	$\widetilde{\Omega}(\frac{mn}{k})$	3.2
Cover Verification	-	$k \leq n/2$	$\Omega(nk)$	5

Table 1: A summary of our algorithms and lower bounds. We use the following notation: $k \geq 1$ denotes the size of the optimum cover; $\alpha \geq 1$ denotes a parameter that determines the trade-off between the approximation quality and query/time complexities; $\rho \geq 1$ denotes the approximation factor of a "black box" algorithm for set cover used as a subroutine; We assume that $\alpha \leq \log n$ and $m \geq n$.

of Vertex Cover under the adjacency-list oracle model has been studied in [30, 23, 29, 33]. Set Cover has been also studied in the sublinear *space* context, most notably for the streaming model of computation [32, 12, 7, 3, 2, 5, 18, 10, 17]. In this model, there are algorithms that compute approximate set covers with only multiplicative errors. Our algorithms use some of the ideas introduced in the last two papers [10, 17].

1.3 Overview of the Algorithms. The algorithmic results presented in Section 4, use the techniques introduced for the streaming Set Cover problem by [10, 17] to get new results in the context of sub-linear time algorithms for this problem. Two components previously used for the set cover problem in the context of streaming are Set Sampling and Element Sampling. Assuming the size of the minimum set cover is k, Set Sampling randomly samples $\widetilde{O}(k)$ sets and adds them to the maintained solution. This ensures that all the elements that are well represented in the input (i.e., appearing in at least m/k sets) are covered by the sampled sets. On the other hand, the Element Sampling technique samples roughly $\widetilde{O}(k/\delta)$ elements, and finds a set cover for the sampled elements. It can be shown that the cover for the sampled elements covers a $(1-\delta)$ fraction of the original elements.

Specifically, the first algorithm performs a constant number of iterations. Each iteration uses element sampling to compute a "partial" cover, removes the elements covered by the sets selected so far and recurses on the remaining elements. However, making this process work in sub-linear time (as opposed to sub-linear space) requires new technical development. For example, the algorithm of [17] relies on the ability to test membership for a set-element pair, which generally cannot be efficiently performed in our model.

The second algorithm performs only one round of set sampling, and then identifies the elements that are not covered by the sampled sets, *without* performing a full scan of those sets. This is possible because with high probability only those elements that belong to few input sets are not covered by the sample sets. Therefore, we can efficiently enumerate all pairs (e_i, S_j), $e_i \in S_j$, for those elements e_i that were not covered by the sampled sets. We then run a black box algorithm only on the set system induced by those pairs. This approach lets us avoid the nk term present in the query and runtime bounds for the first algorithm, which makes the second algorithm highly efficient for large values of k.

1.4 Overview of the Lower Bounds. The Set Cover lower bound for smaller optimal value k. We establish our lower bound for the problem of *estimating* the size of the minimum set cover, by constructing two distributions of set systems. All systems in the same distribution share the same optimal set cover size, but these sizes differ by a factor α between the two distributions; thus, the algorithm is required to determine from which distribution its input set system is drawn, in order to correctly estimate the optimal cover size. Our distributions are constructed by a novel use of the probabilistic method. Specifically, we first probabilistically construct a set system called *median instance* (see Lemma 3.1): this set system has the property that (a) its minimum set cover size is αk and (b) a small number of changes to the instance reduces the minimum set cover size to k. We set the first distribution to be always this median instance. Then, we construct the second distribution by a random process that performs the changes (depicted in Figure 1) resulting in a *modified instance*. This process distributes the changes almost uniformly throughout the instance, which implies that the changes are unlikely to be detected unless the algorithm performs a large number of queries. We believe that this construction might find applications to lower

bounds for other combinatorial optimization problems.

The Set Cover lower bound for larger optimal value k. Our lower bound for the problem of *computing* an approximate set cover leverages the construction above. We create a combined set system consisting of multiple modified instances all chosen independently at random, allowing instances with much larger k. By the properties of the random process generating modified instances, we observe that most of these modified instances have different optimal set cover solution, and that distinguishing these instances from one another requires many queries. Thus, it is unlikely for the algorithm to be able to compute an optimal solution to a large fraction of these modified instances, and therefore it fails to achieve the desired approximation factor for the overall combined instance.

The Cover Verification lower bound for a cover of size k. For Cover Verification, however, we instead give an explicit construction of the distributions. We first create an underlying set structure such that initially, the candidate sets contain all but k elements. Then we may swap in each uncovered element from a non-candidate set. Our set structure is systematically designed so that each swap only modifies a small fraction of the answers from all possible queries; hence, each swap is hard to detect without $\Omega(n)$ queries. The distribution of valid set covers is composed of instances obtained by swapping in every uncovered element, and that of non-covers is similarly obtained but leaving one element uncovered.

2 Preliminaries for the Lower Bounds

First, we formally specify the representation of the set structures of input instances, which applies to both Set Cover and Cover Verification.

Our lower bound proofs rely mainly on the construction of instances that are hard to distinguish by the algorithm. To this end, we define the swap operation that exchanges a pair of elements between two sets, and how this is implemented in the actual representation.

DEFINITION 2.1. (swap OPERATION) *Consider two sets S and S'. A swap on S and S' is defined over two elements e, e' such that $e \in S \setminus S'$ and $e' \in S' \setminus S$, where S and S' exchange e and e'. Formally, after performing* swap(e, e'), $S = (S \cup \{e'\}) \setminus \{e\}$ *and* $S' = (S' \cup \{e\}) \setminus \{e'\}$. *As for the representation via* EltOf *and* SetOf, *each application of* swap *only modifies* 2 *entries for each oracle. That is, if previously* $e = \textsc{EltOf}(S, i)$, $S = \textsc{SetOf}(e, j)$, $e' = \textsc{EltOf}(S', i')$, *and* $S' = \textsc{SetOf}(e', j')$, *then their new values change as follows:* $e' = \textsc{EltOf}(S, i)$, $S' = \textsc{SetOf}(e, j)$, $e = \textsc{EltOf}(S', i')$, *and* $S = \textsc{SetOf}(e', j')$.

In particular, we extensively use the property that the amount of changes to the oracle's answers incurred by each swap is minimal. We remark that when we perform multiple swaps on multiple *disjoint* set-element pairs, every swap modifies distinct entries and do not interfere with one another.

Lastly, we define the notion of query-answer history, which is a common tool for establishing lower bounds for sub-linear algorithms under query models.

DEFINITION 2.2. *By* query-answer history, *we denote the sequence of query-answer pairs* $\langle (q_1, a_1), (q_2, a_2), \ldots, (q_r, a_r) \rangle$ *recording the communication between the algorithm and the oracles, where each new query* q_{i+1} *may only depend on the query-answer pairs* $(q_1, a_1), \ldots, (q_i, a_i)$. *In our case, each* q_i *represents either a* SetOf *query or an* EltOf *query made by the algorithm, and each* a_i *is the oracle's answer to that respective query according to the set structure instance.*

3 Lower Bounds for the Set Cover Problem

In this section, we present lower bounds for Set Cover both for small values of the optimal cover size k (in Section 3.1), and for large values of k (in Section 3.2). For low values of k, we prove the following theorem whose proof is postponed to Appendix A.

THEOREM 3.1. *For* $2 \leq k \leq (\frac{n}{16\alpha \log m})^{\frac{1}{4\alpha+1}}$ *and* $1 < \alpha \leq \log n$, *any randomized algorithm that solves the* Set Cover *problem with approximation factor* α *and success probability at least* $2/3$ *requires* $\widetilde{\Omega}(m(n/k)^{\frac{1}{2\alpha}})$ *queries.*

Instead, in Section 3.1 we focus on the simple setting of this theorem which applies to approximation protocols for distinguishing between instances with minimum set cover sizes 2 and 3, and show a lower bound of $\widetilde{\Omega}(mn)$ (which is tight up to a polylogarithmic factor) for approximation factor $3/2$. This simplification is for the purpose of both clarity and also for the fact that the result for this case is used in Section 3.2 to establish our lower bound for large values of k.

High level idea. Our approach for establishing the lower bound is as follows. First, we construct a *median instance* I^* for Set Cover, whose minimum set cover size is 3. We then apply a randomized procedure **genModifiedInst**, which slightly modifies the median instance into a new instance containing a set cover of size 2. Applying Yao's principle, the distribution of the input to the deterministic algorithm is either I^* with probability $1/2$, or a modified instance generated thru **genModifiedInst**(I^*), which is denoted by $\mathcal{D}(I^*)$, again with probability $1/2$. Next, we consider the execution of the deterministic algorithm. We show

that unless the algorithm asks at least $\widetilde{\Omega}(mn)$ queries, the resulting query-answer history generated over I^* would be the same as those generated over instances constituting a constant fraction of $\mathcal{D}(I^*)$, reducing the algorithm's success probability to below 2/3. More specifically, we will establish the following theorem.

THEOREM 3.2. *Any algorithm that can distinguish whether the input instance is I^* or belongs to $\mathcal{D}(I^*)$ with probability of success greater than 2/3, requires $\Omega(mn/\log m)$ queries.*

COROLLARY 3.1. *For $1 < \alpha < 3/2$, and $k \leq 3$, any randomized algorithm that approximates by a factor of α, the size of the optimal cover for the* Set Cover *problem with success probability at least 2/3 requires $\widetilde{\Omega}(mn)$ queries.*

For simplicity, we assume that the algorithm has the knowledge of our construction (which may only strengthens our lower bounds); this includes I^* and $\mathcal{D}(I^*)$, along with their representation via ELTOF and SETOF. The objective of the algorithm is simply to distinguish them. Since we are distinguishing a distribution of instances $\mathcal{D}(I^*)$ against a single instance I^*, we may individually upper bound the probability that each query-answer pair reveals the modified part of the instance, then apply the union bound directly. However, establishing such a bound requires a certain set of properties that we obtain through a careful design of I^* and **genModifiedInst**. We remark that our approach shows the hardness of *distinguishing* instances with with different cover sizes. That is, our lower bound on the query complexity also holds for the problem of approximating the size of the minimum set cover (without explicitly finding one).

Lastly, in Section 3.2 we provide a construction utilizing Theorem 3.2 to extend Corollary 3.1, establish the following theorem on lower bounds for larger minimum set cover sizes.

THEOREM 3.3. *For any sufficiently small approximation factor $\alpha \leq 1.01$ and $k = O(m/\log n)$, any randomized algorithm that computes an α-approximation to the* Set Cover *problem with success probability at least 0.99 requires $\widetilde{\Omega}(mn/k)$ queries.*

3.1 The Set Cover Lower Bound for Small Optimal Value k

3.1.1 Construction of the Median Instance I^*. Let $\mathcal{F}$ be a collection of m sets such that (independently for each set-element pair (S, e)) S contains e with probability $1 - p_0$, where $p_0 = \sqrt{\frac{9 \log m}{n}}$ (note that since we assume $\log m \leq n/c$ for large enough c, we can assume that $p_0 \leq 1/2$). Equivalently, we may consider the incidence matrix of this instance: each entry is either 0 (indicating $e \notin S$) with probability p_0, or 1 (indicating $e \in S$) otherwise. We write $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$ denoting the collection of sets obtained from this construction.

DEFINITION 3.1. (MEDIAN INSTANCE) *An instance of* Set Cover*, I, is a* median instance *if it satisfies all the following properties.*

(a) No two sets cover all the elements. (The size of its minimum set cover is at least 3.)
(b) For any two sets the number of elements not covered by the union of these sets is at most $18 \log m$.
(c) The intersection of any two sets has size at least $n/8$.
(d) For any pair of elements e, e', the number of sets S s.t. $e \in S$ but $e' \notin S$ is at least $\frac{m\sqrt{9 \log m}}{4\sqrt{n}}$.
(e) For any triple of sets S, S_1 and S_2, $|(S_1 \cap S_2) \setminus S| \leq 6\sqrt{n \log m}$.
(f) For each element, the number of sets that do not contain that element is at most $6m\sqrt{\frac{\log m}{n}}$.

LEMMA 3.1. *There exists a median instance I^* satisfying all properties from Definition 3.1. In fact, with high probability, an instance drawn from the distribution in which $\mathbf{Pr}[e \in S] = 1 - p_0$ independently at random, satisfies the median properties.*

The proof of the lemma follows from standard applications of concentration bounds. See the full version of this paper for detailed proofs.

