New Directions in Streaming Algorithms

by

Ali Vakilian

B.S., Sharif University of Technology (2011)
M.S., University of Illinois at Urbana-Champaign (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Author .....
Department of Electrical Engineering and Computer Science
August 30, 2019

Certified by .....
Erik D. Demaine
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by .....
Piotr Indyk
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

New Directions in Streaming Algorithms

by

Ali Vakilian

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Large volumes of available data have led to the emergence of new computational models for data analysis. One such model is captured by the notion of streaming algorithms: given a sequence of N items, the goal is to compute the value of a given function of the input items by a small number of passes and using a sublinear amount of space in N. Streaming algorithms have applications in many areas such as networking and large scale machine learning. Despite a huge amount of work on this area over the last two decades, there are multiple aspects of streaming algorithms that remained poorly understood, such as (a) streaming algorithms for combinatorial optimization problems and (b) incorporating modern machine learning techniques in the design of streaming algorithms.

In the first part of this thesis, we will describe (essentially) optimal streaming algorithms for set cover and maximum coverage, two classic problems in combinatorial optimization. Next, in the second part, we will show how to augment classic streaming algorithms of the frequency estimation and low-rank approximation problems with machine learning oracles in order to improve their space-accuracy tradeoffs. The new algorithms combine the benefits of machine learning with the formal guarantees available through algorithm design theory.

Thesis Supervisor: Erik D. Demaine
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Piotr Indyk
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to thank my advisors Erik Demaine and Piotr Indyk. I had the chance to learn from Erik while before joining MIT through his lectures on "Introduction to Algorithms" when I was a freshman at Sharif UT. His inspiring style of teaching played a significant role on my passion to pursue TCS in grad school. For me, it was very valuable to have the opportunity to work with Erik. Besides, Erik gave me the freedom to explore different areas and pursue my interests; though always being there to support and advise. The main topic of this thesis, *streaming algorithms*, and all results in this thesis are in collaboration with Piotr Indyk. His guidance and supervision has shaped my academic character and his support during my PhD studies were significant both academic-wise and social-wise. He always had time for me to discuss both research and non-research questions and his continuous passion for science, research and mentoring young researchers will be always a perfect example for me.

Next, I would like to thank Ronitt Rubinfeld who kindly accepted to be in my thesis committee. Ronitt introduced me to other very related areas of research in massive data analysis, namely Local Computation Algorithms and Sublinear Algorithms both through her courses and collaborations. Besides, I benefited a lot from her advice during my PhD years. I also would like to thank my MS advisor at UIUC, Chandra Chekuri. My first experience in doing research in TCS was in collaboration with Chandra and I will be always grateful of his support and supervision. He always had time for me to discuss any topic even after my graduation from UIUC.

During my years in grad school, I had the chance to collaborate with many wonderful researchers and this thesis would not be there without their collaboration. I would like to thank all my co-authors: Anders Aamand, Arturs Backurs, Chandra Chekuri, Yodsawalai Chodpathumwan, Erik Demaine, Alina Ene, Timothy Goodrich, Christoph Gunru, Sariel Har-Peled, Chen-Yu Hsu, Piotr Indyk, Dina Katabi, Kyle Kloster, Nitish Korula, Brian Lavallee, Quanquan Liu, Sepideh Mahabadi, Slobodan Mitrović, Amir Nayyeri, Krzysztof Onak, Merav Parter, Ronitt Rubinfeld, Baruch Schieber, Blair Sullivan, Arash Termehchy, Jonathan Ullman, Andrew van der Poel, Tal Wagner, Marianne Winslett, David Woodruff,

Anak Yodpinyanee and Yang Yuan.

Thanks to all members in the Theory of Computation group and in particular Maryam Aliakbarpour, Arturs, Mohammad Bavarian, Dylan Foster, Mohsen Ghaffari, Themis Gouleakis, Siddhartha Jayanti, Gautam Kamath, Pritish Kamath, William M. Leiserson, Quanquan, Sepideh, Saeed Mehraban, Slobodan, Cameron Musco, Chirstoph Musco, Merav, Madalina Persu, Ilya Razenshteyn, Nicholas Schiefer, Ludwig Schmidt, Adam Sealfon, Tal, Anak, Yang and Henry Yuen, I had great time at MIT. Being away from home, thanks to my friends in Boston area I had wonderful years during my PhD. I would like to thank all my friends and in particular those at MIT: Ehsan, Ardavan, Asma, Maryam, Fereshteh, Elaheh, Sajjad, Mina, Alireza, Amir, Mehrdad, Zahra and Ali.

Last but not least, I would like to thank my family. My parents, Amir and Maryam, were always sources of inspiration for me and I am thankful of all their supports and efforts for me since my childhood. Words are not enough to acknowledge what they have done for me. I also would like to thank my siblings Mohsen and Fatemeh who always care about me. Mohsen was my very first mentor since elementary school and had a significant role in shaping my interest in Math and CS. I have benefited a lot from his advice and helps. Mohsen and Fatemeh, thanks for being so perfect! Sepideh, my beloved wife, is the most valuable gift I had from God during my PhD at MIT. Besides our several collaborations on different projects that contribute to half of this thesis, I was very fortunate to have her honest and unconditional support in my ups and downs during my PhD.

"In the name of God, the most gracious, the most merciful"

To my beloved family

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent decades, massive datasets arose in numerous application areas such as genomics, finance, social networks and Internet of things. The main challenge of massive datasets is that the traditional data-processing architectures are not capable of analyzing them efficiently. This state of affairs has led to the emergence of new computational models for data analysis. In particular, since a large fraction of datasets are generated as a *stream*, algorithms for such data have been studied in the *streaming algorithms* model extensively. Formally, given a sequence of N items, the goal is to compute the value of a given function of the input items by a small number of passes and using a sublinear amount of space in N.

In this thesis, we advance the development of streaming algorithms in two new directions: (a) designing streaming algorithms for *new classes* of problems and (b) designing streaming algorithms using *modern machine learning approaches*.

The streaming model has been studied since early 80s. The focus of the early work was on processing numerical data such as estimating heavy hitters, number of distinct elements and quantiles [74, 35, 13, 129]. Over the last two decades, there has been a significant body of work on designing streaming algorithms for *discrete optimization* and in particular graph problems [102, 143, 73]. However, until recently not much was known about the complexity of *coverage* problems, which is a well-studied class of problems in discrete optimization, in the streaming setting. In the first part of this thesis (Chapters 2-5), we provide optimal streaming algorithms for several variants of set cover and maximum coverage problems, two

well-studied problems in discrete optimization.

Since the classical algorithms have formal worst-case guarantees, they tend to work equally well on all inputs. However, in most applications, the underlying data has certain properties that, if discovered and harnessed, could enable much better performance. Inspired by the success of machine learning, in the second part of this thesis (Chapters 6-7), we design such "learning-based" algorithms for two fundamental tasks in data analysis: *frequency estimation* and *low-rank approximation*. We remark that these are the *first* learning-based streaming algorithms. For both of these problems, we design learning-based algorithms that, in theory and practice, achieve better performance (e.g., space complexity or approximation quality) than the existing algorithms, as long as the input stream has certain patterns. At the same time, the algorithms (usually) retain the worst-case guarantees of the existing algorithms for these problems even on adversarial inputs.

1.1. Streaming Algorithms for Coverage Problems

A fundamental class of problems in the area of combinatorial optimization involves minimizing a given *cost* function under a set of *covering* constraints. Perhaps the most well-studied problem in this family is *set cover*: given a collection $\mathcal{S}$ of sets whose union is $[n]$, the goal is to identify the smallest sub-collection of $\mathcal{S}$ whose union is $[n]$. In the presence of massive data, new techniques are required to solve even this classic problem.

The first part of this thesis is devoted to the design of efficient sublinear (space or time) algorithms for Set Cover and a closely related problem Max k-Cover, here collectively called *coverage problem*, that have in many areas, including operations research, machine learning, information retrieval and data mining [86, 155, 48, 2, 115, 26, 117].

Although both Set Cover and Max k-Cover are NP-complete, a natural greedy algorithm which iteratively picks the "best" remaining set is provably the optimal algorithm for these problems unless P = NP [72, 152, 14, 138, 63] and it is widely used in practice. The algorithm often finds solutions that are very close to optimal. Unfortunately, due to its sequential nature, this algorithm does not scale very well to massive data sets (e.g., see Cormode et al. [56] for an experimental evaluation). This difficulty has motivated a considerable research effort whose goal was to design algorithms that are capable of handling large data efficiently on

modern architectures. Of particular interest are *data stream* algorithms, which compute the solution using only a small number of sequential passes over the data using a limited memory. The study of *streaming* Set Cover problem initiated in [155] and since then there has been a large body of work on coverage problems in massive data analysis models and in particular streaming model [67, 61, 42, 97, 20, 29, 17, 105, 106, 107, 18, 123, 131, 17, 29, 19, 147, 5].

1.1.1. Streaming Algorithms for Set Cover

In the Set Cover problem, given a ground set of n elements $\mathcal{U} = \{e_1, \cdots, e_n\}$, and a family of m sets $\mathcal{F} = \{S_1, \ldots, S_m\}$ where $m \geq n$, the goal is to select a subset $\mathcal{I} \subseteq \mathcal{F}$ such that $\mathcal{I}$ *covers* $\mathcal{U}$ and the number of the sets in $\mathcal{I}$ is as small as possible. In the *streaming* Set Cover problem as introduced in [155], the set of elements $\mathcal{U}$ is stored in the memory in advance; the sets $S_1, \cdots, S_m$ are stored consecutively in a read-only repository and an algorithm can access the sets only by performing sequential scans of the repository. However, the amount of read-write memory available to the algorithm is limited, and is smaller than the input size (which could be as large as mn). The objective is to design efficient approximation algorithms for the Set Cover problem that performs few passes over the data, and uses as little memory as possible. In Chapter 2, we present an efficient streaming algorithm for this problem which has been proved to be optimal [17].

1.1.2. Streaming Algorithms for Fractional Set Cover

The LP relaxation of Set Cover is one of the well-studied mathematical programs in approximation algorithms design. It is a continuous relaxation of the problem where each set $S \in \mathcal{F}$ can be selected "fractionally", i.e., assigned a number x_S from $[0, 1]$, such that for each element e its "fractional coverage" $\sum_{S:e \in S} x_S$ is at least 1, and the sum $\sum_S x_S$ is minimized.

To the best of our knowledge, it is not known whether there exists an efficient and accurate algorithm for this problem that uses only a logarithmic (or even a *polylogarithmic*) number of passes. This state of affairs is perhaps surprising, given the many recent developments on fast LP solvers [119, 174, 124, 12, 11, 167]. The only prior results on streaming Packing/Covering LPs were presented in [6], which studied the LP relaxation of Maximum Matching.

In Chapter 3, we present the first $(1 + \varepsilon)$-approximation algorithm for the fractional Set Cover in the streaming model with constant number of passes and using sublinear amount

of space.

1.1.3. Sublinear Algorithms for Set Cover

Another related natural question to streaming Set Cover is the following: *"is it possible to solve minimum set cover in sub-linear time?"* This question was previously addressed in [145, 172], 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 $k \cdot \ln n + \varepsilon n$), 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, [119] (building on [84]) 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 randomized rounding yields an $O(\log n)$-approximate solution to Set Cover with the same complexity.

In Chapter 4, 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 [119] 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.

1.1.4. Streaming Algorithms for Maximum Coverage

In Max k-Cover, given a ground set $\mathcal{U}$ of n elements, a family of m sets $\mathcal{F}$ (each subset of $\mathcal{U}$), and a parameter k, the goal is to select k sets in $\mathcal{F}$ whose union has the largest cardinality. Moreover, Max k-Cover is an important problem in submodular maximization.

The initial algorithms were developed in the *set arrival* model, where the input sets are listed contiguously. This restriction is natural from the perspective of submodular optimiza-

[1] The method can be further improved to $\widetilde{O}(m + nk)$ (N. Young, personal communication).
[2] Note that polynomial *time* algorithms with sub-logarithmic approximation are unlikely to exist.

tion, but limits the applicability of the algorithms[3]. Avoiding this limitation can be difficult, as streaming algorithms can no longer operate on sets as "unit objects". As a result, the first maximum coverage algorithm for the general *edge arrival* model, where pairs of (set, element) can arrive in arbitrary order, have been developed recently. In particular [29] was first to (explicitly) present a one-pass algorithm with space linear in m and constant approximation factor. We remark that many of the prior bounds (both upper and lower bounds) on set cover and max k-cover problems in set-arrival streams also work in edge arrival streams (e.g. [61, 97, 20, 131, 17, 105]).

A particularly interesting line of research in set arrival streaming set cover and max k-cover is to design efficient algorithms that only use $\widetilde{O}(n)$ space [155, 22, 67, 42, 131]. Previous work have shown that we can adopt the existing greedy algorithm of Max k-Cover to achieve constant factor approximation in $\widetilde{O}(n)$ space [155, 22] (which later improved to $\widetilde{O}(k)$ by [131]). However, the complexity of the problem in the "low space" regime is very different in edge-arrival streams: [29] showed that as long as the approximation factor is a constant, any algorithm must use $\Omega(m)$ space. Still, our understanding of approximation/space tradeoffs in the general case is far from complete. In Chapter 5, we provide tight bounds for the approximation/space tradeoffs of Max k-Cover in the general edge-arrival streams.

1.2. Learning-Based Streaming Algorithms

Classical algorithms are best known for providing formal guarantees over their performance, but often fail to leverage useful patterns in their input data to improve their output. However, in most applications, the underlying data has certain properties that can lead to much better performance if exploited by algorithms. On the other hand, machine learning approaches (in particular, deep learning models) are highly successful at capturing and utilizing complex data patterns, but often lack formal error bounds. The last few years have witnessed a growing effort to bridge this gap and introduce algorithms that can adapt to data properties while delivering worst case guarantees. For example, "learning-based" models have been integrated into the design of data structures [120, 137, 91], online algorithms [128, 151, 81],

[3]For example, consider a situation where the sets correspond to neighborhoods of vertices in a directed graph. Depending on the input representation, for each vertex, either the ingoing edges or the outgoing edges might be placed non-contiguously.

graph optimization [59], similarity search [156, 169] and compressive sensing [33]. This learning-based approach to algorithm design has attracted a considerable attention due to its potential to significantly improve the efficiency of some of the most widely used algorithmic tasks.

Indeed, many applications involve processing streams of data (e.g., videos, data logs, customer activities and social media) by executing the same algorithm on an hourly, daily or weekly basis. These data sets are typically not "random" or "worst-case"; instead, they come from some distribution with useful properties which does not change rapidly from execution to execution. This makes it possible to design better *streaming* algorithms tailored to the specific data distribution, trained on past instances of the problem. In the second part of this thesis, we developed first learning-based algorithms in the streaming model and design such algorithms for two basic problems in this area *frequency estimation* (in Chapter 6) and *low-rank approximation* (in Chapter 7). For both of these problems, we design learning-based algorithms that empirically (and provably) achieve better performance (e.g., space complexity or approximation quality) compared to the existing algorithms if the input stream has certain patterns while achieving the worst-case guarantees of the existing algorithms for these problems even on adversarial inputs.

1.2.1. Learning-Based Algorithms for Frequency Estimation

Estimating the frequencies of elements in a data stream is one of the most fundamental subroutines in data analysis. It has applications in many areas of machine learning, including feature selection [4], ranking [66], semi-supervised learning [164] and natural language processing [82]. It has been also used for network measurements [70, 175, 127] and security [158]. Frequency estimation algorithms have been implemented in popular data processing libraries, such as Algebird at Twitter [36]. They can answer practical questions like: what are the most searched words on the Internet? or how much traffic is sent between any two machines in a network?

The frequency estimation problem is formalized as follows: given a sequence S of elements from some universe U, for any element $i \in U$, estimate f_i, the number of times i occurs in S. If one could store all arrivals from the stream S, one could sort the elements and compute

their frequencies. However, in big data applications, the stream is too large (and may be infinite) and cannot be stored. This challenge has motivated the development of *streaming algorithms*, which read the elements of S in a single pass and compute a good estimate of the frequencies using a limited amount of space. Over the last two decades, many such streaming algorithms have been developed, including Count-Sketch [43], Count-Min [57] and multi-stage filters [70]. The performance guarantees of these algorithms are well-understood, with upper and lower bounds matching up to $O(\cdot)$ factors [110].

However, such streaming algorithms typically assume generic data and do not leverage useful patterns or properties of their input. For example, in text data, the word frequency is known to be inversely correlated with the length of the word. Analogously, in network data, certain applications tend to generate more traffic than others. If such properties can be harnessed, one could design frequency estimation algorithms that are much more efficient than the existing ones. Yet, it is important to do so in a general framework that can harness various useful properties, instead of using handcrafted methods specific to a particular pattern or structure (e.g., word length, application type). In Chapter 6, we introduce "learning-based" frequency estimation streaming algorithms.

1.2.2. Learning-Based Algorithms for Low-Rank Approximation

Low-rank approximation is one of the most widely used tools in massive data analysis, machine learning and statistics, and has been a subject of many algorithmic studies. Formally, in low-rank approximation, given an $n \times d$ matrix A, and a parameter k, the goal is to compute a rank-k matrix

$$[A]_k = \text{argmin}_{A' \colon \text{rank}(A') \leq k} \|A - A'\|_F.$$

In particular, multiple algorithms developed over the last decade use the "sketching" approach, see e.g., [157, 171, 92, 52, 53, 144, 132, 34, 54]. Its idea is to use efficiently computable random projections (a.k.a., "sketches") to reduce the problem size before performing low-rank decomposition, which makes the computation more space and time efficient. For example, [157, 52] show that if S is a random matrix of size $m \times n$ chosen from an appropriate

distribution[4], for m depending on ε, then one can recover a rank-k matrix A' such that

$$\|A - A'\|_F \leq (1+\epsilon)\|A - [A]_k\|_F$$

by performing an SVD on $SA \in \mathbb{R}^{m \times d}$ followed by some post-processing. Typically the sketch length m is small, so the matrix SA can be stored using little space (in the context of streaming algorithms) or efficiently communicated (in the context of distributed algorithms). Furthermore, the SVD of SA can be computed efficiently, especially after another round of sketching, reducing the overall computation time.

In light of the success of learning-based approaches, it is natural to ask whether similar improvements in performance could be obtained for other sketch-based algorithms, notably for low-rank decompositions. In particular, reducing the sketch length m while preserving its accuracy would make sketch-based algorithms more efficient. Alternatively, one could make sketches more accurate for the same values of m. In Chapter 7, we design the first "learning-based" (streaming) algorithm for the low-rank approximation problem.

[4]Initial algorithms used matrices with independent sub-gaussian entries or randomized Fourier/Hadamard matrices [157, 171, 92]. Starting from the seminal work of [53], researchers began to explore sparse binary matrices, see e.g., [144, 132].

Part I

Sublinear Algorithms for Coverage Problems

Chapter 2

Streaming Set Cover

2.1. Introduction

In the *streaming* Set Cover problem as defined in [155], the set of elements $\mathcal{U} = \{e_1, \cdots, e_n\}$ is stored in the memory in advance; the sets $S_1, \cdots, S_m$ are stored consecutively in a read-only repository and an algorithm can access the sets only by performing sequential scans of the repository. However, the amount of read-write memory available to the algorithm is limited, and is smaller than the input size (which could be as large as mn). The objective is to design an efficient approximation algorithm for the Set Cover problem that performs few passes over the data, and uses as little memory as possible.

The last few years have witnessed a rapid development of new streaming algorithms for the Set Cover problem, in both theory and applied communities, see [155, 56, 122, 67, 61, 42]. Table 2.1.1 presents the approximation and space bounds achieved by those algorithms, as well as the lower bounds.[1]

2.1.1. Related Work

The semi-streaming Set Cover problem was first studied by Saha and Getoor [155]. Their result for Max k-Cover problem implies a $O(\log n)$-pass $O(\log n)$-approximation algorithm for the Set Cover problem that uses $\widetilde{O}(n^2)$ space. Adopting the standard greedy algorithm of Set Cover with a thresholding technique leads to $O(\log n)$-pass $O(\log n)$-approximation using

[1] Note that the simple greedy algorithm can be implemented by either storing the whole input (in one pass), or by iteratively updating the set of yet-uncovered elements (in at most n passes).

Result	Approximation	Passes	Space	Note
Greedy Algorithm	$\ln n$	1 n	$O(mn)$ $O(n)$	Deterministic
[155]	$O(\log n)$	$O(\log n)$	$O(n^2 \ln n)$	Deterministic
[67]	$O(\sqrt{n})$	1	$\widetilde{\Theta}(n)$	Randomized (tight bounds) & *deterministic algorithm*
[42]	$O(\frac{n^\delta}{\delta})$	$\frac{1}{\delta}-1$	$\widetilde{\Theta}(n)$	Randomized (tight bounds) & *deterministic algorithm*
[146]	$\frac{1}{2}\log n$	$O(\log n)$	$\widetilde{\Omega}(m)$	Randomized
[61]	$O(4^{\frac{1}{\delta}}\rho)$	$O(4^{\frac{1}{\delta}})$	$\widetilde{O}(mn^\delta)$	Randomized
[61]	$O(1)$	$O(\log n)$	$\widetilde{\Omega}(mn)$	Deterministic
Theorem 2.2.8	$O(\frac{\rho}{\delta})$	$\frac{2}{\delta}$	$\widetilde{O}(mn^\delta)$	Randomized
Theorem 2.3.8	$\frac{3}{2}$	1	$\Omega(mn)$	Randomized
Theorem 2.5.4	1	$\frac{1}{2\delta}-1$	$\widetilde{\Omega}(mn^\delta)$	Randomized
Geometric Set Cover (Theorem 2.4.6)	$O(\rho_g)$	$O(1)$	$\widetilde{O}(n)$	Randomized
s-Sparse Set Cover (Theorem 2.6.6)	1	$\frac{1}{2\delta}-1$	$\widetilde{\Omega}(ms)$	Randomized

Table 2.1.1: Summary of past work and our results. Parameters ρ and ρ_g respectively denote the approximation factor of off-line algorithms for Set Cover and its geometric variant.

$\widetilde{O}(n)$ space. In $\widetilde{O}(n)$ space regime, Emek and Rosen studied designing one-pass streaming algorithms for the Set Cover problem [67] and gave a *deterministic* greedy based $O(\sqrt{n})$-approximation for the problem. Moreover they proved that their algorithm is tight, even for randomized algorithms. The lower/upper bound results of [67] applied also to a generalization of the Set Cover problem, the ε-Partial Set Cover$(\mathcal{U},\mathcal{F})$ problem in which the goal is to cover $(1-\varepsilon)$ fraction of elements $\mathcal{U}$ and the size of the solution is compared to the size of an optimal cover of Set Cover$(\mathcal{U},\mathcal{F})$. Very recently, Chakrabarti and Wirth extended the result of [67] and gave a trade-off streaming algorithm for the Set Cover problem in *multiple passes* [42]. They gave a *deterministic* algorithm with p passes over the data stream that returns a $(p+1)n^{1/(p+1)}$-approximate solution of the Set Cover problem in $\widetilde{O}(n)$ space. Moreover they

proved that achieving $0.99n^{1/(p+1)}/(p+1)^2$ in p passes using $\widetilde{O}(n)$ space is not possible even for randomized protocols which shows that their algorithm is tight up to a factor of $(p+1)^3$. Their result also works for the ε-Partial Set Cover problem.

In a different regime which was first studied by Demaine et al., the goal is to design a "low" approximation algorithms (depending on the computational model, it could be $O(\log n)$ or $O(1)$) in the smallest possible space [61]. They proved that any constant pass *deterministic* $(\log n/2)$-approximation algorithm for the Set Cover problem requires $\widetilde{\Omega}(mn)$ space. It shows that unlike the results in $\widetilde{O}(n)$-space regime, to obtain a sublinear "low" approximation streaming algorithm for the Set Cover problem in a constant number of passes, using *randomness* is necessary. Moreover, [61] presented a $O(4^{1/\delta})$-approximation algorithm that makes $O(4^{1/\delta})$ passes and uses $\widetilde{O}(mn^{\delta})$ memory space.

The Set Cover problem is not polynomially solvable even in the restricted instances with points in $\mathbb{R}^2$ as elements, and geometric objects (either all disks or axis parallel rectangles or fat triangles) in plane as sets [71, 75, 99]. As a result, there has been a large body of work on designing approximation algorithms for the geometric Set Cover problems. See for example [142, 3, 15, 51] and references therein.

2.1.2. Our Results

Despite the progress outlined above, however, some basic questions still remained open. In particular:

(A) Is it possible to design a *single* pass streaming algorithm with a "low" approximation factor[2] that uses sublinear (i.e., $o(mn)$) space?

(B) If such single pass algorithms are not possible, what are the achievable trade-offs between the number of passes and space usage?

(C) Are there special instances of the problem for which more efficient algorithms can be designed?

[2]Note that the lower bound in [61] excluded this possibility only for deterministic algorithms, while the upper bound in [67, 42] suffered from a polynomial approximation factor.

In this chapter, we make a significant progress on each of these questions. Our upper and lower bounds are depicted in Table 2.1.1.

On the algorithmic side, we give a $O(1/\delta)$-pass algorithm with a strongly sub-linear $\widetilde{O}(mn^{\delta})$ space and logarithmic approximation factor. This yields a significant improvement over the earlier algorithm of Demaine et al. [61] which used exponentially larger number of passes. The trade-off offered by our algorithm matches the lower bound of Nisan [146] that holds at the endpoint of the trade-off curve, i.e., for $\delta = \Theta(1/\log n)$, up to poly-logarithmic factors in space[3]. Furthermore, our algorithm is very simple and succinct, and therefore easy to implement and deploy.

Our algorithm exhibits a natural tradeoff between the number of passes and space, which resembles tradeoffs achieved for other problems [87, 88, 90]. It is thus natural to conjecture that this tradeoff might be tight, at least for "low enough" approximation factors. We present the first step in this direction by showing a lower bound for the case when the approximation factor is equal to 1, i.e., the goal is to compute the optimal set cover. In particular, by an information theoretic lower bound, we show that any *streaming* algorithm that computes set cover using $(\frac{1}{2\delta}-1)$ passes must use $\widetilde{\Omega}(mn^{\delta})$ space (even assuming exponential computational power) in the regime of $m = O(n)$. Furthermore, we show that a stronger lower bound holds if all the input sets are sparse, that is if their cardinality is at most s. We prove a lower bound of $\widetilde{\Omega}(ms)$ for $s = O(n^{\delta})$ and $m = O(n)$.

We also consider the problem in the geometric setting in which the elements are points in $\mathbb{R}^2$ and sets are either discs, axis-parallel rectangles, or fat triangles in the plane. We show that a slightly modified version of our algorithm achieves the *optimal* $\widetilde{O}(n)$ space to find an $O(\rho)$-approximation in $O(1)$ passes.

Finally, we show that any randomized one-pass algorithm that distinguishes between covers of size 2 and 3 must use a linear (i.e., $\Omega(mn)$) amount of space. This is the first result showing that a randomized, approximate algorithm cannot achieve a sub-linear space bound.

Recently Assadi et al. [20] generalized this lower bound to any approximation ratio $\alpha =$

[3]Note that to achieve a logarithmic approximation ratio we can use an off-line algorithm with the approximation ratio $\rho = 1$, i.e., one that runs in exponential time (see Theorem 2.2.8).

$O(\sqrt{n})$. More precisely they showed that approximating Set Cover within any factor $\alpha = O(\sqrt{n})$ in a single pass requires $\Omega(\frac{mn}{\alpha})$ space.

Our techniques: basic idea. Our algorithm is based on the idea that whenever a large enough set is encountered, we can immediately add it to the cover. Specifically, we guess (up to factor two) the size of the optimal cover k. Thus, a set is "large" if it covers at least $1/k$ fraction of the remaining elements. A small set, on the other hand, can cover only a "few" elements, and we can store (approximately) what elements it covers by storing (in memory) an appropriate random sample. At the end of the pass, we have (in memory) the projections of "small" sets onto the random sample, and we compute the optimal set cover for this projected instance using an offline solver. By carefully choosing the size of the random sample, this guarantees that only a small fraction of the set system remains uncovered. The algorithm then makes an additional pass to find the residual set system (i.e., the yet uncovered elements), making two passes in each iteration, and continuing to the next iteration.

Thus, one can think about the algorithm as being based on a simple iterative "dimensionality reduction" approach. Specifically, in two passes over the data, the algorithm selects a "small" number of sets that cover all but $n^{-\delta}$ fraction of the uncovered elements, while using only $\widetilde{O}(mn^{\delta})$ space. By performing the reduction step $1/\delta$ times we obtain a complete cover. The dimensionality reduction step is implemented by computing a small cover for a *random subset* of the elements, which also covers the vast majority of the elements in the ground set. This ensures that the remaining sets, when restricted to the random subset of the elements, occupy only $\widetilde{O}(mn^{\delta})$ space. As a result the procedure avoids a complex set of recursive calls as presented in Demaine et al. [61], which leads to a simpler and more efficient algorithm.

Geometric results. Further using techniques and results from computational geometry we show how to modify our algorithm so that it achieves *almost* optimal bounds for the Set Cover problem on geometric instances. In particular, we show that it gives a $O(1)$-pass $O(\rho)$-approximation algorithm using $\widetilde{O}(n)$ space when the elements are points in $\mathbb{R}^2$ and the sets are either discs, axis parallel rectangles, or fat triangles in the plane. In particular, we

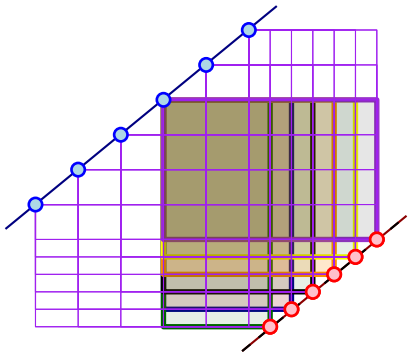


Figure 2.1.1: A collection of $n^2/4$ distinct rectangles (i.e., sets), each containing two points.

use the following surprising property of the set systems that arise out of points and disks: the the number of sets is nearly linear as long as one considers only sets that contain "a few" points.

More surprisingly, this property extends, with a twist, to certain geometric range spaces that might have quadratic number of shallow ranges. Indeed, it is easy to show an example of n points in the plane, where there are $\Omega(n^2)$ distinct rectangles, each one containing exactly two points, see Figure 2.1.1. However, one can "split" such ranges into a small number of canonical sets, such that the number of shallow sets in the canonical set system is near linear. This enables us to store the small canonical sets encountered during the scan explicitly in memory, and still use only near linear space.

We note that the idea of splitting ranges into small canonical ranges is an old idea in orthogonal range searching. It was used by Aronov et al. [15] for computing small ε-nets for these range spaces. The idea in the form we use, was further formalized by Ene et al. [68].

Lower bounds. The lower bounds for multi-pass algorithms for the Set Cover problem are obtained via a careful reduction from Intersection Set Chasing. The latter problem is a communication complexity problem where n players need to solve a certain "set-disjointness-like" problem in p rounds. A recent paper [90] showed that this problem requires $\frac{n^{1+\Omega(1/p)}}{p^{O(1)}}$ bits of communication complexity for p rounds. This yields our desired trade-off of $\widetilde{\Omega}(mn^{\delta})$ space in $1/2\delta$ passes for exact protocols for Set Cover in the communication model and hence in the streaming model for $m = O(n)$. Furthermore, we show a stronger lower bound on memory space of sparse instances of Set Cover in which all input sets have cardinality at most s. By a reduction from a variant of Equal Pointer Chasing which maps the problem to

Algorithm 1 A tight streaming algorithm for the (unweighted) Set Cover problem. Here, OFFLINESC is an offline solver for Set Cover that provides ρ-approximation, and c is some appropriate constant.

procedure ITERSETCOVER($(\mathcal{U}, \mathcal{F}), \delta$)
  ▷ try (in parallel) all possible (2-approx) sizes of optimal cover
  for $k \in \{2^i \mid 0 \leq i \leq \log n\}$ in parallel **do** ▷ $n = |\mathcal{U}|$
    SOL $\leftarrow \emptyset$
    for $i = 1$ to $1/\delta$ **do**
      let S be a sample of $\mathcal{U}$ of size $c\rho k n^{\delta} \log m \log n$
      $\mathsf{L} \leftarrow \mathsf{S}, \quad \mathcal{F}_{\mathsf{S}} \leftarrow \emptyset$
      for $S \in \mathcal{F}$ **do** ▷ by doing one pass
        if $|\mathsf{L} \cap S| \geq |\mathsf{S}|/k$ **then** ▷ size test
          SOL $\leftarrow$ SOL $\cup \{S\}$
          $\mathsf{L} \leftarrow \mathsf{L} \setminus S$
        else
          $\mathcal{F}_{\mathsf{S}} \leftarrow \mathcal{F}_{\mathsf{S}} \cup \{S \cap \mathsf{L}\}$ ▷ store the set $S \cap \mathsf{L}$ explicitly in memory
      $\mathcal{D} \leftarrow$ OFFLINESC($\mathsf{L}, \mathcal{F}_{\mathsf{S}}, k$)
      SOL $\leftarrow$ SOL $\bigcup \mathcal{D}$
      $\mathcal{U} \leftarrow \mathcal{U} \setminus \bigcup_{S \in \text{SOL}} S$ ▷ by doing additional pass over data
  return best SOL computed in all parallel executions.

a *sparse* instance of Set Cover, we show that in order to have an exact streaming algorithm for s-Sparse Set Cover with $o(ms)$ space, $\Omega(\log n)$ passes is necessary. More precisely, any $(\frac{1}{2\delta} - 1)$-pass *exact* randomized algorithm for s-Sparse Set Cover requires $\widetilde{\Omega}(ms)$ memory space, if $s \leq n^{\delta}$ and $m = O(n)$.

Our single pass lower bound proceeds by showing a lower bound for a one-way communication complexity problem in which one party (Alice) has a collection of sets, and the other party (Bob) needs to determine whether the complement of his set is covered by one of the Alice's sets. We show that if Alice's sets are chosen at random, then Bob can decode Alice's input by employing a small collection of "query" sets. This implies that the amount of communication needed to solve the problem is linear in the description size of Alice's sets, which is $\Omega(mn)$.

2.2. Streaming Algorithm for Set Cover

In this section, we design an efficient streaming algorithm for the Set Cover problem that matches the lower bound results we already know about the problem. In the Set Cover problem, for a given set system $(\mathcal{U}, \mathcal{F})$, the goal is to find a subset $\mathcal{I} \subseteq \mathcal{F}$, such that $\mathcal{I}$ covers

$\mathcal{U}$ and its cardinality is minimum. In Algorithm 1, we sketch the IterSetCover algorithm.

In the IterSetCover algorithm, we have access to the OfflineSC subroutine that solves the given Set Cover instance offline (using linear space) and returns a ρ-approximate solution where ρ could be anywhere between 1 and $\Theta(\log n)$ depending on the computational model one assumes. Under exponential computational power, we can achieve the optimal cover of the given instance of the Set Cover ($\rho = 1$); however, under P $\neq$ NP assumption, ρ cannot be better than $c \cdot \ln n$ where c is a constant [72, 152, 14, 138, 63] given polynomial computational power.

Let $n = |\mathcal{U}|$ be the initial number of elements in the given ground set. The IterSetCover algorithm, needs to guess (up to a factor of two) the size of the optimal cover of $(\mathcal{U}, \mathcal{F})$. To this end, the algorithm tries, in parallel, all values k in $\{2^i \mid 0 \leq i \leq \log n\}$. This step will only increase the memory space requirement by a factor of $\log n$.

Consider the run of the IterSetCover algorithm, in which the guess k is correct (i.e., $|\mathrm{OPT}| \leq k < 2|\mathrm{OPT}|$, where OPT is an optimal solution). The idea is to go through $O(1/\delta)$ iterations such that each iteration only makes two passes and at the end of each iteration the number of uncovered elements reduces by a factor of n^δ. Moreover, the algorithm is allowed to use $\widetilde{O}(mn^\delta)$ space.

In each iteration, the algorithm starts with the current ground set of uncovered elements $\mathcal{U}$, and copies it to a *leftover* set L. Let S be a large enough uniform sample of elements $\mathcal{U}$. In a single pass, using S, we estimate the size of all large sets in $\mathcal{F}$ and add $S \in \mathcal{F}$ to the solution sol immediately (thus avoiding the need to store it in memory). Formally, if S covers at least $\Omega(|\mathcal{U}|/k)$ yet-uncovered elements of L then it is a heavy set, and the algorithm immediately adds it to the output cover. Otherwise, if a set is small, i.e., its covers less than $|\mathcal{U}|/k$ uncovered elements of L, the algorithm stores the set S in memory. Fortunately, it is enough to store its projection over the sampled elements explicitly (i.e., $S \cap \mathsf{L}$) – this requires remembering only the $O(|\mathsf{S}|/k)$ indices of the elements of $S \cap \mathsf{L}$.

In order to show that a solution of the Set Cover problem over the sampled elements is a good cover of the initial Set Cover instance, we apply the *relative (p, ε)-approximation* sampling result of [100] (see Definition 2.2.4) and it is enough for S to be of size $\widetilde{O}(\rho k n^\delta)$. Using relative (p, ε)-approximation sampling, we show that after two passes the number of

uncovered elements is reduced by a factor of n^δ. Note that the relative (p, ε)-approximation sampling improves over the *Element Sampling* technique used in [61] with respect to the number of passes.

Since in each iteration we pick $O(\rho k)$ sets and the number of uncovered elements decreases by a factor of n^δ, after $1/\delta$ iterations the algorithm picks $O(\rho k/\delta)$ sets and covers all elements. Moreover, the memory space of the whole algorithm is $\widetilde{O}(\rho m n^\delta)$ (see Lemma 2.2.2).

2.2.1. Analysis of IterSetCover

In the rest of this section we prove that the IterSetCover algorithm with high probability returns a $O(\rho/\delta)$-approximate solution of Set Cover($\mathcal{U}, \mathcal{F}$) in $2/\delta$ passes using $\widetilde{O}(mn^\delta)$ memory space.

Lemma 2.2.1. *The number of passes the* IterSetCover *algorithm makes is* $2/\delta$.

Proof: In each of the $1/\delta$ iterations of the IterSetCover algorithm, the algorithm makes two passes. In the first pass, based on the set of sampled elements S, it decides whether to pick a set or keep its projection over S (i.e., $S \cap \mathsf{L}$) in the memory. Then the algorithm calls OfflineSC which does not require any passes over $\mathcal{F}$. The second pass is for computing the set of uncovered elements at the end of the iteration. We need this pass because we only know the projection of the sets we picked in the current iteration over S and not over the original set of uncovered elements. Thus, in total we make $2/\delta$ passes. Also note that for different guesses for the value of k, we run the algorithm in parallel and hence the total number of passes remains $2/\delta$. □

Lemma 2.2.2. *The memory space used by the* IterSetCover *algorithm is* $\widetilde{O}(mn^\delta)$.

Proof: In each iteration of the algorithm, it picks during the first pass at most m sets (more precisely at most k sets) which requires $O(m \log m)$ memory. Moreover, in the first pass we keep the projection of the sets whose projection over the uncovered sampled elements has size at most $|\mathsf{S}|/k$. Since there are at most m such sets, the total required space for storing the projections is bounded by $O\big(\rho m n^\delta \log m \log n\big)$.

Since in the second pass the algorithm only updates the set of uncovered elements, the amount of space required in the second pass is $O(n)$. Thus, the total required space to

perform each iteration of the IterSetCover algorithm is $\widetilde{O}(mn^{\delta})$. Moreover, note that the algorithm does not need to keep the memory space used by the earlier iterations; thus, the total space consumed by the algorithm is $\widetilde{O}(mn^{\delta})$. □

Next we show the sets we picked before calling OfflineSC has large size on $\mathcal{U}$.

Lemma 2.2.3. *With probability at least $1 - m^{-c}$ all sets that pass the "size test" in the* IterSetCover *algorithm have size at least $|\mathcal{U}|/ck$.*

Proof: Let S be a set of size less than $|\mathcal{U}|/ck$. In expectation, $|S \cap \mathsf{S}|$ is less than $(|\mathcal{U}|/ck) \cdot (|\mathsf{S}|/|\mathcal{U}|) = \rho n^{\delta} \log m \log n$. By Chernoff bound for large enough c,

$$\mathbf{Pr}(|S \cap \mathsf{S}| \geq c\rho n^{\delta} \log m \log n) \leq m^{-(c+1)}.$$

Applying the union bound, with probability at least $1 - m^{-c}$, all sets passing "size test" have size at least $|\mathcal{U}|/(ck)$. □

In what follows we define the *relative (p, ε)-approximation* sample of a set system and mention the result of Har-Peled and Sharir [100] on the minimum required number of sampled elements to get a relative (p, ε)-approximation of the given set system.

Definition 2.2.4. Let $(\mathsf{V}, \mathcal{H})$ be a set system, i.e., V is a set of elements and $\mathcal{H} \subseteq 2^{\mathsf{V}}$ is a family of subsets of the ground set V. For given parameters $0 < \varepsilon, p < 1$, a subset $Z \subseteq \mathsf{V}$ is a *relative (p, ε)-approximation* for $(\mathsf{V}, \mathcal{H})$, if for each $S \in \mathcal{H}$, we have that if $|S| \geq p|\mathsf{V}|$ then

$$(1 - \varepsilon)\frac{|S|}{|\mathsf{V}|} \leq \frac{|S \cap Z|}{|Z|} \leq (1 + \varepsilon)\frac{|S|}{|\mathsf{V}|}.$$

If the range is light (i.e., $|S| < p|\mathsf{V}|$) then it is required that

$$\frac{|S|}{|\mathsf{V}|} - \varepsilon p \leq \frac{|S \cap Z|}{|Z|} \leq \frac{|S|}{|\mathsf{V}|} + \varepsilon p.$$

Namely, Z is $(1 \pm \varepsilon)$-multiplicative good estimator for the size of ranges that are at least p-fraction of the ground set.

The following lemma is a simplified variant of a result in Har-Peled and Sharir [100] – indeed, a set system with M sets, can have VC dimension at most $\log M$. This simplified form

also follows by a somewhat careful but straightforward application of Chernoff's inequality.

Lemma 2.2.5. *Let $(\mathcal{U}, \mathcal{F})$ be a finite set system, and p, ε, q be parameters. Then, a random sample of $\mathcal{U}$ such that $|\mathcal{U}| = \frac{c'}{\varepsilon^2 p}\left(\log|\mathcal{F}| \log\frac{1}{p} + \log\frac{1}{q}\right)$, for an absolute constant c' is a relative (p, ε)-approximation, for all ranges in $\mathcal{F}$, with probability at least $(1-q)$.*

Lemma 2.2.6. *Assuming $|\mathrm{OPT}| \leq k \leq 2|\mathrm{OPT}|$, after any iteration, with probability at least $1 - m^{1-c/4}$ the number of uncovered elements decreases by a factor of n^δ, and this iteration adds $O(\rho|\mathrm{OPT}|)$ sets to the output cover.*

Proof: Let $\mathsf{V} \subseteq \mathcal{U}$ be the set of uncovered elements at the beginning of the iteration and note that the total number of sets that is picked during the iteration is at most $(1+\rho)k$ (see Lemma 2.2.3). Consider all possible such covers, that is $\mathcal{G} = \{\mathcal{F}' \subseteq \mathcal{F} \mid |\mathcal{F}'| \leq (1+\rho)k\}$, and observe that $|\mathcal{G}| \leq m^{(1+\rho)k}$. Let $\mathcal{H}$ be the collection that contains all possible sets of uncovered elements at the end of the iteration, defined as $\mathcal{H} = \left\{\mathsf{V} \setminus \bigcup_{S\in\mathcal{C}} S \;\middle|\; \mathcal{C} \in \mathcal{G}\right\}$. Moreover, set $p = 2/n^\delta$, $\varepsilon = 1/2$ and $q = m^{-c}$ and note that $|\mathcal{H}| \leq |\mathcal{G}| \leq m^{(1+\rho)k}$. Since $\frac{c'}{\varepsilon^2 p}(\log|\mathcal{H}|\log\frac{1}{p} + \log\frac{1}{q}) \leq c\rho k n^\delta \log m \log n = |\mathsf{S}|$ for large enough c, by Lemma 2.2.5, S is a relative (p,ε)-approximation of $(\mathsf{V}, \mathcal{H})$ with $(1-q)$ probability. Let $\mathcal{D} \subseteq \mathcal{F}$ be the collection of sets picked during the iteration which covers all elements in S. Since S is a relative (p,ε)-approximation sample of $(\mathsf{V}, \mathcal{H})$ with probability at least $1 - m^{-c}$, the number of uncovered elements of V (or $\mathcal{U}$) by $\mathcal{D}$ is at most $\varepsilon p|\mathsf{V}| = |\mathcal{U}|/n^\delta$.

Hence, in each iteration we pick $O(\rho k)$ sets and at the end of iteration the number of uncovered elements reduces by n^δ. □

Lemma 2.2.7. *The* IterSetCover *algorithm computes a set cover of $(\mathcal{U}, \mathcal{F})$, whose size is within a $O(\rho/\delta)$ factor of the size of an optimal cover with probability at least $1 - m^{1-c/4}$.*

Proof: Consider the run of IterSetCover for which the value of k is between $|\mathrm{OPT}|$ and $2|\mathrm{OPT}|$. In each of the $(1/\delta)$ iterations made by the algorithm, by Lemma 2.2.6, the number of uncovered elements decreases by a factor of n^δ where n is the number of initial elements to be covered by the sets. Moreover, the number of sets picked in each iteration is $O(\rho k)$. Thus after $(1/\delta)$ iterations, all elements would be covered and the total number of sets in the solution is $O(\rho|\mathrm{OPT}|/\delta)$. Moreover by Lemma 2.2.6, the success probability of all the iterations, is at least $1 - \frac{1}{\delta m^{c/4}} \geq 1 - (1/m)^{\frac{c}{4}-1}$. □

Theorem 2.2.8. *The* IterSetCover$(\mathcal{U}, \mathcal{F}, \delta)$ *algorithm makes* $2/\delta$ *passes, uses* $\widetilde{O}(mn^{\delta})$ *memory space, and finds a* $O(\rho/\delta)$*-approximate solution of the* Set Cover *problem with high probability.*

Furthermore, given enough number of passes the IterSetCover *algorithm matches the known lower bound on the memory space of the streaming* Set Cover *problem up to a* polylog(m) *factor where* m *is the number of sets in the input.*

Proof: The first part of the proof implied by Lemma 2.2.1, Lemma 2.2.2, and Lemma 2.2.7.

As for the lower bound, note that by a result of Nisan [146], any randomized $(\frac{\log n}{2})$-approximation protocol for Set Cover$(\mathcal{U}, \mathcal{F})$ in the one-way communication model requires $\Omega(m)$ bits of communication, no matter how many number of rounds it makes. This implies that any randomized $O(\log n)$-pass, $(\frac{\log n}{2})$-approximation algorithm for Set Cover$(\mathcal{U}, \mathcal{F})$ requires $\widetilde{\Omega}(m)$ space, even under the exponential computational power assumption.

By the above, the IterSetCover algorithm makes $O(1/\delta)$ passes and uses $\widetilde{O}(mn^{\delta})$ space to return a $O(\frac{1}{\delta})$-approximate solution under the exponential computational power assumption ($\rho = 1$). Thus by letting $\delta = c/\log n$, we will have a $(\frac{\log n}{2})$-approximation streaming algorithm using $\widetilde{O}(m)$ space which is optimal up to a factor of polylog(m). □

Theorem 2.2.8 provides a strong indication that our trade-off algorithm is optimal.

2.3. Lower Bound for Single Pass Algorithms

In this section, we study the Set Cover problem in the two-party communication model and give a tight lower bound on the communication complexity of the randomized protocols solving the problem in a single round. In the two-party Set Cover, we are given a set of elements $\mathcal{U}$ and there are two players Alice and Bob where each of them has a collection of subsets of $\mathcal{U}$, $\mathcal{F}_A$ and $\mathcal{F}_B$. The goal for them is to find a minimum size cover $\mathcal{C} \subseteq \mathcal{F}_A \cup \mathcal{F}_B$ covering $\mathcal{U}$ while communicating the fewest number of bits from Alice to Bob (In this model Alice communicates to Bob and then Bob should report a solution).

Our main lower bound result for the single pass protocols for Set Cover is the following theorem which implies that the naive approach in which one party sends all of its sets to the the other one is optimal.

Theorem 2.3.1. *Any single round randomized protocol that approximates* Set Cover$(\mathcal{U}, \mathcal{F})$ *within a factor better than* $3/2$ *and error probability* $O(m^{-c})$ *requires* $\Omega(mn)$ *bits of communication where* $n = |\mathcal{U}|$ *and* $m = |\mathcal{F}|$ *and* c *is a sufficiently large constant.*

We consider the case in which the parties want to decide whether there exists a cover of size 2 for $\mathcal{U}$ in $\mathcal{F}_A \cup \mathcal{F}_B$ or not. If any of the parties has a cover of size at most 2 for $\mathcal{U}$, then it becomes trivial. Thus the question is whether there exist $S_a \in \mathcal{F}_A$ and $S_b \in \mathcal{F}_B$ such that $\mathcal{U} \subseteq S_a \cup S_b$.

A key observation is that to decide whether there exist $S_a \in \mathcal{F}_A$ and $S_b \in \mathcal{F}_B$ such that $\mathcal{U} \subseteq S_a \cup S_a$, one can instead check whether there exists $S_a \in \mathcal{F}_A$ and $S_b \in \mathcal{F}_B$ such that $\overline{S_a} \cap \overline{S_b} = \emptyset$. In other words we need to solve OR of a series of two-party Set Disjointness problems. In two-party Set Disjointness problem, Alice and Bob are given subsets of $\mathcal{U}$, S_a and S_b and the goal is to decide whether $S_a \cap S_b$ is empty or not with the fewest possible bits of communication. Set Disjointness is a well-studied problem in the communication complexity and it has been shown that any randomized protocol for Set Disjointness with $O(1)$ error probability requires $\Omega(n)$ bits of communication where $n = |\mathcal{U}|$ [27, 111, 153].

We can think of the following extensions of the Set Disjointness problem.

I. *Many vs One:* in this variant, Alice has m subsets of $\mathcal{U}$, $\mathcal{F}_A$ and Bob is given a single set S_b. The goal is to determine whether there exists a set $S_a \in \mathcal{F}_A$ such that $S_a \cap S_b = \emptyset$.

II. *Many vs Many:* in this variant, each of Alice and Bob are given a collection of subsets of $\mathcal{U}$ and the goal for them is to determine whether there exist $S_a \in \mathcal{F}_A$ and $S_b \in \mathcal{F}_B$ such that $S_a \cap S_b = \emptyset$.

Note that deciding whether two-party Set Cover has a cover of size 2 is equivalent to solving the (Many vs Many)-Set Disjointness problem. Moreover, any lower bound for (Many vs One)-Set Disjointness clearly implies the same lower bound for the (Many vs Many)-Set Disjointness problem. In the following theorem we show that any single-round randomized protocol that solves (Many vs One)-Set Disjointness(m, n) with $O(m^{-c})$ error probability requires $\Omega(mn)$ bits of communication.

Theorem 2.3.2. *Any randomized protocol for* (Many vs One)-Set Disjointness(m, n) *with*

error probability that is $O(m^{-c})$ requires $\Omega(mn)$ bits of communication if $n \geq c_1 \log m$ where c and c_1 are large enough constants.

The idea is to show that if there exists a single-round randomized protocol for the problem with $o(mn)$ bits of communication and error probability $O(m^{-c})$, then with constant probability one can distinguish $\Omega(2^{mn})$ distinct inputs using $o(mn)$ bits which is a contradiction.

Suppose that Alice has a collection of m uniformly and independently random subsets of $\mathcal{U}$ (in each of her subsets the probability that $e \in \mathcal{U}$ is in the subset is $1/2$). Lets assume that there exists a *single round* protocol $\mathbf{I}$ for (Many vs One)-Set Disjointness(n, m) with error probability $O(m^{-c})$ using $o(mn)$ bits of communication. Let ExistsDisj be Bob's algorithm in protocol $\mathbf{I}$. Then we show that one can recover mn random bits with constant probability using ExistsDisj subroutine and the message s sent by the first party in protocol $\mathbf{I}$. The RecoverBit which is shown in Algorithm 2, is the algorithm to recover random bits using protocol $\mathbf{I}$ and ExistsDisj.

To this end, Bob gets the message s communicated by protocol $\mathbf{I}$ from Alice and considers all subsets of size $c_1 \log m$ and $c_1 \log m + 1$ of $\mathcal{U}$. Note that s is communicated only once and thus the same s is used for all queries that Bob makes. Then at each step Bob picks a random subset S_b of size $c_1 \log m$ of $\mathcal{U}$ and solve the (Many vs One)-Set Disjointness problem with input $(\mathcal{F}_A, S_b)$ by running ExistsDisj(s, S_b). Next we show that if S_b is disjoint from a set in $\mathcal{F}_A$, then with high probability there is *exactly* one set in $\mathcal{F}_A$ which is disjoint from S_b (see Lemma 2.3.3). Thus once Bob finds out that his query, S_b, is disjoint from a set in $\mathcal{F}_A$, he can query all sets $S_b^+ \in \{S_b \cup e | e \in \mathcal{U} \setminus S_b\}$ and recover the set (or union of sets) in $\mathcal{F}_A$ that is disjoint from S_b. By a simple *pruning step* we can detect the ones that are union of more than one set in $\mathcal{F}_A$ and only keep the sets in $\mathcal{F}_A$.

In Lemma 2.3.6, we show that the number of queries that Bob is required to make to recover $\mathcal{F}_A$ is $O(m^c)$ where c is a constant.

Lemma 2.3.3. *Let S_b be a random subset of $\mathcal{U}$ of size $c \log m$ and let $\mathcal{F}_A$ be a collection of m random subsets of $\mathcal{U}$. The probability that there exists exactly one set in $\mathcal{F}_A$ that is disjoint from S_b is at least $\frac{1}{m^{c+1}}$.*

Algorithm 2 RecoverBit uses a protocol for (Many vs One)-Set Disjointness(m, n) to recover Alice's sets, $\mathcal{F}_A$ in Bob's side.

```
1:  procedure RecoverBit(U, s)
2:      F_a ← ∅
3:      for i = 1 to m^c log m do
4:          let S_b be a random subset of U of size c_1 log m
5:          if ExistsDisj(s, S_b) = true then
6:              ▷ discovering the set (or union of sets)
7:              ▷ in F_A disjoint from S_b
8:              S ← ∅
9:              for e ∈ U \ S_b do
10:                 if ExistsDisj(S_b ∪ e, s) = false then
11:                     S ← S ∪ e
12:             if ∃S' ∈ F_a s.t. S ⊂ S' then ▷ pruning step
13:                 F_a ← F_a \ {S'}
14:                 F_a ← F_a ∪ {S}
15:             else if ∄S' ∈ F_a s.t. S' ⊂ S then
16:                 F_a ← F_a ∪ {S}
17:     return F_a
```

Proof: The probability that S_b is disjoint from exactly one set in $\mathcal{F}_A$ is

$$\begin{aligned}\mathbf{Pr}(S_b \text{ is disjoint from } \geq 1 \text{ set in } \mathcal{F}_A) - \mathbf{Pr}(S_b \text{ is disjoint from } \geq 2 \text{ sets in } \mathcal{F}_A)\\ \geq (\frac{1}{2})^{c\log m} - \binom{m}{2}(\frac{1}{2})^{2c\log m} \geq \frac{1}{m^{c+1}}.\end{aligned}$$

First we prove the first term in the above inequality. For an arbitrary set $S \in \mathcal{F}_A$, since any element is contained in S with probability $\frac{1}{2}$, the probability that S is disjoint from S_b is $(1/2)^{c\log m}$.

$$\mathbf{Pr}(S_b \text{ is disjoint from at least } \textbf{one} \text{ set in } \mathcal{F}_A) \geq 2^{-c\log m}.$$

Moreover since there exist $\binom{m}{2}$ pairs of sets in $\mathcal{F}_A$, and for each $S_1, S_2 \in \mathcal{F}_A$, the probability that S_1 and S_2 are disjoint from S_b is m^{-2c},

$$\mathbf{Pr}(S_b \text{ is disjoint from at least } \textbf{two} \text{ sets in } \mathcal{F}_A) \leq m^{-(2c-2)}. \qquad \square$$

A family of sets $\mathcal{M}$ is called *intersecting* if and only if for any sets $A, B \in \mathcal{M}$ either both

$A \setminus B$ and $B \setminus A$ are non-empty or both $A \setminus B$ and $B \setminus A$ are empty; in other words, there exists no $A, B \in \mathcal{M}$ such that $A \subseteq B$. Let $\mathcal{F}_A$ be a collection of subsets of $\mathcal{U}$. We show that with high probability after testing $O(m^c)$ queries for sufficiently large constant c, the RecoverBit algorithm recovers $\mathcal{F}_A$ completely if $\mathcal{F}_A$ is intersecting. First we show that with high probability the collection $\mathcal{F}_A$ is intersecting.

Observation 2.3.4. *Let $\mathcal{F}_A$ be a collection of m uniformly random subsets of $\mathcal{U}$ where $|\mathcal{U}| \geq c \log m$. With probability at least $1 - m^{-c/4+2}$, $\mathcal{F}_A$ is an intersecting family.*

Proof: The probability that $S_1 \subseteq S_2$ is $(\frac{3}{4})^n$ and there are at most $m(m-1)$ pairs of sets in $\mathcal{F}_A$. Thus with probability at least $1 - m^2(\frac{3}{4})^n \geq 1 - 1/m^{\frac{c}{4}-2}$, $\mathcal{F}_A$ is intersecting. □

Observation 2.3.5. *The number of distinct inputs of Alice (collections of random subsets of $\mathcal{U}$), that is distinguishable by* RecoverBit *is $\Omega(2^{mn})$.*

Proof: There are 2^{mn} collections of m random subsets of $\mathcal{U}$. By Observation 2.3.4, $\Omega(2^{mn})$ of them are intersecting. Since we can only recover the sets in the input collection and not their order, the distinct number of input collection that are distinguished by RecoverBit is $\Omega(\frac{2^{mn}}{m!})$ which is $\Omega(2^{mn})$ for $n \geq c \log m$. □

By Observation 2.3.4 and only considering the case such that $\mathcal{F}_A$ is intersecting, we have the following lemma.

Lemma 2.3.6. *Let $\mathcal{F}_A$ be a collection of m uniformly random subsets of $\mathcal{U}$ and suppose that $|\mathcal{U}| \geq c \log m$. After testing at most m^c queries, with probability at least $(1 - \frac{1}{m})p^{m^c}$, $\mathcal{F}_A$ is fully recovered, where p is the success rate of protocol* **I** *for the* (Many vs One)-Set Disjointness *problem.*

Proof: By Lemma 2.3.3, for each $S_b \subset \mathcal{U}$ of size $c_1 \log m$ the probability that S_b is disjoint from exactly one set in a random collection of sets $\mathcal{F}_A$ is at least $1/m^{c_1+1}$. Given S_b is disjoint from exactly one set in $\mathcal{F}_A$, due to symmetry of the problem, the chance that S_b is disjoint from a specific set $S \in \mathcal{F}_A$ is at least $\frac{1}{m^{c_1+2}}$. After $\alpha m^{c_1+2} \log m$ queries where α is a large enough constant, for any $S \in \mathcal{F}_A$, the probability that there is not a query S_b that is only disjoint from S is at most $(1 - \frac{1}{m^{c_1+2}})^{\alpha m^{c_1+2} \log m} \leq e^{-\alpha \log m} = \frac{1}{m^\alpha}$.

Thus after trying $\alpha m^{c_1+2} \log m$ queries, with probability at least $(1 - \frac{1}{2m^{\alpha-1}}) \geq (1 - \frac{1}{m})$,

for each $S \in \mathcal{F}_A$ we have at least one query that is only disjoint from S (and not any other sets in $\mathcal{F}_A \setminus S$).

Once we have a query subset S_b which is only disjoint from a single set $S \in \mathcal{F}_A$, we can ask $n - c\log m$ queries of size $c_1 \log m + 1$ and recover S. Note that if S_b is disjoint from more than one sets in $\mathcal{F}_A$ simultaneously, the process (asking $n - c\log m$ queries of size $c_1 \log m + 1$) will end up in recovering the union of those sets. Since $\mathcal{F}_A$ is an intersecting family with high probability (Observation 2.3.4), by *pruning step* in the RecoverBit algorithm we are guaranteed that at the end of the algorithm, what we returned is exactly $\mathcal{F}_A$. Moreover the total number of queries the algorithm makes is at most

$$n \times (\alpha m^{c_1+2} \log m) \leq \alpha m^{c_1+3} \log m \leq m^c$$

for $c \geq c_1 + 4$.

Thus after testing m^c queries, $\mathcal{F}_A$ will be recovered with probability at least $(1 - \frac{1}{m})p^{m^c}$ where p is the success probability of the protocol $\mathbf{I}$ for (Many vs One)-Set Disjointness(m, n).□

Corollary 2.3.7. *Let $\boldsymbol{I}$ be a protocol for* (Many vs One)-Set Disjointness(m, n) *with error probability $O(m^{-c})$ and s bits of communication such that $n \geq c\log m$ for large enough c. Then* RecoverBit *recovers $\mathcal{F}_A$ with constant success probability using s bits of communication.*

By Observation 2.3.5, since RecoverBit distinguishes $\Omega(2^{mn})$ distinct inputs with constant probability of success (by Corollary 2.3.7), the size of message sent by Alice, should be $\Omega(mn)$. This proves Theorem 2.3.2.

Proof of Theorem 2.3.1: As we showed earlier, the communication complexity of (Many vs One)-Set Disjointness is a lower bound for the communication complexity of Set Cover. Theorem 2.3.2 showed that any protocol for (Many vs One)-Set Disjointness$(n, |\mathcal{F}_A)|$ with error probability less than $O(m^{-c})$ requires $\Omega(mn)$ bits of communication. Thus any single-round randomized protocol for Set Cover with error probability $O(m^{-c})$ requires $\Omega(mn)$ bits of communication. □

Since any p-pass streaming α-approximation algorithm for problem P that uses $O(s)$

memory space, is a p-round two-party α-approximation protocol for problem P using $O(sp)$ bits of communication [88], and by Theorem 2.3.1, we have the following lower bound for Set Cover problem in the streaming model.

