Sketching and Sampling Algorithms for High-Dimensional Data

Rajesh Jayaram

CMU-CS-21-118

July 2021

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

David Woodruff, Chair
Anupam Gupta
Andrej Risteski
Alexandr Andoni (Columbia)
Jelani Nelson (Berkeley)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the partial support from the Office of Naval Research (ONR) grant Number N00014-18-1-2562, and the National Science Foundation (NSF) under Grant Number CCF-1815840.

Keywords: Sketching, Streaming, Sampling, Numerical Linear Algebra

For my mother,
who loved the stars...

Und einer von allen Sternen
müßte wirklich noch sein.
Ich glaube, ich wüßte,
welcher allein
gedauert hat, –
welcher wie eine weiße Stadt
am Ende des Strahls in den Himmeln steht . . .

And one of all the stars
must surely still remain.
I think I know
which alone
endures –
which, at the end of its beam in the sky,
stands like a white city . . .

Klage, Das Buch der Bilder
-Rilke

Abstract

Sketching is a powerful technique which compresses high-dimensional datasets into lower-dimensional *sketches*, while preserving properties of interest. Sampling is a quintessential form of sketching, where one generates samples from the data to use as the basis for a sketch. Sketching and sampling methods have become ubiquitous across computer science, and are a central component of modern algorithm design.

This thesis studies the design and analysis of sketching and sampling algorithms for a variety of computational tasks. In particular, we make contributions to three closely related areas: Streaming and Distributed Algorithms, Numerical Linear Algebra, and Database Query Evaluation. Our contributions to these areas include:

Streaming and Distributed Algorithms:

- The first perfect L_p samplers, near-optimal algorithms for L_p estimation for $p \in (0, 1)$, improved sketches for the Earth Mover's Distance, and the introduction of the *adversarially robust* and *bounded deletion* streaming models.

Numerical Linear Algebra:

- The first property testing algorithms for matrix positive semi-definiteness, and the first input-sparsity solvers for linear regression on design matrices which are **(1)** a Kronecker product of several smaller matrices or **(2)** a database join of several smaller tables.

Database Query Evaluation:

- A characterization of the classes of *conjunctive queries* for which approximate counting and uniformly sampling can be accomplished in polynomial time. Along the way, we obtain the first polynomial time algorithm for estimating the number of strings of length n accepted by a non-deterministic finite automaton.

Contents

Acknowledgments

To begin, I must express how deeply grateful I am to my advisor, David Woodruff, who has been the center of my academic life. David consistently goes above and beyond any expectation for involvement with and attention to his students. He never hesitates to jump into the weeds to work with students through onerous technical challenges, or to spend hours explaining ideas and techniques. As a mentor, he has always put his students first—I am extremely grateful for his unwavering support and encouragement. Working with David has been a life-changing experience, and I am heartened by the knowledge that I will carry with me all that I have learned from him for the rest of my career.

In addition, I am especially thankful for my collaborations with Xi Chen, Amit Levi, and Erik Waingarten. Particularly amidst the tumultuous period of the COVID pandemic, our meetings were a beacon of encouragement and conviviality, wherein we laughed, lamented, thought deeply, and learned. Xi, whose contagious smile inevitably brings liveliness to each meeting, is one of the kindest people I know, and I am grateful for his support and invaluable perspective on research. To Amit, whose unabating compassion and thoughtfulness transforms a job focused on studying objects with "a beauty cold and austere" to one which feels warm, congenial, and human. To Erik, who taught me to slow down, enjoy the process of research, search for beautiful mathematics, and not sweat the anxieties of academia.

I would like to extend a special and warm gratitude towards my collaborators Marcelo Arenas, Luis Alberto Croquevielle, and Cristian Riveros. Marcelo, Alberto, and Cristian introduced me to new vistas in computer science, fostered fruitful connections, and served as invaluable mentors. The time I spent with them in Santiago while visiting PUC Chile was especially formative for me—it was here that I learned how deeply one can appreciate life in congruence to, and not in spite of, the academic career.

I am also extremely grateful to MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni for mentoring and hosting me as an intern at Google Research, which was an especially rewarding summer; I am excited to continue our work full time as part of their team.

In addition, I am immensely grateful for having had many other incredible collaborators and mentors over the past years: Ainesh Bakshi, Omri Ben-Eliezer, Nadiia Chepurko, Huaian Diao, John Kallaugher, Yi Li, Vasileios Nakos, Barna Saha, Alireza Samadian, Gokarna Sharma, Zhao Song, Wen Sun, Srikanta Tirthapura,

Eli Upfal, Peng Ye, Eylon Yogev, Qiuyi Zhang, and Samson Zhou. I also owe a great deal to the faculty in the Computer Science Department at CMU for their support and invaluable advice, especially Anupam Guptam, Andrej Risteski, and Ryan O'Donnell.

Thank you to Paul Valiant for introducing me to the world of algorithms, where he taught me how each algorithm and its proof is an heroic struggle between opposing forces which must be balanced (runtime, approximation, failure probability, ect.)—a view which has been both intellectually fruitful and truly inspiring.

In addition to great collaborators, I am tremendously lucky to have had the *best* friends in Pittsburgh during my time here: Ainesh Bakshi, Pedro Paredes, Roie Levin, Greg Kehne, Marina DiMarco, Nadiia Chepurko, Tim Chu, Liting Chen, Emily Black, Klas Leino, Aria Wang, Goran Zuzic, Nic Resch, David Wajc, Kevin Pratt, Costin Bădescu, Samson Zhou, and Zoe Wellner. A special thanks to Ainesh for being my academic brother and collaborator, among many more things I cannot begin to list. These people have meant the world to me during my time here, and I am exceptionally grateful to have had each of them in my life. I am also grateful to Mike Jehn, with whom I have hiked, camped, canoed, and explored all across Western Pennsylvania, and whose friendship I deeply cherish. Finally, I thank my lifelong friend Calyl, for being with me through the ceaseless ebb and flow of life — he is the most selfless and loyal individual I know, and I will greatly miss our many Pittsburgh adventures.

While much more could be said of the matter, I owe the world, and express my deepest gratitude, to Patrick McAuliffe and Mirjam Lord. They made me into the person I am today, and were a pivotal force in my life.

Lastly, I must thank my family, without whom my path to the present would have been surely impossible. To my father Mohan, who has incessantly and fearlessly fought to support me throughout, and despite, every imaginable circumstance in my life. To my brother Kailash, who is truly my best friend, and who I hope will never tire of hearing how lucky I am to have him. To my step-father Al, who is one of the most respectable men I know, and who has always been there for me. Lastly, I thank my mother, for instilling in me a passion for the unknown, and a love of the world.

Chapter 1

Introduction

This thesis concerns the design and analysis of *sketching algorithms*, known more generally as *dimensionality reduction*. Sketching is a powerful technique which compresses high-dimensional objects into lower-dimensional *sketches*, while preserving properties of interest. These sketches are more efficient to work with, and can stand in place of the original data. Growing out of classic results in mathematics from the 1980s, such as the Johnson-Lindenstrauss Lemma [JL84] and Bourgain's Theorem [Bou85], sketching techniques are now a ubiquitous part of algorithm design in both theory and practice.

While historically rooted in theoretical fields such as functional analysis, the current explosion of sketching techniques over the past decades is due to the proliferation of massive datasets. Namely, the advent of "big data", where companies such as Facebook and Google now process *petabytes* of data on a daily basis [DG08, fac16], has necessitated the reevaluation of our models for algorithm design. For instance, while the classical theory of algorithms established polynomial time as the baseline for efficiency, the sheer size of modern datasets renders polynomial- and often even *linear*-time and space algorithms to be impractical. In addition to these new demands for hyper-efficiency, the traditional algorithmic query models, such the *random access model*, often no longer apply to modern datasets. For instance, data may arrive in real-time as a stream, or may be partitioned across multiple distributed machines.

Amid these developments, the *sketching paradigm* has emerged as an attractive approach to address the manifold challenges posed by modern computational demands. The central tenant of this paradigm is to compress, or *sketch*, the input to a computational problem via an extremely efficient (often randomized) transformation, and then recover a solution to the problem from the sketch. This *sketch-and-solve* method yields algorithms with space, and often runtime, usage

significantly smaller than storing the original dataset. In many cases, the size of the sketch can be made to grow only *logarithmically*, or *polylogarithmically*, in the size of the data. Once a sketch has been generated, due to its hyper-compact size, ultrafast algorithms can be run on the sketch to recover approximations.

The sketching paradigm is best illustrated by an example. A canonical one is the usage of *random projections*, where one takes a set of high dimensional vectors $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^n$ and projects them onto a random k-dimensional subspace spanned by a matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$, where $k \ll n$. The celebrated Johnson-Lindenstrauss Lemma tells us that that distances between all pairs of vectors are approximately preserved even when k is *independent* of the ambient dimension n. Hence, from the considerably smaller k-dimensional sketches $\mathbf{S}x^{(1)}, \ldots, \mathbf{S}x^{(m)}$, we can recover significant geometric information about the original vectors.

The contributions of this thesis to the area of sketching are twofold. Firstly, we develop a range of new algorithms and techniques for fundamental sketching problems that have been previously studied in the literature. These results represent fundamental contributions to established fields such as dimensionality reduction, streaming, distributed algorithms, and numerical linear algebra. Secondly, a major focus of this thesis is the introduction of new application areas and computational models for the study of sketching. This includes both the application of sketching to areas beyond its traditional reach, such as utilizing sketching to resolve a fundamental counting problem in automata theory, as well as developing fresh perspectives on classical sketching domains, such as the *adversarially robust streaming model*.

Sampling as a Building Block for Sketches. A unifying theme of the algorithms in this thesis is the usage of *sampling methods*. Namely, many sketching tasks can be addressed by generating samples from some underlying distribution over the data. By an appropriate choice of distribution, statistical properties about the original dataset can be inferred from the samples. In other words, these samples form a sketch of the data.

For example, an internet search engine provider may want to compute statistics about the billions of terms searched each day, such as the most commonly searched keywords, or more complicated queries such as the *empirical entropy* of the distribution of searches. Storing such a large dataset in memory, or sending it across a distributed network, is extremely inefficient. Instead, the machines serving the requests can store a random sample of the searches to send back to a central machine to aggregate. These samples themselves are a small-space *sketch* of the entire dataset, and one could hope, for instance, to find the most commonly searched keywords or approximate the entropy from this sketch.

In the simplest traditional settings where data is available via random access, such as stored on a fixed array in memory, generating a uniform sample is trivial. However, with the advent of the aforementioned computational paradigms, even generating uniform samples has become a complicated and sometimes intractable task. In particular, data may only be *implicitly* accessible, such as via an online data stream, distributed over multiple machines, or the result of a database query. In these scenarios, aggregating all the data in a central machine, or fully evaluating the query, is computationally infeasible for modern datasets. These challenges demand a new theory of versatile sampling methods.

This thesis develops a variety of novel sampling techniques in addition to refining several known techniques—an example of the latter is *precision sampling*.[1] The focus will be both on the design of algorithms which can efficiently generate samples from a desired distribution, as well as the analysis of properties of these samples once generated. Throughout, we will see how a core set of mathematical tools, such as scaling by exponential variables and p-stable distributions, can be applied across application domains to generate samples from various distributions.

Starting Point: High-Dimensional Objects, Low-Dimensional Representations. We begin our foray into sketching by describing the following overarching scenario: there is a high-dimensional dataset, usually denoted x, which we are only *implicitly* given access to, and our goal is to answer some query $\mathcal{Q}(x)$ about the object. Whereas traditionally a centralized algorithm would be run on the full dataset x, the defining characteristic of the sketching approach is to approximately answer $\mathcal{Q}(x)$ without ever fully storing x. Examples include data streams, where $x \in \mathbb{R}^n$ is represented by a sequence of updates to its coordinates, distributed computation, where one has a network of k machines, each containing some vector $x^{(i)} \in \mathbb{R}^n$ so that $x = \sum_{i=1}^{k} x^{(i)}$, and relational database queries, where $x = \mathbf{X} \in \mathbb{R}^{n \times d}$ is the result of a query on several smaller tables (e.g., $\mathbf{X}$ is a database join of k tables $\mathbf{T}^{(1)}, \ldots, \mathbf{T}^{(k)}$).

This thesis studies the above setting within the context of three central application areas: **Streaming and Distributed Algorithms**, **Numerical Linear Algebra**, and **Database Query Evaluation**. The thesis is split into three parts, Part I, II and III, with one part dedicated to each of the three respective areas. In what follows, we give an brief description of these areas, before delving deeper in Sections 1.1, 1.2, and 1.3.

Streaming and Distributed Computation. The streaming model of computation has become increasingly important for the analysis of massive datasets, where the sheer size of the data and speed of arrival poses significant challenges for algorithm design. Examples of such streams

[1] This elegant technique, introduced in Section 3.2, will be central in Chapters 3, 5, 6, 8, and 10.

include real-time internet traffic, distributed sensor networks, and enormous streams of files read from an external memory device. The goal of a *streaming algorithm* is to generate and store a compact sketch of the stream as it arrives, which is then used to answer queries about the data. Given their ubiquity, a large field of research is devoted to designing hyper-efficient algorithms for these streams. Part I of this thesis will be devoted to this goal.

Streaming algorithms can be viewed as a particularly strong form of sketching algorithms, since they must be able to maintain their sketch as the underlying dataset is modified or updated. This property also means that streaming sketches are generally mergable — given a sketch of $x^{(1)}$ and a sketch of $x^{(2)}$, one can often obtain a sketch of $x^{(1)} + x^{(2)}$. This *mergability* property is extremely useful, and establishes an intimate connection to the field of *distributed computation*, such as the MapReduce (MPC) or message-passing models, where the primary objective is to minimize *communication* across a network. Here, each machine in a distributed network generates a sketch of their local data via a streaming algorithm, and sends the compressed sketch across the network to be merged; as in streaming, the goal is to minimize the size of the sketch. As we will also see, many such sketches which originated in the streaming literature have also proved to be exceptionally applicable to the field of numerical linear algebra.

Due to the generality of streaming sketches and their applicability to many other areas of computation, it is often useful to study the field of sketching as a whole through the *lens of streaming*. Namely, given a large vector $x \in \mathbb{R}^n$ and a query $\mathcal{Q}(x)$, our first question is whether $\mathcal{Q}(x)$ can be approximated when x is observed in a stream. This is an approach which is advocated by this thesis. A small set of examples of our contributions in this area include

- The first sublinear, and in fact optimal, space algorithm to sample a coordinate $i \in [n]$ of $x \in \mathbb{R}^n$ with probability proportional to $|x_i|$, up to variation distance $1/\mathrm{poly}(n)$, when x is given by a stream of positive and negative updates to its coordinates.
- The first optimal space algorithm[2] to estimate the F_p-moment $\|x\|_p^p = \sum_i |x_i|^p$ to $(1 \pm \epsilon)$ error for $p \in (0, 1)$ in a stream of positive updates to the coordinates of x.
- The introduction of the *Adversarially Robust Streaming Model* and the *Bounded Deletion Streaming Model*, and the design of the first algorithms many streaming tasks in both models.

Numerical Linear Algebra (NLA). The modern field of numerical linear algebra has a distinguished history dating back to the work of von Neumann and Turing [Tur48, VNG47], although algorithmic methods in linear algebra date back considerably farther to the work of Chinese

[2]Up to $\log \log n$ and $\log \epsilon^{-1}$ factors

mathematicians employing Gaussian elimination two millennia ago [CBW99]. Conversely, the usage of randomization, and particularly sketching methods, for NLA is a fairly recent phenomenon which, nevertheless, has already been tremendously successful in revolutionizing the field. Randomized sketching methods now yield the best algorithms for various fundemental tasks, such as solving linear systems, regression, and linear programs [PV20, W$^+$14, CLS19, BLSS20].

The common theme of the approach to NLA taken in this thesis is the following: given a computational task on a large matrix $\mathbf{A}$, one first compresses $\mathbf{A}$ into a lower-dimensional sketched matrix $\mathbf{A}'$. From $\mathbf{A}'$, one solves the original computational task significantly faster with traditional algorithms. This template adheres to the previously described sketch-and-solve approach, and is the primary focus of Part II of this thesis. Specialized to this thesis, and keeping with the unifying theme of low-dimensional representations, is scenarios where the matrix $\mathbf{A}$ itself is defined only *implicitly*, and we would like to compute the sketch $\mathbf{A}'$ without even reading the full matrix $\mathbf{A}$. Examples include:

- The matrix $\mathbf{A}$ is a *Kronecker product* $\mathbf{A} = \mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(m)}$ of much smaller matrices $\mathbf{A}^{(i)}$. We give sketching algorithms for solving linear regression, when the constraint matrix is given by such a product $\mathbf{A}$, in time linear only in the factors $\mathbf{A}^{(i)}$.
- $\mathbf{A}$ is a *database join* $\mathbf{A} = \mathbf{T}^{(1)} \bowtie \mathbf{T}^{(2)} \bowtie \cdots \bowtie \mathbf{T}^{(m)}$ of much smaller tables $\mathbf{T}^{(i)}$. Similarly, we give the first sketching algorithms for solving regression on such matrices in time linear in the size of the tables $\mathbf{T}^{(i)}$, without ever computing the join $\mathbf{A}$.

Database Queries and Automata Theory. Databases queries are a classic example of high-dimensional objects with small representations. In this setting, we have a relational database $\mathcal{D}$, which is a set of tables $\mathcal{D} = \{\mathbf{T}^{(1)}, \ldots, \mathbf{T}^{(m)}\}$, and a query $\mathcal{Q}$, which is a logical expression taking as input a database $\mathcal{D}$ and outputting a new database $\mathcal{Q}(\mathcal{D})$, which is a set of "answers" to the query. A quintessential example of a query $\mathcal{Q}$ is a database join on the tables $\mathbf{T}^{(i)}$. Query evaluation is the task of computing $Q(\mathcal{D})$ given as input $(\mathcal{Q}, \mathcal{D})$.

One of many challenges in query evaluation is that the number of answers, namely the size of $\mathcal{Q}(\mathcal{D})$, can grow exponentially in the size of the input $(\mathcal{Q}, \mathcal{D})$.[3] Traditionally, to perform data mining on the result of a query $\mathcal{Q}$ on a database $\mathcal{D}$, one would first fully compute $\mathcal{Q}(\mathcal{D})$, store the result, and run a centralized algorithm on the data. With the massive scale of modern databases and SQL queries, this direct approach is no longer computationally feasible. On the other hand,

[3]There are numerous other challenges as well; for instance, even determining whether $\mathcal{Q}(\mathcal{D})$ is empty or not is NP-Hard for many of the common classes of queries studied in the literature.

notice that the high-dimensional database $\mathcal{Q}(\mathcal{D})$ admits a lower-dimensional representation via the small-scale tables $\{\mathbf{T}^{(1)}, \ldots, \mathbf{T}^{(m)}\}$ and the description of the query $\mathcal{Q}$. This elegantly positions query evaluation as a candidate for sketching methods. In particular, one could hope to generate a sketch of $\mathcal{Q}(\mathcal{D})$ by, for instance, generating uniform samples from $\mathcal{Q}(\mathcal{D})$ without ever computing it. This program is carried out in Part III of this thesis.

The central contribution in Part III is the characterization of the classes of *conjunctive queries* (CQ) for which sampling from $\mathcal{Q}(\mathcal{D})$ or approximating the size of $\mathcal{Q}(\mathcal{D})$ can be accomplished in polynomial time. Conjunctive queries are the most common class of queries used in database systems (e.g., they are equivalent to *select-from-where* queries in SQL), and the best studied in the literature. Perhaps surprisingly, the main ingredient in this result is the resolution of a long-standing open problem in automata theory—an area where sketching and dimensionality reduction are not commonly utilized. Nevertheless, our approach is based fundamentally upon sketching techniques, especially the method of sampling to generate compact sketches.

1.1 Streaming Algorithms

Data streams are ubiquitous in modern computing, and have been now for several decades. Examples of data streams include internet traffic logs, sensor networks, financial transaction data, database logs, and scientific data streams (such as huge experiments in particle physics, genomics, and astronomy). The motivating foundation for streaming is that the sheer size of modern datasets renders traditional algorithms requiring random access to the data infeasible; instead, streaming algorithms must answer queries about the data while seeing it sequentially, using as little working memory as possible. This situation epitomizes the *sketching paradigm*: streaming algorithms must maintain a small sketch of a high-dimensional object, by processing low-dimensional pieces of that object.

An illustrative example is an internet search engine, which receives tens of thousands of queries each second, and billions every day.[4] These queries contain many pieces of data, such as search terms, timestamps, location, and other meta-data. A local machine receiving a subset of this stream of data for statistical analysis will have to process the queries at a breakneck pace. In particular, it will be unable to store and aggregate all of the queries into main memory. Instead, such a machine would have to process updates on the fly, maintaining a sketch of the data small enough to fit in memory. Additionally, because such a large stream will be distributed

[4] See https://trends.google.com/

over multiple machines, one needs a way of merging such sketches together in order to perform aggregate queries.

A second and equally important class of example are those where the data is fixed. Consider an enormous collection of unstructured data, such as experiments from partial physics or biological data, stored on massive external hard-drives. Traditional algorithms that assume random access to the data will be exceedingly slow, since such hard drives have poor random access rates. In this setting, streaming algorithms excel, as they can process the data in a sequential fashion, which is a significantly faster access pattern for many devices.

The *data stream model* is a mathematical abstraction of the above scenarios, which captures the core constraints of data stream computation: namely, sequential access to the data. Specifically, the streaming model consists of a sequence of updates (the stream) to the coordinates of an underlying, high-dimensional vector $x \in \mathbb{R}^n$. The formal model is stated below.

Definition 1.1.1 (The Streaming Model.)**.** *Fix integers $n, m, M > 0$ such that $\log(mM) = O(\log n)$. In the streaming model, a vector $x \in \mathbb{R}^n$, known as the frequency vector, is initialized to $\vec{0}$. Then, it receives a sequence of updates $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$, where $i_t \in [n]$ and $\Delta \in \{-M, \ldots, M\}$. Each update (i, Δ) causes the change:*

$$x_{i_t} \leftarrow x_{i_t} + \Delta$$

At the end of the stream, the vector $x \in \mathbb{R}^n$ is given by $x_i = \sum_{t:i_t=i} \Delta_t$ for all $i \in [n]$.

The goal of a streaming algorithm is to observe the updates (i_t, Δ_t) sequentially, and at the end of the stream answer a query $Q(x)$ about the final frequency vector, using as little space and update time as possible.[5]

Since storing the vector x entirely would trivially require $O(n \log mN) = O(n \log n)$ bits of space, the goal of streaming algorithms is to maintain a *sublinear* space sketch of the data. Specifically, the gold standard in streaming is generally to design algorithms with *polylogarithmic*, instead of polynomial, dependency on the ambient dimension n.

Models of Streaming. Two prominent variants of the streaming model studied in the literature are the *insertion-only* and *turnstile* models. In the insertion-only model, we are promised that $\Delta_t \geq 0$ for every update Δ_t, whereas in the turnstile model updates may be positive or negative.

[5]Note that all space complexity bounds in this thesis will be stated in terms of the number of *bits* required by the algorithm.

The latter is more general, and is the model which is stated in Definition 1.1.1. The turnstile model can be further subdivided into the the *general turnstile* and the *strict turnstile* models, where in the latter updates may be positive or negative, but we are always promised that $x_i \geq 0$ at all points in the stream. The turnstile model captures a wider array of applications than the insertion only model. For instance, turnstile streams can model *differences* between streams, such as the Internet traffic discrepancy between separate routers in different geographical locations. Turnstile algorithms are also useful for dynamic data, such as financial data streams, where prices can fluctuate up and down [BBD+02, M+05].

Since the turnstile model of streaming is more general, the space complexity of streaming tasks is often higher for turnstile streams. In fact, many important streaming problems provably require a $\Theta(\log n)$ multiplicative factor more space for turnstile streams than for insertion-only streams. This complexity gap motivates having an intermediate model, which allows deletions, but only a limited number. This intermediate model is known as the *bounded deletion model*, and it provides a continuous link between the turnstile and streaming models. The bounded deletion model was first formalized in our paper [JW18a], and is the primary topic of Chapter 6.

History. The space complexity of a multitude of different queries in both the above models has been of tremendous interest in the algorithms community over the past three decades. Perhaps the first example of a streaming algorithm is the the work of Morris [Mor78], who demonstrated that it is possible to approximately maintain a counter during a stream of n (unit) increments in only $O(\log \log n)$ space, as opposed to the $O(\log n)$ space required to exactly store such a counter. Another foundational result is the work of Flajolet and Martin [FM85], which demonstrated an extremely efficient algorithm for estimating the number of distinct elements in a stream (this corresponds to the number of non-zero entries in the stream vector x). Also noteworthy is the work of Munro and Paterson [MP80], who gave sublinear space searching algorithms for finding the k-th largest value of a dataset given in a stream

The field of streaming algorithms did not truly explode until several decades later, due to the seminal paper of Alon, Matias, and Szegedy [AMS96], who tackled the problem of approximating the *moments* of a data stream. This explosion was additionally fueled by the growth of the internet in the late 1990s and early 2000s, which necessitated the development of space-efficient, real-time streaming algorithms. Since then, the field has grown enormously, and the space complexity of many fundamental tasks is now understood. An incomplete and informal list of such fundamental streaming problems is given in Figure 1.1.