3.1.2 Distribution $\mathcal{D}(I^*)$ of Modified Instances I' Derived from I^*. Fix a median instance I^*. We now show that we may perform $O(\log m)$ swap operations on I^* so that the size of the minimum set cover in the modified instance becomes 2. Moreover, its incidence matrix differs from that of I^* in $O(\log m)$ entries. Consequently, the number of queries to ELTOF and SETOF that induce different answers from those of I^* is also at most $O(\log m)$.

We define $\mathcal{D}(I^*)$ as the distribution of instances I' generated from a median instance I^* by **genModifiedInst**(I^*) given below in Figure 1 as follows. Assume that $I^* = (\mathcal{U}, \mathcal{F})$. We select two different sets S_1, S_2 from $\mathcal{F}$ uniformly at random; we aim to turn these two sets into a set cover. To do so, we swap out some of the elements in S_2 and bring in the uncovered elements. For each uncovered element e, we pick an element $e' \in S_2$ that is also covered by S_1. Next, consider the candidate set that we may exchange its e with $e' \in S_2$:

Definition 3.2. (Candidate set) *For any pair of elements e, e', the* candidate set *of (e, e') are all sets that contain e but not e'. The collection of candidate sets of (e, e') is denoted by* $\mathsf{Candidate}(e, e')$. *Note that* $\mathsf{Candidate}(e, e') \neq \mathsf{Candidate}(e', e)$ *(in fact, these two collections are disjoint).*

genModifiedInst$(I^* = (\mathcal{U}, \mathcal{F}))$:
  $\mathcal{M} \leftarrow \emptyset$
  pick two *different* sets S_1, S_2 from $\mathcal{F}$ uniformly at random
  for each $e \in \mathcal{U} \setminus (S_1 \cup S_2)$ **do**
    pick $e' \in (S_1 \cap S_2) \setminus \mathcal{M}$ uniformly at random
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{e'\}$
    Pick a random set S in $\mathsf{Candidate}(e, e')$
    $\mathsf{swap}(e, e')$ between S, S_2

Figure 1: The procedure of constructing a modified instance of I^*.

We choose a random set S from $\mathsf{Candidate}(e, e')$, and swap $e \in S$ with $e' \in S_2$ so that S_2 now contains e. We repeatedly apply this process for all initially uncovered e so that eventually S_1 and S_2 form a set cover. We show that the proposed algorithm, **genModifiedInst**, can indeed be executed without getting stuck.

Lemma 3.2. *The procedure* **genModifiedInst** *is well-defined under the precondition that the input instance I^* is a median instance.*

Proof. To carry out the algorithm, we must ensure that the number of the initially uncovered elements is at most that of the elements covered by both S_1 and S_2. This follows from the properties of median instances (Definition 3.1): $|\mathcal{U} \setminus (S_1 \cup S_2)| \leq 18 \log m$ by property (b), and that the size of the intersection of S_1 and S_2 is greater than $n/8$ by property (c). That is, in our construction there are sufficiently many possible choices for e' to be matched and swapped with each uncovered element e. Moreover, by property (d) there are plenty of candidate sets S for performing $\mathsf{swap}(e, e')$ with S_2.

3.1.3 Bounding the Probability of Modification. Let $\mathcal{D}(I^*)$ denote the distribution of instances generated by **genModifiedInst**(I^*). If an algorithm were to distinguish between I^* or $I' \sim \mathcal{D}(I^*)$, it must find some cell in the EltOf or SetOf tables that would have been modified by **genModifiedInst**, to confirm that **genModifiedInst** is indeed executed; otherwise it would make wrong decisions half of the time. We will show an additional property of this distribution: none of the entries of EltOf and SetOf are significantly more likely to be modified during the execution of **genModifiedInst**. Consequently, no algorithm may strategically detect the difference between I^* or I' with the desired probability, unless the number of queries is asymptotically the reciprocal of the maximum probability of modification among any cells.

Define $P_{\mathsf{Elt-Set}} : \mathcal{U} \times \mathcal{F} \to [0, 1]$ as the probability that an element is swapped by a set. More precisely, for an element $e \in \mathcal{U}$ and a set $S \in \mathcal{F}$, if $e \notin S$ in the median instance I^*, then $P_{\mathsf{Elt-Set}}(e, S) = 0$; otherwise, it is equal to the probability that S swaps e. We note that these probabilities are taken over $I' \sim \mathcal{D}(I^*)$ where I^* is a fixed median instance. That is, as per Figure 1, they correspond to the random choices of S_1, S_2, the random matching $\mathcal{M}$ between $\mathcal{U} \setminus (S_1 \cup S_2)$ and $S_1 \cap S_2$, and their random choices of choosing each candidate set S. We bound the values of $P_{\mathsf{Elt-Set}}$ via the following lemma.

Lemma 3.3. *For any $e \in \mathcal{U}$ and $S \in \mathcal{F}$, $P_{\mathsf{Elt-Set}}(e, S) \leq \frac{4800 \log m}{mn}$ where the probability is taken over $I' \sim \mathcal{D}(I^*)$.*

Proof. Let S_1, S_2 denote the first two sets picked (uniformly at random) from $\mathcal{F}$ to construct a modified instance of I^*. For each element e and a set S such that $e \in S$ in the basic instance I^*,

$$\begin{aligned} P_{\mathsf{Elt-Set}}(e, S) = {} & \mathbf{Pr}[S = S_2] \cdot \mathbf{Pr}[e \in S_1 \cap S_2] \\ & \cdot \mathbf{Pr}[e \text{ matches to } \mathcal{U} \setminus (S_1 \cup S_2) \mid e \in S_1 \cap S_2] \\ & + \mathbf{Pr}[S \notin \{S_1, S_2\}] \\ & \cdot \mathbf{Pr}[e \in S \setminus (S_1 \cup S_2) \mid e \in S] \\ & \cdot \mathbf{Pr}[S \text{ swaps } e \text{ with } S_2 \mid e \in S \setminus (S_1 \cup S_2)]\,. \end{aligned}$$

where all probabilities are taken over $I' \sim \mathcal{D}(I^*)$. Next we bound each of the above six terms. Since we choose the sets S_1, S_2 randomly, $\mathbf{Pr}[S = S_2] = 1/m$. We bound the second term by 1. For the third term, since we pick a matching uniformly at random among all possible (maximum) matchings between $\mathcal{U} \setminus (S_1 \cup S_2)$ and $S_1 \cap S_2$, by symmetry, the probability that a certain element $e \in S_1 \cap S_2$ is in the matching is (by properties (b) and (c) of median instances),

$$\frac{|\mathcal{U} \setminus (S_1 \cup S_2)|}{|S_1 \cap S_2|} \leq \frac{18 \log m}{n/8} = \frac{144 \log m}{n}.$$

We bound the fourth term by 1. To compute the fifth term, let d_e denote the number of sets in $\mathcal{F}$ that do not contain e. By property (f) of median instances, the probability that $e \in S$ is in $S \setminus (S_1 \cup S_2)$ given that $S \notin \{S_1, S_2\}$ is at most,

$$\frac{d_e(d_e - 1)}{(m-1)(m-2)} \leq \frac{36m^2 \cdot \frac{\log m}{n}}{m^2/2} = \frac{72 \log m}{n}.$$

Finally for the last term, note that by symmetry, each pair of matched elements ee' is picked by **genModifiedInst** equiprobably. Thus, for any $e \in S \setminus (S_1 \cup S_2)$, the probability that each element $e' \in S_1 \cap S_2$ is matched to e is $\frac{1}{|S_1 \cap S_2|}$. By properties (c)–(e) of median instances, the last term is at most

$$\begin{aligned}
&\sum_{e' \in (S_1 \cap S_2) \setminus S} \mathbf{Pr}[ee' \in \mathcal{M}] \cdot \frac{1}{|\mathsf{Candidate}(e, e')|} \\
&= |(S_1 \cap S_2) \setminus S| \cdot \frac{1}{|S_1 \cap S_2|} \cdot \frac{1}{\mathsf{Candidate}(e, e')} \\
&\leq 6\sqrt{n \log m} \cdot \frac{1}{n/8} \cdot \frac{1}{\frac{m\sqrt{9 \log m}}{4\sqrt{n}}} = \frac{64}{m}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
P_{\mathsf{Elt-Set}}(e, S) &\leq \frac{1}{m} \cdot 1 \cdot \frac{144 \log m}{n} + 1 \cdot \frac{72 \log m}{n} \cdot \frac{64}{m} \\
&\leq \frac{4800 \log m}{mn}.
\end{aligned}$$

3.1.4 Proof of Theorem 3.2. Now we consider a median instance I^*, and its corresponding family of modified sets $\mathcal{D}(I^*)$. To prove the promised lower bound for randomized protocols distinguishing I^* and $I' \sim \mathcal{D}(I^*)$, we apply Yao's principle and instead show that no deterministic algorithm $\mathcal{A}$ may determine whether the input is I^* or $I' \sim \mathcal{D}(I^*)$ with success probability at least $2/3$ using $r = o(\frac{mn}{\log m})$ queries. Recall that if $\mathcal{A}$'s query-answer history $\langle (q_1, a_1), \ldots, (q_r, a_r) \rangle$ when executed on I' is the same as that of I^*, then $\mathcal{A}$ must unavoidably return a wrong decision for the probability mass corresponding to I'. We bound the probability of this event as follows.

LEMMA 3.4. *Let Q be the set of queries made by $\mathcal{A}$ on I^*. Let $I' \sim \mathcal{D}(I^*)$ where I^* is a given median instance. Then the probability that $\mathcal{A}$ returns different outputs on I^* and I' is at most $\frac{4800 \log m}{mn}|Q|$.*

Proof of Theorem 3.2. If $\mathcal{A}$ does not output correctly on I^*, the probability of success of $\mathcal{A}$ is less than $1/2$; thus, we can assume that $\mathcal{A}$ returns the correct answer on I^*. This implies that $\mathcal{A}$ returns an incorrect solution on the fraction of $I' \sim I'(I^*)$ for which $\mathcal{A}(I^*) = \mathcal{A}(I')$. Now recall that the distribution in which we apply Yao's principle consists of I^* with probability $1/2$, and drawn uniformly at random from $\mathcal{D}(I^*)$ also with probability $1/2$. Then over this distribution, by Lemma 3.4,

$$\begin{aligned}
\mathbf{Pr}[\mathcal{A} \text{ suceeds}] &\leq 1 - \frac{1}{2}\mathbf{Pr}_{I' \sim \mathcal{D}(I^*)}[\mathcal{A}(I^*) = \mathcal{A}(I')] \\
&\leq 1 - \frac{1}{2}\left(1 - \frac{4800 \log m}{mn}|Q|\right) \\
&= \frac{1}{2} + \frac{2400 \log m}{mn}|Q|.
\end{aligned}$$

Thus, if the number of queries made by $\mathcal{A}$ is less than $\frac{mn}{14400 \log m}$, then the probability that $\mathcal{A}$ returns the correct answer over the input distribution is less than $2/3$ and the proof is complete.

3.2 The SetCover Lower Bound for Large Optimal Value k. Our construction of the median instance I^* and its associated distribution $\mathcal{D}(I^*)$ of modified instances also leads to the lower bound of $\widetilde{\Omega}(\frac{mn}{k})$ for the problem of computing an approximate solution to SetCover. This lower bound matches the performance of our algorithm for large optimal value k and shows that it is tight for some range of value k, albeit it only applies to sufficiently small approximation factor $\alpha \leq 1.01$.

Proof overview. We construct a distribution over *compounds*: a compound is a SetCover instance that consists of $t = \Theta(k)$ smaller instances $I_1, \ldots, I_t$, where each of these t instances is either the median instance I^* or a random modified instance drawn from $\mathcal{D}(I^*)$. By our construction, a large majority of our distribution is composed of compounds that contains at least $0.2t$ modified instances I_i such that, any deterministic algorithm $\mathcal{A}$ must fail to distinguish I_i from I^* when it is only allowed to make a small number of queries. A deterministic $\mathcal{A}$ can safely cover these modified instances with three sets, incurring a cost (sub-optimality) of $0.2t$. Still, $\mathcal{A}$ may choose to cover such an I_i with two sets to reduce its cost, but it then must err on a different compound where I_i is replaced with I^*. We track down the trade-off between the amount of cost that $\mathcal{A}$ saves on these compounds by covering these I_i's with two sets, and the amount of error on other compounds its scheme incurs. $\mathcal{A}$ is allowed a small probability δ to make errors, which we then use to upper-bound the expected cost that $\mathcal{A}$ may save, and conclude that $\mathcal{A}$ still incurs an expected cost of $0.1t$ overall. We apply Yao's principle (for algorithms with errors) to obtain that randomized algorithms also incur an expected cost of $0.05t$, on compounds with optimal solution size $k \in [2t, 3t]$, yielding the impossibility result for computing solutions with approximation factor $\alpha = \frac{k+0.1t}{k} > 1.01$ when given insufficient queries.