Theorem 2.3.8. *Any single-pass randomized streaming algorithm for* Set Cover$(\mathcal{U}, \mathcal{F})$ *that computes a* $(3/2)$*-approximate solution with probability* $\Omega(1 - m^{-c})$ *requires* $\Omega(mn)$ *memory space (assuming* $n \geq c_1 \log m$*).*

2.4. Geometric Set Cover

In this section, we consider the streaming Set Cover problem in the geometric settings. We present an algorithm for the case where the elements are a set of n points in the plane $\mathbb{R}^2$ and the m sets are either all disks, all axis-parallel rectangles, or all α-fat triangles (which for simplicity we call shapes) given in a data stream. As before, the goal is to find the minimum size cover of points from the given sets. We call this problem the Points-Shapes Set Cover problem.

Note that, the description of each shape requires $O(1)$ space and thus the Points-Shapes Set Cover problem is trivial to be solved in $O(m + n)$ space. In this setting the goal is to design an algorithm whose space is sub-linear in $O(m + n)$. Here we show that almost the same algorithm as IterSetCover (with slight modifications) uses $\widetilde{O}(n)$ space to find an $O(\rho)$-approximate solution of the Points-Shapes Set Cover problem in *constant* passes.

2.4.1. Preliminaries

A triangle $\triangle$ is called α-*fat* (or simply fat) if the ratio between its longest edge and its height on this edge is bounded by a constant $\alpha > 1$ (there are several equivalent definitions of α-fat triangles).

Definition 2.4.1. Let $(\mathcal{U}, \mathcal{F})$ be a set system such that $\mathcal{U}$ is a set of points and $\mathcal{F}$ is a collection of shapes, in the plane $\mathbb{R}^2$. The *canonical representation* of $(\mathcal{U}, \mathcal{F})$ is a collection $\mathcal{F}'$ of regions such that the following conditions hold. First, each $S' \in \mathcal{F}'$ has $O(1)$ description. Second, for each $S' \in \mathcal{F}'$, there exists $S \in \mathcal{F}$ such that $S' \cap \mathcal{U} \subseteq S \cap \mathcal{U}$. Finally, for each $S \in \mathcal{F}$, there exists c_1 sets $S'_1, \cdots, S'_{c_1} \in \mathcal{F}'$ such that $S \cap \mathcal{U} = (S'_1 \cup \cdots \cup S'_{c_1}) \cap \mathcal{U}$ for some constant c_1.

The following two results are from [68] which are the formalization of the ideas in [15].

Lemma 2.4.2. (Lemma 4.18 in [68]) *Given a set of points $\mathcal{U}$ in the plane $\mathbb{R}^2$ and a parameter w, one can compute a set $\mathcal{F}'_{\mathsf{total}}$ of $O(|\mathcal{U}|w^2 \log |\mathcal{U}|)$ axis-parallel rectangles with the following property. For an arbitrary axis-parallel rectangle S that contains at most w points of $\mathcal{U}$, there exist two axis-parallel rectangles $S'_1, S'_2 \in \mathcal{F}'_{\mathsf{total}}$ whose union has the same intersection with $\mathcal{U}$ as S, i.e., $S \cap \mathcal{U} = (S'_1 \cup S'_2) \cap \mathcal{U}$.*

Lemma 2.4.3. (Theorem 5.6 in [68]) *Given a set of points $\mathcal{U}$ in $\mathbb{R}^2$, a parameter w and a constant α, one can compute a set $\mathcal{F}'_{\mathsf{total}}$ of $O(|\mathcal{U}|w^3 \log^2 |\mathcal{U}|)$ regions each having $O(1)$ description with the following property. For an arbitrary α-fat triangle S that contains at most w points of $\mathcal{U}$, there exist nine regions from $\mathcal{F}'_{\mathsf{total}}$ whose union has the same intersection with $\mathcal{U}$ as S.*

Using the above lemmas we get the following lemma.

Lemma 2.4.4. *Let $\mathcal{U}$ be a set of points in $\mathbb{R}^2$ and let $\mathcal{F}$ be a set of shapes (discs, axis-parallel rectangles or fat triangles), such that each set in $\mathcal{F}$ contains at most w points of $\mathcal{U}$. Then, in a single pass over the stream of sets $\mathcal{F}$, one can compute the canonical representation $\mathcal{F}'$ of $(\mathcal{U}, \mathcal{F})$. Moreover, the size of the canonical representation is at most $O(|\mathcal{U}|w^3 \log^2 |\mathcal{U}|)$ and the space requirement of the algorithm is $\widetilde{O}(|\mathcal{F}'|) = \widetilde{O}(|\mathcal{U}|w^3)$.*

Proof: For the case of axis-parallel rectangles and fat triangles, first we use Lemma 2.4.2 and Lemma 2.4.3 to get the set $\mathcal{F}'_{\mathsf{total}}$ offline which require $\widetilde{O}(\mathcal{F}'_{\mathsf{total}}) = \widetilde{O}(|\mathcal{U}|w^3 \log^2 |\mathcal{U}|)$ memory space. Then by making one pass over the stream of sets $\mathcal{F}$, we can find the canonical representation $\mathcal{F}'$ by picking all the sets $S' \in \mathcal{F}'_{\mathsf{total}}$ such that $S' \cap \mathcal{U} \subseteq S \cap \mathcal{U}$ for some $S \in \mathcal{F}$. For discs however, we just make one pass over the sets $\mathcal{F}$ and keep a maximal subset $\mathcal{F}' \subseteq \mathcal{F}$ such that for each pair of sets $S'_1, S'_2 \in \mathcal{F}'$ their projection on $\mathcal{U}$ are different, i.e., $S'_1 \cap \mathcal{U} \neq S'_2 \cap \mathcal{U}$. By a standard technique of Clarkson and Shor [50], it can be proved that the size of the canonical representation, i.e., $|S'|$, is bounded by $O(|\mathcal{U}|w^2)$. Note that this is just counting the number of discs that contain at most w points, namely the at most w-level discs. □

2.4.2. Description of GeomSC Algorithm

The outline of the GeomSC algorithm (shown in Algorithm 3) is very similar to the IterSetCover algorithm presented earlier in Section 2.2.

In the first pass, the algorithm picks all the sets that cover a large number of yet-uncovered elements. Next, we sample S. Since we have removed all the ranges that have large size, in the first pass, the size of the remaining ranges *restricted to* the sample S is small. Therefore by Lemma 2.4.4, the canonical representation of $(\mathsf{S}, \mathcal{F}_{\mathsf{S}})$ has small size and we can afford to store it in the memory. We use Lemma 2.4.4 to compute the canonical representation $\mathcal{F}_{\mathsf{S}}$ in one pass. The algorithm then uses the sets in $\mathcal{F}_{\mathsf{S}}$ to find a cover $\mathtt{sol}_{\mathsf{S}}$ for the points of S. Next, in one additional pass, the algorithm replaces each set in $\mathtt{sol}_{\mathsf{S}}$ by one of its supersets in $\mathcal{F}$.

Finally, note that in IterSetCover, we are assuming that the size of the optimal solution is $O(k)$. Thus it is enough to stop the iterations once the number of uncovered elements is less than k. Then we can pick an arbitrary set for each of the uncovered elements. This would add only k more sets to the solution. Using this idea, we can reduce the size of the sampled elements down to $c\rho k(\frac{n}{k})^{\delta} \log m \log n$ which would help us in getting near-linear space in the geometric setting. Note that the final pass of the algorithm can be embedded into the previous passes but for the sake of clarity we write it separately.

2.4.3. Analysis

By a similar approach to what we used in Section 2.2 to analyze the pass count and approximation guarantee of IterSetCover algorithm, we can show that the number of passes of the GeomSC algorithm is $3/\delta + 1$ (which can be reduced to $3/\delta$ with minor changes), and the algorithm returns an $O(\rho/\delta)$-approximate solution. Next, we analyze the space usage and the correctness of the algorithm. Note that our analysis in this section only works for $\delta \leq 1/4$.

Lemma 2.4.5. *The algorithm uses $\widetilde{O}(n)$ space.*

Proof: Consider an iteration of the algorithm. The memory space used in the first pass of each iteration is $\widetilde{O}(n)$. The size of S is $c\rho k(n/k)^{\delta} \log m \log n$ and after the first pass the size

Algorithm 3 A streaming algorithm for the Points-Shapes Set Cover problem.

procedure GEOMSC($(\mathcal{U}, \mathcal{F}), \delta$)
  for $k \in \{2^i \mid 0 \leq i \leq \log n\}$ in parallel **do** ▷ $n = |\mathcal{U}|$
    let $\mathsf{L} \leftarrow \mathcal{U}$ and SOL $\leftarrow \emptyset$
    for $i = 1$ to $1/\delta$ **do**
      for all $S \in \mathcal{F}$ **do** ▷ one pass
        if $|S \cap \mathsf{L}| \geq |\mathcal{U}|/k$ **then**
          SOL $\leftarrow$ SOL $\cup \{S\}$
          $\mathsf{L} \leftarrow \mathsf{L} \setminus S$
      $\mathsf{S} \leftarrow$ sample of L of size $c\rho k(n/k)^\delta \log m \log n$
      $\mathcal{F}_\mathsf{S} \leftarrow$ COMPCANONICALREP($\mathsf{S}, \mathcal{F}, \frac{|\mathsf{S}|}{k}$) ▷ one pass
      $\mathtt{sol}_\mathsf{S} \leftarrow$ OFFLINESC($\mathsf{S}, \mathcal{F}_\mathsf{S}$)
      for $S \in \mathcal{F}$ **do** ▷ one pass
        if $\exists S' \in \mathtt{sol}_\mathsf{S}$ s.t. $S' \cap \mathsf{S} \subseteq S \cap \mathsf{S}$ **then**
          SOL $\leftarrow$ SOL $\cup \{S\}$
          $\mathtt{sol}_\mathsf{S} \leftarrow \mathtt{sol}_\mathsf{S} \setminus \{S'\}$
          $\mathsf{L} \leftarrow \mathsf{L} \setminus S$
    for $S \in \mathcal{F}$ **do** ▷ final pass
      if $S \cap \mathsf{L} \neq \emptyset$ **then**
        SOL $\leftarrow$ SOL $\cup \{S\}$
        $\mathsf{L} \leftarrow \mathsf{L} \setminus S$
  return smallest SOL computed in parallel

of each set is at most $|\mathcal{U}|/k$. Thus using Chernoff bound for each set $S \in \mathcal{F} \setminus$ SOL,

$$\Pr|S \cap \mathsf{S}| > (1+2)\frac{|\mathcal{U}|}{k} \times \frac{|\mathsf{S}|}{|\mathcal{U}|} \leq \exp\left(-\frac{4|\mathsf{S}|}{3k}\right) \leq (\frac{1}{m})^{c+1}.$$

Thus, with probability at least $1 - m^{-c}$ (by the union bound), all the sets that are not picked in the first pass, cover at most $3|\mathsf{S}|/k = c\rho(n/k)^\delta \log m \log n$ elements of S. Therefore, we can use Lemma 2.4.4 to show that the number of sets in the canonical representation of $(\mathsf{S}, \mathcal{F}_\mathsf{S})$ is at most

$$O(|\mathsf{S}|\left(\frac{3|\mathsf{S}|}{k}\right)^3 \log^2|\mathsf{S}|) = O(\rho^4 n \log^4 m \log^6 n),$$

as long as $\delta \leq 1/4$. To store each set in a canonical representation of $(\mathsf{S}, \mathcal{F})$ only constant space is required. Moreover, by Lemma 2.4.4, the space requirement of the second pass is $\widetilde{O}(|\mathcal{F}_\mathsf{S}|) = \widetilde{O}(n)$. Therefore, the total required space is $\widetilde{O}(n)$ and the lemma follows. □

Theorem 2.4.6. *Given a set system defined over a set $\mathcal{U}$ of n points in the plane, and a*

set of m ranges $\mathcal{F}$ (which are either all disks, axis-parallel rectangles, or fat triangles). Let ρ be the quality of approximation to the offline set-cover solver we have, and let $0 < \delta < 1/4$ be an arbitrary parameter.

Setting $\delta = 1/4$, the algorithm GeomSC *with high probability returns an $O(\rho)$-approximate solution of the optimal set cover solution for the instance $(\mathcal{U}, \mathcal{F})$. This algorithm uses $\widetilde{O}(n)$ space, and performs constant passes over the data.*

Proof: As before consider the run of the algorithm in which $|\mathrm{OPT}| \leq k < 2|\mathrm{OPT}|$. Let V be the set of uncovered elements L at the beginning of the iteration and note that the total number of sets that is picked during the iteration is at most $(1 + c_1\rho)k$ where c_1 is the constant defined in Definition 2.4.4. Let $\mathcal{G}$ denote all possible such covers, that is $\mathcal{G} = \left\{\mathcal{F}' \subseteq \mathcal{F} \;\middle|\; |\mathcal{F}'| \leq (1 + c_1\rho)k\right\}$. Let $\mathcal{H}$ be the collection that contains all possible set of uncovered elements at the end of the iteration, defined as $\mathcal{H} = \left\{\mathsf{V} \setminus \bigcup_{S \in \mathcal{C}} S \;\middle|\; \mathcal{C} \in \mathcal{G}\right\}$. Set $p = (k/n)^{\delta}$, $\varepsilon = 1/2$ and $q = m^{-c}$. Since for large enough c, $\frac{c'}{\varepsilon^2 p}(\log|\mathcal{H}| \log\frac{1}{p} + \log\frac{1}{q}) \leq c\rho k(n/k)^{\delta} \log m \log n = |\mathsf{S}|$ with probability at least $1 - m^{-c}$, by Lemma 2.2.5, the set of sampled elements S is a relative (p, ε)-approximation sample of $(\mathsf{V}, \mathcal{H})$.

Let $\mathcal{C} \subseteq \mathcal{F}$ be the collection of sets picked in the third pass of the algorithm that covers all elements in S. By Lemma 2.4.4, $|\mathcal{C}| \leq c_1\rho k$ for some constant c_1. Since with high probability S is a relative (p, ε)-approximation sample of $(\mathsf{V}, \mathcal{H})$, the number of uncovered elements of V (or L) after adding $\mathcal{C}$ to SOL is at most $\varepsilon p|\mathsf{V}| \leq |\mathcal{U}|(k/n)^{\delta}$. Thus with probability at least $(1 - m^{-c})$, in each iteration and by adding $O(\rho k)$ sets, the number of uncovered elements reduces by a factor of $(n/k)^{\delta}$.

Therefore, after 4 iterations (for $\delta = 1/4$) the algorithm picks $O(\rho k)$ sets and with high probability the number of uncovered elements is at most $n(k/n)^{\delta/\delta} = k$. Thus, in the final pass the algorithm only adds k sets to the solution SOL, and hence the approximation factor of the algorithm is $O(\rho)$. □

Remark 2.4.7. The result of Theorem 2.4.6 is similar to the result of Agarwal and Pan [3] – except that their algorithm performs $O(\log n)$ iterations over the data, while the algorithm of Theorem 2.4.6 performs only a constant number of iterations. In particular, one can use the algorithm of Agarwal and Pan [3] as the offline solver.

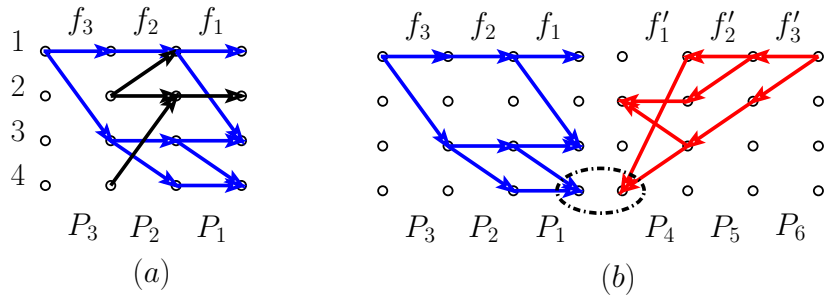


Figure 2.5.1: (a) shows an example of the communication Set Chasing$(4,3)$ and (b) is an instance of the communication Intersection Set Chasing$(4,3)$.

2.5. Lower bound for Multipass Algorithms

In this section we give lower bound on the memory space of multipass streaming algorithms for the Set Cover problem. Our main result is $\Omega(mn^{\delta})$ space for streaming algorithms that return an optimal solution of the Set Cover problem in $O(1/\delta)$ passes for $m = O(n)$. Our approach is to reduce the communication Intersection Set Chasing(n,p) problem introduced by Guruswami and Onak [90] to the communication Set Cover problem.

Consider a communication problem P with n players $P_1, \cdots, P_n$. The problem P is a (n,r)-communication problem if players communicate in r rounds and in each round they speak in order $P_1, \cdots, P_n$. At the end of the rth round P_n should return the solution. Moreover we assume private randomness and public messages. In what follows we define the communication Set Chasing and Intersection Set Chasing problems.

Definition 2.5.1. The Set Chasing(n,p) problem is a $(p, p-1)$ communication problem in which the player i has a function $f_i : [n] \to 2^{[n]}$ and the goal is to compute $\vec{f}_1(\vec{f}_2(\cdots \vec{f}_p(\{1\})\cdots))$ where $\vec{f}_i(S) = \bigcup_{s\in S} f_i(s)$. Figure 2.5.1(a) is an instance of the communication Set Chasing$(4,3)$.

Definition 2.5.2. The Intersection Set Chasing(n,p) is a $(2p, p-1)$ communication problem in which the first p players have an instance of the Set Chasing(n,p) problem and the other p players have another instance of the Set Chasing(n,p) problem. The output of the Intersection Set Chasing(n,p) is 1 if the solutions of the two instances of the Set Chasing(n,p) intersect and 0 otherwise. Figure 2.5.1(b) shows an instance of the Intersection Set Chasing $(4,3)$. The function f_i of each player P_i is specified by a set of directed edges form a copy of vertices labeled $\{1,\cdots,n\}$ to another copy of vertices labeled $\{1,\cdots,n\}$.

The communication Set Chasing problem is a generalization of the well-known commu-

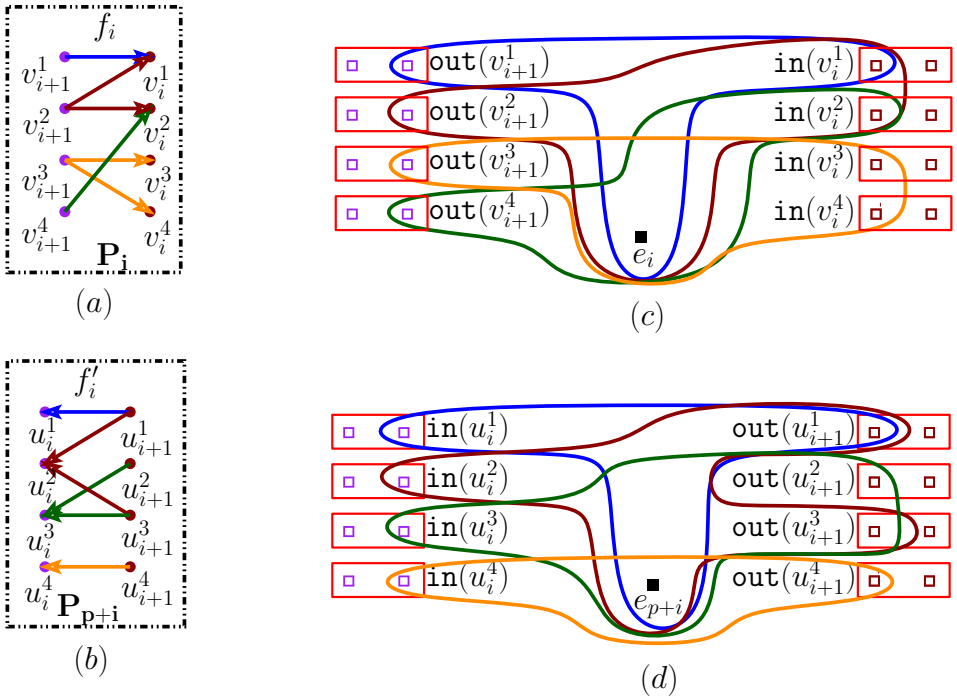


Figure 2.5.2: The gadgets used in the reduction of the communication Intersection Set Chasing problem to the communication Set Cover problem. (a) and (c) shows the construction of the gadget for players 1 to p and (c) and (d) shows the construction of the gadget for players $p+1$ to $2p$.

nication Pointer Chasing problem in which player i has a function $f_i : [n] \to [n]$ and the goal is to compute $f_1(f_2(\cdots f_p(1)\cdots))$.

[90] showed that any randomized protocol that solves Intersection Set Chasing(n,p) with error probability less than $1/10$, requires $\Omega(\frac{n^{1+1/(2p)}}{p^{16}\log^{3/2} n})$ bits of communication where n is sufficiently large and $p \leq \frac{\log n}{\log\log n}$. In Theorem 2.5.4, we reduce the communication Intersection Set Chasing problem to the communication Set Cover problem and then give the first superlinear memory lower bound for the streaming Set Cover problem.

Definition 2.5.3 (Communication Set Cover$(\mathcal{U},\mathcal{F},p)$ Problem). The communication Set Cover(n,p) is a $(p,p-1)$ communication problem in which a collection of elements $\mathcal{U}$ is given to all players and each player i has a collection of subsets of $\mathcal{U}$, $\mathcal{F}_i$. The goal is to solve Set Cover$(\mathcal{U},\mathcal{F}_1\cup\cdots\cup\mathcal{F}_p)$ using the minimum number of communication bits.

Theorem 2.5.4. *Any $(1/2\delta-1)$ passes streaming algorithm that solves the* Set Cover$(\mathcal{U},\mathcal{F})$ *optimally with constant probability of error requires $\widetilde{\Omega}(mn^{\delta})$ memory space where $\delta \geq \frac{\log\log n}{\log n}$ and $m = O(n)$.*

Consider an instance ISC of the communication Intersection Set Chasing(n, p). We construct an instance of the communication Set Cover$(\mathcal{U}, \mathcal{F}, 2p)$ problem such that solving Set Cover$(\mathcal{U}, \mathcal{F})$ optimally determines whether the output of ISC is 1 or not.

The instance ISC consists of $2p$ players. Each player $1, \cdots, p$ has a function $f_i : [n] \to 2^{[n]}$ and each player $p+1, \cdots, 2p$ has a function $f'_i : [n] \to 2^{[n]}$ (see Figure 2.5.1). In ISC, each function f_i is shown by a set of vertices $v_i^1, \cdots, v_i^n$ and $v_{i+1}^1, \cdots, v_{i+1}^n$ such that there is a directed edge from v_{i+1}^j to v_i^ℓ if and only if $\ell \in f_i(j)$. Similarly, each function f'_i is denoted by a set of vertices $u_i^1, \cdots, u_i^n$ and $u_{i+1}^1, \cdots, u_{i+1}^n$ such that there is a directed edge from u_{i+1}^j to u_i^ℓ if and only if $\ell \in f'_i(j)$ (see Figure 2.5.4(a) and Figure 2.5.4(b)).

In the corresponding communication SetCover instance of ISC, we add two elements $\texttt{in}(v_i^j)$ and $\texttt{out}(v_i^j)$ per each vertex v_i^j where $i \leq p+1, j \leq n$. We also add two elements $\texttt{in}(u_i^j)$ and $\texttt{out}(u_i^j)$ per each vertex u_i^j where $i \leq p+1, j \leq n$. In addition to these elements, for each player i, we add an element e_i (see Figure 2.5.4(c) and Figure 2.5.4(d)).

Next, we define a collection of sets in the corresponding Set Cover instance of ISC. For each player P_i, where $1 \leq i \leq p$, we add a single set S_i^j containing $\texttt{out}(v_{i+1}^j)$ and $\texttt{in}(v_i^\ell)$ for all out-going edges (v_{i+1}^j, v_i^ℓ). Moreover, all S_i^j sets contain the element e_i. Next, for each vertex v_i^j we add a set R_i^j that contains the two corresponding elements of v_i^j, $\texttt{in}(v_i^j)$ and $\texttt{out}(v_i^j)$. In Figure 2.5.4(c), the red rectangles denote R-type sets and the curves denote S-type sets for the first half of the players.

Similarly to the sets corresponding to players 1 to p, for each player P_{p+i} where $1 \leq i \leq p$, we add a set S_{p+i}^j containing $\texttt{in}(u_i^j)$ and $\texttt{out}(u_{i+1}^\ell)$ for all in-coming edges (u_{i+1}^ℓ, u_i^j) of u_i^j (denoting $f_i'^{-1}(j)$). The set S_{p+i}^j contains the element e_{p+i} too. Next, for each vertex u_i^j we add a set T_{p+i}^j that contains the two corresponding elements of u_i^j, $\texttt{in}(u_i^j)$ and $\texttt{out}(u_i^j)$. In Figure 2.5.4(d), the red rectangles denote T-type sets and the curves denote S-type sets for the second half of the players.

At the end, we merge v_1^is and u_1^is as shown in Figure 2.5.3. After merging the corresponding sets of v_1^js $(R_1^1, \cdots, R_1^n)$ and the corresponding sets of u_1^js $(T_1^1, \cdots, T_1^n)$, we call the merged sets $T_1^1, \cdots, T_1^n$.

The main claim is that if the solution of ISC is 1 then the size of an optimal solution of its corresponding SetCover instance SC is $(2p+1)n+1$; otherwise, it is $(2p+1)n+2$.

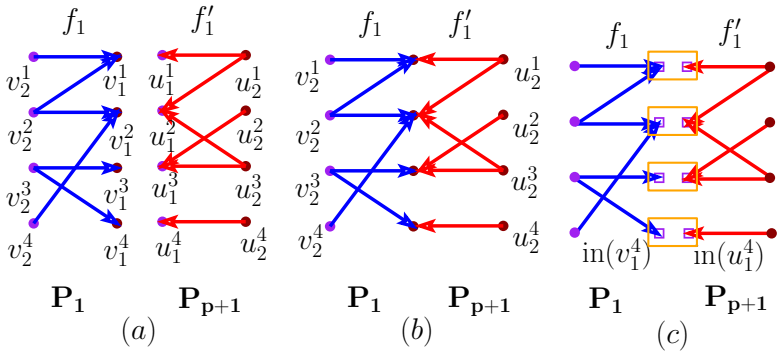


Figure 2.5.3: In (b) two Set Chasing instances merge in their first set of vertices and (c) shows the corresponding gadgets of these merged vertices in the communication Set Cover.

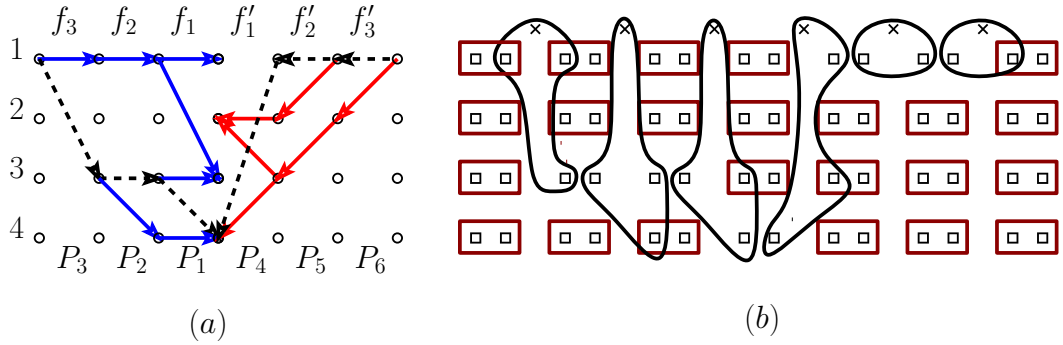


Figure 2.5.4: In (a), path Q is shown with black dashed arcs and (b) shows the corresponding cover of path Q.

Lemma 2.5.5. *The size of any feasible solution of* SC *is at least* $(2p+1)n+1$.

Proof: For each player i $(1 \leq i \leq p)$, since $\texttt{out}(v_{i+1}^j)$s are only covered by R_{i+1}^j and S_i^j, at least n sets are required to cover $\texttt{out}(v_{i+1}^1), \cdots, \texttt{out}(v_{i+1}^n)$. Moreover for player P_p, since $\texttt{in}(v_{p+1}^j)$s are only covered by R_{p+1}^j and e_p is only covered by S_p^1, all $n+1$ sets $R_{p+1}^1, \cdots, R_{p+1}^n, S_p^1$ must be selected in any feasible solution of SC.

Similarly for each player $p+i$ $(1 \leq i \leq p)$, since $\text{in}(u_i^j)$s are only covered by T_i^j and S_{p+i}^j, at least n sets are required to cover $\texttt{in}(u_i^1), \cdots, \texttt{in}(u_i^n)$. Moreover, considering $u_{p+1}^1, \cdots, u_{p+1}^n$, since $\texttt{in}(u_{p+1}^j)$ is only covered by T_{p+1}^j, all n sets $T_{p+1}^1, \cdots, T_{p+1}^n$ must be selected in any feasible solution of SC.

All together, at least $(2p+1)n+1$ sets should be selected in any feasible solution of SC.□

Lemma 2.5.6. *Suppose that the solution of* ISC *is* 1. *Then the size of an optimal solution of its corresponding* Set Cover *instance is exactly* $(2p+1)n+1$.

Proof: By Lemma 2.5.5, the size of an optimal solution of S is at least $(2p+1)n+1$. Here we prove that $(2p+1)n+1$ sets suffice when the solution of ISC is 1. Let $Q =$

$v^1_{p+1}, v^{j_p}_p, \ldots, v^{j_2}_2, v^{j_1}_1, u^{\ell_1}_1, u^{\ell_2}_2, \ldots, u^{\ell_p}_p, u^1_{p+1}$ be a path in ISC such that $j_1 = \ell_1$ (since the solution of ISC is 1 such a path exists). The corresponding solution to Q can be constructed as follows (See Figure 2.5.4):

- Pick S^1_p and all R^j_{p+1}s ($n+1$ sets).
- For each $v^{j_i}_i$ in Q where $1 < i \leq p$, pick the set $S^{j_i}_{i-1}$ in the solution. Moreover, for each such i pick all sets R^j_i where $j \neq j_i$ ($n(p-1)$ sets).
- For $v^{j_1}_1$ (or $u^{\ell_1}_1$), pick the set $S^{j_1}_{p+1}$. Moreover, pick all sets T^j_1 where $j \neq j_1$ (n sets).
- For each $u^{\ell_i}_i$ in Q where $1 < i \leq p$, pick the set $S^{\ell_i}_{p+i}$ in the solution. Moreover, for each such i pick all sets T^ℓ_i where $\ell \neq \ell_i$ ($n(p-1)$ sets).
- Pick all T^j_{p+1}s (n sets).

It is straightforward to see that the solution constructed above is a feasible solution. □

Lemma 2.5.7. *Suppose that the size of an optimal solution of the corresponding* SetCover *instance of* ISC*,* SC*, is* $(2p+1)n+1$*. Then the solution of* ISC *is* 1*.*

Proof: As we proved earlier in Lemma 2.5.5, any feasible solution of SC picks $R^1_{p+1}, \cdots, R^n_{p+1}, S^1_p$ and $T^1_{p+1}, \cdots, T^n_{p+1}$. Moreover, we proved that for each $1 \leq i < p$, at least n sets should be selected from $R^1_{i+1}, \cdots, R^n_{i+1}, S^1_i, \cdots, S^n_i$. Similarly, for each $1 \leq i \leq p$, at least n sets should be selected from $T^1_i, \cdots, T^n_i$, $S^1_{p+i}, \cdots, S^n_{p+1}$. Thus if a feasible solution of SC, OPT, is of size $(2p+1)n+1$, it has exactly n sets from each specified group.

Next we consider the first half of the players and second half of the players separately. Consider i such that $1 \leq i < p$. Let $S^{j_1}_i, \cdots, S^{j_k}_i$ be the sets picked in the optimal solution (because of e_i there should be at least one set of form S^j_i in OPT). Since each $\texttt{out}(v^j_{i+1})$ is only covered by S^j_i and R^j_{i+1}, for all $j \notin \{j_1, \ldots, j_k\}$, R^j_{i+1} should be selected in OPT. Moreover, for all $j \in \{j_1, \cdots, j_k\}$, R^j_{i+1} should not be contained in OPT (otherwise the size of OPT would be larger than $(2p+1)n+1$). Consider $j \in \{j_1, \ldots, j_k\}$. Since R^j_{i+1} is not in OPT, there should be a set S^ℓ_{i+1} selected in OPT such that $\texttt{in}(v^j_{i+1})$ is contained in S^ℓ_{i+1}. Thus by considering S_is in a decreasing order and using induction, if S^j_i is in OPT then v^j_{i+1} is reachable form v^1_{p+1}.

Next consider a set S^j_{p+i} that is selected in OPT ($1 \leq i \leq p$). By similar argument, T^j_i is not in OPT and there exists a set S^ℓ_{p+i-1} (or S^ℓ_1 if $i = 1$) in OPT such that $\texttt{out}(u^j_i)$

is contained in S^{ℓ}_{p+i-1}. Let $u^{\ell_1}_{i+1}, \cdots, u^{\ell_k}_{i+1}$ be the set of vertices whose corresponding $\texttt{out}$ elements are in S^{j}_{p+i}. Then by induction, there exists an index r such that v^r_1 is reachable from v^1_{p+1} and u^r_1 is also reachable from all $u^{\ell_1}_{i+1}, \cdots, u^{\ell_k}_{i+1}$. Moreover, the way we constructed the instance SC guarantees that all sets $S^1_{2p}, \cdots, S^n_{2p}$ contains $\texttt{out}(u^1_{p+1})$. Hence if the size of an optimal solution of SC is $(2p+1)n+1$ then the solution of ISC is 1. □

Corollary 2.5.8. Intersection Set Chasing(n,p) *returns* 1 *if and only if the size of optimal solution of its corresponding* Set Cover *instance (as described here) is* $(2p+1)n+1$.

Observation 2.5.9. *Any streaming algorithm for* Set Cover*, $\mathcal{I}$, that in ℓ passes solves the problem optimally with a probability of error* $\texttt{err}$ *and consumes s memory space, solves the corresponding communication* Set Cover *problem in ℓ rounds using $O(s\ell^2)$ bits of communication with probability error* $\texttt{err}$.

Proof: Starting from player P_1, each player runs $\mathcal{I}$ over its input sets and once P_i is done with its input, she sends the working memory of $\mathcal{I}$ publicly to other players. Then next player starts the same routine using the state of the working memory received from the previous player. Since $\mathcal{I}$ solves the Set Cover instance optimally after ℓ passes using $O(s)$ space with probability error $\texttt{err}$, applying $\mathcal{I}$ as a black box we can solve P in ℓ rounds using $O(s\ell^2)$ bits of communication with probability error $\texttt{err}$. □

Proof of Theorem 2.5.4: By Observation 2.5.9, any ℓ-round $O(s)$-space algorithm that solves streaming Set Cover $(\mathcal{U}, \mathcal{F})$ optimally can be used to solve the communication Set Cover$(\mathcal{U}, \mathcal{F}, p)$ problem in ℓ rounds using $O(s\ell^2)$ bits of communication. Moreover, by Corollary 2.5.8, we can decide the solution of the communication Intersection Set Chasing(n,p) by solving its corresponding communication Set Cover problem. Note that while working with the corresponding Set Cover instance of Intersection Set Chasing(n,p), all players know the collection of elements $\mathcal{U}$ and each player can construct its collection of sets $\mathcal{F}_i$ using f_i (or f'_i).

However, by a result of [90], we know that any protocol that solves the communication Intersection Set Chasing(n,p) problem with probability of error less than $1/10$, requires $\Omega(\frac{n^{1+1/(2p)}}{p^{16}\log^{3/2} n})$ bits of communication. Since in the corresponding Set Cover instance of the communication Intersection Set Chasing(n,p), $|\mathcal{U}| = (2p+1) \times 2n + 2p = O(np)$ and $|\mathcal{F}| \leq (2p+1)n + 2pn = O(np)$, any $(p-1)$-pass streaming algorithm that solves the

Set Cover problem optimally with a probability of error at most 1/10, requires $\Omega(\frac{n^{1+1/(2p)}}{p^{18}\log^{3/2} n})$ bits of communication. Then using Observation 2.5.9, since $\delta \geq \frac{\log\log n}{\log n}$, any $(\frac{1}{2\delta}-1)$-pass streaming algorithm of Set Cover that finds an optimal solution with error probability less than 1/10, requires $\widetilde{\Omega}(|\mathcal{F}| \cdot |\mathcal{U}|^{\delta})$ space. □

2.6. Lower Bound for Sparse Set Cover in Multiple Passes

In this part we give a stronger lower bound for the instances of the streaming Set Cover problem with sparse input sets. An instance of the Set Cover problem is s-Sparse Set Cover, if for each set $S \in \mathcal{F}$ we have $|S| \leq s$. We can us the same reduction approach described earlier in Section 2.5 to show that any $(1/2\delta - 1)$-pass streaming algorithm for s-Sparse Set Cover requires $\Omega(|\mathcal{F}|s)$ memory space if $s < |\mathcal{U}|^{\delta}$ and $\mathcal{F} = O(\mathcal{U})$. To prove this, we need to explain more details of the approach of [90] on the lower bound of the communication Intersection Set Chasing problem. They first obtained a lower bound for Equal Pointer Chasing(n,p) problem in which two instances of the communication Pointer Chasing(n,p) are given and the goal is to decide whether these two instances point to a same value or not; $f_p(\cdots f_1(1)\cdots) = f'_p(\cdots f'_1(1)\cdots)$.

Definition 2.6.1 (r-non-injective functions). A function $f : [n] \to [n]$ is called *r-non-injective* if there exists $A \subseteq [n]$ of size at least r and $b \in [n]$ such that for all $a \in A$, $f(a) = b$.

Definition 2.6.2 (Pointer Chasing Problem). Pointer Chasing(n,p) is a $(p, p-1)$ communication problem in which the player i has a function $f_i : [n] \to [n]$ and the goal is to compute $f_1(f_2(\cdots f_p(1)\cdots))$.

Definition 2.6.3 (Equal Limited Pointer Chasing Problem). Equal Pointer Chasing(n,p) is a $(2p, p-1)$ communication problem in which the first p players have an instance of the Pointer Chasing(n,p) problem and the other p players have another instance of the Pointer Chasing(n,p) problem. The output of the Equal Pointer Chasing(n,p) is 1 if the solutions of the two instances of Pointer Chasing(n,p) have the same value and 0 otherwise. Furthermore in another variant of pointer chasing problem, Equal Limited Pointer Chasing(n,p,r), if there exists r-non-injective function f_i, then the output is 1. Otherwise, the output is the

same as the value in Equal Pointer Chasing(n, p).

For a boolean communication problem P, OR_t(P) is defined to be OR of t instances of P and the output of OR_t(P) is `true` if and only if the output of any of the t instances is `true`. Using a direct sum argument, [90] showed that the communication complexity of OR_t(Equal Limited Pointer Chasing(n, p, r)) is t times the communication complexity of Equal Limited Pointer Chasing(n, p, r).

Lemma 2.6.4 ([90]). *Let n, p, t and r be positive integers such that $n \geq 5p$, $t \leq \frac{n}{4}$ and $r = O(\log n)$. Then the amount of bits of communication to solve OR_t(Equal Limited Pointer Chasing(n, p, r)) with error probability less than $1/3$ is $\Omega(\frac{tn}{p^{16} \log n}) - O(pt^2)$.*

Lemma 2.6.5 ([90]). *Let n, p, t and r be positive integers such that $t^{2p} r^{p-1} < n/10$. Then if there is a protocol that solves Intersection Set Chasing(n, p) with probability of error less than $1/10$ using C bits of communication, there is a protocol that solves OR_t(Equal Limited Pointer Chasing(n, p, r)) with probability of error at most $2/10$ using $C + 2p$ bits of communication.*

Consider an instance of OR_t(Equal Limited Pointer Chasing (n, p, r)) in which $t \leq n^{\delta}, r = \log(n), p = \frac{1}{2\delta} - 1$ where $\frac{1}{\delta} = o(\log n)$. By Lemma 2.6.4, the required amount of bits of communication to solve the instance with constant success probability is $\widetilde{\Omega}(tn)$. Then,applying Lemma 2.6.5, to solve the corresponding Intersection Set Chasing, $\widetilde{\Omega}(tn)$ bits of communication is required.

In the reduction from OR_t(Equal Limited Pointer Chasing(n, p, r)) to Intersection Set Chasing(n, p) (proof of Lemma 2.6.5), the r-non-injective property is preserved. In other words, in the corresponding Intersection Set Chasing instance each player's functions $f_i : [n] \to 2^{[n]}$ is union of t r-non-injective functions $f_i(a) := f_{i,1}(a) \cup \cdots \cup f_{i,t}(a)$[4]. Given that none of the $f_{i,j}$ functions is r-non-injective, the corresponding Set Cover instance will have sets of size at most rt (S-type sets are of size at most t for $1 \leq i \leq p$ and of size at most rt for $p+1 \leq i \leq 2p$). Since $r = O(\log n)$, the corresponding Set Cover instance is $\widetilde{O}(t)$-sparse. As we showed earlier in the reduction from Intersection Set Chasing to Set Cover, the number

[4]The Intersection Set Chasing instance is obtained by overlaying the t instances of Equal Pointer Chasing(n, p, r). To be more precise, the function of player i in instance j is $\pi_{i,j} \circ f_{i,j} \circ \pi^{-1}_{i+1,j}$ (π are randomly chosen permutation functions) and then stack the functions on top of each other.

of elements (and sets) in the corresponding Set Cover instance is $O(np)$. Thus we have the following result for s-Sparse Set Cover problem.

Theorem 2.6.6. *For $s \leq |\mathcal{U}|^{\delta}$, any streaming algorithm that solves s-Sparse Set Cover$(\mathcal{U}, \mathcal{F})$ optimally with probability of error less than $1/10$ in $(\frac{1}{2\delta} - 1)$ passes requires $\widetilde{\Omega}(|\mathcal{F}|s)$ memory space for $\mathcal{F} = O(\mathcal{U})$.*

Chapter 3

Streaming Fractional Set Cover

3.1. Introduction

The LP relaxation of Set Cover (called SetCover-LP) is a continuous relaxation of the problem where each set $S \in \mathcal{F}$ can be selected "fractionally", i.e., assigned a number x_S from $[0, 1]$, such that for each element e its "fractional coverage" $\sum_{S:e\in S} x_S$ is at least 1, and the sum $\sum_S x_S$ is minimized.

Despite the significant developments in streaming Set Cover [67, 42, 61, 97, 20, 29, 17], the results for the *fractional* variant of the problem are still unsatisfactory. To the best of our knowledge, it is not known whether there exists an efficient and accurate algorithm for this problem that uses only a logarithmic (or even a *poly logarithmic*) number of passes. This state of affairs is perhaps surprising, given the many recent developments on fast LP solvers [119, 174, 124, 12, 11, 167]. To the best of our knowledge, the only prior results on streaming Packing/Covering LPs were presented in [6], which studied the LP relaxation of Maximum Matching.

3.1.1. Our Results

In this chapter, we present the first $(1 + \varepsilon)$-approximation algorithm for the fractional Set Cover in the streaming model with constant number of passes. Our algorithm performs p passes over the data stream and uses $\widetilde{O}(mn^{O(\frac{1}{p\varepsilon})} + n)$ memory space to return a $(1 + \varepsilon)$ approximate solution of the LP relaxation of Set Cover for positive parameter $\varepsilon \leq 1/2$.

We emphasize that similarly to the previous work on variants of Set Cover in streaming setting, our result also holds for the *edge arrival* stream in which the pair of (S_i, e_j) (edges) are stored in the read-only repository and all elements of a set are not necessarily stored consecutively.

3.1.2. Other Related Work

Set Cover. The Set Cover problem was first studied in the streaming model in [155], which presented an $O(\log n)$-approximation algorithm in $O(\log n)$ passes and using $\widetilde{O}(n)$ space. This approximation factor and the number of passes can be improved to $O(\log n)$ by adapting the greedy algorithm *thresholding* idea presented in [56] . In the low space regime ($\widetilde{O}(n)$ space), Emek and Rosen [67] designed a *deterministic* single pass algorithm that achieves an $O(\sqrt{n})$-approximation. This is provably the best guarantee that one can hope for in a single pass even considering randomized algorithms. Later Chakrabarti and Wirth [42] generalized this result and provided a *tight* trade-off bounds for Set Cover in multiple passes. More precisely, they gave an $O(pn^{1/(p+1)})$-approximate algorithm in p-passes using $\widetilde{O}(n)$ space and proved that this is the best possible approximation ratio up to a factor of poly(p) in p passes and $\widetilde{O}(n)$ space.

A different line of work started by Demaine et al. [61] focused on designing a "low" approximation algorithm (between $\Theta(1)$ and $\Theta(\log n)$) in the smallest possible amount of space. In contrast to the results in the $\widetilde{O}(n)$ space regime, [61] showed that randomness is necessary: any constant pass deterministic algorithm requires $\Omega(mn)$ space to achieve constant approximation guarantee. Further, they provided a $O(4^p \log n)$-approximation algorithm that makes $O(4^p)$ passes and uses $\widetilde{O}(mn^{1/p} + n)$. Later Har-Peled et al. [97] improved the algorithm to a $2p$-pass $O(p \log n)$-approximation with memory space $\widetilde{O}(mn^{1/p} + n)$[1]. The result was further improved by Bateni et al. where they designed a p-pass algorithm that returns a $(1+\varepsilon)\log n$-approximate solution using $mn^{\Theta(1/p)}$ memory [29].

As for the lower bounds, Assadi et al. [20] presented a lower bound of $\Omega(mn/\alpha)$ memory for any single pass streaming algorithm that computes a α-approxime solution. For the problem of estimating the size of an optimal solution they prove $\Omega(mn/\alpha^2)$ memory lower bound.

[1] In streaming model, space complexity is of interest and one can assume exponentital computation power. In this case the algorithms of [61, 97] save a factor of $\log n$ in the approximation ratio.

For both settings, they complement the results with matching tight upper bounds. Furthermore, Assadi [17] proved a lower bound on the space complexity of streaming algorithms for Set Cover with multiple passes which is tight up to polylog factors: any α-approximation algorithm for Set Cover requires $\Omega(mn^{1/\alpha})$ space, even if it is allowed polylog(n) passes over the stream, and even if the sets are arriving in a random order in the stream. Further, [17] provided the matching upper bound: a $(2\alpha+1)$-pass algorithm that computes a $(\alpha+\varepsilon)$-approximate solution in $\widetilde{O}(\frac{mn^{1/\alpha}}{\varepsilon^2}+\frac{n}{\varepsilon})$ memory (assuming exponential computational resource).

Maximum Coverage. The first result on streaming Max k-Cover showed how to compute a $(1/4)$-approximate solution in one pass using $\widetilde{O}(kn)$ space [155]. It was improved by Badanidiyuru et al. [22] to a $(1/2-\varepsilon)$-approximation algorithm that requires $\widetilde{O}(n/\varepsilon)$ space. Moreover, their algorithm works for a more general problem of Submodular Maximization with cardinality constraints. This result was later generalized for the problem of non-monotone submodular maximization under constraints beyond cardinality [46]. Recently, McGregor and Vu [131] and Bateni et al. [29] independently obtained single pass $(1-1/e-\varepsilon)$-approximation with $\widetilde{O}(m/\varepsilon^2)$ space. On the lower bound side, [131] showed a lower bound of $\widetilde{\Omega}(m)$ for constant pass algorithm whose approximation is better than $(1-1/e)$. Moreover, [17] proved that any streaming $(1-\varepsilon)$-approximation algorithm of Max k-Cover in polylog(n) passes requires $\widetilde{\Omega}(m/\varepsilon^2)$ space even on random order streams and the case $k=O(1)$. This bound is also complemented by the $\widetilde{O}(mk/\varepsilon^2)$ and $\widetilde{O}(m/\varepsilon^3)$ algorithms of [29, 131]. For more detailed survey of the results on streaming Max k-Cover refer to [29, 131, 17, 107].

Covering/Packing LPs. The study of LPs in streaming model was first discussed in the work of Ahn and Guha [6] where they used *multiplicative weights update* (MWU) based techniques to solve the LP relaxation of Maximum (Weighted) Matching problem. They used the fact that MWU returns a near optimal fractional solution with small size support: first they solve the fractional matching problem, then solve the actual matching only considering the edges in the support of the returned fractional solution.

Our algorithm is also based on the MWU method, which is one of the main key techniques in designing fast approximation algorithms for Covering and Packing LPs [150, 173, 76, 16]. We note that the MWU method has been previously studied in the context of *streaming* and *distributed* algorithms, leading to efficient algorithms for a wide range of graph optimization problems [6, 23, 7].

For a related problem, *covering integer LP* (covering ILP), Assadi et al. [20] designed a one pass streaming algorithm that estimates the optimal solution of $\{\min \mathbf{c}^\top \mathbf{x} \mid \mathbf{A}^\top \mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \{0,1\}^n\}$ within a factor of α using $\widetilde{O}(\frac{mn}{\alpha^2} \cdot b_{\max} + m + n \cdot b_{\max})$ where $b_{\max}$ denotes the largest entry of $\mathbf{b}$. In this problem, they assume that columns of $\mathbf{A}$, constriants, are given one by one in the stream.

In a different regime, [65] studied approximating the feasibility LP in streaming model with additive approximation. Their algorithm performs two passes and is most efficient when the input is dense.

3.1.3. Our Techniques

Preprocessing. Let k denote the value of the optimal solution. The algorithm starts by picking a uniform *fractional* vector (each entry of value $O(\frac{k}{m})$) which covers all frequently occurring elements (those appearing in $\Omega(\frac{m}{k})$ sets), and updates the uncovered elements in one pass. This step considerably reduces the memory usage as the uncovered elements have now lower occurrence (roughly $\frac{m}{k}$). Note that we do not need to assume the knowledge of the correct value k: in parallel we try all powers of $(1+\varepsilon)$, denoting our guess by ℓ.

Multiplicative Weight Update (MWU). To cover the remaining elements, we employ the MWU framework and show how to implement it in the streaming setting. In each iteration of MWU, we have a probability distribution $\mathbf{p}$ corresponding to the constraints (elements) and we need to satisfy the *average* covering constraint. More precisely, we need an *oracle* that assigns values to x_S for each set S so that $\sum_S p_S x_S \geq 1$ subject to $\|\mathbf{x}\|_1 \leq \ell$, where p_S is the sum of probabilities of the elements in the set S. Then, the algorithm needs to update $\mathbf{p}$ according to the amount each element has been covered by the oracle's solution. The simple greedy realization of the oracle can be implemented in the streaming

setting efficiently by computing all p_S while reading the stream in one pass, then choosing the heaviest set (i.e., the set with largest p_S) and setting its x_S to ℓ. This approach works, except that the number of rounds T required by the MWU framework is large. In fact, $T = \Omega(\frac{\phi \log n}{\varepsilon^2})$, where ϕ is the width parameter (the maximum amount an oracle solution may over-cover an element), which is $\Theta(\ell)$ in this naïve realization. Next, we show how to decrease T in two steps.

Step 1. A first hope would be that there is a more efficient implementation of the oracle which gives a better width parameter. Nonetheless, no matter how the oracle is implemented, if all sets in $\mathcal{F}$ contain a fixed element e, then the width is inevitably $\Omega(\ell)$. This observation implies that we need to work with a different set system that has small width, but at the same time, it has the same objective value as of the optimal solution. Consequently, we consider the *extended set system* where we replace $\mathcal{F}$ with all subsets of the sets in $\mathcal{F}$. This extended system preserves the optimality, and under this system we may avoid over-covering elements and obtain $T = O(\log n)$ (for constant ε).

In order to turn a solution in our set system into a solution in the extended set system with small width, we need to remove the repeated elements from the sets in the solution so that every covered element appears exactly once, and thereby getting constant width. However, as a side effect, this reduces the total weight of the solution ($\sum_{S \in \text{SOL}} p_S x_S$), and thus the average covering constraint might not be satisfied anymore. In fact, we need to come up with a guarantee that, on one hand, is preserved under the pruning step, and on the other hand, implies that the solution has large enough total weight

Therefore, to fulfill the average constraint under the pruning step, the oracle must instead solve the *maximum coverage* problem: given a budget, choose sets to cover the largest (fractional) amount of elements. We first show that this problem can be solved approximately via the MWU framework using the simple oracle that picks the heaviest set, but this MWU algorithm still requires T passes over the data. To improve the number of passes, we perform *element sampling* and apply the MWU algorithm to find an approximate maximum coverage of a small number of sampled elements, whose subproblem can be stored in memory. Fortunately, while the number of fractional solutions to maximum coverage is unbounded,

by exploiting the structure of the solutions returned by the MWU method, we can limit the number of plausible solutions of this oracle and approximately solve the average constraint, thereby reducing the space usage to $\widetilde{O}(m)$ for a $O(\frac{\log n}{\varepsilon^2})$-pass algorithm.

Step 2. To further reduce the number of required passes, we observe that the weights of the constraints change slowly. Thus, in a single pass, we can sample the elements for multiple rounds in advance, and then perform rejection (sub-)sampling to obtain an unbiased set of samples for each subsequent round. This will lead to a streaming algorithm with p passes and $mn^{O(1/p)}$ space.

Extension. We also extend our result to handle general covering LPs. More specifically, in the LP relaxation of Set Cover, maximize $\mathbf{c}^\top \mathbf{x}$ subject to $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$, $\mathbf{A}$ has entries from $\{0, 1\}$ whereas entries of $\mathbf{b}$ and $\mathbf{c}$ are all ones. If the non-zero entries instead belong to a range $[1, M]$, we increase the number of sampled elements by $\text{poly}(M)$ to handle discrepancies between coefficients, leading to a $\text{poly}(M)$-multiplicative overhead in the space usage.

3.2. MWU Framework of the Streaming Algorithm for Fractional Set Cover

In this section, we present a basic streaming algorithm that computes a $(1+\varepsilon)$-approximate solution of the LP-relaxation of Set Cover for any $\varepsilon > 0$ via the MWU framework. We will, in the next section, improve it into an efficient algorithm that achieves the claimed $O(p)$ passes and $\widetilde{O}(mn^{1/p})$ space complexity.

SetCover-LP ▷ Input: $\mathcal{U}, \mathcal{F}$

$$\begin{aligned} \text{minimize} \quad & \sum_{S \in \mathcal{F}} x_S & \\ \text{subject to} \quad & \sum_{S: e \in S} x_S \geq 1 & \forall e \in \mathcal{U} \\ & x_S \geq 0 & \forall S \in \mathcal{F} \end{aligned}$$

Figure 3.2.1: LP relaxation of Set Cover.

Algorithm 4 FracSetCover returns a $(1+\varepsilon)$-approximate solution of SetCover-LP.

procedure FracSetCover(ε)
  ▷ finds a feasible $(1+\varepsilon)$-approximate solution in $O(\frac{\log n}{\varepsilon})$ iterations
  for $\ell \in \{(1+\varepsilon/3)^i \mid 0 \le i \le \log_{1+\varepsilon/3} n\}$ in parallel **do**
    $\mathbf{x}_\ell \leftarrow$ FeasibilityTest$(\ell, \varepsilon/3)$
    ▷ returns a solution of objective value at most $(1+\varepsilon/3)\ell$ when $\ell \ge k$.
  return x_{ℓ^*} where $\ell^* \leftarrow \min\{\ell : x_\ell$ is not INFEASIBLE$\}$

Let $\mathcal{U}$ and $\mathcal{F}$ be the ground set of elements and the collection of sets, respectively, and recall that $|\mathcal{U}| = n$ and $|\mathcal{F}| = m$. Let $\mathbf{x} \in \mathbb{R}^m$ be a vector indexed by the sets in $\mathcal{F}$, where x_S denotes the value assigned to the set S. Our goal is to compute an approximate solution to the LP in Figure 3.2.1. Throughout the analysis we assume $\varepsilon \le 1/2$, and ignore the case where some element never appears in any set, as it is easy to detect in a single pass that no cover is valid. For ease of reading, we write $\widetilde{O}$ and $\widetilde{\Theta}$ to hide polylog$(m, n, \frac{1}{\varepsilon})$ factors.

Outline of the algorithm. Let k denote the optimal objective value, and $0 < \varepsilon \le 1/2$ be a parameter. The outline of the algorithm is shown in FracSetCover (Algorithm 4). This algorithm makes calls to the subroutine FeasibilityTest, that given a parameter ℓ, with high probability, either returns a solution of objective value at most $(1+\varepsilon/3)\ell$, or detects that the optimal objective value exceeds ℓ. Consequently, we may search for the right value of ℓ by considering all values in $\{(1+\varepsilon/3)^i \mid 0 \le i \le \log_{1+\varepsilon/3} n\}$. As for some value of ℓ it holds that $k \le \ell \le k(1+\varepsilon/3)$, we obtain a solution of size $(1+\varepsilon/3)\ell \le (1+\varepsilon/3)(1+\varepsilon/3)k \le (1+\varepsilon)k$ which gives an approximation factor $(1+\varepsilon)$. This whole process of searching for k increases the space complexity of the algorithm by at most a multiplicative factor of $\log_{1+\varepsilon/3} n \approx \frac{3\log n}{\varepsilon}$.

The FeasibilityTest subroutine employs the multiplicative weights update method (MWU) which is described next.

3.2.1. Preliminaries of the MWU method for solving covering LPs

In the following, we describe the MWU framework. The claims presented here are standard results of the MWU method. For more details, see e.g. Section 3 of [16]. Note that we introduce the general LP notation as it simplifies the presentation later on.

Let $\mathbf{Ax} \ge \mathbf{b}$ be a set of linear constraints, and let $\mathcal{P} \triangleq \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x} \ge \mathbf{0}\}$ be the polytope of the non-negative orthant. For a given error parameter $0 < \beta < 1$, we would like to solve

an approximate version of the feasibility problem by doing one of the following:

- Compute $\hat{\mathbf{x}} \in \mathcal{P}$ such that $\mathbf{A}_i\hat{\mathbf{x}} - b_i \geq -\beta$ for *every* constraint i.
- Correctly report that the system $\mathbf{Ax} \geq \mathbf{b}$ has no solution in $\mathcal{P}$.

The MWU method solves this problem assuming the existence of the following oracle that takes a distribution $\mathbf{p}$ over the constraints and finds a solution $\hat{\mathbf{x}}$ that satisfies the constraints on average over $\mathbf{p}$.

Definition 3.2.1. Let $\phi \geq 1$ be a width parameter and $0 < \beta < 1$ be an error parameter. A $(1, \phi)$-*bounded* $(\beta/3)$-*approximate oracle* is an algorithm that takes as input a distribution $\mathbf{p}$ and does one of the following:

- Returns a solution $\hat{\mathbf{x}} \in \mathcal{P}$ satisfying
  - $\mathbf{p}^\top\mathbf{A}\hat{\mathbf{x}} \geq \mathbf{p}^\top\mathbf{b} - \beta/3$, and
  - $\mathbf{A}_i\hat{\mathbf{x}} - b_i \in [-1, \phi]$ for *every* constraint i.
- Correctly reports that the inequality $\mathbf{p}^\top\mathbf{Ax} \geq \mathbf{p}^\top\mathbf{b}$ has no solution in $\mathcal{P}$.

The MWU algorithm for solving covering LPs involves T rounds. It maintains the (non-negative) weight of each constraint in $\mathbf{Ax} \geq \mathbf{b}$, which measures how much it has been satisfied by the solutions chosen so far. Let $\mathbf{w}^t$ denote the weight vector at the beginning of round t, and initialize the weights to $\mathbf{w}^1 \triangleq \mathbf{1}$. Then, for rounds $t = 1, \ldots, T$, define the probability vector $\mathbf{p}^t$ proportional to those weights $\mathbf{w}^t$, and use the oracle above to find a solution $\mathbf{x}^t$. If the oracle reports that the system $\mathbf{p}^\top\mathbf{Ax} \geq \mathbf{p}^\top\mathbf{b}$ is infeasible, the MWU algorithm also reports that the original system $\mathbf{Ax} \geq \mathbf{b}$ is infeasible, and terminates. Otherwise, define the cost vector incurred by $\mathbf{x}^t$ as $\mathbf{m}^t \triangleq \frac{1}{\phi}(\mathbf{Ax} - \mathbf{b})$, then update the weights so that $w_i^{t+1} \triangleq w_i^t(1 - \beta m_i^t/6)$ and proceed to the next round. Finally, the algorithm returns the average solution $\overline{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^t$.

The MWU theorem (e.g., Theorem 3.5 of [16]) shows that $T = O(\frac{\phi \log n}{\beta^2})$ is sufficient to correctly solve the problem, yielding $\mathbf{A}_i\hat{\mathbf{x}} - b_i \geq -\beta$ for every constraint, where n is the number of constraints. In particular, the algorithm requires T calls to the oracle.

Theorem 3.2.2 (MWU **Theorem [16]**). *For every* $0 < \beta < 1, \phi \geq 1$ *the* MWU *algorithm either solves the* Feasibility-Covering-LP *problem up to an additive error of* β *(i.e., solves* $\mathbf{A}_i\mathbf{x} - b_i \geq -\beta$ *for every* i*) or correctly reports that the LP is infeasible, making only* $O(\frac{\phi \log n}{\beta^2})$ *calls to a* $(1, \phi)$*-bounded* $\beta/3$*-approximate oracle of the LP.*

3.2.2. Semi-streaming MWU-based algorithm for factional Set Cover

Setting up our MWU **algorithm.** As described in the overview, we wish to solve, as a subroutine, the decision variant of SetCover-LP known as Feasibility-SC-LP given in Figure 3.2.2a, where the parameter ℓ serves as the guess for the optimal objective value.

Feasibility-SC-LP ▷ Input: $\mathcal{U}, \mathcal{F}, \ell$

$$\sum_{S \in \mathcal{F}} x_S \leq \ell$$
$$\sum_{S: e \in S} x_S \geq 1 \qquad \forall e \in \mathcal{U}$$
$$x_S \geq 0 \qquad \forall S \in \mathcal{F}$$

(a) LP relaxation of Feasibility Set Cover.

Feasibility-Covering-LP ▷ Input: $\mathbf{A}, \mathbf{b}, \mathbf{c}, \ell$

$\mathbf{c}^\top \mathbf{x} \leq \ell$	(objective value)
$\mathbf{A}\mathbf{x} \geq \mathbf{b}$	(covering)
$\mathbf{x} \geq \mathbf{0}$	(non-negativity)

(b) LP relaxation of Feasibility Covering.