Common Streaming Tasks

Given a stream vector $x \in \mathbb{R}^n$, accuracy $\epsilon \in (0,1)$, and parameter $p > 0$.

- L_p **Estimation.** Output an estimate $\tilde{L}_p$ such that, with probability at least $3/4$, we have
$$(1-\epsilon)\|x\|_p^p \leq \tilde{L}_p \leq (1+\epsilon)\|x\|_p^p$$
where $\|x\|_p^p = \sum_{i=1}^n |x_i|^p$.

- L_p **Sampling.** Sample a coordinate $i \in [n]$ with probability proportional to $|x_i|^p$.

- L_p**-Heavy hitters.** Output a set $S \subset [n]$ containing all i with $|x_i| > \epsilon\|x\|_p$ and containing no $j \in [n]$ with $|x_j| < (\epsilon/2)\|x\|_p$.

- **Inner Product Estimation.** Given two vectors $x, y \in \mathbb{R}^n$ in a stream, estimate their inner product $\langle x, y \rangle$.

- **Entropy Estimation.** Estimate the empirical entropy
$$H(x) = -\sum_{i=1}^n \frac{|x_i|}{\|x\|_1} \log\left(\frac{|x_i|}{\|x\|_1}\right)$$

Figure 1.1: An informal list of streaming tasks.

Connection to Distributed Computation and Linear Sketching. One important property of many streaming sketches is *mergability*, meaning that two sketches of distinct streams can be merged to form a sketch of the union of these streams. Consequently, streaming algorithms are especially well-suited for distributed computation. Here, each machine in a large distributed network can compute a sketch of its local data, and then aggregate the small space sketches at a centralized location. By sketching the data, this results in dramatically smaller amounts of communication between machines, which is the central resource of interest in distributed algorithms. Common models where such mergable sketches apply are the *MapReduce* (MPC) model and the *coordinator* model. Additionally, mergable sketches are useful for *distributed streams*, such as the distributed functional monitoring model [CMYZ12] (see Section 1.1.3).

A central example of a mergable sketch is that of a *linear sketch*. Specifically, a streaming algorithm is said to be a linear sketch if it generates a matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$ (called a *sketching matrix*) from some distribution $\mathcal{D}$ over $k \times n$ matrices. It then maintains the matrix-vector product $\mathbf{S} \cdot x \in \mathbb{R}^k$ in the stream (called a *sketch*), and outputs its response to a query as a function only of the sketch $\mathbf{S}x$. Naturally, setting $\mathbf{S}$ to be the identity matrix would allow for total recovery

of x. However, one generally wants significant compression of the dimension, so that $k \ll n$. A classic example is the Johnson-Lindenstrauss Lemma [JL84], which says that the Euclidean norm can be estimated to $(1 \pm \epsilon)$ error using a Gaussian sketching matrix with $k = O(\epsilon^{-2})$ rows.

Notice that linear sketches can be easily maintained in a stream, since an update (i, Δ) can be processed by setting $\mathbf{S}x \leftarrow \mathbf{S}x + \Delta \cdot \mathbf{S}_{*,i}$, where $\mathbf{S}_{*,i}$ is the i-th column of $\mathbf{S}$. Furthermore, notice that linear sketches can be easily merged by simply summing them, since $\mathbf{S}x + \mathbf{S}y = \mathbf{S}(x + y)$. We remark that essentially all turnstile streaming algorithms in the literature are linear sketches, and in fact there is a formal equivalence between the two (with some minor caveats) [LNW14b, AHLW16]. For some linear sketching algorithms, the output of the algorithm may also depend on the entries of the sketching matrix $\mathbf{S}$ in addition to the sketch $\mathbf{S}x$. For such algorithms, the matrix $\mathbf{S}$ is generally specified in very small space, such as through the usage of hash functions with limited independence. In other cases, such algorithms are only correct in the random oracle model (see below).

Remark 1 (The Random Oracle Model)**.** An nuanced yet important consideration in streaming is whether the algorithm is charged for storing random bits. In the *random oracle* model of streaming, this is not the case, and the algorithm is given random access to an arbitrarily long string of random bits which do not count against the space. In the *standard* model, algorithms are not given this power, and must pay for storing random bits. The random oracle model is very well studied, as it corresponds to the *public coin* model of communication complexity, which is the model most commonly used to prove lower bounds for the space complexity of streaming algorithms. Thus, most streaming lower bounds apply to the random oracle model [BYJKS04, Woo04, KNP+17].

However, having a random oracle is mostly a theoretical concern—in practice, any random bits which are consistently used by the algorithm must necessarily be stored by the algorithm. Thus, we often would like to ensure that our algorithms hold in the standard model. The process of transforming a random oracle algorithm into a standard algorithm is known as *derandomization*, and will be an important concern in this thesis. Unless otherwise stated, all space upper bounds in this thesis hold in the standard model.

Overview of Our Contributions in Streaming. This thesis focuses extensively on the development of streaming algorithms. Streaming is by now a relatively mature field in computer science, and the space complexity of many fundamental tasks in the standard streaming model (Definition 1.1.1) is fully understood. However, in the decades since the popularization of the standard model, many significant developments have occurred in both the theory and practice

which motivated the model to begin with. Such developments include the explosion of machine learning algorithms, massive graph databases (such as social networks), and AI-driven applications such as recommendation systems and autonomous vehicles. Unsurprisingly, the classical model often fails to capture important aspects of these modern developments, necessitating the development of new models in addition to the continual study of existing ones.

The contributions of this thesis to streaming have two distinct flavors. Firstly, we develop new algorithms and improved analyses for core streaming tasks, whose complexity has been the topic of intensive study for decades. The second flavor of results concern the development of *new* models and perspectives on the streaming literature. These latter results address the aforementioned developments in practice by designing sketches which handle scenarios important to modern computation, but not considered by the original models of streaming.

An example of the first flavor is the task of L_p norm estimation in a stream. Norm estimation is one of the oldest problems in the streaming literature, dating back to the AMS sketch [AMS96] and Indyk's celebrated p-stable estimator [Ind06], which, nevertheless, is still not fully understood in the insertion-only case. In Chapter 4, we resolve the complexity of L_p estimation in insertion-only streams for $p < 1$ (up to $\log \log n$ and $\log \epsilon^{-1}$ factors). As another example, in Chapter 3 we give the first sublinear space algorithms for perfect L_p sampling in turnstile streams—a central streaming task whose study dates back to the reservoir sampling algorithm of Vitter [Vit85a]–thereby answering a decade old open question of Cormode, Murthukrishnan, and Rozenbaum [CMR05]. Additionally, in Chapter 8, we improve the approximation of the classic *Quadtree* algorithm for estimating the Earth Mover's Distance between sets of points in $\mathbb{R}^d$, yielding the first improved streaming algorithms for the task since the work of Andoni, Indyk, and Krauthgamer [AIK08a].

Examples of results which fit into the second catagory of contributions include the introduction of the *Bounded Deletion Streaming Model* in Chapter 6. This model considers constrained turnstile streams which satisfy the often more realistic property that not all of the insertions to a stream are deleted by the end, allowing for improved algorithms as a result. Another example is the *Adversarially Robust Streaming Model*, introduced in Chapter 7, where the updates to the stream are chosen adaptively by a potentially malicious adversary, and the algorithm must nevertheless be correct at all time steps. Such adaptive streams are now ubiquitous in practice, yet classical streaming algorithms were unable to handle any amount of adaptivity. Both of these models introduced in this thesis have spurred a flurry of follow-up work, fostering the continual evolution and modernization of sketching as a field.

1.1.1 Chapter 3: Perfect Sampling in Data Streams

A central theme of this thesis is the generation of samples from an underlying distribution. The task of generating samples from data streams is a fundamental primitive for data analysis, and is the backbone for many approximation and statistical tasks. Given a high-dimensional stream vector $x \in \mathbb{R}^n$ and a non-negative *measure function* $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$, the goal of is to sample a coordinate $i \in [n]$ with probability proportional to $G(x_i)$. The most widely widely studied measure in the literature is the L_p *distribution* $\mathcal{D}_p(x)$, obtained by setting $G(x) = |x|^p$, so that $\mathbf{Pr}[i] = |x_i|^p / \|x\|_p^p$. Sampling from the L_p distribution is a powerful primitive which can be utilized to solve many important streaming problems (discussed below).

Ideally, we would like to design a small-space streaming algorithm which, after observing the stream $(i_1, \Delta_1), \ldots, (i_m, \Delta_m)$ which defines the vector $x \in \mathbb{R}^n$, outputs a random variable $X \sim \mathcal{D}_p(x)$. In the special case of the insertion only model and $p = 1$, such an idealistic goal is indeed possible. In fact, the solution for this case has been well-known for several decades, and is the classic reservoir sampling algorithm of Vitter [Vit85a], which uses $O(\log n)$ total bits of space to maintain a sample.

However, when negative updates are allowed (i.e., the turnstile model), or when $p \neq 1$, reservoir sampling fails, and the problem becomes more complicated. In fact, the question of whether such samplers using sublinear space even exist was posed by Cormode, Murthukrishnan, and Rozenbaum in 2005 [CMR05]. In 2010, Monemizadeh and Woodruff made partial progress towards this open question by demonstrating that, if one permits the sampler to be *approximately* correct, such samplers are indeed possible [MW10]. This latter notion is known as *approximate* L_p *sampling*, defined below.

Definition 1.1.2 (Approximate L_p Sampler)**.** *Fix any $p \in [0, 2]$,[6] parameters $\epsilon, \delta \in (0, 1)$, and let $x \in \mathbb{R}^n$ be a non-zero vector. Then an (ϵ, δ)-approximate L_p sampler is an algorithm which outputs a random variable $Z \in [n] \cup \{\perp\}$ such that $\mathbf{Pr}[X = \perp] \leq \delta$, and such that for every $i \in [n]$:*

$$\mathbf{Pr}[X = i \mid X \neq \perp] = (1 \pm \epsilon)\frac{|x_i|^p}{\|x\|_p^p} \pm O(n^{-c})$$

where $c \geq 1$ is an arbitrarily large constant. If $\epsilon = 0$, we say that the algorithm is a perfect L_p sampler.

[6] We note that for $p > 2$, any approximate L_p sampler (even with constant error ϵ) must use polynomial space $\Omega(n^{1-2/p})$. This follows from the fact that $O(1)$ copies of an L_p sampling sketches can be used to solve the $\Omega(1)$ L_p-heavy hitters problem, and the latter admits a well known $\Omega(n^{1-2/p})$ lower bound [BYJKS04, Jay09]. Thus, we are most interested in the regime $0 < p \leq 2$.

The $\perp$ symbol allows the algorithm to produce no output with some probability δ. However, conditioned on not outputting $\perp$, the distribution of the algorithm must be close to the true L_p distribution. The small additive error n^{-c} is seen as a "catastrophic failure" event, and is allowed by a perfect L_p sampler. While one might hope to remove this additive error, we show in Theorem 29 of Chapter 3 that this is impossible to accomplish in sublinear space.

Since their introduction in [MW10], L_p samplers have been utilized to develop alternative algorithms for important streaming problems, such as the heavy hitters problem, L_p estimation, cascaded norm estimation, and finding duplicates in data streams [AKO11, MW10, JST11, BOZ12]. The importance of L_p sampling sketches extends far beyond these applications, and have since been applied as a fundamental tool in distributed computation [WZ18, JSTW19, BMRT20], graph algorithms [AGM12a, AGM12c, Jin18, CLP18, KKP18, CKP+21, CFCHT20, MV20, KMM+20, FKN21, AD21, FKN21], metric approximation [CJLW20a], numerical linear algebra [LNW19, DJS+19, MRWZ20a], computational social choice [MSW20], data analysis [CPW20], and differential privacy [CGSS20], among many other applications.

The one-pass turnstile approximate L_p sampler introduced in [MW10] requires $\mathrm{poly}(\epsilon^{-1}, \log n)$ space, with large exponents. Following [MW10], several further improvements were made to the complexity of approximate L_p samplers by Andoni, Krauthgamer, and Onak, [AKO11], and finally by Jowhari, Sağlam, and Tardos [JST11], resulting in the following theorem.

Theorem 2 (Approximate L_p sampling [JST11])**.** *Fix $0 < p \leq 2$, and $\epsilon, \delta \in (0,1)$. Then there is a (ϵ, δ)-approximate L_p sampler for turnstile streams using $O(\frac{1}{\epsilon^{\max\{1,p\}}} \log^2 n \log \delta^{-1})$ bits of space for $p \in (0,2)\backslash\{1\}$, $O(\frac{\log \epsilon^{-1}}{\epsilon} \log^2 n \log \delta^{-1})$ bits of space for $p = 1$, and $O(\frac{1}{\epsilon^2} \log^3 n \log \delta^{-1})$ bits of space for $p = 2$.*

Furthermore, given that the algorithm produces a sampler $i \in [n]$, it can also produce an estimate $\tilde{x}_i$ such that $\tilde{x}_i = (1 \pm \epsilon)x_i$ with probability at least $1 - \delta$ in the same space.

The work of Jowhari, Sağlam, and Tardos additionally gave an $\Omega(\log^2 n)$-bit lower bound for (ϵ, δ)-approximate L_p samplers for any $p \geq 0$ and constants ϵ, δ. Recently, this lower bound was extended to $\Omega(\log^2 n \log \delta^{-1})$ [KNP+17] bits for any constant ϵ and $\delta > 2^{-n^{.99}}$. By Theorem 2, this closed the space complexity dependence on n, δ for $p \in (0,2) \setminus \{1\}$, and nearly closed the complexity for $p \in \{1,2\}$ in these parameters. However, a large gap in the dependency on ϵ remained. In fact, no lower bound in terms of ϵ was known, leaving open the possibility of a *perfect* L_p sampler. Such perfect samplers are necessary for many statistical tasks which generate multiple samples from the distribution, thus requiring high accuracy, such as the task of simulating random walks on graphs [Jin18, CKP+21].

Contribution: Perfect L_p Sampling

In Chapter 3, based on the results in our paper [JW18b], we explain the phenomenon of the lack of a lower bound in terms of ϵ by showing that ϵ need not enter the space complexity of an L_p sampler at all. In other words, we demonstrate the existence of *perfect L_p samplers* using $O(\log^2 n \log \delta^{-1})$-bits of space for $p \in (0, 2)$, thus resolving the space complexity of the problem. This space bound holds in the random oracle model, however, we show that the random oracle assumption can be removed with only a $(\log \log n)^2$ factor increase in the space.

For $p = 2$, our space is $O(\log^3 n \log \delta^{-1})$-bits, which matches the best known upper bounds in terms of n, δ, yet again has no dependency on ϵ. In addition, for $p < 2$ and the high probability regime of $\delta < 1/n$, we obtain an $O(\log^3 n)$-bit perfect L_p sampler, which also tightly matches the lower bound. Our main theorem is as follows:

Theorem 3 (Perfect L_p sampling)**.** *Fix any $p \in (0, 2]$, and $\delta \in (0, 1)$. Then there is a perfect L_p sampler (Definition 1.1.2) which outputs $\perp$ with probability at most δ. For $p = 2$, the sampler uses space $O(\log^3 n \log \delta^{-1})$. For $p < 2$ the space is $O(\log^2 n \log \delta^{-1})$ in the random oracle model, and without a random oracle the space is:*

$$O\left(\min\left\{\log^2 n \log \frac{1}{\delta}(\log \log n)^2, \log^2 n \log \frac{n}{\delta}\right\}\right)$$

Moreover, for any $\nu, \delta_2 \in (0, 1)$, conditioned on outputting a sample $i \in [n]$, the algorithm can output an estimate $\tilde{x}_i$ such that $\tilde{x}_i = (1 \pm \nu)x_i$ with probabiliaty at least $1 - \delta_2$, using an additive

$$O\left(\min\left\{\nu^{-2}, \nu^{-p} \log \frac{1}{\delta_2}\right\} \log n \log \frac{1}{\delta_2}\right)$$

bits of space.

The space complexity of providing an estimate of the frequency x_i of the sampled coordinate should be compared to the previous best known result of Theorem 2, which required $O(\nu^{-p} \log^2 n)$ bits of space. Our algorithm from Theorem 3 has since been applied several times towards developing high-accuracy statistical algorithms for data streams. In particular, our algorithm has been applied to improve the state of the art bounds for simulating random walks on graphs, where the graph is given in a stream or distributed over multiple machines [Jin18, CKP+21].[7]

[7]Specifically, the usage of our perfect samplers, as opposed to prior approximate samplers, has improved the space dependency on the variational distance ϵ of the walk from $\text{poly}(1/\epsilon)$ to $\log(1/\epsilon)$.

Precision Sampling. The algorithm of Theorem 3 is based on a powerful and elegant technique known as *precision sampling*. Precision sampling was introduced by Andoni, Krauthgamer, and Onak [AKO11, AKO10], and later refined by Jowhari, Sağlam, and Tardos [JST11], to develop approximate L_p samplers. At a high-level, precision sampling reduces the problem of sampling a coordinate from a high-dimensional vector $x \in \mathbb{R}^n$ to the problem of finding the largest coordinate in another high-dimensional vector $z \in \mathbb{R}^n$. Since finding large coordinates, or the *heavy hitters problem*, is well studied in streaming and admits small space algorithms, this reduction is extremely useful. The main contribution of Chapter 3 is a further refinement of the precision sampling technique to produce *perfect* samplers. The precision sampling framework, along with the new insights from Chapter 3, are a central tool in this thesis, and will be applied for a variety of purposes in Chapters 3, 5, 6, 8, and 10.

General Derandomization. One of several main technical challenges involved in the proof of Theorem 3 is the removal of the random oracle assumption. To accomplish this derandomization, we develop a *generic* technique which allows for the black-box derandomization of a large class of linear sketches. This theorem will not be sufficient alone to derandomize our algorithm, but is suitable for a number of other applications. For instance, it provides the first efficient derandomization of the count-sketch variant of Minton and Price [MP14], a discussion of which can be found in Section 3.4.2. Additionally, it gives an alternative and arguably simpler derandomization of the classic Indyk p-stable sketch [Ind06] than that of Kane, Nelson, and Woodruff [KNW10a].

General Measure Functions. In Theorem 29, we demonstrate that the additive n^{-c} error was necessary for sublinear space L_p samplers in the turnstile model. Specifically, we show that any L_p sampler with additive error $\gamma > 0$ requires at least $\Omega(\min\{n, \log \frac{1}{\gamma}\})$ bits of space. However, this does not rule out the possibility of truly perfect samplers, with $\gamma = 0$, in the insertion only model. In Section 3.6, we address this discrepancy by demonstrating that sublinear space truly perfect L_p samplers do exist in the insertion only streaming model. Specifically, we develop a framework for truly-perfect sampling in insertion only streams for *general* measure functions G. The framework applies for a wide class of measure functions G, including L_p sampling, concave functions, and a large number of specific measure functions, such as the $L_1 - L_2$, Fair, Huber, and Tukey estimators.

1.1.2 Chapter 4: Moment Estimation in Streaming and Distributed Models

A closely related area to streaming is the field of *distributed computation*. Sketching plays a fundamental role for distributed computation as well, especially due to the ability for sketches to be merged together across different machines. Dual to distributed algorithms is the field of *communication complexity*, which is, in some sense, the study of the amount of communication required to solve a distributed computational task. In fact, nearly all known lower bounds for the space complexity of streaming algorithms arise through the field of *communication complexity* [Woo04, WZ18, KNW10a, JST11, KNP+17, BCWY16, CKS03, WW15, LW13, MPTW16, JW09]. In Chapter 4, based on a joint work with David Woodruff [JW19], we study the fundamental problem of moment estimation in both streaming and distributed models, making progress towards resolving a long-standing open question in the area.

The problem of estimating the p-th moment of a stream vector $x \in \mathbb{R}^n$ is perhaps the most central in the streaming literature, beginning with the seminal work of Alon, Matias, and Szegedy [AMS96]. Recall that this problem is to estimate $\|x\|_p^p = \sum_{i=1}^n |x_i|^p$, a quantity which we denote by F_p, to a multiplicative error of $(1+\epsilon)$, for some error parameter $\epsilon > 0$. Since it was first studied, nearly two decades of research have been devoted to understanding the space and time complexity of this problem [CKS03, IW05, BYJKS04, Woo04, Ind06, KNW10a, BO10, KNPW11a, BKSV14, CCM16, BKSV14, BDN17, BVWY18].

For $p \in (0, 2]$,[8] the space complexity of F_p estimation in the turnstile model is now understood, with matching upper and lower bounds of $\Theta(\epsilon^{-2} \log n)$. For insertion only streams, however, there is a gap: the best known lower bound is $\Omega(\epsilon^{-2} + \log n)$ bits [Woo04], and in the random oracle model it is only $\Omega(\epsilon^{-2})$, whereas the best upper bound is to just run the turnstile $O(\epsilon^{-2} \log n)$-space algorithm. This means that there is a $O(\log n)$ gap in our understanding of the space complexity of one of the most fundamental streaming tasks.

In Chapter 4, we make progress towards understanding the complexity of insertion-only F_p estimation, as well as other important tasks such as entropy estimation and finding the heavy hitters, in both the streaming and distributed models of computation. Specifically, for $p < 1$, we resolve the space complexity by giving an $\tilde{O}(\epsilon^{-2} + \log n)$-bits of space upper bound, which is tight up to $\log \epsilon^{-1}$ and $\log \log n$ factors. In the random oracle model, our upper bound is $\tilde{O}(\epsilon^{-2})$,[9]

[8] For $p > 2$, it is known that polynomial in n (rather than poly-logarithmic) space is needed [CKS03, IW05], so we focus instead on the case of $p \in (0, 2]$.

[9] This space complexity is measured *between updates*. To read and process the $\Theta(\log n)$-bit identity of an update, the algorithm will use an additional $O(\log n)$-bit working memory tape during an update. Note that all lower bounds

which also matches the lower bound in this setting. For $1 < p \leq 2$, we prove an $\tilde{O}(\epsilon^{-2})$-bits of max-communication upper bound in the distributed models most frequently used to prove *lower bounds* for streaming. This result rules out a large and very commonly used class of approaches for proving lower bounds against the space complexity of streaming algorithms.

To formalize these contributions slightly, we must first introduce the **Message Passing Model**, which is one of the most general distributed models of computation [CRR14, PVZ12, WZ12, BEO+13]. In this model, there are m players, each positioned at a unique vertex in a graph $G = (V, E)$. The i-th player is given as input an integer vector $X_i \in \mathbb{Z}^n$. The goal of the players is to work together to jointly approximate some function $f(\mathcal{X})$ of the aggregate vector $\mathcal{X} = \sum_{i=1}^{n} X_i$, such as the p-th moment $f(\mathcal{X}) = F_p = \|\mathcal{X}\|_p^p = \sum_{i=1}^{n} |\mathcal{X}_i|^p$. The players are only allowed to communicate with each other by sending messages over the edges of G. The goal is for at least one player to output the approximation to $f(\mathcal{X})$, while minimizing the *max-communication*, which is the total number of bits sent over any edge of G.

We point out that the streaming model is just a special case of one-shot multi-party communication, where the graph G is the line graph, and if (i_t, Δ_t) is the t-th stream update then the t-th player gets input $X_t = e_{i_t} \cdot \Delta_t \in \mathbb{R}^n$. Note that the space complexity of a streaming algorithm is just the max-communication used by a corresponding communication protocol. Since we are interested in insertion only stream, we will consider the distributed analogue known as the *non-negative data* model, where $X_i \in \{0, 1, \ldots, M\}^n$ for all input vectors X_i, for some $M > 0$ (as in streaming, we assume $M = \text{poly}(n, m)$).

Chapter 4 Contributions. We can now describe the main results of Chapter 4. Let $d = \text{diam}(G)$ be the *diameter* of the communication graph G. Our first result is a protocol for F_p estimation when $p \in (1, 2]$ which uses a max communication of $\tilde{O}(\epsilon^{-2} \log d)$ bits.

Theorem 30 *For $p \in (1, 2]$, there is a protocol for $(1 \pm \epsilon)$ approximating F_p in the non-negative data message passing model, which succeeds with probability $3/4$, using a max communication of $O(\frac{1}{\epsilon^2}(\log\log n + \log d + \log 1/\epsilon))$ bits.*

Theorem 30 has interesting implications for any attempts to prove *lower-bounds* for streaming algorithms that estimate F_p. Namely, nearly all lower bounds in streaming come from communication topologies with constant diameter d (e.g., 2-party communication or the complete graph). Theorem 30 completely rules out these approaches for improving lower bounds for F_p estimation in a stream; in particular, any improved lower bound will need to use a graph with $\Omega(n)$ diameter, such as the line.

only apply to the space complexity between updates, and allow arbitrary space to process updates.