3.2.1 Overall Lower Bound Argument Compounds. Consider the median instance I^* and its associated distribution $\mathcal{D}(I^*)$ of modified instances for Set Cover with n elements and m sets, and let $t = \Theta(k)$ be a positive integer parameter. We define a *compound* $\mathfrak{I} = \mathfrak{I}(I_1, I_2, \ldots, I_t)$ as a set structure instance consisting of t median or modified instances $I_1, I_2, \ldots, I_t$, forming a set structure $(\mathcal{U}^t, \mathcal{F}^t)$ of $n' \triangleq nt$ elements and $m' \triangleq mt$ sets, in such a way that each instance I_i occupies separate elements and sets. Since the optimal solution to each instance I_i is 3 if $I_i = I^*$, and 2 if I_i is any modified instance, the optimal solution for the compound is $2t$ plus the number of occurrences of the median instance; this optimal objective value is always $\Theta(k)$.

Random distribution over compounds. Employing Yao's principle, we construct a distribution $\mathfrak{D}$ of compounds $\mathfrak{I}(I_1, I_2, \ldots, I_t)$: it will be applied against any deterministic algorithm $\mathcal{A}$ for computing an approximate minimum set cover, which is allowed to err on at most a δ-fraction of the compounds from the distribution (for some small constant $\delta > 0$). For each $i \in [t]$, we pick $I_i = I^*$ with probability $c/\binom{m}{2}$ where $c > 2$ is a sufficiently large constant. Otherwise, simply draw a random modified instance $I_i \sim \mathcal{D}(I^*)$. We aim to show that, in expectation over $\mathfrak{D}$, $\mathcal{A}$ must output a solution that of size $\Theta(t)$ more than the optimal set cover size of the given instance $\mathfrak{I} \sim \mathfrak{D}$.

$\mathcal{A}$ frequently leaves many modified instances undetected. Consider an instance $\mathfrak{I}$ containing at least $0.95t$ modified instances. These instances constitute at least a 0.99-fraction of $\mathfrak{D}$: the expected number of occurrences of the median instance in each compound is only $c/\binom{m}{2} \cdot t = O(t/m^2)$, so by Markov's inequality, the probablity that there are more than $0.05t$ median instances is at most $O(1/m^2) < 0.01$ for large m. We make use of the following useful lemma, whose proof is deferred to Section 3.2.2. In what follow, we say that the algorithm "distinguishes" or "detects the difference" between I_i and I^* if it makes a query that induces different answers, and thus may deduce that one of I_i or I^* cannot be the input instance. In particular, if $I_i = I^*$ then detecting the difference between them would be impossible.

LEMMA 3.5. *Fix $M \subseteq [t]$ and consider the distribution over compounds $\mathfrak{I}(I_1, \ldots, I_t)$ with $I_i \sim \mathcal{D}(I^*)$ for $i \in M$ and $I_i = I^*$ for $i \notin M$. If $\mathcal{A}$ makes at most $o(\frac{mnt}{\log m})$ queries to $\mathfrak{I}$, then it may detect the differences between I^* and at least $0.75t$ of the modified instances $\{I_i\}_{i \in M}$, with probability at most 0.01.*

We apply this lemma for any $|M| \geq 0.95t$ (although the statement holds for any M, even vacuously for $|M| < 0.75t$). Thus, for $0.99 \cdot 0.99 > 0.98$-fraction of $\mathfrak{D}$, $\mathcal{A}$ fails to identify, for at least $0.95t - 0.75t = 0.2t$ modified instances I_i in $\mathfrak{I}$, whether it is a median instance or a modified instance. Observe that the query-answer history of $\mathcal{A}$ on such $\mathfrak{I}$ would not change if we were to replace any combination of these $0.2t$ modified instances by copies of I^*. Consequently, if the algorithm were to correctly cover $\mathfrak{I}$ by using two sets for some of these I_i, it must unavoidably err (return a non-cover) on the compound where these I_i's are replaced by copies of the median instance.

Charging argument. We call a compound $\mathfrak{I}$ *tough* if $\mathcal{A}$ does not err on $\mathfrak{I}$, and $\mathcal{A}$ fails to detect at least $0.2t$ modified instances; denote by $\mathfrak{D}^{\mathsf{tough}}$ the conditional distribution of $\mathfrak{D}$ restricted to tough instances. For tough $\mathfrak{I}$, let $\mathsf{cost}(\mathfrak{I})$ denote the number of modified instances I_i that the algorithm decides to cover with three sets. That is, for each tough compound $\mathfrak{I}$, $\mathsf{cost}(\mathfrak{I})$ measures how far the solution returned by $\mathcal{A}$ is, from the optimal set cover size. Then, there are at least $0.2t - \mathsf{cost}(\mathfrak{I})$ modified instances I_i that $\mathcal{A}$ chooses to cover with only two sets despite not being able to verify whether $I_i = I^*$ or not. Let $R_{\mathfrak{I}}$ denote the set of the indices of these modified instances, so $|R_{\mathfrak{I}}| = 0.2t - \mathsf{cost}(\mathfrak{I})$. By doing so, $\mathcal{A}$ then errs on the *replaced compound* $r(\mathfrak{I}, R_{\mathfrak{I}})$, denoting the compound similar to $\mathfrak{I}$, except that each modified instance I_i for $i \in R_{\mathfrak{I}}$ is replaced by I^*. In this event, we say that the tough compound $\mathfrak{I}$ *charges* the replaced compound $r(\mathfrak{I}, R_{\mathfrak{I}})$ via $R_{\mathfrak{I}}$. Recall that the total error of $\mathcal{A}$ is δ: this quantity upper-bounds the total probability masses of charged instances, which we will then manipulate to obtain a lower bound on $\mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}}[\mathsf{cost}(\mathfrak{I})]$.

Instances must share optimal solutions for R to charge the same replaced instance. Observe that many tough instances may charge to the same replaced instance: we must handle these duplicities. First, consider two tough instances $\mathfrak{I}^1 \neq \mathfrak{I}^2$ charing the same $\mathfrak{I}_{\mathsf{r}} = r(\mathfrak{I}^1, R) = r(\mathfrak{I}^2, R)$ via the same $R = R_{\mathfrak{I}^1} = R_{\mathfrak{I}^2}$. As $\mathfrak{I}^1 \neq \mathfrak{I}^2$ but $r(\mathfrak{I}^1, R) = r(\mathfrak{I}^2, R)$, these tough instances differ on some modified instances with indices in R. Nonetheless, the query-answer histories of $\mathcal{A}$ operating on $\mathfrak{I}^1$ and $\mathfrak{I}^2$ must be the same as their instances in R are both indistinguishable from I^* by the deterministic $\mathcal{A}$. Since $\mathcal{A}$ does not err on tough instances (by definition), both tough $\mathfrak{I}^1$ and $\mathfrak{I}^2$ must share the same optimal set cover on every instance in R. Consequently, for each fixed R, only tough instances that have the same optimal solution for modified instances in R may charge the same replaced instance via R.

Charged instance is much heavier than charging instances combined. By our construction of

$\mathfrak{I}(I_1, \ldots, I_t)$ drawn from $\mathfrak{D}$, $\Pr[I_i = I^*] = c/\binom{m}{2}$ for the median instance. On the other hand, $\sum_{j=1}^{\ell} \Pr[I_i = I^j] \le (1 - c/\binom{m}{2}) \cdot (1/\binom{m}{2}) < 1/\binom{m}{2}$ for modified instances $I^1, \ldots, I^\ell$ sharing the same optimal set cover, because they are all modified instances constructed to have the two sets chosen by **genModifiedInst** as their optimal set cover: each pair of sets is chosen uniformly with probability $1/\binom{m}{2}$. Thus, the probability that I^* is chosen is more than c times the total probability that any I^j is chosen. Generalizing this observation, we consider tough instances $\mathfrak{I}^1, \mathfrak{I}^2, \ldots, \mathfrak{I}^\ell$ charging the same $\mathfrak{I}_{\mathsf{r}}$ via R, and bound the difference in probabilities that $\mathfrak{I}_{\mathsf{r}}$ and any $\mathfrak{I}^j$ are drawn. For each index in R, it is more than c times more likely for $\mathfrak{D}$ to draw the median instance, rather than any modified instances of a fixed optimal solution. Then, for the replaced compound $\mathfrak{I}_{\mathsf{r}}$ that $\mathcal{A}$ errs, $p(\mathfrak{I}_{\mathsf{r}}) \ge c^{|R|} \cdot \sum_{j=1}^{\ell} p(\mathfrak{I}^j)$ (where p denotes the probability mass in $\mathfrak{D}$, not in $\mathfrak{D}^{\mathsf{tough}}$). In other words, the probability mass of the replaced instance charged via R is always at least $c^{|R|}$ times the total probability mass of the charging tough instances.

Bounding the expected cost using δ. In our charging argument by tough instances above, we only bound the amount of charges on the replaced instances via a fixed R. As there are up to 2^t choices for R, we scale down the total amount charged to a replaced instance by a factor of 2^t, so that $\sum_{\mathsf{tough}\ \mathfrak{I}} c^{|R_{\mathfrak{I}}|} p(\mathfrak{I})/2^t$ lower bounds the total probability mass of the replaced instances that $\mathcal{A}$ errs.

Let us first focus on the conditional distribution $\mathfrak{D}^{\mathsf{tough}}$ restricted to tough instances. Recall that at least a $(0.98 - \delta)$-fraction of the compounds in $\mathfrak{D}$ are tough: $\mathcal{A}$ fails to detect differences between $0.2t$ modified instances from the median instance with probability 0.98, and among these compounds, $\mathcal{A}$ may err on at most a δ-fraction. So in the conditional distribution $\mathfrak{D}^{\mathsf{tough}}$ over tough instances, the individual probability mass is scaled-up to $p^{\mathsf{tough}}(\mathfrak{I}) \le \frac{p(\mathfrak{I})}{0.98-\delta}$. Thus,

$$\frac{\sum_{\mathsf{tough}\ \mathfrak{I}} c^{|R_{\mathfrak{I}}|} p(\mathfrak{I})}{2^t} \ge \frac{\sum_{\mathsf{tough}\ \mathfrak{I}} c^{|R_{\mathfrak{I}}|} (0.98 - \delta) p^{\mathsf{tough}}(\mathfrak{I})}{2^t} = \frac{(0.98 - \delta) \mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}^{\mathsf{tough}}} \left[c^{|R_{\mathfrak{I}}|} \right]}{2^t}.$$

As the probability mass above cannot exceed the total allowed error δ, we have

$$\frac{\delta}{0.98 - \delta} \cdot 2^t \ge \mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}^{\mathsf{tough}}} \left[c^{|R_{\mathfrak{I}}|} \right] \ge \mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}^{\mathsf{tough}}} \left[c^{0.2t - \mathsf{cost}(\mathfrak{I})} \right] \ge c^{0.2t - \mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}^{\mathsf{tough}}}[\mathsf{cost}(\mathfrak{I})]},$$

where Jensen's inequality is applied in the last step above. So,

$$\mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}^{\mathsf{tough}}}[\mathsf{cost}(\mathfrak{I})] \ge 0.2t - \frac{t + \log \frac{\delta}{0.98-\delta}}{\log c} = \left(0.2 - \frac{1}{\log c}\right) t - \frac{\log \frac{\delta}{0.98-\delta}}{\log c} \ge 0.11t,$$

for sufficiently large c (and m) when choosing $\delta = 0.02$.

We now return to the expected cost over the entire distribution $\mathfrak{I}$. For simplicity, define $\mathsf{cost}(\mathfrak{I}) = 0$ for any non-tough $\mathfrak{I}$. This yields $\mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}}[\mathsf{cost}(\mathfrak{I})] \ge (0.98 - \delta) \mathbf{E}_{\mathfrak{I} \sim \mathfrak{D}^{\mathsf{tough}}}[\mathsf{cost}(\mathfrak{I})] \ge (0.98 - \delta) \cdot 0.11t \ge 0.1t$, establishing the expected cost of any deterministic $\mathcal{A}$ with probability of error at most 0.02 over $\mathfrak{D}$.