Figure 3.2.2: LP relaxations of the feasibility variant of set cover and general covering problems.

To follow the conventional notation for solving LPs in the MWU framework, consider the more standard form of covering LPs denoted as Feasibility-Covering-LP given in Figure 3.2.2b. For our purpose, $\mathbf{A}_{n \times m}$ is the element-set incidence matrix indexed by $\mathcal{U} \times \mathcal{F}$; that is, $A_{e,S} = 1$ if $e \in S$, and $A_{e,S} = 0$ otherwise. The vectors $\mathbf{b}$ and $\mathbf{c}$ are both all-ones vectors indexed by $\mathcal{U}$ and $\mathcal{F}$, respectively. We emphasize that, unconventionally for our system $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, there are n constraints (i.e. elements) and m variables (i.e. sets).

Employing the MWU approach for solving covering LPs, we define the polytope

$$\mathcal{P}_\ell \triangleq \{\mathbf{x} \in \mathbb{R}^m : \mathbf{c}^\top \mathbf{x} \leq \ell \text{ and } \mathbf{x} \geq \mathbf{0}\}.$$

Observe that by applying the MWU algorithm to this polytope $\mathcal{P}$ and constraints $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, we obtain a solution $\overline{\mathbf{x}} \in \mathcal{P}_\ell$ such that $\mathbf{A}_e \left(\frac{\overline{\mathbf{x}}}{1-\beta}\right) \geq \frac{b_e - \beta}{1-\beta} = 1 = b_e$, where $\mathbf{A}_e$ denotes the row of $\mathbf{A}$ corresponding to e. This yields a $(1 + O(\varepsilon))$-approximate solution for $\beta = O(\varepsilon)$.

Unfortunately, we cannot implement the MWU algorithm on the full input under our

streaming context. Therefore, the main challenge is to implement the following two subtasks of the MWU algorithm in the streaming settings. First, we need to design an oracle that solves the average constraint in the streaming setting. Moreover, we need to be able to efficiently update the weights for the subsequent rounds.

Covering the common elements. Before we proceed to applying the MWU framework, we add a simple first step to our implementation of FeasibilityTest (Algorithm 5) that will greatly reduce the amount of sapce required in implementing the MWU algorithm. This can be interpreted as the fractional version of Set Sampling described in [61]. In our subroutine, we partition the elements into the common elements that occur more frequently, which will be covered if we simply choose a uniform vector solution, and the rare elements that occur less frequently, for which we perform the MWU algorithm to compute a good solution. In one pass we can find all frequently occurring elements by counting the number of sets containing each element. The amount of required space to perform this task is $O(n \log m)$.

we call an element that appears in at least $\frac{m}{\alpha\ell}$ sets *common*, and we call it *rare* otherwise, where $\alpha = \Theta(\varepsilon)$. Since we are aiming for a $(1+\varepsilon)$-approximation, we can define $\mathbf{x}^{\mathsf{cmn}}$ as a vector whose all entries are $\frac{\alpha\ell}{m}$. The total cost of $\mathbf{x}^{\mathsf{cmn}}$ is $\alpha\ell$ and all common elements are covered by $\mathbf{x}^{\mathsf{cmn}}$. Thus, throughout the algorithm we may restrict our attention to the rare elements.

Our goal now is to construct an efficient MWU-based algorithm, which finds a solution $\mathbf{x}^{\mathsf{rare}}$ covering the rare elements, with objective value at most $\frac{\ell}{1-\beta} \leq (1+\varepsilon-\alpha)\ell$. We note that our implementation does not explicitly maintain the weight vector $\mathbf{w}^t$ described in Section 3.2.1, but instead updates (and normalizes) its probability vector $\mathbf{p}^t$ in every round.

3.2.3. First Attempt: Simple Oracle and Large Width

A greedy solution for the oracle. We implement the oracle for MWU algorithm such that $\phi = \ell$, and thus requiring $\Theta(\ell \log n/\beta^2)$ iterations (Theorem 3.2.2). In each iteration, we need an oracle that finds some solution $\mathbf{x} \in \mathcal{P}_\ell$ satisfying $\mathbf{p}^\top \mathbf{A}\mathbf{x} \geq \mathbf{p}^\top \mathbf{b} - \beta/3$, or decides that no solution in $\mathcal{P}_\ell$ satisfies $\mathbf{p}^\top \mathbf{A}\mathbf{x} \geq \mathbf{p}^\top \mathbf{b}$.

Algorithm 5 A generic implementation of FEASIBILITYTEST. Its performance depends on the implementations of ORACLE, UPDATEPROB. We will investigate different implementations of ORACLE.

procedure FEASIBILITYTEST(ℓ, ε)
  $\alpha, \beta \leftarrow \frac{\varepsilon}{3}$, $\mathbf{p}^{\mathsf{curr}} \leftarrow \mathbf{1}_{m\times 1}$ ▷ the initial prob. vector for the MWU algorithm on $\mathcal{U}$
  ▷ compute a cover of *common* elements in **one** pass
  $\mathbf{x}^{\mathsf{cmn}} \leftarrow \frac{\alpha\ell}{m} \cdot \mathbf{1}_{m\times 1}$, $\mathbf{freq} \leftarrow \mathbf{0}_{n\times 1}$
  for all set S in the stream **do**
    for all element $e \in S$ **do**
      $\mathsf{freq}_e \leftarrow \mathsf{freq}_e + 1$
      if e appears in more than $\frac{m}{\alpha\ell}$ sets (i.e. $\mathsf{freq}_e > \frac{m}{\alpha\ell}$) **then** ▷ common element
        $p_e^{\mathsf{curr}} \leftarrow 0$
  $\mathbf{p}^{\mathsf{curr}} \leftarrow \frac{\mathbf{p}^{\mathsf{curr}}}{\|\mathbf{p}^{\mathsf{curr}}\|}$ ▷ $\mathbf{p}^{\mathsf{curr}}$ represents the current prob. vector
  $\mathbf{x}^{\mathsf{total}} \leftarrow \mathbf{0}_{m\times 1}$
  ▷ multiplicative weight update algorithm for covering *rare* elements
  for $i = 1$ to T **do**
    ▷ solve the corresp. oracle of MWU and decide if the solution is feasible
    try $\mathbf{x} \leftarrow$ ORACLE($\mathbf{p}^{\mathsf{curr}}, \ell, \mathcal{F}$)
    $\mathbf{x}^{\mathsf{total}} \leftarrow \mathbf{x}^{\mathsf{total}} + \mathbf{x}$ ▷ in **one** pass, update $\mathbf{p}$ according to $\mathbf{x}$
    $\mathbf{z} \leftarrow \mathbf{0}_{n\times 1}$
    for all set S in the stream **do**
      for all element $e \in S$ **do**
        $z_e \leftarrow z_e + x_S$
    if $(\mathbf{p}^{\mathsf{curr}})^\top \mathbf{z} < 1 - \beta/3$ **then** ▷ detect infeasible solutions returned by ORACLE
      report INFEASIBLE
    $\mathbf{p}^{\mathsf{curr}} \leftarrow$ UPDATEPROB($\mathbf{p}^{\mathsf{curr}}, \mathbf{z}$)
  $\mathbf{x}^{\mathsf{rare}} \leftarrow \frac{\mathbf{x}^{\mathsf{total}}}{(1-\beta)T}$ ▷ scaled up the solution to cover *rare* elements
  return $\mathbf{x}^{\mathsf{cmn}} + \mathbf{x}^{\mathsf{rare}}$

Observe that $\mathbf{p}^\top \mathbf{A}\mathbf{x}$ is maximized when we place value ℓ on x_{S^*} where S^* achieves the maximum value $p_S \triangleq \sum_{e\in S} p_e$. Further, for our application, $\mathbf{b} = \mathbf{1}$ so $\mathbf{p}^\top \mathbf{b} = 1$. Our implementation HEAVYSETORACLE of ORACLE given in Algorithm 6 below is a deterministic greedy algorithm that finds a solution based on this observation. As $\mathbf{A}_e\mathbf{x} \le \|\mathbf{x}\|_1 \le \ell$, HEAVYSETORACLE implements a $(1, \ell)$-bounded $(\beta/3)$-approximate oracle. Therefore, the implementation of FEASIBILITYTEST with HEAVYSETORACLE computes a solution of objective value at most $(\alpha + \frac{1}{1-\beta})\ell < (1 + \frac{\varepsilon}{3})\ell$ when $\ell \ge k$ as promised.

Finally, we track the space usage which concludes the complexities of the current version of our algorithm: it only stores vectors of length m or n, whose entries each requires a logarithmic number of bits, yielding the following theorem.

Algorithm 6 HEAVYSETORACLE computes p_S of every set given the set system in a stream or stored memory, then returns the solution $\mathbf{x}$ that optimally places value ℓ on the corresponding entry. It reports INFEASIBLE if there is no sufficiently good solution

procedure HEAVYSETORACLE($\mathbf{p}, \ell, \mathcal{F}$)
  compute p_S for all $S \in \mathcal{F}$ while reading the set system
  $S^* \leftarrow \mathbf{argmax}_{S \in \mathcal{F}} p_S$
  if $p_S < (1 - \beta/3)/\ell$ **then**
    report INFEASIBLE
  $\mathbf{x} \leftarrow \mathbf{0}_{n \times 1}, x_S \leftarrow \ell$
  return $\mathbf{x}$

Theorem 3.2.3. *There exists a streaming algorithm that w.h.p. returns a $(1+\varepsilon)$-approximate fractional solution of* SetCover-LP$(\mathcal{U}, \mathcal{F})$ *in $O(\frac{k \log n}{\varepsilon^2})$ passes and using $\widetilde{O}(m+n)$ memory for any positive $\varepsilon \leq 1/2$. The algorithm works in both set arrival and edge arrival streams.*

The presented algorithm suffers from large number of passes over the input. In particular, we are interested in solving the fractional Set Cover in constant number of passes using sublinear space. To this end, we first reduce the required number of rounds in MWU by a more complicated implementation of ORACLE.

3.3. Max Cover Problem and its Application to Width Reduction

In this section, we improve the described algorithm in the previous section and prove the following result.

Theorem 3.3.1. *There exists a streaming algorithm that w.h.p. returns a $(1+\varepsilon)$-approximate fractional solution of* SetCover-LP$(\mathcal{U}, \mathcal{F})$ *in p passes and uses $\widetilde{O}(mn^{O(1/p\varepsilon)} + n)$ memory for any $2 \leq p \leq \text{polylog}(n)$ and $0 < \varepsilon \leq 1/2$. The algorithm works in both set arrival and edge arrival streams.*

Recall that in implementing ORACLE, we must find a solution $\mathbf{x}$ of total size $\|\mathbf{x}\|_1 \leq \ell$ with a sufficiently large weight $\mathbf{p}^\top \mathbf{A}\mathbf{x}$. Our previous implementation chooses only one good entry x_S and places its entire *budget* ℓ on this entry. As the width of the solution is roughly the maximum amount an element is over-covered by $\mathbf{x}$, this implementation induces a width of ℓ. In this section, we design an oracle that returns a solution in which the budget is distributed more evenly among the entries of $\mathbf{x}$ to reduce the width. To this end, we design

an implementation of ORACLE of the MWU approach based on the Max ℓ-Cover problem (whose precise definition will be given shortly). The solution to our Max ℓ-Cover aids in reducing the width of our ORACLE solution to a constant, so the required number of rounds of the MWU algorithm decreases to $O(\frac{\log n}{\varepsilon^2})$, independent of ℓ. Note that, if the objective value of an optimal solution of Set Cover$(\mathcal{U}, \mathcal{F})$ is ℓ, then a solution of width $o(\ell)$ may not exist, as shown in Lemma 3.3.2. This observation implies that we need to work with a different set system. Besides having small width, an optimal solution of the Set Cover instance on the new set system should have the same objective value of the optimal solution of Set Cover$(\mathcal{U}, \mathcal{F})$.

Lemma 3.3.2. *There exists a set system in which, under the direct application of the MWU framework in computing a $(1+\varepsilon)$-approximate solution, induces width $\phi = \Omega(k)$, where k is the optimal objective value. Moreover, the exists a set system in which the approach from the previous section (which handles the frequent and rare elements differently) has width $\phi = \Theta(n) = \Theta(\sqrt{m/\varepsilon})$.*

Proof: For the first claim, we consider an arbitrary set system, then modify it by adding a common element e to all sets. Recall that the MWU framework returns an average of the solutions from all rounds. Thus there must exist a round where the oracle returns a solution $\mathbf{x}$ of size $\|\mathbf{x}\|_1 = \Theta(k)$. For the added element e, this solution has $\sum_{S:e\in S} x_S = \sum_{S\in\mathcal{F}} x_S = \Theta(k)$, inducing width $\phi = \Omega(k)$.

For the second claim, consider the following set system with $k = \sqrt{m/\varepsilon}$ and $n = 2k+1$. For $i = 1, \ldots, k$, let $S_i = \{e_i, e_{k+i}, e_{2k+1}\}$, whereas the remaining $m-k$ sets are arbitrary subsets of $\{e_1, \ldots, e_k\}$. Observe that e_{k+i} is contained only in S_i, so $x_{S_i} = 1$ in any valid set cover. Consequently the solution $\mathbf{x}$ where $x_{S_1} = \cdots = x_{S_k} = 1$ and $x_{S_{k+1}} = \cdots = x_{S_{2k+1}} = 0$ forms the unique (fractional) minimum set cover of size $k = \sqrt{m/\varepsilon}$. Next, recall that an element is considered rarely occurring if it appears in at most $\frac{m}{\alpha\ell} > \frac{m}{\varepsilon k}$ sets. As $e_{k+1}, \ldots, e_{2k}$ each only occurs once, and e_{2k+1} only appears in $k = \sqrt{m/\varepsilon} = \frac{m}{\varepsilon k}$ sets, these $k+1$ elements are deemed rare and thus handled by the MWU framework.

The solution computed by the MWU framework satisfies $\sum_{S:e\in S} x_S \geq 1-\beta$ for every e, and in particular, for each $e \in \{e_{k+1}, \ldots, e_{2k}\}$. Therefore, the average solution places a total

weight of at least $(1-\beta)\cdot\Theta(k)$ on $x_{S_1},\ldots,x_{S_k}$, so there must exist a round that places at least the same total weight on these sets. However, these k sets all contain e_{2k+1}, yielding $\sum_{S:e_{2k+1}\in S} x_S \geq (1-\beta)\cdot\Theta(k) = \Omega(k)$, implying a width of $\Omega(k) = \Omega(\sqrt{m/\varepsilon})$. □

Extended set system. First, we consider the *extended set system* $(\mathcal{U}, \breve{\mathcal{F}})$, where $\breve{\mathcal{F}}$ is the collection containing all subsets of sets in $\mathcal{F}$; that is,

$$\breve{\mathcal{F}} \triangleq \{R : R \subseteq S \text{ for some } S \in \mathcal{F}\}.$$

It is straightforward to see that the optimal objective value of $\mathsf{SetCover}$ over $(\mathcal{U}, \breve{\mathcal{F}})$ is equal to that of $(\mathcal{U}, \mathcal{F})$: we only add subsets of the original sets to create $\breve{\mathcal{F}}$, and we may replace any subset from $\breve{\mathcal{F}}$ in our solution with its original set in $\mathcal{F}$. Moreover, we may *prune* any collection of sets from $\mathcal{F}$ into a collection from $\breve{\mathcal{F}}$ of the same cardinality so that, this pruned collection not only covers the same elements, but also each of these elements is covered exactly once. This extended set system is defined for the sake of analysis only: we will never explicitly handle an exponential number of sets throughout our algorithm.

We define *ℓ-cover* as a collection of sets of total weight ℓ. Although the pruning of an ℓ-cover reduces the width, the total weight $\mathbf{p}^\top\mathbf{A}\mathbf{x}$ of the solution will decrease. Thus, we consider the weighted constraint of the form

$$\sum_{e\in\mathcal{U}} \left(p_e \cdot \min\{1, \sum_{S:e\in S} x_S\} \right) \geq 1;$$

that is, we can only gain the value p_e without any multiplicity larger than 1. The problem of maximizing the left hand side is known as the *weighted max coverage* problem: for a parameter ℓ, find an ℓ-cover such that the total value p_e's of the covered elements is maximized.

3.3.1. The Maximum Coverage Problem

In the design of our algorithm, we consider the *weighted* $\mathsf{Max}\,k\text{-}\mathsf{Cover}$ problem, which is closely related to $\mathsf{SetCover}$. Extending upon the brief description given earlier, we fully specify the LP relaxation of this problem. In the weighted $\mathsf{Max}\,k\text{-}\mathsf{Cover}(\mathcal{U}, \mathcal{F}, \ell, \mathbf{p})$, given a ground set of elements $\mathcal{U}$, a collection of sets $\mathcal{F}$ over the ground set, a budget parameter

MaxCover-LP ▷ Input: $\mathcal{U}, \mathcal{F}, \ell, \mathbf{p}$

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in \mathcal{U}} p_e z_e & \\ \text{subject to} \quad & \sum_{S: e \in S} x_S \geq z_e & \forall e \in \mathcal{U} \\ & \sum_{S \in \mathcal{F}} x_S = \ell & \\ & 0 \leq z_e \leq 1 & \forall e \in \mathcal{U} \\ & x_S \geq 0 & \forall S \in \mathcal{F} \end{aligned}$$

Figure 3.3.1: LP relaxation of weighted Max k-Cover.

ℓ, and a weight vector $\mathbf{p}$, the goal is to return ℓ sets in $\mathcal{F}$ whose weighted *coverage*, the total weight of all covered elements, is maximized. Moreover, since we are aiming for a fractional solution of Set Cover, we consider the LP relaxation of weighted Max k-Cover, MaxCover-LP (see Figure 3.3.1); in this LP relaxation, z_e denotes the fractional amount that an element is covered, and hence is capped at 1.

As an intermediate goal, we aim to compute an approximate solution of MaxCover-LP, given that the optimal solution covers all elements in the ground set, or to correctly detect that no solution has weighted coverage of more than $(1 - \varepsilon)$. In our application, the vector $\mathbf{p}$ is always a probability vector: $\mathbf{p} \geq \mathbf{0}$ and $\sum_{e \in \mathcal{U}} p_e = 1$. We make the following useful observation.

Observation 3.3.3. *Let k be the value of an optimal solution of* SetCover-LP$(\mathcal{U}, \mathcal{F})$ *and let $\mathbf{p}$ be an arbitrary probability vector over the ground set. Then there exists a fractional solution of* MaxCover-LP$(\mathcal{U}, \mathcal{F}, \ell, \mathbf{p})$ *whose weighted coverage is one if $\ell \geq k$.*

δ-integral near optimal solution of MaxCover-LP. Our plan is to solve MaxCover-LP over a randomly projected set system, and argue that with high probability this will result in a valid Oracle. Such an argument requires an application of the union bound over the set of solutions, which is generally of unbounded size. To this end, we consider a more restrictive domain of *δ-integral* solutions: this domain has bounded size, but is still guaranteed to contain a sufficiently good solution.

Definition 3.3.4 (δ-integral solution). A fractional solution $\mathbf{x}_{n \times 1}$ of an LP is δ-integral

Algorithm 7 MaxCoverOracle returns a fractional ℓ-cover with weighted coverage at least $1-\beta/3$ w.h.p. if $\ell \geq k$. It provides no guarantee on its behavior if $\ell < k$.

1: **procedure** MaxCoverOracle($(\mathcal{U}, \mathcal{F}), \ell$)
2: $\quad$ $\mathbf{x} \leftarrow$ solution returned by MWU using HeavySetOracle on MaxCover-LP
3: $\quad$ **return** $\mathbf{x}$

if $\frac{1}{\delta} \cdot \mathbf{x}$ is an integral vector. That is, for each $i \in [n]$, $x_i = v_i\delta$ where each v_i is an integer.

Next we claim that MaxCoverOracle given in Algorithm 7 below, which is the MWU algorithm with HeavySetOracle for solving MaxCover-LP, results in a δ-integral solution.

Lemma 3.3.5. *Consider a* MaxCover-LP *with the optimal objective value* OPT *(where the weights of elements form a probability vector). There exists a* $\Theta(\frac{\varepsilon_{\mathrm{MC}}^2}{\log n})$*-integral solution of* MaxCover-LP *whose objective value is at least* $(1-\varepsilon_{\mathrm{MC}})\mathrm{OPT}$*. In particular, if an optimal solution covers all elements* $\mathcal{U}$ *(*$\ell \geq k$*),* MaxCoverOracle *returns a solution whose weighted coverage is at least* $1-\varepsilon_{\mathrm{MC}}$ *in polynomial time.*

Proof: Let $(\mathbf{x}^*, \mathbf{z}^*)$ denote the optimal solution of value OPT to MaxCover-LP, which implies that $\|\mathbf{x}^*\|_1 \leq \ell$ and $\mathbf{A}\mathbf{x}^* \geq \mathbf{z}^*$. Consider the following covering LP: minimize $\|\mathbf{x}\|_1$ subject to $\mathbf{A}\mathbf{x} \geq \mathbf{z}^*$ and $\mathbf{x} \geq \mathbf{0}$. Clearly there exists an optimal solution of objective value ℓ, namely $\mathbf{x}^*$. This covering LP may be solved via the MWU framework. In particular, we may use the oracle that picks one set S with maximum weight (as maintained in the MWU framework) and places its entire budget on x_S. For an accurate guess $\ell' = \Theta(\ell)$ of the optimal value, this algorithm returns an average of $T = \Theta(\frac{\ell' \log n}{\varepsilon_{\mathrm{MC}}^2}) = \Theta(\frac{\ell \log n}{\varepsilon_{\mathrm{MC}}^2})$ oracle solutions. Observe that the outputted solution $\mathbf{x}$ is of the form $x_S = \frac{v_S \ell'}{T} = v_S\delta$ where v_S is the number of rounds in which S is chosen by the oracle, and $\delta = \frac{\ell'}{T} = \frac{\ell' \varepsilon_{\mathrm{MC}}^2}{\ell \log n} = \Theta(\frac{\varepsilon_{\mathrm{MC}}^2}{\log n})$. In other words, $\mathbf{x}$ is $(\frac{\varepsilon_{\mathrm{MC}}^2}{\log n})$-integral. By Theorem 3.2.2, $\mathbf{x}$ satisfies $\mathbf{A}\mathbf{x} \geq (1-\varepsilon_{\mathrm{MC}})\mathbf{z}^*$. Then in MaxCover-LP, the solution $(\mathbf{x}, (1-\varepsilon_{\mathrm{MC}})\mathbf{z}^*)$ yields coverage at least $\mathbf{p}^\top((1-\varepsilon_{\mathrm{MC}})\mathbf{z}^*) = (1-\varepsilon_{\mathrm{MC}})\mathbf{p}^\top\mathbf{z}^* = (1-\varepsilon_{\mathrm{MC}})\mathrm{OPT}$.□

Pruning a fractional ℓ-cover. In our analysis, we aim to solve the Set Cover problem under the extended set system. We claim that any solution $\mathbf{x}$ with coverage $\mathbf{z}$ in the actual set system may be turned into a pruned solution $\breve{\mathbf{x}}$ in the extended set system that provides the same coverage $\mathbf{z}$, but satisfies the strict equality $\sum_{\breve{S} \in \breve{\mathcal{F}}: e \in \breve{S}} \breve{x}_{\breve{S}} = z_e$. Since $z_e \leq 1$, the pruned solution satisfies the condition for an oracle with width *one*.

Algorithm 8 The PRUNE subroutine lifts a solution in $\mathcal{F}$ to a solution in $\breve{\mathcal{F}}$ with the same MaxCover-LP objective value and width 1. The subroutine returns $\mathbf{z}$, the amount by which members of $\breve{\mathcal{F}}$ cover each element. The actual pruned solution $\breve{\mathbf{x}}$ may be computed but has no further use in our algorithm and thus not returned.

procedure PRUNE($\mathbf{x}$)
  $\breve{\mathbf{x}} \leftarrow \mathbf{0}_{|\breve{\mathcal{F}}|\times 1}$, $\mathbf{z} \leftarrow \mathbf{0}_{n\times 1}$ ▷ maintain the pruned solution and its coverage amount
  for all $S \in \mathcal{F}$ **do**
    $\breve{S} \leftarrow S$
    while $x_S > 0$ **do**
      $r \leftarrow \min\{x_S, \min_{e\in\breve{S}}(1 - z_e)\}$ ▷ weight to be moved from x_S to $\breve{x}_{\breve{S}}$
      $x_S \leftarrow x_S - r$, $x_{\breve{S}} \leftarrow x_{\breve{S}} + r$ ▷ move weight to the pruned solution
      for all $e \in \breve{S}$ **do**
        $z_e \leftarrow z_e + r$ ▷ update coverage accordingly
      $\breve{S} \leftarrow \breve{S} \setminus \{e \in \breve{S} : z_e = 1\}$ ▷ remove e with $z_e = 1$ from $\breve{S}$
  return $\mathbf{z}$

Lemma 3.3.6. *A fractional ℓ-cover $\mathbf{x}$ of $(\mathcal{U}, \mathcal{F})$ can be converted, in polynomial time, to a fractional ℓ-cover $\breve{\mathbf{x}}$ of $(\mathcal{U}, \breve{\mathcal{F}})$ such that for each element e, its coverage $z_e = \sum_{\breve{S}\in\breve{\mathcal{F}}:e\in\breve{S}} \breve{x}_{\breve{S}} = \min(\sum_{S:e\in S} x_S, 1)$.*

Proof: Consider the algorithm PRUNE in Algorithm 8. As we pick a valid amount $r \le x_S$ to move from x_S to $\breve{x}_{\breve{S}}$ at each step, $\breve{\mathbf{x}}$ must be an ℓ-cover (in the extended set system) when PRUNE finishes. Observe that if $\sum_{S:e\in S} x_S < 1$ then e will never be removed from any $\breve{S}$, so z_e is increased by x_S for every S, and thus $z_e = \sum_{S:e\in S} x_S$. Otherwise, the condition $r \le 1 - z_e$ ensures that z_e stops increasing precisely when it reaches 1. Each S takes up to $n + 1$ rounds in the while loop as one element $e \in S$ is removed at the end of each round. There are at most m sets, so the algorithm must terminate (in polynomial time).

We note that in Section 3.3.4, we need to adjust PRUNE to instead achieves the condition $z_e = \min(\mathbf{A}_e\mathbf{x}, 1)$ where entries of $\mathbf{A}$ are arbitrary non-negative values. We simply make the following modifications: choose $r \leftarrow \min(x_S, \min_{e\in\breve{S}} \frac{1-z_e}{A_{e,S}})$ and update $z_e \leftarrow z_e + r \cdot A_{e,S}$, and the same proof follows. □

Remark that to update the weights in the MWU framework, it is sufficient to have the coverage $\sum_{\breve{S}\in\breve{\mathcal{F}}:e\in\breve{S}} \breve{x}_{\breve{S}}$, which are the z_e's returned by PRUNE; the actual solution $\breve{\mathbf{x}}$ is not necessary. Observe further that our MWU algorithm can still use $\mathbf{x}$ instead of $\breve{\mathbf{x}}$ as its solution because $\mathbf{x}$ has no worse coverage than $\breve{\mathbf{x}}$ in every iteration, and so does the final,

average solution. Lastly, notice that the coverage $\mathbf{z}$ returned by PRUNE has the simple formula $z_e = \min(\sum_{S:e\in S} x_S, 1)$. That is, we introduce PRUNE to show an existence of $\breve{\mathbf{x}}$, but will never run PRUNE in our algorithm.

3.3.2. Sampling-Based Oracle for Fractional Max Coverage

In the previous section, we simply needed to compute the values p_S's in order to construct a solution for the ORACLE. Here as we aim to bound the width of ORACLE, our new task is to find a fractional ℓ-cover $\mathbf{x}$ whose weighted coverage is at least $1-\beta/3$. The *element sampling* technique, which is also known from prior work in streaming Set Cover and Max k-Cover, is to sample a few elements and solve the problem over the sampled elements only. Then, by applying the union bound over all possible candidate solutions, it is shown that w.h.p. a nearly optimal cover of the sampled elements also covers a large fraction of the whole ground set. This argument applies to the aforementioned problems precisely because there are standard ways of bounding the number of all integral candidate solutions (e.g. ℓ-covers).

However, in the fractional setting, there are infinitely many solutions. Consequently, we employ the notion of δ-integral solutions where the number of such solutions is bounded. In Lemma 3.3.6, we showed that there always exists a δ-integral solution to MaxCover-LP whose coverage is at least a $(1-\varepsilon_{\text{MC}})$-fraction of an optimal solution. Moreover, the number of all possible solutions is bounded by the number of ways to divide the budget ℓ into ℓ/δ equal parts of value δ and distribute them (possibly with repetition) among m entries:

Observation 3.3.7. *The number of feasible δ-integral solutions to* MaxCover-LP$(\mathcal{U}, \mathcal{F}, \ell, \mathbf{p})$ *is $O(m^{\ell/\delta})$ for any multiple ℓ of δ.*

Next, we design our algorithm using the element sampling technique: we show that a $(1-\beta/3)$-approximate solution of MaxCover-LP can be computed using the projection of all sets in $\mathcal{F}$ over a set of elements of size $\Theta(\frac{\ell \log n \log mn}{\beta^4})$ picked according to $\mathbf{p}$. For every fractional solution $(\mathbf{x}, \mathbf{z})$ and subset of elements $\mathcal{V} \subseteq \mathcal{U}$, let $\mathcal{C}_{\mathcal{V}}(\mathbf{x}) \triangleq \sum_{e\in\mathcal{V}} p_e z_e$ denote the coverage of elements in $\mathcal{V}$ where $z_e = \min(1, \sum_{S:e\in S} x_S)$. We may omit the subscript $\mathcal{V}$ in $\mathcal{C}_{\mathcal{V}}$ if $\mathcal{V} = \mathcal{U}$.

The following lemma, which is essentially an extension of the Element Sampling lemma of [61] for our application, MaxCover-LP, shows that a $(1-\varepsilon_{\text{MC}})$-approximate ℓ-cover over

a set of sampled elements of size $\Theta(\ell \log n \log mn/\gamma^4)$ w.h.p. has a weighted coverage of at least $(1-2\gamma)(1-\varepsilon_{\text{MC}})$ if there exists a fractional ℓ-cover whose coverage is 1. Thus, choosing $\varepsilon_{\text{MC}} = \gamma = \beta/9$ yields the desired guarantee for MaxCoverOracle, leading to the performance given in Theorem 3.3.9.

Lemma 3.3.8. *Let ε_{MC} and γ be parameters. Consider the* MaxCover-LP$(\mathcal{U}, \mathcal{F}, \ell, \mathbf{p})$ *with optimal solution of value* OPT*, and let $\mathcal{L}$ be a multi-set of $s = \Theta(\ell \log n \log(mn)/\gamma^4)$ elements sampled independently at random according to the probability vector $\mathbf{p}$. Let $\mathbf{x}^{\text{SOL}}$ be a $(1-\varepsilon_{\text{MC}})$-approximate $\Theta(\frac{\gamma^2}{\log n})$-integral ℓ-cover over the sampled elements. Then with high probability, $\mathcal{C}(\mathbf{x}^{\text{SOL}}) \geq (1-2\gamma)(1-\varepsilon_{\text{MC}})\text{OPT}$.*

Proof: Consider the MaxCover-LP$(\mathcal{U}, \mathcal{F}, \ell, \mathbf{p})$ with optimal solution $(\mathbf{x}^{\text{OPT}}, \mathbf{z}^{\text{OPT}})$ of value OPT, and let $\mathbf{x}^{\text{SOL}}$ be a $(1-\varepsilon_{\text{MC}})$-approximate $\Theta(\frac{\gamma^2}{\log n})$-integral ℓ-cover over the sampled elements and $\mathbf{z}^{\text{SOL}}$ be its corresponding coverage vector. Denote the sampled elements with $\mathcal{L} = \{\hat{e}_1, \cdots, \hat{e}_s\}$. Observe that by defining each X_i as a random variable that takes the value $z^{\text{OPT}}_{\hat{e}_i}$ with probability $p_{\hat{e}_i}$ and 0 otherwise, the expected value of $\mathsf{X} = \sum_{i=1}^{s} \mathsf{X}_i$ is

$$\mathbf{E}[\mathsf{X}] = \sum_{i=1}^{s} \mathbf{E}[\mathsf{X}_i] = s \sum_{e \in \mathcal{U}} p_e \cdot z_e^{\text{OPT}} = s \cdot \mathcal{C}(\mathbf{x}^{\text{OPT}}) = s \cdot \text{OPT}.$$

Let $\tau = s(1-\gamma)\text{OPT}$. Since $\mathsf{X}_i \in [0,1]$, by applying Chernoff bound on X, we obtain

$$\begin{aligned}\Pr \mathcal{C}_{\mathcal{L}}(\mathbf{x}^{\text{OPT}}) \leq \tau = \Pr \mathsf{X} &\leq (1-\gamma)\mathbf{E}[\mathsf{X}] \\ &\leq e^{-\frac{\gamma^2 \mathbf{E}[\mathsf{X}]}{3}} \leq e^{-\frac{\Omega(\ell \log(mn) \log n/\gamma^2)}{3}} = (mn)^{-\Omega(\ell \log n/\gamma^2)}.\end{aligned}$$

Therefore, since $\mathbf{x}^{\text{SOL}}$ is a $(1-\varepsilon_{\text{MC}})$-approximate solution of MaxCover-LP$(\mathcal{L}, \mathcal{F}, \ell, \mathbf{p})$, with probability $1-(mn)^{-\Omega(\ell \log n/\gamma^2)}$, we have $\mathcal{C}_{\mathcal{L}}(\mathbf{x}^{\text{SOL}}) \geq (1-\varepsilon_{\text{MC}})\tau$.

Next, by a similar approach, we show that for any fractional solution $\mathbf{x}$, if $\mathcal{C}_{\mathcal{L}}(\mathbf{x}) \geq \mathcal{C}_{\mathcal{L}}(\mathbf{x}^{\text{OPT}})$, then with probability $1-(mn)^{-\Omega(\ell \log n/\gamma^2)}$, $\mathcal{C}(\mathbf{x}) \geq \left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT}$. Consider a fractional ℓ-cover $(\mathbf{x}, \mathbf{z})$ whose coverage is less than $\left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT}$. Let Y_i denote a random variable that takes value $z_{\hat{e}_i}$ with probability $p_{\hat{e}_i}$, and define $\mathsf{Y} = \sum_{i=1}^{s} \mathsf{Y}_i$. Then, $\mathbf{E}[\mathsf{Y}_i] = \mathcal{C}(\mathbf{x}) < \left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT}$. For ease of analysis, let each $\overline{\mathsf{Y}}_i \in [0,1]$ be an auxiliary random variable that stochastically dominates Y_i with expectation $\mathbf{E}[\overline{\mathsf{Y}}_i] =$

$\left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT}$, and $\overline{\mathsf{Y}} = \sum_{i=1}^{s} \overline{\mathsf{Y}}_i$ which stochastically dominates $\overline{\mathsf{Y}}$ with expectation $\mathbf{E}[\overline{\mathsf{Y}}] = s \cdot \left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT} = \frac{(1-\varepsilon_{\text{MC}})\tau}{1+\gamma}$. We then have

$$\begin{aligned}\Pr \mathcal{C}_{\mathcal{L}}(\mathbf{x}) > (1-\varepsilon_{\text{MC}})\tau = \Pr \mathsf{Y} > (1-\varepsilon_{\text{MC}})\tau = \Pr \mathsf{Y} > (1+\gamma)\mathbf{E}[\overline{\mathsf{Y}}] \\ \leq \Pr \overline{\mathsf{Y}} > (1+\gamma)\mathbf{E}[\overline{\mathsf{Y}}] \leq e^{-\frac{\gamma^2 \mathbf{E}[\overline{\mathsf{Y}}]}{3}} \leq (mn)^{-\Omega(\ell \log n/\gamma^2)},\end{aligned}$$

using the fact that $\left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}}) = \Theta(1)$ for our interested range of parameters. Thus,

$$\Pr \mathcal{C}(\mathbf{x}) \leq \left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT} \text{ and } \mathcal{C}_{\mathcal{L}}(\mathbf{x}) > (1-\varepsilon_{\text{MC}})\tau \leq (mn)^{-\Omega(\ell \log n/\gamma^2)}.$$

In other words, except with probability $(mn)^{-\Omega(\ell \log n/\gamma^2)}$, a chosen solution $\mathbf{x}$ that offers at least as good empirical coverage over $\mathcal{L}$ as $\mathbf{x}^{\text{OPT}}$ (namely $\mathbf{x}^{\text{SOL}}$) does have actual coverage of at least $\left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT}$.

Since the total number of $\Theta(\frac{\gamma^2}{\log n})$-integral ℓ-covers is $O(m^{\ell \log n/\gamma^2})$ (Observation 3.3.7), applying union bound, with probability at least $1 - O(m^{\ell \log n/\gamma^2}) \cdot (mn)^{-\Omega(\ell \log n/\gamma^2)} = 1 - \frac{1}{\text{poly}(mn)}$, a $(1-\varepsilon_{\text{MC}})$-approximate $\Theta(\frac{\gamma^2}{\log n})$-integral solution of MaxCover-LP$(\mathcal{L}, \mathcal{F}, \ell, \mathbf{p})$ has weighted coverage of at least $\left(\frac{1-\gamma}{1+\gamma}\right)(1-\varepsilon_{\text{MC}})\text{OPT} > (1-2\gamma)(1-\varepsilon_{\text{MC}})\text{OPT}$ over $\mathcal{U}$. □

Theorem 3.3.9. *There exists a streaming algorithm that w.h.p. returns a $(1+\varepsilon)$-approximate fractional solution of* SetCover-LP$(\mathcal{U}, \mathcal{F})$ *in $O(\log n/\varepsilon^2)$ passes and uses $\widetilde{O}(m/\varepsilon^6 + n)$ memory for any positive $\varepsilon \leq 1/2$. The algorithm works in both set arrival and edge arrival streams.*

Proof: The algorithm clearly requires $\Theta(T)$ passes to simulate the MWU algorithm. The required amount of memory, besides $\widetilde{O}(n)$ for counting elements, is dominated by the projected set system. In each pass over the stream, we sample $\Theta(\ell \log mn \log n/\varepsilon^4)$ elements, and since they are rarely occurring, each is contained in at most $\Theta(\frac{m}{\varepsilon \ell})$ sets. Finally, we run $\log_{1+\Theta(\varepsilon)} n = O(\log n/\varepsilon)$ instances of the MWU algorithm in parallel to compute a $(1+\varepsilon)$-approximate solution. In total, our space complexity is $\Theta(\ell \log mn \log n/\varepsilon^4) \cdot \Theta(\frac{m}{\varepsilon \ell}) \cdot O(\log n/\varepsilon) = \widetilde{O}(m/\varepsilon^6)$. □

3.3.3. Final Step: Running Several MWU Rounds Together

We complete our result by further reducing the number of passes at the expense of increasing the required amount of memory, yielding our full algorithm FastFeasibilityTest in Algorithm 9. More precisely, aiming for a p-pass algorithm, we show how to execute $R \triangleq \frac{T}{\Theta(p)} = \Theta(\frac{\log n}{p\beta^2})$ rounds of the MWU algorithm in a single pass. We show that this task may be accomplished with a multiplicative factor of $f \cdot \Theta(\log mn)$ increase in memory usage, where $f \triangleq n^{\Theta(1/(p\beta))}$.

Advance sampling. Consider a sequence of R consecutive rounds $i = 1, \ldots, R$. In order to implement the MWU algorithm for these rounds, we need (multi-)sets of sampled elements $\mathcal{L}_1, \ldots, \mathcal{L}_R$ according to probabilities $\mathbf{p}^1, \ldots, \mathbf{p}^R$, respectively (where $\mathbf{p}^i$ is the probability corresponding to round i). Since the probabilities of subsequent rounds are not known in advance, we circumvent this problem by choosing these sets $\mathcal{L}_i$'s with probabilities according to $\mathbf{p}^1$, but the number of samples in each set will be $|\mathcal{L}_i| = s \cdot f \cdot \Theta(\log mn)$ instead of s. Then, once $\mathbf{p}^i$ is revealed, we sub-sample elements from $\mathcal{L}_i$ to obtain $\mathcal{L}'_i$ as follow: for a (copy of) sampled element $\hat{e} \in \mathcal{L}_i$, add $\hat{e}$ to $\mathcal{L}'_i$ with probability $\frac{p^i_{\hat{e}}}{p^1_{\hat{e}} f}$; otherwise, simply discard it. Note that it is still left to be shown that the probability above is indeed at most 1.

Since each e was originally sampled with probability p^1_e, then in $\mathcal{L}'_i$, the probability that a sampled element $\hat{e} = e$ is exactly p^i_e/f. By having $f \cdot \Theta(\log mn)$ times the originally required number of samples s in the first place, in expectation we still have $\mathbf{E}[|\mathcal{L}'_i|] = |\mathcal{L}_i| \sum_{e \in \mathcal{U}} \frac{p^i_e}{f} = (s \cdot f \cdot \Theta(\log mn))\frac{1}{f} = s \cdot \Theta(\log mn)$. Due to the $\Theta(\log mn)$ factor, by the Chernoff bound, we conclude that with w.h.p. $|\mathcal{L}'_i| \geq s$. Thus, we have a sufficient number of elements sampled with probability according to $\mathbf{p}^i$ to apply Lemma 3.3.8, as needed.

Change in probabilities. As noted above, we must show that the probability that we sub-sample each element is at most 1; that is, $p^i_e/p^1_e \leq f = n^{\Theta(1/(p\beta))}$ for every element e and every round $i = 1, \ldots, R$. We bound the multiplicative difference between the probabilities of two consecutive rounds as follows.

Lemma 3.3.10. *Let* $\mathbf{p}$ *and* $\mathbf{p}'$ *be the probability of elements before and after an update. Then for every element* e, $p'_e \leq (1 + O(\beta))p_e$.

Proof: Recall the weight update formula $w_e^{t+1} = w_e^t(1 - \frac{\beta(\breve{\mathbf{A}}_e\breve{\mathbf{x}} - b_e)}{6\phi})$ for the MWU framework, where $\breve{\mathbf{A}}_{n\times|\breve{\mathcal{F}}|}$ represents the membership matrix corresponding to the extended set system $(\mathcal{U}, \breve{\mathcal{F}})$. In our case, the desired coverage amount is $b_e = 1$. By construction, we have $\breve{\mathbf{A}}_e\breve{\mathbf{x}} = z_e \leq 1$; therefore, our width is $\phi = 1$, and $-1 \leq \breve{\mathbf{A}}_e\breve{\mathbf{x}} - b_e \leq 0$. That is, the weight of each element cannot decrease, but may increase by at most a multiplicative factor of $1+\beta/6$, before normalization. Thus even after normalization no weight may increase by more than a factor of $1 + \beta/6 = 1 + O(\beta)$. □

Therefore, after $R = \Theta(\frac{\log n}{p\beta^2})$ rounds, the probability of any element may increase by at most a factor of $(1 + O(\beta))^{\Theta(\frac{\log n}{p\beta^2})} \leq e^{\Theta(\frac{\log n}{p\beta})} = n^{\Theta(1/(p\beta))} = f$, as desired. This concludes the proof of Theorem 7.8.9.

Implementation details. We make a few remarks about the implementation given in Algorithm 9. First, even though we perform all sampling in advance, the decisions of MaxCoverOracle do not depend on any $\mathcal{L}_i$ of later rounds, and UpdateProb is entirely deterministic: there is no dependency issue between rounds. Next, we only need to perform UpdateProb on the sampled elements $\mathcal{L} = \mathcal{L}_1 \cup \cdots \cup \mathcal{L}_R$ during the current R rounds. We therefore denote the probabilities with a different vector $\mathbf{q}^i$ over the sampled elements $\mathcal{L}$ only. Probabilities of elements outside $\mathcal{L}$ are not required by MaxCoverOracle during these rounds, but we simply need to spend one more pass after executing R rounds of MWU to aggregate the new probability vector $\mathbf{p}$ over all (rare) elements. Similarly, since MaxCoverOracle does not have the ability to verify, during the MWU algorithm, that each solution $\mathbf{x}^i$ returned by the oracle indeed provides a sufficient coverage, we check all of them during this additional pass. Lastly, we again remark that this algorithm operates on the extended set system: the solution $\mathbf{x}$ returned by MaxCoverOracle has at least the same coverage as $\breve{\mathbf{x}}$. While $\breve{\mathbf{x}}$ is not explicitly computed, its coverage vector $\mathbf{z}$ can be computed exactly.

3.3.4. Extension to general covering LPs

We remark that our MWU-based algorithm can be extended to solve a more general class of covering LPs. Consider the problem of finding a vector $\mathbf{x}$ that minimizes $\mathbf{c}^\top\mathbf{x}$ subject to

Algorithm 9 An efficient implementation of FEASIBILITYTEST which performs in p passes and consumes $\widetilde{O}(mn^{O(\frac{1}{p\varepsilon})} + n)$ space.

procedure FASTFEASIBILITYTEST(ℓ, ε)
  $\alpha, \beta \leftarrow \frac{\varepsilon}{3}$, $\mathbf{p}^{\mathsf{curr}} \leftarrow \mathbf{1}_{m\times 1}$ ▷ the initial prob. vector for the MWU algorithm on $\mathcal{U}$
  ▷ see Algorithm 5 for implementation of the below line
  compute a cover of *common* elements in **one** pass
  $\mathbf{x}^{\mathsf{total}} \leftarrow \mathbf{0}_{m\times 1}$
  ▷ MWU algorithm for covering *rare* elements
  for $i = 1$ to p **do**
    $R \leftarrow \Theta(\frac{\log n}{p\beta^2})$ ▷ number of MWU iterations performed together
    ▷ in **one** pass, projects all sets in $\mathcal{F}$ over the collections of samples $\mathcal{L}_1, \cdots \mathcal{L}_R$
    sample $\mathcal{L}_1, \ldots, \mathcal{L}_R$ according to $\mathbf{p}^{\mathsf{curr}}$ each of size $\ell n^{\Theta(1/(p\beta))}\operatorname{poly}(\log mn)$
    $\mathcal{L} \leftarrow \mathcal{L}_1 \cup \cdots \cup \mathcal{L}_R$, $\mathcal{F}_{\mathcal{L}} \leftarrow \emptyset$ ▷ $\mathcal{L}$ is a set whereas $\mathcal{L}_1, \ldots, \mathcal{L}_R$ are multi-sets
    for all set S in the stream **do**
      $\mathcal{F}_{\mathcal{L}} \leftarrow \mathcal{F}_{\mathcal{L}} \cup \{S \cap \mathcal{L}\}$
    ▷ each pass simulates R rounds of MWU
    for all $e \in \mathcal{L}$ **do**
      $q_e^1 \leftarrow p_e^{\mathsf{curr}}$ ▷ project $\mathbf{p}_{n\times 1}^{\mathsf{curr}}$ to $\mathbf{q}_{|\mathcal{L}|\times 1}^1$ over sampled elements
    $\mathbf{q}^1 \leftarrow \frac{\mathbf{q^1}}{\|\mathbf{q^1}\|}$
    for all round $i = 1, \ldots, R$ **do**
      $\mathcal{L}_i' \leftarrow$ **sample** each elt $e \in \mathcal{L}_i$ with probab. $\frac{q_e^i}{q_e^1 n^{\Theta(1/(p\beta))}}$ ▷ rejection sampling
      $\mathbf{x}^i \leftarrow \text{MAXCOVERORACLE}(\mathcal{L}_i', \mathcal{F}_{\mathcal{L}}, \ell)$ ▷ w.h.p. $\mathcal{C}(\mathbf{x}^i) \geq 1 - \beta/3$ when $\ell \geq k$
      ▷ in **no additional** pass, updates probab. $\mathbf{q}$ over sampled elts accord. to $\mathbf{x}^i$
      $\mathbf{z} \leftarrow \mathbf{0}_{|\mathcal{L}|\times 1}$ ▷ compute coverage over *sampled* elements
      for all element-set pair $e \in S$ where $S \in \mathcal{F}_{\mathcal{L}}$ **do**
        $z_e \leftarrow \min(z_e + x_S^i, 1)$
      $\mathbf{q}^{i+1} \leftarrow \text{UPDATEPROB}(\mathbf{q}^i, \mathbf{z})$ ▷ only update weights of elements in $\mathcal{L}$
    ▷ in **one** pass, updates probab. $\mathbf{p}^{\mathsf{curr}}$ over *all* (rare) elts according to $\mathbf{x}^1, \ldots, \mathbf{x}^R$
    $\mathbf{z}^1, \ldots, \mathbf{z}^R \leftarrow \mathbf{0}_{n\times 1}$ ▷ compute coverage over *all* (rare) elements
    for all element-set pair $e \in S$ in the stream **do**
      for all round $i = 1, \ldots, R$ **do**
        $z_e^i \leftarrow \min(z_e^i + x_S^i, 1)$
    for all round $i = 1, \ldots, R$ **do**
      if $(\mathbf{p}^{\mathsf{curr}})^\top \mathbf{z}^i < 1 - \beta/3$ **then** ▷ detect infeasible solutions
        report INFEASIBLE
      $\mathbf{x}^{\mathsf{total}} \leftarrow \mathbf{x}^{\mathsf{total}} + \mathbf{x}^i$, $\mathbf{p}^{\mathsf{curr}} \leftarrow \text{UPDATEPROB}(\mathbf{p}^{\mathsf{curr}}, \mathbf{z}^i)$ ▷ perform actual updates
  $\mathbf{x}^{\mathsf{rare}} \leftarrow \frac{\mathbf{x}^{\mathsf{total}}}{(1-\beta)T}$ ▷ scaled up the solution to cover *rare* elements
  return $\mathbf{x}^{\mathsf{cmn}} + \mathbf{x}^{\mathsf{rare}}$

constraints $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$. In terms of the Set Cover problem, $A_{e,S} \geq 0$ indicates the multiplicity of an element e in the set S, $b_e > 0$ denotes the number of times we wish e to

be covered, and $c_S > 0$ denotes the cost per unit for the set S. Now define

$$L \triangleq \min_{(e,S):A_{e,S}\neq 0} \frac{A_{e,S}}{b_e c_S} \quad \text{and} \quad U \triangleq \max_{(e,S)} \frac{A_{e,S}}{b_e c_S}.$$

Then, we may modify our algorithm to obtain the following result.

Theorem 3.3.11. *There exists a streaming algorithm that w.h.p. returns a $(1+\varepsilon)$-approximate fractional solution to general covering LPs in p passes and using $\widetilde{O}(\frac{mU}{\varepsilon^6 L} \cdot n^{O(\frac{1}{p\varepsilon})} + n)$ memory for any $3 \leq p \leq \text{polylog}(n)$, where parameters L and U are defined above. The algorithm works in both set arrival and edge arrival streams.*

Proof: We modify our algorithm and provide an argument of its correctness as follows. First, observe that we can convert the input LP into an equivalent LP with all entries $b_e = c_S = 1$ by simply replacing each $A_{e,S}$ with $\frac{A_{e,S}}{b_e c_S}$. Namely, let the new parameters be $\mathbf{A}', \mathbf{b}'$ and $\mathbf{c}'$, and we consider the variable $\mathbf{x}'$ where $x'_S = c_S x_S$. It is straightforward to verify that $\mathbf{c}'^{\top}\mathbf{x}' = \mathbf{c}^{\top}\mathbf{x}$ and $\mathbf{A}'_e\mathbf{x}' = \frac{\mathbf{A}_e\mathbf{x}}{b_e}$, reducing the LP into the desired case. Thus, we may afford to record $\mathbf{b}$ and $\mathbf{c}$, so that each value $\frac{A_{e,S}}{b_e c_S}$ may be computed on-the-fly. Henceforth we assume that all entries $b_e = c_S = 1$ and $A_{e,S} \in \{0\} \cup [L, U]$. Observe as well that the optimal objective value k may be in the expanded range $[1/U, n/L]$, so the number of guesses must be increased from $\frac{\log n}{\varepsilon}$ to $\frac{\log(nU/L)}{\varepsilon}$.

Next consider the process for covering the rare elements. We instead use a uniform solution $\mathbf{x}^{\mathsf{cmn}} = \frac{\alpha\ell L}{m} \cdot \mathbf{1}$. Observe that if an element occurs in at least $\frac{m}{\alpha\ell L}$ sets, then $\mathbf{A}_e\mathbf{x}^{\mathsf{cmn}} = \sum_{S:e\in S} A_{e,S} \cdot \frac{\alpha\ell}{m} \geq \frac{m}{\alpha\ell L} \cdot L \cdot \frac{\alpha\ell}{m} = 1$. That is, we must adjust our definition so that an element is considered common if it appears in at least $\frac{m}{\alpha\ell L}$ sets. Consequently, whenever we perform element sampling, the required amount of memory to store information of each element increases by a factor of $1/L$.

Next consider Lemma 3.3.5, where we show an existence of integral solutions via the MWU algorithm with a greedy oracle. As the greedy implementation chooses a set S and places the entire budget ℓ on x_S, the amount of coverage $A_{e,S}x_S$ may be as large as ℓU as $A_{e,S}$ is no longer bounded by 1. Thus this application of the MWU algorithm has width $\phi = \Theta(\ell U)$ and requires $T = \Theta(\frac{\ell U \log n}{\varepsilon_{\text{MC}}^2})$ rounds. Consequently, its solution becomes $\Theta(\frac{\ell}{T}) = \Theta(\frac{\varepsilon_{\text{MC}}^2}{U \log n})$-integral. As noted in Observation 3.3.7, the number of potential solutions from the greedy

oracle increases by a power of U. Then, in Lemma 3.3.8, we must reduce the error probability of each solution by the same power. We increase the number of samples s by a factor of U to account for this change, increasing the required amount of memory by the same factor.

As in the previous case, any solution $\mathbf{x}$ may always be pruned so that the width is reduced to 1: our algorithm PRUNE still works as long as the entries of $\mathbf{A}$ are non-negative. Therefore, the fact that entries of $\mathbf{A}$ may take on values other than 0 or 1 does not affect the number of rounds (or passes) of our overall application of the MWU framework. Thus, we may handle general covering LPs using a factor of $\widetilde{O}(U/L)$ larger memory within the same number of passes. In particular, if the non-zero entries of the input are bounded in the range $[1, M]$, this introduces a factor of $\widetilde{O}(U/L) \leq \widetilde{O}(M^3)$ overhead in memory usage. $\square$

Chapter 4

Sublinear Set Cover

4.1. Introduction

It is well-known that although the problem of finding an optimal solution is NP-complete, a natural greedy algorithm which iteratively picks the "best" remaining set is provably optimal. 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 [145, 172], 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, [119] (building on [84]) 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 randomized rounding yields an $O(\log n)$-approximate solution to Set Cover with the same complexity.

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

In this chapter we study the complexity of sub-linear time algorithms for set cover with multiplicative approximation guarantees. Our upper bounds complement the aforementioned result of [119] by presenting algorithms which are fast when k is *large*, as well as algorithms that provide more accurate solutions 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 upper and lower bounds are presented in Table 4.1.1.

Data access model. As in the prior work [145, 172] 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.

4.1.1. Our Results

In this chapter, we present algorithms and lower bounds for the Set Cover problem. The results are summarized in Table 4.1.1. The NP-hardness of this problem (or even its $o(\log n)$-approximate version [72, 152, 14, 138, 63]) precludes the existence of highly accurate algorithms with fast running times, while (as we show) it is still possible to design algorithms

[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., [45, 80, 31].

Problem	Approximation	Constraints	Query Complexity	Section
Set Cover	$\alpha\rho+\varepsilon$	$\alpha \geq 2$	$\widetilde{O}(\frac{1}{\varepsilon}(m(\frac{n}{k})^{\frac{1}{\alpha-1}}+nk))$	4.4.1
	$\rho+\varepsilon$	-	$\widetilde{O}(\frac{mn}{k\varepsilon^2})$	4.4.2
	α	$k < (\frac{n}{\log m})^{\frac{1}{4\alpha+1}}$	$\widetilde{\Omega}(m(\frac{n}{k})^{1/(2\alpha)})$	4.6
	α	$\alpha \leq 1.01$ and $k = O(\frac{n}{\log m})$	$\widetilde{\Omega}(\frac{mn}{k})$	4.3.2
Cover Verification	-	$k \leq n/2$	$\Omega(nk)$	4.5

Table 4.1.1: 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.

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 [97] 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 [119], 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.

4.1.2. Related work

Sub-linear 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 [145] consider Set Cover under the oracle model we employ in this chapter, 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. [172]. Additionally, the results of Kuhn et al. [121] on general packing/covering LPs in the distributed $\mathcal{LOCAL}$ model, together with the reduction method of Parnas and Ron [149], 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 of Vertex Cover under the adjacency-list oracle model has been studied in [149, 130, 148, 172]. Set Cover has been also studied in the sub-linear *space* context, most notably for the streaming model of computation [155, 67, 42, 20, 17, 29, 105, 61, 97]. 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 [61, 97].

4.1.3. Our Techniques

Overview of the algorithms. The algorithmic results presented in Section 4.4, use the techniques introduced for the streaming Set Cover problem by [61, 97] 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 [97] 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.

The SetCover 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 4.3.6): 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 Algorithm 10) 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.

4.2. Preliminaries

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 4.2.1 (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 $\mathsf{swap}(e, e')$, $S = (S \cup \{e'\}) \setminus \{e\}$ 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 4.2.2. 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.

4.3. 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 4.3.1), and for large values of k (in Section 4.3.2). For low values of k, we prove the following theorem whose proof is postponed to Section 4.6.

Theorem 4.3.1. *For $2 \le k \le (\frac{n}{16\alpha \log m})^{\frac{1}{4\alpha+1}}$ and $1 < \alpha \le \log n$, 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 4.3.1 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 4.3.2 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 4.3.2. *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 4.3.3. *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 4.3.2 we provide a construction utilizing Theorem 4.3.2 to extend Corollary 4.3.3, establish the following theorem on lower bounds for larger minimum set cover sizes.

Theorem 4.3.4. *For any sufficiently small approximation factor $\alpha \leq 1.01$ and $k = O(\frac{m}{\log n})$, 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.*

4.3.1. The Set Cover Lower Bound for Small Optimal Value k

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 4.3.5 (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 4.3.6. *There exists a median instance I^* satisfying all properties from Definition 4.3.5. In fact, with high probability, an instance drawn from the distribution in which*

$\Pr e \in S = 1 - p_0$ *independently at random, satisfies the median properties.*

The proof of the lemma follows from standard applications of concentration bounds. Specifically, it follows from the union bound and Lemmas 4.7.1–4.7.6, appearing in Section 4.7.

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 Algorithm 10 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 4.3.7 (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).

Algorithm 10 The procedure of constructing a modified instance of I^*.

procedure 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 all $e \in \mathcal{U} \setminus (S_1 \cup S_2)$ **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

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 4.3.8. *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 (see Definition 4.3.5): $|\mathcal{U} \setminus (S_1 \cup S_2)| \leq 18 \log m$ 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. □

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 Algorithm 10, 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 4.3.9. *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^*,

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

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, $\Pr S = S_2 = 1/m$. 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

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

Therefore,

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

Proof of Theorem 4.3.2. 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 4.3.10. *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: Let $\mathcal{A}(I)$ denote the algorithm's output for input instance I (whether the given instance is I^* or drawn from $\mathcal{D}(I^*)$). For each query q, let $\mathsf{ans}_I(q)$ denote the answer of I to query q. Observe that since $\mathcal{A}$ is deterministic, if all of the oracle's answers to its previous queries are all the same, then it must make the same next query. Combining this fact with the union bound, we may lower bound the probability that $\mathcal{A}$ returns the same

outputs on I^* and $I' \sim \mathcal{D}(I^*)$ as follows:

$$\Pr \mathcal{A}(I^*) \neq \mathcal{A}(I') \leq \sum_{t=1}^{|Q|} \Pr \mathsf{ans}_{I^*}(q_t) \neq \mathsf{ans}_{I'}(q_t).$$

For each $q \in Q$, let $S(q)$ and $e(q)$ denote respectively the set and element queried by q. Applying Lemma 4.3.9, we obtain

$$\Pr \mathcal{A}(I^*) \neq \mathcal{A}(I') \leq \sum_{t=1}^{|Q|} \Pr \mathsf{ans}_{I^*}(q_t) \neq \mathsf{ans}_{I'}(q_t) \leq \sum_{t=1}^{|Q|} P_{\mathsf{Elt-Set}}(e(q_t), S(q_t)) \leq \frac{4800 \log m}{mn} |Q| \square$$

Proof of Theorem 4.3.2. 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 4.3.10,

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

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. $\square$

4.3.2. 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 Set Cover 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. Next, we provide an high-level overview of the 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 4.3.2. 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 4.3.11. *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, $\sum_{j=1}^{\ell} \Pr[I_i = I^j] \leq (1 - c/\binom{m}{2}) \cdot (1/\binom{m}{2}) < 1/\binom{m}{2}$ 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, $p(\mathfrak{I}_\mathsf{r}) \geq c^{|R|} \cdot \sum_{j=1}^{\ell} p(\mathfrak{I}^j)$ (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 $p^{\mathsf{tough}}(\mathfrak{I}) \leq \frac{p(\mathfrak{I})}{0.98-\delta}$. Thus,

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

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

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

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

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

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 $\mathbf{E}_{\mathfrak{I}\sim\mathfrak{D}}[\mathsf{cost}(\mathfrak{I})] \geq (0.98-\delta)\mathbf{E}_{\mathfrak{I}\sim\mathfrak{D}^{\mathsf{tough}}}[\mathsf{cost}(\mathfrak{I})] \geq (0.98-\delta)\cdot 0.11t \geq 0.1t$, 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 4.3.4 then follows, noting the substitution of our problem size: $\frac{mnt}{\log m} = \frac{(m'/t)(n'/t)t}{\log(m'/t)} = \Theta(\frac{m'n'}{k'\log m'})$.

Proof of Lemma 4.3.11. First, we recall the following result from Lemma 4.3.10 for distinguishing between I^* and a random $I' \sim \mathcal{D}(I^*)$.

Corollary 4.3.12. *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

[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 [139] and the description therein.