On the other hand, for the regime $p \in (0,1)$, we improve the upper bound for F_p estimation, giving an algorithm that is *independent* of d – specifically, the complexity of our algorithm is always $\tilde{O}(\epsilon^{-2})$. Notice that this is independent of the graph topology, and thus holds for the line graph. Consequently, we obtain an algorithm for F_p estimation on insertion-only streams using $\tilde{O}(\epsilon^{-2})$ bits of space in the random oracle model, and $\tilde{O}(\epsilon^{-2} + \log n)$ bits after derandomizing. Both bounds are optimal, closing a long line of research on the problem for this range of p [Woo04, Ind06, KNW10a, KNPW11a, CCM16, BVWY18]. Specifically, we prove

Theorem 33 *There is an insertion only streaming algorithm for F_p estimation, $p \in (0,1)$, which outputs a $(1 \pm \epsilon)$ approximation to $\|x\|_p^p$ with probability at least $2/3$. The algorithm uses $O((\frac{1}{\epsilon^2}(\log\log n + \log 1/\epsilon) + \frac{\log 1/\epsilon}{\log\log 1/\epsilon}\log n)$-bits of space. In the random oracle model, the space is $O(\frac{1}{\epsilon^2}(\log\log n + \log 1/\epsilon))$.*

As an application of our protocol for F_p estimation, we give a $\tilde{O}(\epsilon^{-2})$-bits of space, insertion only *streaming* algorithm for entropy estimation in the random oracle model, and also prove a matching $\Omega(\epsilon^{-2})$ lower bound. This also closes the complexity of the task of entropy estimation in the random oracle streaming model.

1.1.3 Chapter 5: Sampling from Distributed Streams

An important theme of Chapter 4, described above in Section 1.1.2, is the connection between streaming algorithms, where the data is distributed across multiple updates which arrive consecutively in time, and distributed algorithms, where the data is distributed across multiple machines, which must communicate to aggregate sketches of the data. A natural and practically motivated extension is to *combine* the models, and study distributed streams. This is precisely the focus of Chapter 5, based on a joint work with Gokarna Sharma, Srikanta Tirthapura, and David P. Woodruff [JSTW19], where we consider the *continuous, distributed, streaming model [CMYZ12]*, also known as the *distributed functional monitoring model*.

Formally, in this model, there are k physically distributed sites, each of which is connected to a central coordinator, to which they can send and recieve messages. Each site i receives a local stream of data $\mathcal{S}_i$, consisting of a sequence of items of the form (e, w) where e is an item identifier and $w \in \mathbb{R}_{>0}$ is a positive weight. Queries are posed to the coordinator, asking for an aggregate over $\mathcal{S} = \cup_{i=1}^{k} \mathcal{S}_i$, the union of all streams observed so far. The goal is to minimize the number of messages sent over the network, over the entire observation. Oftentimes, the predominant bottleneck for distributed data processing is the network bandwidth, and so it is highly desirable to have algorithms that communicate as few messages as possible. In particular, as the volume

of the data and the number of distributed sites observing the streams scale, it becomes infeasible to communicate all of the data points observed to a central coordinator. Thus, it is necessary to develop algorithms which send significantly fewer messages than the total number of data points received.

As we have already seen, many streaming tasks can be reduced to sampling from the underlying data. For instance, a search engine that uses multiple distributed servers can maintain the set of "typical" queries posed to it through continuously maintaining a random sample of all the queries seen thus far. It follows that having extremely efficient sampling algorithms is an invaluable primitive for more complicated data mining tasks. In Chapter 5, we study the development of such primitives. Specifically, we consider the fundamental problem of maintaining a weighted sample from distributed streams, which generalizes the classic *reservoir sampling* problem [Knu97b, Vit85a] from a centralized to a distributed setting.

Formally, in the distributed weighted random sampling problem, the task is for the coordinator to continuously maintain, at every time step $t \geq s$, a weighted random sample of size s from $\mathcal{S}$. Here, a single weighted random sample is item (e, w) sampled with probability proportional to w. There are two important variations of weighted sampling: **sampling with replacement** (SWR) and **sampling without replacement** (SWOR). In the latter, once an item is sampled, it is removed from the set and cannot be sampled again. Thus, for streams containing a skewed distribution of weights, such as a stream with several *heavy hitters*, the resulting samples can be drastically different.

The problem of unweighted distributed sampling, where all weights are equal, is well studied and admits tight upper and lower bounds on message complexity [TW11, CMYZ12, CTW16]. Moreover, for weighted SWR, there is a simple (and tight) reduction to the unweighted case. Thus, for distributed SWR, the complexity of the problem is understood both for the weighted and unweighted cases. However, in many applications the stream has only a few heavy items which may dominate a random sample when chosen with replacement. For instance, in a large database of internet traffic logs, a small set of domains will dominate the overall traffic, and a weighted SWR would likely consist only of samples from this set. Weighted sampling *without replacement* (weighted SWOR) eludes this issue, since such heavy items can be sampled at most once. The design of efficient, distributed weighted SWOR algorithms is therefore a critical task for many data processing applications.

Main Contributions for Chapter 5. The main contribution of Chapter 5 is the first message-optimal algorithm for weighted SWOR from a distributed stream. In addition, the algorithm has

optimal space and time complexity. Formally

Theorem 41. *There is an algorithm for distributed WSWOR that uses an expected $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ messages between the sites and the coordinator, and continuously maintains at every point in the stream a WSWOR of size s from the items seen so far. Moreover, this communication is optimal up to a constant factor.*

The space required for the coordinator is $O(s)$ machine words, and the space required for each site is $O(1)$ words. Each site requires $O(1)$ processing time per update, and the total runtime of the coordinator is $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$.

The algorithm of Theorem 41 crucially utilizes the *precision sampling framework* from Chapter 3 to maintain the sample. Specifically, the framework assigns a *key* v to each item (e, w) by setting $v = w/t$ where t is exponentially distributed. One then seeks to maintain the items with the top s largest keys. By taking this approach, we ensure that the sites only send a message to the coordinator when they generate a key which is "very likely" to be among the top s.

As an application of our sampling algorithm, we derive the first distributed streaming algorithms for tracking **heavy hitters with residual error**. Here the goal is to output a set $U \subset \mathcal{S}$ with $|U| = O(1/\epsilon)$, such that U contains all items (e, w) with w at least an ϵ fraction of the total weight in the stream *after* the top $1/\epsilon$ heaviest items are removed. We prove upper and lower bounds for this problem which are tight up to a $\log(1/\epsilon)$ factor (Section 5.5).

Finally, we apply our sampling algorithm to improve the message complexity of **distributed L_1 tracking**, also known as count tracking, which is a widely studied problem in distributed streaming (Section 5.6). Here, if $\mathcal{S} = (e_1, w_1), \ldots, (e_n, w_n)$, the L_1 tracking problem is to, after the first τ updates for any $\tau \in [n]$, return a $(1 \pm \epsilon)$ approximation to $W_\tau = \sum_{i=1}^{\tau} w_i$. In addition to an upper bound for this problem, we also derive a tight message lower bound, which closes the message complexity of this fundamental problem.

1.1.4 Chapter 6: Data Streams with Bounded Deletions

As discussed in Section 1.1, the two most prevalent models in the data stream literature are the insertion-only and turnstile models. Unfortunately, since the turnstile model is more general and expressive, many important streaming problems provably require a $\Theta(\log n)$ multiplicative factor more space for turnstile streams than for insertion-only streams. For one example, identifying a coordinate $i \in [n]$ for which $|x_i| > \frac{1}{10}\sum_{j=1}^{n} |x_j|$ can be accomplished using only $O(\log n)$-bits of space in the insertion-only model [BDW16], but requires $\Omega(\log^2 n)$ bits in the turnstile model

[JST11].

This dichotomy leaves a perhaps unsatisfactory resolution to the matter for both theory and practice. Specifically, in practice, one often does not expect an unlimited number of deletions in the stream, thus a natural question is whether we need to pay this additional $\log n$ factor in such cases. In Chapter 6, based on a joint work with David Woodruff [JW18a], we introduce an intermediate model, known as the *bounded-deletion model*, which bridges the complexity of the insertion-only and turnstile models. Since our introduction of this model, a flurry of work has studied the complexity of various streaming tasks therein, and discovered surprising properties and benefits of the model [BGW20, KP20, HKM+20b, ZMW+21].

In the bounded deletion model, we are given a parameter $\alpha \geq 1$, and we are then promised that the L_1 norm $\|x\|_1$ of the stream is at least a $1/\alpha$-fraction of the norm of the stream had all updates been positive. Formally, we define the *insertion vector* $I \in \mathbb{R}^n$ to be the frequency vector of the substream of positive updates ($\Delta_t \geq 0$), and the *deletion vector* $D \in \mathbb{R}^n$ to be the entry-wise absolute value of the frequency vector of the substream of negative updates. In other words, we have:

$$I_j = \sum_{\substack{t \,:\, i_t = j \\ \Delta_t \geq 0}} \Delta_t \qquad \text{and} \qquad D_j = \sum_{\substack{t \,:\, i_t = j \\ \Delta_t < 0}} |\Delta_t|$$

for each $j \in [n]$ on a stream with updates $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$. Notice that $x = I - D$ by definition. We then define the bounded deletion model as follows:

Definition 6.1.1 (α-Bounded Deletion Model). *For $\alpha \geq 1$, a stream satisfies the L_p α-property if*

$$\|I + D\|_p \leq \alpha \|x\|_p$$

Definition 6.1.1 states that, at the *end* of the stream, at least an α fraction of the "L_p mass" contained in the updates of the stream must actually contribute to the norm of x. Observe for $\alpha = 1$, we recover the insertion-only model, whereas for $\alpha = mM$ we recover the turnstile model (excluding streams with $x = \vec{0}$). The choice of p in Definition 6.1.1 depends on the application of interest, e.g. for estimating the L_p norm, one would generally consider the L_p α-property. In Chapter 6, we focus primarily on the important cases of $p \in \{0, 1\}$.

The main contribution of Chapter 6 is to demonstrate that, for many streaming tasks, the extra $\log n$ factor required by turnstile algorithms can be replaced with a $\log \alpha$ factor. This is a significant improvement for moderate values of α. In addition to upper bounds, we give

matching or nearly matching lower bounds in the α-property setting for all the problems we consider. The problems for which we obtain these results include L_1-Heavy Hitters, L_1 & L_0 estimation, L_1 & L_0 sampling, and inner product estimation. For instance, we give upper and lowers bounds of $\tilde{\Theta}(\epsilon^{-1} \log n \log \alpha)$ bits for the L_1 Heavy Hitters problem, whereas the same task requires $\Theta(\epsilon^{-1} \log^2 n)$ space in the turnstile model [JST11, CCFC02a] and $\Theta(\epsilon^{-1} \log n)$ space in the insertion-only model [MG82]. A complete table of these results can be found in Figure 6.1 in Chapter 6.

At a high level, the main insights of Chapter 6 can be summarized as follows. Informally, streams with the α-property have a ratio of at most α between the "noise" in the stream (i.e., insertions that are later removed by a deletion), and the "signal" (insertions that are not deleted). Thus, we can hope to recover information about the "signal" by uniformly sampling at most $\text{poly}(\alpha)$ updates in the stream, and then running a standard turnstile algorithm on the subsampled stream. The advantage of this is that the effective stream length is now only $\text{poly}(\alpha)$ instead of $\text{poly}(n)$, so the counters stored in a turnstile algorithm can be maintained in $O(\log \alpha)$ bits instead of $O(\log n)$, resulting in the desired space savings. This main conceptual contribution can then be summarized by the following statement:

For bounded deletion streams, sampling updates before running turnstile algorithms is an effective method to reduce space while maintaining correctness.

This message demonstrates the utility of sampling as a pre-processing tool to efficiently reduce the space complexity of algorithms before applying other data-analytic procedures. It is important to note that this technique is generally *not* helpful for linear sketching algorithms, which are equivalent to turnstile algorithms under some restrictions [LNW14a], and constitute nearly all turnstile algorithms in the literature. Specifically, linear sketches cannot sample updates in a stream.

1.1.5 Chapter 7: Adversarially Robust Streaming Algorithms

Up until this point, the focus of this thesis has been on the design of sketching algorithms for compressing *static* datasets. Here, by static, we mean that the content of the data is independent of the behavior of the algorithm. Specifically, in the static setting the data is first fixed in advance (although it is not known to the algorithm), and then the randomness for the algorithm is generated. The guarantee of randomized sketching algorithms in the static setting is that they are correct with good probability for any such fixed dataset.

While randomized streaming algorithms are very well-studied, the vast majority of them provide no provable guarantees whatsoever if the "static" assumption is dropped. However, assuming that the stream is independent of the randomness of the sketch, and in particular that future elements of the stream do not depend on previous outputs of the streaming algorithm, may not be realistic [MNS11, GHR+12, GHS+12, HW13, NY15a, BY20]. The failure to handle any amount of adaptivity is a major limitation of many known randomized sketching algorithms.

One potential solution to the above limitation is to use *deterministic algorithms*, since they are guaranteed to be correct on all possible inputs. Unfortunately, many central problems in the streaming literature provably do not admit sublinear-space deterministic algorithms, so for such problems randomization is necessary. Thus, a natural and important problem is whether it is possible to design randomized sketching algorithms that are robust to adaptive streams. We remark that such adaptive streams are very common. For example, in machine learning, an adversary can use maliciously chosen examples to fool a trained model [SZS+14, MHS19]. Moreover, in the field of online learning [Haz16], adversaries are typically adaptive [SS17, LMPL18]. Another example comes from recommendation systems, where an online store suggests recommended items based on a sample of previous purchases, which in turn influences future sales [SS+11, GHR+12].

A streaming algorithm that is correct at all time steps even when the stream is adaptively chosen by an adversary is said to be *adversarially robust*. The focus of Chapter 7 is the design of randomized, adversarially robust streaming algorithms, with space complexity only slightly larger than that of their static counter-parts.

Chapter 7 is based on a joint work with Omri Ben-Eliezer, David Woodruff, and Eylon Yogev [BJWY20], where the adversarial streaming model was first introduced. The paper appeared in PODS 2020, where it received the Best Paper Award. The paper also received a 2021 ACM SIGMOD Research Highlight Award, an invitation to Highlights of Algorithms (HALG) 2021, and was invited to the Journal of the ACM.

The Adversarial Streaming Model. In order to capture the unlimited number of scenarios where future updates to the stream depend in some way on past outputs of the algorithm, we study perhaps the most general non-degenerate model, where the adversary is allowed unbounded computational power and resources. At each point in time, the streaming algorithm publishes its output to a query for the stream. The adversary observes these outputs one-by-one, and can choose the next update to the stream adaptively, depending on the full history of the outputs and stream updates.

Formally, the model is defined via a two-player game between a StreamingAlgorithm and an Adversary. At the beginning, a query $\mathcal{Q}$ is fixed, which the StreamingAlgorithm must reply to after every time step. The game proceeds in rounds, where in the t-th round:

Round t in game between StreamingAlgorithm and Adversary

1. Adversary chooses an update $u_t = (a_t, \Delta_t)$ for the stream, which can depend on all previous stream updates and outputs of StreamingAlgorithm.

2. StreamingAlgorithm processes the new update u_t and outputs its current response R^t to the query $\mathcal{Q}$.

3. Adversary observes R^t (stores it) and proceeds to the next round.

The goal of the Adversary is to make the StreamingAlgorithm output an incorrect response R^t to $\mathcal{Q}$ at some point t in the stream. For example, in the distinct elements problem, the adversary's goal is that on some step t, the estimate R^t will fail to be a $(1+\epsilon)$-approximation of the true current number of distinct elements $\|x\|_0 = |\{i \in [n] : x_i^{(t)} \neq 0\}|$.

Attack on AMS. Given the adversarial model defined above, the first and most natural question is whether we even require new algorithms to handle it. In particular, a priori it is not clear that the well-known static algorithms from the streaming literature are not already adversarially robust. Unfortunately, we demonstrate that this is not the case. In particular, we begin by showing that the classic Alon-Matias-Szegedy sketch (AMS sketch) [AMS96], the first and perhaps most well-known sublinear space L_2 estimation algorithm, is *not* adversarially robust, even in the insertion only model (Theorem 92). We remark that the failure of the AMS-sketch, and in fact all linear-sketches, for robust L_2 estimation was already known for the turnstile model [HW13]. Our proof of this fact involves a relatively simple adversary which, furthermore, can be used to attack several other well-known algorithms from the literature. This failure of standard static streaming algorithms, even under simple attacks, demonstrates that new solutions are required for the adversarial streaming model.

Main Contribution of Chapter 7. The main contribution of Chapter 7 is to design the first adversarially robust randomized streaming algorithms for many fundamental streaming problems, including distinct element (L_0) estimation, L_p-norm estimation, heavy hitters, entropy estimation, and several others. Moreover, the space complexity of all our algorithms is only a small factor larger than that of the best non-robust (static) algorithm for the same problem. Specifically,

for a problem with precision parameter $\epsilon > 0$, the complexity of our robust algorithm is usually at most a $\frac{1}{\epsilon}\log n$ factor larger than that of the best possible static algorithm for the problem, and we often reduce this blow-up to only $\frac{1}{\epsilon}\log\frac{1}{\epsilon}$. A full summary of our results can be found in Table 7.1 in Chapter 7.

Instead of developing ad-hoc solutions separately for each problem, our central contribution is the introduction of a generic **framework for adversarially robust algrorithms**, which can applied to nearly any streaming problem. Namely, we give a highly general framework which allows one to take any static (i.e., non-robust) steaming algorithm, and *transform* it into a robust algorithm with minor space and runtime overhead.

Our robustification methods rely on the fact that functions of interest do not drastically change their value too many times along the stream. Specifically, the transformed algorithms have space dependency on the *flip-number* λ of the function which is being estimated by the algorithm. Informally, for a function $g : \mathbb{R}^n \to \mathbb{R}$, the flip number is a bound on the number of times the function $g(x^{(t)})$ can change by a factor of $(1 \pm \epsilon)$ in the stream. This is formalized below:

Definition 1.1.3 (ϵ-flip number.)**.** *Fix any $m \geq 1$, and consider any stream of length m with intermediate frequency vectors $x^{(1)}, \ldots, x^{(m)}$. Let $g : \mathbb{R}^n \to \mathbb{R}$ be any function. Then the ϵ-flip number of g, denoted $\lambda_{\epsilon,m}(g)$ is the maximum size of any sub-sequence $1 \leq i_1 < i_2 < \cdots < i_k \leq m$ such that $g(x^{(i_j)}) \neq (1 \pm \epsilon)g(x^{(i_{j+1})})$ for all $j = 1, 2, \ldots, k-1$.*

We present two distinct and incomparable methods for transforming a static algorithm into an adversarially robust one. The first method, called *sketch switching*, maintains multiple instances of the non-robust algorithm and switches between them in a way that cannot be exploited by the adversary. The second technique bounds the number of *computation paths* possible in the two-player adversarial game after small modifications to the output, and sets the failure probability small enough to be correct on all of them. We state here an informal version of the first technique, demonstrating the utility of a small flip number.

Lemma 7.2.5 (Sketch Switching, Informal). *Fix any function $g : \mathbb{R}^n \to \mathbb{R}$ and $0 < \epsilon, \delta < 1$. Suppose there exists a static streaming algorithm that uses space $L(\epsilon, \delta)$ to $(1 \pm \epsilon)$-approximate $g(x^{(t)})$ for all time steps t with probability $1 - \delta$, for any particular fixed stream. Then there is an adversarially robust algorithm for $(1 + \epsilon)$-approximating $g(f^{(t)})$ at every step $t \in [m]$ with success probability $1 - \delta$, whose space is $O\left(L(\epsilon, \frac{\delta}{\lambda_{\epsilon,m}(g)}) \cdot \lambda_{\epsilon,m}(g)\right)$.*

It is not hard to see that many standard functions g admit small flip number in the *insertion-only* model. For instance, if $g(x) = \|x\|_p^p$ for any constant $p \geq 0$, then $\lambda_{\epsilon,m}(g) = O(\epsilon^{-1}\log(nm))$

because $\|x\|_p^p$ is monotone, and can only increase by a factor of $(1+\epsilon)$ at most $O(\epsilon^{-1}\log(nm))$ times. This allows us to apply our transformation to static streaming algorithms, in addition to some further techniques which we develop along the way, to obtain adversarially robust algorithms with very small space overhead.

Open Problem and Follow-up Work: Extensions to the Turnstile Model.

Since the introduction of the adversarially robust streaming model and the framework in our paper [BJWY20], several follow up results have been made [HKM+20a, KMNS21, WZ20b]. Notably, Hassidim, Kaplan, Mansour, Matias, and Stemmer [HKM+20a] developed a transformation which improves the dependency on the flip number to $\sqrt{\lambda}$, as opposed to linear in λ, but at the cost of additional $\mathrm{poly}((\log n)/\epsilon)$ factors. So far, this is still the best known bound on the dependency of robust algorithms on λ when ϵ is a constant (some improvements for subconstant ϵ were made in [WZ20b]).

Unfortunately, it is easy to see that most streaming tasks in the *turnstile model* can have unbounded flip number. Thus, despite the above progress, to date, there are no known sublinear space streaming algorithms for any important turnstile streaming problem, such as L_p estimation. Furthermore, in [KMNS21], the authors demonstrated the existence of a specialized streaming task for which any adversarial robust algorithm requires space $\Omega(\sqrt{\lambda})$, whereas $O(\log^2\lambda)$ space is possible in the non-robust model. This demonstrates that, in general, one cannot hope to improve the λ-dependency of transformation type theorems. However, one could still hope for sublinear space algorithms for problems such as L_p estimation in the turnstile model, motivating the following question:

Do sublinear space turnstile streaming algorithms exist for typical streaming problems, such as L_p estimation or heavy hitters?

A negative answer to the above would generate significant new insight into the power of the adversarial model, and represent a strong limitation on the robustness of sketching general data streams. Such a result may also motivate new models or additional restrictions on an adversary to obtain notions of robustness which are both reasonable and feasible.

1.1.6 Chapter 8: Streaming Algorithms for Earth Movers Distance

Chapter 8 of this thesis concerns the development of *geometric streaming algorithms*, where points p lying in some underlying metric space are inserted and deleted, and one seeks to approximate some geometric function of the active dataset. In particular, we focus on the fundemental *Earth Movers Distance* (EMD), which is a classic geometric distance function. Given two multisets $A, B \subset \mathbb{R}^d$, each of size s, the EMD between A and B is the minimum cost of a perfect bipartite matching between the points in A and B, where cost is measured by distance in $\mathbb{R}^d$ – most commonly, the L_1 distance is used. Formally:

$$\mathrm{EMD}(A, B) = \min_{\substack{\text{matching} \\ M \subset A \times B}} \sum_{(a,b) \in M} \|a - b\|_1$$

EMD is an extremely popular distance measure in practice, such as for computer vision [BVDPPH11, SDGP+15], image retrival [RTG00a], biology [NW70], document similarity [KSKW15a], machine learning [ACB17, MJ15, FCCR18], among other areas. However, computing EMD exactly is notoriously computationally inefficient: the best known general algorithms run in $O(s^{2.5})$ time, and any exact algorithm likely require $\Omega(s^2)$ time even for simple metrics [Roh19]. In what follows, we focus on the case of $A, B \subset \{0,1\}^d$, since all results extend naturally by low-distortion metric embeddings from $\mathbb{R}^d$ to $\{0,1\}^{d'}$.

Due to the computational resources required for exact computation, a common approach, especially for modern applications, is to quickly approximate the EMD between two multisets using the method of *randomized tree embeddings* [Bar96, Bar98, CCG+98, FRT04]. In this method, the points $A \cup B$ are embedded into a random tree via recursive space partitions, and then a matching is greedily computed bottom-up in the tree. The resulting data structure is broadly referred to as a *quadtree*, which has a long history of study in geometric algorithms [Sam84]. Quadtree is an extremely efficient algorithm for approximating the EMD; for instance, if $A, B \subset \{0,1\}^d$ are points in the hypercube, then the quadtree runs in *linear* time, and is the best known linear time algorithm for EMD. Moreover, one particularly important property is that, for certain types of quadtree embeddings, one can often *sketch* the output of the quadtree in small space, allowing one to compute the approximation while observing A, B in a stream.

In Chapter 8, based on a join work with Amit Levi, Xi Chen, and Erik Waingarten [CJLW20b], we demonstrate that the approximation factor achieved by the quadtree algorithm for EMD is substantially better than what was previously known. Specifically, we show that the approximation is $\tilde{\Theta}(\log s)$, instead of the prior best known bound $O(\min\{\log s, \log d\} \cdot \log s)$

[AIK08b, BDI+20]. In particular, this demonstrates that the approximation factor of the quadtree is *independent* of the dimension, and immediately improves the state of the art for linear time approximations of high-dimensional EMD.

From the perspective of streaming, note that the previous best approximation obtained by any sublinear space streaming algorithm (in any number of passes over the data), was also $O(\min\{\log s, \log d\} \log s)$, which was obtained by sketching the quadtree cost[AIK08b]. Using our improved analysis, one could hope to immediately obtain streaming algorithms with the improved approximation factor. Unfortunately, our improved analysis does not immediately or obviously result in a streaming algorithm; roughly, this is due to the fact that the edge weights utilized in the quadtree data structure we consider are *data-dependent*, as in prior work [BDI+20]. Nevertheless, by extending the L_p sampling techniques developed in Chapter 3, we design streaming algorithms for EMD with the new $\tilde{\Theta}(\log s)$ approximation. Specifically, we show that, given two passes over the data, a $\tilde{\Theta}(\log s)$ approximation can always be obtained.