Establishing the lower bound for randomized algorithms. Lastly, we apply Yao's principle[4] to obtain that, for any randomized algorithm with error probability $\delta/2 = 0.01$, its expected cost under the worst input is at least $\frac{1}{2} \cdot 0.1t = 0.05t$. Recall now that our cost here lower-bounds the sub-optimality of the computed set cover (that is, the algorithm uses at least cost more sets to cover the elements than the optimal solution does). Since our input instances have optimal solution $k \in [2t, 3t]$ and the randomized algorithm returns a solution with cost at least $0.05t$ in expectation, it achieves an approximation factor of no better than $\alpha = \frac{k+0.05t}{k} > 1.01$ with $o(\frac{mnt}{\log m})$ queries. Theorem 3.3 then follows, noting the substitution of our problem size: $\frac{mnt}{\log m} = \frac{(m'/t)(n'/t)t}{\log(m'/t)} = \Theta(\frac{m'n'}{k' \log m'})$.

3.2.2 Proof of Lemma 3.5 First, we recall the following result from Lemma 3.4 for distinguishing between I^* and a random $I' \sim \mathcal{D}(I^*)$.

COROLLARY 3.2. *Let q be the number of queries made by $\mathcal{A}$ on $I_i \sim \mathcal{D}(I^*)$ over n elements and m sets, where I^* is a median instance. Then the probability that $\mathcal{A}$ detects a difference between I_i and I^* in one of its queries is at most $\frac{4800q \log m}{mn}$.*

Marbles and urns. Fix a compound $\mathfrak{I}(I_1, \ldots, I_t)$. Let $s \triangleq \frac{mn}{4800 \log m}$, and then consider the following, entirely different, scenario. Suppose that we have t urns, where each urn contains s marbles. In the i^{th} urn, in case I_i is a modified instance, we put in this urn one red marble and $s - 1$ white marbles; otherwise if $I_i = I^*$, we put in s white marbles. Observe that the probability of obtaining a red marble by drawing q

[4] Here we use the Monte Carlo version where the algorithm may err, and use cost instead of the time complexity as our measure of performance. See, e.g., Proposition 2.6 in [27] and the description therein.

marbles from a single urn *without replacement* is exactly q/s (for $q \leq s$). Now, we will relate the probability of drawing red marbles to the probability of successfully distinguishing instances. We emphasize that we are only comparing the probabilities of events for the sake of analysis, and we do not imply or suggest any direct analogy between the events themselves.

Corollary 3.2 above bounds the probability that the algorithm successfully distinguishes a modified instance I_i from I^* with $\frac{4800q\log m}{mn} = q/s$. Then, the probability of distinguishing between I_i and I^* using q queries, is bounded from above by the probability of obtaining a red marble after drawing q marbles from an urn. Consequently, the probability that the algorithm distinguishes $3t/4$ instances is bounded from above by the probability of drawing the red marbles from at least $3t/4$ urns. Hence, to prove that the event of Lemma 3.5 occurs with probability at most 0.01, it is sufficient to upper-bound the probability that an algorithm obtains $3t/4$ red marbles by 0.01.

Consider an instance of t urns; for each urn $i \in [t]$ corresponding to a modified instance I_i, exactly one of its s marbles is red. An algorithm may draw marbles from each urn, one by one without replacement, for potentially up to s times. By the principle of deferred decisions, the red marble is equally likely to appear in any of these s draws, independent of the events for other urns. Thus, we can create a tuple of t random variables $\mathcal{T} = (T_1, \ldots, T_t)$ such that for each $i \in [t]$, T_i is chosen uniformly at random from $\{1, \ldots, s\}$. The variable T_i represents the number of draws required to obtain the red marble in the i^{th} urn; that is, only the T_i^{th} draw from the i^{th} urn finds the red marble from that urn. In case I_i is a median instance, we simply set $T_i = s+1$ indicating that the algorithm never detects any difference as I_i and I^* are the same instance.

We now show the following two lemmas in order to bound the number of red marbles the algorithm may encounter throughout its execution.

LEMMA 3.6. *Let $b > 3$ be a fixed constant and define $\mathcal{T}_{\text{high}} = \{i \mid T_i \geq \frac{s}{b}\}$. If $t \geq 14b$, then $|\mathcal{T}_{\text{high}}| \geq (1 - \frac{2}{b})t$ with probability at least 0.99.*

Proof. Let $\mathcal{T}_{\text{low}} = \{1, \ldots, t\} \setminus \mathcal{T}_{\text{high}}$. Notice that for the i^{th} urn, $\mathbf{Pr}[i \in \mathcal{T}_{\text{low}}] < \frac{1}{b}$ independently of other urns, and thus $|\mathcal{T}_{\text{low}}|$ is stochastically dominated by $\mathsf{B}(t, \frac{1}{b})$, the binomial distribution with t trials and success probability $\frac{1}{b}$. Applying Chernoff bound, we obtain

$$\mathbf{Pr}\left[|\mathcal{T}_{\text{low}}| \geq \frac{2t}{b}\right] \leq e^{-\frac{t}{3b}} < 0.01.$$

Hence, $|\mathcal{T}_{\text{high}}| \geq t - \frac{2t}{b} = (1 - \frac{2}{b})t$ with probability at least 0.99, as desired.

LEMMA 3.7. *If the total number of draws made by the algorithm is less than $(1 - \frac{3}{b})\frac{st}{b}$, then with probability at least 0.99, the algorithm will not obtain red marbles from at least $\frac{t}{b}$ urns.*

Proof. If the total number of such draws is less than $(1-\frac{3}{b})\frac{st}{b}$, then the number of draws from at least $\frac{3t}{b}$ urns is less than $\frac{s}{b}$ each. Assume the condition of Lemma 3.6: for at least $(1-\frac{2}{b})t$ urns, $T_i \geq \frac{s}{b}$. That is, the algorithm will not encounter a red marble if it makes less than $\frac{s}{b}$ draws from such an urn. Then, there are at least $\frac{t}{b}$ urns with $T_i \geq \frac{s}{b}$ from which the algorithm makes less than $\frac{s}{b}$ draws, and thus does not obtain a red marble. Overall this event holds with probability at least 0.99 due to Lemma 3.6.

We substitute $b = 4$ and assume sufficiently large t. Suppose that the deterministic algorithm makes less than $(1-\frac{3}{4})\frac{st}{4} = \frac{st}{16}$ queries, then for a fraction of 0.99 of all possible tuples $\mathcal{T}$, there are $t/4$ instances I_i that the algorithm fails to detect their differences from I^*: the probability of this event is lower-bounded by that of the event where the red marbles from those corresponding urns i are not drawn. Therefore, the probability that the algorithm makes queries that detect differences between I^* and more than $3t/4$ instances I_i's is bounded by 0.01, concluding our proof of Lemma 3.5.

4 Sub-Linear Algorithms for the Set Cover Problem

In this paper, we present two different approximation algorithms for Set Cover with sub-linear query in the oracle model: **smallSetCover** and **largeSetCover**. Both of our algorithms rely on the techniques from the recent developments on Set Cover in the streaming model. However, adopting those techniques in the oracle model requires novel insights and technical development.

Throughout the description of our algorithms, we assume that we have access to a black box subroutine that given the full Set Cover instance (where all members of all sets are revealed), returns a ρ-approximate solution[5].

The first algorithm (**smallSetCover**) returns a $(\alpha\rho+\varepsilon)$ approximate solution of the Set Cover instance using $\widetilde{O}(\frac{1}{\varepsilon}(m(\frac{n}{k})^{\frac{1}{\alpha-1}} + nk))$ queries, while the second algorithm (**largeSetCover**) achieves an approximation factor of $(\rho + \varepsilon)$ using $\widetilde{O}(\frac{mn}{k\varepsilon^2})$ queries, where k is the size of the minimum set cover. These algorithms can be combined so that the number of queries of the algorithm becomes asymptotically the minimum of the two:

[5]The approximation factor ρ may take on any value between 1 and $\Theta(\log n)$ depending on the computational model one assumes.

THEOREM 4.1. *There exists a randomized algorithm for* Set Cover *in the oracle model that w.h.p.*[6] *computes an $O(\rho \log n)$-approximate solution and uses $\widetilde{O}(\min\{m\left(\frac{n}{k}\right)^{1/\log n} + nk\ ,\ \frac{mn}{k}\}) = \widetilde{O}(m + n\sqrt{m})$ number of queries.*

4.1 Preliminaries. Our algorithms use the following two sampling techniques developed for Set Cover in the streaming model [10]: Element Sampling and Set Sampling. The first technique, Element Sampling, states that in order to find a $(1-\delta)$-cover of $\mathcal{U}$ w.h.p., it suffices to solve Set Cover on a subset of elements of size $\widetilde{O}(\frac{\rho k \log m}{\delta})$ picked uniformly at random. It shows that we may restrict our attention to a subproblem with a much smaller number of elements, and our solution to the reduced instance will still cover a good fraction of the elements in the original instance. The next technique, Set Sampling, shows that if we pick ℓ sets uniformly at random from $\mathcal{F}$ in the solution, then each element that is not covered by any of picked sets w.h.p. only occurs in $\widetilde{O}(\frac{m}{\ell})$ sets in $\mathcal{F}$; that is, we are left with a much sparser subproblem to solve. The formal statements of these sampling techniques are as follows. See [10] for the proofs.

LEMMA 4.1. (ELEMENT SAMPLING) *Consider an instance of* Set Cover *on $(\mathcal{U}, \mathcal{F})$ whose optimal cover has size at most k. Let $\mathcal{U}_{\text{smp}}$ be a subset of $\mathcal{U}$ of size $\Theta\left(\frac{\rho k \log m}{\delta}\right)$ chosen uniformly at random, and let $\mathcal{C}_{\text{smp}} \subseteq \mathcal{F}$ be a ρ-approximate cover for $\mathcal{U}_{\text{smp}}$. Then, w.h.p. $\mathcal{C}_{\text{smp}}$ covers at least $(1-\delta)|\mathcal{U}|$ elements.*

LEMMA 4.2. (SET SAMPLING) *Consider an instance $(\mathcal{U}, \mathcal{F})$ of* Set Cover. *Let $\mathcal{F}_{\mathsf{rnd}}$ be a collection of ℓ sets picked uniformly at random. Then, w.h.p. $\mathcal{F}_{\mathsf{rnd}}$ covers all elements that appear in $\Omega(\frac{m \log n}{\ell})$ sets of $\mathcal{F}$.*

4.2 The smallSetCover Algorithm. The algorithm of this section is a modified variant of the streaming algorithm of Set Cover in [17] that works in the sublinear query model. Similarly to the algorithm of [17], our algorithm **smallSetCover** considers different guesses of the value of an optimal solution ($\varepsilon^{-1} \log n$ guesses) and performs the core *iterative* algorithm **iterSetCover** for all of them in parallel. For each guess ℓ of the size of an optimal solution, the **iterSetCover** goes through $1/\alpha$ iterations and by applying Element Sampling, guarantees that w.h.p. at the end of each iteration, the number of uncovered elements reduces by a factor of $n^{-1/\alpha}$. Hence, after $1/\alpha$ iterations all elements will be covered. Furthermore, since the number of sets picked in each iteration is at most ℓ, the final solution has at most $\rho\ell$ sets where ρ is the performance of the *offline* block **algOfflineSC** that **iterSetCover** uses to solve the reduced instances constructed by Element Sampling.

Although our general approach in **iterSetCover** is similar to the iterative core of the streaming algorithm of Set Cover, there are challenges that we need to overcome so that it works *efficiently* in the query model. Firstly, the approach of [17] relies on the ability to test membership for a set-element pair when executing its *set filtering* subroutine: given a subset S, the algorithm of [17] requires to compute $|S \cap \mathsf{S}|$ which cannot be implemented efficiently in the query model (in the worst case, requires $m|\mathsf{S}|$ queries). Instead, here we employ the *set sampling* which w.h.p. guarantees that the number of sets that contain an (yet uncovered) element is small.