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 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 4.3.12 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 4.3.11 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 4.3.13. *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, $\Pr i \in \mathcal{T}_{\text{low}} < \frac{1}{b}$ independently

of other urns, and thus $|\mathcal{T}_{\text{low}}|$ is stochastically dominated by $\mathrm{B}(t, \frac{1}{b})$, the binomial distribution with t trials and success probability $\frac{1}{b}$. Applying Chernoff bound, we obtain

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

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

Lemma 4.3.14. *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 4.3.13: 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 4.3.13. □

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 4.3.11.

4.4. Sub-Linear Algorithms for the Set Cover Problem

In this chapter, 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:

Theorem 4.4.1. *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.*

Our algorithms use the following two sampling techniques developed for Set Cover in the streaming model [61]: 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 [61] for the proofs.

Lemma 4.4.2 (Element Sampling). *Consider an instance of* Set Cover *on $(\mathcal{U}, \mathcal{F})$ whose optimal cover has size at most k. Let $\mathcal{U}_{\mathsf{smp}}$ be a subset of $\mathcal{U}$ of size $\Theta\left(\frac{\rho k \log m}{\delta}\right)$ chosen uniformly at random, and let $\mathcal{C}_{\mathsf{smp}} \subseteq \mathcal{F}$ be a ρ-approximate cover for $\mathcal{U}_{\mathsf{smp}}$. Then, w.h.p. $\mathcal{C}_{\mathsf{smp}}$ covers at least $(1-\delta)|\mathcal{U}|$ elements.*

Lemma 4.4.3 (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*

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

[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.

appear in $\Omega(\frac{m \log n}{\ell})$ *sets of* $\mathcal{F}$.

4.4.1. First Algorithm: small values of k

The algorithm of this section is a modified variant of the streaming algorithm of Set Cover in [97] that works in the sub-linear query model. Similarly to the algorithm of [97], 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 OfflineSC 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 [97] 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 [97] 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 [97] 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 4.2 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 $(1+\varepsilon)$ 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 4.4.2, 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 constructed 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 4.4.4. SmallSetCover *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 proce-

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

Algorithm 11 IterSetCover is the main procedure of the SmallSetCover algorithm for the Set Cover problem.

procedure IterSetCover($\alpha, \varepsilon, l, u$)
  ▷ 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**
    $\text{SOL}_\ell \leftarrow$ collection of ℓ sets picked uniformly at random ▷ Set Sampling
    $\mathcal{U}_{\text{rem}} \leftarrow \mathcal{U} \setminus \bigcup_{S \in \text{SOL}_\ell} S$ ▷ $n\ell$ EltOf
    for $i = 1$ to $\alpha - 2$ **do**
      $\mathsf{S} \leftarrow$ sample of $\mathcal{U}_{\text{rem}}$ of size $\widetilde{O}(\rho\ell \left(\frac{n}{\ell}\right)^{\frac{1}{\alpha-1}})$
      $\mathcal{D} \leftarrow \text{OfflineSC}(\mathsf{S}, \ell)$
      if $\mathcal{D} =$ null **then**
        break ▷ Try the next value of ℓ
      $\text{SOL}_\ell \leftarrow \text{SOL}_\ell \bigcup \mathcal{D}$
      $\mathcal{U}_{\text{rem}} \leftarrow \mathcal{U}_{\text{rem}} \setminus \bigcup_{S \in \mathcal{D}} S$▷ $\rho n\ell$EltOf
    if $|\mathcal{U}_{\text{rem}}| \le \ell \left(\frac{n}{\ell}\right)^{1/(\alpha-1)}$ **then** ▷ Feasibility Test
      $\mathcal{D} \leftarrow \text{OfflineSC}(\mathcal{U}_{\text{rem}}, \ell)$
      if $\mathcal{D} \neq$ null **then**
        $\text{SOL}_\ell \leftarrow \text{SOL}_\ell \bigcup \mathcal{D}$
        return SOL_ℓ

dures invoked by SmallSetCover: IterSetCover and OfflineSC. The OfflineSC procedure 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 OfflineSC 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 4.4.5. *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* OfflineSC(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 4.4.6. *The cover constructed by the outer loop of* IterSetCover$(\alpha, \varepsilon, l, u)$ *with the parameter $\ell > k$,* SOL_ℓ*, w.h.p. covers $\mathcal{U}$.*

Algorithm 12 OFFLINESC(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.

procedure OFFLINESC(S, ℓ)
  $\mathcal{F}_{\mathsf{S}} \leftarrow \emptyset$
  for all 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 a solver
  ▷ If there exists no such cover, then $\mathcal{D} = \mathsf{null}$
  return $\mathcal{D}$

Proof: After picking ℓ sets uniformly at random, by Set Sampling (Lemma 5.2.3), 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 4.4.2 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, OFFLINESC is invoked on the remaining elements; hence, SOL_ℓ 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 4.4.7. *Given that* $l \le k \le \frac{u}{1+\varepsilon/(2\alpha\rho)}$, *w.h.p.* ITERSETCOVER$(\alpha, \varepsilon, l, u)$ *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 4.4.6, ITERSETCOVER terminates with a guess value $\ell \le \ell_k$. In the following we compute the query complexity of the run of ITERSETCOVER with a parameter $\ell \le \ell_k$.

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 5.2.3), 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 $\widetilde{O}(m/\ell)$, OFFLINESC 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 4.4.5). By the guarantee of **Element Sampling** (Lemma 4.4.2), 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 OFFLINESC 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 4.4.6, 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:

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

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 OFFLINESC 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.

Algorithm 13 The description of the SMALLSETCOVER algorithm.

```
1: procedure SmallSetCover(α, ε)
2:     sol ← IterSetCover(log n, 1, 1, n)
3:     k' ← |sol| ▷ Find a ρ log n estimate of k.
4:     return IterSetCover(α, ε, ⌊k'/(ρ log n)⌋, ⌈k'(1 + ε/(2αρ))⌉)
```

Proof of Theorem 4.4.4. 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 $k \leq k' = |\text{sol}| \leq (\rho\log n)k$, 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 4.4.7, 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:

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

4.4.2. Second Algorithm: large values of k

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 4.5 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 Algorithm 14, first randomly picks $\varepsilon\ell/3$ sets. By Set Sampling (Lemma 5.2.3), 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$(e, \frac{cm\log n}{\varepsilon\ell})$. We now prove the desired performance of LargeSetCover.

Lemma 4.4.8. 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

Algorithm 14 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 $|\mathcal{F}|$.

```
1:  procedure LargeSetCover(ε)
2:      ▷ try all (1 + ε/(3ρ))-approximate guesses of k
3:      for ℓ ∈ {(1 + ε/(3ρ))^i | 0 ≤ i ≤ log_{1+ε/(3ρ)} n} in the decreasing order do
4:          rnd_ℓ ← collection of εℓ/3 sets picked uniformly at random ▷ set sampling
5:          F_rare ← ∅ ▷ intersection with rare elements
6:          for all e ∈ U do
7:              if e appears in ≤ cm log n/(εℓ) sets then ▷ size test: SetOf(e, cm log n/(εℓ))
8:                  F_e ← collection of sets containing e ▷ Õ(m/(εℓ)) SetOf queries
9:                  F_rare ← F_rare ∪ F_e
10:                 S ← S ⋃ {e}
11:         D ← solution of Set Cover(S, F_rare) returned by a ρ-approximation algorithm
12:         if |D| ≤ ρℓ then
13:             SOL ← rnd_ℓ ∪ D
14:         else
15:             return SOL ▷ solution for the previous value of ℓ
```

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 4.4.9. *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

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

4.5. 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 4.5.1. *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.*

Proof: 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 4.3.2: 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 4.3.13-4.3.14, 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

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

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. □

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

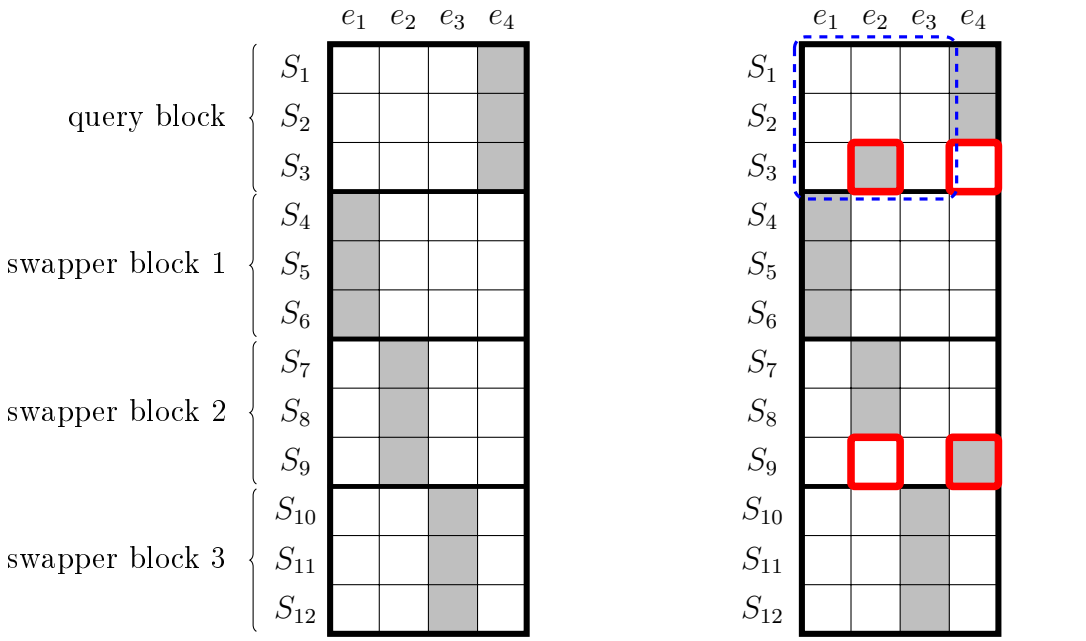
approximate solution itself; any algorithm bypassing this $\Omega(nk)$ lower bound may not solve Cover Verification as a subroutine.

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.

4.5.1. 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 4.5.1 (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 $\{e_1, \ldots, e_{n/k}\} \setminus \{e_r\}$. 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 4.5.1; 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.

Lastly, we may create the full instance by placing all k slabs together, as shown in Figure 4.5.2, shifting the elements' indices as necessary. The structure of our sets may be



(a) a basic slab **(b)** after performing a $(3, 2)$-swap

Figure 4.5.1: A basic slab and an example of a swapping operation.

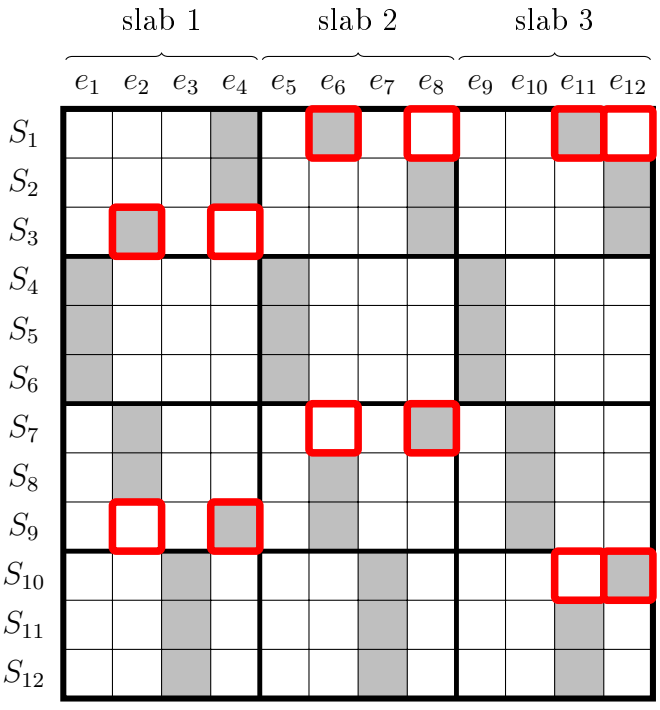


Figure 4.5.2: A example structure of a Yes instance; all elements are covered by the first 3 sets.

specified solely by the swaps made on these slabs. We define the structure of our instances as follows.

1. For a Yes instance, we make one random swap on each slab. This allows the first k sets to cover all elements.

2. 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. Figure 4.6.1 provides a sample realization of a basic slab with EltOf and SetOf, as well as a sample result of applying a swap on this basic slab; these correspond to the incidence matrices in Figure 4.5.1a and Figure 4.5.1b, respectively. Such a construction can be extended to include all k slabs. Observe here that no two distinct swaps modify the same entry; that is, the swaps do not interfere with one another on these two functions. We also note that many entries do not participate in any swap.

4.6. Generalized Lower Bounds for the Set Cover Problem

In this section we generalize the approach of Section 4.3 and prove our main lower bound result (Theorem 4.3.1) 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 4.3.1.

Theorem 4.6.1. *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^**

Before: EltOf table for a basic slab

EltOf	1	2	3
S_1	e_1	e_2	e_3
S_2	e_1	e_2	e_3
S_3	e_1	e_2	e_3
S_4	e_2	e_3	e_4
S_5	e_2	e_3	e_4
S_6	e_2	e_3	e_4
S_7	e_1	e_3	e_4
S_8	e_1	e_3	e_4
S_9	e_1	e_3	e_4
S_{10}	e_1	e_2	e_4
S_{11}	e_1	e_2	e_4
S_{12}	e_1	e_2	e_4

After: EltOf table after applying a swap

EltOf	1	2	3
S_1	e_1	e_2	e_3
S_2	e_1	e_2	e_3
S_3	e_1	e_4	e_3
S_4	e_2	e_3	e_4
S_5	e_2	e_3	e_4
S_6	e_2	e_3	e_4
S_7	e_1	e_3	e_4
S_8	e_1	e_3	e_4
S_9	e_1	e_3	e_2
S_{10}	e_1	e_2	e_4
S_{11}	e_1	e_2	e_4
S_{12}	e_1	e_2	e_4

Before: SetOf table for a basic slab

SetOf	1	2	3	4	5	6	7	8	9
e_1	S_1	S_2	S_3	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
e_2	S_1	S_2	S_3	S_4	S_5	S_6	S_{10}	S_{11}	S_{12}
e_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
e_4	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}

After: SetOf table after applying a swap

SetOf	1	2	3	4	5	6	7	8	9
e_1	S_1	S_2	S_3	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
e_2	S_1	S_2	S_9	S_4	S_5	S_6	S_{10}	S_{11}	S_{12}
e_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
e_4	S_4	S_5	S_6	S_7	S_8	S_3	S_{10}	S_{11}	S_{12}

Figure 4.6.1: Tables illustrating the representation of a slab under EltOf and SetOf before and after a swap; cells modified by $\mathsf{swap}(e_2, e_4)$ between S_3 and S_9 are highlighted in red.

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.*

Proof: Applying the same argument as that of Lemma 4.3.10, we derive that the probability that $\mathcal{A}$ returns different outputs on I^* and I' is at most

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

via the result of Lemma 4.6.12. Then, over the distribution in which we applied Yao's lemma, we have

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

where the last inequality follows from Lemma 4.6.2. 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. □

4.6.1. 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 4.6.2. *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 4.6.3 (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 4.6.4. *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 4.6.3. 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 4.6.5–4.6.10. □

The proofs of the Lemmas 4.6.5–4.6.10 follow from standard applications of concentration bounds. We include them here for the sake of completeness.

Lemma 4.6.5. *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 $\alpha k + 1$.*

Proof: The probability that an element $e \in \mathcal{U}$ is covered by a specific collection of αk sets in $\mathcal{F}$ is at most $1 - p_0^{\alpha k} = 1 - \frac{8(\alpha k+2)\log m}{n}$. Thus, the probability that the union of the αk sets covers all elements in $\mathcal{U}$ is at most $(1 - \frac{8(\alpha k+2)\log m}{n})^n < m^{-8(\alpha k+2)}$. Applying the union bound, with probability at least $1-m^{-2}$ the size of an optimal set cover is at least $\alpha k+1$.□

Lemma 4.6.6. *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.*

Proof: Let $S_1, \cdots, S_k$ be a collection of k sets from $\mathcal{F}$. For each element $e \in \mathcal{U}$, the probability that e is not covered by the union of the k sets is p_0^k. Thus,

$$\mathbb{E}[|\mathcal{U} \setminus (S_1 \cup \cdots \cup S_k)|] = p_0^k n \geq p_0^{\alpha k} n = 8(\alpha k + 2)\log m.$$

By Chernoff bound,

$$\Pr |\mathcal{U} \setminus (S_1 \cup \cdots \cup S_k)| \geq 2p_0^k n \leq e^{-\frac{p_0^k n}{3}} \leq e^{-(\alpha k+2)\log m} \leq m^{-k-2}.$$

Thus with probability at least $1 - m^{-2}$, for any collection of k sets in $\mathcal{F}$, the number of uncovered elements by the union of the sets is at most $2p_0^k n$. □

Lemma 4.6.7. *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 $mp_0(1-p_0)/2$.*

Proof: For each set S, $\Pr e \in S$ and $e' \notin S = (1-p_0)p_0$. This implies that the expected

number of such sets S satisfying the condition for e and e' is

$$p_0(1-p_0)m \geq p_0 \cdot p_0^{k/4} \cdot m \geq p_0^{\alpha k} n = 8(\alpha k + 2)\log m$$

by Lemma 4.6.2 and $m \geq n$. By Chernoff bound, the probability that the number of sets containing e but not e' is less than $mp_0(1-p_0)/2$ is at most

$$e^{-\frac{p_0(1-p_0)m}{8}} \leq e^{-(\alpha k+2)\log m} \leq m^{-\alpha k-2}.$$

Thus with probability at least $1 - m^{-2}$ property (c) holds for any pair of elements in $\mathcal{U}$. □

Lemma 4.6.8. *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 $1 - m^{-2}$, $|S_k \cap (S_1 \cup \cdots \cup S_{k-1})| \geq (1-p_0)(1-p_0^{k-1})n/2$.*

Proof: For each element e, $\Pr e \in S_k \cap (S_1 \cup \cdots \cup S_{k-1}) = (1-p_0)(1-p_0^{k-1})$. This implies that the expected size of $S_k \cap (S_1 \cup \cdots \cup S_{k-1})$ is

$$(1-p_0)(1-p_0^{k-1})n \geq p_0^{k/4} \cdot p_0^{k/4} \cdot n \geq p_0^{\alpha k} n = 8(\alpha k + 2)\log m.$$

by Lemma 4.6.2. By Chernoff bound, the probability that $|S_k \cap (S_1 \cup \cdots \cup S_{k-1})| \leq (1-p_0)(1-p_0^{k-1})n/2$ is at most

$$e^{-\frac{(1-p_0)(1-p_0^{k-1})n}{8}} \leq e^{-(\alpha k+2)\log m} \leq m^{-\alpha k-2}.$$

Thus with probability at least $1 - m^{-2}$ property (d) holds for any sets $S_1, \cdots, S_k$ in $\mathcal{F}$. □

Lemma 4.6.9. *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 $1-m^{-2}$, $|(S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus S| \leq 2p_0(1-p_0)(1-p_0^{k-1})n$.*

Proof: For each element e, $\Pr e \in (S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus S = p_0(1-p_0)(1-p_0^{k-1})$. Then,

$$\mathbb{E}(|(S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus S|) = p_0(1-p_0)(1-p_0^{k-1})n \geq p_0 \cdot p_0^{k/4} \cdot p_0^{k/4} \geq p_0^{\alpha k} n$$
$$= 8(\alpha k + 2)\log m$$

by Lemma 4.6.2. By Chernoff bound, the probability that $|(S_k \cap (S_1 \cup \cdots \cup S_{k-1})) \setminus S| \geq 2p_0(1-p_0)(1-p_0^{k-1})n$ is

$$e^{-\frac{p_0(1-p_0)(1-p_0^{k-1})n}{3}} \leq e^{-2(\alpha k+2)\log m} \leq m^{-2\alpha k-4}.$$

Thus with probability at least $1 - m^{-2}$ property (e) holds for any sets $S_1, \cdots, S_k$ and S in $\mathcal{F}$. □

Lemma 4.6.10. *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* $(1 + \frac{1}{k})p_0 m$.

Proof: First, note that $k \leq \left(\frac{n}{16\alpha \log m}\right)^{\frac{1}{4\alpha+1}} \leq \sqrt{\frac{m}{27 \ln m}}$ as $m \geq n$ and $\alpha \geq 1$.

Next, for each element e, $\mathbf{Pr}_{S\sim\mathcal{F}}[e \notin S] = p_0$. This implies that $\mathbb{E}_S(|\{S \mid e \notin S\}|) = p_0 m$. By Chernoff bound, the probability that $|\{S \mid e \notin S\}| \geq (1 + \frac{1}{k})p_0 m$ is at most $e^{\frac{-mp_0}{3k^2}}$. Now if $k \geq \log n$, then $p_0 \geq 1/e$ and thus this probability would be at most $\exp(\frac{-m}{3ek^2}) \leq m^{-3}$ for any $k \leq \sqrt{\frac{m}{27 \ln m}}$. Otherwise, we have that the above probability is at most $\exp(\frac{-mn^{-1/\alpha k}}{3\log^2 n}) \leq \exp(\frac{-m^{1-1/\alpha k}}{3\log^2 m}) \leq m^{-3}$ given $m \geq n$ and sufficiently large n. Thus with probability at least $1 - m^{-2}$ property (f) holds for any element $e \in \mathcal{U}$. □

4.6.2. 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 $\widetilde{O}(n^{1-1/\alpha}k^{1/\alpha})$. 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 Algorithm 15. 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.

Algorithm 15 The procedure of constructing a modified instance of I^*.

procedure 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 all $e \in \mathcal{U} \setminus (S_1 \cup \cdots \cup S_k)$ **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

Lemma 4.6.11. *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_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 4.6.3, these values satisfy $|\mathcal{U} \setminus (S_1 \cup \cdots \cup S_k)| \leq 2p_0^k n$ and $|S_k \cap (S_1 \cup \cdots \cup S_{k-1})| \geq (1-p_0)(1-p_0^{k-1})n/2$, respectively. By Lemma 4.6.2, $p_0^{k/4} \leq 1/2$. Using this and Lemma 4.6.2 again,

$$(1-p_0)(1-p_0^{k-1})n/2 \geq p_0^{k/4} \cdot p_0^{k/4} \cdot n/2 \geq p_0^{k/2} n/2 \geq 2p_0^k n.$$

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, $|\mathsf{Candidate}(e, e')| \geq (1-p_0)p_0 m/2$ (by property (c)), and there are plenty of candidates for each swap. □

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 4.6.12. *For any $e \in \mathcal{U}$ and $S \in \mathcal{F}$, $P_{\mathsf{Elt-Set}}(e, S) \leq \frac{64p_0^k}{(1-p_0)^2 m}$ 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^*,

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

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

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

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

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

Therefore,

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

4.7. Omitted Proofs from Section 4.3

Lemma 4.7.1. *With probability at least $1-m^{-1}$ 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 greater than 2.*

Proof: The probability that an element $e\in\mathcal{U}$ is covered by two sets selected from $\mathcal{F}$ is at most:

$$\mathbf{Pr}[e\in S_1\cup S_2] = 1-p_0^2 = 1-\frac{9\log m}{n}.$$

Thus, the probability that $S_1\cup S_2$ covers all elements in $\mathcal{U}$ is at most $(1-\frac{9\log m}{n})^n < m^{-9}$. Applying the union bound, with probability at least $1-m^{-1}$ the size of optimal set cover is greater than 2. $\square$

Lemma 4.7.2. *Let S_1 and S_2 be two sets in $\mathcal{F}$ where $\mathcal{F}\sim\mathcal{I}(\mathcal{U},p_0)$. Then with probability at least $1-m^{-1}$, $|\mathcal{U}\setminus(S_1\cup S_2)|\le 18\log m$.*

Proof: For an element e, $\mathbf{Pr}[e\notin S_1\cup S_2] = p_0^2 = \frac{9\log m}{n}$. So, $\mathbb{E}[|\mathcal{U}\setminus(S_1\cup S_2)|] = 9\log m$. By Chernoff bound, $\mathbf{Pr}[|\mathcal{U}\setminus(S_1\cup S_2)|\ge 18\log m]$ is at most $e^{-9\log m/3}\le m^{-3}$. Thus with probability at least $1-m^{-1}$, for any pair of sets in $\mathcal{F}$, the number of element not covered by their union is at most $18\log m$. $\square$

Lemma 4.7.3. *Let S_1 and S_2 be two sets in $\mathcal{F}$ where $\mathcal{F}\sim\mathcal{I}(\mathcal{U},p_0)$. Then $|S_1\cap S_2|\ge n/8$ with probability at least $1-m^{-1}$.*

Proof: For each element e, it is either covered by both S_1,S_2, one of S_1,S_2 or none of them. Since $p_0\le 1/2$, the probability that an element is covered by both sets is greater than other cases, i.e., $\mathbf{Pr}[e\in S_1\cap S_2] > 1/4$. Thus, $\mathbb{E}[|\mathcal{U}\setminus(S_1\cap S_2)|] > n/4$. By Chernoff bound, $\mathbf{Pr}[|\mathcal{U}\setminus(S_1\cap S_2)|\le n/8]$ is exponentially small. Thus with probability at least $1-m^{-1}$, the

intersection of any pairs of sets in $\mathcal{F}$ is greater than $n/8$. □

Lemma 4.7.4. *Suppose that $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$ and let e, e' be two elements in $\mathcal{U}$. With probability at least $1 - m^{-1}$, the number of sets $S \in \mathcal{F}$ such that $e \in S$ but $e' \notin S$ is at least $\frac{m\sqrt{9 \log m}}{4\sqrt{n}}$.*

Proof: For each set S, $\mathbf{Pr}[e \in S \text{ and } e' \notin S] = (1 - p_0)p_0 \geq p_0/2$. This implies that the expected number of S satisfying the condition for e and e' is at least $\frac{m}{2} \cdot \sqrt{\frac{9 \log m}{n}}$ and by Chernoff bound, the probability that the number of sets containing e but not e' is less than $\frac{m\sqrt{9 \log m}}{4\sqrt{n}}$ is exponentially small. Thus with probability at least $1 - m^{-1}$ property (d) holds for any pair of elements in $\mathcal{U}$. □

Lemma 4.7.5. *Suppose that $\mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0)$ and let S_1, S_2 and S be sets in $\mathcal{F}$. With probability at least $1 - n^{-1}$, $|(S_1 \cap S_2) \setminus S| \leq 6\sqrt{n \log m}$.*

Proof: For each element e, $\mathbf{Pr}[e \in (S_1 \cap S_2) \setminus S] = (1 - p_0)^2 p_0 \leq p_0$. This implies that the expected size of $(S_1 \cap S_2) \setminus S$ is less than $\sqrt{9n \log m}$ and by Chernoff bound, the probability that $|(S_1 \cap S_2) \setminus S| \geq 6\sqrt{n \log m}$ is exponentially small. Thus with probability at least $1 - m^{-1}$ property (e) holds for any sets S_1, S_2 and S in $\mathcal{F}$. □

Lemma 4.7.6. *For each element, the number of sets that do not contain the element is at most $6m\sqrt{\frac{\log m}{n}}$.*

Proof: For each element e, $\mathbf{Pr}_S[e \notin S] = p_0$. This implies that $\mathbb{E}_S(|\{S \mid e \notin S\}|)$ is less than $m\sqrt{\frac{9 \log m}{n}}$ and by Chernoff bound, the probability that $|\{S \mid e \notin S\}| \geq 2m\sqrt{\frac{9 \log m}{n}}$ is exponentially small. Thus with probability at least $1 - m^{-1}$ property (f) holds for any element $e \in \mathcal{U}$. □

Chapter 5

Streaming Maximum Coverage

5.1. Introduction

In maximum k-coverage (Max k-Cover), given a ground set $\mathcal{U}$ of n elements, a family of m sets $\mathcal{F}$ (each subset of $\mathcal{U}$), and a parameter k, the goal is to select k sets in $\mathcal{F}$ whose union has the largest cardinality. The initial streaming algorithms for this problem were developed in the *set arrival* model, where the input sets are listed contiguously. This restriction is natural from the perspective of submodular optimization, but limits the applicability of the algorithms.[1] Avoiding this limitation can be difficult, as streaming algorithms can no longer operate on sets as "unit objects". As a result, the first maximum coverage algorithm for the general *edge arrival* model, where pairs of (set, element) can arrive in arbitrary order, have been developed recently. In particular [29] presented a one-pass algorithm with space linear in m and constant approximation factor.[2]

A particularly interesting line of research in set arrival streaming set cover and max k-cover is to design efficient algorithms that only use $\widetilde{O}(n)$ space [155, 22, 67, 42, 131]. Previous work have shown that we can adopt the existing greedy algorithm of Max k-Cover to achieve constant factor approximation in $\widetilde{O}(n)$ space [155, 22] (which later improved to

[1] For example, consider a situation where the sets correspond to neighborhoods of vertices in a directed graph. Depending on the input representation, for each vertex, either the ingoing edges or the outgoing edges might be placed non-contiguously.

[2] We remark that many of the prior bounds (both upper and lower bounds) on set cover and max k-cover problems in set-arrival streams also work in edge arrival streams (e.g. [61, 97, 20, 131, 17, 105]). However, the design of efficient streaming algorithms for the coverage problems on edge arrival streams was first studied explicitly in [29].

$\widetilde{O}(k)$ by [131]). However, the complexity of the problem in the "low space" regime is very different in edge-arrival streams: [29] showed that as long as the approximation factor is a constant, any algorithm must use $\Omega(m)$ space. Still, our understanding of approximation/space tradeoffs in the general case is far from complete. Table 5.1.1 summarizes the known results.

5.1.1. Our Results

In this chapter, we complement the work of [29] by designing space-efficient $(1/\alpha)$-approximation algorithms for super-constant values of α. In fact, we show a tight (up to polylogarithmic factors) tradeoff between the two: the optimal space bound is $\widetilde{O}(m/\alpha^2)$ for estimating the maximum coverage value, and $\widetilde{O}(m/\alpha^2+k)$ for reporting an approximately optimal solution.[3] The approximation factor α can take any value in $[1/\widetilde{\Theta}(\sqrt{m}), 1-1/e)$.

5.1.2. Our Techniques

In the edge arrival model, elements of each set can arrive irregularly and out of order. This necessitates the use of methods that aggregate the information about the input sets, or their coverage. In particular, distinct element sketches were used both in [29] (implicitly) and [131] (explicitly). In this chapter we expand the use of sketching toolkit. Specifically, in addition to distinct element estimation [13, 28, 112, 113, 32], we also need algorithms for *heavy hitters* with respect to the L_2 norm [44, 165, 38, 37], as well as a frequency-based partitioning of elements, and detecting sets that "substantially contribute" to the solution [108]. Application of vector-sketching techniques (e.g. L_p-sampling/estimation and heavy hitters) in graph streaming settings have been studied extensively (e.g. [8, 9, 89, 21, 49, 114]). We believe that our algorithms can lead to further connections between vector sketching methods and streaming algorithms for the coverage problems.

[3] We note that similar tradeoffs were previously obtained for the set cover problem, as [20] showed a $\Theta(mn/\alpha^2)$ bound for estimation, and a $\Theta(mn/\alpha)$ bound for reporting. Interestingly, the $1/\alpha^2$ vs. $1/\alpha$ gap does not occur for our problem.

[4] Their result works for the general submodular maximization assuming access to a value oracle that given a collection of sets computes their coverage. A careful adoption of their result to Max k-Cover (without the value oracle) uses $\widetilde{O}(n)$ space.

Problem	Stream Model	Approximation	Upper Bound	Lower Bound
Estimation	Edge Arrival	$1-\varepsilon$	–	$\widetilde{\Omega}(\frac{m}{\varepsilon^2})$[17]
		$1-\frac{1}{e}-\varepsilon$		$\Omega(\frac{m}{k^2})$[131]
Reporting	Edge Arrival	$1-\frac{1}{e}-\varepsilon$	$\widetilde{O}(\frac{m}{\varepsilon^3})$[29]	–
			$\widetilde{O}(\frac{m}{\varepsilon^2})$[131]	
	Set Arrival	$\frac{1}{4}$ [155], $\frac{1}{2}$ [22][4]	$\widetilde{O}(n)$	
		$\frac{1}{2}-\varepsilon$	$\widetilde{O}(\frac{k}{\varepsilon^3})$[131]	
Estimation	Edge Arrival	$1/\alpha$	$\widetilde{O}(\frac{m}{\alpha^2})$ **[here]**	$\Omega(\frac{m}{\alpha^2})$ **[here]**,[29]
Reporting	Edge Arrival	$1/\alpha$	$\widetilde{O}(k+\frac{m}{\alpha^2})$ **[here]**	–

Table 5.1.1: The summary of known results on the space complexity of single-pass streaming algorithms of Max k-Cover.

Lower bound. Our algorithm was inspired by the lower bound. Specifically, it was previously shown by [29] that approximating Max k-Cover by a factor better than 2 requires $\Omega(m)$ space. Similar approach works for larger values of α, by showing a reduction from the α-player *set disjointess* problem ($\mathsf{DSJ}_{[\mathsf{m}]}$) with *unique intersection* guarantee (i.e., either players' sets are disjoint or there is a unique item that appears in all sets) to the task of α-approximating Max k-Cover.

The specific hard instances in the aforementioned lower bound can be distinguished in the streaming model using space $O(m/\alpha^2)$. To this end, we compute an α-approximation to the L_∞-norm of a vector v that, for each element e, counts the number of sets e belongs to. This problem can be solved in $O(m/\alpha^2)$ space, by using L_2-norm sketches [13]. This suggests that it might be possible to solve the *general* Max k-Cover using sketching techniques as well.

Upper bound. We start our algorithm with a "coverage boosting" *universe reduction* technique which constructs a reduced size instance (i.e., with reduced ground set) whose optimal k-cover has constant fraction coverage (see Section 5.3.1). This step is particularly important as the space complexity of the existing methods for Max k-Cover is proportional to the reciprocal of the fraction of covered elements in an optimal solution.

Once we have a constant fraction coverage guarantee, our algorithm exploits three different approaches so that on any instance, at least one of them reports a "good" approximate solution.

Multi-layered set sampling. By extending the *set sampling* approach (see Section 5.2.1) and trying a larger range of sampling rate, $[\frac{k}{m}, \frac{\alpha k}{m}]$, we design a *smooth* variant of set sampling: a collection of sets sampled uniformly and independently[5] at rate $\widetilde{O}(\beta k/m)$ w.h.p., covers all elements that appear in at least $m/(\beta k)$ sets. Besides expanding the application of set sampling in finding $(1/\alpha)$-approximate k-cover, this smooth variant implies more structure on the number of elements in a wider range of frequency levels which is specifically crucial in our approach for detecting sets with "low contribution".

Unlike the set sampling based technique whose success in finding an α-approximate k-cover *only* depends on the structure of the set system, the performance of the next two approaches rely on the structure of optimal solutions as well: whether the majority of the coverage (in a specific optimal k-cover) is due to (few) "large" sets or, (many) "small" sets.

Heavy hitters and contributing frequencies. The high level idea in this approach is that *if in an optimal solution, a sufficiently*[6] *small number of sets cover the majority of the elements (covered by the optimal solution), it is enough to find a single large set*, which naturally hints the use of ideas related to heavy hitters. For the sake of efficiency (in space complexity), we *randomly* partition sets into *supersets* of size at most k. However, once we merge sets into a single superset, we can no longer distinguish between their coverage and their total size. Since we combine sets at random, if all elements have "low" frequency in the set system, then the gap between the total size of all sets in a superset and their coverage is just $\widetilde{O}(1)$. This observation implies that if there is no "common" element in the set system, then we can use the total size of the sets in a superset as an estimate of its coverage size. To get around the case with (many) "common" elements, we show that performing the heavy hitter based algorithm on a sampled set of elements will find a sufficiently large superset as desired (see Section 5.8).

[5] In fact, $O(\log mn)$-wise independent is sufficient for all applications in this chapter.
[6] Depending on how large the value of α is.

Detecting k-covers with many small sets. Finally, we address the case in which an optimal k-cover consists of many "small" sets. In this case, we can show that after subsampling sets uniformly with probability $1/\alpha$, a $(\frac{k}{\alpha})$-cover with coverage at least $\Theta(1/\alpha)$ times the coverage of an optimal k-cover survives. This sampling method will save us a factor of α in the memory usage of the algorithm. Further, by exploiting the structural property guaranteed due to the multi-layered set sampling[7], we can show that element sampling can save another factor of α in the space complexity once applied to find a constant factor approximate Max $(\frac{k}{\alpha})$-Cover of the subsampled sets.

5.1.3. Other Related Work

Another important related question in this area is to design a "low-approximation" (i.e., better than the 2-approximation guarantee of the greedy approach) streaming algorithm for the max k-cover problem in the *set arrival* setting. Recently, Norouzi-Fard et al. [147] presented the first streaming algorithm that improves upon 2-approximation guarantee of greedy approach on *random arrival* streams. Very recently, Agrawal et al. [5] achieved an almost $(1-1/e)$-approximation in $\widetilde{O}(n)$ space which is essentially the optimal bound [131].[8] Still it is an important question to design such algorithms on adversarial order streams. We also remark that the algorithms of [147, 5] do not work on *edge arrival* streams.

In many scenarios, space is the most critical factor, and thus the question becomes: what approximation guarantees are possible within the given space bounds? This question has been studied before in the context of *set cover* in set arrival streams (e.g. [67, 42]), leading to poly(n, m)-factor approximation algorithms.

5.2. Preliminaries and Notations

5.2.1. Sampling Methods for Max k-Cover and Set Cover

Here we describe two sampling methods that have been used widely in the design of streaming algorithms for Max k-Cover and Set Cover [123, 61, 97, 20, 131, 17, 29, 105]. For a collection

[7] If the multi-layered set sampling fails to return a $(1/\alpha)$-approximate estimate, we can infer strong conditions on the maximum number of elements that belong to each frequency level in $[\frac{m}{k}, \frac{m}{\alpha k}]$.

[8] Both [147, 5] study the more general problem of submodular maximization and their results are stated with different notation and assuming *oracle* access. Here, we state their guarantees for max cover on set arrival streams

of sets $\mathcal{Q}$, we define $\mathcal{C}(\mathcal{Q})$ to denote the set of elements that are covered by $\mathcal{Q}$; $\mathcal{C}(\mathcal{Q}) := \bigcup_{S\in\mathcal{Q}} S$. Moreover, we denote an optimal k-cover of $(\mathcal{U},\mathcal{F})$ by OPT.

Set Sampling. Roughly speaking, it says that by selecting sets *uniformly at random*, with high probability, all elements that appear in large number of sets will be covered.

Definition 5.2.1. An element $e \in \mathcal{U}$ is called λ-*common* if it appears in at least $cm\,\text{polylog}(m,n)/\lambda$ sets in $\mathcal{F}$. Furthermore, We denote the set of λ-common elements by $\mathcal{U}^{\mathsf{cmn}}_{\lambda}$.

Observation 5.2.2. *For any* $0 \le \lambda_1 \le \lambda_2$, $\mathcal{U}^{\mathsf{cmn}}_{\lambda_1} \subseteq \mathcal{U}^{\mathsf{cmn}}_{\lambda_2}$.

Lemma 5.2.3 (Set Sampling [61]). *Consider a set system* $(\mathcal{U},\mathcal{F})$ *and let* $\mathcal{F}^{\mathsf{rnd}} \subseteq \mathcal{F}$ *be a collection of sets such that each set* S *is picked in* $\mathcal{F}^{\mathsf{rnd}}$ *with probability* $\frac{\lambda}{m}$. *With high probability,* $\mathcal{F}^{\mathsf{rnd}}$ *covers all elements that appear in* $\widetilde{\Omega}(m/\lambda)$ *sets (i.e.* λ*-common elements).*

Element Sampling for Max k-Cover . This sampling method shows that if we sample elements of $\mathcal{U}$ uniformly with a large enough rate (i.e. proportional to $(k|\mathcal{U}|)/|\mathcal{C}(\text{OPT})|$), then a constant factor approximate k-cover over the sampled elements w.h.p., is a constant factor approximate solution of the original instance.

Lemma 5.2.4 (Element Sampling Lemma [123, 61]). *Consider an instance of* Max k-Cover$(\mathcal{U},\mathcal{F})$. *Let's assume that an optimal* k*-cover of* $(\mathcal{U},\mathcal{F})$ *covers* $(1/\eta)$*-fraction of* $\mathcal{U}$. *Let* $\mathcal{L} \subset \mathcal{U}$ *be a set of elements of size* $\widetilde{\Theta}(\eta k)$ *picked uniformly at random. Then, with high probability, a* $\Theta(1)$*-approximate* k*-cover of* $(\mathcal{L},\mathcal{F})$ *is a* $\Theta(1)$*-approximate* k*-cover of* $(\mathcal{U},\mathcal{F})$.

Observation 5.2.5. *Let* $\mathcal{Q}$ *be a collection of* (βk)*-cover in* $\mathcal{F}$. *Then, in any partitioning of* $\mathcal{Q}$ *into* β *groups, there exists one group with coverage at least* $|\mathcal{C}(\mathcal{Q})|/\beta$. *In particular, an optimal* k*-cover in* $\mathcal{Q}$ *covers at least* $|\mathcal{C}(\mathcal{Q})|/\beta$.

This simple observation is in particular interesting because it relates the task of α-approximating Max k-Cover to solving instances of Max (βk)-Cover where $\beta \le \alpha$.

5.2.2. HeavyHitters and Contributing Classes

Suppose that a sequence of items $p_1,\cdots,p_T$ arrive in a data stream where for each $j \le T$, $p_j \in [m]$. We can think of the stream as a sequence of (insertion only) updates on an initially zero vector $\vec{a}$ such that upon arrival of p_j in the stream, $\vec{a}[j] \leftarrow \vec{a}[j]+1$. Here, we review

the notion of F_2-*heavy hitter* and *contributing coordinates* that are used in our algorithm for approximating Max k-Cover.

Definition 5.2.6. Given an m-dimensional vector $\vec{a}$, an item j (corresponding to $\vec{a}[j]$) is a ϕ-HeavyHitter of $F_2(\vec{a})$, if $\vec{a}[j]^2 \geq \phi \cdot F_2(\vec{a}) = \phi \cdot \sum_{j\in[m]} \vec{a}[j]^2$. Intuitively, the set of items that appear frequently in the stream are the heavy hitters.

We conceptually partition coordinates of $\vec{a}$ into classes $R_i = \{j \mid 2^{i-1} < \vec{a}[j] \leq 2^i\}$.

Definition 5.2.7. A class of coordinates R_t is **γ-contributing** if $|R_t| \cdot 2^{2t} \geq \gamma F_2(\vec{a}) = \gamma \sum_{j\in[m]} \vec{a}[j]^2$.

Let R_{t^*} be a γ-contributing class and let n_{t^*} denote the size of R_{t^*}; $n_{t^*} = |R_{t^*}|$. Further, let's assume that $i^* = \lceil \log n_{t^*} \rceil$; $2^{i^*-1} < n_{t^*} \leq 2^{i^*}$. Let $h : [m] \to [(12m \log m)/2^{i^*}]$ be a function chosen uniformly at random from a family of $\Theta(\log(mn))$-wise independent hash functions. We define $\mathcal{S}_{i^*}$ as a *sampled substream* of the input stream with rate $1/2^{i^*}$. More precisely, $\mathcal{S}_{i^*}$ only contains the updates corresponding to the coordinates $\mathcal{F}_{i^*} = \{j \mid h(j) = 1\}$ that are mapped to one under h. Next, we show that the survived coordinates of R_{t^*} ($j \in R_{t^*}$) in $\vec{a}_{i^*}$, which is the vector $\vec{a}$ restricted to the items in $\mathcal{F}_{i^*}$, are $\widetilde{\Omega}(\gamma)$-HeavyHitters of $F_2(\vec{a}_{i^*})$; $\vec{a}[j]^2 \geq \widetilde{\Omega}(\gamma) \cdot F_2(\vec{a}_{i^*})$. Roughly speaking, if we subsample the stream so that only polylog(m) coordinates of R_{t^*} survive, then with high probability these coordinates are $\widetilde{\Omega}(\gamma)$-HeavyHitters of the sampled substream.

Claim 5.2.8. *With probability at least $1 - m^{-1}$, the number of survived coordinates in the sampled substream $\mathcal{S}_{i^*}$ is at least $(6m \log m)/2^{i^*}$.*

Proof: Let X_j be a random variable which is one if item $i_j \in \mathcal{F}_{i^*}$ and zero otherwise. We define $X := X_1 + \cdots + X_m$. Note that X_j are $\Theta(\log(mn))$-wise independent and $\mathbf{E}[X] = (12m \log m)/2^{i^*}$. Then, by an application of Chernoff bound with limited independence (Lemma 5.7.3),

$$\Pr(X < (6m \log m)/2^{i^*}) \leq m^{-1}.$$

Hence, with high probability, $\mathcal{F}_{i^*}$ has size at least $(6m \log m)/2^{i^*}$. □

Lemma 5.2.9. *With probability at least $1 - 2/(9 \log^2 n \log^c m)$, a coordinate $j \in R_{t^*}$ is a*

$(\frac{\gamma}{162\log^2 n\log^{c+1} m})$-HeavyHitter *in the sampled substream* $\mathcal{S}_{t^*}$.

Proof: Let X_i be a random variable corresponding to the i-th coordinate in R_{t^*} such that $X_i = 1$ if $i \in \mathcal{F}_{i^*}$ and zero otherwise. Moreover, we define $X := X_1 + \cdots + X_{n_{t^*}}$. Then, $\mathbf{E}[X] = (12n_{t^*}\log m)/2^{i^*}$ and by an application of Chernoff bound with limited independence,

$$\Pr(X < 1) < \Pr(X < (1-\sqrt{1/6})\mathbf{E}[X]) \leq m^{-1}.$$

Next, we show that with probability at least $1 - 1/\log^c m$, $F_2(\vec{a}_{i^*}) \leq \widetilde{O}(F_2(\vec{a})/2^{i^*})$. It is straightforward to check that $\mathbf{E}[F_2(\vec{a}_{i^*})] = (12F_2(\vec{a})\log m)/2^{i^*}$. Hence, by Markov's inequality,

$$\Pr(F_2(\vec{a}_{i^*}) \geq (108F_2(\vec{a})\log^2 n\log^{c+1} m)/2^{i^*}) \leq 1/(9\log^2 n\log^c m).$$

Hence, with probability at least $1 - 2/(9\log^2 n\log^c m)$, a coordinate $j \in R_{t^*}$ survives in $\mathcal{F}_{i^*}$ and $F_2(\vec{a}_{i^*}) \leq (108F_2(\vec{a})\log^2 n\log^{c+1} m)/2^{i^*}$. Thus, with probability at least $1 - 2/(9\log^2 n\log^c m)$,

$$(\vec{a}[i])^2 \geq (2^{t^*-1})^2 \geq \frac{\gamma}{4n^{t^*}}F_2(\vec{v}) \geq \frac{\gamma}{4\cdot 2^{i^*}}(\frac{2^{i^*}}{108\log^2 n\log^{c+1} m})F_2(\vec{v}) = (\frac{\gamma}{432\log^2 n\log^{c+1} m})F_2(\vec{v})$$

In other words, with probability at least $1 - 2/(9\log^2 n\log^c m)$, a coordinate $i \in S_{t^*}$ is a $(\frac{\gamma}{432\log^2 n\log^{c+1} m})$-HeavyHitter of $F_2(\vec{a}_{i^*})$. □

Next, we can use the exiting algorithms for F_2-HeavyHitters to complete this section.

Theorem 5.2.10 (F_2-heavy hitters [44, 165, 38, 37]). *Let's assume that $\vec{a}$ is an m-dimensional vector initialized to zero. Let $\mathcal{S}$ be a stream of items $p_1, \cdots, p_T$ where for each $j \in [T]$, $p_j \in [m]$. Then, there is a single pass algorithm F_2-HeavyHitter that uses $\widetilde{O}(1/\gamma)$ space and with high probability returns all coordinates i such that $\vec{a}[i]^2 \geq \gamma F_2(\vec{a})$. In addition, it returns $(1 \pm \frac{1}{2})$-approximate values of these coordinates.*

Finally, there exists an algorithm that with probability at least $1 - 2/(9\log n\log^c m)$, finds at least one coordinate in each γ-contributing class of $\vec{a}$ using $\widetilde{O}(1/\gamma)$ space.

Theorem 5.2.11 (γ-contributing [108]). *Let's assume that $\vec{a}$ is an m-dimensional vector*

Algorithm 16 A single pass streaming algorithm that given a vector a in a stream, for each contributing class R, returns an index j along with a $(1 \pm (1/2))$-estimate of its frequency.

procedure F_2-CONTRIBUTING(γ, S)
  ▷ **Input:** stream of updates $i \in [m]$ on vector $\vec{a}$
  ▷ S is an upper bound on the size of a γ-contributing class
  for all $n_t \in \{2^i \mid i \in [\log S]\}$ in parallel **do**
    ▷ n_t : #coordinates in a γ-contributing class
    $\phi \leftarrow (\frac{\gamma}{432 \log n \log^{c+1} m})$ ▷ parameter of HeavyHitter
    let HH_ϕ be a instance of F_2-HEAVYHITTER(ϕ)
    $\rho \leftarrow (12 \log m)/2^i$ ▷ sample rate of the substream
    pick $h : [m] \to [\frac{m}{\rho}]$ from a family of $\Theta(\log(mn))$-wise independent hash functions
    for all item p_i in the input stream **do**
      if $h(p_i) = 1$ **then**
        feed p_i to HH_ϕ
    return output of HH_ϕ
    ▷ returns heavy coordinates together with their approximate frequencies

initialized to zero. Let $\mathcal{S}$ be a stream of items $p_1, \cdots, p_T$ where for each $j \in [T]$, $p_j \in [m]$. Moreover, let's assume no item in $\mathcal{S}$ has frequency more than n. There exists a single pass algorithm F_2-CONTRIBUTING that uses $\widetilde{O}(1/\gamma)$ space and with probability at least $1 - 2/(9 \log n \log^c m)$ returns a coordinate i from each γ-contributing class. In addition, it returns $(1 \pm \frac{1}{2})$-approximate frequency of these coordinates.

Proof: There are at most $\log n$ (the total number of classes) γ-contributing classes for $\vec{a}$ and for each γ-contributing class R_t, by Lemma 5.2.9, with probability at least $1 - 2/(9 \log^2 n \log^c m)$, a coordinate in R_t will be a $\widetilde{\Omega}(\gamma)$-HeavyHitter of $F_2(\vec{a}_{i^*})$ (where $i^* = \lceil \log(n_t) \rceil$). By trying all values of $i^* \in [\log n]$, with probability at least $1 - \log n(\frac{2}{9 \log^2 n \log^c m}) \geq 1 - \frac{2}{9 \log n \log^c m}$, F_2-CONTRIBUTING algorithm outputs a coordinate from each γ-contributing class. □

5.2.3. L_0-Estimation

Norm estimation is one of the fundamental problems in the area of streaming algorithms where we are given an m-dimensional vector $\vec{a}$ which is initialized to zero and a sequence of items $p_1, \cdots, p_T$ (updates for the vector $\vec{a}$) where for each $j \in [T]$, $p_j \in [m]$ arrive in a data stream. In the well-studied task L_0-estimation (also known as Count-distinct problem), the goal is to output a $(1 \pm \varepsilon)$-estimate of the number of distinct elements (i.e.,

$L_0(\vec{a}) := |\{i \mid \vec{a}[i] \neq 0\}|$) after reading the whole stream.

Theorem 5.2.12 (L_0-estimation [13, 28, 112, 113, 32]). *Let's assume that $\vec{a}$ is an m-dimensional vector initialized to zero. Let $\mathcal{S}$ be a stream of items $p_1, \cdots, p_T$ where for each $j \in [T]$, $i_j \in [m]$. There exists a single pass algorithm that returns a $(1 \pm 1/2)$-approximation of $L_0(\vec{a})$ and uses $\widetilde{O}(1)$ space.*

5.3. Estimating Optimal Coverage of Max k-Cover

In this section, we describe the outline of our single-pass algorithm that approximates the coverage size of an optimal k-cover of $(\mathcal{U}, \mathcal{F})$ within a factor of α using $\widetilde{O}(m/\alpha^2)$ space in *arbitrary order* edge arrival streams. The input to our algorithm is $k, \alpha, n = |\mathcal{U}|$ and $m = |\mathcal{F}|$. In high level, we perform three different subroutines in parallel and show that for any given Max k-Cover instance, at least one of the subroutines estimates the optimal coverage size within the desired factor in the promised space.

Theorem 5.3.1. *For any $\alpha \in [1/\widetilde{\Theta}(\sqrt{m}), 1/\widetilde{\Theta}(1))$, there exists one pass streaming algorithm that uses $\widetilde{O}(m/\alpha^2)$ space and with probability at least $3/4$ computes the* **size an optimal coverage** *of* Max k-Cover$(\mathcal{U}, \mathcal{F})$ *within a factor of $1/\alpha$ in edge-arrival streams.*

Note that Theorem 5.3.1 together with the $O(1)$-approximation algorithms of [131, 29] that use $\widetilde{O}(m)$ space, implies that for any $\alpha \in [1/\widetilde{\Theta}(\sqrt{m}), 1 - 1/e)$, there exists a single-pass streaming algorithm that computes an α-approximation of the optimal coverage size of Max k-Cover$(\mathcal{U}, \mathcal{F})$ in $\widetilde{O}(m/\alpha^2)$ space. Later in Section 5.5, we extend our approach further to achieve a single pass algorithm that computes an $(1/\alpha)$-approximate k-cover in $\widetilde{O}(m/\alpha^2 + k)$ space.

Theorem 5.3.2. *For any $\alpha \in [1/\widetilde{\Theta}(\sqrt{m}), 1/\widetilde{\Theta}(1))$, there exists a single-pass algorithm that uses $\widetilde{O}(m/\alpha^2 + k)$ space and with probability at least $3/4$ returns an* $(1/\alpha)$**-approximate solution** *of* Max k-Cover *$(\mathcal{U}, \mathcal{F})$ in edge-arrival streams.*

Finally, we complement our upper bounds with a matching lower bound in Section 5.6.

Theorem 5.3.3. *Any single pass (possibly randomized) algorithm on edge-arrival streams that $(1/\alpha)$-approximates the optimal coverage size of* Max k-Cover *requires $\Omega(m/\alpha^2)$ space.*

As a first step, we provide a mapping from the ground set $\mathcal{U}$ to a small size set of *pseudo-elements* such that the optimal k-cover on the pseudo-elements covers a constant fraction of the pseudo-elements. This reduction is in particular useful for bounding the number of required samples in methods such as element sampling.

5.3.1. Universe Reduction

In this section we show that in order to solve Max k-Cover on *edge-arrival* streams, it suffices to solve the instances whose optimal coverage size are at least a constant fraction of $|\mathcal{U}|$. To this end, suppose that we have an algorithm $\mathcal{A}$ for Max k-Cover in edge-arrival streams with the following properties:

Definition 5.3.4 ((α, δ, η)-oracle for Max k-Cover). An algorithm $\mathcal{A}$ is an *(α, δ, η)-oracle* for Max k-Cover if it satisfies the following properties (α denotes the approximation guarantee, δ denotes the failure probability and η is the promised coverage of an optimal k-cover):

- If the optimal coverage size of Max k-Cover$(\mathcal{U}, \mathcal{F})$ is at least $|\mathcal{U}|/\eta$, then with probability at least $1 - \delta$, $\mathcal{A}$ returns a $(1/\alpha)$-approximation of the optimal coverage size.
- If $\mathcal{A}$ returns z, then an optimal solution of Max k-Cover $(\mathcal{U}, \mathcal{F})$ with high probability, has coverage at least z.

Using an (α, δ, η)-oracle of Max k-Cover, we design an $(1/\widetilde{O}(\alpha))$-approximation algorithm EstimateMaxCover for general Max k-Cover with success probability at least $1 - \widetilde{O}(\delta)$ as in Algorithm 17.

As in EstimateMaxCover, let $h : \mathcal{U} \to [z]$ be a hash function picked uniformly at random from a family of 4-wise independent hash functions mapping the ground set $\mathcal{U}$ onto *pseudo-elements* $\mathcal{V} = \{1, \cdots, z\}$. Furthermore, for a subset of elements S, we define $h(S) := \bigcup_{e \in S} h(e)$.

Lemma 5.3.5. *Let $h : \mathcal{U} \to [z]$ be a hash function picked uniformly at random from a family of 4-wise independent hash functions where $z \geq 32$. Further, let S be a subset of $\mathcal{U}$ of size at least z. Then, with probability at least $3/4$, $|h(S)| \geq z/4$.*

Proof: For any pair of elements $e_i, e_j \in S$, let $X_{i,j}$ be a random variable which is *one* if $h(e_i) = h(e_j)$ (i.e. they collide) and *zero* otherwise. Let $X := \sum_{e_i, e_j \in S} X_{i,j}$ denote the the

Algorithm 17 A single-pass streaming algorithm that uses an (α, δ, η)-oracle of Max k-Cover to compute an $(1/\widetilde{O}(\alpha))$-approximation of the optimal coverage size of Max k-Cover.

```
1: procedure EstimateMaxCover(k, α)
2:     ▷ Input: A is an (α, δ, η)-oracle of Max k-Cover
3:     ▷        S is an upper bound on the size of a γ-contributing class
4:     if kα ≥ m then
5:         return n/α ▷ trivial bound
       ▷ run for different guesses on the optimal coverage size in parallel
6:     for all z ∈ {2^i | i ∈ [log n]} do
7:         est_z ← 0
8:         for i = 1 to log(1/δ) do ▷ boost the success probability
9:             pick h_i : U → [z] from a family of 4-wise independent hash functions.
10:            for all (S, e) in the data stream do
11:                feed (S, h_i(e)) to (α, δ, η)-oracle A
12:                est_z ← max(output of A on the stream constructed by h_i, est_z)
13:    return max{est_z | est_z ≥ z/(4α)}
```

total number of collision among the elements of S under h.

First, we show that if $X \leq |S|^2/\gamma$, then $|h(S)| \geq \gamma/4$. Let's assume $h(S) = \{v_1, \cdots, v_q\}$ and let n_i denote the number of elements in S that are mapped to v_i by h. Then, the total number of collision, X, is

$$X = \sum_{i=1}^{q} \binom{n_i}{2} \geq \sum_{i=1}^{q} (\frac{n_i}{2})^2 \geq \frac{1}{4} \cdot q \cdot (\frac{|S|}{q})^2 = \frac{|S|^2}{4q}.$$

This implies that $q = |h(S)| \geq \gamma/4$. Using this observation, it only remains to show that with probability at least 3/4, $|X| \leq |S|^2/z$. Since h is selected from a family of 4-wise independent hash functions, $\{X_{i,j}\}_{e_i,e_j \in S}$ are *pairwise independent*. Hence,

$$\mathbf{E}[X] = \sum_{e_i,e_j \in S} \mathbf{E}[X_{i,j}] = \binom{|S|}{2} \cdot (\frac{1}{z}) \leq \frac{|S|^2}{2z},$$
$$\mathbf{Var}[X] = \sum_{e_i,e_j \in S} \mathbf{Var}[X_{i,j}] = \binom{|S|}{2} \cdot (\frac{1}{z} - \frac{1}{z^2}) \geq \frac{z}{8}.$$

Applying Chebyshev's inequality,

$$\Pr(X > |S|^2/z) \leq \Pr(X > \mathbf{E}[X] + \mathbf{Var}[X]) \leq 1/\mathbf{Var}[X] \leq 8/z \leq 1/4.$$

Hence, with probability at least $3/4$, $X \leq |S|^2/z$ which implies that with probability at least $3/4$, $|h(S)| \geq z/4$. □

Theorem 5.3.6. *If there exists a single pass (α, δ, η)-oracle for* Max k-Cover$(\mathcal{U}, \mathcal{F})$ *on edge-arrival streams that uses $f(m, \alpha)$ space with $\eta \geq 4$, then* EstimateMaxCover *is a $(1/O(\alpha))$-approximation algorithm for* Max k-Cover *with failure probability at most $4\delta \log n$ that uses $\widetilde{O}(f(m, \alpha))$ space on edge-arrival streams.*

Proof: Let OPT be an optimal solution of Max k-Cover$(\mathcal{U}, \mathcal{F})$. First, we show that with high probability, EstimateMaxCover returns $\Omega(|\mathcal{C}(\text{OPT})|/\alpha)$. Note that for each guess on the optimal coverage size $z \leq |\mathcal{C}(\text{OPT})|$, by Lemma 5.3.5, the probability that in none of $\log(1/\delta)$ iterations $|h(\mathcal{C}(\text{OPT}))| > \mathcal{C}(\text{OPT})/4$ is at most δ (i.e., none of the iterations preserve the optimal coverage size up to a factor of 4). Moreover, by the guarantee of (α, δ, η)-oracles for Max k-Cover, each run of $\mathcal{A}$ fails with probability at most δ. Thus, by an application of union bound, with probability at least $1 - 2\delta \log n$, est_z is at least $z/(4\alpha)$ for all $z \leq |\mathcal{C}(\text{OPT})|$. This in particular implies that the solution returned by EstimateMaxCover is at least $|\mathcal{C}(\text{OPT})|/(8\alpha)$. Moreover, since the coverage of a k-cover never increases after applying the "universe reduction" step (i.e. for each $S \subseteq \mathcal{U}$, $|h(\mathcal{C}(S))| \leq |S|$) and the estimate returned by the (α, δ, η)-oracle $\mathcal{A}$ is with high probability less than the optimal coverage size, the output of EstimateMaxCover is in $[|\mathcal{C}(\text{OPT})|/(8\alpha), |\mathcal{C}(\text{OPT})|]$ with probability at least $1 - 4\delta \log n$.

Finally, since EstimateMaxCover runs $(\log n)(\log 1/\delta)$ instances of $\mathcal{A}$ with parameter (α, δ, η) in parallel and each instance has m sets, the total space of EstimateMaxCover is $\widetilde{O}(f(m, \alpha))$. □

The *universe reduction* step basically enables us to only focus on the instances of Max k-Cover in which the optimal solution covers a constant fraction of the ground set, namely at least $|\mathcal{U}|/4$ elements. Next, in Section 5.4, we design an $\widetilde{O}(m/\alpha)$-space (α, δ, η)-oracle for Max k-Cover with $\alpha = 1/\widetilde{\Omega}(1), \eta \geq 4$ and $\delta \leq \varepsilon/\log n$ $(\varepsilon < 1)$, which together with Theorem 5.3.6 completes the proof of Theorem 5.3.1. Our (α, δ, η)-oracle for Max k-Cover performs three different subroutines in parallel that together guarantee the required properties of (α, δ, η)-oracles and only use $\widetilde{O}(m/\alpha^2)$ space:

- **Set sampling based approach.** This subroutine which provides the guarantee of (α, δ, η)-oracles when the number of *common* elements is large (see Definition 5.2.1) is an application of a "multi-layered" variant of set sampling. This subroutine is presented in Section 5.4.1.