Theorem 95. *There is a two-round linear sketch using* $\mathrm{poly}(\log s, \log d)$ *bits of space which, given a pair of size-*s *multi-sets* $A, B \subset \{0,1\}^d$*, outputs* $\hat{\boldsymbol{\eta}} \in \mathbb{R}$ *satisfying*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\eta}} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B)$$

with probability at least $2/3$*.*

The algorithm from the above theorem requires one pass to run L_p sampling algorithms, and a second pass to compute *meta-data* associated with the samples obtained from the first pass. By developing new specialized algorithms for a problem known as *Sampling with Meta-Data*, we show that it is possible to compress these two phases into a single pass. Formally:

Theorem 96. *Given* $\epsilon \in (0,1)$*, there is a (one-round) linear sketch using* $O(1/\epsilon) \cdot \mathrm{poly}(\log s, \log d)$ *bits of space which, given a pair of size-*s *multi-sets* $A, B \subset \{0,1\}^d$*, outputs* $\hat{\boldsymbol{\eta}} \in \mathbb{R}$ *satisfying*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\eta}} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B) + \epsilon s d$$

with probability at least $2/3$*.*

Notice that $\mathrm{EMD}(A, B) \geq s$ when $A \cap B = \emptyset$, in which case Theorem 96 yields an $\tilde{O}(\log s)$ approximation in $\tilde{O}(d)$ space. More generally, if the *Jaccard Index* of A and B is bounded away from 1, we have the following corollary.

Corollary 8.1.1. *Given* $\epsilon \in (0,1)$ *there is a (one-round) linear sketch using* $O(d/\epsilon) \cdot \mathrm{poly}(\log s, \log d)$

space which, given size-s $A, B \subset \{0,1\}^d$ such that $|A \cap B|/|A \cup B| \leq 1 - \epsilon$, outputs a $\tilde{O}(\log s)$ multiplicative approximation to $\mathrm{EMD}(A, B)$ with probability at least $2/3$.

1.2 Numerical Linear Algebra

Sketching as a tool for randomized Numerical Linear Algebra (NLA) has a distinguished history, and has been the central ingredient in a number of breakthrough results in the past few decades [FKV04, W+14, CW17]. The core approach employed by sketching for linear algebra is the following: given a computational task on a large matrix $\mathbf{A}$, one first compresses $\mathbf{A}$ into a lower-dimensional matrix $\mathbf{A}'$. From $\mathbf{A}'$, one solves the original computational task with traditional algorithms, and hopes that the resulting solution is a good approximation. This template is known as the *sketch-and-solve* paradigm, and is the primary focus of Part II of this thesis.

The sketch-and-solve paradigm is best illustrated by an example, and perhaps the most canonical example is that of linear regression. Here, we are given a design matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, and a vector $b \in \mathbb{R}^n$ of *observations*. The goal is to discover an underlying linear relationship between the design matrix $\mathbf{A}$ and the vector of observations b, so that $b \approx \mathbf{A} \cdot x$ for some vector $x \in \mathbb{R}^d$.

We will focus primarily on the *over-constrained* case for linear regression, where $\mathbf{A}$ contains more rows that columns: namely $n \gg d$. The over-constrained setting is extremely common in practice [W+14], as the rows of $\mathbf{A}$ generally represent distinct data-points, each living in an ambient dimension of d, and the size of the data-set often exceeds the ambient dimension. For instance, each row could correspond to a given customer's movie ratings, or the purchasing habits of a user across d different product categories. In both such scenarios, the number of customers (data points) will generally significantly exceed the ambient dimension.

Given a design matrix $\mathbf{A}$ and observation vector b, we are tasked with finding a good linear relationship between the two. In the over-constrained case, because there are more constraints than variables in the linear system, there often will not exist an exact solution x such that $\mathbf{A}x = b$. Instead, one seeks to find a "good enough" solution x. The most common measure of fit is the *least squares* loss, which tasks one with solving the optimization problem:

$$\min_{x \in \mathbb{R}^d} \|\mathbf{A}x - b\|_2^2 \tag{1.1}$$

where $\|y\|_2^2 = \sum_i y_i^2$ is the squared-Euclidean norm. The optimal solution x^* to the least squares loss objective has a clear geometric interpretation: $\mathbf{A}x^*$ is the orthogonal projection of b onto

the column span of $\mathbf{A}$. Due to this geometric interpretation, a closed form solution exists to the optimization problem 1.1, and is given by $x^* = \mathbf{A}^+ b$, where $\mathbf{A}^+$ is the Moore-Penrose psuedo-inverse of $\mathbf{A}$.[10] Unfortunately, computing $\mathbf{A}^+$ is a costly operation, requiring $O(nd^2)$ time in general.

Instead of computing the solution directly, the sketch-and-solve paradigm seeks to first reduce the dimension of the problem, while *approximately* maintaining the solution space. In other words, the goal of the sketch-and-solve method is to produce a vector $\hat{x}$ such that

$$\|\mathbf{A}\hat{x} - b\|_2^2 \leq (1+\epsilon) \min_{x \in \mathbb{R}^d} \|\mathbf{A}x - b\|_2^2$$

for some specified error parameter $\epsilon > 0$. The key steps in the paradigm for obtaining such a solution to linear regression are as follows:

Sketch-and-Solve Template for Least Squares Regression

1. **Initialization:** Sample a random matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$ from some distribution, where $k \ll n$.
2. **Sketching Step:** Computer $\mathbf{SA}$ and $\mathbf{S}b$.
3. **Solving Step:** Output $x' \in \mathbb{R}^d$, where

$$x' = \arg\min_{x \in \mathbb{R}^d} \|\mathbf{SA} - \mathbf{S}b\|_2^2$$

Notice that the last step requires only $O(kd^2)$ time, since the sketched design matrix satisfies $\mathbf{A} \in \mathbb{R}^{k \times d}$. Consequentially, if $k \ll n$, this step is significantly faster than computing the pseduoinverse $\mathbf{A}^+$. Moreover, as we will see multiples times in this thesis (and as is now standard in sketching theory), one can choose a family of random matrices $\mathbf{S}$ such that the product $\mathbf{SA}$ can be computed extremely efficiently: namely in $\text{nnz}(\mathbf{A})$ time, where $\text{nnz}(\mathbf{A}) < nd$ is the number of non-zero entries in $\mathbf{A}$. This is achieved either through usage of very sparse sketching matrices $\mathbf{S}$, or matrices $\mathbf{S}$ with special structure that allows for fast computation of matrix products. Taken together, the total runtime is $O(\text{nnz}(\mathbf{A}) + \text{poly}(kd))$, which is dominated by the first term in the over-constrained case. The primary challenge in this approach is showing that the solution space of the problem is approximately preserved after the sketching step. The key property which the sketch should satisfy for this to occur is known as the *subspace embedding*:

[10] Recall that if $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the singular value decomposition of $\mathbf{A}$, we have $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T$, where Σ^+ is the diagonal matrix with $\mathbf{\Sigma}^+_{i,i} = 1/\mathbf{\Sigma}_{i,i}$ if $\mathbf{\Sigma}_{i,i} > 0$, and $\mathbf{\Sigma}^+_{i,i} = 0$ otherwise.

Definition 1.2.1 (Subspace Embedding)**.** *Fix any matrix* $\mathbf{A} \in \mathbb{R}^{n \times d}$ *and parameter* $\epsilon > 0$*. Then a matrix* $\mathbf{S} \in \mathbb{R}^{k \times d}$ *is said to be an* ϵ*-subspace embedding for* $\mathbf{A}$ *if simultaneously for all* $x \in \mathbb{R}^d$ *we have*

$$\|\mathbf{S}\mathbf{A}x\|_2 = (1 \pm \epsilon)\|\mathbf{A}x\|_2$$

Notice that if $\mathbf{S}$ is a subspace embedding for the augmented matrix $[\mathbf{A}, b]$, then the solution space of the objective 1.1 is pointwise approximately preserved, namely:

$$\|\mathbf{S}(\mathbf{A}x - b)\|_2^2 = (1 \pm \epsilon)\|\mathbf{A}x - b\|_2^2$$

for all $x \in \mathbb{R}^d$. This follows since the vector $(\mathbf{A}x - b)$ can always be written the the form $[\mathbf{A}, b]y$ for a vector $y \in \mathbb{R}^{d+1}$. Subspace embeddings are therefore a crucial ingredient in sketching based approaches to least squares regression, as well as in a variety of other computational tasks, such as low-rank approximation.

The natural question that remains, of course, is whether families of random matrices exist which satisfy the subspace embedding guarantee with good probability. As it happens, many families of matrices will work for this purpose. Such families often do not even depend on the matrix $\mathbf{A}$ (these are known as *oblivious sketches*)! For instance, it turns out that the *count-sketch* matrix ([CCFC02b] Section 2.3.2) developed in the streaming literature, and crucially utilized throughout Part I of this thesis, satisfies the subspace embedding property with good probability, so long as the number of rows satisfies $k = \Omega(d^2/\epsilon^2)$. This fact is one of the many celebrated connections between streaming and numerical linear algebra.

L_p-norm Linear Regression. The least squares loss function in 1.1, although popular, is somewhat arbitrary, and there are many situations where other loss functions would be preferable. In particular, the least squares loss is often overly sensitive to outliers in the data. A common method which is more robust to outliers is *least-absolute deviation* regression, which corresponds to using the L_1 norm instead of the L_2. Namely, the optimization problem is the following:

$$\min_{x \in \mathbb{R}^d} \|\mathbf{A}x - b\|_1 \tag{1.2}$$

The linear, instead of quadratic, dependency on the coordinates of the residual error $\mathbf{A}x - b$ results in a significantly lower sensitivity to outliers in the data than least squares regression. More generally, if one desires a finer calibration of the sensitivity to outliers, one may replace the L_1 norm with any L_p norm for $p \in [1, 2]$.

While there is no closed-form solution to Equation 1.2, the optimizer can be solved for in

polynomial time via linear programming. Unfortunately, this runtime is even worse than the $O(nd^2)$ needed to solve least squares regression. Fortunately, beginning with the work of Sohler and Woodruff [SW11], it is known how the same high-level sketching template can be used to significantly speed up algorithms for least-absolute deviation regression. A selection of the wide-array of sketching techniques utilized to solve L_p norm regression will be introduced in Chapter 10, where we attack the problem for an important class of *structured* matrices $\mathbf{A}$.

Low Rank Approximation. In addition to regression, another fundamental linear algebraic optimization task which has benefitted considerably from sketching techniques is *low-rank approximation*. Here, one is given a large matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ as well as an integer $k \geq 1$. The goal is to find a rank-k matrix $\mathbf{B}$ which is a good approximation to $\mathbf{A}$. The most common "goodness-of-fit" measure is the *Frobenius norm* error, which is equivalent to the Euclidean loss when the matrices are thought of as vectors. Specifically, one solves the optimization problem:

$$\min_{\mathbf{B} \text{ rank-}k} \|\mathbf{A} - \mathbf{B}\|_F^2$$

Where $\|\mathbf{A}\|_F^2 = \sum_{i,j} \mathbf{A}_{i,j}$ is the squared Euclidean norm of $\mathbf{A}$ thought of as a vector in $\mathbb{R}^{nd}$. Low-rank approximation is a classical task in machine learning and statistics, allowing one to uncover latent low dimensional structure inside of a high-dimensional dataset. Additionally, a rank-k approximation $\mathbf{B}$ can be written in the form $\mathbf{B} = \mathbf{UV}$, where $\mathbf{U}, \mathbb{R}^{n \times k}$ and $\mathbf{V} \in \mathbb{R}^{k \times d}$; thus, storing $\mathbf{B}$ requires one to store only $(n + d)k$ entires, rather than the nd entries required to store the original matrix $\mathbf{A}$. Compressing a matrix in this way is particularly useful for distributed and streaming algorithms, as well as for faster matrix multiplication.

Sublinear Algorithms. Sketching is a remarkable tool for speeding up the approximate computation of linear algebraic tasks, which otherwise would require large polynomial runtimes. The sketch-and-solve paradigm involves reading the input matrix, compressing it, and solving the compressed problem. However, there are circumstances under which one may be able to sketch the input *without even reading all of it*, resulting in algorithms with runtime *sublinear* in the size of the matrix. Specifically, the field of sublinear algorithms is concerned with the development of such procedures where, given a large matrix $\mathbf{A}$, one approximately answers some query about the matrix while querying only a small number of entries of $\mathbf{A}$.

A motivating example, related to the task of low-rank approximation, is estimating the rank of a large matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ without reading the entire matrix. Rank estimation is essential for many applications in statistics and machine learning; for instance, before attempting to produce a low-rank approximation of $\mathbf{A}$, one would first like to know whether the matrix $\mathbf{A}$ is in fact

low-rank (or close to low-rank). In general, computing the rank of a matrix requires $O(n^3)$ time for an eigenvalue decomposition.[11] Surprisingly, however, recent results in the field of matrix property testing have demonstrating that one can test whether $\mathbf{A}$ has rank at most d, or requires changing an ϵ-fraction of its entries to have rank at most d, using at most $O(d^2/\epsilon)$ non-adaptive queries to the entries of $\mathbf{A}$ [PR03, KS03, LWW14, BBG18, BLWZ19]. Such a non-adaptive set of queries naturally yields a small space sketch of the matrix. For small values of d, this represents a significant compression in the amount of information required to test rank.

An important and related model is the *linear sketching* model, where one designs a distribution over matrices $\mathbf{S} \in \mathbb{R}^{k \times n^2}$ such that from $\mathbf{S} \cdot \mathtt{vec}(\mathbf{A})$ one can recovery information about the matrix $\mathbf{A}$. Here $\mathtt{vec}(\mathbf{A}) \in \mathbb{R}^{n^2}$ is the vectorization of the matrix $\mathbf{A}$, and the size of the sketch is the number of rows k in the matrix $\mathbf{S}$. Notice that this is precisely the same notion of linear sketching used for streaming algorithms (discussed in Section 1.1). Thus, such linear sketches have many of the same benefits and motivating applications as streaming algorithms, such as message compression for distribution computation, and enabling the usage of algorithms which only require sequential access to the data. The *sketching complexity*, namely the minimum number of entries k required of such a sketch, has been investigated for many fundamental properties in NLA, such as *stable rank*[12] and Schatten p-norms[13] [ACK+16, SWYZ19, BLWZ19].

While many impressive results such as the above have been established over the past decade, the field of sublinear matrix algorithms is still rather nascent, and there are many important estimation tasks for which it is not known whether sublinear algorithms exist. An important and long-standing problem is whether small space oblivious sketches exist for estimating the Schatten 1 norm of a matrix. In Chapter 9 of this thesis, we make strides towards developing the theory of sublinear linear algebra by studying, amoung other tasks, the problem of testing whether a matrix is *positive semi-definite*.

Contributions to Sublinear Numerical Linear Algebra. Since the landmark papers which first established the sketch-and-solve paradigm [Sar06, FKV04, CW17] as a means to speed up general approximation tasks on matrices, an important line of work has been to design sublinear NLA algorithms for matrices which admit nice structural properties. While in the general case one must read every entry of an input matrix for common tasks such as regression and low-rank approximation, in structured cases which arise frequently in practice, this may not be

[11] In theory, this can be reduced to $O(n^\omega)$ time where $\omega < 2.373$ is the exponent of fast matrix multiplication.

[12] Stable rank is a smooth generalization of the rank of a matrix. Specifically, for a matrix $\mathbf{A}$, it is defined as $\frac{\|\mathbf{A}\|_F^2}{\|\mathbf{A}\|_2^2}$, where $\|\mathbf{A}\|_2^2$ is the squared spectral norm of the matrix $\mathbf{A}$.

[13] The Schatten-p norm of a matrix $\mathbf{A}$ is defined as $\|\mathbf{A}\|_{\mathcal{S}_p} = (\sum_i \sigma_i^p)^{1/p}$ is the ℓ_p norm of the vector of singular values of $\mathbf{A}$.

the case. Common examples of such matrices including positive semi-definite (PSD) matrices, Vandermonde Matrices, and various matrix products such as Kronecker products and database joins.

In Part II of this thesis, we design sublinear NLA algortihms for several important classes of such structured matrices. Specifically, in Chapter 9, we design sublinear algorithms for testing whether a matrix is positive semi-definite, an important spectral property which allows for the usage of specialized sublinear algorithms for regression and low-rank approximation [MW17, BCW20]. In Chapters 10 and 11, we design sublinear algorithms for linear regression and low rank approximation on important classes of structured matrices: namely, for *Kronecker product* matrices and *database joins* respectively. These results serve to further establish the field of sublinear NLA, and to develop the sketching toolkit needed for such algorithms. Throughout, there will be substantial technical connections between the sketching and sampling primitives which were developed in Part I, such as *precision sampling*, heavy hitters data structures, and the spectral analysis of random matrices and submatrices.

1.2.1 Chapter 9: Testing Positive Semi-Definiteness via Randomly Sampled Submatrices

We begin Part II with the development of sublinear algorithm for testing whether a real-valued matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD). Recall that a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be PSD if all its eigenvalues are non-negative, and we write $\mathbf{A} \succeq 0$ to denote this property. Certifying whether a matrix is PSD is a central task algorithm design, and continues to be utilized extensively in optimization, spectral graph theory, numerical linear algebra, statistics, and dynamical systems, and many other areas [GW95, VB96, ST04, AHK05, Ste10, DL09, Wai19, DK19].

In general, testing whether a matrix is PSD requires reading the entire matrix, and incurs a runtime of $O(n^3)$ in practice and $O(n^\omega)$ in theory to compute an eigenvalue decomposition. However, if we allow ourselves some flexibility in our notion of testing, it is possible that significantly more efficient algorithms exist. Specifically, in the *property testing model* [GGR98, Gol17], one need only distinguish between $\mathbf{A}$ being, or being far from *any* PSD matrix under an appropriate metric. This flexibility can be seen as a "robust" variant of testing, since we allow $\mathbf{A}$ to lightly violate the PSD constraint without requiring a tester to notice this.

The starting point for our testers is a simple fact: a matrix $\mathbf{A}$ is PSD if and only if all *prin-*

PSD Testing in the Bounded Entry Model

Given $\epsilon \in (0, 1]$ and a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ such that $\|\mathbf{A}\|_\infty \leq 1$, distinguish with probability at least $2/3$ whether $\mathbf{A}$ satisfies:

(1) $\mathbf{A}$ is PSD.

(2) $\mathbf{A}$ is "ϵ-far" from the PSD cone in either the spectral norm, meaning:

$$\min_{\mathbf{B} \succeq 0} \|\mathbf{A} - \mathbf{B}\|_2 \geq \epsilon n$$

or "ϵ-far" from the PSD in squared Euclidean norm distance, meaning:

$$\min_{\mathbf{B} \succeq 0} \|\mathbf{A} - \mathbf{B}\|_F^2 \geq \epsilon n^2 \tag{1.3}$$

If $\mathbf{A}$ satisfies neither **(1)** or **(2)**, the algorithm can return either "PSD" or "not PSD".

Figure 1.2: The PSD Testing Gap Problems.

cipal[14] submatrices of $\mathbf{A}$ are PSD. This equivalence raises the natural question: if $\mathbf{A}$ is far from being PSD, does that mean that *many* submatrices of $\mathbf{A}$ are not PSD? Examining the question lead to a natural algorithm to test definiteness: sample multiple principal submatrices and test if they are PSD; if any are not, then $\mathbf{A}$ is also not PSD. Moreover, this raises the following broader question:

Can we test spectral properties of matrices by sampling small, random submatrices?

In Chapter 9, we answer this question in the affirmative for the property of positive semi-definiteness. In order to avoid degenerate and unlikely instances where an arbitrarily large entry is hidden in $\mathbf{A}$, rendering sublinear testing impossible, we work in the *bounded-entry model*, where the input matrix has bounded entries: $\|\mathbf{A}\|_\infty \leq 1$. Originally proposed by Balcan, Li, Woodruff, and Zhang [BLWZ19] to test spectral properties of matrices, boundedness is often a natural assumption in practice, and has numerous real world applications, such as recommended systems as in the Netflix Challenge [KBV09], or for adjacency matrices of graphs [Gol10, GGR98]. Given this single restriction, we can now formalize the PSD testing problem in Figure 1.2.

We call the first "gap" instance *PSD testing with spectral norm gap*, and the latter *PSD testing with Euclidean norm gap*. Notice that both problems result in a "robust" form of testing, as one

[14]Recall that a principal submatrix $\mathbf{A}_{T \times T}$ for $T \subseteq [n]$ is the restriction of $\mathbf{A}$ to the rows and columns indexed by T.

need only return "not PSD" if there is a non-negligible separation between $\mathbf{A}$ and the nearest PSD matrix. Such robust variants of testing are widely applicable, as oftentimes in practice one need only ensure that the input does not violate the PSD constraint too grossly (see Section 9.1.3 for a further discussion). Also note that, by definition, if $\mathbf{A}$ is ϵ-far from PSD in spectral norm, then it is ϵ^2-far in the squared Euclidean norm. This means that the latter is, in some sense, a more general problem, as the converse is clearly not true.

Contribution: PSD Testing

In Chapter 9, we provide matching upper and lower bounds for the spectral norm gap, and nearly matching bounds for the Euclidean norm gap. These represent the first study of sublinear algorithms for testing positive semi-definiteness, and moreover they demonstrate the utility of random submatrix sampling for determining spectral properties of matrices. Specifically, our two main results are as follows:

Theorem 4 (Theorems 116 and 120)**.** *There is a non-adaptive sampling algorithm which solves the PSD Testing with spectral norm gap problem using at most $\tilde{O}(1/\epsilon^2)$ queries to the entries of $\mathbf{A}$. Moreover, any adaptive or non-adaptive algorithm must query $\widetilde{\Omega}(1/\epsilon^2)$ entries of $\mathbf{A}$.*

Theorem 5 (Theorems 119 and 123)**.** *There is a non-adaptive sampling algorithm which solves the PSD Testing with Euclidean Norm gap problem using at most $\tilde{O}(1/\epsilon^4)$ queries to the entries of $\mathbf{A}$. Moreover, such algorithmm must query $\widetilde{\Omega}(1/\epsilon^2)$ entries of $\mathbf{A}$.*

Since ϵ-gap in spectral norm implies ϵ^2-gap in the squared Euclidean norm, the lower bound of $\widetilde{\Omega}(1/\epsilon^2)$ for the latter implies a separation between the complexity of spectral norm and Euclidean norm testing. Both of the testers have the simple aforementioned structure: they randomly sample principal submatrices and check if they are PSD. The main challenge, therefore, is to analyze the spectrum of random submatrices. We develop an array of new tools and sketching-based arguments for this task.

In addition to the testers, our lower bound for the Euclidean norm gap is highly general, and is based on a new construction of "locally indistinguishable" graphs. Exemplifying the applicability of the construction, we obtain as an immediate corollary new lower bounds for estimating the Schatten-1 norm of $\mathbf{A}$, as well as the *Ky-Fan* norms, both of which are important measures of intrinsic dimensionality of a matrix, and are used extensively throughout machine learning and statistics.

This Chapter is based on a joint work with Ainesh Bakshi and Nadiia Chepurko [BCJ20], and appeared in FOCS 2020.

1.2.2 Chapter 10: Kronecker Product Regression and Low-Rank Approximation

In Chapter 10, we study sublinear algorithms for regression and low-rank approximation for *Kronecker Product* matrices. Kronecker product regression is a special case of ordinary regression in which the design matrix is a product of several smaller matrices. Such matrices naturally arise in applications such as spline regression, signal processing, and multivariate data fitting [VL92, VLP93, GVL13, DSSW18]. Specifically, we consider the over-constrained L_p regression task:

$$\min_{x\in\mathbb{R}^d} \|\mathbf{A}x - b\|_p$$

where the design matrix $\mathbf{A}$ is a Kronecker product of several smaller matrices; namely, we have $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q \in \mathbb{R}^{n\times d}$, and one is given as input the matrices $\mathbf{A}_i \in \mathbb{R}^{n_i\times d_i}$, with $n_i \gg d_i$ for each $i \in [q]$. Note that $n = n_1 \cdots n_q$ and $d = d_1 \cdots d_q$; namely, the design matrix grows exponentially in the number of factors. Consequentially, even computing the Kronecker product is extremely expensive. However, Kronecker products are a quintessential example of the driving theme of this thesis: they are high-dimensional objects which admit significantly lower dimensional representations via their individual factors. Thus, they are a ripe target for the application of sketching techniques.

The study of fast regression on Kronecker products was initiated by Diao, Song, Sun, and Woodruff [DSSW18], who gave an algorithm which runs in time sublinear in the size of the design matrix $\mathbf{A} \in \mathbb{R}^{n_1\cdots n_q\times d_1\cdots d_q}$. Specifically, for $p = 2$ they achieve a running time of $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{nnz}(b))$, which is sublinear in the sparsity of $\mathbf{A}$, but may still require $\Omega(n)$ time due to the depenedency on $\mathrm{nnz}(b)$. For $1 \le p < 2$, their runtime is suffers additional polynomial factors; for instance, when $p = 1$, $q = 2$ and $n_1 = n_2$, their runtime is $O(n_1^{3/2}\mathrm{poly}(d_1d_2) + \mathrm{nnz}(b))$.