Next challenge is achieving $m(n/k)^{1/(\alpha-1)} + nk$ query bound for computing an α-approximate solution. As mentioned earlier, both our approach and the algorithm of [17] need to run the algorithm in parallel for different guesses ℓ of the size of an optimal solution. However, since **iterSetCover** performs $m(n/\ell)^{1/(\alpha-1)} + n\ell$ queries, if **smallSetCover** invokes **iterSetCover** with guesses in an increasing order then the query complexity becomes $mn^{1/(\alpha-1)} + nk$; on the other hand, if it invokes **iterSetCover** with guesses in a decreasing order then the query complexity becomes $m(n/k)^{1/(\alpha-1)} + mn$! To solve this issue, **smallSetCover** performs in two stages: in the first stage, it finds a $(\log n)$-estimate of k by invoking **iterSetCover** using $m + nk$ queries (assuming guesses are evaluated in an increasing order) and then in the second rounds it only invokes **iterSetCover** with approximation factor α in the smaller $O(\log n)$-approximate region around the $(\log n)$-estimate of k computed in the first stage. Thus, in our implementation, besides the desired approximation factor, **iterSetCover** receives an upper bound and a lower bound on the size of an optimal solution.

Now, we provide a detailed description of **iterSetCover**. It receives α, ϵ, l and u as its arguments, and it is guaranteed that the size of an optimal cover of the input instance, k, is in $[l, u]$. Note that the algorithm does not know the value of k and the sampling techniques described in Section 4.1 rely on k. Therefore, the algorithm needs to find a $(1+\varepsilon)$ estimate[7] of k denoted as ℓ. This can be done by trying all powers of

[6] An algorithm succeeds *with high probability* (w.h.p.) if its failure probability can be decreased to n^{-c} for any constant $c > 0$ without affecting its asymptotic performance, where n denotes the input size.

[7] The exact estimate that the algorithm works with is a $(1 + \frac{\varepsilon}{2\rho\alpha})$ estimate.

$(1+\varepsilon)$ in $[l,u]$. The parameter α denotes the trade-off between the query complexity and the approximation guarantee that the algorithm achieves. Moreover, we assume that the algorithm has access to a ρ-approximate black box solver of Set Cover.

iterSetCover first performs Set Sampling to cover all elements that occur in $\widetilde{\Omega}(m/\ell)$ sets. Then it goes through $\alpha-2$ iterations and in each iteration, it performs Element Sampling with parameter $\delta = \widetilde{O}((\ell/n)^{1/(\alpha-1)})$. By Lemma 4.1, after $(\alpha-2)$ iterations, w.h.p. only $\ell\left(\frac{n}{\ell}\right)^{1/(\alpha-1)}$ elements remain uncovered, for which the algorithm finds a cover by invoking the *offline* set cover solver. The parameters are set so that all $(\alpha-1)$ instances that are required to be solved by the offline set cover solver (the $(\alpha-2)$ instances costructed by Element Sampling and the final instance) are of size $\widetilde{O}(m\left(\frac{n}{\ell}\right)^{1/(\alpha-1)})$.

In the rest of this section, we show that **smallSetCover** w.h.p. returns an almost $(\rho\alpha)$-approximate solution of Set Cover$(\mathcal{U},\mathcal{F})$ with query complexity $\widetilde{O}(m\left(\frac{n}{k}\right)^{\frac{1}{\alpha-1}}+nk)$ where k is the size of a minimum set cover.

THEOREM 4.2. *The* **smallSetCover** *algorithm outputs a $(\alpha\rho+\varepsilon)$-approximate solution of* Set Cover$(\mathcal{U},\mathcal{F})$ *using $\widetilde{O}(\frac{1}{\varepsilon}(m(n/k)^{\frac{1}{\alpha-1}}+nk))$ number of queries w.h.p., where k is the size of an optimal solution of $(\mathcal{U},\mathcal{F})$.*

To analyze the performance of **smallSetCover**, first we need to analyze the procedures invoked by **smallSetCover**: **iterSetCover** and **algOfflineSC**. The procedure **algOfflineSC**(S,ℓ) receives as an input a subset of elements S and an estimate on the size of an optimal cover of S using sets in $\mathcal{F}$. The **algOfflineSC** algorithm first determines all occurrences of S in $\mathcal{F}$. Then it invokes a black box subroutine that returns a cover of size at most $\rho\ell$ (if there exists a cover of size ℓ for S) for the reduced Set Cover instance over S.

Moreover, we assume that all subroutines have access to the EltOf and SetOf oracles, $|\mathcal{U}|$ and $|\mathcal{F}|$.

LEMMA 4.3. *Suppose that each $e\in\mathsf{S}$ appears in $\widetilde{O}(\frac{m}{\ell})$ sets of $\mathcal{F}$ and lets assume that there exists a set of ℓ sets in $\mathcal{F}$ that covers S. Then* **algOfflineSC**(S,ℓ) *returns a cover of size at most $\rho\ell$ of S using $\widetilde{O}(\frac{m|\mathsf{S}|}{\ell})$ queries.*

Proof. Since each element of S is contained by $\widetilde{O}(\frac{m}{\ell})$ sets in $\mathcal{F}$, the information required to solve the reduced instance on S can be obtained by $\widetilde{O}(\frac{m|\mathsf{S}|}{\ell})$ queries (i.e. $\widetilde{O}(\frac{m}{\ell})$ SetOf query per element in S).

LEMMA 4.4. *The cover constructed by the outer loop of* **iterSetCover**(α,ε,l,u) *with the parameter $\ell>k$, $\mathtt{sol}_\ell$, w.h.p. covers $\mathcal{U}$.*

iterSetCover(α,ε,l,u):
▷ Try all $(1+\frac{\varepsilon}{2\alpha\rho})$-approximate guesses of k
for $\ell\in\{(1+\frac{\varepsilon}{2\alpha\rho})^i \mid \log_{1+\frac{\varepsilon}{2\alpha\rho}} l\le i\le \log_{1+\frac{\varepsilon}{2\alpha\rho}} u\}$
do in order:
$\mathtt{sol}_\ell \leftarrow$ collection of ℓ sets picked uniformly at random ▷ Set Sampling
$\mathcal{U}_{\text{rem}} \leftarrow \mathcal{U}\setminus\bigcup_{\mathsf{r}\in\mathtt{sol}_\ell}\mathsf{r}$ ▷ $n\ell$ EltOf
repeat $(\alpha-2)$ times
$\mathsf{S}\leftarrow$ sample of $\mathcal{U}_{\text{rem}}$ of size $\widetilde{O}(\rho\ell\left(\frac{n}{\ell}\right)^{\frac{1}{\alpha-1}})$
$\mathcal{D}\leftarrow$ **algOfflineSC**(S,ℓ)
if $\mathcal{D}=$ null **then**
break ▷ Try the next value of ℓ
$\mathtt{sol}_\ell\leftarrow\mathtt{sol}_\ell\bigcup\mathcal{D}$
$\mathcal{U}_{\text{rem}}\leftarrow\mathcal{U}_{\text{rem}}\setminus\bigcup_{\mathsf{r}\in\mathcal{D}}\mathsf{r}$ ▷ $\rho n\ell$ EltOf
if $|\mathcal{U}_{\text{rem}}|\le\ell\left(\frac{n}{\ell}\right)^{1/(\alpha-1)}$ ▷ Feasibility Test
$\mathcal{D}\leftarrow$ **algOfflineSC**$(\mathcal{U}_{\text{rem}},\ell)$
if $\mathcal{D}\neq$ null **then**
$\mathtt{sol}_\ell\leftarrow\mathtt{sol}_\ell\bigcup\mathcal{D}$
return $\mathtt{sol}_\ell$

Figure 2: **iterSetCover** is the main procedure of the **smallSetCover** algorithm for the Set Cover problem.

Proof. After picking ℓ sets uniformly at random, by Set Sampling (Lemma 4.2), w.h.p. each element that is not covered by the sampled sets appears in $\widetilde{O}(\frac{m}{\ell})$ sets of $\mathcal{F}$. Next, by Element Sampling (Lemma 4.1 with $\delta=\left(\frac{\ell}{n}\right)^{1/(\alpha-1)}$), at the end of each *inner* iteration, w.h.p. the number of uncovered elements decreases by a factor of $\left(\frac{\ell}{n}\right)^{1/(\alpha-1)}$. Thus after at most $(\alpha-2)$ iterations, w.h.p. less than $\ell\left(\frac{n}{\ell}\right)^{1/(\alpha-1)}$ elements remain uncovered. Finally, **algOfflineSC** is invoked on the remaining elements; hence, $\mathtt{sol}_\ell$ w.h.p. covers $\mathcal{U}$.

Next we analyze the query complexity and the approximation guarantee of **iterSetCover**. As we only apply Element Sampling and Set Sampling polynomially many times, all invocations of the corresponding lemmas during an execution of the algorithm must succeed w.h.p., so we assume their *high probability* guarantees for the proofs in rest of this section.

LEMMA 4.5. *Given that $l\le k\le\frac{u}{1+\varepsilon/(2\alpha\rho)}$, w.h.p.* **iterSetCover**(α,ε,l,u) *finds a $(\rho\alpha+\varepsilon)$-approximate solution of the input instance using $\widetilde{O}\left(\frac{1}{\varepsilon}(m(\frac{n}{l})^{1/(\alpha-1)}+nk)\right)$ queries.*

Proof. Let $\ell_k=(1+\frac{\varepsilon}{2\alpha\rho})^{\lceil\log_{1+\frac{\varepsilon}{2\alpha\rho}}k\rceil}$ be the smallest power of $1+\frac{\varepsilon}{2\alpha\rho}$ greater than or equal to k. Note that it is guaranteed that $\ell_k\in[l,u]$. By Lemma 4.4, **iterSetCover** terminates with a guess value $\ell\le\ell_k$. In

algOfflineSC(S, ℓ):
$\mathcal{F}_{\mathsf{S}} \leftarrow \emptyset$
for each element $e \in \mathsf{S}$ **do**
$\mathcal{F}_e \leftarrow$ the collection of sets containing e
$\mathcal{F}_{\mathsf{S}} \leftarrow \mathcal{F}_{\mathsf{S}} \cup \mathcal{F}_e$
$\mathcal{D} \leftarrow$ solution of size at most $\rho\ell$ for Set Cover on $(\mathsf{S}, \mathcal{F}_{\mathsf{S}})$ constructed by the black box solver
▷ If there exists no such cover, then $\mathcal{D} =$ null
return $\mathcal{D}$

Figure 3: **algOfflineSC**(S, ℓ) invokes a black box that returns a cover of size at most $\rho\ell$ (if there exists a cover of size ℓ for S) for the Set Cover instance that is the projection of $\mathcal{F}$ over S.

the following we compute the query complexity of the run of **iterSetCover** with a parameter $\ell \leq \ell_k$.

Set Sampling component picks ℓ sets and then update the set of elements that are not covered by those sets, $\mathcal{U}_{\text{rem}}$, using $O(n\ell)$ EltOf queries. Next, in each iteration of the inner loop, the algorithm samples a subset S of size $\widetilde{O}\left(\ell(n/\ell)^{1/(\alpha-1)}\right)$ from $\mathcal{U}_{\text{rem}}$. Recall that, by Set Sampling (Lemma 4.2), each $e \in \mathsf{S} \subset \mathcal{U}_{\text{rem}}$ appears in at most $\widetilde{O}(m/\ell)$ sets. Since each element in $\mathcal{U}_{\text{rem}}$ appears in $\widetilde{O}(m/\ell)$, **algOfflineSC** returns a cover $\mathcal{D}$ of size at most $\rho\ell$ using $\widetilde{O}\left(m\,(n/\ell)^{1/(\alpha-1)}\right)$ SetOf queries (Lemma 4.3). By the guarantee of Element Sampling (Lemma 4.1), the number of elements in $\mathcal{U}_{\text{rem}}$ that are not covered by $\mathcal{D}$ is at most $(\ell/n)^{1/(\alpha-1)}|\mathcal{U}_{\text{rem}}|$. Finally, at the end of each inner loop, the algorithm updates the set of uncovered elements $\mathcal{U}_{\text{rem}}$ by using $\widetilde{O}(n\ell)$ EltOf queries. The Feasibility Test which is passed w.h.p. for $\ell \leq \ell_k$ ensures that the final run of **algOfflineSC** performs $\widetilde{O}(m(n/\ell)^{1/(\alpha-1)})$ SetOf queries. Hence, the total number of queries performed in each iteration of the outer loop of **iterSetCover** with parameter $\ell \leq \ell_k$ is $\widetilde{O}\left(m\,(n/\ell)^{1/(\alpha-1)} + n\ell\right)$.