- **HeavyHitter based approach.** We relate the problem of α-estimating/approximating of Max k-Cover to the problem of finding *contributing classes* and *heavy hitters* on properly sampled substreams (see Section 5.2.2) when the main contribution of an optimal solution of Max k-Cover is due to "large" sets. In particular, this subroutine finds an $(1/\alpha)$-estimation of the optimal coverage size when $\alpha = \Omega(k)$. This approach is presented in Section 5.4.2.

- **Element sampling based approach.** Finally, we employ element sampling together with a new sampling technique that samples a collection of sets to find a desired estimate of Max k-Cover on instances for which the main contribution to an optimal solution comes from "small" sets. Here, we also take advantage of the structure guaranteed by the multi-layered set sampling on the number of elements in different frequency levels. This subroutine is presented in Section 5.4.3.

5.4. (α, δ, η)-Oracle of Max k-Cover

In this section, we design the promised (α, δ, η)-oracle for Max k-Cover. Let OPT denote an optimal solution of Max k-Cover $(\mathcal{U}, \mathcal{F})$. As described in Definition 5.3.4, the solution returned by a (α, δ, η)-oracle with high probability, is smaller than $|\mathcal{C}(\text{OPT})|$ and if $|\mathcal{C}(\text{OPT})| \geq |\mathcal{U}|/\eta$, with probability at least $(1 - \delta)$, it outputs a value not smaller than $|\mathcal{C}(\text{OPT})|/\alpha$. The following theorem together with Theorem 5.3.6 proves Theorem 5.3.1.

Theorem 5.4.1. Oracle(α, k) *performs a single pass on edge arrival streams of the set system* $(\mathcal{U}, \mathcal{F})$ *and implements a* $(\widetilde{O}(\alpha), 1/(\log n \log^c m), \eta)$*-oracle of* Max k-Cover$(\mathcal{U}, \mathcal{F})$ *using* $\widetilde{O}(m/\alpha^2)$ *space.*

Proof: The proof follows from the guarantees of LargeCommon (Theorem 5.4.4), LargeSet (Theorem 5.4.8) and SmallSet (Theorem 5.4.22). The total space of the algorithm

is clearly $\widetilde{O}(m/\alpha^2)$ which is the space complexity of the each of subroutines invoked by ORACLE. □

To design the promised (α, δ, η)-oracle, we design different subroutines such that each guarantees the properties required by the oracle if certain conditions based on the the size/value of following notions hold.

Common elements. An important property in design of our oracle is whether there exists $\beta \leq \alpha$ such that the number of (βk)-*common* elements is relatively large (see Definition 5.2.1).

We also take advantage of another useful notion which is a property of a k-cover (though, here we only describe it for optimal k-covers).

Contribution to the optimal coverage. Given the input argument α and a parameter s as defined in Table 5.4.1, we define the following notion of *contribution* for the sets in an (optimal) k-cover.

Definition 5.4.2. For a k-cover $\text{OPT} = \{O_1, \cdots, O_k\}$, we consider an arbitrary ordering of the sets in OPT and define the *contribution* of O_i to $\mathcal{C}(\text{OPT})$ as $|O'_i|$ where $O'_i := O_i \setminus \bigcup_{1 \leq j < i} O_i$. Note that O'_i are disjoint and $|\bigcup_{i \in [k]} O'_i| = |\mathcal{C}(\text{OPT})| = z$. We (conceptually) define $\text{OPT}_{\mathsf{large}}$ to be the collection of all sets in OPT that contribute more than $z/(\mathsf{s}\alpha)$ to $\mathcal{C}(\text{OPT})$ according to O'_is; $\text{OPT}_{\mathsf{large}} = \{O_i \in \text{OPT} \mid |O'_i| \geq z/(\mathsf{s}\alpha)\}$ for $\mathsf{s} < 1$ (as in Table 5.4.1). Note that since O'_is are disjoint, $|\text{OPT}_{\mathsf{large}}| \leq \mathsf{s}\alpha$.

$\mathsf{w} = \min\{k, \alpha\}$, $\quad \mathsf{s} = \frac{9}{2500\sqrt{2\eta \log(\mathsf{s}\alpha)} \log^2(mn)} \cdot \frac{\mathsf{w}}{\alpha}$, $\quad \mathsf{t} = \frac{2500 \log^2(mn)}{\mathsf{s}}$, $\quad \mathsf{f} = 7\log(mn)$, $\quad \sigma = \frac{1}{2500 \log^2(mn)}$, $\quad \eta = 4$

Table 5.4.1: The values of parameters used in our algorithms.

Design of (δ, α, η)-oracle of max k-cover. Here we sketch a high-level outline of our (δ, α, η)-oracle for Max k-Cover (refer to Algorithm 18 for a formal description). In the following cases, $\sigma = \Omega(\frac{1}{\log^2(mn)})$ (as in Table 5.4.1).

- **If there exists a $\beta \leq \alpha$ such that $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \geq \frac{\sigma\beta|\mathcal{U}|}{\alpha}$.** In this case, by Observation 5.2.5, to approximate Max k-Cover$(\mathcal{U}, \mathcal{F})$ within a factor of $\widetilde{O}(\alpha)$, it suffices to find βk sets that cover $\mathcal{U}^{\mathsf{cmn}}_{\beta k}$ which can be done via set sampling (see Section 5.4.1).

- **$|\mathcal{C}(\text{OPT}_{\mathsf{large}})| \geq |\mathcal{C}(\text{OPT})|/2$ and $\forall \beta \leq \alpha, |\mathcal{U}^{\mathsf{cmn}}_{\beta k}| < \frac{\sigma\beta|\mathcal{U}|}{\alpha}$.** The subroutine for this case which is presented in Section 5.4.2, handles the instances of the problem in which $\mathsf{s}\alpha \geq$

Algorithm 18 An (α, δ, η)-oracle of Max k-Cover.

```
1: procedure ORACLE(k, α)
2:     ▷ for instances in which ∃β ≤ α s.t. |U^cmn_βk| ≥ σβ|U|/α
3:     SOL_cmn ← LARGECOMMON(k, α)
4:     if sα ≥ 2k then
5:         ▷ if sα ≥ 2k, then |OPT_large| ≥ |OPT|/2
6:         SOL_HH ← LARGESET(k, α, k)
7:     else
8:         ▷ for instances with sα < 2k and |OPT_large| ≥ |OPT|/2
9:         SOL_HH ← LARGESET(k, α, α)
10:        ▷ for instances with |OPT_large| < |OPT|/2
11:        SOL_small ← SMALLSET(k, α)
12:    return max(SOL_cmn, SOL_HH, SOL_small)
```

$2k$ or, $\mathsf{s}\alpha < 2k$ and there exists an optimal solution OPT such that $|\mathcal{C}(\mathrm{OPT}_{\mathsf{large}})| \geq |\mathcal{C}(\mathrm{OPT})|/2$.

Claim 5.4.3. *If* $\mathsf{s}\alpha \geq 2k$, *then* $|\mathcal{C}(\mathrm{OPT}_{\mathsf{large}})| \geq |\mathcal{C}(\mathrm{OPT})|/2$.

Proof: Consider the optimal solution OPT and ignore the sets in OPT whose contribution to the coverage is less than $|\mathcal{C}(\mathrm{OPT})|/(2k)$. Note that the survived sets belong to $\mathrm{OPT}_{\mathsf{large}}$ and their total coverage is at least $|\mathcal{C}(\mathrm{OPT})| - k \cdot \frac{|\mathcal{C}(\mathrm{OPT})|}{2k} \geq |\mathcal{C}(\mathrm{OPT})|/2$.□

- $|\mathcal{C}(\mathbf{OPT}_{\mathsf{large}})| < |\mathcal{C}(\mathbf{OPT})|/2$ **and** $\forall \beta \leq \alpha, |\mathcal{U}^{\mathsf{cmn}}_{\beta k}| < \frac{\sigma\beta|\mathcal{U}|}{\alpha}$**.** In this case, the main contribution to the coverage of OPT comes from "small" sets. This enables us to show that if we sample sets with probability $1/\alpha$, then $\widetilde{\Omega}(1/\alpha)$-fraction of sets in OPT survive and with high probability, their coverage is $\widetilde{\Omega}(|\mathcal{C}(\mathrm{OPT})|/\alpha)$. In Section 5.4.3, we show that element sampling method with some new ideas can take care of this case which can only happen when $\mathsf{s}\alpha < 2k$.

5.4.1. Multi-layered Set Sampling: $\exists \beta \leq \alpha$ s.t. $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \geq \frac{\sigma\beta|\mathcal{U}|}{\alpha}$

Here, we first guess the value of β (more precisely, a 2-approximate estimate of β) and then pick βk sets $\mathcal{F}^{\mathsf{rnd}}_{\beta}$ at random and compute their coverage in one pass using $\widetilde{O}(1)$ space. To get the desired space complexity, we use the implementation of set sampling with $O(\log(mn))$ random bits as described in Section 5.7.1.

Theorem 5.4.4. *Consider an instance* $(\mathcal{U}, \mathcal{F})$ *of* Max k-Cover. *The* LARGECOMMON *algorithm uses* $\widetilde{O}(1)$ *space and if there exists* $\beta \leq \alpha$ *such that* $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \geq \frac{\sigma\beta|\mathcal{U}|}{\alpha}$, *then with*

Algorithm 19 A (α, δ, η)-oracle of Max k-Cover that handles the case in which the number of common elements is large.

procedure LARGECOMMON((k, α))
  for all $\beta_{\mathsf{g}} \in \{2^i \mid 1 \le i \le \log \alpha\}$ in parallel **do**
    ▷ perform set sampling in one pass using $\widetilde{O}(1)$ space.
    pick $h_{\mathsf{g}} : \mathcal{F} \to [\frac{cm \log m}{\beta_{\mathsf{g}} k}]$ from $\Theta(\log(mn))$-wise independent hash functions
    let DE_{g} be a $(1 \pm 1/2)$-approximation streaming algorithm of L_0-estimation
    for all (S, e) in the data stream **do**
      if $h_{\mathsf{g}}(S) = 1$ **then**
        feed e to DE_{g} ▷ computing the coverage of $\mathcal{F}^{\mathsf{rnd}}_{\beta_{\mathsf{g}}}$
    if $\mathrm{VAL}(\mathsf{DE}_{\mathsf{g}}) \ge \sigma\beta_{\mathsf{g}}|\mathcal{U}|/(4\alpha)$ **then**
      return $2\mathrm{VAL}(\mathsf{DE}_{\mathsf{g}})/(3\beta_{\mathsf{g}})$
  return infeasible ▷ $\nexists \beta \in [\alpha]$ s.t. $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \ge \frac{\sigma\beta|\mathcal{U}|}{\alpha k}$

high probability, the algorithm returns at least $\sigma|\mathcal{U}|/(6\alpha)$. *Moreover, with high probability the output of* LARGECOMMON *is smaller than the coverage size of an optimal solution of* Max k-Cover$(\mathcal{U}, \mathcal{F})$.

Claim 5.4.5. *For each* $\beta_{\mathsf{g}} \in \{2^i \mid 1 \le i \le \log \alpha\}$, *with high probability,* $|\mathcal{F}^{\mathsf{rnd}}_{\beta_{\mathsf{g}}}| \le \beta_{\mathsf{g}} k$.

Lemma 5.4.6. *If there exists* $\beta \le \alpha$ *such that* $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \ge \sigma\beta|\mathcal{U}|/\alpha$, *then with high probability the output of* LARGECOMMON *is at least* $\sigma|\mathcal{U}|/(6\alpha)$.

Proof: Let 2^i be the smallest power of two which is larger than or equal to β; $i := \lceil \log \beta \rceil$. Consider the iteration of LARGECOMMON in which $\beta_{\mathsf{g}} = 2^i$. Since $2\beta > \beta_{\mathsf{g}} \ge \beta$ and by Observation 5.2.2,

$$|\mathcal{U}^{\mathsf{cmn}}_{\beta_{\mathsf{g}} k}| \ge |\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \ge \frac{\sigma\beta|\mathcal{U}|}{\alpha} \ge \frac{\sigma\beta_{\mathsf{g}}|\mathcal{U}|}{2\alpha}.$$

Hence, by the guarantee of existing streaming algorithms for L_0-estimation (Theorem 5.2.12) and set sampling (Lemma 5.2.3 and 5.7.7), w.h.p., $\mathrm{VAL}(\mathsf{DE}_{\mathsf{g}}) \ge \frac{1}{2} \cdot \frac{\sigma\beta_{\mathsf{g}}|\mathcal{U}|}{2\alpha} = \frac{\sigma\beta_{\mathsf{g}}|\mathcal{U}|}{4\alpha}$. Hence, the estimate returned by the algorithm which is a lower bound on the coverage of the best k sets in $\mathcal{F}^{\mathsf{rnd}}_{\beta_{\mathsf{g}}}$ (see Observation 5.2.5), w.h.p., is at least $\frac{2}{3} \cdot \frac{1}{\beta_{\mathsf{g}}} \cdot \frac{\sigma\beta_{\mathsf{g}}|\mathcal{U}|}{4\alpha} = \frac{\sigma|\mathcal{U}|}{6\alpha}$.

Moreover, it is straightforward to check that by the guarantee of the streaming algorithm for L_0-estimation (Theorem 5.2.12), the value returned by LARGECOMMON with high probability is not more than the actual coverage of the best k-cover in the collection of sampled sets using h_{g}. □

Lemma 5.4.7. *If* LargeCommon *returns* **infeasible**, *then with high probability, for all* $\beta \leq \alpha$, $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \leq \sigma\beta|\mathcal{U}|/\alpha$.

Proof: Since the algorithm returns **infeasible**, by the guarantee of the $(1\pm1/2)$-approximation algorithm for L_0-estimation (Theorem 5.2.12) and set sampling (Lemma 5.2.3), for all values of $\beta_{\mathsf{g}} \in \{2^i \mid i \leq \log\alpha\}$, with high probability,

$$|\mathcal{U}^{\mathsf{cmn}}_{\beta_{\mathsf{g}} k}| \leq |\mathcal{C}(\mathcal{F}^{\mathsf{rnd}}_{\beta_{\mathsf{g}}})| \leq 2\mathrm{VAL}(\mathsf{DE}_{\mathsf{g}}) < \sigma\beta_{\mathsf{g}}|\mathcal{U}|/(2\alpha). \tag{5.4.1}$$

Now, for any given value $\beta \leq \alpha$, consider $\beta_{\mathsf{g}} := 2^{\lceil \log\beta \rceil}$ (i.e. set β_{g} to be the smallest power of two which is larger than or equal to β). By Observation 5.2.2, $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \leq |\mathcal{U}^{\mathsf{cmn}}_{\beta_{\mathsf{g}} k}|$ which together with Eq. 5.4.1 implies that $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| \leq \sigma\beta_{\mathsf{g}}|\mathcal{U}|/(2\alpha) \leq \sigma\beta|\mathcal{U}|/\alpha$. □

Proof of Theorem 5.4.4: The guarantee on the output of LargeCommon follows from Lemma 5.4.6. Moreover, by Theorem 5.2.12, the total amount of space to compute the coverage of each collection $\mathcal{F}^{\mathsf{rnd}}_{\beta_{\mathsf{g}}}$ (via existing L_0-estimation algorithms in streams) is $\widetilde{O}(1)$. Hence, the total space to compute the coverage of all $\log\alpha$ collections considered in LargeCommon is $\widetilde{O}(1)$. □

5.4.2. Heavy Hitters and Contributing Classes: $|\mathcal{C}(\mathbf{OPT}_{\mathsf{large}})| \geq |\mathcal{C}(\mathbf{OPT}_{\mathsf{small}})|$

In this section, we show that if there exists an optimal solution OPT of Max k-Cover$(\mathcal{U},\mathcal{F})$ such that the main contribution in the coverage of OPT is due to *large* sets, which are formally defined to be the sets whose contribution to $\mathcal{C}(\mathrm{OPT})$ is at least $|\mathcal{C}(\mathrm{OPT})|/(\mathsf{s}\alpha)$, then we can approximate the optimal coverage size within a factor of $1/\widetilde{O}(\alpha)$ by detecting $\widetilde{\Omega}(\alpha^2/m)$-HeavyHitters in properly sampled substreams. Following is the main result of this section.

Theorem 5.4.8. *Consider an instance* $(\mathcal{U},\mathcal{F})$ *of* Max k-Cover. *In a single pass,* LargeSet *uses* $\widetilde{O}(m/\alpha^2)$ *space and if the optimal coverage size of the instance is* $\Omega(|\mathcal{U}|)$, *then with probability at least* $1 - 1/(\log n \log^c m)$, *it returns at least* $\widetilde{\Omega}(|\mathcal{U}|/\alpha)$. *Moreover, with high probability, the estimate returned by* LargeSet *is smaller than the optimal coverage size.*

We defer the proof of Theorem 5.4.8 to Section 5.8. In this section, we prove the same

guarantees on the performance of a simplified variant of LargeSet, LargeSetSimple, when $\mathcal{U}$ contains no "common" elements (wee will define them formally later in this section) which essentially presents the main technical ideas.

Partitioning sets into supersets. We partition sets of $\mathcal{F}$ *randomly* into $(cm \log m)/\mathsf{w}$ *supersets* $\mathcal{Q} := \{\mathcal{D}_1, \cdots, \mathcal{D}_{(cm \log m)/\mathsf{w}}\}$ via a hash function $h : \mathcal{F} \to [(cm \log m)/\mathsf{w}]$ chosen from a family of $\Theta(\log(mn))$-wise independent hash functions. More precisely, each set $S \in \mathcal{F}$ belongs to the superset $\mathcal{D}_{h(S)}$.

The parameter w denotes the desired upper bound on the maximum number of sets in a superset in $\mathcal{Q}$ defined by h and is set to $\min(\alpha, k)$. In fact, given w, we define h to be a function picked uniformly at random from a family of $\Theta(\log(mn))$-wise independent hash functions $\{\mathcal{F} \to [(cm \log m)/\mathsf{w}]\}$.

Claim 5.4.9. *With high probability, no superset in $\mathcal{Q}$ has more than w sets.*

Claim 5.4.10. *With high probability, for each $e \in \mathcal{U} \setminus \mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}}$ and $\mathcal{D} \in \mathcal{Q}$, the number of sets in $\mathcal{D}$ that contain e is at most f where $\mathsf{f} = \Theta(\log(mn))$.*

This implies that assuming $\mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} = \emptyset$, to get a $(1/\widetilde{O}(\alpha))$-approximation of Max k-Cover$(\mathcal{U}, \mathcal{F})$, it suffices to find a superset whose total size of its sets is $\widetilde{\Omega}(1/\alpha)$ times the optimal coverage size. Now, we are ready to exploit the results on F_2-heavy hitters and F_2-contributing classes mentioned in Section 5.2.2 to describe our (α, δ, η)-oracle for Max k-Cover assuming $\mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} = \emptyset$. Later in Section 5.8, we show how to remove this assumption by performing our algorithm on a set of sampled elements in $\mathcal{U}$ instead.

Partitioning supersets by their total size. First, setting $z = |\mathcal{C}(\mathrm{OPT})|$, we partition the supersets in $\mathcal{Q}$ (conceptually) according to the total size of their sets into $O(\log \alpha)$ classes

as follows:

$$\mathcal{Q}_0 = \{\mathcal{D} \mid \sum_{S\in\mathcal{D}} |S| \geq z/2\},$$
$$\mathcal{Q}_i = \{\mathcal{D} \mid z/2^{i+1} \leq \sum_{S\in\mathcal{D}} |S| < z/2^i\}, \quad \forall i \in [1, \log(\alpha))$$
$$\mathcal{Q}_{\mathsf{small}} = \{\mathcal{D} \mid \sum_{S\in\mathcal{D}} |S| < z/\alpha\}.$$

Further, let n_i denote the number of supersets in $\mathcal{Q}_i$; $n_i = |\mathcal{Q}_i|$. Next, we define the vector $\vec{v}$ of size $(cm\log m)/\mathsf{w}$ such that $\vec{v}[i] = \sum_{S\in\mathcal{D}_i} |S|$ denotes the total size of the sets in $\mathcal{D}_i$. In the following, we show that a subset of supersets with large total size form an $\widetilde{\Omega}(m/\alpha^2)$-contributing class of $F_2(\vec{v})$ and any superset in this $\widetilde{\Omega}(m/\alpha^2)$-contributing class is a $(1/\alpha)$-*approximate* k-cover of $(\mathcal{U}, \mathcal{F})$.

We consider the following two cases depending on whether the coordinates corresponding to small supersets, $\mathcal{Q}_{\mathsf{small}}$, contribute to $F_2(\vec{v})$; $F_2(\vec{v}_{\mathsf{small}}) \geq F_2(\vec{v})/2$ where $\vec{v}_{\mathsf{small}}$ denotes the vector $\vec{v}$ restricted to the coordinates corresponding to supersets in $\mathcal{Q}_{\mathsf{small}}$.

Case 1: Supersets with total size less than z/α contribute to $F_2(\vec{v})$. This implies that $F_2(\vec{v}) \leq 2F_2(\vec{v}_{\mathsf{small}}) \leq 2 \cdot (\frac{cm\log m}{\mathsf{w}}) \cdot (\frac{z}{\alpha})^2 = \frac{2cm\log m}{\mathsf{w}} \cdot \frac{z^2}{\alpha^2}$.

Claim 5.4.11. *If $F_2(\vec{v}_{\mathsf{small}}) \geq F_2(\vec{v})/2$, then there exists an $\widetilde{\Omega}(\alpha^2/m)$-contributing class $\mathcal{Q}_{i^*}$ of $F_2(\vec{v})$ for an index $i^* < \log(\mathsf{s}\alpha)$.*

Proof: Since each set in $\mathrm{OPT}_{\mathsf{large}}$ has contribution at least $z/(\mathsf{s}\alpha)$ to the optimal coverage, each set of $\mathrm{OPT}_{\mathsf{large}}$ lands in one of $\mathcal{Q}_0, \cdots, \mathcal{Q}_{\log(\mathsf{s}\alpha)-1}$. Moreover, since $\mathrm{OPT}_{\mathsf{large}}$ has coverage at least $z/2$,

$$\sum_{i=0}^{\log(\mathsf{s}\alpha)-1} n_i \cdot \frac{z}{2^i} \geq \sum_{O_i\in\mathrm{OPT}_{\mathsf{large}}} |O_i| \geq |\mathcal{C}(\mathrm{OPT}_{\mathsf{large}})| \geq z/2,$$

which implies that there exists an index $i^* < \log(\mathsf{s}\alpha)$ such that $n_{i^*} \geq 2^{i^*}/(2\log(\mathsf{s}\alpha))$. Hence,

$\mathcal{Q}_{i^*}$ is an $\widetilde{\Omega}(\frac{\alpha^2}{m})$-contributing class of $F_2(\vec{v})$:

$$|\mathcal{Q}_{i^*}| \cdot (\frac{z}{2^{i^*+1}})^2 \geq \frac{2^{i^*}}{2\log(\mathsf{s}\alpha)} \cdot \frac{z^2}{4(2^{i^*})^2} = \frac{\mathsf{w}}{2cm\log m} \cdot \alpha^2 \cdot \frac{1}{2^{i^*+3}\log(\mathsf{s}\alpha)} \cdot F_2(\vec{v}) \qquad \triangleright\ (2^{i^*+1} \leq \mathsf{s}\alpha)$$
$$\geq (\frac{\mathsf{w}}{\mathsf{s}\alpha} \cdot \frac{1}{8c\log(\mathsf{s}\alpha)\log m}) \cdot \frac{\alpha^2}{m} \cdot F_2(\vec{v})$$

More formally, since $\frac{\mathsf{w}}{\mathsf{s}\alpha} = \widetilde{\Omega}(1)$ (see Table 5.4.1), $\mathcal{Q}_{i^*}$ is a ϕ_1-contributing class of $F_2(\vec{v})$ where

$$\phi_1 = (\frac{\mathsf{w}}{\mathsf{s}\alpha} \cdot \frac{1}{8c\log(\mathsf{s}\alpha)\log m}) \cdot \frac{\alpha^2}{m} = \widetilde{\Omega}(\frac{\alpha^2}{m}). \tag{5.4.2}$$

□

Hence, by Theorem 5.2.11, a superset of total size at least $\frac{2}{3} \cdot \frac{1}{2} \cdot \frac{z}{\mathsf{s}\alpha}$ will be identified by the subroutine F_2-CONTRIBUTING$(\phi_1, \mathsf{s}\alpha)$ using $\widetilde{O}(m/\alpha^2)$ space.

Remark 5.4.12. Recall that in order to find a coordinate in a ϕ-contributing class R_{t^*}, F_2-CONTRIBUTING subsamples the stream proportional to $1/|R_{t^*}|$ (so that only $O(1)$ coordinates of R_{t^*} survive) and then with high probability any survived coordinate of R_{t^*} becomes a $\widetilde{\Omega}(\phi)$-HeavyHitter in the sampled substream. However, here we show that there exists a ϕ-contributing class R_{t^*} whose intersection with $\mathrm{OPT}_{\mathsf{large}}$ is a ϕ-contributing class of coordinates too. Hence, it suffices to only search for a coordinate in a ϕ-contributing class of size at most $|\mathrm{OPT}_{\mathsf{large}}| \leq \mathsf{s}\alpha$.

Case 2. Supersets with coverage less than z/α do not contribute to $F_2(\vec{v})$.

Claim 5.4.13. *If $F_2(\vec{v}_{\mathsf{small}}) < F_2(\vec{v})/2$, then there exists an $\widetilde{\Omega}(1)$-contributing class $\mathcal{Q}_{i^*}$ of $F_2(\vec{v})$ for an index $i^* < \log\alpha$.*

Proof: In this case, since supersets in $\mathcal{Q}_{\mathsf{small}}$ are not contributing, there exists an index $i^* < \log(\alpha)$ (note that we consider all classes $\mathcal{Q}_0, \cdots, \mathcal{Q}_{\log\alpha - 1}$ in this case) such that

$$n_{i^*} \cdot (z/2^{i^*})^2 \geq F_2(\vec{v})/(2\log\alpha);$$

Algorithm 20 An (α, δ, η)-oracle of Max k-Cover that handles the case in which the majority of the coverage in an optimal solution is by the sets whose coverage contributions are at least $1/(\mathsf{s}\alpha)$ fraction of the optimal coverage size.

procedure LARGESETSIMPLE($(\mathcal{V}, \mathsf{w}, \mathrm{thr}_1, \mathrm{thr}_2)$)
  ▷ **Input:** w is an upper bound on the desired number of sets in a superset
  ▷ **Parameters:** $\phi_1 = \widetilde{\Omega}(\alpha^2/m)$ and $\phi_2 = \widetilde{\Omega}(1)$
  let $\mathsf{Cntr}_{\mathsf{small}}$ be an instance of F_2-CONTRIBUTING$(\phi_1, \mathsf{s}\alpha)$ ▷ for Case 1
  let $\mathsf{Cntr}_{\mathsf{large}}$ be an instance of F_2-CONTRIBUTING$(\phi_2, \frac{cm\log m}{\mathsf{w}})$ ▷ for Case 2
  pick $h : \mathcal{F} \to [(cm\log m)/\mathsf{w}]$ from $\Theta(\log(mn))$-wise independent hash functions
  for all (S, e) in the data stream **do**
    if $e \in \mathcal{V}$ **then**
      feed $h(S)$ to both $\mathsf{Cntr}_{\mathsf{small}}$ and $\mathsf{Cntr}_{\mathsf{large}}$
  ▷ $\mathsf{output}(\mathsf{Cntr})$ contains coordinates along with $(1 \pm 1/2)$-estimate of frequencies
  if there exists $i \in \mathsf{output}(\mathsf{Cntr}_{\mathsf{small}})$ such that $\tilde{v}_i \geq \frac{1}{2} \cdot \mathrm{thr}_1$ **then**
    return $2\tilde{v}_i/(3\mathsf{f})$
  if there exists $i \in \mathsf{output}(\mathsf{Cntr}_{\mathsf{large}})$ such that $\tilde{v}_i \geq \frac{1}{2} \cdot \mathrm{thr}_2$ **then**
    return $2\tilde{v}_i/(3\mathsf{f})$
  return infeasible

in other words, $\mathcal{Q}_{i^*}$ is a ϕ_2-contributing class of $F_2(\vec{v})$ where $\phi_2 = (\frac{1}{2\log\alpha})$. □

Note that, by Theorem 5.2.11, a superset of total size at least $\frac{2}{3} \cdot \frac{1}{2} \cdot \frac{z}{\alpha}$ will be identified by the subroutine F_2-CONTRIBUTING$(\phi_2, \frac{cm\log m}{\mathsf{w}})$ using $\widetilde{O}(1)$ space.

Lemma 5.4.14. *If $|\mathcal{C}(\mathrm{OPT})| \geq \frac{|\mathcal{U}|}{\eta}$, then with probability at least $1 - 1/(3\log n \log^c m)$, the estimate returned by* LARGESETSIMPLE *with parameters $(\mathcal{V} = \mathcal{U}, \mathsf{w}, \mathrm{thr}_1 = \frac{|\mathcal{U}|}{\eta\mathsf{s}\alpha}, \mathrm{thr}_2 = \frac{|\mathcal{U}|}{\eta\alpha})$ has coverage at least $|\mathcal{U}|/(3\mathsf{f}\eta\alpha) = \widetilde{\Omega}(|\mathcal{U}|/\alpha)$.*

Proof: With probability at least $1 - 2/(9\log n \log^c m)$, the algorithm returns a superset whose total size is at least $\frac{2}{3} \cdot \frac{|\mathcal{U}|}{2\eta\alpha} \geq \frac{|\mathcal{U}|}{3\eta\alpha}$ (in fact, if it is in Case 1, then the estimate is at least $\frac{|\mathcal{U}|}{3\mathsf{s}\alpha\eta}$). Then, by Claim 5.4.10 and assumption $\mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} = \emptyset$, the coverage of the reported superset is at least $\frac{1}{\mathsf{f}} \cdot \frac{|\mathcal{U}|}{3\eta\alpha}$. □

Lemma 5.4.15. *The amount of space used by* LARGESETSIMPLE *is $\widetilde{O}(m/\alpha^2)$.*

Proof: By Theorem 5.2.11, the amount of space to perform $\mathsf{Cntr}_{\mathsf{small}}$ and $\mathsf{Cntr}_{\mathsf{large}}$ as defined in Algorithm 25 is respectively $\widetilde{O}(1/\phi_1) = \widetilde{O}(m/\alpha^2)$ and $\widetilde{O}(1/\phi_2) = \widetilde{O}(1)$. □

5.4.3. Element Sampling: $|\mathcal{C}(\mathbf{OPT}_{\mathsf{large}})| < |\mathcal{C}(\mathbf{OPT}_{\mathsf{small}})|$

Here we design an (α, δ, η)-oracle of Max k-Cover with the desired parameters for the case in which $|\mathcal{C}(\mathrm{OPT}_{\mathsf{small}})| > |\mathcal{C}(\mathrm{OPT}_{\mathsf{large}})|$. Intuitively speaking, in this case, after sampling each set in $\mathcal{F}$ with probability $\widetilde{\Theta}(1/\alpha)$ still we can find $O(k/\alpha)$ sets whose coverage is at least $\widetilde{\Omega}(|\mathcal{C}(\mathrm{OPT})|/\alpha)$. As proved in Claim 5.4.3, if $\alpha = \widetilde{\Omega}(k)$, then the main contribution to the coverage of OPT is due to $\mathrm{OPT}_{\mathsf{large}}$ and we can $(1/\alpha)$-approximate the optimal coverage size by LargeSet. Hence, in this section we assume that $\alpha = \widetilde{O}(k)$. Moreover, throughout this section we assume that for all $\beta \leq \alpha$, $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| < \frac{\sigma\beta|\mathcal{U}|}{\alpha}$ (otherwise, our multi-layered set sampling approach described in Section 5.4.1 returns a $(1/\alpha)$-approximation of the optimal coverage size).

Lemma 5.4.16. *Consider an instance of* Max k-Cover *$(\mathcal{U}, \mathcal{F})$. Suppose that $\mathcal{D}$ is a collection of k* disjoint *sets with coverage z such that no $S \in \mathcal{D}$ has size more than $z/(\mathsf{s}\alpha)$ where $\mathsf{s} < 1$ and $\mathsf{s} = \widetilde{\Omega}(1)$. Let's assume that $\mathcal{D}_{\mathsf{smp}} := \mathcal{D} \cap \mathcal{M}$ where each $S \in \mathcal{F}$ survives in $\mathcal{M}$ with probability $c/(\mathsf{s}\alpha)$ where $c > 1$ is a fixed constant. Then, with probability at least $(1 - 6/c)$, $\mathcal{D}_{\mathsf{smp}}$ has size at most $(2ck)/(\mathsf{s}\alpha)$ and covers at least $(cz)/(2\mathsf{s}\alpha)$ elements.*

Proof: Let $\mathcal{D} = \{S'_1, \cdots, S'_k\}$ and for each i, let X_i to be the random variable corresponding to S'_i such that $X_i = |S'_i|$ if $S'_i \in \mathcal{D}_{\mathsf{smp}}$ and zero otherwise.

Claim 5.4.17. $\mathbf{E}[X_i] = \frac{c}{\mathsf{s}\alpha} \cdot |S'_i|$ *and* $\mathbf{Var}[X_i] \leq \frac{c}{\mathsf{s}\alpha} \cdot |S'_i|^2$.

Next, we define $X := X_1 + \cdots + X_k$. Note that $\mathbf{E}[X] = (cz)/(\mathsf{s}\alpha)$ and, by the pairwise independence of X_is and the assumption that $|S'_i| \leq z/(\mathsf{s}\alpha)$,

$$\begin{aligned}\mathbf{Var}[X] &\leq \frac{c}{\mathsf{s}\alpha} \cdot \sum_{i=1}^{k} |S'_i|^2 \\ &\leq \frac{c}{\mathsf{s}\alpha} \cdot \mathsf{s}\alpha \cdot (\frac{z}{\mathsf{s}\alpha})^2 = c \cdot (\frac{z}{\mathsf{s}\alpha})^2,\end{aligned}$$

Finally, applying Chebyshev inequality,

$$\Pr[X < \frac{cz}{2\mathsf{s}\alpha}] = \Pr[X < (\frac{cz}{\mathsf{s}\alpha} - \frac{\sqrt{c}}{2} \cdot (\frac{\sqrt{c}}{\mathsf{s}\alpha} \cdot z)] < 4/c.$$

Hence, with probability at least $1 - 4/c$, $\mathcal{D}_{\mathsf{smp}}$ covers at least $(cz)/(2\mathsf{s}\alpha)$ elements.

Next, we show that with probability at least $1 - 2/c$, $0 < |\mathcal{D}_{\mathsf{smp}}| < (2ck)/(\mathsf{s}\alpha)$. For each i, let Y_i denote the random variable corresponding to S_i which is equal to one if $S_i \in \mathcal{D}_{\mathsf{smp}}$ and zero otherwise.

Claim 5.4.18. $\mathbf{E}[Y_i] = (c/\mathsf{s}\alpha)$ *and* $\mathbf{Var}[Y_i] \leq (c/\mathsf{s}\alpha)$.

We define $Y = Y_1 + \cdots + Y_\ell$ which denotes the size of $\mathcal{D}_{\mathsf{smp}}$. Then by pairwise independence of Y_is, $\mathbf{E}[Y] = (ck)/(\mathsf{s}\alpha)$ and $\mathbf{Var}[Y] \leq (ck)/(\mathsf{s}\alpha)$. Applying Chebyshev inequality ($\Pr[|Y - \mathbf{E}[Y]| \geq t\mathbf{Var}[Y]] \leq 1/(t^2\mathbf{Var}[Y])$), with probability at least $1 - (\mathsf{s}\alpha)/(ck) \geq 1 - 2/c$ (since in this case, $\alpha \leq 2k/\mathsf{s}$), $0 < |\mathcal{D}_{\mathsf{smp}}| < (2ck)/(\mathsf{s}\alpha)$.

Hence, with probability at least $(1 - 6/c)$, $\mathcal{D}_{\mathsf{smp}}$ is a subset of size at most $(2ck)/(\mathsf{s}\alpha)$ that covers at least $(cz)/(2\mathsf{s}\alpha)$ elements. □

Corollary 5.4.19. *Consider an instance* $(\mathcal{U}, \mathcal{F})$ *of* Max k-Cover *and let* OPT *be an optimal solution of this instance such that* $|\mathcal{C}(\mathrm{OPT}_{\mathsf{small}})| \geq \frac{1}{2} \cdot |\mathcal{C}(\mathrm{OPT})| \geq |\mathcal{U}|/(2\eta)$. *Moreover, let* $\mathcal{M} \subset \mathcal{F}$ *be a collection of* $\widetilde{O}(|\mathcal{F}|/\alpha)$ *pairwise independent sets picked uniformly at random such that each* $S \in \mathcal{F}$ *belongs to* $\mathcal{M}$ *with probability* $18/(\mathsf{s}\alpha)$. *With probability at least* $2/3$, Max $(\frac{36k}{\mathsf{s}\alpha})$-Cover$(\mathcal{U}, \mathcal{M})$ *has an optimal solution with coverage at least* $9|\mathcal{U}|/(\mathsf{s}\alpha\eta)$.

Proof: By definition of $\mathrm{OPT}_{\mathsf{small}}$, for each $O \in \mathrm{OPT}_{\mathsf{small}}$, the contribution of O to OPT (i.e. O') is at most $z/(\mathsf{s}\alpha)$ (see Definition 5.4.2). Then, the result follows from an application of Lemma 5.4.16 on collection $\mathcal{D} := \{O' \mid O \in \mathrm{OPT}_{\mathsf{small}}\}$ by setting $c = 18$ and $z = |\mathrm{OPT}_{\mathsf{small}}| \geq |\mathcal{U}|/(2\eta)$. □

Next, we show that we can perform "element sampling" and find an $\widetilde{O}(1)$-approximation of Max $(\frac{36k}{\mathsf{s}\alpha})$-Cover of the specified instance in Corollary 5.4.19, $(\mathcal{U}, \mathcal{M})$, in one pass and using $\widetilde{O}(m/\alpha^2)$ space. To this end, first we compute the space complexity of $(\mathcal{L}, \mathcal{F})$ where $\mathcal{L} \subseteq \mathcal{U}$ is a subset of size $\widetilde{O}(k)$ which is picked by element sampling.

Lemma 5.4.20. *Suppose that an optimal solution of* Max $(\frac{k}{\alpha})$-Cover$(\mathcal{U}, \mathcal{F})$ *has coverage* $|\mathcal{U}|/\gamma = \widetilde{\Omega}(|\mathcal{U}|/\alpha)$. *Let* $\mathcal{L} \subset \mathcal{U}$ *be a collection of elements of size* $\widetilde{O}(\frac{k\gamma}{\alpha})$ *picked uniformly at random. With high probability, the total amount of space to store the set system* $(\mathcal{L}, \mathcal{F})$ *is* $\widetilde{O}(m/\alpha)$.

Proof: Recall that $(\mathcal{U}, \mathcal{F})$ has the property that for all $\beta \leq \alpha$, $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| < \sigma\beta|\mathcal{U}|/\alpha$ (otherwise,

the result of Section 5.4.1 can be applied). Next, we (conceptually) partition the elements in $\mathcal{U}$ into $\log\alpha + 1$ groups as follows:

$$\mathcal{W}_0 = \mathcal{U} \setminus \mathcal{U}^{\mathsf{cmn}}_{\alpha k}$$
$$\mathcal{W}_i = \mathcal{U}^{\mathsf{cmn}}_{(\alpha/2^{i-1})k} \setminus \mathcal{U}^{\mathsf{cmn}}_{(\alpha/2^i)k} \quad \forall i \in [\log\alpha].$$

Note that $|\mathcal{W}_0| \leq |\mathcal{U}|$ and for each $i \in [\log\alpha]$, $|\mathcal{W}_i| \leq \sigma|\mathcal{U}|/2^{i-1}$. Since each element $e \in \mathcal{U}$ survives in $\mathcal{L}$ with probability $\widetilde{O}(\frac{k\gamma}{\alpha|\mathcal{U}|})$, w.h.p., for each $i \in [\log\alpha]$, $|\mathcal{W}_i \cap \mathcal{L}| = \widetilde{O}(1 + \frac{\sigma\gamma k}{\alpha 2^{i-1}})$. Furthermore, since each element in $\mathcal{W}_i$ appears in at most $\widetilde{O}(\frac{2^i m}{\alpha k})$ sets in $\mathcal{F}$, the total amount of space required to store $(\mathcal{L}, \mathcal{F})$ is at most

$$\begin{aligned}\sum_{i=0}^{\log\alpha} |\mathcal{W}_i \cap \mathcal{L}| \cdot \max_{e\in\mathcal{W}_i} \mathsf{freq}(e) &= \widetilde{O}(\frac{k\gamma}{\alpha}) \cdot \widetilde{O}(\frac{m}{\alpha k}) + \sum_{i=1}^{\log\alpha} \widetilde{O}(1 + \frac{\sigma\gamma k}{\alpha 2^{i-1}}) \cdot \widetilde{O}(\frac{2^i m}{\alpha k}) \\ &= \widetilde{O}(\frac{\gamma m}{\alpha^2}) + \sum_{i=1}^{\log\alpha} \widetilde{O}(\frac{2^i m}{\alpha k} + \frac{\sigma\gamma m}{\alpha^2}) \\ &= \widetilde{O}(m/\alpha).\end{aligned}$$

□

Next, we show that after subsampling the sets by a factor of $\widetilde{\Theta}(1/\alpha)$, we can save another factor of $\widetilde{\Omega}(\alpha)$ in the space complexity; in other words, $(\mathcal{L}, \mathcal{M})$ uses $\widetilde{O}(m/\alpha^2)$ space. Note that since $k\alpha$ may be as large as $\widetilde{\Omega}(m)$ we cannot hope to show directly that each element in $\mathcal{W}_i$ appears in at most $\widetilde{O}(m/(2^i\alpha k))$. However, we can show that the total size of the intersection of all sets in $\mathcal{M}$ with $\mathcal{L}$ is $\widetilde{O}(m/\alpha^2)$ using the properties of the max cover instance.

Lemma 5.4.21. *Suppose that an optimal solution of* Max $(\frac{k}{\alpha})$-Cover$(\mathcal{U}, \mathcal{F})$ *has coverage* $|\mathcal{U}|/\gamma = \widetilde{\Omega}(|\mathcal{U}|/\alpha)$. *Let* $\mathcal{L} \subset \mathcal{U}$ *be a collection of elements of size* $\widetilde{O}(\frac{k\gamma}{\alpha})$ *picked uniformly at random and let* $\mathcal{M} \subset \mathcal{F}$ *be a collection of sets of size* $\widetilde{O}(m/\alpha)$ *picked uniformly at random. With high probability, the total amount of space required to store the set system* $(\mathcal{L}, \mathcal{M})$ *is* $\widetilde{O}(m/\alpha^2)$.

Proof: First note that since an optimal $(\frac{k}{\alpha})$-cover of $(\mathcal{U}, \mathcal{F})$ has coverage $|\mathcal{U}|/\gamma$, with high probability, for each set $S \in \mathcal{M}$, $|S \cap \mathcal{L}| = \widetilde{O}(k/\alpha)$. Moreover, by Lemma 5.4.20, the size

Algorithm 21 A single pass streaming algorithm that estimates the optimal coverage size of Max k-Cover$(\mathcal{U},\mathcal{F})$ within a factor of $1/\widetilde{O}(\alpha)$ in $\widetilde{O}(m/\alpha^2)$ space. To return a $\widetilde{O}(\frac{k}{\alpha})$-cover that achieves the computed estimate it suffices to change the line marked by **(⋆⋆)** to return the corresponding $\widetilde{O}(\frac{k}{\alpha})$-cover as well (see Section 5.5).

1: **procedure** SmallSet$((k,\alpha))$
2: **for all** $\gamma_g \in \{2^{-i} \mid i \in [\log \alpha]\}$ in parallel **do**
3: ▷ γ_g is an estimate of the optimal coverage of $\widetilde{O}(\frac{k}{\alpha})$-cover
4: **for** $\log n$ times **in parallel do**
5: $\mathcal{M} \leftarrow$ uniformly selected samples of size $\widetilde{\Theta}(m/\alpha)$ from $\mathcal{F}$
6: $\mathcal{L} \leftarrow$ uniformly selected samples of size $\widetilde{\Theta}(\gamma_g \cdot (\frac{k}{\alpha}))$ from $\mathcal{U}$
7: **initialize** $\mathbf{S}(\mathcal{L},\mathcal{M})$ to be an empty set ▷ $\mathbf{S}(\mathcal{L},\mathcal{M})$ stores $(\mathcal{L},\mathcal{M})$
8: **for all** (S,e) in the data stream **do**
9: **if** $S \in \mathcal{M}$ and $e \in \mathcal{L}_i$ **then**
10: **add** (S,e) to $\mathbf{S}(\mathcal{L},\mathcal{M})$
11: **if** $\mathbf{S}(\mathcal{L},\mathcal{M}) > \widetilde{O}(m/\alpha^2)$ **then**
12: **terminate**
13: $\text{SOL}_{\gamma_g} \leftarrow \max_{\mathcal{L},\mathcal{M}}\{O(1)\text{-approx solution of Max } (\frac{36k}{s\alpha})\text{-Cover}(\mathbf{S}(\mathcal{L},\mathcal{M}))\}$ **(⋆⋆)**
14: **return** $\max_{\gamma_g}\{(\frac{|\mathcal{U}|}{\gamma_g(k/\alpha)} \cdot \text{SOL}_{\gamma_g}) \mid \text{SOL}_{\gamma_g} = \widetilde{\Omega}(k/\alpha)\}$

of the intersection of all sets in $\mathcal{F}$ with $\mathcal{L}$ is $\widetilde{O}(m/\alpha)$. Next, we (conceptually) partition the sets of $\mathcal{F}$ into $O(\log k)$ groups based on their intersection size with $\mathcal{L}$ as follows ($c = \widetilde{O}(1)$):

$$\mathcal{Q}_i = \{S \in \mathcal{F} \mid \frac{1}{2^i} \cdot \frac{ck}{\alpha} \leq |S \cap \mathcal{L}| < \frac{1}{2^{i-1}} \cdot \frac{ck}{\alpha}\}, \quad \forall 1 \leq i \leq \log k$$

Since the total size of the intersection of all sets with the sampled set $\mathcal{L}$ is w.h.p. $\widetilde{O}(m/\alpha)$, for each $i \leq \log k$, $|\mathcal{Q}_i| \leq \frac{\widetilde{O}(m/\alpha)}{(ck)/(2^i\alpha)} = \widetilde{O}(\frac{2^i \cdot m}{k})$. Since we have the assumption that $\frac{m}{k\alpha} \geq 1$ (we took care of the case $m < k\alpha$ in the first line of EstimateMaxCover separately), w.h.p., for each $\mathcal{Q}_i$, $|\mathcal{Q}_i \cap \mathcal{M}| = \widetilde{O}(\frac{2^i m}{k\alpha})$. Hence, the total amount of space to store $(\mathcal{L},\mathcal{M})$ is at most

$$\sum_{i=1}^{\log k} \frac{1}{2^{i-1}} \cdot \frac{ck}{\alpha} \cdot \widetilde{O}(\frac{2^i m}{k\alpha}) = \widetilde{O}(\frac{m}{\alpha^2}). \qquad \square$$

Theorem 5.4.22. *If $|\mathcal{C}(\text{OPT}_{\mathsf{small}})| \geq |\mathcal{U}|/(2\eta)$ and for all $\beta \leq \alpha$, $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| < \frac{\sigma\beta|\mathcal{U}|}{\alpha}$, then with high probability,* SmallSet *outputs a $(1/\widetilde{O}(\alpha))$-approximate of the optimal coverage size of* Max k-Cover$(\mathcal{U},\mathcal{F})$ *in $\widetilde{O}(m/\alpha^2)$ space.*

Proof: By Corollary 5.4.19, for any sampled collection of sets $\mathcal{M}$ of size $\widetilde{\Theta}(m/\alpha)$ (as in

SmallSet), with probability at least 2/3, there exists a subset of size at most $(\frac{36k}{\mathsf{s}\alpha})$ in $\mathcal{M}$ whose coverage is $9|\mathcal{U}|/(\mathsf{s}\alpha\eta) = |\mathcal{U}|/\gamma$. Moreover, by the guarantee of element sampling, Lemma 5.2.4, when $\gamma/2 < \gamma_g \leq \gamma$, an $O(1)$-approximate solution of $\mathsf{Max}\, k\text{-}\mathsf{Cover}(\mathcal{L}, \mathcal{M})$ with high probability is an $O(1)$-approximate solution of $\mathsf{Max}\, k\text{-}\mathsf{Cover}(\mathcal{U}, \mathcal{M})$. Hence, with probability $1 - n^{-1}$, in at least one of the $(\log n)$ instances with the desired γ_{g}, $\textsc{sol}_{\gamma_{\mathsf{g}}}$ has coverage $\widetilde{\Omega}(\frac{|\mathcal{U}|}{\gamma} \cdot \frac{\gamma_{\mathsf{g}}(k/\alpha)}{|\mathcal{U}|}) = \widetilde{\Omega}(k/\alpha)$ over the sampled elements $\mathcal{L}$. Note that we need to scale $\textsc{sol}_{\gamma_{\mathsf{g}}}$ by a factor of $\widetilde{\Theta}(|\mathcal{U}|/(\gamma_{\mathsf{g}} \cdot \frac{k}{\alpha}))$ to reflects its coverage on $\mathcal{U}$.

Further, by Lemma 5.4.21, the amount of space required to store each $(\mathcal{L}, \mathcal{M})$ with high probability is $\widetilde{O}(m/\alpha^2)$ and since SmallSet stores $\widetilde{O}(1)$ different instances $(\mathcal{L}, \mathcal{F})$, the total space of the algorithm is $\widetilde{O}(m/\alpha^2)$. □

To complete the SmallSet is indeed an (α, δ, η)-oracle with the desired parameters, we need to show that it never overestimates the optimal coverage size.

Lemma 5.4.23. *The output of* SmallSet *with high probability is not larger than the optimal coverage size of* $\mathsf{Max}\, k\text{-}\mathsf{Cover}(\mathcal{U}, \mathcal{F})$.

Proof: Note that if the optimal coverage size $\mathsf{Max}\, \widetilde{O}(\frac{k}{\alpha})\text{-}\mathsf{Cover}(\mathcal{L}, \mathcal{M})$ is less than $\widetilde{\Omega}(|\mathcal{U}|/(\alpha\gamma_{\mathsf{g}}))$, then with high probability in none of the $\log n$ iterations $\textsc{sol}_{\gamma_{\mathsf{g}}} = \widetilde{\Omega}(k/\alpha)$. Hence, SmallSet with high probability does not overestimate the size of an optimal $\widetilde{O}(k/\alpha)$-cover in $(\mathcal{U}, \mathcal{F})$.□

5.5. Approximating Max k-Cover Problem in Edge Arrival Streams

In this section, we show that we can modify $\textsc{Oracle}(k, \alpha)$ and the subroutines within it slightly to *report* a $(1/\alpha)$-approximate solution as well. Currently, in the algorithms for estimating maximum coverage size, we only maintain the coverage of a collection of sets. To provide the actual $(1/\alpha)$-approximate solution, we also need to report the indices of the sets that belong to the collection.

Reporting a k-cover in LargeCommon. In this subroutine, the coverage of different covers with possibly size more than k are computed. For the task of estimating the maximum coverage size, this does not hurt as long as we scale the coverage by a factor that bounds how much larger the size of cover is compared to k. However, in the task of reporting an

Algorithm 22 A single pass streaming algorithm that reports a $(1/\alpha)$-approximate solution of Max k-Cover$(\mathcal{U}, \mathcal{F})$ in $\widetilde{O}(\beta + k)$ space when the set of common elements is large.

procedure ReportLargeCommon$((k, \alpha, \beta))$
  for all $\beta_{\mathsf{g}} \in \{2^i \mid 0 \leq i \leq \log \beta\}$ in parallel **do**
    ▷ perform set sampling in one pass using $\widetilde{O}(1)$ space.
    pick $h : \mathcal{F} \to [\frac{cm \log m}{(\beta_{\mathsf{g}} k)}]$ from $\Theta(\log(mn))$-wise independent hash functions
    pick $h_2 : \mathcal{F} \to [\beta_{\mathsf{g}} \log m]$ from $\Theta(\log(mn))$-wise independent hash functions
    let $\{\mathsf{DE}_{\mathsf{g},i} \mid i \in [\beta_{\mathsf{g}}]\}$ be β_{g} instances of $(1/2)$-approximation of L_0-estimation
    for all (S, e) in the data stream **do**
      if $h(S) = 1$ **then**
        feed e to $\mathsf{DE}_{\mathsf{g},h_2(S)}$ ▷ computing $\mathcal{C}(\mathcal{F}^{\mathsf{rnd}}_{\beta_{\mathsf{g}},i})$
    if $\exists i^* \in [\log \beta]$ s.t. $\text{VAL}(\mathsf{DE}_{\mathsf{g},i^*}) \geq \sigma|\mathcal{U}|/(4\alpha)$ **then**
      return $2\text{VAL}(\mathsf{DE}_{\mathsf{g}})/3$ and $\{S \mid h(S) = 1 \text{ and } h_2(S) = i^*\}$
  return infeasible ▷ $\exists$ no $\beta \in [\alpha]$ s.t. $|\mathcal{U}^{\mathsf{cmn}}_{\beta k}| = \widetilde{\Omega}(\frac{3\sigma\beta|\mathcal{U}|}{\alpha k})$

actual k-cover, it becomes crucial to have at most k sets. To this end, instead of computing the coverage of a randomly selected $(\beta_{\mathsf{g}} k)$-cover, we partition the collection into $\widetilde{O}(\beta_{\mathsf{g}})$ sub-collections each of size at most k and by Observation 5.2.5, at least one of them achieves the reported coverage size up to polylogarithmic factors. Moreover, we need to maintain the coverage of $\widetilde{O}(\beta_{\mathsf{g}}) = \widetilde{\Theta}(\beta)$ covers which add an extra β_{g} term in the space complexity (see Figure 22).

Lemma 5.5.1. *The space complexity of* ReportLargeCommon(k, α, β) *is* $\widetilde{O}(\beta + k)$.

Proof: Similarly to the space analysis of LargeCommon (Theorem 5.4.4) and by employing an existing $(1 \pm 1/2)$-approximation algorithm of L_0-estimation that uses $\widetilde{O}(1)$ space (Theorem 5.2.12), the total amount of space used in the algorithm to maintain $\mathsf{DE}_{\mathsf{g},i}$ for all $\beta \log \beta$ possible choices of (β_{g}, i) is $\widetilde{O}(\beta)$. Moreover, the algorithm uses $\widetilde{O}(k)$ space to compute the indices of the best k-cover. In total, the space complexity is $\widetilde{O}(\beta + k)$. □

Note that the approximation analysis of ReportLargeCommon is essentially the same as the analysis of LargeCommon in Theorem 5.4.4.

Reporting a k-cover in LargeSet. This subroutine invokes LargeSetComplete (Algorithm 24) on different sets of sampled elements and in order to report an actual k-cover along with a $(1/\widetilde{O}(\alpha))$-approximation of the optimal coverage size, we modify the LargeSetComplete subroutine slightly.

In LargeSetComplete, it is always guaranteed that a superset with sufficiently large coverage is detected and its coverage is computed and reported. To get the actual collection of sets that achieve the coverage, we need to try h on all sets (which belong to $[m]$) and report the indices of those that are mapped to the superset with large coverage. More precisely, anywhere in Figure 24 that the algorithm returns a (non **infeasible**) value (the lines that are marked by **(⋆⋆)**), it also returns the set $\{S \mid h(S) = i^*\}$ which is guaranteed to have size at most $\mathsf{w} = \min(\alpha, k)$.

Lemma 5.5.2. *The space complexity of the modified* LargeSetComplete *with the suggested modification at lines* **(⋆⋆)** *is* $\widetilde{O}(\frac{m}{\alpha^2} + k)$.

Proof: Similarly to the space analysis of LargeSetComplete without the modification (Lemma 5.8.7), the modified algorithm consumes $\widetilde{O}(m/\alpha^2)$ space to compute a $(1/\widetilde{O}(\alpha))$-approximation of the optimal coverage size. Moreover, the new modified algorithm uses additional $\widetilde{O}(\mathsf{w}) = \widetilde{O}(k)$ space to find the k-cover corresponding to the returned estimate of the optimal coverage size. Hence, the total space complexity of LargeSetComplete with the suggested modification at lines **(⋆⋆)** is $\widetilde{O}(\frac{m}{\alpha^2} + k)$. □

Corollary 5.5.3. *The space complexity of the* LargeSet *that invokes* LargeSetComplete *with the suggested modification at lines* **(⋆⋆)** *is* $\widetilde{O}(\frac{m}{\alpha^2} + k)$.

Reporting a k-cover in SmallSet. This is the most straightforward case. The only required change is to modify the line marked by **(⋆⋆)** in SmallSet to return both an $O(1)$-approximation of the $\mathsf{Max}\,\widetilde{O}(\frac{k}{\alpha})$-$\mathsf{Cover}$ instance at line **(⋆⋆)** and its corresponding $\widetilde{O}(\frac{k}{\alpha})$-cover. Clearly, the extra amount of space to implement this modification is $\widetilde{O}(\frac{k}{\alpha})$.

Lemma 5.5.4. *The space complexity of the modified* SmallSet *with the suggested modification at line* **(⋆⋆)** *is* $\widetilde{O}(\frac{m}{\alpha^2} + \frac{k}{\alpha})$.

We also need to slightly modify the described Oracle in Algorithm 18 so that, besides a $(1/\widetilde{O}(\alpha))$-estimate of the optimal coverage size, it reports a $(1/\widetilde{O}(\alpha))$-approximate k-cover. Note that it is crucial to run ReportLargeCommon with different parameters depending on whether $\mathsf{s}\alpha \geq 2k$ and $\mathsf{s}\alpha < 2k$. The main reason is that the space complexity of ReportLargeCommon as described above in Algorithm 22 has an additive $\Theta(\beta)$ term

Algorithm 23 An (α, δ, η)-oracle of Max k-Cover that reports a $(1/\widetilde{O}(\alpha))$-approximate k-cover.

procedure ReportOracle((k, α))
  if $\mathsf{s}\alpha \geq 2k$ **then**
    ▷ if $|\mathcal{U}_k^{\mathsf{cmn}}| \geq \frac{\sigma|\mathcal{U}|}{\alpha}$
    $\text{sol}_{\mathsf{cmn}} \leftarrow$ ReportLargeCommon$(k, \alpha, 1)$
    ▷ if $\mathsf{s}\alpha \geq 2k$, then $|\text{OPT}_{\mathsf{large}}| \geq |\text{OPT}|/2$
    $\text{sol}_{\mathsf{HH}} \leftarrow$ LargeSet(k, α, k) ▷ the described modified LargeSetComplete
  else
    ▷ for instances in which $\exists \beta \leq \alpha$ s.t. $|\mathcal{U}_{\beta k}^{\mathsf{cmn}}| \geq \frac{\sigma\beta|\mathcal{U}|}{\alpha}$
    $\text{sol}_{\mathsf{cmn}} \leftarrow$ ReportLargeCommon(k, α, α)
    ▷ for instances with $\mathsf{s}\alpha < 2k$ and $|\text{OPT}_{\mathsf{large}}| \geq |\text{OPT}|/2$
    $\text{sol}_{\mathsf{HH}} \leftarrow$ LargeSet(k, α, α) ▷ the described modified LargeSetComplete
    ▷ for instances with $|\text{OPT}_{\mathsf{large}}| < |\text{OPT}|/2$
    $\text{sol}_{\mathsf{small}} \leftarrow$ SmallSet(k, α) ▷ with the described modification in line $(\star\star)$
  ▷ each sol computed above is of from (estimate of the optimal coverage size, k-cover)
  return $\max(\text{sol}_{\mathsf{cmn}}, \text{sol}_{\mathsf{HH}}, \text{sol}_{\mathsf{small}})$ ▷ max is over the estimate size of sols

which can be as large as α in our implementation of LargeCommon in Algorithm 19.

By all minor modifications in subroutines LargeCommon, LargeSet and SmallSet, we design a modified variant of Oracle called ReportOracle as in Algorithm 23.

Theorem 5.5.5. ReportOracle(k, α) *performs a single pass on the edge arrival stream of the set system* $(\mathcal{U}, \mathcal{F})$ *and implements an* $(\widetilde{O}(\alpha), 1/(\log n \log^c m), \eta)$*-oracle of* Max k-Cover$(\mathcal{U}, \mathcal{F})$ *using* $\widetilde{O}(m/\alpha^2 + k)$ *space. Moreover, the modified oracle also reports a* k*-cover with the promised coverage size.*

Proof: The approximation analysis of the algorithm is basically the same as the analysis of Oracle and we do not repeat it here. We only need show that structural property on the number of elements in different frequency levels provided by LargeCommon with input parameters $(k, \alpha, 1)$ is sufficient for the (modified variant of) LargeSet in the case $\mathsf{s}\alpha \geq k$. In LargeSet with input $\alpha = \widetilde{\Omega}(k)$, the only bound on the number of common elements that is used in the analysis is $|\mathcal{U}_{\mathsf{w}}^{\mathsf{cmn}}| = |\mathcal{U}_k^{\mathsf{cmn}}| \leq \frac{\sigma|\mathcal{U}|}{\alpha}$ (in Theorem 5.8.6). In other words, in this case, unlike the case $\alpha = \widetilde{O}(k)$, we do not need the guarantee on the size of $\mathcal{U}_{\beta k}^{\mathsf{cmn}}$ for all values of $\beta \leq \alpha$. Hence, for the case $\alpha = \widetilde{\Omega}(k)$, it is sufficient to invoke LargeCommon with $\beta = 1$. Note that the values of the rest of parameters are the same as in Oracle.