In Chapter 10, we provide significantly faster algorithms for Kronecker product regression. For $p = 2$, our running time is $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i))$, which has no dependence on $\mathrm{nnz}(b)$. For $1 \le p < 2$, our running time is $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{nnz}(b))$, which matches the prior best running time for $p = 2$. Recently, by applying new analyses of Lewis Weights as well as a rejection-sampling technique, follow-up work [CD21, PPP21, CM21] has demonstrated that the

$\mathrm{nnz}(b)$ term can also be removed for all $p \geq 1$.

In addition, we study the related *all-pairs regression* problem. Given $\mathbf{A} \in \mathbb{R}^{n \times d}, b \in \mathbb{R}^n$, the goal is to approximately solve the L_p regression problem

$$\min_x \|\bar{\mathbf{A}}x - \bar{b}\|_p$$

where $\bar{\mathbf{A}} \in \mathbb{R}^{n^2 \times d}$ is the matrix formed by taking all pairwise differences of the rows of $\mathbf{A}$ (and $\bar{b}$ is defined similarly). For $p = 1$, this is known as the *rank regression estimator*, which has a long history in statistics [WKL09, WL09, WPB+18, Wan19]. Note that one can rewrite $\bar{\mathbf{A}} \in \mathbb{R}^{n^2 \times d}$ as the difference of Kronecker products $\bar{\mathbf{A}} = \mathbf{A} \otimes \mathbf{1}^n - \mathbf{1}^n \otimes \mathbf{A}$ where $\mathbf{1}^n \in \mathbb{R}^n$ is the all ones vector. However, since $\bar{\mathbf{A}}$ is not a Kronecker product itself, our algorithm for Kronecker product regression is not directly applicable. Therefore, we develop new sketching techniques to obtain an $\widetilde{O}(\mathrm{nnz}(A) + \mathrm{poly}(d/\epsilon))$ time algorithm for $p \in [1, 2]$, which improves substantially on the $O(n^2 d)$ time required to even compute $\bar{\mathbf{A}}$, by a factor of at least n.

Lastly, we initiate the study of Kronecker product low-rank approximation, where the goal is to output a low-rank matrix $\mathbf{B}$ which is a good approximation to a Kronecker product matrix $\mathbf{A} = \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q$. Specifically, the goal is to output a rank-k matrix $\mathbf{B} \in \mathbb{R}^{n \times d}$ such that

$$\|\mathbf{B} - \mathbf{A}\|_F^2 \leq (1 + \epsilon) \min_{\mathbf{B} \text{ rank-}k} \|\mathbf{B} - \mathbf{A}\|_F^2$$

Note that the fastest general purpose algorithms for this problem run in time $\Omega(\mathrm{nnz}(\mathbf{A}))$ [CW17]. Since the sparsity of the product $\mathrm{nnz}(\mathbf{A})$ grows exponentially in the number of tables, such algorithms quickly become inefficient. To remedy this, we develop algorithms which run in input sparsity time, namely with runtime bounded by $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i))$. It is easy to see that this runtime is the best possible up to constants, which resolves the complexity of Kronecker product low-rank approximation.

This Chapter is based on a joint work with Huain Diao, Zhao Song, Wen Sun, and David Woodruff [DJS+19], which appeared in NeurIPS 2019.

1.2.3 Chapter 11: Regression on Database Joins

In Chapter 11, we consider another important class of structured matrices: *database joins*. To introduce joins, we will first need some light notation. Let $\mathcal{C}$ be a finite universe of *attributes*. Recall that a relation $\mathbf{R}$ over a set of attributes $S \subseteq \mathcal{C}$ is just a subset $\mathbf{R} \subseteq \mathbb{R}^S$, so that each

$r \in \mathbf{R}$ is of the form $(a_{i_1}, a_{i_2}, \ldots, a_{i_{|S|}})$. Given two relations $\mathbf{R}_1 \subseteq \mathbb{R}^{S_1}$ and $\mathbf{R}_2 \subseteq \mathbb{R}^{S_2}$ with $S_1 \cap S_2 \neq \emptyset$, we define the join $\mathbf{R}_1 \bowtie \mathbf{R}_2$ as follows:

$$\mathbf{R}_1 \bowtie \mathbf{R}_2 = \{a \in \mathbb{R}^{S_1 \cup S_2} \ : \ a|_{S_1} \in \mathbf{R}_1, \ a|_{S_2} \in \mathbf{R}_2\}$$

where for a set $S \subset \mathcal{C}$ we define $a|_S \in \mathbb{R}^S$ to be the projection of a onto the attributes (coordinates) in S. Notice that if $|\mathbf{R}_1| = n_1$ and $|\mathbf{R}_2| = n_2$, then the size of the join $\mathbf{R}_1 \bowtie \mathbf{R}_2$ can be as large as $n_1 n_2$. Also note that the join itself is a relation over $(\mathbf{R}_1 \bowtie \mathbf{R}_2) \subset \mathbb{R}^{S_1 \cup S_2}$. So more generally, given relations $\mathbf{R}_1, \ldots, \mathbf{R}_k$ with $\mathbf{R}_i \subset \mathbb{R}^{S_i}$, one can define a join of many tables

$$\mathbf{J} = \mathbf{R}_1 \bowtie \mathbf{R}_2 \bowtie \cdots \bowtie \mathbf{R}_k = \{a \in \mathbb{R}^{\cup_i S_i} \ : \ a|_{S_i} \in \mathbf{R}_i, \ \forall i = 1, 2, \ldots, k\}$$

In this case, if $|\mathbf{R}_i| = n_i$ the size of the join can be as large as $n_1 n_2 \cdots n_k$. Consequentially, constructing the join can be extremely expensive compared to the time required to read the individual relations $\mathbf{R}_i$. On the other hand, in keeping with a common theme of this thesis, notice that $\mathbf{J}$ admits a lower-dimensional representation as a "product" of many smaller components. Thus, one may hope to perform computational tasks on $\mathbf{J}$ in time significantly sublinear in its size.

In Chapter 11, we do precisely this, by developing sublinear algorithms for linear regression when the design matrix is a database join. Specifically, we are given as input q relations $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_q$, where $\mathbf{R}_i \in \mathbb{R}^{n_i \times d_i}$ can be thought of as a matrix with d_i columns (i.e., attributes) and n_i rows (i.e., size of the relation). One can then define the join $\mathbf{J} = \mathbf{R}_1 \bowtie \mathbf{R}_2 \bowtie \cdots \bowtie \mathbf{R}_q \in \mathbb{R}^{N \times d}$ over some specified set of attributes,[15] and ask to solve the optimization problem:

$$\min_{x \in \mathbb{R}^d} \|\mathbf{J}x - b\|_2^2$$

where $b \in \mathbb{R}^n$ is a vector specified as input.

Previously, the fastest known algorithms for this task were based on so-called *Functional Aggregation Queries* (FAQ) methods [AKNR16, AKNN+18]. For instance, given a join on two relations $\mathbf{R}_1 \bowtie \mathbf{R}_2$, the runtime of this method is $\tilde{O}(nd^2)$, where $n = \max(n_1, n_2), d = \max(d_1, d_2)$. While sublinear in the size of $\mathbf{J}$, this runtime is still polynomially larger than the input to the problem. This polynomial blow-up is exacerbated by the presence of categorical features. Namely, it is a common practice to convert categorical data to their so-called *one-hot encoding* before optimizing any statistical model. Such an encoding creates one column for each possible value

[15] Notice that the set of columns being joined on, namely the structure of the join query, is not specified in the above notation, and must therefore be specified contextually.

of the categorical feature, blowing up the number of attributes d by the number of categories. Thus, especially in the presence of categorical features, one would ideally have *input sparsity* algorithms, with runtime bounded by the number of non-zero entries in the input relations.

We demonstrate that this idealistic goal is in fact possible for the important case of two-table joins. We note that the two-table case is very well-studied [AMS96, AGMS99, GWWZ15], and moreover one can always reduce to the two-table case by first precomputing part of a general join. Specifically, we give *input-sparsity* algorithms which produce a value $\hat{x} \in \mathbb{R}^d$ satisfying

$$\|\mathbf{J}\hat{x} - b\|_2 \leq (1+\epsilon) \min_{x \in \mathbb{R}^d} \|\mathbf{J}x - b\|_2$$

in runtime which is the minimum of $\tilde{O}(((n_1+n_2)d+d^3)\log\log\epsilon^{-1})$ and $\tilde{O}((\text{nnz}(\mathbf{T}_1)+\text{nnz}(\mathbf{T}_2)+d^5)\log\epsilon^{-1})$. Notice that the second runtime is preferable for sparse matrices, whereas the former is preferable for dense matrices. When n_1, n_2 are polynomially larger than d, this runtime is linear in the sparsity of the input. This demonstrates a substantial improvement over prior FAQ based methods, which perform poorly as the number of attributes grows.

In addition, we consider regression for arbitrary joins on more than two tables, and introduce a general framework to apply sketching methods to obtain faster regression algorithms than those obtained by prior FAQ-based methods. We empirically evaluate our algorithms on various real-world databases, and compare them to the previous best FAQ-based algorithm. Our experiments demonstrate significant speedups over the latter, with only a small sacrifice in accuracy. For example, for the join of two tables in the MovieLens data set, which has 23 features, we obtain a 10-fold speedup over the FAQ-based algorithm, while suffering only 0.66% relative error.

Chapter 11 is based on a paper with Alireza Samadian, David P. Woodruff, and Peng Ye [JSWY21] in ICML 2021.

1.3 Query Evaluation and Automata Theory

The quintessential problem in database systems and theory is that of *query evaluation*. A relational database $\mathcal{D}$ over a universe $\mathcal{U}$ is given by a set of relations $\mathcal{D} = \{\mathbf{R}_1, \ldots, \mathbf{R}_m\}$, where each $\mathbf{R}_i \subset \mathcal{U}^{k_i}$ is a k_i-ary relation taking values in $\mathcal{U}$. A query Q is a logical expression which takes as input a relational database $\mathcal{D}$, and outputs a new relation $Q(\mathcal{D})$ of "answers" to the query Q over the database $\mathcal{D}$. Query evaluation is the task of computing $Q(\mathcal{D})$, given as input the database $\mathcal{D}$ and the query Q.

The most common class of queries used in database systems, and the best studied in the literature, are the *conjunctive queries* (CQ). Conjunctive queries are formulas which express a significant fragment of first-order logic. They are equivalent to *select-project-join* queries in relational algebra, *select-from-where* queries in SQL, and are closely related to *constraint satisfaction problems* (CSPs). Moreover, they are ubiquitous in practice, and their evaluation is crucial for modern data analysis [CMN99a]. Therefore, the majority of the literature on query evaluation is focused on CQs and subsets thereof [CM77a, GMUW09, Abi95, PS13a].

In addition to query evaluation, an important problem is to approximately count the number of answers to a query (i.e., approximate $|Q(D)|$). The counting problem for CQ's is of fundamental importance for query optimization [RGG03, PS13b]. Specifically, relational query engines decide the order in which to evaluate a complicated query by utilizing estimates of the sizes of intermediate solutions to the query. To take an example from Chapter 11 (see Section 1.2.3), a complicated database join $\mathbf{J} = \mathbf{R}_1 \bowtie \mathbf{R}_2 \bowtie \cdots \bowtie \mathbf{R}_m$, which is a type of conjunctive query, can in general be equivalently evaluated in up to $m!$ distinct orders. Different relations $\mathbf{R}_i, \mathbf{R}_j, \mathbf{R}_k$ may share different subsets of attributes, so if $\mathbf{R}_i \bowtie \mathbf{R}_j$ is significantly larger than $\mathbf{R}_i \bowtie \mathbf{R}_k$, it would generally be faster to first evaluate $\mathbf{J}_1 = \mathbf{R}_i \bowtie \mathbf{R}_k$ and then $\mathbf{J}_1 \bowtie \mathbf{R}_j$, rather than to evaluate $\mathbf{J}_2 = \mathbf{R}_i \bowtie \mathbf{R}_j$ and then $\mathbf{J}_2 \bowtie \mathbf{R}_k$. Thus, having estimates of the sizes of the queries $|\mathbf{J}_1|, |\mathbf{J}_2|$ is invaluable to a query optimizer.

Due to its significance in database applications, the problem of counting the number of distinct answers to a query, also known as the *count-distinct problem*, has been studied considerably from the perspective of *streaming* [FM85, BYJK+02, IW03, Woo09, KNW10b]. In this setting, the entries of the database query are seen (or computed[16]) sequentially, possibly with duplicates, and one must estimate the number of unique entries in small space. This is precisely the problem of L_0 estimation studied throughout Part I of this thesis; for instance, we design improved and robust L_0 estimation algorithms in Chapters 6 and 7. This connection between streaming and databases has been particularly fruitful, and as we will see the technical approaches employed in L_0 sketches will be useful for approximating counting for CQ's

In many ways dual to approximate counting, and equally as important, is the task of uniformly sampling answers from $Q(D)$. Sampling is used to efficiently generate representative subsets of the data, instead of computing the entire query, which are often sufficient for data mining and statistical tasks [AD20]. Due to the high computational complexity of query evaluation, fast

[16] *Constant delay enumeration* algorithms, which compute and output the answers to a query sequentially, each in constant or near-constant time, are a topic of intensive study in the database literature [Seg13, Seg14, Seg15]. The existence of constant delay enumeration algorithms for a variety of queries further strengthens the connection between databases and streaming.

algorithms for uniform sampling are central to the analysis of modern databases. Starting with the work of Chaudhuri, Motwani and Narasayya [CMN99b], the study of random sampling from queries has attracted significant attention from the database community [ZCL+18, CY20].

Unfortunately, even the most basic tasks related to conjunctive queries, including approximate counting and uniform sampling, are computationally intractable. For instance, given a 3-satisfiability instance $\phi(x_1, \ldots, x_n)$, one can easily construct a polynomially sized CQ Q and database $\mathcal{D}$ such that $Q(\mathcal{D})$ coincides with the set of satisfying assignments to ϕ. Consequentially, deciding whether $Q(\mathcal{D}) = \emptyset$ (known as the *decision problem*), and thus multiplicatively approximating $|Q(\mathcal{D})|$, is NP-Hard. Moreover, exactly computing $|Q(\mathcal{D})|$ is #P-Hard. Thus, a major focus of investigation in the area has been to find tractable special cases of CQs [Yan81, CR97, GLS98, GSS01, GLS02, FG06, GGLS16].

Part III of this thesis is concerned with a sequence of two papers [ACJR19, ACJR21] which characterize the classes of conjunctive queries for which approximate counting and uniform sampling can be accomplished in polynomial time. In particular, the main result is a fully polynomial time randomized approximation scheme (FPRAS) and polynomial time sampler for the class of conjunctive queries with so-called bounded *hypertree-width*. By previous impossibility results [GSS01], this class is roughly the largest for which an FPRAS can exist.

The main ingredient in this result is the resolution of a fundamental counting problem from automata theory via sketching techniques. Specifically, the answers to a class of CQs can be modeled by the accepting inputs to families of finite automata. In [ACJR19], we demonstrate the first FPRAS for counting the number of words of length n accepted by a *non-deterministic finite automton* (NFA). In [ACJR21], we extend this algorithm to the class of *tree automaton*, which significantly generalize NFAs, and can express the answers to CQs with bounded hypertree-width. Previously, the best known algorithm for either task was a quasi-polynomial time randomized approximation scheme (QPRAS) of Gore, Jerrum, Kannan, Sweedyk, and Mahaney [GJK+97].

1.3.1 Chapter 12: Approximate Counting and Uniform Sampling from Conjunctive Queries

In order to describe the main result of Chapter 12, it will help to formally define the conjunctive query. Let $\mathbf{C}$ and $\mathbf{V}$ be two disjoint sets of *constants* and *variables*, respectively. Then a

conjunctive query (CQ) is an expression of the form:

$$Q(\bar{x}) \quad \leftarrow \quad R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n), \tag{1.4}$$

where for every $i \in [n]$, R_i is a k_i-ary relation symbol ($k_i \geq 1$), $\bar{u}_i \in (\mathbf{C} \cup \mathbf{V})^{k_i}$ is a k_i-ary tuple of variables and constants, and $\bar{x} = (x_1, \ldots, x_m)$ is a tuple of variables such that each variable x_i in $\bar{x}$ occurs in at least one $\bar{u}_i$. A database D is an instantiation $D = \{R_1^D, R_2^D, \ldots, R_n^D\}$ of the relations, so that each $R_i^D \subset \mathbf{C}^{k_i}$ is a finite subset of k_i-ary tuples $\bar{a}$ of constants. A homomorphism from Q to D is a function $h : \mathbf{V} \to \mathbf{C}$ such that for every $i \in [n]$, we have $h(\bar{u}_i) \in R_i^D$, where $h(\bar{u}_i)$ is applied coordinate-wise and acts as the identity on $\mathbf{C}$. Given such a homomorphism h, the tuple $h(\bar{x})$ is called an *answer* to Q over D, and the set of all such answers is denoted $Q(D)$.

As mentioned earlier, evaluating conjunctive queries $Q(D)$ is an intractible task in general. Beginning work in [Yan81], a fruitful line of research for finding tractable cases has been to study the *degree of acyclicity* of a CQ. Specifically, given a CQ Q, one can define a graph $G(Q) = (V, E)$, known as the *constraint-graph*, by setting V to be the set of variables in Q, and connecting by an edge any two variables that appear in the same relation (constraint) R_i. More generally, one can define the constraint *hypergraph* $\mathcal{H}(Q)$, with the same set of vertices, and each hyperedge corresponding to the subset of variables that appears in a relation R_i. Given these two definitions, one can define the *treewidth* $\mathrm{tw}(Q)$ of Q [CR97, GSS01], and more generally the *hypertree width* $\mathrm{hw}(Q)$ of Q [GLS02], by the treewidth (resp. hypertree width) of $G(Q)$ (resp. $\mathcal{H}(Q)$).

It is known that many computational tasks on databases are easier for queries with bounded (i.e., constant) tree or hypertree width. For instance, deciding if $Q(D) = \emptyset$ can be accomplished in polynomial time for every such class $\mathcal{C}$ of CQs [CR97, GSS01, GLS02]. Unfortunately, uniform generation and exact counting are more challenging tasks. Specifically, given as input a conjunctive query Q and database D, computing $|Q(D)|$ is #P-complete even when $\mathrm{tw}(Q) = 1$ [PS13b]. Moreover, recall that even multiplicative approximations are intractable in general. On the other hand, these facts do not preclude the existence of polynomial time approximation algorithms for classes of CQs with bounded treewidth, as the associated query decision problem is in P. Despite this possibility, prior to our work, no such approximation algorithms were known.

The main result of Chapter 12 is precisely such an algorithm. Specifically, we demonstrate the existence of a fully polynomial-time randomized approximation scheme (FPRAS) and a fully

polynomial-time almost uniform sampler (FPAUS) for every class of CQs with bounded hypertree width. Here, an FPAUS is an sampler which returns each element $e \in Q(D)$ with probability $(1 \pm \epsilon)|Q(D)|^{-1}$, and has $\mathrm{poly}(\log \epsilon^{-1})$ dependency on ϵ. Since $\mathrm{hw}(Q) \leq \mathrm{tw}(Q)$ for every CQ Q [GLS02], our result also includes every class of CQs with bounded treewidth. Formally, we show the following.

Theorem 140 *Let $\mathcal{C}$ be a class of CQs with bounded hypertree width. Then there exists a fully polynomial-time randomized approximation scheme (FPRAS) that, given $Q \in \mathcal{C}$ and a database D, estimates $|Q(D)|$ to multiplicative error $(1 \pm \epsilon)$. Moreover, there is a fully polynomial-time almost uniform sampler (FPAUS) that generates samples from $Q(D)$.*

Our algorithm of Theorem 140 in fact holds for a larger class of queries, including *unions* of conjunctive queries with bounded hypertree width, which are a well-studied generalization of CQs.

Still, one may ask whether our algorithm is suboptimal, and if there exists an even larger class of queries $\mathcal{C}$ that admit an FPRAS. As it turns out, this cannot be the case in general, and the algorithm of Theorem 140 is essentially optimal. Specifically, a seminal result of Grohe, Schwentick, and Segoufin [GSS01] demonstrates that given a class $\mathcal{G}$ of graphs, the evaluation of all conjunctive queries Q whose constraint graph is in $\mathcal{G}$ is tractable if, and only if, all graphs in $\mathcal{G}$ have bounded treewidth. Since multiplicative approximations solve the decision problem, we obtain the following corollary, which characterizes the class of all conjunctive queries for which approximate counting and uniform sampling is tractable.

Corollary 12.1.1. *Let $\mathcal{G}$ be a class of graphs and $\mathcal{C}$ be the class of all CQs whose constraint graph is in $\mathcal{G}$. Then assuming $\mathrm{W}[1] \neq \mathrm{FPT}$ and $\mathrm{BPP} = \mathrm{P}$,[17] the following are equivalent:*

1. *The problems of approximately computing $|Q(D)|$ and of almost uniform sampling from $Q(D)$, given as input $Q \in \mathcal{C}$ and a database D, admit an FPRAS and an FPAUS, respectively.*
2. *$\mathcal{G}$ has bounded treewidth.*

In other words, Corollary 12.1.1 demonstrates that, for classes of conjunctive queries of the form $\mathcal{C} = \{Q \mid \textsc{ConstraintGraph}(Q) \in \mathcal{G}\}$, the existence of an FPRAS for $\mathcal{C}$ is equivalent

[17] Recall that BPP is the class of problems solvable in polynomial time by a randomized algorithm with error at most $1/3$; a common conjecture in complexity theory is that $\mathrm{BPP} = \mathrm{P}$. Also recall that FPT is the class of *fixed parameter tractable* (FPT) problems, which, given a parameter k and input size n, can be solved in time $f(k) \cdot \mathrm{poly}(n)$ for some computable function f. Lastly, $\mathrm{W}[1]$ is the class of parameterized problems reducible in time $f(k) \cdot \mathrm{poly}(n)$ to the problem of deciding if a graph has a k-clique. A common conjecture in paramaterized complexity is that k-clique does not admit a FPT algorithm, and therefore that $\mathrm{W}[1] \neq \mathrm{FPT}$.

to $\mathcal{G}$ having bounded treewidth. This extends the seminal hardness result of [GSS01] from query evaluation to the counting problem for CQ's.

Approximate Counting in Automata Theory

As previously mentioned, the key ingredient in our result is the development of the first FPRAS for counting accepting inputs to fundamental classes of automata. By accomplishing this for a family of automata which can express precisely the class of answers $Q(D)$ to a conjunctive query with bounded hypertree-width, the result of Theorem 140 follows immediately.

The first step towards this end is the main result of our paper [ACJR19], which considered the family of non-deterministic finite automton (NFA). Specifically, in [ACJR19], we developed an algorithmic template, based on sketching techniques, which resulted in the first FPRAS for counting words accepted by an NFA. This improved on the prior best known quasi-polynomial time algorithm of [GJK+97]. We focus on approximate counting in the following theorem statements, noting that (almost) uniform sampling follows as a corollary. Formally:

Theorem 6 (Main result, [ACJR19])**.** *Given an NFA $\mathcal{N}$ and $n \geq 1$, there is an algorithm which runs in time $\mathrm{poly}(|\mathcal{N}|, n, \epsilon^{-1})$ and with probability $2/3$, outputs an estimate $\widetilde{N}$ with:*

$$(1-\epsilon)|\mathcal{L}_n(\mathcal{N})| \leq \widetilde{N} \leq (1+\epsilon)|\mathcal{L}_n(\mathcal{N})|$$

Where $\mathcal{L}_n(\mathcal{N})$ is the set of words of length n accepted by $\mathcal{N}$.

We remark that for the case of a *deterministic finite automata* (DFA) $\mathcal{M}$, which is an NFA where each $x \in \mathcal{L}_n(\mathcal{M})$ admits exactly one accepting computation in the automata, *exact* counting and uniform sampling are both possible in polynomial time via a simple dynamic program. However, due to their non-determinism, the case of NFA's is significantly more challenging – in particular, the task of computing $|\mathcal{L}_n(\mathcal{N})|$ exactly is #P-Hard, so polynomial time exact algorithms are unlikely (and would imply, for instance, that P = NP).

In contrast to the QPRAS of [GJK+97], the algorithmic template developed in [ACJR19] is based on sketching. Specifically, given an NFA $\mathcal{N}$, one can write down a natural dynamic program (DP) to compute the set $\mathcal{L}_n(\mathcal{N})$. The intermediate states of this DP consist of sets of substrings which can be composed together via unions and concatenations to ultimately form $\mathcal{L}_n(\mathcal{N})$. However, since $|\mathcal{L}_n(\mathcal{N})|$ can be exponential in n, the intermediate states are also exponentially large. Our approach in [ACJR19] is to compress these states, by replacing each set

with a sketch consisting of i.i.d. uniform samples from the set. These sketches can then be used to estimate the size of unions between non-disjoint subsets corresponding to states of the DP. The main challenge is generating the samples required to form these sketches, and doing so bottom-up through the DP. To the best of our knowledge, this is the first example of a core automata-theoretical problem being solved via sketching techniques.