By Lemma 4.4, if $\ell_k \leq u$, then the outer loop of **iterSetCover** is executed for $l \leq \ell \leq \ell_k$ before it terminates. Thus, the total number of queries made by **iterSetCover** is:

$$\sum_{i=\lceil \log_{1+\frac{\varepsilon}{2\alpha\rho}} l\rceil}^{\log_{1+\frac{\varepsilon}{2\alpha\rho}} \ell_k} \widetilde{O}(m\left(\frac{n}{(1+\frac{\varepsilon}{2\alpha\rho})^i}\right)^{\frac{1}{\alpha-1}} + n(1+\frac{\varepsilon}{2\alpha\rho})^i)$$
$$= \widetilde{O}\left(m\left(\frac{n}{l}\right)^{\frac{1}{\alpha-1}}\left(\log_{1+\frac{\varepsilon}{2\alpha\rho}}\frac{\ell_k}{l}\right) + \frac{n\ell_k}{\varepsilon/(\rho\alpha)}\right)$$
$$= \widetilde{O}\left(\frac{1}{\varepsilon}\left(m\left(\frac{n}{l}\right)^{1/(\alpha-1)} + nk\right)\right).$$

Now, we show that the number of sets returned by **iterSetCover** is not more than $(\alpha\rho+\varepsilon)\ell_k$. Set Sampling picks ℓ sets and each run of **algOfflineSC** returns at most $\rho\ell$ sets. Thus the size of the solution returned by **iterSetCover** is at most $(1+(\alpha-1)\rho)\ell_k < (\alpha\rho+\varepsilon)k$.

Next, we prove the main theorem of the section.

smallSetCover(α, ε):
$\texttt{sol} \leftarrow$ **iterSetCover**$(\log n, 1, 1, n)$
$k' \leftarrow |\texttt{sol}|$ ▷ Find a $\rho \log n$ estimate of k.
return **iterSetCover**$(\alpha, \epsilon, \lfloor \frac{k'}{\rho \log n} \rfloor, \lceil k'(1+\frac{\varepsilon}{2\alpha\rho})\rceil)$

Figure 4: The description of the **smallSetCover** algorithm.

Proof of Theorem 4.2. The algorithm **smallSetCover** first finds a $(\rho \log n)$-approximate solution of Set Cover$(\mathcal{U}, \mathcal{F})$, sol, with $\widetilde{O}(m + nk)$ queries by calling **iterSetCover**$(\log n, 1, 1, n)$. Having that $k \leq k' = |\texttt{sol}| \leq (\rho \log n)k$, the algorithm calls **iterSetCover** with α as the approximation factor and $[\lfloor k'/(\rho \log n)\rfloor, \lceil k'(1+\frac{\varepsilon}{2\alpha\rho})\rceil]$ as the range containing k. By Lemma 4.5, the second call to **iterSetCover** in **smallSetCover** returns a $(\alpha\rho+\varepsilon)$-approximate solution of Set Cover$(\mathcal{U}, \mathcal{F})$ using the following number of queries:

$$\widetilde{O}(\frac{1}{\varepsilon}(m(\frac{n}{\frac{k}{\rho\log n}})^{\frac{1}{\alpha-1}} + nk)) = \widetilde{O}(\frac{1}{\varepsilon}(m(\frac{n}{k})^{\frac{1}{\alpha-1}} + nk)).$$

4.3 The largeSetCover Algorithm. The second algorithm, **largeSetCover**, works strictly better than **smallSetCover** for large values of k ($k \geq \sqrt{m}$). The advantage of **largeSetCover** is that it does not need to update the set of uncovered elements at any point and simply avoids the additive nk term in the query complexity bound; the result of Section 5 suggests that the nk term may be unavoidable if one wishes to maintain the uncovered elements. Note that the guarantees of **largeSetCover** is that at the end of the algorithm, w.h.p. the ground set $\mathcal{U}$ is covered.

The algorithm **largeSetCover**, given in Figure 5, first randomly picks $\varepsilon\ell/3$ sets. By Set Sampling (Lemma 4.2), w.h.p. every element that occurs in $\widetilde{\Omega}(m/(\varepsilon\ell))$ sets of $\mathcal{F}$ will be covered by the picked sets. It then solves the Set Cover instance over the elements that occur in $\widetilde{O}(m/(\varepsilon\ell))$ sets of $\mathcal{F}$ by an offline solver of Set Cover using $\widetilde{O}(m/(\varepsilon\ell))$ queries; note that this set of elements may include some already covered elements. In order to get the promised query complexity, **largeSetCover** enumerates the guesses ℓ of the size of an optimal set cover in the decreasing order. The algorithm returns

feasible solutions for $\ell \geq k$ and once it cannot find a feasible solution for ℓ, it returns the solution constructed for the previous guess of k, i.e., $\ell(1+\varepsilon/(3\rho))$. Since **largeSetCover** performs Set Sampling for $\widetilde{O}(\varepsilon^{-1})$ iterations, w.h.p. the total query complexity of **largeSetCover** is $\widetilde{O}(mn/(k\varepsilon^2))$. Note that testing whether the number of occurrences of an element is $\widetilde{O}(m/(\varepsilon\ell))$ only requires a single query, namely SetOf$(e, \frac{cm\log n}{\varepsilon\ell})$.

largeSetCover(ε):
▷ Try all $(1+\frac{\varepsilon}{3\rho})$-approximate gueses of k
for $\ell \in \{(1+\frac{\varepsilon}{3\rho})^i \mid 0 \leq i \leq \log_{1+\frac{\varepsilon}{3\rho}} n\}$
do in the decreasing order:
$\mathsf{rnd}_\ell \leftarrow$ collection of $\frac{\varepsilon\ell}{3}$ sets picked uniformly at random ▷ Set Sampling
$\mathcal{F}_{\mathsf{rare}} \leftarrow \emptyset$ ▷ intersection with *rare* elements
for $e \in \mathcal{U}$ **do**
if e appears in $\leq \frac{cm\log n}{\varepsilon\ell}$ sets **then**
▷ Size Test: SetOf$(e, \frac{cm\log n}{\varepsilon\ell})$
$\mathcal{F}_e \leftarrow$ collection of sets containing e
▷ $\widetilde{O}(\frac{m}{\varepsilon\ell})$ SetOf queries
$\mathcal{F}_{\mathsf{rare}} \leftarrow \mathcal{F}_{\mathsf{rare}} \cup \mathcal{F}_e$, $\quad \mathsf{S} \leftarrow \mathsf{S} \bigcup \{e\}$
$\mathcal{D} \leftarrow$ solution of Set Cover$(\mathsf{S}, \mathcal{F}_{\mathsf{rare}})$ returned by a ρ-approximate black box algorithm
if $|\mathcal{D}| \leq \rho\ell$ **then** $\mathtt{sol} \leftarrow \mathsf{rnd}_\ell \cup \mathcal{D}$
else return $\mathtt{sol}$
▷ solution for the previous value of ℓ

Figure 5: A $(\rho+\varepsilon)$-approximation algorithm for the Set Cover problem. We assume that the algorithm has access to EltOf and SetOf oracles for Set Cover$(\mathcal{U}, \mathcal{F})$, as well as $|\mathcal{U}|$ and $|\mathcal{F}|$.

We now prove the desired performance of **largeSetCover**.

Lemma 4.6. **largeSetCover** *returns a* $(\rho+\varepsilon)$*-approximate solution of* Set Cover$(\mathcal{U}, \mathcal{F})$ *w.h.p.*

Proof. The algorithm **largeSetCover** tries to construct set covers of decreasing sizes until it fails. Clearly, if $k \leq \ell$ then the black box algorithm finds a cover of size at most $\rho\ell$ for any subset of $\mathcal{U}$, because k sets are sufficient to cover $\mathcal{U}$. In other words, the algorithm does not terminate with $\ell \geq k$. Moreover, since the algorithm terminates when ℓ is smaller than k, the size of the set cover found by **largeSetCover** is at most $(\frac{\varepsilon}{3}+\rho)(1+\frac{\varepsilon}{3\rho})\ell < (\frac{\varepsilon}{3}+\rho)(1+\frac{\varepsilon}{3\rho})k < (\rho+\varepsilon)k$.

Lemma 4.7. *The number of queries made by* **largeSetCover** *is* $\widetilde{O}(\frac{mn}{k\varepsilon^2})$.

Proof. The value of ℓ in any *successful* iteration of the algorithm is greater than $k/(\rho+\varepsilon)$; otherwise, the size of the solution constructed by the algorithm is at most $(\rho+\varepsilon)\ell < k$ which is a contradiction.

Set Sampling guarantees that w.h.p. each uncovered element appears in $\widetilde{\Theta}(m/\varepsilon\ell)$ sets and thus the algorithm needs to perform $\widetilde{O}(\frac{mn}{\varepsilon\ell})$ SetOf queries to construct $\mathcal{F}_{\mathsf{rare}}$. Moreover, the number of required queries in the *size test* step is $O(n)$ because we only need one SetOf query per each element in $\mathcal{U}$. Thus, the query complexity of **largeSetCover**(ε) is bounded by

$$\sum_{i=\log_{1+\frac{\varepsilon}{3\rho}}\frac{k}{\rho+\varepsilon}}^{\log_{1+\frac{\varepsilon}{3\rho}} n} \widetilde{O}\left((n+\frac{mn}{\varepsilon(1+\frac{\varepsilon}{3\rho})^i})\right) = \widetilde{O}\left((n+\frac{mn}{\varepsilon k})\log_{1+\frac{\varepsilon}{3\rho}}\frac{n}{k}\right) = \widetilde{O}\left(\frac{mn}{k\varepsilon^2}\right).$$

5 Lower Bound for the Cover Verification Problem

In this section, we give a tight lower bound on a feasibility variant of the Set Cover problem which we refer to as Cover Verification. In Cover Verification$(\mathcal{U}, \mathcal{F}, \mathcal{F}_k)$, besides a collection of m sets $\mathcal{F}$ and n elements $\mathcal{U}$, we are given indices of k sets $\mathcal{F}_k \subseteq \mathcal{F}$, and the goal is to determine whether they are covering the whole universe $\mathcal{U}$ or not. We note that, throughout this section, the parameter k is a candidate for, but not necessarily the value of, the size of the minimum set cover.

A naive approach for this decision problem is to query all elements in the given k sets and then check whether they cover $\mathcal{U}$ or not; this approach requires $O(nk)$ queries. However, in what follows we show that this approach is tight and no *randomized* protocol can decide whether the given k sets cover the whole universe with probability of success at least 0.9 using $o(nk)$ queries.

Theorem 5.1. *Any (randomized) algorithm for deciding whether a given* $k = \Omega(\log n)$ *sets covers all elements with probability of success at least* 0.9*, requires* $\Omega(nk)$ *queries.*

While this lower bound does not directly lead to a lower bound on Set Cover, it suggests that verifying the feasibility of a solution may even be more costly than finding the approximate solution itself; any algorithm bypassing this $\Omega(nk)$ lower bound may not solve Cover Verification as a subroutine.

We prove our lower bound by designing the Yes and No instances that are hard to distinguish, such that for a Yes instance, the union of the given k sets is $\mathcal{U}$, while for a No instance, their union only covers $n-1$ elements. Each Yes instance is indistinguishable from a good fraction of No instances. Thus any algorithm must unavoidably answer incorrectly on half of these

fractions, and fail to reach the desired probability of success.

5.1 Underlying Set Structure. Our instance contains n sets and n elements (so $m = n$), where the first k sets forms $\mathcal{F}_k$, the candidate for the set cover we wish to verify. We first consider the incidence matrix representation, such that the rows represent the sets and the columns represent the elements. We focus on the first n/k elements, and consider a *slab*, composing of n/k columns of the incidence matrix. We define a *basic slab* as the structure illustrated in Figure 6 (for $n = 12$ and $k = 3$), where the cell (i, j) is *white* if $e_j \in S_i$, and is *gray* otherwise. The rows are divided into blocks of size k, where first block, the *query block*, contains the rows whose sets we wish to check for coverage; notice that only the last element is not covered. More specifically, in a basic slab, the query block contains sets $S_1, \ldots, S_{n/k}$, each of which is equal to $\{e_1, \ldots, e_{n/k-1}\}$. The subsequent rows form the *swapper blocks* each consisting of n/k sets. The r^{th} swapper block consists of sets $S_{(r+1)n/k+1}, \ldots, S_{(r+2)n/k}$, each of which is equal to $\{e_1, \ldots, e_{n/k}\} \setminus \{e_r\}$.