To bound the space complexity, we consider two cases: 1) $\mathsf{s}\alpha \geq 2k$ and 2) $\mathsf{s}\alpha < 2k$.

1. **$s\alpha \geq 2k$.** By Lemma 5.5.1 and Corollary 5.5.3, ReportLargeCommon$(k, \alpha, 1)$ and LargeSet that invokes the modified variant of LargeSetComplete respectively use $\widetilde{O}(k)$ and $\widetilde{O}(\frac{m}{\alpha^2} + k)$ space. Hence, the total amount of space used in the case is $\widetilde{O}(\frac{m}{\alpha^2} + k)$.

2. **$s\alpha < 2k$.** In this case, by Lemma 5.5.1, ReportLargeCommon(k, α, α) uses $\widetilde{O}(k + \alpha) = \widetilde{O}(k)$ space. Moreover, by Corollary 5.5.3 and Lemma 5.5.4, the space complexity of the other two subroutines with the described modification is $\widetilde{O}(\frac{m}{\alpha^2} + k)$. Hence, the total amount of space that the algorithm uses in this case is $\widetilde{O}(\frac{m}{\alpha^2} + k)$.

Finally, by the modifications we described above, ReportOracle correctly reports an actual k-cover whose coverage is within a $\widetilde{O}(\alpha)$ factor of the optimal coverage as well. □

The theorem above together with Theorem 5.3.6 completes the proof of Theorem 5.3.2.

5.6. Lower Bound for Estimating Max k-Cover in Edge Arrival Streams

By the result of [29], it is known that estimating the size of an optimal coverage of Max k-Cover within a factor of two requires $\Omega(m)$ space. Their argument relies on a reduction from Set Disjointness problem and implies the mentioned bound for 1-cover instances. In the following, we generalize their approach and provide lower bounds for the all range of approximation guarantees α smaller than $\sqrt{m}$. We remark that both our lower bound result and the lower bound result of [29] are basically similar to the lower bound of L_∞ and L_k estimation first proved in [13, 27].

The lower bound result we explain in this section is based on the well-known r-player Set Disjointness problem with unique set intersection promise which has been studied extensively in communication complexity (e.g. [28, 41, 85]). The setting of the problem is as follows: There are r players and each has a set $T_i \subseteq [m]$. The promise is that the input is in one of the following forms:

- **No Case:** There is a unique element $j \in [m]$ such that for all $i \leq r$, $j \in T_i$.
- **Yes Case:** All sets are pair-wise disjoint.

Moreover, a round of communication consists of each player i sending a message to player $i+1$ in order from $i = 1$ to $r-1$. The goal is that at end of a single round, player r be able to correctly output whether the input belongs to the family of **Yes** instances or **No** instances. Chakrabarti et al. [41] showed the following tight lower bound on the one-way communication complexity of the r-player Set Disjointness problem with unique set intersection promise.

Theorem 5.6.1 (From [41]). *Any randomized one-way protocol that solves r-player* Set Disjointness(m) *with success probability at least $2/3$ requires $\Omega(m/r)$ bits of communication.*

We remark that the same $\Omega(m/r)$ communication lower bound was later proved for the general model (i.e. with multiple rounds) by Gronemeier [85]. However, for our application, the lower bound on the one-way communication model suffices.

Corollary 5.6.2. *Any single-pass streaming algorithm that solves r-player* Set Disjointness(m) *with success probability at least $2/3$ consumes $\Omega(m/r^2)$ space.*

Next, we sketch a reduction from r-player Set Disjointness(m) to Max k-Cover with m sets such that an α-approximation protocol of Max k-Cover solves the corresponding instance of r-player Set Disjointness(m).

To this end, consider an arbitrary instance $\mathcal{I}$ of α-player Set Disjointness(m) problem in which each player i has a set $T_i \subset [m]$. Define $\mathcal{U}_{\mathcal{I}} = \{e_1, \cdots, e_\alpha\}$ to be the set of elements in the Max 1-Cover instance and for each player i if $j \in T_i$ then add $(e_i, S_{j,})$ to the stream. In other words, in the constructed Max 1-Cover($\mathcal{U}_{\mathcal{I}}, \mathcal{F}_{\mathcal{I}} := \{S_1, \cdots, S_m\}$) instance, we have an element e_i corresponding to each player i and there exists a set S_j corresponding to each item $j \in [m]$. Moreover, each set S_j in the Max 1-Cover instance $(\mathcal{U}_{\mathcal{I}}, \mathcal{F}_{\mathcal{I}})$ denotes the set of players in the Set Disjointness(m) instance $\mathcal{I}$ whose input sets contain j; $S_j := \{i \in [\alpha] \mid j \in T_i\}$.

Claim 5.6.3. *If $\mathcal{I}$ is a* **No** *instance, then the optimal coverage of the* Max 1-Cover *instance $(\mathcal{U}_{\mathcal{I}}, \mathcal{F}_{\mathcal{I}})$ is α.*

Proof: In this case, by the unique intersection promise, there exists an item j that belongs to all T_i (for $i \in [\alpha]$). Hence, by the construction of the Max 1-Cover instance, S_j covers the whole $\mathcal{U}_{\mathcal{I}}$. Thus, the optimal 1-cover has size α. □

Claim 5.6.4. *If $\mathcal{I}$ is a* **Yes** *instance, then the optimal coverage of the* Max 1-Cover *instance $(\mathcal{U}_{\mathcal{I}}, \mathcal{F}_{\mathcal{I}})$ is 1.*

Proof: Since T_is are disjoint, for each $j \in [m]$, the set S_j has cardinality one. □

Corollary 5.6.2 together with Claims 5.6.3 and 5.6.4 implies the stated lower bound on α-approximating the optimal coverage size of Max k-Cover in edge-arrival streams in Theorem 5.3.3: *Any single pass (possibly randomized) algorithm on edge-arrival streams that $(1/\alpha)$-approximates the optimal coverage size of* Max k-Cover *requires $\Omega(m/\alpha^2)$ space.*

5.7. Chernoff Bound for Applications with Limited Independence

In this section, we mention some of the results in [159] on applications of Chernoff bound with limited independence that are used in our analysis.

Definition 5.7.1 (Family of d-wise Independent Hash Functions). A family of functions $\mathcal{H} = \{h : [m] \to [n]\}$ is d-wise independent, if for any set of d distinct values $x_1, \cdots, x_d$, the random variables $h(x_1), \cdots, h(x_d)$ are independent and uniformly distributed in $[n]$ when h is picked uniformly at random from $\mathcal{H}$.

Next, we exploit the results that show for small values of d, we can store a family of d-wise hash function in small space and in the same time it suffices for our application of Chernoff bound.

Lemma 5.7.2 (Corollary 3.34 in [166]). *For every values of m, n, and d, there is a family of d-wise independent hash functions $\mathcal{H} = \{h : [m] \to [n]\}$ such that a selecting a random function from $\mathcal{H}$ only requires $d \cdot \log(mn)$.*

Lemma 5.7.3 (Theorem 5 in [159]). *Let $X_1, \cdots X_n$ be binary d-wise independent random variables and let $X := X_1 + \cdots + X_n$. Then,*

$$\Pr(|X - \mathbf{E}[X]| \geq \delta \mathbf{E}[X]) \leq \begin{cases} e^{-\mathbf{E}[X]\delta^2/3} : & \text{if } \delta < 1 \text{ and } d = \Omega(\delta^2 \mathbf{E}[X]); \\ e^{-\mathbf{E}[X]\delta/3} : & \text{if } \delta \geq 1 \text{ and } d = \Omega(\delta \mathbf{E}[X]). \end{cases}$$

Lemma 5.7.4 (Theorem 6 in [159]). *Let $X_1, \cdots, X_n$ and $Y_1, \cdots, Y_n$ be Bernoulli trials*

such that for each i, $\mathbf{E}[X_i] = \mathbf{E}[Y_i] = p_i$. Let assume that Y_is are independent, but X_is are only d-wise independent. Further, let $p(r)$ and $p_d(r)$ respectively denote $\Pr(\sum_{i=1}^{n} Y_i = r)$ and $\Pr(\sum_{i=1}^{n} X_i = r)$ and let $\mu = \sum_{i=1}^{n} p_i$ be the expected number of success in the trials.

If $d \geq e\mu + \ln(1/p(0)) + r + D$, then $|p_d(r) - p(r)| \leq e^{-D}p(r)$.

5.7.1. An Application: Set Sampling with $\Theta(\log(mn))$-wise Independence

Consider a set system $(\mathcal{U}, \mathcal{F})$ and let $h : \mathcal{F} \to [(cm \log m)/\gamma]$ be a function selected uniformly at random from a family of $\Theta(\log(mn))$-wise independent hash functions where c is a sufficiently large constant. Then, we think of our randomly selected sets $\mathcal{F}^{\mathsf{rnd}}$ to be the collection of sets in $\mathcal{F}$ that are mapped to one by h; $\mathcal{F}^{\mathsf{rnd}} := \{S \in \mathcal{F} \mid h(S) = 1\}$.

Lemma 5.7.5. *Assuming $\gamma \geq 6c \log^2 m$, with probability at least $1 - m^{-1}$, $|\mathcal{F}^{\mathsf{rnd}}| \leq \gamma$.*

Proof: Let X_i be a random variable which is one if $S_i \in \mathcal{F}^{\mathsf{rnd}}$ and zero otherwise. We define $X := X_1 + \cdots + X_m$. Note that X_i are $\Theta(\log(mn))$-wise independent and $\mathbf{E}[X] = \gamma/(c \log m)$. Then, by an application of Chernoff bound with limited independence (Lemma 5.7.3),

$$\Pr(X > \gamma) \leq \Pr(X > \underbrace{(1 + \sqrt{\frac{6c}{\gamma}} \log m)\mathbf{E}[X]}_{\leq 2\gamma/c}) < m^{-1}.$$

Hence, with high probability, $\mathcal{F}^{\mathsf{rnd}}$ has size at most γ. □

Next, we show that $\mathcal{F}^{\mathsf{rnd}}$ covers the set of elements $\mathcal{U}_{\gamma}^{\mathsf{cmn}}$ (see Definition 5.2.1).

Lemma 5.7.6. *With probability at least $1 - n^{-1}$, $\mathcal{F}^{\mathsf{rnd}}$ covers $\mathcal{U}_{\gamma}^{\mathsf{cmn}}$.*

Proof: Let $e \in \mathcal{U}_{\gamma}^{\mathsf{cmn}}$ and let $S_1, \cdots, S_q$ be the sets in $\mathcal{F}$ that cover e: for each $i \leq q$, $e \in S_i$. We define X_i to be a random variable which is one if $S_i \in \mathcal{F}^{\mathsf{rnd}}$ and is zero otherwise. We also define $X := X_1 + \cdots + X_q$ to denote the number of sets in $\mathcal{F}^{\mathsf{rnd}}$ that cover e. Note that X_i are $\Theta(\log(mn))$-wise independent and $\mathbf{E}[X] = (\gamma/(cm \log m)) \cdot q \geq \log n \log(mn)$. Then, applying Chernoff bound on random variables with limited independence (Lemma 5.7.3),

$$\Pr(X < 1) < \Pr(X < (1 - \underbrace{\sqrt{(6 \log n)/\mathbf{E}[X]}}_{\leq 1/2})\mathbf{E}[X]) < n^{-2}.$$

Hence, by union bound over all elements in $\mathcal{U}_{\gamma}^{\mathsf{cmn}}$, with high probability, $\mathcal{F}^{\mathsf{rnd}}$ covers $\mathcal{U}_{\gamma}^{\mathsf{cmn}}$.□

Lemma 5.7.7. $\Theta(\log(mn))$ *random bits suffice to implement set sampling method.*

5.8. Generalization of Section 5.4.2

In this section we generalize the approach of Section 5.4.2 which relates the results on heavy hitters and contributing classes to approximating the optimal coverage size. In Section 5.4.2, to simplify the presentation, we had the assumption that $\mathcal{U}_{\mathsf{w}}^{\mathsf{cmn}}$ is empty. This is in particular important since we can then assume that the total size of k-covers are roughly the same as their coverage size (up to polylogarithmic factors).

Here, we take care of the case in which $\mathcal{U}_{\mathsf{w}}^{\mathsf{cmn}}$ is non-empty and complete the description of our (α, δ, η)-oracle of Max k-Cover for the case $|\mathcal{C}(\mathrm{OPT}_{\mathsf{large}})| \geq |\mathcal{C}(\mathrm{OPT})|/2$. The high level idea is to sample enough number of elements so that the algorithm using heavy hitters and contributing classes still work but in the same time no w-common element is among the sampled elements with at least a constant probability.

Step 1. Sampling Elements. We sample a subset $\mathcal{L} \subset \mathcal{U}$ in which each element e is in $\mathcal{L}$ with probability $\rho = (\mathsf{t} \cdot \mathsf{s}\alpha \cdot \eta)/|\mathcal{U}|$ where $\mathsf{t} = \widetilde{O}(1)$ (see Table 5.4.1 for the exact values). We implement the process of sampling $\mathcal{L}$ via a hash function from a family of $\Theta(\log(mn))$-wise independent functions $\mathcal{H} = \{h : \mathcal{U} \to [\frac{n}{\mathsf{t}\cdot\mathsf{s}\alpha\cdot\eta}]\}$ such that $\mathcal{L} = \{e \in \mathcal{U} \mid h(e) = 1\}$.

Claim 5.8.1. *With high probability,* $(\rho|\mathcal{U}|)/2 \leq |\mathcal{L}| \leq (3\rho|\mathcal{U}|)/2$.

For each collection of set $\mathcal{D}$, we define $\mathcal{D}'$ to be the intersection of $\mathcal{D}$ with $\mathcal{L}$; $\mathcal{D}' := \{S \cap \mathcal{L} \mid S \in \mathcal{D}\}$.

Claim 5.8.2. *If* $|\mathcal{C}(\mathcal{D})| \geq |\mathcal{U}|/(54\mathsf{f}\eta\alpha)$, *then with probability at least* $1 - m^{-2}$, $|\mathcal{C}(\mathcal{D}')| \geq \rho|\mathcal{C}(\mathcal{D})|/2$. *Moreover, if* $|\mathcal{C}(\mathcal{D})| < |\mathcal{U}|/(54\mathsf{f}\eta\alpha)$, *then with probability at least* $1 - m^{-2}$, $|\mathcal{C}(\mathcal{D}')| < \mathsf{ts}/(36\mathsf{f})$.

Proof: Since $\mathsf{ts} \geq 27 \cdot 24\mathsf{f} \log m$ (as in Table 5.4.1), it follows from two applications of Chernoff bound on random variables with limited independence (Lemma 5.7.3). □

Similarly to Lemma 5.4.14, we have the following guarantee for LargeSetSimple using the sampled set of elements $\mathcal{L}$.

Lemma 5.8.3. *If $|\mathcal{C}(\mathrm{OPT})| \geq |\mathcal{U}|/\eta$ and $\mathcal{L} \cap \mathcal{U}^{\mathsf{cmn}} = \emptyset$, then with probability at least $1 - 1/(2\log n \log^c m)$, the output of* LargeSetSimple *with parameters $(\mathcal{L}, \mathsf{w}, S_1 = \mathsf{s}_{\mathcal{L}}\alpha, S_2 = \frac{cm\log m}{\mathsf{w}}, \mathrm{thr}_1 = \frac{|\mathcal{L}|}{18\eta\mathsf{s}\alpha}, \mathrm{thr}_2 = \frac{|\mathcal{L}|}{6\eta\alpha})$ is a superset whose coverage is at least $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$.*

Proof: By Claim 5.8.1, $(\rho|\mathcal{U}|)/2 \leq |\mathcal{L}| \leq (3\rho|\mathcal{U}|)/2$. Moreover, by Claim 5.8.2 and since $|\mathcal{C}(\mathrm{OPT})| \geq |\mathcal{U}|/\eta$, with probability at least $1 - m^{-2}$, $|\mathcal{C}(\mathrm{OPT}_{\mathsf{large}}) \cap \mathcal{L}| \geq \rho|\mathcal{C}(\mathrm{OPT})|/4 \geq |\mathcal{L}|/(6\eta)$. We define $\eta_{\mathcal{L}} := 6\eta$ to denote the coverage of $\mathrm{OPT}_{\mathsf{large}}$ over the sampled elements $\mathcal{L}$.

Now, consider the collection $\mathrm{OPT}_{\mathsf{large}} := \{O_1, \cdots, O_q\}$. Since for each $i \leq q$, the contribution of O_i to the coverage of OPT is at least $1/(\mathsf{s}\alpha)$ fraction (i.e. $|O_i \setminus \bigcup_{j<i} O_j| \geq |\mathcal{C}(\mathrm{OPT})|/(\mathsf{s}\alpha) \geq |\mathcal{U}|/(\eta\mathsf{s}\alpha)$), by Claim 5.8.2, with probability at least $1 - m^{-1}$, for all $i \leq q$,

$$|(O_i \setminus \bigcup_{j<i} O_j) \cap \mathcal{L}| \geq \rho|\mathcal{C}(\mathrm{OPT})|/(2\mathsf{s}\alpha).$$

This implies that $\{O_i \cap \mathcal{L} \mid O_i \in \mathrm{OPT}_{\mathsf{large}}\}$ is a collection of sets whose contribution to $\mathcal{C}(\mathrm{OPT}) \cap \mathcal{L}$ w.h.p. is at least $(\frac{\rho|\mathcal{C}(\mathrm{OPT})|}{2\mathsf{s}\alpha})/(\frac{3\rho|\mathcal{C}(\mathrm{OPT})|}{2}) = 1/(3\mathsf{s}\alpha)$. We define $\mathsf{s}_{\mathcal{L}} := 3\mathsf{s}$ which denotes the contribution of sets in $\mathrm{OPT}_{\mathsf{large}}$ compared to the coverage of OPT over the sampled elements $\mathcal{L}$.

By an application of Lemma 5.4.14 with parameters $(\mathcal{V} := \mathcal{L}, \mathrm{thr}_1 := \frac{|\mathcal{L}|}{\eta_{\mathcal{L}}\mathsf{s}_{\mathcal{L}}\alpha}, \mathrm{thr}_2 := \frac{|\mathcal{L}|}{\eta_{\mathcal{L}}\alpha})$, with probability at least $1 - 1/(3\log n \log^c m)$, the algorithm returns a superset $\mathcal{D}'_i$ whose coverage on the sampled set $\mathcal{L}$ is at least

$$\frac{|\mathcal{L}|}{3\mathsf{f}\eta_{\mathcal{L}}\alpha} \geq \frac{\rho|\mathcal{U}|}{36\mathsf{f}\alpha\eta} = \frac{\mathsf{ts}}{36\mathsf{f}}.$$

Then, by Claim 5.8.2, with probability at least $1 - m^{-2}$, $\mathcal{D}_i$ has coverage at least $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$.□

Step 2. Handling Common Elements. Next, we turn our attention to the case $\mathcal{L} \cap \mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} \neq \emptyset$. Although common elements may be covered $\widetilde{\Omega}(1)$ times within a single superset, it is important to note that the contribution of common elements to all supersets are roughly the same.

Claim 5.8.4. *Let $\mathcal{L}^{\mathsf{cmn}}_{\mathsf{w}} := \mathcal{L} \cap \mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}}$ be the set of w-common elements that are sampled in $\mathcal{L}$.*

Then, with high probability, for each superset $\mathcal{D}$, the total number of times that elements of $\mathcal{L}_{\mathsf{w}}^{\mathsf{cmn}}$ appear in $\mathcal{D}$ (counting duplicates) belongs to $[P, 2P]$ where P is a fixed number larger than $\log n$.

Proof: Let $e \in \mathcal{U}_{\mathsf{w}}^{\mathsf{cmn}}$ be a w-common element and let $S_1, \cdots, S_q$ be the collection of sets that contain e. For a superset $\mathcal{D}_j$, define $X_{i,j}$ to be a binary random variable that denotes whether $S_i \in \mathcal{D}_j$. Moreover, let $Y_{j,e} := X_{1,j} + \cdots + X_{q,j}$. Then, $\mathbf{E}[Y_{j,e}] = \mathsf{w}q/(cm \log m)$. By an application of Chernoff bound with limited independence (Lemma 5.7.3) and since $\mathsf{w}q \geq cm \log m \log n \log(mn)$ (see Definition 5.2.1),

$$\Pr(|Y_{j,e} - \mathbf{E}[Y_{j,e}]| \geq \underbrace{\sqrt{\frac{6c \log(mn) m \log m}{\mathsf{w}q}}}_{\leq 1/3} \mathbf{E}[Y_{j,e}]) \leq (mn)^{-2}. \tag{5.8.1}$$

Note that for any w-common element e and any pair of supersets $\mathcal{D}_j, \mathcal{D}_i$, $\mathbf{E}[Y_{j,e}] = \mathbf{E}[Y_{i,e}]$. In particular, we define $Y_e := \mathbf{E}[Y_{j,e}]$ whose value is independet of the supersets. Next, we define $Y_{\mathsf{cmn}} := \sum_{e \in \mathcal{U}^{\mathsf{cmn}}} Y_e$ to denote the expected contribution of w-common elements to any superset. Hence, for each superset $\mathcal{D}_j$, with probability at least $1 - 1/(nm^2)$, the total number of times that w-common elements are covered by $\mathcal{D}_j$ belongs to the range $[2Y_{\mathsf{cmn}}/3, 4Y_{\mathsf{cmn}}/3]$. Hence, with probability at least $1 - (mn)^{-1}$, the contribution of sampled w-common elements to each superset belongs to $[2Y_{\mathsf{cmn}}/3, 4Y_{\mathsf{cmn}}/3]$ where $Y_{\mathsf{cmn}} \geq \log n \log(mn) |\mathcal{L}_{\mathsf{w}}^{\mathsf{cmn}}| \geq \log n$. □

Next, we show that if a w-common element is picked in $\mathcal{L}$ the algorithm still does not return a superset with small coverage (though it may missed all large supersets). To this end, we modify LargeSetSimple and design a new subroutine LargeSetComplete as in Figure 24. The high level idea is to guarantee that if the main contribution of a superset is just from the duplicate counts of w-common elements ($\propto P$), it will not be returned. To achieve this, unlike LargeSetSimple we do not allow F_2-Contributing to check for all contributing class of any size (up to $|\mathcal{Q}|$). Instead, we set parameters S_1 and S_2 which denotes how large the size of a contributing class that we are looking for is. To handle the case in which the size of a contributing class is large (i.e. larger than S_2), we sample supersets proportional to $1/S_2$ and compute their coverage by existing L_0-estimation algorithms.

Algorithm 24 A (α, δ, η)-oracle of Max k-Cover that handles the case in which the majority of the coverage in an optimal solution is by the sets whose coverage contributions are at least $1/(\mathsf{s}\alpha)$ fraction of the optimal coverage size. By modifying the lines marked by **(⋆⋆)** slightly, the algorithm returns the k-cover that achieves the returned coverage estimate (see Section 5.5 for more explanation).

procedure LargeSetComplete($(\mathcal{V}, \mathsf{w}, S_1, S_2, \mathrm{thr}_1, \mathrm{thr}_2)$)
  ▷ **Input:** w is an upper bound on the desired number of sets in a superset
  ▷ **Parameters:** $\phi_1 = \widetilde{\Omega}(\alpha^2/m)$ and $\phi_2 = \widetilde{\Omega}(1)$
  let $\mathsf{Cntr}_{\mathsf{small}}$ be an instance of F_2-Contributing(ϕ_1, S_1) ▷ for Case 1
  let $\mathsf{Cntr}_{\mathsf{large}}$ be an instance of F_2-Contributing(ϕ_2, S_2) ▷ for Case 2
  pick $h : \mathcal{F} \to [(cm \log m)/\mathsf{w}]$ from $\Theta(\log(mn))$-wise independent hash functions
  for all (S, e) in the data stream **do**
    if $e \in \mathcal{V}$ **then**
      feed $h(S)$ to both $\mathsf{Cntr}_{\mathsf{small}}$ and $\mathsf{Cntr}_{\mathsf{large}}$
  ▷ $\mathsf{output}(\mathsf{Cntr})$ contains coordinates along with $(1 \pm 1/2)$-estimate of frequencies
  if there exists $i^* \in \mathsf{output}(\mathsf{Cntr}_{\mathsf{small}})$ s.t. $\tilde{v}_{i^*} \geq \frac{1}{2} \cdot \mathrm{thr}_1$ **then**
    return $2\tilde{v}_{i^*}/(3\mathsf{f})$ **(⋆⋆)** ▷ add **return** $\{S \mid h(S) = i^*\}$ to get the k-cover
  if there exists $i^* \in \mathsf{output}(\mathsf{Cntr}_{\mathsf{large}})$ s.t. $\tilde{v}_{i^*} \geq \frac{1}{2} \cdot \mathrm{thr}_2$ **then**
    return $2\tilde{v}_{i^*}/(3\mathsf{f})$ **(⋆⋆)** ▷ add **return** $\{S \mid h(S) = i^*\}$ to get the k-cover
  ▷ case 2: if size of the contributing class is large; $\widetilde{\Omega}(|\mathcal{Q}|)$
  let $\mathcal{M} \subset \mathcal{Q}$ be a collection of size $12|\mathcal{Q}| \log m / S_2$ picked uniformly at random
  for all $i \in \mathcal{M}$ **do** ▷ DE to estimate the coverage of the supersets in $\mathcal{L}$
    let DE_i be a $(1/2)$-approximation algorithm of L_0-estimation initialized to zero
  pick $h : \mathcal{F} \to [(cm \log m)/\mathsf{w}]$ from $\Theta(\log(mn))$-wise independent hash functions
  for all (S, e) in the data stream **do**
    if $e \in \mathcal{V}$ and $h(S) \in \mathcal{M}$ **then feed** $h(S)$ to $\mathsf{DE}_{h(S)}$
  if there exists $i^* \in \mathcal{M}$ such that $\mathrm{VAL}(\mathsf{DE}_{i^*}) \geq \frac{1}{2} \cdot \mathrm{thr}_2$ **then**
    return $2\mathrm{VAL}(\mathsf{DE}_{i^*})/3$ **(⋆⋆)** ▷ add **return** $\{S \mid h(S) = i^*\}$ to get the k-cover
  return infeasible

Lemma 5.8.5. *Even if $\mathcal{L} \cap \mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} \neq \emptyset$, with probability at least $1 - m^{-1}$, none of the solutions returned by* LargeSetComplete *with parameters $(\mathcal{L}, \mathsf{w}, S_1 = \mathsf{s}_{\mathcal{L}}\alpha, S_2 = \widetilde{\Theta}(\frac{cm \log m}{\mathsf{w}}), \mathrm{thr}_1 = \frac{|\mathcal{L}|}{18\eta\mathsf{s}\alpha}, \mathrm{thr}_2 = \frac{|\mathcal{L}|}{6\eta\alpha})$ is a superset whose coverage is at less than $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$.*

Proof: Here, we need to revisit Case 1 and Case 2 of Section 5.4.2 and redo the calculations with respect to the sampled set of elements $\mathcal{L}$.

Case 1. Suppose that F_2-Contributing$(\widetilde{\Omega}(\alpha^2/m), \mathsf{s}_{\mathcal{L}}\alpha)$ returns a solution whose coverage is less than $|\mathcal{U}|(54\mathsf{f}\eta\alpha)$.

Let $\vec{r}$ be a vector of size $(cm \log m)/\mathsf{w}$ whose ith entry denotes the total size of the

intersection of sets in $\mathcal{D}_i$ and $\mathcal{L}^{\mathsf{rare}}_{\mathsf{w}} := \mathcal{L} \setminus \mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}}$; $\vec{r}[i] := \sum_{S \in \mathcal{D}_i} |S \cap \mathcal{L}^{\mathsf{rare}}_{\mathsf{w}}|$. By Claim 5.8.2, if $|\mathcal{C}(\mathcal{D}_j)| < |\mathcal{U}|/(54\mathsf{f}\eta\alpha)$, with probability at least $1 - m^{-2}$, $|\mathcal{C}(\mathcal{D}_i) \cap \mathcal{L}^{\mathsf{rare}}_{\mathsf{w}}| < |\mathcal{C}(\mathcal{D}_i) \cap \mathcal{L}| < \mathsf{ts}/(36\mathsf{f})$. Hence, together with Claim 5.4.10, with probability at least $1 - 2m^{-2}$, $\vec{r}[i] < \mathsf{ts}/36$.

Similarly, let $\vec{v}$ be a vector of size $(cm \log m)/\mathsf{w}$ whose ith entry denotes the total size of the intersection of sets in $\mathcal{D}_i$ with the sampled elements $\mathcal{L}$; $\vec{v}[i] := \sum_{S \in \mathcal{D}_i} |S \cap \mathcal{L}|$. Note that Since $P \geq 1$, for each superset $\mathcal{D}_i$ with coverage less than $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$, $\vec{v}[i] \leq 2P + \vec{r}[i] \leq (\mathsf{ts}/36)P$. On the other hand, by Claim 5.8.4, with probability at least $1 - m^{-1}$, the value of $\vec{v}[j]$ for all j in the sampled substream is at least P.

Moreover, since the size of contributing classes in this case is at most $\mathsf{s}_{\mathcal{L}}\alpha = 3\mathsf{s}\alpha$ (more precisely, we can always find at most $3\mathsf{s}\alpha$ sets that are $\widetilde{\Omega}(\frac{\alpha^2}{m})$-contributing), by Claim 5.2.8, with probability at least $1 - \frac{\log m}{m}$, all sampled substreams considered by F_2-CONTRIBUTING$(\phi_1, \mathsf{s}_{\mathcal{L}}\alpha)$ have size at least $cm \log m/(\mathsf{ws}\alpha)$. Hence, by Claim 5.8.4, with probability at least $1 - \frac{2\log m}{m}$, $F_2(\vec{v}_{\mathsf{smp}}) \geq (\frac{cm \log m}{\mathsf{ws}\alpha})P^2$ where $\vec{v}_{\mathsf{smp}}$ is a vector corresponding to a sampled substream considered in F_2-CONTRIBUTING$(\phi_1, \mathsf{s}_{\mathcal{L}}\alpha)$. Since $\mathsf{s}^4\mathsf{t}^2 \leq 81/(2\eta \log(\mathsf{s}\alpha))$ (see Table 5.4.1) and by the value of ϕ_1 (in Eq. 5.4.2), the following holds:

$$(\frac{\mathsf{ts}}{36})^2 P^2 < \phi_1 \cdot \frac{cm \log m}{\mathsf{ws}\alpha} P^2,$$

which implies that with probability at least $1 - 3\log m/m$, an entry corresponding to a superset with coverage less than $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$ cannot be a ϕ_1-HeavyHitter in any of the sampled substreams considered in F_2-CONTRIBUTING$(\phi_1, \mathsf{s}_{\mathcal{L}}\alpha)$.

Case 2. The high level idea in this case is similar to the previous case. In Case 1, we heavily use the fact that there exists a class containing at most $\mathsf{s}_{\mathcal{L}}\alpha$ coordinates that is $\widetilde{\Omega}(\alpha^2/m)$-contributing. This observation is crucial because then we could argue that all sampled substreams considered in F_2-CONTRIBUTING$(\phi_1, \mathsf{s}_{\mathcal{L}}\alpha)$ have size at least $\widetilde{\Omega}(m/(\mathsf{ws}_{\mathcal{L}}\alpha))$ which rules out the possibility that a coordinate corresponding to a small superset is a $\widetilde{\Omega}(\alpha^2/m)$-HeavyHitter for sufficiently small values of s (recall that $\mathsf{s}_{\mathcal{L}} = 3\mathsf{s}$).

However, in this case, a contributing class may have size $\widetilde{\Omega}(m/\mathsf{w})$ which results in a sampled substream with only $\widetilde{O}(1)$ coordinates in the run of F_2-CONTRIBUTING! To address

the issue, we handle the case in which a contributing class has more than S_2 coordinates separately (S_2 is a parameter to be determined shortly):

- **$\widetilde{\Omega}(1)$-contributing class has size less than $S_2 := \frac{cm\log m}{\mathsf{w}} \cdot \gamma$.** Since $P \geq 1$ and by Claim 5.8.2 and 5.4.10, for each superset $\mathcal{D}_j$ with coverage less than $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$, with probability at least $1-2m^{-2}$, $\vec{v}[j] \leq (\mathsf{ts}/36)P$. On the other hand, by Claim 5.8.4, with probability at least $1-m^{-1}$, the value of $\vec{v}[j]$ for all j in the sampled substream is at least P. Moreover, by Claim 5.2.8, with probability at least $1-\frac{\log m}{m}$, all sampled substreams considered in F_2-CONTRIBUTING($\phi_2 = \frac{1}{2\log(\alpha)}, S_2$) invoked by LARGESETCOMPLETE (which is to handle Case 2) have at least $(\frac{3cm\log m}{\mathsf{w}\cdot S_2}) = \frac{3}{\gamma}$ coordinates. Hence, $F_2(\vec{v}_{\mathsf{smp}}) \geq \frac{3}{\gamma}\cdot P^2$. By setting

$$\gamma < \frac{3\phi_2}{(\mathsf{ts}/36)^2} = \frac{1944}{\log(\alpha)\mathsf{t}^2\mathsf{s}^2}, \tag{5.8.2}$$

  $\vec{v}[j]^2 \leq (\mathsf{ts}/36)^2P^2 < (\frac{1}{2\log\alpha}F_2(\vec{v}_{\mathsf{smp}}))$, which implies that an entry corresponding to a superset with coverage less than $|\mathcal{U}|/(27\mathsf{f}\eta\alpha)$ cannot be a ϕ_2-HeavyHitter in any of the sampled substream considered in F_2-CONTRIBUTING(ϕ_2, S_2).

- **$\widetilde{\Omega}(1)$-contributing class has size at least $S_2 := \frac{cm\log m}{\mathsf{w}} \cdot \gamma$.** Here, we also need to consider an extra case compared to Lemma 5.4.14 and 5.8.3 because we do not allow S_2 to try all values up to $(\frac{cm\log m}{\mathsf{w}})$. To address the case in which the number of coordinates in a contributing class is larger than S_2, we sample $\ell = (12\log m)|\mathcal{Q}|/S_2$ supersets $\mathcal{M}$ uniformly at random from $\mathcal{Q}$; with high probability, $\mathcal{M}$ contains a superset from the contributing class. Then, we compute the coverage of sampled supersets via an existing algorithm for L_0-estimation. By Claim 5.4.13, the coverage of supersets corresponding to the ϕ_2-contributing class whose size is larger than S_2 on the sampled set $\mathcal{L}$ is at least $|\mathcal{L}|/(\eta_{\mathcal{L}}\alpha) \geq \mathsf{ts}/(12\mathsf{f})$. Hence, the algorithm finds a superset with coverage at least $\mathsf{ts}/(36\mathsf{f})$ on $\mathcal{L}$ which by Claim 5.8.2, it implies that the returned superset has coverage at least $|\mathcal{U}|/(54\mathsf{f}\eta\mathsf{s}\alpha)$. □

Theorem 5.8.6. *If $|\mathcal{C}(\mathrm{OPT})| \geq |\mathcal{U}|/\eta$, then with probability at least $1-1/(\log n\log^c m)$,* LARGESET(k,α) *returns at least $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$. Moreover, if* LARGESET$(k,\alpha)$ *returns a value*

other than **infeasible**, *then with probability at least* $1 - 4m^{-1}$, $|\mathcal{C}(\mathrm{OPT})| \geq |\mathcal{U}|/(54\mathsf{f}\eta\alpha)$.

Proof: By Lemma 5.8.5, with probability at least $1 - 3m^{-1}$, LargeSet will never return a superset with coverage less than $|\mathcal{U}|/(27\mathsf{f}\eta\alpha)$; either it returns a large enough estimate or returns **infeasible**. Here, we show that if $|\mathcal{C}(\mathrm{OPT})| > |\mathcal{U}|/\eta$, then the algorithm will return an estimate at least $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$ with probability at least $1 - 1/(2\log n \log^c m) - n^{-1}$. To this end, we show that with high probability, in one of the $O(\log n)$ parallel runs of LargeSet, the sampled sets of elements $\mathcal{L}$ does not contain any common element. Then, by Lemma 5.8.3, the algorithm with probability at least $1 - 1/(2\log n \log^c m)$ returns $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$ in the iteration in which the sampled set of elements that does not contain any common element.

Now, we show that with probability at least $1 - n^{-1}$, the sampled set of one of $O(\log n)$ parallel runs of LargeSet does not contain any common element. Let $q = |\mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}}|$ and define $Y_1, \cdots, Y_q$ to be independent Bernoulli trials with probability of success equal to ρ. Recall that, we have the assumption that $|\mathcal{U}^{\mathsf{cmn}}_k| \leq \frac{\sigma|\mathcal{U}|}{\alpha}$ and since $\mathsf{w} \leq k$, $|\mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}}| \leq |\mathcal{U}^{\mathsf{cmn}}_k| \leq \frac{\sigma|\mathcal{U}|}{\alpha}$,

$$\mu = \mathbf{E}[\sum_{i=1}^{q} Y_i] \leq \frac{\sigma|\mathcal{U}|}{\alpha} \cdot \rho = \mathsf{ts}\eta\sigma$$

$$\Pr(\sum_{i=1}^{q} Y_i = 0) = (1-\rho)^q \geq e^{-2\rho q} \geq e^{-2\mathsf{ts}\eta\sigma}$$

Next, let's assume that $\mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} = \{e_1, \cdots, e_q\}$. Further, define $X_1, \cdots, X_q$ to be random variables such that $X_i = 1$ if $e_i \in \mathcal{L}$. Hence, $X_1, \cdots, X_q$ are $\Theta(\log(mn))$-wise independent Bernoulli trials with success probability $\mathbf{E}[X_i] = \rho$. Next, we apply Lemma 5.7.4 with the following parameters:

$$r = 0, \ln(1/p(0)) \leq 2\mathsf{ts}\eta\sigma, \mu = \mathsf{ts}\eta\sigma, D = \Theta(\log(mn)) = 12\log(mn),$$

and show that

$$\Pr(\mathcal{L} \cap \mathcal{U}^{\mathsf{cmn}}_{\mathsf{w}} = \emptyset) = \Pr(\sum_{i=1}^{q} X_i = 0) \geq \Pr(\sum_{i=1}^{q} Y_i)(1 - e^{-D}) \geq e^{-2\mathsf{ts}\eta\sigma}/2.$$

Algorithm 25 A single pass streaming algorithm.

procedure LARGESET((k, α, w))
  ▷ run LARGESETCOMPLETE on sampled elements in parallel for $O(\log n)$ instances
  for $O(\log n)$ times **in parallel do**
    let $\mathcal{L} \subseteq \mathcal{U}$ s.t. each $e \in \mathcal{L}$ ($\Theta(\log(mn))$-wise independent) w.p. $\rho = \frac{\mathsf{t}\cdot\mathsf{s}\alpha\cdot\eta}{|\mathcal{U}|}$
    $S_1 \leftarrow \mathsf{s}_{\mathcal{L}}\alpha$, $S_2 \leftarrow \frac{cm\log m}{\mathsf{w}} \cdot \gamma$, $\mathrm{thr}_1 \leftarrow |\mathcal{L}|/(18\eta\mathsf{s}\alpha)$, $\mathrm{thr}_2 \leftarrow |\mathcal{L}|/(6\eta\alpha)$ ▷ $\gamma := \frac{1944}{\mathsf{t}^2\mathsf{s}^2\log\alpha}$
    SOL $\leftarrow$ LARGESETCOMPLETE($\mathcal{L}, \mathsf{w}, S_1, S_2, \mathrm{thr}_1, \mathrm{thr}_2$)
    if SOL $\neq$ **infeasible then**
      return $|\mathcal{U}|/(54\mathsf{f}\eta\alpha)$
  return infeasible

Hence, since $\mathsf{ts}\eta\sigma = \eta = O(1)$ (see Table 5.4.1),

$$\Pr(\text{In all runs, } \mathcal{L} \cap \mathcal{U}^{\mathsf{cmn}} \neq \emptyset) \leq (1 - e^{-2\mathsf{ts}\eta\sigma})^{O(\log n)} \leq n^{-1}.$$

The second property follows from Lemma 5.8.5: if the algorithm returns a value other than **infeasible**, then with probability at least $1 - 4m^{-1}$, $|\mathcal{C}(\mathrm{OPT})| \geq |\mathcal{U}|/(54\mathsf{f}\eta\alpha)$. □

Lemma 5.8.7. *The amount of space used by* LARGESET *is* $\widetilde{O}(m/\alpha^2)$.

Proof: Note that LARGESET performs $O(\log n)$ instances of LARGESETCOMPLETE in parallel. Hence, the total amount of space use by LARGESET is $O(\log n)$ times the space complexity of LARGESETCOMPLETE.

Similarly to the space analysis of LARGESETSIMPLE, the amount of space to perform $\mathsf{Cntr}_{\mathsf{small}}$ and $\mathsf{Cntr}_{\mathsf{large}}$ as defined in LARGESETCOMPLETE is respectively $\widetilde{O}(1/\phi_1) = \widetilde{O}(m/\alpha^2)$ and $\widetilde{O}(1/\phi_2) = \widetilde{O}(1)$. Moreover, for the last case in which the contributing class has size larger than S_2, by Theorem 5.2.12, in total $\widetilde{O}(\frac{m}{\mathsf{w}\cdot S_2}) = \widetilde{O}(1)$ space is required to compute the coverage of sampled supersets in $\mathcal{M}$. Note that, in all cases, by Lemma 5.7.2, the algorithm can store h in $\widetilde{O}(1)$ space.

Hence, the total amount of space required to implement LARGESET is $\widetilde{O}(m/\alpha^2)$. □

Proof of Theorem 5.4.8: The guarantee on the quality of the returned estimate follows from Theorem 5.8.6 with $\mathsf{w} = \min\{\alpha, k\}$ and $\mathsf{s} = \widetilde{O}(\mathsf{w}/\alpha)$ (as in Table 5.4.1). Moreover, Lemma 5.8.7 shows that the space complexity of LARGESET is $\widetilde{O}(m/\alpha^2)$.

Moreover, since with high probability the estimate returned by the algorithm is a lower bound on the coverage size of a k-cover of $\mathcal{F}$, the output of LARGESET with high probability,

is smaller than the optimal coverage size of $\mathsf{Max}\, k\text{-}\mathsf{Cover}(\mathcal{U}, \mathcal{F})$. $\square$

Part II

Learning-Based Streaming Algorithms

Chapter 6

The Frequency Estimation Problem

6.1. Introduction

The frequency estimation problem is formalized as follows: given a sequence S of elements from some universe U, for any element $i \in U$, estimate f_i, the number of times i occurs in S. If one could store all arrivals from the stream S, one could sort the elements and compute their frequencies. However, in big data applications, the stream is too large (and may be infinite) and cannot be stored. This challenge has motivated the development of *streaming algorithms*, which read the elements of S in a single pass and compute a good estimate of the frequencies using a limited amount of space. Specifically, the goal of the problem is as follows. Given a sequence S of elements from U, the desired algorithm reads S in a single pass while writing into memory C (whose size can be much smaller than the length of S). Then, given any element $i \in U$, the algorithm reports an estimation of f_i based only on the content of C. Over the last two decades, many such streaming algorithms have been developed, including Count-Sketch [43], Count-Min [57] and multi-stage filters [70]. The performance guarantees of these algorithms are well-understood, with upper and lower bounds matching up to $O(\cdot)$ factors [110].

However, such streaming algorithms typically assume generic data and do not leverage useful patterns or properties of their input. For example, in text data, the word frequency is known to be inversely correlated with the length of the word. Analogously, in network data, certain applications tend to generate more traffic than others. If such properties

can be harnessed, one could design frequency estimation algorithms that are much more efficient than the existing ones. Yet, it is important to do so in a general framework that can harness various useful properties, instead of using handcrafted methods specific to a particular pattern or structure (e.g., word length, application type).

In this chapter, we introduce learning-based frequency estimation streaming algorithms. Our algorithms are equipped with a learning model that enables them to exploit data properties without being specific to a particular pattern or knowing the useful property a priori. We further provide theoretical analysis of the guarantees associated with such learning-based algorithms.

We focus on the important class of "hashing-based" algorithms, which includes some of the most used algorithms such as Count-Min, Count-Median and Count-Sketch. Informally, these algorithms hash data items into B buckets, count the number of items hashed into each bucket, and use the bucket value as an estimate of item frequency. The process can be repeated using multiple hash functions to improve accuracy. Hashing-based algorithms have several useful properties. In particular, they can handle item deletions, which are implemented by decrementing the respective counters. Furthermore, some of them (notably Count-Min) never underestimate the true frequencies, i.e., $\tilde{f}_i \geq f_i$ holds always. However, hashing algorithms lead to estimation errors due to collisions: when two elements are mapped to the same bucket, they affect each others' estimates. Although collisions are unavoidable given the space constraints, the overall error significantly depends on the pattern of collisions. For example, collisions between high-frequency elements ("heavy hitters") result in a large estimation error, and ideally should be minimized. The existing algorithms, however, use random hash functions, which means that collisions are controlled only probabilistically.

Our idea is to use a small subset of S, call it S', to learn the heavy hitters. We can then assign heavy hitters their own buckets to avoid the more costly collisions. It is important to emphasize that we are learning the *properties* that identify heavy hitters as opposed to the *identities* of the heavy hitters themselves. For example, in the word frequency case, shorter words tend to be more popular. The subset S' itself may miss many of the popular words, but whichever words popular in S' are likely to be short. Our objective is *not* to learn the identity of high frequency words using S'. Rather, we hope that a learning model trained

on S' learns that short words are more frequent, so that it can identify popular words even if they did not appear in S'.

Our main contributions are as follows:

- We introduce *learning-based* frequency estimation streaming algorithms, which learn the properties of heavy hitters in their input and exploit this information to reduce errors.
- We provide performance guarantees showing that our learned algorithms can deliver a logarithmic factor improvement in the error bound over their non-learning counterparts. Furthermore, we show that our learning-based instantiation of Count-Min, a widely used algorithm, is asymptotically optimal among *all* instantiations of that algorithm. See Table 6.4.1 in Section 6.4 for the details.
- We evaluate our learning-based algorithms using two real-world datasets: network traffic and search query popularity. In comparison to their non-learning counterparts, our algorithms yield performance gains that range from 18% to 71%.

6.1.1. Related Work

Frequency estimation in data streams. Frequency estimation, and the closely related problem of finding frequent elements in a data stream, are some of the most fundamental and well-studied problems in streaming algorithms, see [55] for an overview. Hashing-based algorithms such as Count-Sketch [43], Count-Min [57] and multi-stage filters [70] are widely used solutions for these problems. These algorithms also have close connections to sparse recovery and compressed sensing [40, 64], where the hashing output can be considered as a compressed representation of the input data [79].

Several "non-hashing" algorithms for frequency estimation have been also proposed [136, 62, 116, 133]. These algorithms do not possess many of the properties of hashing-based methods listed in the introduction (such as the ability to handle deletions), but they often have better accuracy/space tradeoffs. For a fair comparison, our evaluation focuses only on hashing algorithms. However, our approach for learning heavy hitters should be useful for non-hashing algorithms as well.

Some papers have proposed or analyzed frequency estimation algorithms customized to data that follows Zipf Law [43, 58, 133, 135, 154]; the last algorithm is somewhat similar to the "lookup table" implementation of the heavy hitter oracle that we use as a baseline in our experiments. Those algorithms need to know the data distribution a priori, and apply only to one distribution. In contrast, our learning-based approach applies to any data property or distribution, and does not need to know that property or distribution a priori.

Learning-based algorithms. Recently, researchers have begun exploring the idea of integrating machine learning models into algorithm design. In particular, researchers have proposed improving compressed sensing algorithms, either by using neural networks to improve sparse recovery algorithms [140, 33], or by designing linear measurements that are optimized for a particular class of vectors [25, 141], or both. The latter methods can be viewed as solving a problem similar to ours, as our goal is to design "measurements" of the frequency vector $(f_1, f_2 \ldots, f_{|U|})$ tailored to a particular class of vectors. However, the aforementioned methods need to explicitly represent a matrix of size $B \times |U|$, where B is the number of buckets. Hence, they are unsuitable for streaming algorithms which, by definition, have space limitations much smaller than the input size.

Another class of problems that benefited from machine learning is *distance estimation*, i.e., compression of high-dimensional vectors into compact representations from which one can estimate distances between the original vectors. Early solutions to this problem, such as *Locality-Sensitive Hashing*, have been designed for worst case vectors. Over the last decade, numerous methods for learning such representations have been developed [156, 169, 109, 168]. Although the objective of those papers is similar to ours, their techniques are not usable in our applications, as they involve a different set of tools and solve different problems.

More broadly, there have been several recent papers that leverage machine learning to design more efficient algorithms. The authors of [59] show how to use reinforcement learning and graph embedding to design algorithms for graph optimization (e.g., TSP). Other learning-augmented combinatorial optimization problems are studied in [101, 24, 128]. More recently, [120, 137] have used machine learning to improve indexing data structures, including Bloom filters that (probabilistically) answer queries of the form "is a given element in the

data set?" As in those papers, our algorithms use neural networks to learn certain properties of the input. However, we differ from those papers both in our design and theoretical analysis. Our algorithms are designed to reduce collisions between heavy items, as such collisions greatly increase errors. In contrast, in existence indices, all collisions count equally. This also leads to our theoretical analysis being very different from that in [137].

6.2. Preliminaries

Estimation Error. To measure and compare the overall accuracy of different frequency estimation algorithms, we will use the *expected* estimation error which is defined as follows: let $\mathcal{F} = \{f_1, \cdots, f_n\}$ and $\tilde{\mathcal{F}}_{\mathcal{A}} = \{\tilde{f}_1, \cdots, \tilde{f}_n\}$ respectively denote the actual frequencies and the estimated frequencies obtained from algorithm $\mathcal{A}$ of items in the input stream. We remark that when $\mathcal{A}$ is clear from the context we denote $\tilde{\mathcal{F}}_{\mathcal{A}}$ as $\tilde{\mathcal{F}}$. Then we define

$$\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{\mathcal{A}}) := \mathbf{E}_{i\sim\mathcal{D}}|f_i - \tilde{f}_i|, \tag{6.2.1}$$

where $\mathcal{D}$ denotes the *query distribution* of the items. Here, we assume that the query distribution $\mathcal{D}$ is the same as the frequency distribution of items in the stream, i.e., for any $i^* \in [n]$, $\Pr_{i\sim\mathcal{D}}[i = i^*] \propto f_{i^*}$ (more precisely, for any $i^* \in [n]$, $\Pr_{i\sim\mathcal{D}}[i = i^*] = f_{i^*}/N$ where $N = \sum_{i\in[n]} f_i$ denotes the total sum of all frequencies in the stream).

We note that the theoretical guarantees of frequency estimation algorithms are typically phrased in the "(ε, δ)-form", e.g., $\Pr[|\tilde{f}_i - f_i| > \varepsilon N] < \delta$ for *every* i (see e.g., [57]). However, this formulation involves two objectives (ε and δ). We believe that the (single objective) expected error in Equation (6.2.1) is more natural from the machine learning perspective.

Remark 6.2.1. As all upper/lower bounds in this paper are proved by bounding the expected error of estimating the frequency a single item, $\mathbf{E}[|\tilde{f}_i - f_i|]$, then using linearity of expectation, in fact we obtain analogous bounds for any query distribution $(p_i)_{i\in[n]}$. In particularly this means that the bounds of Table 6.4.1 for CM and CS hold for any query distribution. For L-CM and L-CS the factor of $\log(n/B)/\log n$ gets replaced by $\sum_{i=B_h+1}^{n} p_i$ where $B_h = \Theta(B)$ is the number of buckets reserved for heavy hitters.

6.2.1. Algorithms for Frequency Estimation

In this section, we recap three variants of hashing-based algorithms for frequency estimation.

Count-Min. We have k distinct hash functions $h_i : U \to [B]$ and an array C of size $k \times B$. The algorithm maintains C, such that at the end of the stream we have $C[\ell, b] = \sum_{j:h_\ell(j)=b} f_j$. For each $i \in U$, the frequency estimate $\tilde{f}_i$ is equal to $\min_{\ell \le k} C[\ell, h_\ell(i)]$, and always satisfies $\tilde{f}_i \ge f_i$.

Count-Sketch. Similarly to Count-Min, we have k distinct hash functions $h_i : U \to [B]$ and an array C of size $k \times B$. Additionally, in Count-Sketch, we have k sign functions $g_i : U \to \{-1, 1\}$, and the algorithm maintains C such that $C[\ell, b] = \sum_{j:h_\ell(j)=b} f_j \cdot g_\ell(j)$. For each $i \in U$, the frequency estimate $\tilde{f}_i$ is equal to the median of $\{g_\ell(i) \cdot C[\ell, h_\ell(i)]\}_{\ell \le k}$. Note that unlike the previous two methods, here we may have $\tilde{f}_i < f_i$.

6.2.2. Zipfian Distribution

In our analysis we assume that the frequency distribution of items follows Zipf's law. That is, if we sort the items according to their frequencies with no loss of generality assuming that $f_1 \ge f_2 \ge \cdots \ge f_n$, then for any $j \in [n]$, $f_j \propto 1/j$. Given that the frequencies of items follow Zipf's law and assuming that the query distribution is the same as the distribution of the frequency of items in the input stream (i.e., $\Pr_{i \sim \mathcal{D}}[i^*] = f_{i^*}/N = 1/(i^* \cdot H_n)$ where H_n denotes the n-th harmonic number), we can write the expected error defined in (6.2.1) as follows:

$$\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{\mathcal{A}}) = \mathbf{E}_{i \sim \mathcal{D}}[|f_i - \tilde{f}_i|] = \frac{1}{N} \cdot \sum_{i \in [n]} |\tilde{f}_i - f_i| \cdot f_i = \frac{1}{H_n} \cdot \sum_{i \in [n]} |\tilde{f}_i - f_i| \cdot \frac{1}{i} \tag{6.2.2}$$

Throughout this paper, we present our results with respect to the objective function in the right hand side of (6.2.2), i.e., $\frac{1}{H_n} \cdot \sum_{i=1}^{n} |\tilde{f}_i - f_i| \cdot f_i$. We further assume that in fact $f_i = 1/i$. At first sight this assumption may seem strange since it says that item i appears a non-integral number of times in the stream. This is however just a matter of scaling and the assumption is convenient as it removes the dependence on the length of the stream in

Algorithm 26 Learning-Based Frequency Estimation

```
1: procedure LearnedSketch(B, B_r, HH, SketchAlg)
2:     for each stream element i do
3:         if HH(i) = 1 then ▷ if i is a heavy item
4:             if i is already stored in a unique bucket then
5:                 count_i ← count_i + 1
6:             else
7:                 initialize count_i = 1
8:         else
9:             feed i to SketchAlg
```

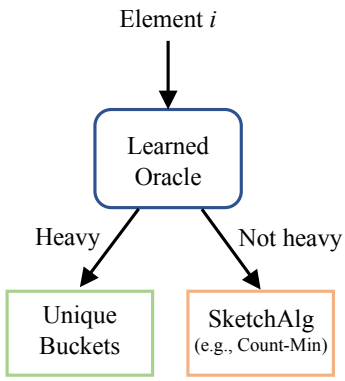


Figure 6.3.1: Pseudo-code and block-diagram representation of our algorithms our bounds.

6.3. Learning-Based Frequency Estimation Algorithms

We aim to develop frequency estimation algorithms that exploit data properties for better performance. To do so, we learn an oracle that identifies heavy hitters, and use the oracle to assign each heavy hitter its unique bucket to avoid collisions. Other items are simply hashed using any classic frequency estimation algorithm (e.g., Count-Min, or Count-Sketch), as shown in the block-diagram in Figure 6.3.1. This design has two useful properties: first, it allows us to augment a classic frequency estimation algorithm with learning capabilities, producing a learning-based counterpart that inherits the original guarantees of the classic algorithm. For example, if the classic algorithm is Count-Min, the resulting learning-based algorithm never underestimates the frequencies. Second, it provably reduces the estimation errors, and for the case of Count-Min it is (asymptotically) optimal.

Algorithm 26 provides pseudo code for our design. The design assumes an oracle $\mathsf{HH}(i)$ that attempts to determine whether an item i is a "heavy hitter" or not. All items classified as heavy hitters are assigned to one of the B_r *unique* buckets reserved for heavy items. All other items are fed to the remaining $B - B_r$ buckets using a conventional frequency estimation algorithm SketchAlg (e.g., Count-Min or Count-Sketch).

The estimation procedure is analogous. To compute $\tilde{f}_i$, the algorithm first checks whether i is stored in a unique bucket, and if so, reports its count. Otherwise, it queries the SketchAlg procedure. Note that if the element is stored in a unique bucket, its reported count is exact, i.e., $\tilde{f}_i = f_i$.

Algorithm	Expected Error	
	$k = 1$	$k > 1$
Count-Min	$\Theta(\frac{\log n}{B})$	$\Theta(\frac{k \log(\frac{kn}{B})}{B})$
Learned Count-Min	$\Theta(\frac{\log^2(\frac{n}{B})}{B \log n})$	$\Omega(\frac{\log^2(\frac{n}{B})}{B \log n})$
Count-Sketch	$\Theta(\frac{\log B}{B})$	$\Omega(\frac{k^{1/2}}{B \log k})$ and $O(\frac{k^{1/2}}{B})$
Learned Count-Sketch	$\Theta(\frac{\log n(n/B)}{B \log n})$	$\Omega(\frac{\log(\frac{n}{B})}{B \log n})$

Table 6.4.1: The performance of different algorithms on streams with frequencies obeying Zipf Law. k is the number of hash functions, B is the number of buckets, and n is the number of distinct elements. The space complexity of all algorithms is $\Theta(B)$.

The oracle is constructed using machine learning and trained with a small subset of S, call it S'. Note that the oracle learns the *properties* that identify heavy hitters as opposed to the *identities* of the heavy hitters themselves. For example, in the case of word frequency, the oracle would learn that shorter words are more frequent, so that it can identify popular words even if they did not appear in the training set S'.

6.4. Analysis

Our algorithms combine simplicity with strong error bounds. Below, we summarize our theoretical results, and leave all theorems, lemmas, and proofs to the appendix. In particular, Table 6.4.1 lists the results proven in this chapter, where each row refers to a specific streaming algorithm and its corresponding error bound.

First, we show that if the heavy hitter oracle is accurate, then the error of the learned variant of Count-Min and Count-Sketch are up to logarithmic factors smaller than that of their standard counterparts.

Second, we show that, asymptotically, our learned Count-Min algorithm cannot be improved any further by designing a better hashing scheme. Specifically, for the case of Learned Count-Min with a perfect oracle, our design achieves the same asymptotic error as the "Ideal Count-Min", which optimizes its hash function for the given input.

We remark that for both Count-Min and Count-Sketch we aim at analyzing the expected value of the variable $\sum_{i \in [n]} f_i \cdot |\tilde{f}_i - f_i|$ where $f_i = 1/i$ and $\tilde{f}_i$ is the estimate of f_i output by

the relevant sketching algorithm. Throughout this paper we use the following notation: for an event E we denote by $[E]$ the random variable in $\{0,1\}$ which is 1 if and only if E occurs.

6.4.1. Tight Bounds for Count-Min with Zipfians

We begin by presenting our tight analysis of Count-Min with Zipfians. The main theorem is the following.

Theorem 6.4.1. *Let $n, B, k \in \mathbb{N}$ with $k \geq 2$ and $B \leq n/k$. Let further $h_1, \ldots, h_k : [n] \to [B]$ be independent and truly random hash functions. For $i \in [n]$, define the random variable $\tilde{f}_i = \min_{\ell \in [k]} \left(\sum_{j \in [n]} [h_\ell(j) = h_\ell(i)] f_j \right)$. For any $i \in [n]$ it holds that*

$$\mathbf{E}[|\tilde{f}_i - f_i|] = \Theta\left(\frac{\log\left(\frac{n}{B}\right)}{B} \right)$$

Replacing B by B/k in Theorem 6.4.1 and using linearity of expectation we obtain the desired bound for Count-Min in the upper right hand side of Table 6.4.1. The natural assumption that $B \leq n/k$ simply says that the total number of buckets is upper bounded by the number of items.

To prove Theorem 6.4.1 we start with the following special case of the theorem.

Lemma 6.4.2. *Suppose that we are in the setting of Theorem 6.4.1 and further that*[1] *$n = B$. Then*

$$\mathbf{E}[|\tilde{f}_i - f_i|] = O\left(\frac{1}{n} \right).$$

Proof: It suffices to show the result when $k = 2$ since adding more hash functions and corresponding tables only decreases the value of $|\tilde{f}_i - f_i|$. Define $Z_\ell = \sum_{j \in [n] \setminus \{i\}} [h_\ell(j) = h_\ell(i)] f_j$ for $\ell \in [2]$ and note that these variables are independent. For a given $t \geq 3/n$ we wish to upper bound $\Pr[Z_\ell \geq t]$. Let $s < t$ and note that if $Z_\ell \geq t$ then either of the following two events must hold:

E_1: There exists a $j \in [n] \setminus \{i\}$ with $f_j > s$ and $h_\ell(j) = h_\ell(i)$.

[1] In particular we dispose with the assumption that $B \leq n/k$.

E_2: The set $\{j \in [n] \setminus \{i\} : h_\ell(j) = h_\ell(i)\}$ contains at least t/s elements.

Union bounding we find that

$$\Pr[Z_\ell \geq t] \leq \Pr[E_1] + \Pr[E_2] \leq \frac{1}{ns} + \binom{n}{t/s} n^{-t/s} \leq \frac{1}{ns} + \left(\frac{es}{t}\right)^{t/s}.$$

Choosing $s = \frac{t}{\log(tn)}$, a simple calculation yields that $\Pr[Z_\ell \geq t] = O\left(\frac{\log(tn)}{tn}\right)$. As Z_1 and Z_2 are independent, $\Pr[Z \geq t] = O\left(\left(\frac{\log(tn)}{tn}\right)^2\right)$, so

$$\mathbf{E}[Z] = \int_0^\infty \Pr[Z \geq t]\, dt \leq \frac{3}{n} + O\left(\int_{3/n}^\infty \left(\frac{\log(tn)}{tn}\right)^2 dt\right) = O\left(\frac{1}{n}\right). \qquad \square$$

Before proving the full statement of Theorem 6.4.1 we recall Bennett's inequality.