Building on this framework, in our subsequent work [ACJR21] we extended the FPRAS from NFAs to the significantly more general class of *tree automata* . Roughly speaking, a tree automata $\mathcal{T}$ accepts a language of *trees* as inputs; they can be seen as running many NFAs simultaneously along separate branches of a tree, allowing for "splitting" of branches. Moreover, given a CQ Q and a database D with bounded hypertree-width, one can construct a polynomial sized tree automata $\mathcal{T}$ with $\mathcal{L}_n(\mathcal{T}) = Q(D)$ (up to a natural bijection), where $\mathcal{L}_n(\mathcal{T})$ is the set of trees of size n accepted by $\mathcal{T}$. Formally, we show:

Theorem 141 (Main result, [ACJR21]). *Given a tree automaton $\mathcal{T}$ and $n \geq 1$, there is an algorithm which runs in time $\mathrm{poly}(|\mathcal{T}|, n, \epsilon^{-1})$ and with probability $2/3$, outputs an estimate $\widetilde{N}$ with:*

$$(1-\epsilon)|\mathcal{L}_n(\mathcal{T})| \leq \widetilde{N} \leq (1+\epsilon)|\mathcal{L}_n(\mathcal{T})|$$

Theorem 141 is proven across the two sections 12.5 and 12.6. While built on the same sketching-based template as the NFA FPRAS, the algorithm of Theorem 141 is significantly more involved. In particular, the procedure for generating the samples used in the sketches of the states of the DP is considerably more challenging, and is the main focus of Section 12.6.

1.4 Roadmap of the Thesis

This thesis is divided into three parts, each focusing on one of the distinct application areas of sketching discussed in this section. While the goals and motivations of the problems studied in separate parts varies, the techniques utilized across these parts share substantial similarities. Chapter 2 presents important mathematical background and notation which will be useful throughout the thesis, with a special emphasis placed on the techniques which will span multiple chapters. While we will refer to results from Chapter 2 throughout the thesis, each individual chapter is designed to be self-contained; therefore, results which are required for a chapter will be restated in that chapter.

The results presented in this thesis are drawn from a variety of papers, cataloged below.

Part I: Streaming and Distributed Algorithms

- Chapter 3: **Perfect L_p Sampling in Data Streams**, is based on the following two works:

  I [JW18b] *Perfect L_p Sampling in a Data Stream*, with David P. Woodruff. FOCS 2018.

  II [JWZ21] *Truly Perfect Samplers for Data Streams and Sliding Windows*, with David P. Woodruff and Samson Zhou.

- Chapter 4: **Moment Estimation in Streaming and Distributed Models**, is based on:

  [JW19] *Towards Optimal Moment Estimation in Streaming and Distributed Models* with David P. Woodruff. APPROX 2019.

- Chapter 5: **Sampling from Distributed Streams**, is based on:

  [JSTW19] *Weighted Reservoir Sampling from Distributed Streams* with Gokarna Sharma, Srikanta Tirthapura, and David P. Woodruff. PODS 2019.

- Chapter 6: **Data Streams with Bounded Deletions**, is based on:

  [JW18a] *Data Streams with Bounded Deletions*, with David P. Woodruff. PODS 2018.

- Chapter 7: **Adversarially Robust Streaming**, is based on:

  [BJWY20] *A Framework for Adversarially Robust Streaming Algorithms*, with Omri Ben-Eliezer, David Woodruff, and Eylon Yogev. PODS 2020

- Chapter 8: **Streaming Algorithms for Earth Movers Distance**, is based on:

  [CJLW20a] *An Improved Analysis of the Quadtree for High Dimensional EMD*, with Xi Chen, Amit Levi, and Erik Waingarten.

Part II: Numerical Linear Algebra

- Chapter 9: **Testing Positive Semi-Definiteness**, is based on:

  [BCJ20] *Testing Positive Semi-Definiteness via Random Submatrices*, with Ainesh Bakshi and Nadiia Chepurko. FOCS 2020.

- Chapter 10: **Kronecker Product Regression and Low-Rank Approximation**, is based on:

  [DJS+19] *Optimal Sketching for Kronecker Product Regression and Low Rank Approximation*, with Huain Diao, Zhao Song, Wen Sun, and David Woodruff. NeurIPS

2019.

- Chapter 11: **In-Database Regression**, is based on:

  [JSWY21] *In-Database Regression in Input Sparsity Time*, with Alireza Samadian, David P. Woodruff, and Peng Ye. ICML 2021.

Part III: Database Query Evaluation

- Chapter 12: **Approximate Counting and Uniform Sampling from Database Queries**, is based on the two papers:

  I [ACJR19] *Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation*, with Marcelo Arenas, Luis Alberto Croquevielle, and Cristian Riveros. PODS 2019.

  II [ACJR21] *When is Approximate Counting for Conjunctive Queries Tractable?* with Marcelo Arenas, Luis Alberto Croquevielle, and Cristian Riveros. STOC 2021.

Chapter 2

Background and Preliminaries

In this chapter, we introduce the fundamental mathematical tools and techniques which will be used throughout the thesis. While the basic notation and results presented here will be assumed as background in the remainder of the thesis, all of the more involved results will be restarted and described where needed, so that the individual chapters are as self-contained as possible. Beyond establishing basic sketching tools, a important role served by this chapter is presentation of several tools, such as the *count-sketch* algorithm, which exist in the literature in various equivalent and non-equivalent formulations, in a unified fashion. This will allow us to speak of these concepts in a well-defined and cohesive manner in the chapters that follow.

2.1 Notation and Basic Preliminaries

We begin with several basic definitions, as well as an introduction to the notational conventions used throughout the thesis.

General Notation. For $a, b, \epsilon \in \mathbb{R}$, we write $a = b \pm \epsilon$ to denote the containment $a \in [b - \epsilon, b + \epsilon]$. For positive integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For any function $f : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$, we use the notation $\tilde{O}(f(x))$ and $\tilde{\Omega}(f(x))$ to hide $\mathrm{poly}(\log(f(x))$ factors.

Throughout the thesis, given a function $f : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$, we use the notation $\tilde{O}(f(x))$ and

$\tilde{\Omega}(f(x))$ to hide $\text{poly}(\log(f(x))$ factors. In other words, for any constant $c \geq 1$ we have $f(x) \cdot \log^c(f(x)) = \tilde{O}(f(x))$ and $f(x) \cdot \log^{-c}(f(x)) = \tilde{\Omega}(f(x))$. We use the term *with high probability* (w.h.p.) to describe events that occur with probability $1 - n^{-c}$, where n is the input size and c is a constant independent of n.

All space bounds in the thesis will be stated in the unit of *bits* (as opposed to *words*). For our run-time complexity, we assume the unit cost RAM model, where a word of $O(\log n)$-bits can be operated on in constant time, where n is the size of the input to the problem. For instance, in the streaming model where updates to an implicit vector x are being made, n can be taken as the dimension of x.

Vectors and Vector L_p Norms. For a vector $x \in \mathbb{R}^n$ and any integer $k \in [n]$, we define $x_{\text{tail}(k)}$ to be x but with the top k coordinates (in absolute value) set equal to 0. We also write $|x|$ to denote the entry-wise absolute value of x, so $|x|_j = |x_j|$ for all $j \in [n]$. Given a subset $S \subset [n]$, we will also write $x_S \in \mathbb{R}^n$ to denote the vector obtained after setting equal to zero all coordinates x_i with $i \notin S$.

For any real $p > 0$, the L_p *norm*[1] of a vector x is defined as $L_p = \|x\|_p = (\sum_i |x_i|^p)^{1/p}$, and the *$p$-th moment*, denoted F_p, is defined as $F_p = \|x\|_p^p = \sum_i |x_i|^p$, so that $F_p = L_p^p$. For the case of $p = 0$, we define L_0 norm of a vector $x \in \mathbb{R}^n$ as $L_0 = \|x\|_0 = |\{i \in [n] \mid x_i \neq 0\}|$. Note that the L_0 norm (which is in fact a norm) is the same as the and F_0 moment and the Hamming norm of a vector, and is given by the support size of the vector in question.

We remark that the difference between estimating the quantity L_p and F_p up to a factor of $(1 \pm \epsilon)$ is often negligible so long as p is bounded by a constant. The justification for the separate notations is due primarily to historical reasons. Generally speaking, in the streaming literature the notation F_p is used in the insertion-only model, whereas one often seeks to estimate L_p in the turnstile model (of course, up to a constant in the space complexity, one could have as easily done the opposite). In the insertion-only literature, F_p is sometimes referred to as the *p-th frequency moment*. In this thesis, we will use both notations, generally keeping with this convention where

[1]Note that this is only a norm for $p \geq 1$, although we will sometimes abuse terminology and call L_p a norm for $p < 1$.

possible.

2.1.1 Linear Algebra Preliminaries.

The usage of concepts from linear algebra plays a critical role in this thesis, even beyond Part II. Therefore, it will be important to unify our treatment of the subject here.

Matrices and Submatrices. Throughout the thesis, matrices will always be displayed using boldface capital notation, e.g., $\mathbf{A}, \mathbf{B}, \boldsymbol{\mathcal{X}}, \boldsymbol{\Sigma}$. Given a positive integer $n \geq 1$ we write $\mathbb{I} \in \mathbb{R}^{n \times n}$ to denote the $n \times n$ identity matrix. Additionally, we will write $\mathrm{nnz}(\mathbf{A})$ to denote the number of non-zero entries of a matrix $\mathbf{A}$.

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, and given subsets $S \subset [n], T \subseteq [d]$, we denote the matrix $\mathbf{A}_{S \times T} \in \mathbb{R}^{|S| \times |T|}$ as the matrix $\mathbf{A}$ restricted to the submatrix of the rows in S and the columns in T. If $n = d$ and $S = T$, then the square submatrix $\mathbf{A}_{S \times T} = \mathbf{A}_{S \times S}$ is called a *principal submatrix* of $\mathbf{A}$. We will use the notation $\mathbf{A}_{i,*}$ to denote the i-th row of $\mathbf{A}$, and $\mathbf{A}_{*,i}$ to denote the i-th column of $\mathbf{A}$.

Singular Values and Eigenvalues. Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ be any real-valued matrix. We can then denote the $\min\{n, d\}$ singular values of $\mathbf{A}$ via

$$\sigma_{\max}(\mathbf{A}) = \sigma_1(\mathbf{A}) \geq \sigma_2(\mathbf{A}) \geq \cdots \geq \sigma_{\min\{n,d\}}(\mathbf{A}) = \sigma_{\min}(\mathbf{A})$$

and write $\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}$ to denote the *condition number* of $\mathbf{A}$. We refer to singular matrices $\mathbf{A}$, having $\sigma_{\min}(\mathbf{A}) = 0$, as having an infinite condition number. In the case that $\mathbf{A}$ is a symmetric matrix, requiring that $n = d$, it follows that $\mathbf{A}$ has n real-valued eigenvalues. In this case, we can order these eigenvalues, and denote them similarly via

$$\lambda_{\max}(\mathbf{A}) = \lambda_1(\mathbf{A}) \geq \lambda_2(\mathbf{A}) \geq \cdots \geq \lambda_n(\mathbf{A}) = \lambda_{\min}(\mathbf{A})$$

We note that the majority of matrices in this thesis will not necessarily be symmetric, and therefore singular values will often by considered when this is the case.

Matrix Norms. We now define the matrix norms which will be used in the thesis. The *spectral norm* of a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ is defined by its maximum singular value, namely $\|\mathbf{A}\|_2 = \sigma_{\max}(\mathbf{A})$. Equivalently, the spectral norm of $\mathbf{A}$ is the maximum of the modified *Rayleigh–Ritz quotient*:

$$\|\mathbf{A}\|_2 = \max_{x\in\mathbb{R}^d} \frac{\|\mathbf{A}x\|_2}{\|x\|_2}$$

Next, the *Frobenius norm* of $\mathbf{A}$ is just the L_2 norm of $\mathbf{A}$ thought of as a vector in $\mathbb{R}^{nd}$, namely

$$\|\mathbf{A}\|_F = \left(\sum_{i,j} \mathbf{A}_{i,j}^2\right)^{1/2} = \left(\sum_{i=1}^{n} \sigma_i^2(\mathbf{A})\right)^{1/2}$$

where the latter inequality is an easy consequence of the singular value decomposition (see the following paragraph) and the Pythagorean theorem. For $p \geq 1$, we write define the *Schatten p-norm* of $\mathbf{A}$ by $\|\mathbf{A}\|_{\mathcal{S}_p} = (\sum_{i=1}^n \sigma_i^p(\mathbf{A}))^{1/p}$, and the (p,k)*-Ky-Fan norm* of $\mathbf{A}$ by $\|\mathbf{A}\|_{\mathsf{KF}(p,k)} = (\sum_{i=1}^k \sigma_i^p(\mathbf{A}))^{1/p}$. If p is not specified for a Ky-Fan norm, it is assumed to be 1, namely $\|\mathbf{A}\|_{\mathsf{KF}(k)} = \|\mathbf{A}\|_{\mathsf{KF}(1,k)}$. We remark that the latter is what is usually referred to in the literature as a Ky-Fan norm.

Singular Value Decomposition (SVD) Suppose the matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ is rank $r \leq \min\{n,d\}$. Then we can consider the *singular value decomposition* (SVD) of $\mathbf{A}$, which we denote by $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where both matrices $\mathbf{U} \in \mathbb{R}^{n\times r}, \mathbf{V} \in \mathbb{R}^{d\times r}$ have orthonormal columns, and $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ is a diagonal matrix with the (non-zero) singular values σ_i on the diagonal (i.e, $\mathbf{\Sigma}_{i,i} = \sigma_i(\mathbf{A})$). Recall that the columns of $\mathbf{U}$ are the left singular vectors of the matrix $\mathbf{A}$, and the rows of $\mathbf{V}$ are the right singular vectors of $\mathbf{A}$. Given the SVD, we can define the *Moore-Penrose pseudoinverse* of $\mathbf{A}$, which is denoted by $\mathbf{A}^+$, and defined via $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T$.

Given a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ and an integer $0 \leq k \leq n$, we define $\mathbf{D}_{-k}$ to be the diagonal matrix resulting from setting the top k (in magnitude) largest entries in $\mathbf{D}$ equal to 0. Further, we write $\mathbf{D}_k = \mathbf{D} - \mathbf{D}_{-k}$, i.e., $\mathbf{D}_k$ is the result of setting the bottom $n-k$ smallest

(in magnitude) entries to 0. Using this notation, for any matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ we can introduce *truncated SVD* of $\mathbf{A}$, denoted by $\mathbf{A}_k$, by defining $\mathbf{A}_k = \mathbf{U}\mathbf{\Sigma}_k\mathbf{V}^\top$. An important fact about the truncated SVD is that it is the best rank-k approximation to $\mathbf{A}$ under the Frobenius norm error:

$$\|\mathbf{A} - \mathbf{A}_k\|_F^2 = \sum_{i>k} \sigma^2(\mathbf{A}) = \min_{\mathbf{B} \text{ rank-k}} \|\mathbf{A} - \mathbf{B}\|_F^2$$

We remark that the truncated SVD is also the best rank-k approximation to $\mathbf{A}$ under the spectral norm. Similarly, we define the matrix $\mathbf{A}_{-k} = \mathbf{U}\mathbf{\Sigma}_{-k}\mathbf{V}^\top$.

For the special case when $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric, we can define its *eigenvalue decomposition* via $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ is the diagonal matrix containing the eigenvalues of $\mathbf{A}$, i.e., $\mathbf{\Lambda}_{i,i} = \lambda_i(\mathbf{A})$. A real-symmetric matrix $\mathbf{A}$ is said to be *Positive Semi-Definite* (PSD) if $\lambda_{\min} \geq 0$, which is equivalent to having $x^\top \mathbf{A} x \geq 0$ for all $x \in \mathbb{R}^n$. Given this definition, we introduce the *Loewner ordering* on symmetric matrices.

Definition 2.1.1 (Loewner Ordering)**.** *For symmetric matrices* $\mathbf{B}, \mathbf{D}$*, we write* $\mathbf{B} \succeq \mathbf{D}$ *if* $\mathbf{B} - \mathbf{D}$ *is PSD.*

Notice that if $\mathbf{B} \succeq \mathbf{D}$, then by definition $x^\top \mathbf{B} x \geq x^\top \mathbf{D} x$ for all $x \in \mathbb{R}^n$. Then by an application of the Courant-Fischer variational principle for eigenvalues, we have that $\lambda_i(\mathbf{B}) \geq \lambda_i(\mathbf{D})$ for all $i \in [n]$.

2.2 Probability Background for Sketching

In this section, we describe classic tools, as well as recent developments, from probability that will be commonly applied in the thesis. We begin in Section 2.2.1 with results for the concentration of random scalar variables, as well as matrix concentration inequalities, and then in Section 2.2.2, we introduce the central family of p-stable distributions,

2.2.1 Concentration Inequalities

We will use several concentration inequalities in this thesis. For completeness, we begin by stating the standard Chernoff-Hoeffding bound for sums of independent, bounded random variables.

Proposition 2.2.1 (Chernoff-Hoeffding, [Hoe94]). *Let $X_1, \ldots, X_n$ be independent variables, and $\{a_i\}_{i\in[n]}, \{b_i\}_{i\in[n]}$ reals satisfying $a_i \leq X_i \leq b_i$ for each $i \in [n]$ almost surely. Let $X = \sum_{i=1}^n X_i$, $\mu = \mathbf{E}[X]$, and fix any $\delta > 0$. Then we have the upper tail*

$$\mathbf{Pr}[X \geq (1+\delta)\mu] \leq \exp\left(-\frac{2\delta^2\mu^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

and the lower tail

$$\mathbf{Pr}[X \leq (1-\delta)\mu] \leq \exp\left(-\frac{2\delta^2\mu^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

Additionally, we will utilize several other common concentration inequalities: Khintchine's ineqality, McDiarmid's inequality (also known as the method of bounded differences), and Bernstein's inequality.

Proposition 2.2.2 (Khintchine's inequality [Haa81]). *Let $x \in \mathbb{R}^n$ and let $\varphi_1, \ldots, \varphi_n$ be independent and uniformly distributed on $\{1, -1\}$. Then for any $t > 0$, setting $Q = \sum_{i=1}^n \varphi_i x_i$ we have*

$$\mathbf{Pr}[|Q| > t\|x\|_2] < 2e^{-t^2/2}$$

Proposition 2.2.3 (McDiarmid's inequality [McD89]). *Let $X_1, X_2, \ldots, X_n$ be independent random variables, and let $\psi(x_1, \ldots, x_n)$ by any function that satisfies*

$$\sup_{x_1,\ldots,x_n,\hat{x}_i} \left|\psi(x_1, x_2, \ldots, x_n) - \psi(x_1, \ldots, x_{i-1}, \hat{x}_i, x_{i+1}, \ldots, x_n)\right| \leq c_i \quad \text{for } 1 \leq i \leq n$$

Then for any $\epsilon > 0$, we have $\mathbf{Pr}\left[\left|\psi(X_1, \ldots, X_n) - \mathbf{E}[\psi(X_1, \ldots, X_n)]\right| \geq \epsilon\right] \leq 2\exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}\right)$.

Proposition 2.2.4 (Bernstein's inequality [McD89]). *Let $X_1, X_2, \ldots, X_n$ be independent mean-zero random variables, such that $|X_i| \leq M$ almost surely for each $i \in [n]$. Then for any $t > 0$,*

we have

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{n} X_i\right| \geq t\right] \leq \exp\left(-\frac{\frac{1}{2}t^2}{\sum_{i=1}^{n}\mathbf{E}\left[X_i^2\right]+\frac{1}{3}Mt}\right)$$

Matrix Concentration. A common tool in sketching theory is the application of random linear transformations to project high dimensional objects onto lower dimensional subspaces. However, the analysis of such random projections often require bounding properties of the *spectrum* of the random matrices which result from these transformations. This is particularly the case in numerical linear algebra.

For an example, given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, we will often want to generate a random matrix $\mathbf{S} \in \mathbb{R}^{k\times n}$ with the strong property that $\|\mathbf{SA}x\|_2 = (1 \pm \epsilon)\|\mathbf{A}x\|_2$ for *all* $x \in \mathbb{R}^d$.[2] While proving this fact for a *single* $x \in \mathbb{R}^n$ can be accomplished via the usage of standard scalar-value concentration bounds as described above, demonstrating such results simultaneously for all x will be more easily accomplished by bounding the spectrum of $\mathbf{SA}$.

In order to bound the spectrum of random matrices, we will need results from the theory of *random matrix concentration*. We state only one such result here, which is the matrix-valued extension of the Chernoff bound, as it will be referred to several times in the thesis. More detailed results in the theory of random matrices, such as the *interior eigenvalue matrix Chernoff bounds* (117, due to Gittens and Tropp [GT11]), will be introduced where needed.

Theorem 7 (Matrix Chernoff, Chapter 5 [Tro15])**.** *Consider a finite sequence* $\{\mathbf{X}_j\}$ *of independent, random, positive-semidefinite matrices with dimension* $d \times d$*, and assume that* $\|\mathbf{X}_j\|_2 \leq L$ *for some value* L *almost surely. Given an integer* $k \leq n$*, define*

$$\mu_{\max} = \lambda_{\max}\left(\sum_j \mathbf{E}\left[\mathbf{X}_j\right]\right)$$
$$\mu_{\min} = \lambda_{\min}\left(\sum_j \mathbf{E}\left[\mathbf{X}_j\right]\right)$$

[2] This is known as a *subspace embedding*, see Section 2.4.

then for any $\delta \in [0,1)$ and $\epsilon \geq 0$, we have the tail inequalities

$$\mathbf{Pr}\left[\lambda_{\min}(\sum_j \mathbf{X}_j) \leq (1-\delta)\mu_{\min}\right] \leq d \cdot \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu_{\min}/L}$$
$$\mathbf{Pr}\left[\lambda_{\max}(\sum_j \mathbf{X}_j) \geq (1+\epsilon)\mu_{\max}\right] \leq d \cdot \left[\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right]^{\mu_{\max}/L}$$

2.2.2 Stable Distributions

A particular important class of distributions utilized in this thesis are the *stable distribution*. Formally, a distribution $\mathcal{D}$ is said to be stable if for any constants a, b, then whenever $X, Y, Z \sim \mathcal{D}$ we have that $aX + bY \stackrel{d}{=} cZ + d$ for some constants c, d, where the symbol $\stackrel{d}{=}$ denotes equality in distribution. In fact, a full characterization of such random variables is known. As it happens, every such stable distribution $\mathcal{D}$ is a *p-stable distribution* (to be defined shortly) for some parameter $p \in (0,2]$. In fact, a stable distribution is fully parameterized by its *stability* parameter $p \in (0,2]$, a *skewness* parameter $\beta \in [-1,1]$, a *scale* parameter $\gamma > 0$, and a *location* parameter $\delta \in \mathbb{R}$. Following Nolan [Nol], we denote this distribution as $S(p,\beta,\gamma,\delta)$, and refer the reader to [Nol] and [UZ11] for a further discussion on the parameterization and behavior of stable distributions.

Definition 2.2.5 (Stable distribution, general)**.** *A random variable X is said to be p-stable if it is drawn from the distribution $S(p,\beta,\gamma,\delta)$ for some $p \in (0,2]$, $\beta \in [-1,1]$, $\gamma > 0$, and $\delta \in \mathbb{R}$, where $S(p,\beta,\gamma,\delta)$ is defined via its characteristic function:*

$$\mathop{\mathbf{E}}_{Z \sim S(p,\beta,\gamma,\delta)}\left[e^{-itZ}\right] = \begin{cases} \exp\left(-\gamma^p |t|^p \left[1 - i\beta \tan(\frac{\pi p}{2}) \textit{sign}(t)\right] + i\delta t\right) & \text{if } p \neq 1 \\ \exp\left(-\gamma|t|\left[1 + i\beta\frac{2}{\pi}\textit{sign}(t)\log(|t|)\right] + i\delta t\right) & \text{if } p = 1 \end{cases}$$

where $\textit{sign}(t) \in \{1,-1\}$ is the sign of a real $t \in \mathbb{R}$.

We will be mostly concerned with a particular class of p-stable distributions, known as the

standard p-stable distribution, which result when one sets $\beta = \delta = 0$ and $\gamma = 1$. For $p \in (0, 2]$, we denote this distribution by $\mathcal{D}_p$. Standard p-stable distributions are central to many algorithms in streaming and numerical linear algebra. Two prominent examples of standard p-stable distributions are the cases of $p \in \{1, 2\}$: for $p = 2$ the distribution $\mathcal{D}_2$ is the well-known Gaussian distribution $\mathcal{N}(0, 1)$, and for $p = 1$ the distribution $\mathcal{D}_1$ is the *Cauchy* distribution. The formal definition follows.