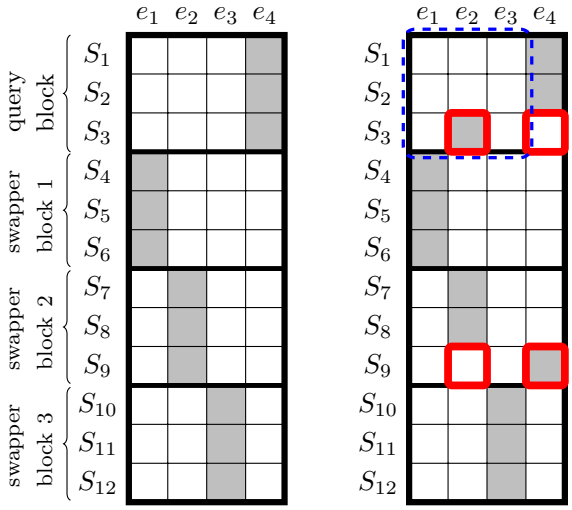

Figure 6: A basic slab and an example of a swapping operation.

We perform one swap in this slab. Consider a parameter (x, y) representing the index of a white cell within the query block. We exchange the color of this white cell with the gray cell on the same row, and similarly exchange the same pair of cells on swapper block y. An example is given in Figure 6; the dashed blue rectangle corresponds to the indices parameterizing possible swaps, and the red squares mark the modified cells. This modification corresponds to a single swap operation; in this example, choosing the index $(3, 2)$ swaps (e_2, e_4) between S_3 and S_9. Observe that there are $k \times (n/k - 1) = n - k$ possible swaps on a single slab, and any single swap allows the query sets to cover all n/k elements.

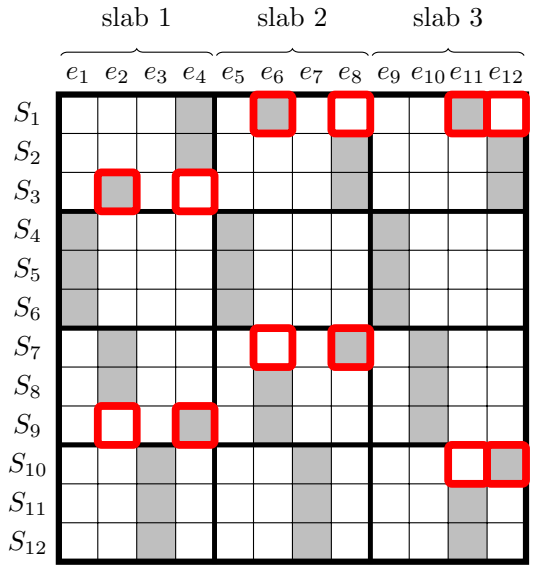

Figure 7: A example structure of a Yes instance; all elements are covered by the first 3 sets.

Lastly, we may create the full instance by placing all k slabs together, as shown in Figure 7, shifting the elements' indices as necessary. The structure of our sets may be specified solely by the swaps made on these slabs. We define the structure of our instances as follows.

- For a Yes instance, we make one random swap on each slab. This allows the first k sets to cover all elements.
- For a No instance, we make one random swap on each slab except for exactly one of them. In that slab, the last element is not covered by any of the first k sets.

Now, to properly define an instance, we must describe our structure via EltOf and SetOf. We first create a temporary instance consisting of k basic slabs, where none of the cells are swapped. Create EltOf and SetOf lists by sorting each list in an increasing order of indices. Each instance from the above construction can then be obtained by applying up to k swaps on this temporary instance.

5.2 Proof of Theorem 5.1. Observe that according to our instance construction, the algorithm may verify, with a single query, whether a certain swap occurs in a certain slab. Namely, it is sufficient to query an entry of EltOf or SetOf that would have been modified by

that swap, and check whether it is actually modified or not. For simplicity, we assume that the algorithm has the knowledge of our construction. Further, without loss of generality, the algorithm does not make multiple queries about the same swap, or make a query that is not corresponding to any swap.

We employ Yao's principle as follows: to prove a lower bound for randomized algorithms, we show a lower bound for any deterministic algorithm on a fixed distribution of input instances. Let $s = n - k$ be the number of possible swaps in each slab; assume $s = \Theta(n)$. We define our distribution of instances as follows: each of the s^k possible Yes instances occurs with probability $1/(2s^k)$, and each of the ks^{k-1} possible No instances occurs with probability $1/(2ks^{k-1})$. Equivalently speaking, we create a random Yes instance by making one swap on each basic slab. Then we make a coin flip: with probability $1/2$ we pick a random slab and undo the swap on that slab to obtain a No instance; otherwise we leave it as a Yes instance. To prove by contradiction, assume there exists a deterministic algorithm that solves the Cover Verification problem over this distribution of instances with $r = o(sk)$ queries.

Consider the Yes instances portion of the distribution, and observe that we may alternatively interpret the random process generating them as as follows. For each slab, one of its s possible swaps is chosen uniformly at random. This condition again follows the scenario considered in Section 3.2: we are given k urns (slabs) of each consisting of s marbles (possible swap locations), and aim to draw the red marble (swapped entry) from a large fraction of these urns. Following the proof of Lemmas 3.6-3.7, we obtain that if the total number of queries made by the algorithm is less than $(1 - \frac{3}{b})\frac{sk}{b}$, then with probability at least 0.99, the algorithm will not see any swaps from at least $\frac{k}{b}$ slabs.

Then, consider the corresponding No instances obtained by undoing the swap in one of the slabs of the Yes instance. Suppose that the deterministic algorithm makes less than $(1 - \frac{3}{b})\frac{sk}{b}$ queries, then for a fraction of 0.99 of all possible tuples $\mathcal{T}$, the output of the Yes instance is the same as the output of $\frac{1}{b}$ fraction of No instances, namely when the slab containing no swap is one of the $\frac{k}{b}$ slabs that the algorithm has not detected a swap in the corresponding Yes instance; the algorithm must answer incorrectly on half of the corresponding weight in our distribution of input instances. Thus the probability of success for any algorithm with less than $(1 - \frac{3}{b})\frac{sk}{b}$ queries is at most

$$1 - \mathbf{Pr}\left[|\mathcal{T}_{\text{high}}| \geq (1 - \frac{2}{b})k\right](\frac{1}{b})(\frac{1}{2}) \leq 1 - \frac{0.495}{b} < 0.9,$$

for a sufficiently small constant $b > 3$ (e.g. $b = 4$). As $s = \Theta(n)$ and by Yao's principle, this implies the lower bound of $\Omega(nk)$ for the Cover Verification problem.

Acknowledgement

We would like to thank Jonathan Ullman for helpful discussions.

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] 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.

[3] 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.

[4] M. Bateni, H. Esfandiari, and V. S. Mirrokni. Distributed coverage maximization via sketching. *CoRR*, abs/1612.02327, 2016.

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

[6] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proc. 47th Annu. ACM Sympos. Theory Comput.* (STOC), pages 173–182, 2015.

[7] 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.

[8] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.

[9] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *Proc. 19th Int. Conf. World Wide Web* (WWW), pages 231–240, 2010.

[10] 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 of *Lect. Notes in Comp. Sci.*, pages 484–498, 2014.

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

[12] 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.

[13] U. Feige. A threshold of ln n for approximating set

cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[14] A. Goel, M. Kapralov, and S. Khanna. Perfect matchings in $O(n \log n)$ time in regular bipartite graphs. *SIAM Journal on Computing*, 42(3):1392–1404, 2013.

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

[16] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. *Euro. J. Oper. Res.*, 101(1):81–92, 1997.

[17] 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), 2016.

[18] P. Indyk, S. Mahabadi, R. Rubinfeld, J. Ullman, A. Vakilian, and A. Yodpinyanee. Fractional set cover in the streaming model. *Approximation, Randomization, and Combinatorial Optimization* (APPROX/RANDOM), pages 198–217, 2017.

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

[20] 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.

[21] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *Proc. 17th ACM-SIAM Sympos. Discrete Algs.* (SODA), 2006.

[22] 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.

[23] S. Marko and D. Ron. Distance approximation in bounded-degree and general sparse graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 475–486. Springer, 2006.

[24] A. McGregor and H. T. Vu. Better streaming algorithms for the maximum coverage problem. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 22:1–22:18, 2017.

[25] V. S. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *Proc. 47th Annu. ACM Sympos. Theory Comput.* (STOC), pages 153–162, 2015.

[26] 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. Springer, 2012.

[27] R. Motwani and P. Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[28] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 327–336. IEEE, 2008.

[29] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proc. 23rd ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1123–1131, 2012.

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

[31] 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.

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

[33] 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.

A Generalized Lower Bounds for the Set Cover Problem

In this section we generalize the approach of Section 3 and prove our main lower bound result (Theorem 3.1) for the number of queries required for approximating with factor α the size of an optimal solution to the Set Cover problem, where the input instance contains m sets, n elements, and a minimum set cover of size k. The structure of our proof is largely the same as the simplified case, but the definitions and the details of our analysis will be more complicated. The size of the minimum set cover of the median instance will instead be at least $\alpha k + 1$, and **genModifiedInst** reduces this down to k. We now aim to prove the following statement which implies the lower bound in Theorem 3.1.

THEOREM A.1. *Let k be the size of an optimal solution of I^* such that $1 < \alpha \leq \log n$ and $2 \leq k \leq \left(\frac{n}{16\alpha \log m}\right)^{\frac{1}{4\alpha+1}}$. Any algorithm that distinguishes whether the input instance is I^* or belongs to $\mathcal{D}(I^*)$ with probability of success at least $2/3$ requires $\widetilde{\Omega}(m(\frac{n}{k})^{1/(2\alpha)})$ queries.*

A.1 Construction of the Median Instance I^*.

Let $\mathcal{F}$ be a collection of m sets such that independently for each set-element pair (S, e), S contains e with probability $1 - p_0$, where we modify the probability to $p_0 = \left(\frac{8(\alpha k+2)\log m}{n}\right)^{1/(\alpha k)}$. We start by proving some inequalities involving p_0 that will be useful later on, which hold for any k in the assumed range.

LEMMA A.1. *For $2 \leq k \leq \left(\frac{n}{16\alpha \log m}\right)^{\frac{1}{4\alpha+1}}$, we have*

that
(a) $1 - p_0 \geq p_0^{k/4}$,
(b) $p_0^{k/4} \leq 1/2$,
(c) $\frac{p_0^k}{(1-p_0)^2} \leq \left(\frac{8(\alpha k+2)\log m}{n}\right)^{\frac{1}{2\alpha}}$.

Proof. Recall as well that $\alpha > 1$. In the given range of k, we have $k^{4\alpha} \leq \frac{n}{16\alpha k \log m} \leq \frac{n}{8(\alpha k+2)\log m}$ because $k\alpha \geq 2$. Thus

$$p_0 = \left(\frac{8(\alpha k+2)\log m}{n}\right)^{\frac{1}{\alpha k}} \leq \left(\frac{1}{k^{4\alpha}}\right)^{\frac{1}{\alpha k}} = k^{-4/k}.$$

Next, rewrite $k^{-4/k} = e^{-\frac{4\ln k}{k}}$ and observe that $\frac{4\ln k}{k} \leq \frac{4}{e} < 1.5$. Since $e^{-x} \leq 1 - \frac{x}{2}$ for any $x < 1.5$, we have $p_0 \leq e^{-\frac{4\ln k}{k}} < 1 - \frac{2\ln k}{k}$. Further, $p_0^{k/4} \leq e^{-\ln k} = 1/k$. Hence $p_0 + p_0^{k/4} \leq 1 - \frac{2\ln k}{k} + \frac{1}{k} \leq 1$, implying the first statement.

The second statement easily follows as $p_0^{k/4} \leq 1/k \leq 1/2$ since $k \geq 2$. For the last statement, we make use of the first statement:

$$\frac{p_0^k}{(1-p_0)^2} \leq \frac{p_0^k}{(p_0^{k/4})^2} = p_0^{k/2} = \left(\frac{8(\alpha k+2)\log m}{n}\right)^{\frac{1}{2\alpha}}$$

which completes the proof of the lemma.

Next, we give the new, generalized definition of median instances.