Theorem 6.4.3 (Bennett's inequality [30]). *Let $X_1, \ldots, X_n$ be independent, mean zero random variables. Let $S = \sum_{i=1}^n X_i$, and $\sigma^2, M > 0$ be such that $\mathbf{Var}[S] \leq \sigma^2$ and $|X_i| \leq M$ for all $i \in [n]$. For any $t \geq 0$,*

$$\Pr[S \geq t] \leq \exp\left(-\frac{\sigma^2}{M^2} h\left(\frac{tM}{\sigma^2}\right)\right),$$

where $h : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is defined by $h(x) = (x+1)\log(x+1) - x$. The same tail bound holds on the probability $\Pr[S \leq -t]$.

Remark 6.4.4. It is well known and easy to check that for $x \geq 0$,

$$\frac{1}{2} x \log(x+1) \leq h(x) \leq x \log(x+1).$$

We will use these asymptotic bounds repeatedly in this paper.

Proof of Theorem 6.4.1. We start out by proving the upper bound. Let $N_1 = [B] \setminus \{i\}$ and $N_2 = [n] \setminus ([B] \cup \{i\})$. Let $b \in [k]$ be such that $\sum_{j \in N_1} f_j \cdot [h_b(j) = h_b(i)]$ is minimized. Note

that b is itself a random variable. We also define

$$Y_1 = \sum_{j \in N_1} f_j \cdot [h_b(j) = h_b(i)],$$
$$Y_2 = \sum_{j \in N_2} f_j \cdot [h_b(j) = h_b(i)]$$

Clearly $|\tilde{f}_i - f_i| \leq Y_1 + Y_2$. Using Lemma 6.4.2, we obtain that $\mathbf{E}[Y_1] = O(\frac{1}{B})$. For Y_2 we observe that

$$\mathbf{E}[Y_2 \mid b] = \sum_{j \in N_2} \frac{f_j}{B} = O\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right).$$

We conclude that

$$\mathbf{E}[|\tilde{f}_i - f_i|] \leq \mathbf{E}[Y_1] + \mathbf{E}[Y_2] = \mathbf{E}[Y_1] + \mathbf{E}[\mathbf{E}[Y_2 \mid b]] = O\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right),$$

as desired.

Next we show the lower bound. For $j \in [n]$ and $\ell \in [k]$ we define $X_\ell^{(j)} = f_j \cdot \left([h_\ell(j) = h_\ell(i)] - \frac{1}{B}\right)$. Note that the variables $(X_\ell^{(j)})_{j \in [n]}$ are independent. We also define $S_\ell = \sum_{j \in N_2} X_\ell^{(j)}$ for $\ell \in [k]$. Observe that $|X_\ell^{(j)}| \leq f_j \leq \frac{1}{B}$ for $j \geq B$, $\mathbf{E}[X_\ell^{(j)}] = 0$, and that

$$\mathbf{Var}[S_\ell] = \sum_{j \in N_2} f_j^2 \left(\frac{1}{B} - \frac{1}{B^2}\right) \leq \frac{2}{B^2}.$$

Applying Bennett's inequality with $\sigma^2 = \frac{2}{B^2}$ and $M = 1/B$ thus gives that

$$\Pr[S_\ell \leq -t] \leq \exp\left(-2h\left(\frac{tB}{2}\right)\right).$$

Defining $W_\ell = \sum_{j \in N_2} f_j \cdot [h_\ell(j) = h_\ell(i)]$ it holds that $\mathbf{E}[W_\ell] = \Theta\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right)$ and $S_\ell = W_\ell - \mathbf{E}[W_\ell]$, so putting $t = \mathbf{E}[W_\ell]/2$ in the inequality above we obtain that

$$\Pr[W_\ell \leq \mathbf{E}[W_\ell]/2] = \Pr[S_\ell \leq -\mathbf{E}[W_\ell]/2] \leq \exp\left(-2h\left(\Omega\left(\log\frac{n}{B}\right)\right)\right).$$

Appealing to Remark 6.4.4 and using that $B \leq n/k$ the above bound becomes

$$\begin{aligned}\Pr[W_\ell \leq \mathbf{E}[W_\ell]/2] &\leq \exp\left(-\Omega\left(\log\frac{n}{B}\cdot\log\left(\log\frac{n}{B}+1\right)\right)\right)\\ &= \exp(-\Omega(\log k\cdot\log(\log k+1))) = k^{-\Omega(\log(\log k+1))}. \qquad (6.4.1)\end{aligned}$$

By the independence of the events $(W_\ell > E[W_\ell]/2)_{\ell\in[k]}$ we have that

$$\Pr\left[|\tilde{f}_i - f_i| \geq \frac{\mathbf{E}[W_\ell]}{2}\right] \geq (1-k^{-\Omega(\log(\log k+1))})^k = \Omega(1),$$

and so $\mathbf{E}[|\tilde{f}_i - f_i|] = \Omega(\mathbf{E}[W_\ell]) = \Omega\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right)$, as desired. □

6.4.2. Learned Count-Min with Zipfians

One hash function. By the tight analysis of Count-Min, we have the following theorem on the expected error of the learned Count-Min with $k = 1$. By taking $B_1 = B_h = \Theta(B)$ and $B_2 = B - B_h = \Theta(B)$ in the theorem below the result on learned Count-Min for $k = 1$ claimed in Table 6.4.1 follows immediately.

Theorem 6.4.5. *Let $n, B_1, B_2 \in \mathbb{N}$ and let $h : [n] \setminus [B_1] \to [B_2]$ be an independent and truly random hash function. For $i \in [n] \setminus [B_1]$, define the random variable $\tilde{f}_i = \sum_{j\in[n]}[h(j) = h(i)]f_j$. For any $i \in [n] \setminus [B_1]$ it holds that*

$$\mathbf{E}[|\tilde{f}_i - f_i|] = \Theta\left(\frac{\log\left(\frac{n}{B_1}\right)}{B_2}\right)$$

Multiple hash functions. Next we compute an asymptotic lower bound on the expected error of the Count-Min no matter how the hash functions are picked. In particular, this implies a lower bound on the expected error of the learned Count-Min.

Claim 6.4.6. *For a set of items I, let $f(I)$ denote the total frequency of all items in I; $f(I) := \sum_{i\in I} f_i$. In any hash function of form $\{h : I \to [B_I]\}$ with minimum expected error, any item with frequency at least $\frac{f(I)}{B_I}$ does not collide with any other items in I.*

Proof: For each bucket $b \in [B_I]$, lets $f_h(b)$ denotes the frequency of items mapped to b under

h; $f(b) := \sum_{i \in I : h(i)=b} f_i$. Then we can rewrite $\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_h(I))$ as

$$\begin{aligned}\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_h(I)) = \sum_{i \in I} f_i \cdot (f(h(i)) - f_i) &= \sum_{i \in I} f_i \cdot f(h(i)) - \sum_{i \in I} f_i^2 \\ &= \sum_{b \in [B_I]} f(b)^2 - \sum_{i \in I} f_i^2. \end{aligned} \tag{6.4.2}$$

Note that in (6.4.2) the second term is independent of h and is a constant. Hence, an optimal hash function minimizes the first term, $\sum_{b \in B_I} f(b)^2$.

Suppose that an item i^* with frequency at least $\frac{f(I)}{B_I}$ collides with a (non-empty) set of items $I^* \subseteq I \setminus \{i^*\}$ under an optimal hash function h^*. Since the total frequency of the items mapped to the bucket b^* containing i^* is greater than $\frac{f(I)}{B_I}$ (i.e., $f(h(i^*)) > \frac{f(I)}{B_I}$), there exists a bucket $\bar{b}$ such that $f(\bar{b}) < \frac{f(I)}{B_I}$. Next, we define a new hash function $\overline{h}$ with smaller estimation error compared to h^* which contradicts the optimality of h^*:

$$\overline{h}(i) = \begin{cases} h^*(i) & \text{if } i \in I \setminus I^* \\ \bar{b} & \text{otherwise.} \end{cases}$$

Formally,

$$\begin{aligned}\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_{h^*}(I)) - \mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_{\overline{h}}(I)) &= f_{h^*}(b^*)^2 + f_{h^*}(\bar{b})^2 - f_{\overline{h}}(b^*)^2 - f_{\overline{h}}(\bar{b})^2 \\ &= (f_{i^*} + f(I^*))^2 + f_{h^*}(\bar{b})^2 - f_{i^*}^2 - (f_{h^*}(\bar{b}) + f(I^*))^2 \\ &= 2f_{i^*} \cdot f(I^*) - 2f_{h^*}(\bar{b}) \cdot f(I^*) \\ &= 2f(I^*) \cdot (f_{i^*} - f_{h^*}(\bar{b})) \\ &> 0 \quad \triangleright \text{ Since } f_{i^*} > \frac{f(I)}{B} > f_{h^*}(\bar{b}). \end{aligned}$$

□

Next, we show that in any optimal hash function $h^* : [n] \to [B]$ and assuming Zipfian distribution, $\Theta(B)$ most frequent items do not collide with any other items under h^*.

Lemma 6.4.7. *Suppose that $B = n/\gamma$ where γ is a large enough constant and lets assume that items follow Zipfian distribution. In any hash function $h^* : [n] \to [B]$ with minimum expected error, none of the $\frac{B}{2 \ln \gamma}$ most frequent items collide with any other items (i.e., they*

are mapped to a singleton bucket).

Proof: Let i_{j^*} be the most frequent item that is not mapped to a singleton bucket under h^*. If $j^* > \frac{B}{2\ln\gamma}$ then the statement holds. Suppose it is not the case and $j^* \leq \frac{B}{2\ln\gamma}$. Let I denote the set of items with frequency at most $f_{j^*} = 1/j^*$ (i.e., $I = \{i_j \mid j \geq j^*\}$) and let B_I denote the number of buckets that the items with index at least j^* mapped to; $B_I = B - (j^* - 1)$. It is straightforward to show that $f(I) < \ln(\frac{n}{j^*}) + 1$. Next, by Claim 6.4.6, we show that h^* does not hash the items $\{j^*, \cdots, n\}$ to B_I optimally. In particular, we show that the frequency of item j^* is more than $\frac{f(I)}{B_I}$. To prove this, first we observe that the function $g(j) := j \cdot (\ln(n/j) + 1)$ is strictly increasing in $[1, n]$. Hence, for any $j^* \leq \frac{B}{2\ln\gamma}$,

$$\begin{aligned} j^* \cdot (\ln(\frac{n}{j^*}) + 1) &\leq \frac{B}{2\ln\gamma} \cdot (\ln(2\gamma\ln\gamma) + 1) \\ &\leq B \cdot (1 - \frac{1}{2\ln\gamma}) \quad \triangleright \text{ Since } \ln(2\ln\gamma) + 2 < \ln\gamma \text{ for sufficiently large } \gamma \\ &< B_I \end{aligned}$$

Thus, $f_{j^*} = \frac{1}{j^*} > \frac{\ln(n/j^*)+1}{B_I} > \frac{f(I)}{B_I}$ which implies that an optimal hash function must map j^* to a singleton bucket. □

Theorem 6.4.8. *If n/B is sufficiently large, then the expected error of any hash function that maps a set of n items following Zipfian distribution to B buckets is $\Omega(\frac{\ln^2(n/B)}{B})$.*

Proof: Let $\gamma = n/B$. By Lemma 6.4.7, in any hash function with minimum expected error, the $(\frac{B}{2\ln\gamma})$ most frequent items do not collide with any other items (i.e., they are mapped into a singleton bucket) where γ is a sufficiently large constant.

Hence, the goal is to minimize (6.4.2) for the set of items I which consist of all items other than the $(\frac{B}{2\ln\gamma})$ most frequent items. Since the sum of squares of m items that summed to S is at least $\frac{S^2}{m}$, the expected error of any optimal hash function is at least:

$$
\begin{aligned}
\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_{h^*}(I)) &= \sum_{b\in[B]} f(b)^2 - \sum_{i\in[n]} f_i^2 && \triangleright \text{ by (6.4.2)} \\
&\geq \frac{(\sum_{i\in I} f_i)^2}{B(1-\frac{1}{2\ln\gamma})} - \sum_{i\in I} f_i^2 \\
&\geq \frac{(\ln(2\gamma\ln\gamma)-1)^2}{B} - \frac{2\ln\gamma}{B} + \frac{1}{n} \\
&= \Omega(\frac{\ln^2\gamma}{B}) && \triangleright \text{ for sufficiently large } \gamma \\
&= \Omega(\frac{\ln^2(n/B)}{B}). && \square
\end{aligned}
$$

Next, we show a more general statement which basically shows that the estimation error of any Count-Min with B buckets is $\Omega(\frac{\ln^2(n/B)}{B})$ no matter how many rows it has.

Theorem 6.4.9. *If n/B is sufficiently large, then the estimation error of any Count-Min that maps a set of n items following Zipfian distribution to B buckets is $\Omega(\frac{\ln^2(n/B)}{B})$.*

Proof: We prove the statement by showing a reduction that given a Count-Min $CM(k)$ with hash functions $h_1, \cdots, h_k \in \{h : [n] \to [B/k]\}$ constructs a single hash function $h^* : [n] \to [B]$ whose estimation error is less than or equal to the estimation error of $CM(k)$.

For each item i, we define $C'[i]$ to be the bucket whose value is returned by $CM(k)$ as the estimate of f_i; $C'[i] := \mathrm{argmin}_{j\in[k]} C[j, h_j(i)]$. Since the total number of buckets in $CM(k)$ is B, $|\{C'[i] \mid i \in [n]\}| \leq B$; in other words, we only consider the subset of buckets that $CM(k)$ uses to report the estimates of $\{f_i \mid i \in [n]\}$ which trivially has size at most B. We define h^* as follows:

$$
h^*(i) = (j^*, h_{j^*}(i)) \quad \triangleright \text{ for each } i \in [n], \text{ where } j^* = \mathrm{argmin}_{j\in[k]} C[j, h_j(i)]
$$

Unlike $CM(k)$, h^* maps each item to exactly one bucket in $\{C[\ell, j] \mid \ell \in [k], j \in [B/k]\}$; hence, for each item i, $C'[h^*(i)] \leq C[h^*(i)] = \tilde{f}_i$ where $\tilde{f}_i$ is the estimate of f_i returned by $CM(k)$. Moreover, since for each i, $C'[h^*(i)] \geq f_i$, $\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{h^*}) \leq \mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{CM(k)})$. Finally, by Theorem 6.4.8, the estimation error of h^* is $\Omega(\frac{\ln^2(n/B)}{B})$ which implies that the estimation error of $CM(k)$ is $\Omega(\frac{\ln^2(n/B)}{B})$ as well. $\square$

6.4.3. (Nearly) tight Bounds for Count-Sketch with Zipfians

In this section we proceed to analyze Count-Sketch for Zipfians either using a single or more hash functions. We start with two simple lemmas which for certain frequencies $(f_i)_{i\in[n]}$ of the items in the stream can be used to obtain respectively good upper and lower bounds on $\mathbf{E}[|\tilde{f}_i - f_i|]$ in Count-Sketch with a single hash function. We will use these two lemmas both in our analysis of standard and learned Count-Sketch for Zipfians.

Lemma 6.4.10. *Let $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$, $\eta_1, \ldots, \eta_n$ independent Bernoulli variables taking value 1 with probability p, and $\sigma_1, \ldots, \sigma_n \in \{-1, 1\}$ independent Rademachers, i.e., $\Pr[\eta_i = 1] = \Pr[\eta_i = -1] = 1/2$. Let $S = \sum_{i=1}^n w_i\eta_i\sigma_i$. Then*

$$\mathbf{E}[|S|] = O\left(\sqrt{p}\|w\|_2\right).$$

Proof: Using that $\mathbf{E}[\sigma_i\sigma_j] = 0$ for $i \neq j$ and Jensen's inequality

$$\mathbf{E}[|S|]^2 \leq \mathbf{E}[S^2] = \mathbf{E}\left[\sum_{i=1}^n w_i^2\eta_i\right] = p\|w\|_2^2,$$

from which the result follows. □

Lemma 6.4.11. *Suppose that we are in the setting of Lemma 6.4.10. Let $I \subset [n]$ and let $w_I \in \mathbb{R}^n$ be defined by $(w_I)_i = [i \in I] \cdot w_i$. Then*

$$\mathbf{E}[|S|] \geq \frac{1}{2}p\left(1-p\right)^{|I|-1}\|w_I\|_1.$$

Proof: Let $J = [n] \setminus I$, $S_1 = \sum_{i\in I} w_i\eta_i\sigma_i$, and $S_2 = \sum_{i\in J} w_i\eta_i\sigma_i$. Let E denote the event that S_1 and S_2 have the same sign or $S_2 = 0$. Then $\Pr[E] \geq 1/2$ by symmetry. For $i \in I$ we denote by A_i the event that $\{j \in I : \eta_j \neq 0\} = \{i\}$. Then $\Pr[A_i] = p(1-p)^{|I|-1}$ and furthermore A_i and E are independent. If $A_i \cap E$ occurs, then $|S| \geq |w_i|$ and as the events $(A_i \cap E)_{i\in I}$ are disjoint it thus follows that

$$\mathbf{E}[|S|] \geq \sum_{i\in I}\Pr[A_i \cap E] \cdot |w_i| = \frac{1}{2}p\left(1-p\right)^{|I|-1}\|w_I\|_1.$$ □

One hash-function. We are now ready to commence our analysis of Count-Sketch for Zipfians. As in the discussion succeeding Theorem 6.4.1 the following theorem yields the desired result for a single hash function as presented in Table 6.4.1.

Theorem 6.4.12. *Suppose that $B \leq n$ and let $h : [n] \to [B]$ and $s : [n] \to \{-1, 1\}$ be truly random hash functions. Define the random variable $\tilde{f}_i = \sum_{j \in [n]} [h(j) = h(i)] s(j) f_j$ for $i \in [n]$. Then*

$$\mathbf{E}[|\tilde{f}_i - s(i) f_i|] = O\left(\frac{\log B}{B}\right).$$

Proof: Let $i \in [n]$ be fixed. We start by defining $N_1 = [B] \setminus \{i\}$ and $N_2 = [n] \setminus ([B] \cup \{i\})$ and note that

$$|\tilde{f}_i - s(i) f_i| \leq \left| \sum_{j \in N_1} [h(j) = h(i)] s(j) f_j \right| + \left| \sum_{j \in N_2} [h(j) = h(i)] s(j) f_j \right| := X_1 + X_2.$$

Using the triangle inequality

$$\mathbf{E}[X_1] \leq \frac{1}{B} \sum_{j \in N_1} f_j = O\left(\frac{\log B}{B}\right).$$

Also, by Lemma 6.4.10, $\mathbf{E}[X_2] = O\left(\frac{1}{B}\right)$ and combining the two bounds we obtain the desired upper bound.

For the lower bound we apply Lemma 6.4.11 with $I = N_1$ concluding that

$$\mathbf{E}[|\tilde{f}_i - s(i) f_i|] \geq \frac{1}{2B} \left(1 - \frac{1}{B}\right)^{|N_1|-1} \sum_{i \in N_1} f_i = \Omega\left(\frac{\log B}{B}\right). \qquad \square$$

Multiple hash functions. Let $k \in \mathbb{N}$ be odd. For a tuple $x = (x_1, \ldots, x_k) \in \mathbb{R}^k$ we denote by $\mathsf{median}\, x$ the median of the entries of x. The following theorem immediately leads to the result on CS with $k \geq 3$ hash functions claimed in Table 6.4.1.

Theorem 6.4.13. *Let $k \geq 3$ be odd, $n \geq kB$ and all $h_1, \ldots, h_k : [n] \to [B]$ and $s_1, \ldots, s_k : [n] \to \{-1, 1\}$ be truly random hash functions. For each $i \in [n]$, define the random variable*

$\tilde{f}_i = \mathsf{median}_{\ell\in[k]}\left(\sum_{j\in[n]}[h_\ell(j)=h_\ell(i)]s_\ell(j)f_j\right)$. *Assume that*[2] $k \le B$. *Then*

$$\mathbf{E}[|\tilde{f}_i - s(i)f_i|] = \Omega\left(\frac{1}{B\sqrt{k}\log k}\right), \quad \textit{and} \quad \mathbf{E}[|\tilde{f}_i - s(i)f_i|] = O\left(\frac{1}{B\sqrt{k}}\right)$$

The assumption $n \ge kB$ simply says that the total number of buckets is upper bounded by the number of items. Again using linearity of expectation for the summation over $i \in [n]$ and replacing B by B/k we obtain the claimed upper and lower bounds of $\frac{\sqrt{k}}{B\log k}$ and $\frac{\sqrt{k}}{B}$ respectively. We note that even if the bounds above are only tight up to a factor of $\log k$ they still imply that it is asymptotically optimal to choose $k = O(1)$. To settle the correct asymptotic growth is thus of merely theoretical interest. We need the following claim to the prove the theorem.

Claim 6.4.14. *Let $I \subset \mathbb{R}$ be the closed interval centered at the origin of length $2t$, i.e., $I = [-t,t]$. Suppose that $\frac{1}{\sqrt{kB}} \le t \le \frac{1}{2B}$. For $\ell \in [k]$, $\Pr[X^{(\ell)} \in I] = \Omega(tB)$.*

Proof: We first show that with probability $\Omega(1)$, $X_2^{(\ell)}$ lies in the interval $[1/B, \gamma/B]$ for some constant γ. To see this we note that by Lemma 6.4.10, $\mathbf{E}[|X_2^{(\ell)}|] = O\left(\frac{1}{B}\right)$, so it follows by Markov's inequality that if $\gamma = O(1)$ is large enough, the probability that $|X_2^{(\ell)}| \ge \gamma/B$ is at most $\frac{1}{200}$. For a constant probability lower bound on $|X_2^{(\ell)}|$ we write

$$X_2^{(\ell)} = \sum_{j\in N_2\cap\{B+1,\ldots,2B\}} [h_\ell(j)=h_\ell(i)]s_\ell(j)f_j + \sum_{j\in N_2\cap\{2B+1,\ldots,n\}} [h_\ell(j)=h_\ell(i)]s_\ell(j)f_j := S_1 + S_2.$$

Condition on S_2. If $B \ge 4$ the probability that there exist exactly two $j \in N_2 \cap \{B+1,\ldots,2B\}$ with $h_\ell(j) = h_\ell(i)$ is at least

$$\binom{|N_2\cap\{B+1,\ldots,2B\}|}{2}\frac{1}{B^2}\left(1-\frac{1}{B}\right)^{B-2} \ge \binom{B-1}{2}\frac{1}{eB^2} \ge \frac{1}{8e}.$$

With probability $1/4$ the corresponding signs $s_\ell(j)$ are both the same as that of S_2. By independence of s_ℓ and h_ℓ the probability that this occurs is at least $\frac{1}{32e}$ and if it does, $|X_2^{(\ell)}| \ge 1/B$. Combining these two bounds it follows that $|X_2^{(\ell)}| \in [\frac{1}{B}, \frac{\gamma}{B}]$ with probability

[2] This very mild assumption can probably be removed at the cost of a more technical proof. In our proof it can even be replaced by $k \le B^{2-\varepsilon}$ for any $\varepsilon = \Omega(1)$.

at least $\frac{1}{32e} - \frac{1}{200} \geq \frac{1}{200}$. By symmetry, $\Pr[X_2^{(\ell)} \in [\frac{1}{B}, \frac{\gamma}{B}]] \geq \frac{1}{400} = \Omega(1)$. Denote this event by E. Also let F be the event that $|\{j \in N_1 : h_\ell(j) = h_\ell(i)\}| = 1$. Then $\Pr[F] = \Omega(1)$ and as E and F are independent $\Pr[E \cap F] = \Omega(1)$. Conditioned on $E \cap F$ we now lower bound the probability that $X^{(\ell)} \in I$. For this it suffices to fix $X_2^{(\ell)} = \frac{\eta}{B}$ for some $1 \leq \eta \leq \gamma$ and lower bound the probability that $\frac{\eta}{B} + \sigma f \in [-t, t]$ where σ is a Rademacher and $f \in \{1/j : j \in N_1\}$ is chosen uniformly at random. Let $m_1, m_2 \in \mathbb{R}_{>0}$ be such that

$$\frac{\eta}{B} - \frac{1}{m_1} = -t \quad \text{and} \quad \frac{\eta}{B} - \frac{1}{m_2} = t.$$

Then $m_2 - m_1 = B\frac{2tB}{\eta^2 - t^2B^2} = \Omega(tB^2)$. Using that B is larger than a big enough constant and the mild assumption $k \leq B$, we have that $m_2 - m_1 = \Omega(B/\sqrt{k}) = \Omega(\sqrt{B}) \geq 1$, and so $\lfloor m_2 - m_1 \rfloor = \Omega(m_2 - m_1) = \Omega(tB^2)$ as well. As we have $|N_2| \geq B - 1$ options for f it follows that

$$\Pr\left[\frac{\eta}{B} + \sigma f \in [-t, t]\right] \geq \frac{\lfloor m_2 - m_1 \rfloor}{B - 1} = \Omega(tB),$$

as desired. □

Proof of Theorem 6.4.13. If B (and hence k) is a constant the result follows easily from Lemma 6.4.10, so in what follows we assume that B is larger than a sufficiently large constant.

We first prove the upper bound. Define $N_1 = [B] \setminus \{i\}$ and $N_2 = [n] \setminus ([B] \cup \{i\})$. Let for $\ell \in [k]$, $X_1^{(\ell)} = \sum_{j \in N_1} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j$ and $X_2^{(\ell)} = \sum_{j \in N_2} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j$. Finally write $X^{(\ell)} = X_1^{(\ell)} + X_2^{(\ell)}$.

As the absolute error in Count-Sketch with one pair of hash functions (h, s) is always upper bounded by the corresponding error in Count-Min with the single hash function h, we can use the bound in the proof of Lemma 6.4.2 to conclude that

$$\Pr[|X_1^{(\ell)}| \geq t] = O\left(\frac{\log(tB)}{tB}\right),$$

when $t \geq 3/B$. Also

$$\mathbf{Var}[X_2^{\ell}] = \left(\frac{1}{B} - \frac{1}{B^2}\right) \sum_{j \in N_2} f_j^2 \leq \frac{2}{B^2},$$

so by Bennett's inequality (with $M = 1/B$ and $\sigma^2 = 2/B^2$) and Remark 6.4.4,

$$\Pr[|X_2^{(\ell)}| \geq t] \leq 2\exp\left(-2h(tB/2)\right) \leq 2\exp\left(-\frac{1}{2}tB\log\left(\frac{tB}{2}+1\right)\right) = O\left(\frac{\log(tB)}{tB}\right),$$

for $t \geq \frac{3}{B}$. It follows that for $t \geq 3/B$,

$$\Pr[|X^{(\ell)}| \geq 2t] \leq \Pr[(|X_1^{(\ell)}| \geq t)] + \Pr(|X_2^{(\ell)}| \geq t)] = O\left(\frac{\log(tB)}{tB}\right).$$

Let C be the implicit constant in the O-notation above. If $|\tilde{f}_i - s(i)f_i| \geq 2t$, at least half of the values $(|X^{(\ell)}|)_{\ell\in[k]}$ are at least $2t$. For $t \geq 3/B$ it thus follows by a union bound that

$$\Pr[|\tilde{f}_i - s(i)f_i| \geq 2t] \leq 2\binom{k}{\lceil k/2\rceil}\left(C\frac{\log(tB)}{tB}\right)^{\lceil k/2\rceil} \leq 2\left(4C\frac{\log(tB)}{tB}\right)^{\lceil k/2\rceil}. \tag{6.4.3}$$

If $\alpha = O(1)$ is chosen sufficiently large it thus holds that

$$\begin{aligned}\int_{\alpha/B}^{\infty}\Pr[|\tilde{f}_i - s(i)f_i| \geq t]\,dt &= 2\int_{\alpha/(2B)}^{\infty}\Pr[|\tilde{f}_i - s(i)f_i| \geq 2t]\,dt \\ &\leq \frac{4}{B}\int_{\alpha/2}^{\infty}\left(4C\frac{\log(t)}{t}\right)^{\lceil k/2\rceil}dt \\ &\leq \frac{1}{B2^k} \leq \frac{1}{B\sqrt{k}}.\end{aligned}$$

Here the first inequality uses (6.4.3) and a change of variable. The second inequality uses that $\left(4C\frac{\log t}{t}\right)^{\lceil k/2\rceil} \leq (C'/t)^{2k/5}$ for some constant C' followed by a calculation of the integral. For our upper bound it therefore suffices to show that $\int_0^{\alpha/B}\Pr[|\tilde{f}_i - s(i)f_i| \geq t]\,dt = O\left(\frac{1}{B\sqrt{k}}\right)$.

For this let $\frac{1}{\sqrt{k}B} \leq t \leq \frac{1}{B}$ be fixed. If $|\tilde{f}_i - s(i)f_i| \geq t$, at least half of the values $(X^{(\ell)})_{\ell\in[k]}$ are at least t or at most $-t$. Let us focus on bounding the probability that at least half are at least t, the other bound being symmetric giving an extra factor of 2 in the probability bound. By symmetry and Claim 6.4.14, $\Pr[X^{(\ell)} \geq t] = \frac{1}{2} - \Omega(tB)$. For $\ell \in [k]$ we define $Y_\ell = [X^{(\ell)} \geq t]$, and we put $S = \sum_{\ell\in[k]} Y_\ell$. Then $\mathbf{E}[S] = k\left(\frac{1}{2} - \Omega(tB)\right)$. If at least half of the values $(X^{(\ell)})_{\ell\in[k]}$ are at least t then $S \geq k/2$. By Hoeffding's inequality we can bound

the probability of this event by

$$\Pr[S \geq k/2] = \Pr[S - \mathbf{E}[S] = \Omega(ktB)] = \exp(-\Omega(kt^2B^2)).$$

It follows that $\Pr[|\tilde{f}_i - s(i)f_i| \geq t] \leq 2\exp(-\Omega(kt^2B^2))$. Thus

$$\begin{aligned}\int_0^{\alpha/B} \Pr[|\tilde{f}_i - s(i)f_i| \geq t]\,dt &\leq \frac{1}{B\sqrt{k}} + \int_{\frac{1}{B\sqrt{k}}}^{\frac{1}{2B}} 2\exp(-\Omega(kt^2B^2))\,dt + \int_{\frac{1}{2B}}^{\alpha/B} 2\exp(-\Omega(k))\,dt \\ &= O\left(\frac{1}{B\sqrt{k}}\right) + \frac{1}{B\sqrt{k}}\int_1^{\infty} \exp(-t^2)\,dt = O\left(\frac{1}{B\sqrt{k}}\right).\end{aligned}$$

This completes the proof of the upper bound and we proceed with the lower bound. Fix $\ell \in [k]$ and let $M_1 = [B\log k] \setminus \{i\}$ and $M_2 = [n] \setminus ([B\log k] \cup \{i\})$. Write

$$S := \sum_{j \in M_1} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j + \sum_{j \in M_2} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j := S_1 + S_2.$$

We also define $J := \{j \in M_1 : h_\ell(j) = h_\ell(i)\}$. Let $I \subseteq \mathbb{R}$ be the closed interval around $s_\ell(i)f_i$ of length $\frac{1}{B\sqrt{k}\log k}$. We now upper bound the probability that $S \in I$ conditioned on the value of S_2. To ease the notation the conditioning on S_2 has been left out in the notation to follow. Note first that

$$\Pr[S \in I] = \sum_{r=0}^{|M_1|} \Pr[S \in I \mid |J| = r] \cdot \Pr[|J| = r].$$

For a given $r \geq 1$ we now proceed to bound $\Pr[S \in I \mid |J| = r]$. This probability is the same as the probability that $S_2 + \sum_{j \in R} \sigma_j f_j \in I$, where $R \subseteq M_1$ is a uniformly random r-subset and the σ_j's are independent Rademachers. Suppose that we sample the elements from R as well as the corresponding signs $(\sigma_i)_{i \in R}$ sequentially, and let us condition on the values and signs of the first $r-1$ sampled elements. At this point at most $\frac{B\log k}{\sqrt{k}} + 1$ possible samples for the last element in R brings S into I. Indeed, the minimum distance between consecutive points in $\{f_j : j \in M_1\}$ is at most $1/(B\log k)^2$ so at most

$$\frac{1}{B\sqrt{k}\log k} \cdot (B\log k)^2 + 1 = \frac{B\log k}{\sqrt{k}} + 1$$

samples brings S into I. For $1 \le r \le (B \log k)/2$ we can thus can upper bound

$$\Pr[S \in I \mid |J| = r] \le \frac{\frac{B \log k}{\sqrt{k}} + 1}{|M_1| - r + 1} \le \frac{2}{\sqrt{k}} + \frac{2}{B \log k} \le \frac{3}{\sqrt{k}}.$$

By a standard Chernoff bound

$$\Pr[|J| \ge B \log k/2] = \exp\left(-\Omega(B \log k)\right) = k^{-\Omega(B)}.$$

If we assume that B is larger than a constant, then $\Pr[|J| \ge B \log k/2] \le k^{-1}$. Finally, $\Pr[|J| = 0] = (1 - 1/B)^{B \log k} \le k^{-1}$. Combining these three bounds,

$$\begin{aligned}\Pr[S \in I] \le \Pr[|J| = 0] + &\sum_{r=1}^{(B \log k)/2} \Pr[S \in I \mid |J| = r] \cdot \Pr[|J| = r] \\ &+ \sum_{r=(B \log k)/2}^{|M_1|} \Pr[|J| = r] = O\left(\frac{1}{\sqrt{k}}\right),\end{aligned}$$

which holds even after removing the conditioning on S_2. We now show that with probability $\Omega(1)$ at least half the values $(X^{(\ell)})_{\ell \in [k]}$ are at least $\frac{1}{2B\sqrt{k} \log k}$. Let p_0 be the probability that $X^{(\ell)} \ge \frac{1}{2B\sqrt{k} \log k}$. This probability does not depend on $\ell \in [k]$ and by symmetry and what we showed above, $p_0 = 1/2 - O(1/\sqrt{k})$. Define the function $f : \{0, \dots, k\} \to \mathbb{R}$ by

$$f(t) = \binom{k}{t} p_0^t (1 - p_0)^{k-t}.$$

Then $p(t)$ is the probability that exactly t of the values $(X^{(\ell)})_{\ell \in [k]}$ are at least $\frac{1}{B\sqrt{k} \log k}$. Using that $p_0 = 1/2 - O(1/\sqrt{k})$, a simple application of Stirling's formula gives that $f(t) = \Theta\left(\frac{1}{\sqrt{k}}\right)$ for $t = \lceil k/2 \rceil, \dots, \lceil k/2 + \sqrt{k} \rceil$ when k is larger than some constant C. It follows that with probability $\Omega(1)$ at least half the $(X^{(\ell)})_{\ell \in [k]}$ are at least $\frac{1}{B\sqrt{k} \log k}$ and in particular

$$\mathbf{E}[|\tilde{f}_i - f_i|] = \Omega\left(\frac{1}{B\sqrt{k} \log k}\right).$$

Finally we handle the case where $k \le C$. It is easy to check (e.g. with Lemma 6.4.11) that $X^{(\ell)} = \Omega(1/B)$ with probability $\Omega(1)$. Thus this happens for all $\ell \in [k]$ with probability

$\Omega(1)$ and in particular $\mathbf{E}[|\tilde{f}_i - f_i|] = \Omega(1/B)$, which is the desired for constant k. □

6.4.4. Learned Count-Sketch for Zipfians

We now proceed to analyze the learned Count-Sketch algorithm. First, we estimate the expected error when using a single hash function and next we show that the expected error only increases when using more hash functions. Recall that we assume on the number of buckets B_h used to store the heavy hitters that $B_h = \Theta(B - B_h) = \Theta(B)$.

One hash function. By taking $B_1 = B_h = \Theta(B)$ and $B_2 = B - B_h = \Theta(B)$ in the theorem below the result on L-CS for $k = 1$ claimed in Table 6.4.1 follows immediately.

Theorem 6.4.15. *Let $h : [n] \setminus [B_1] \to [B_2]$ and $s : [n] \to \{-1, 1\}$ be truly random hash functions where $n, B_1, B_2 \in \mathbb{N}$ and[3] $n - B_1 \geq B_2 \geq B_1$. Define the random variable $\tilde{f}_i = \sum_{j=B_1+1}^{n} [h(j) = h(i)]s(j)f_j$ for $i \in [n] \setminus [B_1]$. Then*

$$\mathbf{E}[|\tilde{f}_i - s(i)f_i|] = \Theta\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right)$$

Proof: Let $N_1 = [B_1 + B_2] \setminus ([B_1] \cup \{i\})$ and $N_2 = [n] \setminus ([B_1 + B_2] \cup \{i\})$. Let $X_1 = \sum_{j \in N_1} [h(j) = h(i)]s(j)f_j$ and $X_2 = \sum_{j \in N_2} [h(j) = h(i)]s(j)f_j$. By the triangle inequality and linearity of expectation,

$$\mathbf{E}[|X_1|] = O\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right).$$

Moreover, it follows directly from Lemma 6.4.10 that $\mathbf{E}[|X_2|] = O\left(\frac{1}{B_2}\right)$. Thus

$$\mathbf{E}[|\tilde{f}_i - s(i)f_i|] \leq \mathbf{E}[|X_1|] + \mathbf{E}[|X_2|] = O\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right),$$

as desired.

[3]The first inequality is the standard assumption that we have at least as many items as buckets. The second inequality says that we use at least as many buckets for non-heavy items as for heavy items (which doesn't change the asymptotic space usage).

For the lower bound on $\mathbf{E}\left[\left|\tilde{f}_i - s(i)f_i\right|\right]$ we apply Lemma 6.4.11 to obtain that,

$$\mathbf{E}\left[\left|\tilde{f}_i - s(i)f_i\right|\right] \geq \frac{1}{2B_2}\left(1-\frac{1}{B_2}\right)^{|N_1|-1}\sum_{i\in N_1} f_i = \Omega\left(\frac{\log\frac{B_2+B_1}{B_1}}{B_2}\right). \qquad \square$$

Corollary 6.4.16. *Let $h : [n] \setminus [B_h] \to [B - B_h]$ and $s : [n] \to \{-1, 1\}$ be truly random hash functions where $n, B_1, B_2 \in \mathbb{N}$ and $B_h = \Theta(B) \leq B/2$. Define the random variable $\tilde{f}_i = \sum_{j=B_h+1}^{n}[h(j) = h(i)]s(j)f_j$ for $i \in [n] \setminus [B_h]$. Then*

$$\mathbf{E}[|\tilde{f}_i - s(i)f_i|] = \Theta\left(\frac{1}{B}\right)$$

More hash functions. We now show that, like for Count-Min, using more hash functions does not decrease the expected error. We first state the Littlewood-Offord lemma as strengthened by Erdős.

Theorem 6.4.17 (Littlewood-Offord [126], Erdős [69]). *Let $a_1, \ldots, a_n \in \mathbb{R}$ with $|a_i| \geq 1$ for $i \in [n]$. Let further $\sigma_1, \ldots, \sigma_n \in \{-1, 1\}$ be random variables with $\Pr[\sigma_i = 1] = \Pr[\sigma_i = -1] = 1/2$ and define $S = \sum_{i=1}^{n}\sigma_i a_i$. For any $v \in \mathbb{R}$ it holds that*

$$\Pr[|S - v| \leq 1] \leq \binom{n}{\lfloor n/2 \rfloor}\cdot\frac{1}{2^n} = O\left(\frac{1}{\sqrt{n}}\right).$$

Setting $B_1 = B_h = \Theta(B)$ and $B_2 = B - B_h = \Theta(B)$ in the theorem below gives the final bound from Table 6.4.1 on L-CS with $k \geq 3$.

Theorem 6.4.18. *Let $n \geq B_1 + B_2 \geq 2B_1$, $k \geq 3$ odd, and $h_1, \ldots, h_k : [n] \setminus [B_1] \to [B_2/k]$ and $s_1, \ldots, s_k : [n] \setminus [B_1] \to [B_2/k]$ be independent and truly random. Define the random variable $\tilde{f}_i = \mathsf{median}_{\ell\in[k]}\left(\sum_{j\in[n]\setminus[B_1]}[h_\ell(j) = h_\ell(i)]s_\ell(j)f_j\right)$ for $i \in [n] \setminus [B_1]$. Then*

$$\mathbf{E}[|\tilde{f}_i - s(i)f_i|] = \Omega\left(\frac{1}{B_2}\right).$$

Proof: Like in the proof of the lower bound of Theorem 6.4.13 it suffices to show that for each i the probability that the sum $S_\ell := \sum_{j\in[n]\setminus([B_1]\cup\{i\})}[h_\ell(j) = h_\ell(i)]s_\ell(j)f_j$ lies in the interval $I = [-1/(2B_2), 1/(2B_2)]$ is $O(1/\sqrt{k})$. Then at least half the $(S_\ell)_{\ell\in[k]}$ are at least

$1/(2B_2)$ with probability $\Omega(1)$ by an application of Stirling's formula, and it follows that $\mathbf{E}[|\tilde{f}_i - s(i)f_i|] = \Omega(1/B_2)$.

Let $\ell \in [k]$ be fixed, $N_1 = [2B_2] \setminus ([B_2] \cup \{i\})$, and $N_2 = [n] \setminus (N_1 \cup \{i\})$, and write

$$S_\ell = \sum_{j \in N_1} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j + \sum_{j \in N_2} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j := X_1 + X_2.$$

Now condition on the value of $X_2 = x_2$ of X_2. Letting $J = \{j \in N_1 : h_\ell(j) = h_\ell(i)\}$ it follows by Theorem 6.4.17 that

$$\Pr[S_\ell \in I \mid X_2 = x_2] = O\left(\sum_{J' \subseteq N_1} \frac{\Pr[J = J']}{\sqrt{|J'| + 1}}\right) = O\left(\Pr[|J| < k/2] + 1/\sqrt{k}\right).$$

An application of Chebyshev's inequality gives that $\Pr[|J| < k/2] = O(1/k)$, so $\Pr[S_\ell \in I] = O(1/\sqrt{k})$. Since this bound holds for any possible value of x_2 we may remove the conditioning and the desired result follows. □

Remark 6.4.19. The bound above is probably only tight for $B_1 = \Theta(B_2)$. Indeed, we know that it cannot be tight for all $B_1 \leq B_2$ since when B_1 becomes very small, the bound from the standard Count-Sketch with $k \geq 3$ takes over—and this is certainly worse than the bound in the theorem. It is an interesting open problem (that requires a better anti-concentration inequality than the Littlewood-Offord lemma) to settle the correct bound when $B_1 \ll B_2$.

6.5. Experiments

Baselines. We compare our learning-based algorithms with their non-learning counterparts. Specifically, we augment Count-Min with a learned oracle as in Algorithm 26, and call the learning-augmented algorithm "Learned Count-Min". We then compare Learned Count-Min with traditional Count-Min. We also compare it with "Learned Count-Min with Ideal Oracle" where the neural-network oracle is replaced with an ideal oracle that knows the identities of the heavy hitters in the test data, and "Count-Min with Lookup Table" where the heavy hitter oracle is replaced with a lookup table that memorizes heavy hitters in the training set. The comparison with the latter baseline allows us to show the ability of Learned Count-Min to generalize and detect heavy items unseen in the training set. We

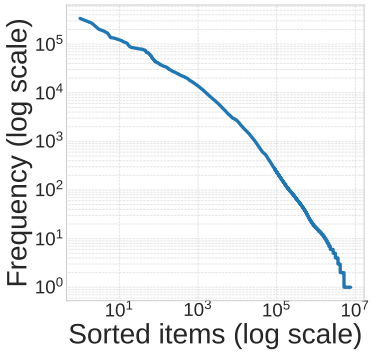


Figure 6.5.1: Frequency of Internet flows

repeat the evaluation where we replace Count-Min (CM) with Count-Sketch (CS) and the corresponding variants.

Training a Heavy Hitter Oracle. We construct the heavy hitter oracle by training a neural network to predict the heaviness of an item. Note that the prediction of the network is not the final estimation. It is used in Algorithm 26 to decide whether to assign an item to a unique bucket. We train the network to predict the item counts (or the log of the counts) and minimize the squared loss of the prediction. Empirically, we found that when the counts of heavy items are few orders of magnitude larger than the average counts (as is the case for the Internet traffic data set), predicting the log of the counts leads to more stable training and better results. Once the model is trained, we can select the optimal cutoff threshold using validation data, and use the model as the oracle described in Algorithm 26.

6.5.1. Internet Traffic Estimation

For our first experiment, the goal is to estimate the number of packets for each network flow. A flow is a sequence of packets between two machines on the Internet. It is identified by the IP addresses of its source and destination and the application ports. Estimating the size of each flow i—i.e., the number of its packets f_i—is a basic task in network management [161].

Dataset. The traffic data is collected at a backbone link of a Tier1 ISP between Chicago and Seattle in 2016 [39]. Each recording session is around one hour. Within each minute, there are around 30 million packets and 1 million unique flows. For a recording session, we use the first 7 minutes for training, the following minute for validation, and estimate the packet counts in subsequent minutes. The distribution of packet counts over Internet flows

is heavy tailed, as shown in Figure 6.5.1.

Model. The patterns of the Internet traffic are very dynamic, i.e., the flows with heavy traffic change frequently from one minute to the next. However, we hypothesize that the space of IP addresses should be smooth in terms of traffic load. For example, data centers at large companies and university campuses with many students tend to generate heavy traffic. Thus, though the individual flows from these sites change frequently, we could still discover regions of IP addresses with heavy traffic through a learning approach.

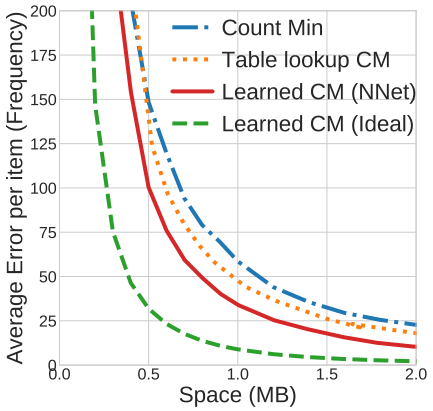
We trained a neural network to predict the log of the packet counts for each flow. The model takes as input the IP addresses and ports in each packet. We use two RNNs to encode the source and destination IP addresses separately. The RNN takes one bit of the IP address at each step, starting from the most significant bit. We use the final states of the RNN as the feature vector for an IP address. The reason to use RNN is that the patterns in the bits are hierarchical, i.e., the more significant bits govern larger regions in the IP space. Additionally, we use two-layer fully-connected networks to encode the source and destination ports. We then concatenate the encoded IP vectors, encoded port vectors, and the protocol type as the final features to predict the packet counts.[4] The inference time takes 2.8 microseconds per item on a single GPU without any optimizations.[5]

Results. We plot the results of two representative test minutes (the 20th and 50th) in Figure 6.5.2. All plots in the figure refer to the estimation error (6.2.2) as a function of the used space. The space includes space for storing the buckets and the model. Since we use the same model for all test minutes, the model space is amortized over the 50-minute testing period.

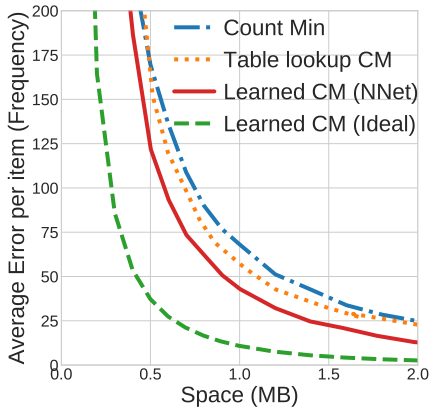
Figure 6.5.2 reveals multiple findings. First, the figure shows that our learning-based algorithms exhibit a better performance than their non-learning counterparts. Specifically, Learned Count-Min, compared to Count-Min, reduces the the error by 32% with space of

[4]We use RNNs with 64 hidden units. The two-layer fully-connected networks for the ports have 16 and 8 hidden units. The final layer before the prediction has 32 hidden units.

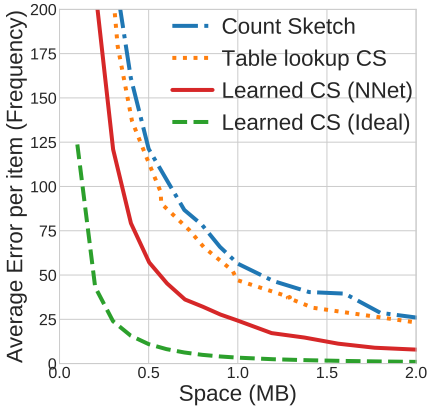
[5]Note that new specialized hardware such as Google TPU, hardware accelerators and network compression [93, 163, 47, 94, 95] can drastically improve the inference time. Further, Nvidia has predicted that GPU will get 1000x faster by 2025. Because of these trends, the overhead of neural network inference is expected to be less significant in the future [120].



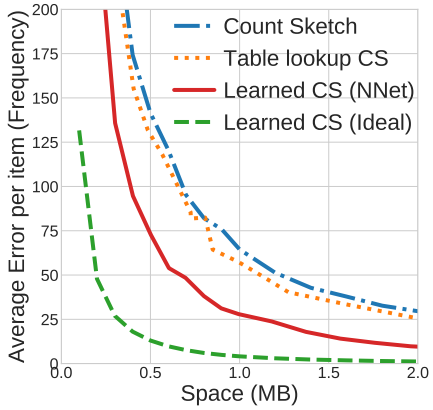
(a) Learned Count-Min - 20th test minute



(b) Learned Count-Min - 50th test minute



(c) Learned Count-Sketch - 20th test minute



(d) Learned Count-Sketch - 50th test minute

Figure 6.5.2: Comparison of our algorithms with Count-Min and Count-Sketch on Internet traffic data.

0.5 MB and 42% with space of 1.0 MB (Figure 6.5.2a). Learned Count-Sketch, compared to Count-Sketch, reduces the error by 52% at 0.5 MB and 57% at 1.0 MB (Figure 6.5.2c). In our experiments, each regular bucket takes 4 bytes. For the learned versions, we account for the extra space needed for the unique buckets to store the item IDs and the counts. One unique bucket takes 8 bytes, twice the space of a normal bucket.[6]

Second, the figure also shows that our neural-network oracle performs better than memorizing the heavy hitters in a lookup table. This is likely due to the dynamic nature of Internet traffic –i.e., the heavy flows in the training set are significantly different from those in the test data. Hence, memorization does not work well. On the other hand, our model is able to extract structures in the input that generalize to unseen test data.

[6]By using hashing with open addressing, it suffices to store IDs hashed into $\log B_r + t$ bits (instead of whole IDs) to ensure there is no collision with probability $1 - 2^{-t}$. $\log B_r + t$ is comparable to the number of bits per counter, so the space for a unique bucket is twice the space of a normal bucket.

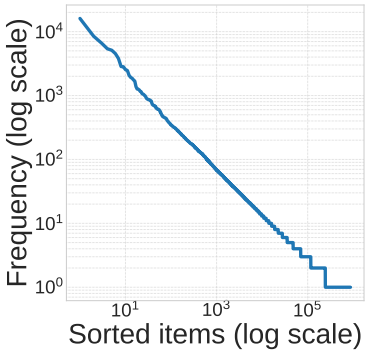


Figure 6.5.3: Frequency of search queries

Third, the figure shows that our model's performance stays roughly the same from the 20th to the 50th minute (Figure 6.5.2b and Figure 6.5.2d), showing that it learns properties of the heavy items that generalize over time.

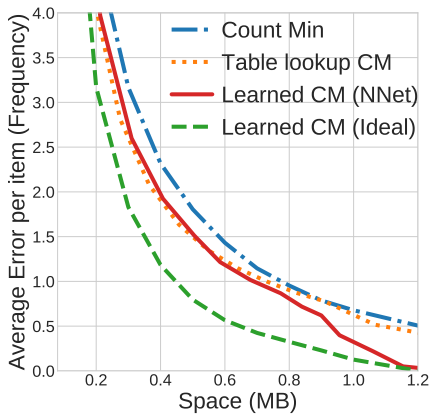
Lastly, although we achieve significant improvement over Count-Min and Count-Sketch, our scheme can potentially achieve even better results with an ideal oracle, as shown by the dashed green line in Figure 6.5.2. This indicates potential gains from further optimizing the neural network model.

6.5.2. Search Query Estimation

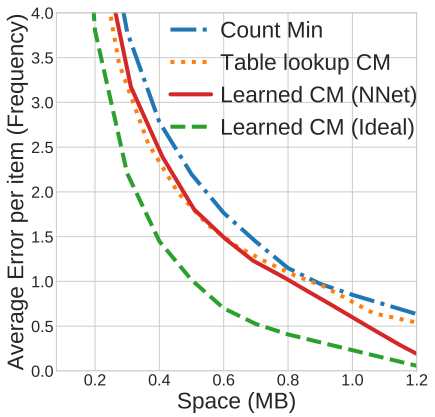
For our second experiment, the goal is to estimate the number of times a search query appears.

Dataset. We use the AOL query log dataset, which consists of 21 million search queries collected from 650 thousand users over 90 days. The users are anonymized in the dataset. There are 3.8 million unique queries. Each query is a search phrase with multiple words (e.g., "periodic table element poster"). We use the first 5 days for training, the following day for validation, and estimate the number of times different search queries appear in subsequent days. The distribution of search query frequency follows the Zipfian law, as shown in Figure 6.5.3.

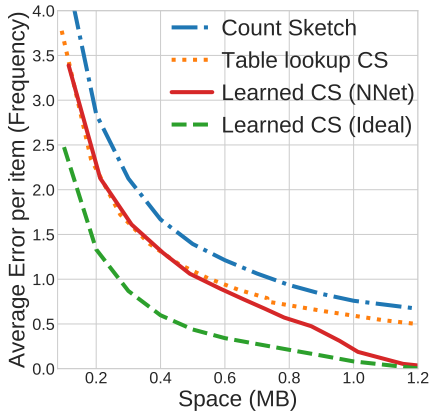
Model. Unlike traffic data, popular search queries tend to appear more consistently across multiple days. For example, "google" is the most popular search phrase in most of the days in the dataset. Simply storing the most popular words can easily construct a reasonable heavy hitter predictor. However, beyond remembering the popular words, other factors also



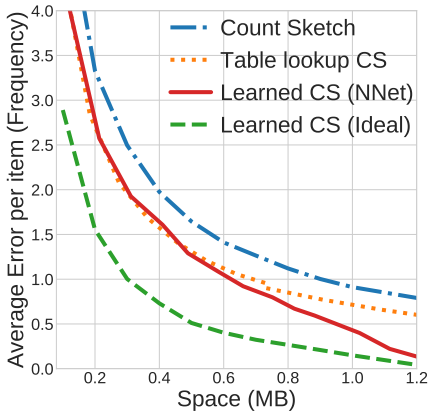
(a) Learned Count-Min 50th test day



(b) Learned Count-Min 80th test day



(c) Learned Count-Sketch 50th test day



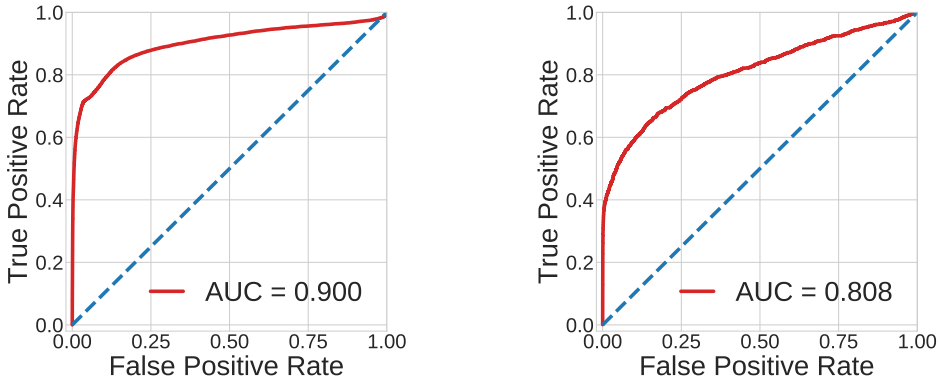
(d) Learned Count-Sketch 80th test day

Figure 6.5.4: Comparison of our algorithms with Count-Min and Count-Sketch on search query data.

contribute to the popularity of a search phrase that we can learn. For example, popular search phrases appearing in slightly different forms may be related to similar topics. Though not included in the AOL dataset, in general, metadata of a search query (e.g., the location of the search) can provide useful context of its popularity.

To construct the heavy hitter oracle, we trained a neural network to predict the number of times a search phrase appears. To process the search phrase, we train an RNN with LSTM cells that takes characters of a search phrase as input. The final states encoded by the RNN are fed to a fully-connected layer to predict the query frequency. Our character vocabulary includes lower-case English alphabets, numbers, punctuation marks, and a token for unknown characters. We map the character IDs to embedding vectors before feeding them to the RNN.[7] We choose RNN due to its effectiveness in processing sequence data [162, 83, 120].

[7] We use an embedding size of 64 dimensions, an RNN with 256 hidden units, and a fully-connected layer with 32 hidden units.



(a) Internet traffic 50th test minute **(b)** Search query 50th test day

Figure 6.5.5: ROC curves of the learned models.

Results. We plot the estimation error vs. space for two representative test days (the 50th and 80th day) in Figure 6.5.4. As before, the space includes both the bucket space and the space used by the model. The model space is amortized over the test days since the same model is used for all days.

Similarly, our learned sketches outperforms their conventional counterparts. For Learned Count-Min, compared to Count-Min, it reduces the loss by 18% at 0.5 MB and 52% at 1.0 MB (Figure 6.5.4a). For Learned Count-Sketch, compared to Count-Sketch, it reduces the loss by 24% at 0.5 MB and 71% at 1.0 MB (Figure 6.5.4c). Further, our algorithm performs similarly for the 50th and the 80th day (Figure 6.5.4b and Figure 6.5.4d), showing that the properties it learns generalize over a long period.

The figures also show an interesting difference from the Internet traffic data: memorizing the heavy hitters in a lookup table is quite effective in the low space region. This is likely because the search queries are less dynamic compared to Internet traffic (i.e., top queries in the training set are also popular on later days). However, as the algorithm is allowed more space, memorization becomes ineffective.

6.5.3. Analyzing Heavy Hitter Models

We analyze the accuracy of the neural network heavy hitter models to better understand the results on the two datasets. Specifically, we use the models to predict whether an item is a heavy hitter (top 1% in counts) or not, and plot the ROC curves in Figure 6.5.5. The figures show that the model for the Internet traffic data has learned to predict heavy items more effectively, with an AUC score of 0.9. As for the model for search query data, the AUC

score is 0.8. This also explains why we see larger improvements over non-learning algorithms in Figure 6.5.2.

Chapter 7

The Low-Rank Approximation Problem

7.1. Introduction

In the *low-rank decomposition* problem, given an $n \times d$ matrix A and a parameter k, the goal is to compute a rank-k matrix $[A]_k$ defined as below

$$[A]_k = \text{argmin}_{A' \text{: rank}(A') \leq k} \|A - A'\|_F.$$

Low-rank approximation is one of the most widely used tools in massive data analysis, machine learning and statistics, and has been a subject of many algorithmic studies. In particular, multiple algorithms developed over the last decade use the "sketching" approach, see e.g., [157, 171, 92, 52, 53, 144, 132, 34, 54]. Its idea is to use efficiently computable random projections (a.k.a., "sketches") to reduce the problem size before performing low-rank decomposition, which makes the computation more space and time efficient. For example, [157, 52] show that if S is a random matrix of size $m \times n$ chosen from an appropriate distribution[1], for m depending on ε, then one can recover a rank-k matrix A' such that

$$\|A - A'\|_F \leq (1 + \epsilon)\|A - [A]_k\|_F$$

[1] Initial algorithms used matrices with independent sub-gaussian entries or randomized Fourier/Hadamard matrices [157, 171, 92]. Starting from the seminal work of [53], researchers began to explore sparse binary matrices, see e.g., [144, 132]. In this chapter we mostly focus on the latter distribution.

by performing an SVD on $SA \in \mathbb{R}^{m \times d}$ followed by some post-processing. Typically the sketch length m is small, so the matrix SA can be stored using little space (in the context of streaming algorithms) or efficiently communicated (in the context of distributed algorithms). Furthermore, the SVD of SA can be computed efficiently, especially after another round of sketching, reducing the overall computation time. See the survey [170] for an overview of these developments.

In light of recent developments on learning-based algorithms [168, 59, 120, 24, 128, 151, 137, 141, 25, 33, 134, 96, 103, 1, 118], it is natural to ask whether similar improvements in performance could be obtained for other sketch-based algorithms, notably for low-rank decompositions. In particular, reducing the sketch length m while preserving its accuracy would make sketch-based algorithms more efficient. Alternatively, one could make sketches more accurate for the same values of m. This is the problem we address in this chapter.

7.1.1. Our Results

Our main finding is that learned sketch matrices can indeed yield (much) more accurate low-rank decompositions than purely random matrices. We focus our study on a streaming algorithm for low-rank decomposition due to [157, 52], described in more detail in Section 7.2. Specifically, suppose we have a training set of matrices $\mathsf{Tr} = \{A_1, \ldots, A_N\}$ sampled from some distribution $\mathcal{D}$. Based on this training set, we compute a matrix S^* that (locally) minimizes the empirical loss

$$\sum_i \|A_i - \mathrm{SCW}(S^*, A_i)\|_F \tag{7.1.1}$$

where $\mathrm{SCW}(S^*, A_i)$ denotes the output of the aforementioned Sarlos-Clarkson-Woodruff streaming low-rank decomposition algorithm on matrix A_i using the sketch matrix S^*. Once the sketch matrix S^* is computed, it can be used instead of a random sketch matrix in all future executions of the SCW algorithm.

We demonstrate empirically that, for multiple types of data sets, an optimized sketch matrix S^* can substantially reduce the approximation loss compared to a random matrix S, sometimes by one order of magnitude (see Figure 7.5.1 or 7.5.2). Equivalently, the optimized sketch matrix can achieve the same approximation loss for lower values of m.