Definition 2.2.6 (Standard p-stable Distribution)**.** *Fix any $p \in (0, 2]$. Then the standard p-stable distribution, denoted $\mathcal{D}_p$, is the distribution with characteristic function*

$$\mathop{\mathbf{E}}_{Z \sim \mathcal{D}_p} \left[e^{itZ} \right] = e^{-|t|^p}$$

The distribution $\mathcal{D}_p$ has the property that if $X_1, \ldots, X_n, Z \sim \mathcal{D}_p$, and $a \in \mathbb{R}^n$ is a fixed vector, we have

$$\sum_{i=1}^{n} a_i X_i \stackrel{d}{=} \|a\|_p Z$$

Standard methods for generating p-stable random variables are discussed in [Nol] and [UZ11] . We present a particularly elegant method for generation here, which is due to Chambers [CMS76].

Proposition 2.2.7 ([CMS76])**.** *Fix any $p \in (0, 1)$. Then $X \sim \mathcal{D}_p$ can be generated as follows:*

1. *Generate $\theta \sim$ Uniform$[-\frac{\pi}{2}, \frac{\pi}{2}]$.*
2. *Generate $r \sim$ Uniform$[0, 1]$.*
3. *Set*

$$X \leftarrow \frac{\sin(p\theta)}{\cos^{1/p}(\theta)} \cdot \left(\frac{\cos(\theta(1-p))}{\ln(1/r)} \right)^{\frac{1-p}{p}}$$

An peculiar and important property of p-stable distributions for $p < 2$ is that they heavy tails – namely we have the following fact.

Fact 2.2.8 (Tail behavior of standard p-stable, Theorem 1.12 [Nol])**.** *Fix $p \in (0, 2)$, and let $\mathcal{D}_p$*

be the standard p-stable distribution. Then we have

$$\lim_{t\to\infty} \mathop{\mathbf{Pr}}_{X\sim S(p,\beta,\gamma,0)}[|X| > t] \cdot \frac{t^p}{c_p} = 1$$

where $c_p = \frac{1}{\pi}\sin\left(\frac{\pi\cdot p}{2}\right)\Gamma(p)$.

As a consequence of Fact 2.2.8, for $p < 2$ the variance of p-stable variables infinite, and for $p \leq 1$ the expectation of p-stable variables is undefined.

Indyk's p-Stable Sketch We now state a celebrated and central application of p-stable variables to sketching tasks, known as Indyk's *p-stable estimator*, introduced in the paper [Ind06].

Theorem 8 (Indyk's p-Stable Sketch [Ind06])**.** *Fix any $p \in (0, 2]$ bounded away from 0 by a constant, $\epsilon \in (0, 1)$, and let $x \in \mathbb{R}^n$ be a fixed vector. Let $\theta_p = \text{median}(|\mathcal{D}_p|)$ be the median of the random variable $|Z|$ where $Z \sim \mathcal{D}_p$. Let $\mathbf{S} \in \mathbb{R}^{k\times n}$ be a matrix consisting of independent identically distributed variables from $\mathcal{D}_p$, where $k = \Theta(\frac{1}{\epsilon^2}\log(1/\delta))$. Then with probability $1-\delta$ over the draw of $\mathbf{S}$, we have*

$$(1-\epsilon)\|x\|_p < \text{median}_{i\in[k]}\left\{\frac{|(\mathbf{S}x)_i|}{\theta_p}\right\} < (1+\epsilon)\|x\|_p$$

Notice that the above theorem immediately gives rise to a $O(\frac{1}{\epsilon^2}\log n)$-bits of space algorithm for estimating $\|x\|_p$ in the random oracle turnstile streaming model. One generates $\mathbf{S}$ to $\gamma = 1/\text{poly}(n)$ precision, and outputs the estimator above from the sketch $\mathbf{S}x$. The resulting estimate will be within $\gamma n\|x\|_\infty < 1/\text{poly}(n)$ of the estimate had it been generated to infinite precision, and moreover $\mathbf{S}x$ requires only $O(\frac{1}{\epsilon^2}\log n)$-bits of space to store. The task of derandomizing this algorithm is more nuanced, and was accomplished in [Ind06] by usage of *Nisan's Pseudorandom Generator* [Nis92], albiet with an increased space complexity of $O(\frac{1}{\epsilon^2}\log^2 n)$. This derandomization was later improved to an optimal $O(\frac{1}{\epsilon^2}\log n)$-bits by Kane, Nelson, and Woodruff [KNW10a] by utilizing limited independence.

2.3 Streaming Background

The Data Stream Model. We begin by restating, in more detail, the formal definition of the data stream model, as first introduced in Section 1.1.

Definition 1.1.1 (The Streaming Model). *Fix integers $n, m, M > 0$ such that $\log(mM) = O(\log n)$. In the streaming model, a vector $x \in \mathbb{R}^n$, known as the frequency vector, is initialized to $\vec{0}$. Then, it received a sequence of updates $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$, where $i_t \in [n]$ and $\Delta \in \{-M, \ldots, M\}$. Each update (i, Δ) causes the change:*

$$x_{i_t} \leftarrow x_{i_t} + \Delta$$

*At the end of the stream, the vector $x \in \mathbb{R}^n$ is given by $x_i = \sum_{t:i_t=i} \Delta_t$ for all $i \in [n]$. This model is known as the **general turnstile model.** If we are also promised that $x_i \geq 0$ for all $i \in [n]$ and at all points in the stream, this is known as the **strict turnstile model**. Furthermore, if we are promised that $\Delta_t \geq 0$ for all $t \in [m]$, this is known as the **insertion-only model**.*

The goal of a streaming algorithm is to observe the updates (i_t, Δ_t) sequentially, and at the end of the stream answer a query about the final frequency vector, using as little space and update time as possible.

Given a stream $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$ of length m, for any time step $t \in [m]$ we write $x^{(t)}$ to denote the state of the frequency vector after the first t updates. In other words, $x^{(t)} \in \mathbb{R}^n$ is the vector defined by:

$$x_i^{(t)} = \sum_{\tau \leq t \,:\, i_\tau = i} \Delta_\tau$$

for all $i \in [n]$. More generally, if $g : \mathbb{R}^n \to \Omega$ is any function of the stream taking values in some set Ω, we write $g^{(t)}$ to denote the value of $g(x)$ after the first t updates, namely: $g^{(t)} = g(x^{(t)})$.

Bounded Deletion Model. In addition to the turnstile and insertion-only models defined above, we also formally introduced the bounded deletion model, which is introduced and the primary

subject of Chapter 6, and which will be referred to in several other chapters in this thesis. Given a stream of updates $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$ for a stream vector $x \in \mathbb{R}^n$, we define the *insertion vector* $I \in \mathbb{R}^n$ to be the frequency vector of the substream of positive updates ($\Delta_t \geq 0$), and the *deletion vector* $D \in \mathbb{R}^n$ to be the entry-wise absolute value of the frequency vector of the substream of negative updates. More formally:

$$I_j = \sum_{\substack{t \,:\, i_t = j \\ \Delta_t \geq 0}} \Delta_t \qquad \text{and} \qquad D_j = \sum_{\substack{t \,:\, i_t = j \\ \Delta_t < 0}} |\Delta_t|$$

for each $j \in [n]$. Notice that $x = I - D$ by definition. We then define the bounded deletion model as follows:

Definition 6.1.1 (α-Bounded Deletion Streams). *For $\alpha \geq 1$ and $p \geq 0$, a data stream satisfies the L_p α-property if*

$$\|I + D\|_p \leq \alpha \|x\|_p$$

Notice that for $p = 1$, the definition simply asserts that the final L_1 norm of x must be no less than a $1/\alpha$ fraction of the total weight of updates in the stream $\sum_{t=1}^{m} |\Delta_t|$. For strict turnstile streams, this is equivalent to the number of deletions being less than a $(1 - 1/\alpha)$ fraction of the number of insertions, hence a bounded deletion stream. For $p = 0$, the α-property simply states that $\|x\|_0$, the number of non-zero coordinates at the end of the stream, is no smaller than a $1/\alpha$ fraction of the number of distinct elements seen in the stream.

Observe for $\alpha = 1$, we recover the insertion-only model, whereas for $\alpha = mM$ in the L_1 case or $\alpha = n$ in the L_0 case we recover the turnstile model (with the minor exception of streams with $\|x\|_p = 0$). So α-property streams are a natural parameterized intermediate model between the insertion-only and turnstile models.

Random Oracle Model. We now state again the random oracle model of streaming. In several cases, we will first develop algorithms or intuition for the random oracle case, before later removing this assumption (i.e., *derandomizing* the algorithm). We also remark that lower bounds

proved in the *public coin* model of communication immediately apply to the random oracle streaming model. Since many lower bounds in the streaming literature indeed derive from the public coin model, most known lower bounds additionally apply to the random oracle model.

Definition 2.3.1 (Random Oracle Model)**.** *In the random oracle model of streaming, an algorithm $\mathcal{A}$ is given oracle access to an arbitrarily arbitrarily long string $r \in \{0,1\}^*$ of random bits, which is generated once at the start of the stream. At any time, the algorithm can query the value for r_i for any $i \in \mathbb{N}$, without counting against the space complexity of the algorithm.*

2.3.1 Some Standard Streaming Problems and their Complexity

We now introduce three central streaming tasks in the literature, which will be referred to throughout Part I of the thesis. In all the following, we fix some vector $x \in \mathbb{R}^n$ which is defined by a stream of updates $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$. For different classes of streams—such as turnstile, insertion only, and bounded deletion streams—we obtain distinct streaming streaming problems for the same task defined below. For instance, the heavy hitters problem in turnstile streams will have a different complexity than the heavy hitters problem for insertion-only streams.

In all of the following, we fix some precision parameter $\epsilon \in (0,1)$. We begin with the task of norm estimation.

Definition 2.3.2 (L_p norm estimation)**.** *Given a parameter $p > 0$, the L_p-norm[3] estimation problem is to output a value $\tilde{L}$ at the end of the stream satisfying*

$$(1-\epsilon)\|x\|_p \leq \tilde{L} \leq (1+\epsilon)\|x\|_p$$

with probability at least $2/3$.

For $p > 2$, it is known that polynomial space is required for L_p estimation [CKS03, IW05], even in the insertion-only model. For turnstile streams and $p \in (0,2]$, the complexity of L_p norm

[3]Note that the following is only a norm for $p \geq 1$, and we abuse terminology for the case of $p \in (0,1)$.

estimation is understood, with tight upper and lower bounds of $\Theta(\frac{1}{\epsilon^2}\log n)$ given in [KNW10a]. For the insertion-only case and $p \in (1, 2]$, however, thus far there is still a gap in our understanding of the space complexity. Namely, the best known lower bound is $\Omega(\epsilon^{-2} + \log n)$ [Woo04], and in the random oracle model the lower bound is only $\Omega(\epsilon^{-2})$. On the other hand, the best upper bound for $p \in (1, 2]$ is simply to run the $\Theta(\frac{1}{\epsilon^2}\log n)$ turnstile algorithm. We resolve the space complexity of L_p norm estimation for $p \in (0, 1)$, up to $\log\log n, \log(1/\epsilon)$ factors, in Chapter 4, where we give bounds of $\tilde{O}(\frac{1}{\epsilon^2})$ in the random oracle model and $\tilde{O}(\frac{1}{\epsilon^2} + \log n)$ in the standard model.

We often treat the case of $p = 0$ for L_p estimation separately; this task is also referred to as the distinct elements problem.

Definition 2.3.3 (Distinct elements/L_0 estimation)**.** *The distinct elements, or L_0, estimation problem is to output a value $\tilde{L}$ at the end of the stream satisfying*

$$(1 - \epsilon)\|x\|_0 \leq \tilde{L} \leq (1 + \epsilon)\|x\|_0$$

with probability at least $2/3$, where $\|x\|_0 = |\{i \in [n] \mid x_i \neq 0\}$ is the number of non-zero coordinates in x.

Tight bounds for L_0 estimation in the insertion-only model were given by Kane, Nelson, and Woodruff [KNW10b], using $O(\frac{1}{\epsilon^2} + \log n)$ bits of space, matching known lower bounds [AMS96, IW03, Woo04]. For the turnstile model, the same paper [KNW10b] gave an upper bound of $\tilde{O}(\frac{1}{\epsilon^2}\log n)$, which matches the lower bounds of [AMS96, KNW10a] up to $\log\log n, \log(1/\epsilon)$ factors.

We now state the well-known *heavy hitters* problem, which is to find the largest coordinates in the frequency vector x.

Definition 2.3.4 (L_p Heavy Hitters)**.** *Fix a parameter $p > 0$. The heavy hitters problem is to output a set $S \subset [n]$ such that with probability at least $2/3$:*

1. *For every $i \in [n]$ with $|x_i| \geq \epsilon\|x\|_p$ we have $i \in S$, and*

2. *For every $j \in S$ we have $|x_i| \geq (\epsilon/2)\|x\|_p$.*

A coordinate i satisfying $|x_i| \geq \epsilon\|x\|_p$ is known as an ϵ-heavy hitter (for the L_p norm).

A slightly weaker version of the heavy hitters problem only requires the first condition that $i \in S$ for all i such that $|x_i| \geq \epsilon\|x\|_p$, and instead adds the secondary condition that $|S| = O(1/\epsilon^p)$—note that this is weaker since at most $O(1/\epsilon^p)$ coordinates can satisfy $|x_i| \geq (\epsilon/2)\|x\|_p$. Almost always the case of either $p = 1$ or $p = 2$ is treated in the literature, with the case of $p = 2$ being slightly more common in theory.

For the case of $p = 2$ and the insertion only model, an algorithm using $O(\epsilon^{-2}\log(1/\epsilon)\log n)$-bits of space was given, which is optimal up to the factor of $\log(1/\epsilon)$ simply since the set S of ϵ-heavy hitters itself can require $\Omega(\epsilon^{-2}\log n)$ bits to store. For the turnstile case, the classic count-sketch algorithm [CCFC02b] (described in the following section), achieves $O(\epsilon^{-2}\log^2 n)$-bits of space, tightly matching the lower bound of [JST11].

2.3.2 Count-Sketch

We now describe the well-known streaming algorithm known as *count-sketch*, introduced by Charikar, Chen, and Farach-Colton [CCFC02b] in the streaming literature, which will play a prominent role in many of the results in this thesis. The core idea of count-sketch is exceptionally elegant: given a high-dimensional vector $x \in \mathbb{R}^n$, randomly hash the coordinates x_i of x into $k \ll n$ buckets, multiply each coordinate by a random sign, sum up the coordinates in each bucket with their signs, and repeat (some number of times). Since its introduction in since [CCFC02b], this concept has been rediscovered at least three times in machine learning, numerical linear algebra, and compressed sensing [WDL+09, CW17, Don06], and also goes under the name of *feature hashing* and the *sparse Johnson-Lindenstrauss transformation*. In what follows, we present a unified and comprehensive introduction to count-sketch, which will serve as a foundation for its various applications throughout the thesis.

Count-sketch is a linear sketch, meaning it stores the matrix-vector product $\mathbf{S}x$, where $x \in \mathbb{R}^n$ is a vector—often it will be the frequency vector of a stream—and $\mathbf{S} \in \mathbb{R}^{t\times n}$ is a randomized

sketching matrix. The matrix $\mathbf{S}$ can be broken into d blocks with k rows each, where $t = dk$, with the property that each of the blocks are drawn independently from the same distribution (i.e., each of the blocks is an independent repetition of the aforementioned hashing process). For this reason, it will often be convenient to think of reshaping the product $\mathbf{S}x \in \mathbb{R}^t$ into a matrix $\mathbf{A} \in \mathbb{R}^{d \times k}$, where each row of $\mathbf{A}$ corresponds to one of the d independent blocks. This interpretation is often taken in the literature, and is the one we will follow in the subsequent exposition.

Formally, the (reshaped) count-sketch data structure is a matrix $\mathbf{A}$ with d rows and k columns. When run on a stream $x \in \mathbb{R}^n$, for each row $i \in [d]$, count-sketch picks a uniform random mapping $h_i : [n] \to [k]$ and $g_i : [n] \to \{1, -1\}$. In general, h_i and g_i need only be 4-wise independent hash functions, but in this chapter we will use fully-independent hash functions (and later relax this condition when derandomizing). Whenever an update Δ to item $v \in [n]$ occurs, count-sketch performs the following updates:

$$\mathbf{A}_{i,h_i(v)} \leftarrow \mathbf{A}_{i,h_i(v)} + \Delta \cdot g_i(v) \quad \text{for } i = 1, 2, \ldots, d$$

Thus, at the end of the stream, the state of the count-sketch table $\mathbf{A} \in \mathbb{R}^{d \times k}$ is given by:

$$\mathbf{A}_{i,j} = \sum_{v \in [n] \,:\, h_i(v) = j} g_i(v) \cdot x_v \tag{2.1}$$

for every $i \in [d]$ and $j \in [k]$. Note that while we will not implement the h_i's as explicit hash functions, and instead generate i.i.d. random variables $h_i(1), \ldots, h_i(n)$, we will still use the terminology of hash functions. In other words, by *hashing* the update (v, Δ) into the row $\mathbf{A}_i$ of count-sketch, we mean that we are updating $\mathbf{A}_{i,h_i(v)}$ by $\Delta \cdot g_i(v)$. By hashing the coordinate x_v into $\mathbf{A}$, we mean updating $\mathbf{A}_{i,h_i(v)}$ by $g_i(v)x_v$ for each $i = 1, 2, \ldots, d$. Using this terminology, each row of count-sketch corresponds to randomly hashing the indices in $[n]$ into k buckets, and then each bucket in the row is a sum of the frequencies x_i of the items which hashed to it multiplied by random ± 1 signs. As noted, the entries of $\mathbf{A}$ are given by the entries of the matrix vector product $\mathbf{S}x \in \mathbb{R}^{dk}$ for a matrix $\mathbf{S} \in \{-1, 1, 0\}^{dk \times n}$. By indexing into the rows of $\mathbf{S}$ by

tuples $(i, j) \in [d] \times [k]$, the entries of $\mathbf{S}$ are given by

$$\mathbf{S}_{(i,j),v} = \mathbf{1}\left(h_i(v) = j\right) \cdot g_i(v)$$

One of the main features of count-sketch is that given $\mathbf{A}$ and knowledge of the hash functions h_i, g_i, one can obtain an estimate vector $y \in \mathbb{R}^n$ of the frequency vector x such that $\|y - x\|_\infty$ is small. Formally, this vector y satisfies the following guarantee:

Theorem 9 ([CCFC02b])**.** *If $d = \Theta(\log \frac{1}{\delta})$ and $k = 6/\epsilon^2$, then for a fixed $v \in [n]$ we have*

$$|y_v - x_v| < \epsilon \|x_{\text{tail}(1/\epsilon^2)}\|_2$$

with probability at least $1 - \delta$, where y is given

$$y_v = \text{median}_{i \in [d]} \left\{ g_i(v) \cdot \mathbf{A}_{i,h_i(v)} \right\} \tag{2.2}$$

Moreover, setting $d = 1$, the result holds with probability at least $2/3$. Consequentially, if $c \geq 1$ is any constant, and we set $d = \Theta(\log n)$, we have

$$\|y - x\|_\infty < \epsilon \|x_{\text{tail}(1/\epsilon^2)}\|_2$$

with probability at least $1 - n^{-c}$, where y is as defined in Equation 2.2. Furthermore, if we instead set $y_v = \text{median}_{i \in [d]} |\mathbf{A}_{i,h_i(v)}|$, then the same bounds above above hold replacing x with $|x|$.

As we will see later in Section 2.4.1, count-sketch features prominently in the application of sketching to numerical linear algebra. In particular, the count-sketch matrix $\mathbf{S}$, with $d = 1$ repetitions of the hashing process, satisfies a powerful property known as the *subspace embedding* guarantee. This guarantee is crucial for both linear regression and low-rank approximation, and will be utilized in Chapters 10 and 11 for this purpose.

2.4 Numerical Linear Algebra

We now introduce several concepts and tools from linear algebra which will be particularly useful for the design and analysis of sketches in Part II. We begin with the notion of the *leverage scores* of a matrix $\mathbf{A}$.

Definition 2.4.1 (Leverage Scores)**.** *Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be its singular value decomposition. For each $i \in [n]$, we define the i-th leverage score of $\mathbf{A}$, denoted $\tau_i(\mathbf{A}) = \tau_i$, by*

$$\tau_i = \|\mathbf{U}_{i,*}\|_2^2$$

The leverage scores can be equivalently defined in terms of quadratic forms with the psuedo-inverse of $\mathbf{A}^T\mathbf{A}$. Namely, for each $i \in [n]$, we have:

$$\tau_i(\mathbf{A}) = \mathbf{A}_{i,*}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}_{i,*}^T$$

Intuitively, the i-th leverage score of a matrix $\mathbf{A}$ can be interpreted as the "uniqueness" of the i-th row of $\mathbf{A}$—namely, rows that "stick out" from the rest of $\mathbf{A}$ have high leverage. Consequently, if one would like to sample a subset of the rows of $\mathbf{A}$ while approximately preserving important geometric information about $\mathbf{A}$, then rows $\mathbf{A}_{i,*}$ with large leverage score should be sampled with *higher* probability. We will see one way in which this can be formalized to obtain subspace embeddings below.

2.4.1 Subspace Embeddings

We now formally define the notion of a *subspace embedding* (SE), which was informally described and introduced in Section 1.2.

Definition 2.4.2 (Subspace Embedding)**.** *Fix a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, and fix $\epsilon \in (0, 1)$. Then a matrix $\mathbf{S} \in \mathbb{R}^{n \times k}$ is said to be an ϵ-subspace embedding for $\mathbf{A}$ if simultaneously for all $x \in \mathbb{R}^d$*

we have

$$\|\mathbf{S}\mathbf{A}x\|_2 = (1 \pm \epsilon)\|\mathbf{A}x\|_2$$

Moreover, a distribution $\mathcal{D}$ over matrices $\mathbf{S} \in \mathbb{R}^{n\times k}$ is said to be a ϵ-subspace embedding for $\mathbf{A}$ if $\mathbf{S} \sim \mathcal{D}$ is an ϵ-subspace embedding for $\mathbf{A}$ with probability $2/3$ over the draw of $\mathbf{S}$.

In Section 1.2 we described at a high level how subspace embeddings are a crucial ingredient in solving optimization problems in linear algebra via sketching. We recall here how they can be utilized for linear regression, where one wants to solve $\min_x \|\mathbf{A}x - b\|_2$ for a tall matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ with $n \gg d$. Note that directly solving for x requires computing the covariance matrix of $\mathbf{A}$, which takes $O(nd^{\omega-1})$ time for general $\mathbf{A}$, where $\omega \approx 2.376$ is the exponent of fast matrix multiplication. Instead, one can first compute $\mathbf{S}\mathbf{A}$ and $\mathbf{S}b$, where $\mathbf{S} \in \mathbb{R}^{k\times n}$ is a random matrix. Then if $\mathbf{S}$ is an ϵ-subspace embedding for $[\mathbf{A}, b]$, since the vector $(\mathbf{A}x - b)$ is in the span of $[\mathbf{A}, b]$ for any $x \in \mathbb{R}^d$, it follows that the solution $\hat{x}$ to $\min_x \|\mathbf{S}\mathbf{A}x - \mathbf{S}b\|_2$ will be within a $(1 + \epsilon)$ factor of the optimal solution cost. However, if $k \ll n$, then solving for $\hat{x}$ can now be accomplished much faster—in $O(nd^{\omega-1})$ time.

Note that the bottleneck in the above procedure is computing the product $\mathbf{S}\mathbf{A}$, which requires $(d^\omega n)$ time even if $k = O(d)$.[4] Thus, we will often try to find subspace embeddings $\mathbf{S}$ which can be applied to $\mathbf{A}$ very quickly, ideally in $O(\text{nnz}(\mathbf{A}))$ time. This will be possible when $\mathbf{S}$ is very sparse, or has special structure.

It turns out that there are many classes of random matrices which will satisfy the subspace embedding guarantee with good probability. We state three of the most common classes here. The simplest example is the distribution of matrices with independent Gaussian entries.

Theorem 10 (Theorem 2.3 [W+14])**.** *Fix $\mathbf{A} \in \mathbb{R}^{n\times d}$, and let $\mathbf{S} = \frac{1}{\sqrt{k}} \cdot \mathbf{R} \in \mathbb{R}^{k\times n}$, where $\mathbf{R}$ is a matrix of independent standard Gaussian random variables (i.e., $\mathcal{N}(0, 1)$). Then if $k = \Omega(\frac{1}{\epsilon^2}(d + \log(1/\delta)))$, the matrix $\mathbf{S}$ is a subspace embedding for $\mathbf{A}$ with probability at least $1 - \delta$.*

[4] It is not hard to see that $k \geq d$ is necessary for any subspace embedding. The reason for this is that if $k < d$, then for a rank-d matrix $\mathbf{A}$ there is a non-zero vector x such that $\mathbf{A}x$ is in the kernel of $\mathbf{S}$, meaning that $\mathbf{S}\mathbf{A}x = 0$ but $\mathbf{A}x \neq 0$, thus violating the subspace embedding guarantee.