Definition A.1. (Median instance) *An instance of* Set Cover, $I = (\mathcal{U}, \mathcal{F})$, *is a* median instance *if it satisfies all the following properties.*
(a) No αk sets cover all the elements. (The size of its minimum set cover is greater than αk.)
(b) The number of uncovered elements of the union of any k sets is at most $2np_0^k$.
(c) For any pair of elements e, e', the number of sets $S \in \mathcal{F}$ s.t. $e \in S$ but $e' \notin S$ is at least $(1-p_0)p_0 m/2$.
(d) For any collection of k sets $S_1, \cdots, S_k$, $|S_k \cap (S_1 \cup \cdots \cup S_{k-1})| \geq (1-p_0)(1-p_0^{k-1})n/2$.
(e) For any collection of $k+1$ sets $S, S_1, \cdots, S_k$, $|(S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus S| \leq 2p_0(1-p_0)(1-p_0^{k-1})n$.
(f) For each element, the number of sets that do not contain the element is at most $(1+\frac{1}{k})p_0 m$.

Lemma A.2. *For $k \leq \min\{\sqrt{\frac{m}{27\ln m}}, (\frac{n}{16\alpha\log m})^{\frac{1}{4\alpha+1}}\}$, there exists a median instance I^* satisfying all the median properties from Definition A.1. In fact, most of the instances constructed by the described randomized procedure satisfy the median properties.*

Proof. The lemma follows from applying the union bound on the results of Lemmas A.3–A.8.

The proofs of the Lemmas A.3–A.8 follow from standard applications of concentration bounds. See the full version of this paper for detailed proofs.

Lemma A.3. *With probability at least $1 - m^{-2}$ over $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$, the size of the minimum set cover of the instance $(\mathcal{F}, \mathcal{U})$ is at least $\alpha k + 1$.*

Lemma A.4. *With probability at least $1 - m^{-2}$ over $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$, any collection of k sets has at most $2np_0^k$ uncovered elements.*

Lemma A.5. *Suppose that $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$ and let e, e' be two elements in $\mathcal{U}$. Given $k \leq \left(\frac{n}{16\alpha\log m}\right)^{\frac{1}{4\alpha+1}}$, with probability at least $1 - m^{-2}$, the number of sets $S \in \mathcal{F}$ such that $e \in S$ but $e' \notin S$ is at least $mp_0(1-p_0)/2$.*

Lemma A.6. *Suppose that $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$ and let $S_1, \cdots, S_k$ be k different sets in $\mathcal{F}$. Given $k \leq \left(\frac{n}{16\alpha\log m}\right)^{\frac{1}{4\alpha+1}}$, with probability at least $1 - m^{-2}$, $|S_k \cap (S_1 \cup \cdots \cup S_{k-1})| \geq (1-p_0)(1-p_0^{k-1})n/2$.*

Lemma A.7. *Suppose that $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$ and let $S_1, \cdots, S_k$ and S be $k+1$ different sets in $\mathcal{F}$. Given $k \leq \left(\frac{n}{16\alpha\log m}\right)^{\frac{1}{4\alpha+1}}$, with probability at least $1 - m^{-2}$, $|(S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus S| \leq 2p_0(1-p_0)(1-p_0^{k-1})n$.*

Lemma A.8. *Given that $k \leq \left(\frac{n}{16\alpha\log m}\right)^{\frac{1}{4\alpha+1}}$, for each element, the number of sets that do not contain the element is at most $(1+\frac{1}{k})p_0 m$.*

A.2 Distribution $\mathcal{D}(I^*)$ of the Modified Instances Derived from I^*. Fix a median instance I^*. We now show that we may perform $\tilde{O}(n^{1-1/\alpha}k^{1/\alpha})$ swap operations on I^* so that the size of the minimum set cover in the modified instance becomes k. So, the number of queries to EltOf and SetOf that induce different answers from those of I^* is at most $\tilde{O}(n^{1-1/\alpha}k^{1/\alpha})$. We define $\mathcal{D}(I^*)$ as the distribution of instances I' that is generated from a median instance I^* by **genModifiedInst**(I^*) given below in Figure 8. The main difference from the simplified version are that we now select k different sets to turn them into a set cover, and the swaps may only occur between S_k and the candidates.

Lemma A.9. *The procedure* **genModifiedInst** *is well-defined under the precondition that the input instance I^* is a median instance.*

Proof. To carry out the algorithm, we must ensure that the number of the initially uncovered elements is at

genModifiedInst$(I^* = (\mathcal{U}, \mathcal{F}))$:
  $\mathcal{M} \leftarrow \emptyset$
  pick k *different* sets $S_1, \cdots S_k$ from $\mathcal{F}$ uniformly at random
  for each $e \in \mathcal{U} \setminus (S_1 \cup \cdots \cup S_k)$ **do**
    pick $e' \in (S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus \mathcal{M}$ uniformly at random
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{ee'\}$
    pick a random set S in $\mathsf{Candidate}(e, e')$
    $\mathsf{swap}(e, e')$ between S, S_k

Figure 8: The procedure of constructing a modified instance of I^*.

most that of the elements covered by both S_k and some other set from $S_1, \ldots, S_{k-1}$. Since I^* is a median instance, by properties (b) and (d) from Definition A.1, these values satisfy $|\mathcal{U} \setminus (S_1 \cup \cdots \cup S_k)| \le 2p_0^k n$ and $|S_k \cap (S_1 \cup \cdots \cup S_{k-1})| \ge (1-p_0)(1-p_0^{k-1})n/2$, respectively. By Lemma A.1, $p_0^{k/4} \le 1/2$. Using this and Lemma A.1 again,

$$\begin{aligned}(1-p_0)(1-p_0^{k-1})n/2 &\ge p_0^{k/4} \cdot p_0^{k/4} \cdot n/2 \\ &\ge p_0^{k/2} n/2 \ge 2p_0^k n.\end{aligned}$$

That is, in our construction there are sufficiently many possible choices for e' to be matched and swapped with each uncovered element e. Moreover, since I^* is a median instance, $|\mathsf{Candidate}(e, e')| \ge (1-p_0)p_0 m/2$ (by property (c)), and there are plenty of candidates for each swap.

A.2.1 Bounding the Probability of Modification. Similarly to the simplified case, define $P_{\mathsf{Elt-Set}} : \mathcal{U} \times \mathcal{F} \to [0,1]$ as the probability that an element is swapped by a set, and upper bound it via the following lemma.

LEMMA A.10. *For any $e \in \mathcal{U}$ and $S \in \mathcal{F}$, $P_{\mathsf{Elt-Set}}(e, S) \le \frac{64p_0^k}{(1-p_0)^2 m}$ where the probability is taken over the random choices of $I' \sim \mathcal{D}(I^*)$.*

Proof. Let $S_1, \ldots, S_k$ denote the first k sets picked (uniformly at random) from $\mathcal{F}$ to construct a modified instance of I^*. For each element e and a set S such that $e \in S$ in the basic instance I^*,

$$\begin{aligned}P_{\mathsf{Elt-Set}}(e,S) &= \mathbf{Pr}[S = S_k] \cdot \mathbf{Pr}\big[e \in \cup_{i\in[k-1]} S_i \mid e \in S_k\big] \\ &\cdot \mathbf{Pr}\big[e \text{ matches to } \mathcal{U} \setminus (\cup_{i\in[k]} S_i) \mid e \in S_k \cap (\cup_{i\in[k-1]} S_i)\big] \\ &+ \mathbf{Pr}[S \notin \{S_1, \ldots, S_k\}] \cdot \mathbf{Pr}\big[e \in S \setminus (\cup_{i\in[k]} S_i) \mid e \in S\big] \\ &\cdot \mathbf{Pr}[S \text{ swaps } e \text{ with } S_k \mid e \in S \setminus (S_1 \cup \cdots \cup S_k)],\end{aligned}$$

where all probabilities are taken over $I' \sim \mathcal{D}(I^*)$. Next we bound each of the above six terms. Clearly, since we choose the sets $S_1, \cdots, S_k$ randomly, $\mathbf{Pr}[S = S_k] = 1/m$. We bound the second term by 1. Next, by properties (b) and (d) of median instances, the third term is at most

$$\frac{|\mathcal{U} \setminus (\cup_{i\in[k]} S_i)|}{|S_k \cap (\cup_{i\in[k-1]} S_i)|} \le \frac{2p_0^k n}{(1-p_0)(1-p_0^{k-1})\frac{n}{2}} \le \frac{4p_0^k}{(1-p_0)^2}.$$

We bound the fourth term by 1. Let d_e denote the number of sets in $\mathcal{F}$ that do not contain e. Using property (f) of median instances, the fifth term is at most

$$\begin{aligned}\frac{d_e(d_e-1)\cdots(d_e-k+1)}{(m-1)(m-2)\cdots(m-k)} &\le \left(\frac{d_e}{m-1}\right)^k \\ &\le \left(\frac{(1+1/k)p_0 m}{m(1-\frac{1}{k+1})}\right)^k \le e^2 p_0^k,\end{aligned}$$

Finally for the last term, note that by symmetry, each pair of matched elements ee' is picked by **genModifiedInst** equiprobably. Thus, for any $e \in S \setminus (S_1 \cup \cdots \cup S_k)$, the probability that each element $e' \in S_k \cap (S_1 \cup \cdots \cup S_{k-1})$ is matched to e is $\frac{1}{|S_k \cap (S_1 \cup \cdots \cup S_{k-1})|}$. By properties (c)-(e) of median instances, the last term is at most

$$\begin{aligned}&\sum_{e' \in (S_k \cap (\cup_{i\in[k-1]} S_i)) \setminus S} \mathbf{Pr}[ee' \in \mathcal{M}] \cdot \mathbf{Pr}[(S, S_k) \text{ swap } (e, e')] \\ &\le |(S_k \cap (\cup_{i\in[k-1]} S_i)) \setminus S| \cdot \frac{1}{|S_k \cap (\cup_{i\in[k-1]} S_i)|} \\ &\qquad \cdot \frac{1}{|\mathsf{Candidate}(e, e')|} \\ &\le 2p_0(1-p_0)(1-p_0^{k-1})n \cdot \frac{1}{(1-p_0)(1-p_0^{k-1})n/2} \\ &\qquad \cdot \frac{1}{p_0(1-p_0)m/2} \\ &\le \frac{8}{(1-p_0)m}\end{aligned}$$

Therefore,

$$\begin{aligned}P_{\mathsf{Elt-Set}}(e,S) &\le \frac{1}{m} \cdot 1 \cdot \frac{4p_0^k}{(1-p_0)^2} + 1 \cdot e^2 p_0^k \cdot \frac{8}{(1-p_0)m} \\ &\le \frac{4p_0^k}{(1-p_0)^2} + \frac{60p_0^k}{(1-p_0)m} \le \frac{64p_0^k}{(1-p_0)^2 m}.\end{aligned}$$

A.3 Proof of Theorem A.1. The remaining part of our proof follows that of the simplified version almost exactly.

Proof of Theorem A.1. Applying the same argument as that of Lemma 3.4, we derive that the probability that

$\mathcal{A}$ returns different outputs on I^* and I' is at most

$$\begin{aligned}\mathbf{Pr}[\mathcal{A}(I^*) \neq \mathcal{A}(I')] &\leq \sum_{t=1}^{|Q|} \mathbf{Pr}[\mathsf{ans}_{I^*}(q_t) \neq \mathsf{ans}_{I'}(q_t)] \\ &\leq \sum_{t=1}^{|Q|} P_{\mathsf{Elt-Set}}(e(q_t), S(q_t)) \\ &\leq \frac{64p_0^k}{m(1-p_0)^2}|Q|,\end{aligned}$$

via the result of Lemma A.10. Then, over the distribution in which we applied Yao's lemma, we have

$$\begin{aligned}\mathbf{Pr}[\mathcal{A} \text{ succeeds}] &\leq 1 - \frac{1}{2}\mathbf{Pr}_{I' \sim \mathcal{D}(I^*)}[\mathcal{A}(I^*) = \mathcal{A}(I')] \\ &\leq 1 - \frac{1}{2}\left(1 - \frac{64p_0^k}{m(1-p_0)^2}|Q|\right) \\ &= \frac{1}{2} + \frac{32p_0^k}{m(1-p_0)^2}|Q| \\ &\leq \frac{1}{2} + \frac{32}{m}\left(\frac{8(k\alpha+2)\log m}{n}\right)^{\frac{1}{2\alpha}}|Q|\end{aligned}$$

where the last inequality follows from Lemma A.1. Thus, if the number of queries made by $\mathcal{A}$ is less than $\frac{m}{192}(\frac{n}{8(k\alpha+2)\log m})^{1/(2\alpha)}$, then the probability that $\mathcal{A}$ returns the correct answer over the input distribution is less than $2/3$ and the proof is complete.