A possible disadvantage of learned sketch matrices is that an algorithm that uses them no

longer offers *worst-case* guarantees. As a result, if such an algorithm is applied to an input matrix that does not conform to the training distribution, the results might be worse than if random matrices were used. To alleviate this issue, we also study *mixed* sketch matrices, where (say) half of the rows are trained and the other half are random. We observe that if such matrices are used in conjunction with the SCW algorithm, its results are no worse than if only the random part of the matrix was used (Theorem 7.4.1 in Section 7.4)[2]. Thus, the resulting algorithm inherits the worst-case performance guarantees of the random part of the sketching matrix. At the same time, we show that mixed matrices still substantially reduce the approximation loss compared to random ones, in some cases nearly matching the performance of "pure" learned matrices with the same number of rows. Thus, mixed random matrices offer "the best of both worlds": improved performance for matrices from the training distribution, and worst-case guarantees otherwise.

Finally, in order to understand the theoretical aspects of our approach further, we study the special case of $m = 1$. This corresponds to the case where the sketch matrix S is just a single vector. Our results are two-fold:

- We give an approximation algorithm for minimizing the empirical loss as in Equation 7.1.1, with an approximation factor depending on the stable rank of matrices in the training set. See Appendix 7.7.
- Under certain assumptions about the robustness of the loss minimizer, we show generalization bounds for the solution computed over the training set. See Appendix 7.8.

7.1.2. Related work

As outlined in the introduction, over the last few years there has been multiple papers exploring the use of machine learning methods to improve the performance of "standard" algorithms. Among those, the closest to the topic of this chapter are the works on learning-based compressive sensing, such as [141, 25, 33, 134], and on learning-based streaming algorithms [103]. Since neither of these two lines of research addresses computing matrix spectra, the technical development therein was quite different from ours.

[2]We note that this property is non-trivial, in the sense that it does not automatically hold for *all* sketching algorithms. See Section 7.4 for further discussion.

Algorithm 27 Rank-k approximation of a matrix A using a sketch matrix S (refer to Section 4.1.1 of [52])

1: **Input:** $A \in \mathbb{R}^{n\times d}, S \in \mathbb{R}^{m\times n}$
2: $U, \Sigma, V^\top \leftarrow \textsc{CompactSVD}(SA)$ ▷ $r = \text{rank}(SA), U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{d\times r}$
3: **return** $[AV]_k V^\top$

In this chapter we focus on learning-based optimization of low-rank approximation algorithms that use *linear sketches*, i.e., map the input matrix A into SA and perform computation on the latter. There are other sketching algorithms for low-rank approximation that involve *non-linear* sketches [125, 78, 77]. The benefit of linear sketches is that they are easy to update under linear changes to the matrix A, and (in the context of our work) that they are easy to differentiate, making it possible to compute the gradient of the loss function as in Equation 7.1.1. We do not know whether it is possible to use our learning-based approach for non-linear sketches, but we believe this is an interesting direction for future research.

7.2. Preliminaries

Notation. Consider a distribution $\mathcal{D}$ on matrices $A \in \mathbb{R}^{n\times d}$. We define the training set as $\{A_1, \cdots, A_N\}$ sampled from $\mathcal{D}$. For matrix A, its *singular value decomposition (SVD)* can be written as $A = U\Sigma V^\top$ such that both $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{d\times n}$ have *orthonormal columns* and $\Sigma = \text{diag}\{\lambda_1, \cdots, \lambda_d\}$ is a diagonal matrix with nonnegative entries. Moreover, if $\text{rank}(A) = r$, then the first r columns of U are an orthonormal basis for the *column space* of A (we denote it as $\text{colsp}(A)$), the first r columns of V are an orthonormal basis for the *row space* of A (we denote it as $\text{rowsp}(A)$)[3] and $\lambda_i = 0$ for $i > r$. In many applications it is quicker and more economical to compute the *compact SVD* which only contains the rows and columns corresponding to the non-zero singular values of Σ: $A = U^c\Sigma^c(V^c)^\top$ where $U^c \in \mathbb{R}^{n\times r}, \Sigma^c \in \mathbb{R}^{r\times r}$ and $V^c \in \mathbb{R}^{d\times r}$.

How sketching works. We start by describing the SCW algorithm for low-rank matrix approximation, see Algorithm 27. The algorithm computes the singular value decomposition of $SA = U\Sigma V^\top$, and compute the best rank-k approximation of AV. Finally it outputs $[AV]_k V^\top$ as a rank-k approximation of A.

[3] The remaining columns of U and V respectively are orthonormal bases for the nullspace of A and $A^\top$.

Note that if m is much smaller than d and n, the space bound of this algorithm is significantly better than when computing a rank-k approximation for A in the naïve way. Thus, minimizing m automatically reduces the space usage of the algorithm.

Sketching matrix. We use matrix S that is sparse[4] Specifically, each column of S has exactly one non-zero entry, which is either $+1$ or -1. This means that the fraction of non-zero entries in S is $1/m$. Therefore, one can use a vector to represent S, which is very memory efficient. It is worth noting, however, after multiplying the sketching matrix S with other matrices, the resulting matrix (e.g., SA) is in general not sparse.

7.3. Training Algorithm

In this section, we describe our learning-based algorithm for computing a data dependent sketch S. The main idea is to use back-propagation algorithm to compute the stochastic gradient of S with respect to the rank-k approximation loss in Equation (7.1.1), where the initial value of S is the same random sparse matrix used in SCW. Once we have the stochastic gradient, we can run stochastic gradient descent (SGD) algorithm to optimize S, in order to improve the loss. Our algorithm maintains the sparse structure of S, and only optimizes the values of the n non-zero entries (initially $+1$ or -1).

However, the standard SVD implementation (step 2 in Algorithm 27) is not differentiable, which means we cannot get the gradient in the straightforward way. To make SVD implementation differentiable, we use the fact that the SVD procedure can be represented as m individual top singular value decompositions (see e.g. [10]), and that every top singular value decomposition can be computed using the power method. See Figure 7.3.1 and Algorithm 28. We store the results of the i-th iteration into the i-th entry of the list U, Σ, V, and finally concatenate all entries together to get the matrix (or matrix diagonal) format of U, Σ, V. This allows gradients to flow easily.

Due to the extremely long computational chain, it is infeasible to write down the explicit form of loss function or the gradients. However, just like how modern deep neural networks

[4]The original papers [157, 52] used dense matrices, but the work of [53] showed that sparse matrices work as well. We use sparse matrices since they are more efficient to train and to operate on.

Algorithm 28 Differentiable implementation of SVD

Input: $A_1 \in \mathbb{R}^{m \times d} (m < d)$
$U, \Sigma, V \leftarrow \{\}, \{\}, \{\}$
for $i \leftarrow 1 \dots m$ **do**
    $v_1 \leftarrow$ random initialization in $\mathbb{R}^d$
    for $t \leftarrow 1 \dots T$ **do**
        $v_{t+1} \leftarrow \frac{A_i^\top A_i v_t}{\|A_i^\top A_i v_t\|_2}$ ▷ power method
    $V[i] \leftarrow v_{T+1}$
    $\Sigma[i] \leftarrow \|A_i V[i]\|_2$
    $U[i] \leftarrow \frac{A_i V[i]}{\Sigma[i]}$
    $A_{i+1} \leftarrow A_i - \Sigma[i] U[i] V[i]^\top$
return U, Σ, V

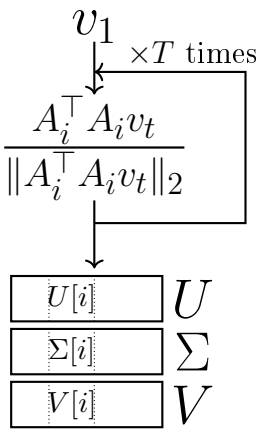


Figure 7.3.1: The i-th iteration of power method.

compute their gradients, we used the autograd feature in PyTorch to *numerically* compute the gradient with respect to the sketching matrix S.

We emphasize again that our method is only optimizing S for the training phase. After S is fully trained, we still call Algorithm 27 for low rank approximation, which has exactly the same running time as the SCW algorithm, but with better performance.

7.4. Worst Case Bound

In this section, we show that concatenating two sketching matrices S_1 and S_2 (of size respectively $m_1 \times n$ and $m_2 \times n$) into a single matrix S_* (of size $(m_1 + m_2) \times n$) will not increase the approximation loss of the final rank-k solution computed by Algorithm 27 compared to the case in which only one of S_1 or S_2 are used as the sketching matrix. In the rest of this section, the sketching matrix S_* denotes the concatenation of S_1 and S_2 as follows:

$$S_{*((m_1+m_2)\times n)} = \begin{bmatrix} S_{1(m_1 \times n)} \\ \hline S_{2(m_2 \times n)} \end{bmatrix}$$

Formally, we prove the following theorem in this section.

Theorem 7.4.1. *Let $U_* \Sigma_* V_*$ and $U_1 \Sigma_1 V_1$ respectively denote the SVD of $S_* A$ and $S_1 A$.*

Then,

$$||[AV_*]_k V_*^\top - A||_F \leq ||[AV_1]_k V_1^\top - A||_F.$$

In particular, the above theorem implies that the output of Algorithm 27 with the sketching matrix S_* is a better rank-k approximation to A compared to the output of the algorithm with S_1. In the rest of this section we prove Theorem 7.4.1.

Fact 7.4.2 (Pythagorean Theorem). *If A and B are matrices with the same number of rows and columns, then $AB^\top = 0$ implies $||A+B||_F^2 = ||A||_F^2 + ||B||_F^2$.*

Claim 7.4.3. *Let V_1 and V_2 be two matrices such that* $\text{colsp}(V_1) \subseteq \text{colsp}(V_2)$. *Then,*

$$\min_{\substack{\text{rowsp}(X) \subseteq \text{colsp}(V_1); \\ rank(X) \leq k}} ||X - A||_F^2 \geq \min_{\substack{\text{rowsp}(X) \subseteq \text{colsp}(V_2); \\ rank(X) \leq k}} ||X - A||_F^2$$

Before proving the main theorem, we state the following helpful lemma from [52].

Lemma 7.4.4 (Lemma 4.3 in [52]). *Suppose that V is a matrix with orthonormal columns. Then, a best rank-k approximation to A in the* $\text{colsp}(V)$ *is given by $[AV]_k V^\top$.*

Proof: Note that $AVV^\top$ is a row projection of A on the $\text{colsp}(V)$. Then, for any conforming Y,

$$\begin{aligned}(A - AVV^\top)(AVV^\top - YV^\top)^\top &= A(I - VV^\top)V(AV - Y)^\top \\ &= A(V - VV^\top V)(AV - Y)^\top = 0.\end{aligned}$$

where the last equality follows from the fact if V has orthonormal columns then $VV^\top V = V$ (e.g., see Lemma 3.5 in [52]). Then, by the Pythagorean Theorem (Fact 7.4.2), we have

$$||A - YV^\top||_F^2 = ||A - AVV^\top||_F^2 + ||AVV^\top - YV^\top||_F^2 \tag{7.4.1}$$

Since V has orthonormal columns, for any conforming x, $||x^\top V^\top|| = ||x||$. Thus, for any Z

of rank at most k,

$$||AVV^\top - [AV]_k V^\top||_F = ||(AV - [AV]_k)V^\top||_F = ||AV - [AV]_k||_F$$
$$\leq ||AV - Z||_F = ||AVV^\top - ZV^\top||_F \tag{7.4.2}$$

Hence,

$$||A - [AV]_k V^\top||_F^2 = ||A - AVV^\top||_F^2 + ||AVV^\top - [AV]_k V^\top||_F^2 \quad \triangleright \text{By (7.4.1)}$$
$$\leq ||A - AVV^\top||_F^2 + ||AVV^\top - ZV^\top||_F^2 \quad \triangleright \text{By (7.4.2)}$$

This implies that $[AV]_k V^\top$ is a best rank-k approximation of A in the colsp(V). □

Since the above statement is a transposed version of the lemma from [52], we include the proof in the appendix for completeness.

Proof: (Proof of Theorem 7.4.1) First, we show that colsp$(V_1) \subseteq$ colsp(V_*). By the properties of the (compact) SVD, colsp$(V_1) =$ rowsp$(S_1 A)$ and colsp$(V_*) =$ rowsp$(S_* A)$. Since, S_* has all rows of S_1, then

$$\text{colsp}(V_1) \subseteq \text{colsp}(V_*). \tag{7.4.3}$$

By Lemma 7.4.4,

$$||A - [AV_*]_k V_*^\top||_F = \min_{\substack{\text{rowsp}(X) \subseteq \text{colsp}(V_*); \\ \text{rank}(X) \leq k}} ||X - A||_F$$
$$||A - [AV_1]_k V_1^\top||_F = \min_{\substack{\text{colsp}(X) \subseteq \text{colsp}(V_1); \\ \text{rank}(X) \leq k}} ||X - A||_F$$

Finally, together with (7.4.3) and Claim 7.4.3

$$||A - [AV_*]_k V_*^\top||_F = \min_{\substack{\text{rowsp}(X)\subseteq\text{colsp}(V_*);\\ \text{rank}(X)\leq k}} ||X - A||_F$$
$$\leq \min_{\substack{\text{rowsp}(X)\subseteq\text{colsp}(V_1);\\ \text{rank}(X)\leq k}} ||X - A||_F = ||A - [AV_1]_k V_1^\top||_F.$$

which completes the proof. □

Finally, we note that the property of Theorem 7.4.1 is not universal, i.e., it does not hold for all sketching algorithms for low-rank decomposition. For example, an alternative algorithm proposed in [54] proceeds by letting Z to be the top k singular vectors of SA and then reports $AZZ^\top$. It is not difficult to see that, by adding extra rows to the sketching matrix S, one can skew the output of the algorithm so that it is far from the optimal.

7.5. Experimental Results

The main question considered here is whether, for natural matrix datasets, optimizing the sketch matrix S can improve the performance of the sketching algorithm for the low-rank decomposition problem. To answer this question, we implemented and compared the following methods for computing $S \in \mathbb{R}^{m\times n}$.

- **Sparse Random**. Sketching matrices are generated at random as in [53]. Specifically, we select a random hash function $h : [n] \to [m]$, and for all $i = 1 \ldots n$, $S_{h[i],i}$ is selected to be either $+1$ or -1 with equal probability. All other entries in S are set to 0. Therefore, S has exactly n non-zero entries.
- **Dense Random**. All the nm entries in the sketching matrices are sampled from Gaussian distribution (we include this method for comparison).
- **Learned**. Using the sparse random matrix as the initialization, we run Algorithm 28 to optimize the sketching matrix using the training set, and return the optimized matrix.
- **Mixed (J)**. We first generate two sparse random matrices $S_1, S_2 \in \mathbb{R}^{\frac{m}{2}\times n}$ (assuming m is even), and define S to be their combination. We then run Algorithm 28 to optimize S using the training set, but only S_1 will be updated, while S_2 is fixed. Therefore, S is

a mixture of learned matrix and random matrix, and the first matrix is trained *jointly* with the second one.

- **Mixed (S)**. We first compute a learned matrix $S_1 \in \mathbb{R}^{\frac{m}{2} \times n}$ using the training set, and then append another sparse random matrix S_2 to get $S \in \mathbb{R}^{m \times n}$. Therefore, S is a mixture of learned matrix and random matrix, but the learned matrix is trained *separately*.

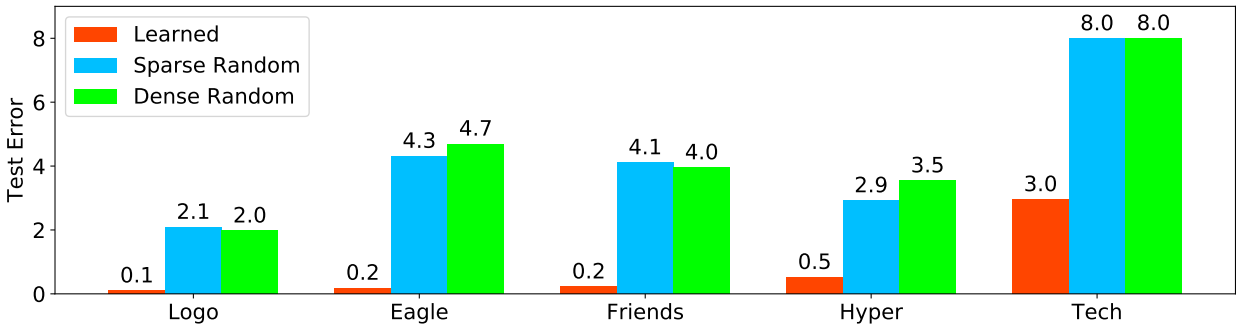


Figure 7.5.1: Test error by datasets and sketching matrices For $k = 10, m = 20$

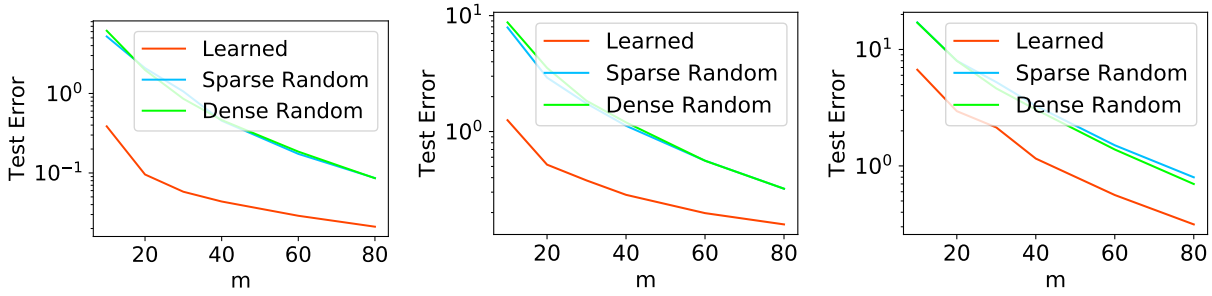


Figure 7.5.2: Test error for Logo (left), Hyper (middle) and Tech (right) when $k = 10$.

Datasets. We used a variety of datasets to test the performance of our methods:

- **Videos[5]: Logo, Friends, Eagle.** We downloaded three high resolution videos from Youtube, including logo video, Friends TV show, and eagle nest cam. From each video, we collect 500 frames of size $1920 \times 1080 \times 3$ pixels, and use 400 (100) matrices as the training (test) set. For each frame, we resize it as a 5760×1080 matrix.

- **Hyper.** We use matrices from HS-SOD, a dataset for hyperspectral images from natural scenes [104]. Each matrix has 1024×768 pixels, and we use 400 (100) matrices

[5]They can be downloaded from http://youtu.be/L5HQoFIaT4I, http://youtu.be/xmLZsEfXEgE and http://youtu.be/ufnf_q_3Ofg

as the training (test) set.

- **Tech.** We use matrices from TechTC-300, a dataset for text categorization [60]. Each matrix has $835,422$ rows, but on average only $25,389$ of the rows contain non-zero entries. On average each matrix has 195 columns. We use 200 (95) matrices as the training (test) set.

Evaluation metric. To evaluate the quality of a sketching matrix S, it suffices to evaluate the output of Algorithm 27 using the sketching matrix S on different input matrices A. We first define the optimal approximation loss for test set Te as follows: $\text{App}^*_{\mathsf{Te}} \triangleq \mathbf{E}_{A\sim\mathsf{Te}}\|A - [A]_k\|_F$.

Note that $\text{App}^*_{\mathsf{Te}}$ does not depend on S, and in general it is not achievable by any sketch S with $m < d$, because of information loss. Based on the definition of the optimal approximation loss, we define the error of the sketch S for Te as $\text{Err}(\mathsf{Te}, S) \triangleq \mathbf{E}_{A\sim\mathsf{Te}}\|A - \text{SCW}(S, A)\|_F - \text{App}^*_{\mathsf{Te}}$.

In our datasets, some of the matrices have much larger singular values than the others. To avoid imbalance in the dataset, we normalize the matrices so that their top singular values are all equal.

Figure 7.5.3: Low rank approximation results for Logo video frame: the best rank-10 approximation (left), and rank-10 approximations reported by Algorithm 27 using a sparse learned sketching matrix (middle) and a sparse random sketching matrix (right).

7.5.1. Average test error

We first test all methods on different datasets, with various combination of k, m. See Figure 7.5.1 for the results when $k = 10, m = 20$. As we can see, for video datasets, learned sketching matrices can get $20\times$ better test error than the sparse random or dense random sketching matrices. For other datasets, learned sketching matrices are still more than $2\times$ better. Similar improvement of the learned sketching matrices over the random sketching

Table 7.5.1: Test error in various settings.

k, m, Sketch	Logo	Eagle	Friends	Hyper	Tech
10, 10, Learned	0.39	0.31	1.03	1.25	6.70
10, 10, Random	5.22	6.33	11.56	7.90	17.08
10, 20, Learned	0.10	0.18	0.22	0.52	2.95
10, 20, Random	2.09	4.31	4.11	2.92	7.99
20, 20, Learned	0.61	0.66	1.41	1.68	7.79
20, 20, Random	4.18	5.79	9.10	5.71	14.55
20, 40, Learned	0.18	0.41	0.42	0.72	3.09
20, 40, Random	1.19	3.50	2.44	2.23	6.20
30, 30, Learned	0.72	1.06	1.78	1.90	7.14
30, 30, Random	3.11	6.03	6.27	5.23	12.82
30, 60, Learned	0.21	0.61	0.42	0.84	2.78
30, 60, Random	0.82	3.28	1.79	1.88	4.84

Table 7.5.2: Performance of mixed sketches

k, m, Sketch	Logo	Hyper	Tech
10, 10, Learned	0.39	1.25	6.70
10, 10, Random	5.22	7.90	17.08
10, 20, Learned	0.10	0.52	2.95
10, 20, Mixed (J)	0.20	0.78	3.73
10, 20, Mixed (S)	0.24	0.87	3.69
10, 20, Random	2.09	2.92	7.99
10, 40, Learned	0.04	0.28	1.16
10, 40, Mixed (J)	0.05	0.34	1.31
10, 40, Mixed (S)	0.05	0.34	1.20
10, 40, Random	0.45	1.12	3.28
10, 80, Learned	0.02	0.16	0.31
10, 80, Random	0.09	0.32	0.80

matrices can be observed when $k = 10, m = 10, 20, 30, 40, \cdots, 80$, see Figure 7.5.2. We also include the test error results in Table 7.5.1 for the case when $k = 20, 30$. Finally, in Figure 7.5.3, we visualize an example output of the algorithm for the case $k = 10, m = 20$ for the Logo dataset.

7.5.2. Comparing Random, Learned and Mixed

In Table 7.5.2, we investigate the performance of the mixed sketching matrices by comparing them with random and learned sketching matrices. In all scenarios, mixed sketching matrices yield much better results than random sketching matrices, and sometimes the results are comparable to those of learned sketching matrices. This means, in most cases it suffices to train one half of the sketching matrix to obtain good empirical results, and at the same time, by our Theorem 7.4.1, we can use the remaining random half of the sketch matrix to obtain worst-case guarantee.

7.6. The case of $m = 1$

In this section, we denote the SVD of A as $U^A \Sigma^A (V^A)^\top$ such that both U^A and V^A have *orthonormal columns* and $\Sigma^A = \mathrm{diag}\{\lambda_1^A, \cdots, \lambda_d^A\}$ is a diagonal matrix with nonnegative entries. For simplicity, we assume that for all $A \sim \mathcal{D}$, $1 = \lambda_1 \geq \cdots \geq \lambda_d$. We use U_i^A to denote the i-th column of U^A, and similarly for V_i^A. Denote $\Sigma^A = \mathrm{diag}\{\lambda_1^A, \cdots, \lambda_d^A\}$.

We want to find $[A]_k$, the rank-k approximation of A. In general, it is hard to obtain a closed form expression of the output of Algorithm 27. However, for $m = 1$, such expressions can be calculated. Indeed, if $m = 1$, the sketching matrix becomes a vector $s \in \mathbb{R}^{1 \times n}$.

Therefore A_V in Algorithm 27 has rank at most one, so it suffices to set $k = 1$. Consider a matrix $A \sim \mathcal{D}$ as the input to Algorithm 27. By calculation, $A_S = \sum_i \lambda_i^A \langle s, U_i^A \rangle (V_i^A)^\top$, which is a vector. For example, if $S = U_1^A$, we obtain $\lambda_1^A (V_1^A)^\top$.

Since SA is a vector, applying SVD on it is equivalent to performing normalization. Therefore,

$$V = \frac{\sum_{i=1}^d \lambda_i^A \langle s, U_i^A \rangle (V_i^A)^\top}{\sqrt{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle^2}}$$

Ideally, we hope that V is as close to V_1^A as possible, because that means $[AV]_1 V^\top$ is close to $\lambda_1^A U_1^A (V_1^A)^\top$, which captures the top singular component of A, i.e., the optimal solution. More formally,

$$AV = \frac{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle U_i^A}{\sqrt{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle^2}}$$

We want to maximize its norm, which is:

$$\frac{\sum_{i=1}^d (\lambda_i^A)^4 \langle s, U_i^A \rangle^2}{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle^2} \tag{7.6.1}$$

We note that one can simplify (7.6.1) by considering only the contribution from the top left singular vector U_1^A, which corresponds to the maximization of the following expression:

$$\frac{\langle s, U_1^A \rangle^2}{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle^2} \tag{7.6.2}$$

7.7. Optimization Bounds

Motivated by the empirical success of sketch optimization, we investigate the complexity of optimizing the loss function. We focus on the simple case where $m = 1$ and therefore S is just a (dense) vector. Our main observation is that a vector s picked *uniformly at random* from the d-dimensional unit sphere achieves an approximately optimal solution, with the approximation factor depending on the maximum *stable rank* of matrices $A_1, \cdots, A_N$. This

algorithm is not particularly useful for our purpose, as our goal is to *improve* over the random choice of the sketching matrix S. Nevertheless, it demonstrates that an algorithm with a non-trivial approximation factor exists.

Definition 7.7.1 (stable rank (r')). For a matrix A, the stable rank of A is defined as the squared ratio of Frobenius and operator norm of A. I.e.,

$$r'(A) = \frac{||A||_F^2}{||A||_2} = \frac{\sum_i (\lambda_i^A)^2}{\max_i (\lambda_i^A)^2}.$$

Note that since we assume for all matrices $A \sim \mathcal{D}$, $1 = \lambda_1^A \geq \cdots \geq \lambda_d^A > 0$, for all these matrices $r'(A) = \sum_i (\lambda_i^A)^2$.

First, we consider the simplified objective function as in (7.6.2).

Lemma 7.7.2. *A random vector s which is picked uniformly at random from the d-dimensional unit sphere, is an $O(r')$-approximation to the optimum value of the simplified objective function in Equation (7.6.2), where r' is the maximum stable rank of matrices $A_1, \cdots, A_N$.*

Proof: We will show that

$$E\left[\frac{\langle s, U_1^A\rangle^2}{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A\rangle^2}\right] = \Omega(1/r'(A))$$

for all $A \sim \mathcal{D}$ where s is a vector picked uniformly at random from $\mathbf{S}^{d-1}$. Since for all $A \sim \mathcal{D}$ we have $\frac{\langle s, U_1^A\rangle^2}{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A\rangle^2} \leq 1$, by the linearity of expectation we have that the vector s achieves an $O(r')$-approximation to the maximum value of the objective function

$$\sum_{j=1}^N \frac{\langle s, U_1^{A_j}\rangle^2}{\sum_{i=1}^d (\lambda_i^{A_j})^2 \langle s, U_i^{A_j}\rangle^2}$$

.

First, recall that to sample s uniformly at random from $\mathbf{S}^{d-1}$ we can generate s as $\sum_{i=1}^d \alpha_i U_i^A / \sqrt{\sum_{i=1}^d \alpha_i^2}$ where for all $i \leq d$, $\alpha_i \sim \mathcal{N}(0,1)$. This helps us evaluate $\mathbf{E}\left[\frac{\langle s, U_1^A\rangle^2}{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A\rangle^2}\right]$ for an arbitrary matrix $A \sim \mathcal{D}$:

$$E = \mathbf{E}\left[\frac{\langle s, U_1^A\rangle^2}{\sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A\rangle^2}\right] = \mathbf{E}\left[\frac{(\alpha_1)^2}{\sum_{i=1}^d (\lambda_i^A)^2 \cdot (\alpha_i)^2}\right]$$
$$\geq \mathbf{E}\left[\frac{(\alpha_1)^2}{\sum_{i=1}^d (\lambda_i^A)^2 \cdot (\alpha_i)^2} | \Psi_1 \cap \Psi_2\right] \cdot \Pr(\Psi_1 \cap \Psi_2)$$

where the events Ψ_1, Ψ_2 are defined as:

$$\Psi_1 \triangleq \mathbf{1}\left[|\alpha_1| \geq \frac{1}{2}\right] \text{ and } \Psi_2 \triangleq \mathbf{1}\left[\sum_{i=2}^d (\lambda_i^A)^2 (\alpha_i)^2 \leq 2 \cdot r'(A)\right]$$

Since α_is are independent, we have

$$E \geq \mathbf{E}\left[\frac{(\alpha_1)^2}{(\alpha_1)^2 + 2 \cdot r'(A)} | \Psi_1 \cap \Psi_2\right] \cdot \Pr(\Psi_1) \cdot \Pr(\Psi_2) \geq \frac{1}{8 \cdot r'(A) + 1} \cdot \Pr(\Psi_1) \cdot \Pr(\Psi_2)$$

where we used that $\frac{(\alpha_1)^2}{(\alpha_1)^2 + 2 \cdot r'(A)}$ is increasing for $(\alpha_1)^2 \geq \frac{1}{4}$. It remains to prove that $\Pr(\Psi_1), \Pr(\Psi_2) = \Theta(1)$. We observe that, since $\alpha_i \sim \mathcal{N}(0,1)$, we have

$$\Pr(\Psi_1) = \Pr\left(|\alpha_1| \geq \frac{1}{2}\right) = \Theta(1)$$

Similarly, by Markov inequality, we have

$$\Pr(\Psi_2) = \Pr\left(\sum_{i=1}^d (\lambda_i^A)^2 (\alpha_i)^2 \leq 2r'(A)\right) \geq 1 - \Pr\left(\sum_{i=1}^d (\lambda_i^A)^2 (\alpha_i)^2 > 2r'(A)\right) \geq \frac{1}{2} \qquad \square$$

Next, we prove that a random vector $s \in \mathbf{S}^{d-1}$ achieves an $O(r'(A))$-approximation to the optimum of the main objective function as in Equation (7.6.1).

Lemma 7.7.3. *A random vector s which is picked uniformly at random from the d-dimensional unit sphere, is an $O(r')$-approximation to the optimum value of the objective function in Equation (7.6.1), where r' is the maximum stable rank of matrices $A_1, \cdots, A_N$.*

Proof: We assume that the vector s is generated via the same process as in the proof of

Lemma 7.7.2. It follows that

$$\mathbf{E}\left[\frac{\sum_{i=1}^{d}(\lambda_i^A)^4\langle s,U_i^A\rangle^2}{\sum_{i=1}^{d}(\lambda_i^A)^2\langle s,U_i^A\rangle^2}\right] \geq \mathbf{E}\left[\frac{(\alpha_1)^2}{\sum_{i=1}^{d}(\lambda_i^A)^2\cdot(\alpha_i)^2}\right] = \Omega(1/r'(A)) \qquad \square$$

7.8. Generalization Bounds

Define the loss function as $\mathsf{L}(s) \triangleq -\mathbf{E}_{A\sim\mathcal{D}}[\frac{\sum_{i=1}^{d}\left(\lambda_i^A\right)^4\langle s,U_i^A\rangle^2}{\sum_{i=1}^{d}\left(\lambda_i^A\right)^2\langle s,U_i^A\rangle^2}]$ We want to find a vector $s\in\mathbf{S}^{d-1}$ to minimize $\mathsf{L}(s)$, where $\mathbf{S}^{d-1}$ is the d-dimensional unit sphere. Since $\mathcal{D}$ is unknown, we are optimizing $\overline{\mathsf{L}}_{\mathsf{Tr}}(s) \triangleq -\frac{1}{N}\sum_{j=1}^{N}[\frac{\sum_{i=1}^{d}\left(\lambda_i^{A_j}\right)^4\langle s,U_i^{A_j}\rangle^2}{\sum_{i=1}^{d}\left(\lambda_i^{A_j}\right)^2\langle s,U_i^{A_j}\rangle^2}]$.

The importance of robust solutions. We start by observing that if s minimizes the training loss $\overline{\mathsf{L}}$, it is not necessarily true that s is the optimal solution for the population loss L. For example, it could be the case that $\{A_j\}_{j=1,\cdots,N}$ are diagonal matrices with only 1 non-zeros on the top row, while $s=(\epsilon,\sqrt{1-\epsilon^2},0,\cdots,0)$ for ε close to 0. In this case, we know that $\overline{\mathsf{L}}_{\mathsf{Tr}}(s)=-1$, which is at its minimum value.

However, such a solution is not robust. In the population distribution, if there exists a matrix A such that $A=\mathrm{diag}(\sqrt{1-100\varepsilon^2},10\varepsilon,0,0,\cdots,0)$, insert s into (7.6.1),

$$\frac{\sum_{i=1}^{d}\left(\lambda_i^A\right)^4\langle s,U_i^A\rangle^2}{\sum_{i=1}^{d}\left(\lambda_i^A\right)^2\langle s,U_i^A\rangle^2} = \frac{(1-100\varepsilon^2)^2\varepsilon^2+10^4\varepsilon^4(1-\varepsilon^2)}{(1-100\varepsilon^2)\varepsilon^2+100\varepsilon^2(1-\varepsilon^2)} < \frac{\varepsilon^2+10^4\varepsilon^4}{101\varepsilon^2-100\varepsilon^4} = \frac{1+10^4\varepsilon^2}{101-100\varepsilon^2}$$

The upper bound is very close to 0 if ε is small enough. This is because when the denominator is extremely small, the whole expression is susceptible to minor perturbations on A. This is a typical example showing the importance of finding a *robust* solution. Because of this issue, we will show a generalization guarantee for a *robust* solution s.

Definition of robust solution. First, define event $\zeta_{A,\delta,s} \triangleq \mathbf{1}\left[\sum_{i=1}^{d}\left(\lambda_i^A\right)^2\langle s,U_i^A\rangle^2<\delta\right]$, which is the denominator in the loss function. Ideally, we want this event to happen with a small probability, which indicates that for most matrices, the denominator is large, therefore s is robust in general. We have the following definition of robustness.

Definition 7.8.1 ((ρ, δ)-robustness). s is (ρ, δ)-robust with respect to $\mathcal{D}$ if $\mathbf{E}_{A\sim\mathcal{D}}[\zeta_{A,\delta,s}] \leq \rho$. s is (ρ, δ)-robust with respect to Tr if $\mathbf{E}_{A\sim\mathsf{Tr}}[\zeta_{A,\delta,s}] \leq \rho$.

For a given $\mathcal{D}$, we can define robust solution set that includes all robust vectors.

Definition 7.8.2 ((ρ, δ)-robust set). $\mathbf{M}_{\mathcal{D},\rho,\delta}$ is defined to be the set of all vectors $s \in \mathbf{S}^{d-1}$ s.t. s is (ρ, δ)-robust with respect to $\mathcal{D}$.

Estimating $\mathbf{M}_{\mathcal{D},\rho,\delta}$. The drawback of the above definition is that $\mathbf{M}_{\mathcal{D},\rho,\delta}$ is defined by the unknown distribution $\mathcal{D}$, so for fixed δ and ρ, we cannot tell whether s is in $\mathbf{M}_{\mathcal{D},\rho,\delta}$ or not. However, we can estimate the robustness of s using the training set. Specifically, we have the following lemma:

Lemma 7.8.3 (Estimating robustness). *For a training set Tr of size N sampled uniformly at random from $\mathcal{D}$, and a given $s \in \mathbb{R}^d$, a constant $1 > \eta > 0$, if s is (ρ, δ)-robust with respect to Tr, then with probability at least $1 - e^{-\frac{\eta^2 \rho N}{2}}$, s is $\left(\frac{\rho}{1-\eta}, \delta\right)$-robust with respect to $\mathcal{D}$.*

Proof: Suppose that $\Pr_{A\sim\mathcal{D}}[\zeta_{A,\delta,s}] = \rho_1$, which means $\mathbf{E}\left[\sum_{A_i\in\mathsf{Tr}} \zeta_{A_i,\delta,s}\right] = \rho_1 N$. Since events $\zeta_{A_i,\delta,s}$'s are 0-1 random variables, by Chernoff bound,

$$\Pr\left(\sum_{A_i\in\mathsf{Tr}} \zeta_{A_i,\delta,s} \leq (1-\eta)\rho_1 N\right) \leq e^{-\frac{\eta^2\rho_1 N}{2}}$$

If $\rho_1 < \rho < \rho/(1-\eta)$, our claim is immediately true. Otherwise, we know $e^{-\frac{\eta^2\rho_1 N}{2}} \leq e^{-\frac{\eta^2\rho N}{2}}$. Hence, with probability at least $1 - e^{-\frac{\eta^2\rho N}{2}}$, $N\rho = \sum_{A_i\sim\mathsf{Tr}} \zeta_{A_i,\delta,s} > (1-\eta)\rho_1 N$. This implies that with probability at least $1 - e^{-\frac{\eta^2\rho N}{2}}$, $\rho_1 \leq \frac{\rho}{1-\eta}$. □

Lemma 7.8.3 implies that for a fixed solution s, if it is (ρ, δ)-robust in Tr, it is also $(O(\rho), \delta)$-robust in $\mathcal{D}$ with high probability. However, Lemma 7.8.3 only works for a single solution s, but there are infinitely many potential s on the d-dimensional unit sphere.

To remedy this problem, we discretize the unit sphere to bound the number of potential solutions. Classical results tell us that discretizing the unit sphere into a grid of edge length $\frac{\varepsilon}{\sqrt{d}}$ gives $\frac{C}{\varepsilon^d}$ points on the grid for some constant C (e.g., see Section 3.3 in [98] for more details). We will only consider these points as potential solutions, denoted as $\hat{\mathbf{B}}^d$. Thus, we

can find a "robust" solution $s \in \hat{\mathbf{B}}^d$ with decent probability, using Lemma 7.8.3 and union bound.

Lemma 7.8.4 (Picking robust s). *For a fixed constant $\rho > 0, 1 > \eta > 0$, with probability at least $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 \rho N}{2}}$, any (ρ, δ)-robust $s \in \hat{\mathbf{B}}^d$ with respect to Tr is $\left(\frac{\rho}{1-\eta}, \delta\right)$-robust with respect to $\mathcal{D}$.*

Since we are working on the discretized solution, we need a new definition of robust set.

Definition 7.8.5 (Discretized (ρ, δ)-robust set). $\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}$ is defined to be the set of all vector $s \in \hat{\mathbf{B}}^d$ s.t. s is (ρ, δ)-robust with respect to $\mathcal{D}$.

Using similar arguments as Lemma 7.8.4, we know all solutions from $\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}$ are robust with respect to Tr as well.

Lemma 7.8.6. *With probability at least $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 \rho N}{3}}$, for a constant $\eta > 0$, all solutions in $\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}$, are $((1+\eta)\rho, \delta)$-robust with respect to Tr.*

Proof: Consider a fixed solution $s \in \hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}$. Note that $\mathbf{E}\left[\sum_{A_i \in \mathsf{Tr}} \zeta_{A_i,\delta,s}\right] = \rho N$ and $\zeta_{A_i,\delta,s}$ are 0-1 random variables. Therefore by Chernoff bound,

$$\Pr\left(\sum_{A_i \in \mathsf{Tr}} \zeta_{A_i,\delta,s} \geq (1+\eta)\rho N\right) \leq e^{-\frac{\eta^2 \rho N}{3}}.$$

Hence, with probability at least $1 - e^{-\frac{\eta^2 \rho N}{3}}$, s is $((1+\eta)\rho, \delta)$-robust with respect to Tr. By union bound on all points in $\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \subseteq \hat{\mathbf{B}}^d$, the proof is complete. □

7.8.1. Generalization Bound for Robust Solutions

Finally, we show the generalization bounds for robust solutions. To this can we use Rademacher complexity to prove generalization bound. Define Rademacher complexity $R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr})$ as

$$R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr}) \triangleq \frac{1}{N} \mathop{\mathbf{E}}_{\sigma \sim \{\pm 1\}^N} \sup_{s \in \hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \left[\frac{\sigma_j \sum_{i=1}^d \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j} \rangle^2}{\sum_{i=1}^d \left(\lambda_i^{A_j}\right)^2 \langle s, U_i^{A_j} \rangle^2} \right].$$

$R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr})$ is handy, because we have the following theorem (notice that the loss function takes value in $[-1, 0]$):

Theorem 7.8.7 (Theorem 26.5 in [160]). *Given constant $\delta > 0$, with probability of at least $1-\delta$, for all $s \in \hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}$,*

$$\mathsf{L}(s) - \overline{\mathsf{L}}_{\mathsf{Tr}}(s) \le 2R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr}) + 4\sqrt{\frac{2\log(4/\delta)}{N}}$$

That means, it suffices to bound $R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr})$ to get the generalization bound.

Lemma 7.8.8 (Bound on $R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr})$). *For a constant $\eta > 0$, with probability at least $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 pN}{3}}$, $R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr}) \le (1+\eta)\rho + \frac{1-\delta}{2\delta} + \frac{d}{\sqrt{N}}$.*

Proof: Define $\rho' = (1+\eta)\rho$. By Lemma 7.8.6, we know that with probability $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 pN}{3}}$, any $s \in \hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}$ is (ρ', δ)-robust with respect to Tr, hence $\sum_{A \in \mathsf{Tr}} \zeta_{A,\delta,s} \le \rho' N$. The analysis below is conditioned on this event.

Define $h_{A,\delta,s} \triangleq \max\{\delta, \sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle^2\}$. We know that with probability $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 pN}{3}}$,

$$\begin{aligned} N \cdot R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr}) &= \mathbf{E}_{\sigma \sim \{\pm 1\}^N} \sup_{s \in \hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^N \frac{\sigma_j \sum_{i=1}^d \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j} \rangle^2}{\sum_{i=1}^d \left(\lambda_i^{A_j}\right)^2 \langle s, U_i^{A_j} \rangle^2} \\ &\le \rho' N + \mathbf{E}_\sigma \sup_{s \in \hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^N \frac{\sigma_j \sum_{i=1}^d \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j} \rangle^2}{h_{A,\delta,s}} \end{aligned} \tag{7.8.1}$$

where (7.8.1) holds because by definition, $h_{A,\delta,s} \ge \sum_{i=1}^d (\lambda_i^A)^2 \langle s, U_i^A \rangle^2$ if and only if $\zeta_{A,\delta,s} = 1$, which happens for at most $\rho' N$ matrices. Note that for any matrix A_j,

$$\frac{\sigma_j \sum_{i=1}^d \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j} \rangle^2}{\sum_{i=1}^d \left(\lambda_i^{A_j}\right)^2 \langle s, U_i^{A_j} \rangle^2} \le 1.$$

Now,

$$\begin{aligned}
&\mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \frac{\sigma_j \sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2}{h_{A,\delta,s}} \\
\leq\ & \mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} (\mathbf{1}_{\sigma_j=1} \frac{\sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2}{\delta} - \mathbf{1}_{\sigma_j=-1} \sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2) \qquad (7.8.2) \\
=\ & \mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \left(\sigma_j \sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2 + \mathbf{1}_{\sigma_j=1} \sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2 (\frac{1}{\delta} - 1) \right) \\
\leq\ & \frac{N}{2\delta} - \frac{N}{2} + \mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \sigma_j \sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2 \qquad (7.8.3)
\end{aligned}$$

The first inequality (7.8.2) holds as $\frac{\sum_{i=1}^{d}\left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2}{h_{A,\delta,s}} \in [\delta, 1]$. It remains to bound the last term (7.8.3).

$$\mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \sigma_j \sum_{i=1}^{d} \left(\lambda_i^{A_j}\right)^4 \langle s, U_i^{A_j}\rangle^2 \leq \sum_{i=1}^{d} \mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \sigma_j \left\langle s, \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\rangle^2 \qquad (7.8.4)$$

By contraction lemma of Rademacher complexity, we have

$$\begin{aligned}
\mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \sigma_j \left\langle s, \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\rangle^2 &\leq \mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \sum_{j=1}^{N} \sigma_j \left\langle s, \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\rangle \\
&= \mathbf{E}_\sigma \sup_{s\in\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta}} \left\langle s, \sum_{j=1}^{N} \sigma_j \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\rangle \\
&\leq \mathbf{E}_\sigma \left\| \sum_{j=1}^{N} \sigma_j \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\|_2
\end{aligned}$$

Where the last inequality is by Cauchy-Schwartz inequality. Now, by Jensen's inequality,

$$\mathbf{E}_\sigma \left\| \sum_{j=1}^{N} \sigma_j \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\|_2 \leq \left(\mathbf{E}_\sigma \left\| \sum_{j=1}^{N} \sigma_j \left(\lambda_i^{A_j}\right)^2 U_i^{A_j} \right\|_2^2 \right)^{1/2} = \left(\sum_{j=1}^{N} \left(\lambda_i^{A_j}\right)^4 \right)^{1/2} \leq \sqrt{N} \qquad (7.8.5)$$

Combining (7.8.1), (7.8.3), (7.8.4) and (7.8.5), we have $R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr}) \leq \rho' + \frac{1-\delta}{2\delta} + \frac{d}{\sqrt{N}}$. $\square$

Combining with Theorem 7.8.7, we get our main theorem:

Theorem 7.8.9 (Main Theorem). *Given a training set* $\mathsf{Tr} = \{A_j\}_{j=1}^N$ *sampled uniformly from* $\mathcal{D}$*, and fixed constants* $1 > \rho \geq 0, \delta > 0, 1 > \eta > 0$*, if there exists a* (ρ, δ)*-robust solution* $s \in \hat{\mathbf{B}}^d$ *with respect to* Tr*, then with probability at least* $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 pN}{2}} - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 pN}{3(1-\eta)}}$*, for* $s \in \hat{\mathbf{B}}^d$ *that is a* (ρ, δ)*-robust solution with respect to* Tr*,*

$$\mathsf{L}(s) \leq \overline{\mathsf{L}}_{\mathsf{Tr}}(s) + \frac{2(1+\eta)\rho}{1-\eta} + \frac{1-\delta}{\delta} + \frac{2d}{\sqrt{N}} + 4\sqrt{\frac{2\log(4/\delta)}{N}}$$

Proof: Since we can find $s \in \hat{\mathbf{B}}^d$ s.t. s is (ρ, δ)-robust with respect to Tr, by Lemma 7.8.4, with probability $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 pN}{2}}$, s is $(\frac{\rho}{1-\eta}, \delta)$-robust with respect to $\mathcal{D}$. Therefore, $s \in \hat{\mathbf{M}}_{\mathcal{D},\frac{\rho}{1-\eta},\delta}$. By Lemma 7.8.8, with probability at least $1 - \frac{C}{\varepsilon^d} e^{-\frac{\eta^2 \rho N}{3(1-\eta)}}$, $R(\hat{\mathbf{M}}_{\mathcal{D},\rho,\delta} \circ \mathsf{Tr}) \leq \frac{\rho(1+\eta)}{1-\eta} + \frac{1-\delta}{2\delta} + \frac{d}{\sqrt{N}}$. Combined with Theorem 7.8.7, the proof is complete. $\square$

In summary, Theorem 7.8.9 states that if we can find a *robust* solution s which "fits" the training set then it generalizes to the test set.

Bibliography

[1] A. Aamand, P. Indyk, and A. Vakilian. (learned) frequency estimation algorithms under zipfian distribution. *arXiv preprint arXiv:1908.05198*, 2019.

[2] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–40, 2013.

[3] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Proc. 30th Annu. Sympos. Comput. Geom.* (SoCG), pages 271–279, 2014.

[4] A. Aghazadeh, R. Spring, D. LeJeune, G. Dasarathy, A. Shrivastava, and R. G. Baraniuk. Mission: Ultra large-scale feature selection using count-sketches. In *International Conference on Machine Learning* (ICML), 2018.

[5] S. Agrawal, M. Shadravan, and C. Stein. Submodular secretary problem with shortlists. In *10th Innovations in Theoretical Computer Science Conference* (ITCS), pages 1:1–1:19, 2019.

[6] K. J. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Proc. 38th Int. Colloq. Automata Lang. Prog.* (ICALP), pages 526–538, 2011.

[7] K. J. Ahn and S. Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Proc. 27th ACM Sympos. Parallel Alg. Arch.* (SPAA), pages 202–211, 2015.

[8] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proc. 23rd ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 459–467, 2012.

[9] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proc. 31st ACM Sympos. on Principles of Database Systems* (PODS), pages 5–14, 2012.

[10] Z. Allen-Zhu and Y. Li. Lazysvd: even faster svd decomposition yet without agonizing pain. In *Advances in Neural Information Processing Systems*, pages 974–982, 2016.

[11] Z. Allen-Zhu and L. Orecchia. Nearly-linear time positive LP solver with faster convergence rate. In *Proc. 47th Annu. ACM Sympos. Theory Comput.* (STOC), pages 229–236, 2015.

[12] Z. Allen-Zhu and L. Orecchia. Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. In *Proc. 26th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1439–1456, 2015.

[13] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

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

[15] B. Aronov, E. Ezra, and M. Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, 2010.

[16] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[17] 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.

[18] S. Assadi, N. Karpov, and Q. Zhang. Distributed and streaming linear programming in low dimensions. In *Proc. 38th ACM Sympos. on Principles of Database Systems* (PODS), pages 236–253, 2019.

[19] S. Assadi and S. Khanna. Tight bounds on the round complexity of the distributed maximum coverage problem. In *Proc. 29th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 2412–2431, 2018.

[20] 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.

[21] S. Assadi, S. Khanna, Y. Li, and G. Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proc. 27th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1345–1364, 2016.

[22] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680, 2014.

[23] B. Bahmani, A. Goel, and K. Munagala. Efficient primal-dual graph algorithms for mapreduce. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–78, 2014.

[24] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik. Learning to branch. In *International Conference on Machine Learning* (ICML), pages 353–362, 2018.

[25] L. Baldassarre, Y.-H. Li, J. Scarlett, B. Gözcü, I. Bogunovic, and V. Cevher. Learning-based compressive subsampling. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):809–822, 2016.

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

[27] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Sys. Sci.*, 68(4):702–732, 2004.

[28] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10, 2002.

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

[30] G. Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.

[31] 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.

[32] J. Blasiok. Optimal streaming and tracking distinct elements with high probability. In *Proc. 29th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 2432–2448, 2018.

[33] A. Bora, A. Jalal, E. Price, and A. G. Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning* ICML, pages 537–546, 2017.

[34] C. Boutsidis and A. Gittens. Improved matrix algorithms via the subsampled randomized hadamard transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, 2013.

[35] R. S. Boyer and J. S. Moore. Mjrty—a fast majority vote algorithm. In *Automated Reasoning*, pages 105–117. 1991.

[36] O. Boykin, A. Bryant, E. Chen, ellchow, M. Gagnon, M. Nakamura, S. Noble, S. Ritchie, A. Singhal, and A. Zymnis. Algebird. https://twitter.github.io/algebird/, 2016.

[37] V. Braverman, S. R. Chestnut, N. Ivkin, J. Nelson, Z. Wang, and D. P. Woodruff. Bptree: An ℓ_2 heavy hitters algorithm using constant memory. In *Proc. 36th ACM Sympos. on Principles of Database Systems* (PODS), pages 361–376, 2017.

[38] V. Braverman, S. R. Chestnut, N. Ivkin, and D. P. Woodruff. Beating countsketch for heavy hitters in insertion streams. In *Proc. 48th Annu. ACM Sympos. Theory Comput.* (STOC), pages 740–753, 2016.

[39] CAIDA. Caida internet traces 2016 chicago. http://www.caida.org/data/monitors/passive-equinix-chicago.xml.

[40] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[41] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity*, pages 107–117, 2003.

[42] 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.

[43] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. 29th Int. Colloq. Automata Lang. Prog.* (ICALP), pages 693–703, 2002.

[44] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. 29th Int. Colloq. Automata Lang. Prog.* (ICALP), pages 693–703, 2002.

[45] 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.

[46] C. Chekuri, S. Gupta, and K. Quanrud. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages, and Programming*, pages 318–330, 2015.

[47] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[48] 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.

[49] R. Chitnis, G. Cormode, H. Esfandiari, M. Hajiaghayi, A. McGregor, M. Monemizadeh, and S. Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proc. 27th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1326–1344, 2016.

[50] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[51] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete Comput. Geom.*, 37(1):43–58, 2007.

[52] K. L. Clarkson and D. P. Woodruff. Numerical linear algebra in the streaming model. In *Proc. 41st Annu. ACM Sympos. Theory Comput.* (STOC), pages 205–214, 2009.

[53] K. L. Clarkson and D. P. Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):54, 2017.

[54] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proc. 47th Annu. ACM Sympos. Theory Comput.* (STOC), pages 163–172, 2015.

[55] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

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

[57] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[58] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 44–55. SIAM, 2005.

[59] H. Dai, E. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6351–6361, 2017.

[60] D. Davidov, E. Gabrilovich, and S. Markovitch. Parameterized generation of labeled datasets for text categorization based on a hierarchical directory. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 250–257, 2004.

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

[62] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proc. 10th Annu. European Sympos. Algorithms* (ESA), pages 348–360, 2002.

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

[64] D. L. Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[65] P. Drineas, R. Kannan, and M. W. Mahoney. Sampling sub-problems of heterogeneous Max-Cut problems and approximation algorithms. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 57–68, 2005.

[66] F. Dzogang, T. Lansdall-Welfare, S. Sudhahar, and N. Cristianini. Scalable preference learning from data streams. In *Proc. 24th Int. Conf. World Wide Web* (WWW), pages 885–890, 2015.

[67] 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.

[68] A. Ene, S. Har-Peled, and B. Raichel. Geometric packing under non-uniform constraints. In *Proc. 28th Annu. Sympos. Comput. Geom.* (SoCG), pages 11–20, 2012.

[69] P. Erdös. On a lemma of littlewood and offord. *Bulletin of the American Mathematical Society*, 51(12):898–902, 1945.

[70] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems* (TOCS), 21(3):270–313, 2003.

[71] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.* (STOC), pages 434–444, 1988.

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

[73] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[74] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[75] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[76] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[77] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

[78] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proc. 25th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 707–717, 2014.

[79] A. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

[80] 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.

[81] S. Gollapudi and D. Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning* (ICML), pages 2319–2327, 2019.

[82] A. Goyal, H. Daumé III, and G. Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1093–1103, 2012.

[83] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

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

[85] A. Gronemeier. Asymptotically optimal lower bounds on the NIH-Multi-Party information complexity of the AND-function and disjointness. In *26th International Symposium on Theoretical Aspects of Computer Science,* (STACS), pages 505–516, 2009.

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

[87] S. Guha and A. McGregor. Lower bounds for quantile estimation in random-order and multi-pass streaming. In *Proc. 34th Int. Colloq. Automata Lang. Prog.* (ICALP), volume 4596 of *Lect. Notes in Comp. Sci.*, pages 704–715, 2007.

[88] S. Guha and A. McGregor. Tight lower bounds for multi-pass stream computation via pass elimination. In *Proc. 35th Int. Colloq. Automata Lang. Prog.* (ICALP), volume 5125 of *Lect. Notes in Comp. Sci.*, pages 760–772, 2008.

[89] S. Guha, A. McGregor, and D. Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proc. 34th ACM Sympos. on Principles of Database Systems* (PODS), pages 241–247, 2015.

[90] V. Guruswami and K. Onak. Superlinear lower bounds for multipass graph processing. In *Proc. 28th Conf. Comput. Complexity* (CCC), pages 287–298, 2013.

[91] A. Hadian and T. Heinis. Interpolation-friendly b-trees: Bridging the gap between algorithmic and learned indexes. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology* (EDBT), pages 710–713, 2019.

[92] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[93] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84, 2017.

[94] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254, 2016.

[95] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[96] P. Hand and V. Voroninski. Global guarantees for enforcing deep generative priors by empirical risk. In *Conference On Learning Theory* (COLT), pages 970–978, 2018.

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

[98] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[99] S. Har-Peled and K. Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In *Proc. 23rd Annu. European Sympos. Algorithms* (ESA), volume 9294 of *Lect. Notes in Comp. Sci.*, pages 717–728.

[100] S. Har-Peled and M. Sharir. Relative (p, ε)-approximations in geometry. *Discrete Comput. Geom.*, 45(3):462–496, 2011.

[101] H. He, H. Daume III, and J. M. Eisner. Learning to search in branch and bound algorithms. In *Advances in neural information processing systems* (NeurIPS), pages 3293–3301, 2014.

[102] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. pages 107–118, 1998.

[103] C.-Y. Hsu, P. Indyk, D. Katabi, and A. Vakilian. Learning-based frequency estimation algorithms. *International Conference on Learning Representations* (ICLR), 2019.

[104] N. Imamoglu, Y. Oishi, X. Zhang, G. Ding, Y. Fang, T. Kouyama, and R. Nakamura. Hyperspectral image dataset for benchmarking on salient object detection. In *Tenth International Conference on Quality of Multimedia Experience, (QoMEX)*, pages 1–3, 2018.

[105] 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.

[106] P. Indyk, S. Mahabadi, R. Rubinfeld, A. Vakilian, and A. Yodpinyanee. Set cover in sub-linear time. In *Proc. 29th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 2467–2486, 2018.

[107] P. Indyk and A. Vakilian. Tight trade-offs for the maximum k-coverage problem in the general streaming model. In *Proc. 38th ACM Sympos. on Principles of Database Systems* (PODS), pages 200–217, 2019.

[108] P. Indyk and D. P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proc. 37th Annu. ACM Sympos. Theory Comput.* (STOC), pages 202–208, 2005.

[109] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[110] H. Jowhari, M. Sağlam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58, 2011.

[111] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[112] D. M. Kane, J. Nelson, and D. P. Woodruff. Revisiting norm estimation in data streams. *arXiv preprint arXiv:0811.3648*, 2008.

[113] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proc. 29th ACM Sympos. on Principles of Database Systems* (PODS), pages 41–52, 2010.

[114] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017.

[115] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 312–320, 1982.

[116] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems* (TODS), 28(1):51–55, 2003.

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

[118] M. Khani, M. Alizadeh, J. Hoydis, and P. Fleming. Adaptive neural signal detection for massive MIMO. *CoRR*, abs/1906.04610, 2019.

[119] 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.

[120] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data* (SIGMOD), pages 489–504, 2018.

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

[122] 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.

[123] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proc. 23rd ACM Sympos. Parallel Alg. Arch.* (SPAA), pages 85–94, 2011.

[124] Y. T. Lee and A. Sidford. Path finding methods for linear programming: Solving linear programs in $O(vrank)$ iterations and faster algorithms for maximum flow. In *Proc. 55th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 424–433, 2014.

[125] E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588, 2013.

[126] J. E. Littlewood and A. C. Offord. On the number of real roots of a random algebraic equation. ii. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 35, pages 133–148. Cambridge University Press, 1939.

[127] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[128] T. Lykouris and S. Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning* ICML, pages 3302–3311, 2018.

[129] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998.

[130] 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. 2006.

[131] A. McGregor and H. T. Vu. Better streaming algorithms for the maximum coverage problem. In *20th International Conference on Database Theory* (ICDT), pages 22:1–22:18, 2017.

[132] X. Meng and M. W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proc. 45th Annu. ACM Sympos. Theory Comput.* (STOC), pages 91–100, 2013.

[133] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pages 398–412, 2005.

[134] C. Metzler, A. Mousavi, and R. Baraniuk. Learned d-amp: Principled neural network based compressive image recovery. In *Advances in Neural Information Processing Systems* (NeurIPS), pages 1772–1783, 2017.

[135] G. T. Minton and E. Price. Improved concentration bounds for count-sketch. In *Proc. 25th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 669–686, 2014.

[136] J. Misra and D. Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[137] M. Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems* (NeurIPS), pages 464–473, 2018.

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

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

[140] A. Mousavi, G. Dasarathy, and R. G. Baraniuk. Deepcodec: Adaptive sensing and recovery via deep convolutional neural networks. *arXiv preprint arXiv:1707.03386*, 2017.

[141] A. Mousavi, A. B. Patel, and R. G. Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 1336–1343, 2015.

[142] N. H. Mustafa, R. Raman, and S. Ray. Settling the apx-hardness status for geometric set cover. In *Proc. 55th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 541–550, 2014.

[143] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[144] J. Nelson and H. L. Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proc. 54th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 117–126, 2013.

[145] 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, 2008.

[146] N. Nisan. The communication complexity of approximate set packing and covering. In *Proc. 29th Int. Colloq. Automata Lang. Prog.* (ICALP), pages 868–875, 2002.

[147] A. Norouzi-Fard, J. Tarnawski, S. Mitrovic, A. Zandieh, A. Mousavifar, and O. Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning* (ICML), pages 3826–3835, 2018.

[148] 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.

[149] 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.

[150] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[151] M. Purohit, Z. Svitkina, and R. Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems* (NeurIPS), pages 9661–9670, 2018.

[152] 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.

[153] A. A. Razborov. On the distributional complexity of disjointness. *Theo. Comp. Sci.*, 106(2):385–390, 1992.

[154] P. Roy, A. Khan, and G. Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data* (SIGMOD), pages 1449–1463, 2016.

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

[156] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[157] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proc. 47th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 143–152, 2006.

[158] S. Schechter, C. Herley, and M. Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*, pages 1–8, 2010.

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

[160] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[161] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176, 2017.

[162] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (NeurIPS), pages 3104–3112, 2014.

[163] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[164] P. Talukdar and W. Cohen. Scaling graph-based semi supervised learning to large number of labels using count-min sketch. In *Artificial Intelligence and Statistics*, pages 940–947, 2014.

[165] M. Thorup and Y. Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012.

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

[167] D. Wang, S. Rao, and M. W. Mahoney. Unified acceleration method for packing and covering problems via diameter reduction. In *Proc. 43rd Int. Colloq. Automata Lang. Prog.* (ICALP), pages 50:1–50:13, 2016.

[168] J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[169] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in neural information processing systems* (NeurIPS), pages 1753–1760, 2009.

[170] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[171] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.

[172] 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.

[173] N. E. Young. Randomized rounding without solving the linear program. In *Proc. 6th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 170–178, 1995.

[174] N. E. Young. Nearly linear-work algorithms for mixed packing/covering and facility-location linear programs. *arXiv preprint arXiv:1407.3015*, 2014.

[175] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.