The downside of setting $\mathbf{S}$ to be a dense matrix of Gaussian random variables is that computing the product $\mathbf{SA}$ is expensive — it requires time $O(nd^{\omega-1})$. Instead, we will often like to use random matrices which are sparse enough so that $\mathbf{SA}$ can be computed quickly. In fact, we have already seen a family of random matrices which will satisfy this property: the *count-sketch* matrix from Section 2.3.2. Namely, we will set $\mathbf{S} \in \mathbb{R}^{k \times n}$ to be the linear sketching matrix from Section 2.3.2 such that a streaming algorithm, on stream vector x, stores the sketch $\mathbf{S}x$. Setting the parameters of $\mathbf{S}$ so that there is only one "block" (or repetition) of the hashing scheme, we will obtain a subspace embedding with $k = \Omega(d^2/\epsilon^2)$. This fact is one of the many celebrated connections between the streaming literature and numerical linear algebra.

Theorem 11 ([CW17])**.** *Fix $\mathbf{A} \in \mathbb{R}^{n \times d}$, and let $\mathbf{S} = \mathbb{R}^{k \times n}$ be a matrix such that for each $i \in [n]$ we have $\mathbf{S}_{*,i} = \alpha_i \cdot e_{j_i}^\top$, where $j_i \sim [k]$ is drawn uniformly at random, and $\{\alpha_i\}_{i \in [n]}$ are independent random signs (i.e., drawn uniformly from $\{1, -1\}$). Then, if $k = \tilde{\Omega}(\frac{d^2}{\epsilon^2})$, we have that $\mathbf{S}$ is an ϵ-subspace embedding for $\mathbf{A}$ with probability at least $9/10$.*

Notice that, because each column of $\mathbf{S}$ has only one non-zero entry, the product $\mathbf{SA}$ can be computed in $O(\text{nnz}(\mathbf{A}))$ time, resulting in huge speed-ups for linear regression via the above sketch-and-solve method.

Finally, as hinted earlier, we demonstrate that by sampling rows of $\mathbf{A}$ according to their leverage scores, we obtain a subspace embedding as well. To implement this procedure, one first needs to approximately compute the leverage scores of $\mathbf{A}$; methods for doing this in $\tilde{O}(\text{nnz}(\mathbf{A}))$ time are demonstrated in [CW17].

Theorem 12 (Leverage score sampling is a subspace embedding, see [MI10] or Section 2.4 of [W+14])**.** *Fix any matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\epsilon, \delta \in (0, 1)$. Let $q = (q_1, \ldots, q_n)$ be a distribution over the rows of $\mathbf{A}$ such that there exists a real $\beta > 0$ satisfying $q_i \geq \beta \frac{\tau_i(\mathbf{A})}{\sum_{j=1}^n \tau_j(\mathbf{A})}$ for all $i \in [n]$. Consider a random matrix $\mathbf{S} \in \mathbb{R}^{k \in n}$ generated as follows: initally $\mathbf{S}$ is the zero-matrix, and then for each $i = 1, 2, \ldots, k$, we sample a random $j_i \in [n]$ from the distribution q, and set $\mathbf{S}_{i,j_i} = \frac{1}{\sqrt{q_{j_i} k}}$. Then, assuming $k = \Omega(\frac{1}{\beta \epsilon^2} d \log(k/\delta))$, we have that $\mathbf{S}$ is a ϵ-subspace embedding for $\mathbf{A}$ with probability at least $1 - \delta$.*

Part I

Streaming and Distributed Algorithms

Chapter 3

Perfect L_p Sampling in Data Streams

We begin this thesis by studying the problem of sampling from a stream of data. Given a high-dimensional vector $x \in \mathbb{R}^n$ updated in a stream, and a non-negative *measure function* $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$, the goal of is to sample a coordinate $i \in [n]$ with probability proportional to $G(x_i)$. Of particular interest is the L_p *Distribution*, obtained by setting $G(x) = |x|^p$. Sketches for L_p sampling are the most widely studied sampling algorithms for data streams, and have been used to great success in a multitude of data analytic tasks. Examples include applications in distributed computation [WZ18, JSTW19, BMRT20], graph algorithms [AGM12a, AGM12c, Jin18, CLP18, KKP18, CKP+21, CFCHT20, MV20, KMM+20, FKN21, AD21, FKN21], metric approximation [CJLW20a], numerical linear algebra [LNW19, DJS+19, MRWZ20a], computational social choice [MSW20], data analysis [CPW20], and differential privacy [CGSS20], among many other applications.

Despite their considerable importance, for many decades the only case of L_p sampling which admitted small spaces sketches was when $p = 1$ and the stream contained only positive updates (i.e, the insertion-only model). Here, the well known reservoir sampling scheme of Vitter can be applied [Vit85a]. On the other hand, when negative updates are allowed (i.e., turnstile streams), or if $p \neq 1$, the task becomes significantly more challenging. In fact, the question of whether sublinear space L_p samplers even exist for turnstile streams was posed by Cormode, Murthukrishnan, and Rozenbaum in 2005 [CMR05]. Subsequent work made partial progress on this question, by demonstrating the existence of small space *approximate* L_p samplers [MW10, AKO11, JST11], which sampled a coordinate i with probability proportional to $(1 \pm \epsilon)|x_i|^p$ for some error parameter $\epsilon > 0$.

In this Chapter, we discuss the resolution of this open question of over a decade. In particular,

we describe the first *perfect* L_p samplers in data streams, which have no space dependency on ϵ, i.e, $\epsilon = 0$. Our algorithms are moreover are space optimal for $p < 2$, and optimal up to a $\log n$ factor for $p = 2$. For $p > 2$, the task is known to require polynomial space, thus our result essentially resolves the space complexity of L_p sampling in data streams, and has since been applied to design statistical algorithms with improved accuracy, such as for simulating random walks on graphs [Jin18, CKP+21].

Highlighted Contributions

The materials from this chapter are drawn from our paper [JW18b], and the manuscript [JWZ21]. The main contributions therein are as follows:

- The first perfect L_p sampling for turnstile data streams, $p \in (0, 2]$, using an optimal space for $p < 2$ and matching the best known dependency on n for prior approximate samplers when $p = 2$.(Section 3.3).

- A general derandomization technique for linear sketches, which gives the first efficient derandomization of the count-sketch variant of Minton and Price [MP14], (Section 3.4.2).

- A new framework for sampling from insertion-only streams for general measure functions G, including L_p sampling, concave functions, and a large number of specific measure functions, such as the $L_1 - L_2$, Fair, Huber, and Tukey estimators (Section 3.6).

3.1 Background

A core component of this thesis concerns the problem of generating and analyzing *samples*. As previously mentioned, sampling is an incredibly powerful and versatile method for data analysis, and this is particularly true for the analysis of streaming. Substantial literature has been devoted to the study of sampling for problems in big data [Vit85a, Knu98, GM98a, EV03, M+05, CMR05, GLH06, GLH08, CDK+09, CCD11, CCD12, MM12, CDK+14, Coh15, Haa16, CJ19, CGP20], with applications to network traffic analysis [GKMS01, Duf04, MCS+06, HNG+07, TLJ10], databases [LNS90, Olk93, HS92, LN95, HNSS96, Haa16], distributed computation [CMYZ10, TW11, CMYZ12, WZ16], and low-rank approximation [FKV04, DV06, WZ16].

Sketches for generating samples from data streams are a fundamental primitive, and are uti-

lized as a subroutine in more complicated approximation and statistical tasks. Given a stream vector $x \in \mathbb{R}^n$ and a non-negative *measure function* $G : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, the goal of is to sample a coordinate $i \in [n]$ with probability proportional to $G(x_i)$. The most widely used measure function, and more generally the most widely studied form of sampling in data streams, is to sample from the L_p *Distribution*, obtained by setting $G(x) = |x|^p$, which we now formally introduce.

Definition 3.1.1 (L_p Distribution)**.** *Fix $0 \leq p$, and let $x \in \mathbb{R}^n$ be a non-zero vector. Then the L_p distribution over x, denoted $\mathcal{D}_p(x)$, is a distribution over $\{1, 2, \ldots, n\}$ given by:*

$$\mathop{\mathbf{Pr}}_{X \sim \mathcal{D}_p(x)}[X = i] = \frac{|x_i|^p}{\|x\|_p^p}$$

For $p = 0$, we define $\mathcal{D}_0(x)$ to be the uniform distribution over the support of x.

Ideally, one would like to design a small-space streaming algorithm which, after observing the stream updates $(i_1, \Delta_1), \ldots, (i_m, \Delta_m)$ defining the frequency vector $x \in \mathbb{R}^n$, outputs a random variable $X \sim \mathcal{D}_p(x)$ from the L_p distribution over x. In the special case of the insertion only model and $p = 1$, such an idealistic goal is indeed possible. In fact, the solution for this case has been known for several decades, and is the well-known reservoir sampling algorithm of Vitter [Vit85a], which uses $O(\log n)$ total bits of space to maintain a sample. This algorithm and the analysis are so simple that they bears stating here:

Reservoir Sampling:
Initialize counter $C \leftarrow 0$, and current sample $s \leftarrow$ `null`.
On update (i, Δ):

1. Set $C \leftarrow C + \Delta$
2. With probability $\frac{\Delta}{C}$, set $s \leftarrow i$.

To analyze the algorithm, let $s_t = \sum_{i=1}^{t} \Delta_i$ be the sum of all updates up to time step t. For any $i \in [n]$, the probability that an update (i, Δ_t) causes the current sample to change $s \leftarrow i$ is $\frac{\Delta_t}{s_t}$, and the probability that the sample never changes again thereafter is:

$$\begin{aligned}\left(1 - \frac{\Delta_{t+1}}{s_{t+1}}\right)\left(1 - \frac{\Delta_{t+2}}{s_{t+2}}\right)\cdots\left(1 - \frac{\Delta_m}{s_m}\right) &= \left(\frac{s_{t+1} - \Delta_{t+1}}{s_{t+1}}\right)\left(\frac{s_{t+2} - \Delta_{t+2}}{s_{t+2}}\right)\cdots\left(\frac{s_m - \Delta_m}{s_m}\right) \\ &= \left(\frac{s_t}{s_{t+1}}\right)\left(\frac{s_{t+1}}{s_{t+2}}\right)\cdots\left(\frac{s_{m-1}}{s_m}\right) \\ &= \frac{s_t}{s_m}\end{aligned}$$

Taken together, the probability that the update (i, Δ_t) causes $s = i$ at the end of the stream is $\frac{\Delta_t}{s_t} \cdot \frac{s_t}{s_m} = \frac{\Delta_t}{s_m}$. Summing over all updates to i, we have that $\mathbf{Pr}[s = i] = \sum_{t:i_t=i} \frac{\Delta_t}{s_m} = \frac{x_i}{\|x\|_1}$ as needed.

Unfortunately, when negative updates are allowed in the stream, or $p \neq 1$, the problem becomes significantly harder. In 2005, it was posed as an open question by Cormode, Murthukrishnan, and Rozenbaum whether sublinear space L_p samplers even existed for turnstile streams [CMR05]. Several years later, Monemizadeh and Woodruff made partial progress on this question, by demonstrating that, if we allow the sampler to be *approximately* correct, small space samplers indeed exist [MW10]. We now formally state the guarantee given by an approximate L_p sample. We note that the following definition is stated more generally than the earlier Definition 1.1.2 in Section 1.1.1, as it includes the additional additive error parameter γ, and considers the edge case of zero-valued streams.

Definition 3.1.2 (L_p Sampler)**.** *Fix any $p \geq 0$, values $\epsilon, \delta, \gamma \in (0, 1)$, and let $x \in \mathbb{R}^n$ be a non-zero vector. Then a $(\epsilon, \delta, \gamma)$-approximate L_p sampler is an algorithm which, when run on x, outputs a random variable $Z \in [n] \cup \{\perp, \mathbf{ZERO}\}$ such that $\mathbf{Pr}[Z = \perp] \leq \delta$, $\mathbf{Pr}[Z = \mathbf{ZERO}] \leq \gamma$, and such that for every $i \in [n]$:*

$$\mathbf{Pr}[Z = i \mid Z \notin \{\perp, \mathbf{ZERO}\}] = (1 \pm \epsilon)\frac{|x_i|^p}{\|x\|_p^p} \pm \gamma$$

Moreover, when run on the zero vector $x = \vec{0} \in \mathbb{R}^n$, a $(\epsilon, \delta, \gamma)$-approximate L_p sampler should output $Z = \mathbf{ZERO}$ with probability at least $1 - \gamma$.[1]

We say that an algorithm is a (ϵ, δ)-approximate L_p sampler if, for any constant $c \geq 1$, it is a $(\epsilon, \delta, n^{-c})$-approximate L_p sampler. A $(0, \delta)$-approximate sampler called a perfect L_p sampler, and a $(0, \delta, 0)$-approximate sampler is called a truly perfect L_p sampler.[2]

The one-pass approximate L_p sampler introduced in [MW10] requires $\text{poly}(\epsilon^{-1}, \log n)$ space, albeit with rather large exponents. Later on, in [AKO11], the complexity was reduced significantly to $O(\epsilon^{-p} \log^3 n \log \delta^{-1})$-bits for $p \in [1, 2]$, using a technique known as *precision sampling*. Using a tighter analysis of this technique, Jowhari, Sağlam, and Tardos reduced the space complexity of L_p sampling for $p < 2$ to $O(\epsilon^{-\max\{1,p\}} \log^2 n \log \delta^{-1})$-bits for $p \in (0, 2) \setminus \{1\}$,

[1] We remark that prior works, including our paper [JW18b], did not explicitly consider the case when the input stream x was the zero-vector. The definition shown here captures this case, and the correct behavior of a sampler on such a stream.

[2] We will refer to an algorithm as being a *approximate L_p Sampler* if it is a (ϵ, δ)-approximate sampler for $\epsilon \neq 0$. Otherwise, we will refer to such a sampler as a perfect or truly perfect sampler.

L_p sampling upper bound (bits)	p range	Notes	Citation
$O(\log^3 n)$	$p = 0$	perfect L_0 sampler, $\delta = 1/\text{poly}(n)$	[FIS08]
$O(\log^2 n \log \delta^{-1})$	$p = 0$	perfect L_0 sampler	[JST11]
$\text{poly}\log(\epsilon^{-1}, n)$	$p \in [0, 2]$	$\delta = 1/\text{poly}(n)$	[MW10]
$O(\epsilon^{-p} \log^3 n \log \delta^{-1})$	$p \in [1, 2]$	$(1 \pm \epsilon)$-relative error	[AKO11]
$O(\epsilon^{-\max\{1,p\}} \log^2 n \log \delta^{-1})$	$p \in (0, 2) \setminus \{1\}$	$(1 \pm \epsilon)$-relative error	[JST11]
$O(\epsilon^{-1} \log(\epsilon^{-1}) \log^2 n \log \delta^{-1})$	$p = 1$	$(1 \pm \epsilon)$-relative error	[JST11]
$O(\log^2 n \log \delta^{-1})$	$p \in (0, 2)$	perfect L_p sampler, random oracle model, matches lower bound	*
$O(\log^2 n \log \delta^{-1} (\log \log n)^2)$	$p \in (0, 2)$	perfect L_p sampler	*
$O(\log^3 n \log \delta^{-1})$	$p = 2$	perfect L_2 sampler	*
$O(\log^3 n)$	$p \in (0, 2)$	$\delta = 1/\text{poly}(n)$	*

Figure 3.1: Evolution of one pass L_p sampling upper bounds for $p \in [0, 2]$. The last four results marked with * are our contributions from [JW18b]. Note that the best known lower bound for $p \in [0, 2]$ is $\Omega(\log^2 n \log \delta^{-1})$ for $p \geq 0$ [KNP+17], which holds in the random oracle model (see also [JST11] for a lower bound for constant δ).

and $O(\epsilon^{-1} \log(\epsilon^{-1}) \log^2 n \log \delta^{-1})$ bits of space for $p = 1$ [JST11]. In addition, they give an $O(\log^2 n \log \delta^{-1})$ perfect L_0 sampler. The historical evolution of L_p sampling algorithms is given in Table 3.1.

In addition to upper bounds, in the same paper, Jowhari, Sağlam, and Tardos demonstrated an $\Omega(\log^2 n)$-bit lower bound for L_p samplers for any $p \geq 0$. Recently, this lower bound was extended to $\Omega(\log^2 n \log \delta^{-1})$ [KNP+17] bits for all $\delta \geq 1/\text{poly}(n)$, which closes the complexity of the problem for $p = 0$. Moreover, for $p \in (0, 2)$, this means that the upper and lower bounds for L_p samplers are tight in terms of the parameters n, δ, but a gap exists in the dependency on ϵ. Being the case, it would seem natural to search for an $\Omega(\epsilon^{-p} \log^2 n \log \delta^{-1})$ lower bound to close the complexity of the problem. It is perhaps surprising, therefore, that no lower bound in terms of ϵ exists – not even an $\Omega(\epsilon^{-1})$ bound is known. This raises the following question:

Is the $\Omega(\log^2 n \log \delta^{-1})$ lower bound for L_p sampling tight? Namely, does ϵ need to

3.1.1 Contributions for Perfect Sampling

The main result of this chapter is an answer to the prior question. Specifically, we demonstrate the existence of a *perfect* L_p sampler for $p \in (0, 2]$, which moreover is space optimal for $p < 2$, and matches the prior dependency on n, δ for $p = 2$, but has no dependency on ϵ. Our main theorem is as follows.

Theorem 3 (Perfect L_p sampling). *Fix any $p \in (0, 2]$, and $\delta \in (0, 1)$. Then there is a perfect L_p sampler, meaning a $(0, \delta)$-approximate L_p sampler (Definition 3.1.2), using space $O(\log^2 n \log \delta^{-1})$ for $p < 2$ in the random oracle model, and $O(\log^3 n \log \delta^{-1})$ for $p = 2$ in general. If the random oracle assumption is dropped for $p < 2$, the space changes to*

$$O\left(\min\left\{\log^2 n \log \frac{1}{\delta}(\log\log n)^2, \log^2 n \log \frac{n}{\delta}\right\}\right)$$

Moreover, for any $\nu, \delta_2 \in (0, 1)$, conditioned on outputting a sample $i \in [n]$, the algorithm can additionally output an estimate $\tilde{x}_i$ such that $\tilde{x}_i = (1 \pm \nu)x_i$ with probability at least $1 - \delta_2$, using an additive

$$O\left(\min\left\{\nu^{-2}, \nu^{-p} \log \frac{1}{\delta_2}\right\} \log n \log \frac{1}{\delta_2}\right)$$

bits of space.

Due tor the $\Omega(\log^2 n \log \delta^{-2})$-bit lower bound for approximately L_p sampling even with constant ϵ [KNP+17], which moreover holds in the random oracle model, Theorem 3 closes the space complexity of L_p sampling for $p \in (0, 2)$. We remark that the $\log n$ gap between upper and lower bounds for the case of $p = 2$ was present in prior works which developed approximate L_p samplers [AKO11, JST11]. Specifically, prior to our contribution in Theorem 3, the state of the art complexity for (ϵ, δ)-approximate L_p sampling was $O(\frac{1}{\epsilon^{\max\{1,p\}}} \log^2 n \log \delta^{-1})$ bits of space for $p \in (0, 2) \setminus \{1\}$, $O(\frac{\log \epsilon^{-1}}{\epsilon} \log^2 n \log \delta^{-1})$ bits of space for $p = 1$, and $O(\frac{1}{\epsilon^2} \log^3 n \log \delta^{-1})$ bits of space for $p = 2$ (Theorem 2, [JST11]). It is an outstanding open problem whether this extra $\log n$ factor for the case of $p = 2$ is necessary.

In addition to our upper bound, we also demonstrate a new lower bound for algorithms which, in addition to being an approximate L_p sampler and returning an index $i \in [n]$, also output an estimate of the magnitude of the coordinate x_i with relative error $(1 \pm \nu)$, where $\nu > 0$ is an error parameter. Specifically, in Theorem 27, we demonstrate that any such algorithm

requires $\Omega(\nu^{-p}\log n \log \delta_2^{-1})$ bits of space. We remark that previously, the best upper bounds for estimating a coordinate in addition the sampling was to simply run the previous approximate samplers (Theorem 2) with $\epsilon = \nu$.

A natural complaint about the above sampler is that while it is *perfect* (i.e, $\epsilon = 0$), it is not *truly perfect*, in the sense that it still succumbs to an additive $\gamma = O(n^{-c})$ error. This raises the question of whether it is possible to design truly perfect L_p samplers in the turnstile streaming model, which have no additive error γ in their distribution. While such a sampler would be useful, we demonstrate in Theorem 29 that this is in fact impossible impossible. Specifically, we show the following:

Theorem 29 (Abbreviated) *Fix any constant $\epsilon_0 \in [0, 1)$, let $p > 0$, and let $2^{-n/2} < \gamma < 1/2$. Then any $(\epsilon_0, \frac{1}{2}, \gamma)$-approximate L_p sampler must use $\Omega\left(\min\left\{n, \log \frac{1}{\gamma}\right\}\right)$ bits of space.*

Theorem 29 demonstrates that sublinear space truly perfect samplers, which have no additive error γ, cannot exist in the turnstile model, motivating the need to study perfect samplers.

Generic Derandomization of Linear Sketches. Along the way to derandomizing the main L_p sampling algorithm from Theorem 3, we develop a *generic* technique which allows for the black-box derandomization of a large class of linear sketches. This theorem will not be sufficient alone to derandomize the algorithm of Theorem 3, but is suitable for a number of other applications. For instance, it provides the first efficient derandomization of the count-sketch variant of Minton and Price [MP14], a discussion of which can be found in Section 3.4.2. We state the generic theorem here. In what follows, for a matrix $\mathbf{A}$, let $\texttt{vec}(\mathbf{A})$ denote the vectorization of $\mathbf{A}$, obtained by stacking the columns of $\mathbf{A}$ together.

Theorem 17 (Generic Derandomization of Linear Sketches). *Fix $n, t, k \geq 1$, and fix $x \in \{-M, \ldots, M\}^n$, where $M = \text{poly}(n)$. Let $\mathbf{X} \in \mathbb{R}^{t \times nk}$ be any fixed matrix with entries contained within $\{-M, \ldots, M\}$, and let $\mathcal{D}$ be any distribution over matrices $\mathbf{A} \in \mathbb{R}^{k \times n}$ such that the entries $\mathbf{A} \sim \mathcal{D}$ are i.i.d. and can be sampled using $O(\log n)$-bits.*

Let $\sigma : \mathbb{R}^k \times \mathbb{R}^t \to \{0, 1\}$ be any function, and fix any constant $c \geq 1$. Then there is a distribution $\mathcal{D}'$ over matrices $\mathbf{A}' \in \mathbb{R}^{k \times n}$ such that $\mathbf{A}' \sim \mathcal{D}'$ can be generated via a random seed of length $O((k + t)\log(n)(\log\log n)^2)$, such that:

$$|\mathbf{Pr}\left[\sigma(\mathbf{A}f, \mathbf{X} \cdot \texttt{vec}(\mathbf{A})) = 1\right] - \mathbf{Pr}\left[\sigma(\mathbf{A}'f, \mathbf{X} \cdot \texttt{vec}(\mathbf{A}')) = 1\right]| < n^{-c(k+t)}$$

and such that each entry of $\mathbf{A}'$ can be computed in time $\tilde{O}(1)$ using working space linear in the seed length.

In the above theorem, σ can be defined as a tester with $\sigma(\mathbf{A}x, \mathbf{X} \cdot \texttt{vec}(\mathbf{A})) = 1$ whenever a streaming algorithm depending only on $\mathbf{A}x$ and $\mathbf{X} \cdot \texttt{vec}(\mathbf{A})$ succeeds. Note that the matrix $\mathbf{X}$ allows an algorithm to also depend lightly on $\mathbf{A}$, in addition to the linear sketch $\mathbf{A}x$. For instance, $\mathbf{X} \cdot \texttt{vec}(\mathbf{A})$ could store a entire column of $\mathbf{A}$, as is needed for count-sketch. Thus, Theorem 17 can be seen as a step towards a universal derandomization of streaming algorithms.

We remark that many streaming algorithms, such as the p-stable sketches of Indyk [Ind06] for L_p estimation, depend only on a linear sketch $\mathbf{A}x$. Specifically, for $0 < p \leq 2$, to estimate $\|x\|_p^p = \sum_{i \in [n]} |x_i|^p$ to relative error $(1 \pm \epsilon)$, the algorithm of [Ind06] generates a matrix $\mathbf{A}$ with $O(\epsilon^{-2})$ rows and entries drawn independently from a p-stable distribution. The original derandomization of [Ind06] applied Nisan's pseudo-random generator [Nis92], resulting in a $\log n$ blow-up in the space; namely, the algorithm required $O(\epsilon^{-2} \log^2 n)$ bits of space. This blow-up was shown to be unnecessary by Kane, Nelson, and Woodruff [KNW10a], who gave an involved argument demonstrating that it suffices to generate the entries of $\mathbf{A}$ with limited independence, requiring a total of $O(\epsilon^{-2} \log n)$ bits. Theorem 17, therefore, gives an alternative derandomization of Indyk's p-stable sketches, although it requires a slightly suboptimal $O(\epsilon^{-2} \log n (\log \log n)^2)$ bits of space.

Sampling from General Measures. While Theorem 29 demonstrated that the additive γ error was necessary for sublinear space L_p samplers in the turnstile model, it does not rule out the possibility of truly perfect samplers, with $\gamma = 0$, in the insertion only model. In Section 3.6, we address this discrepancy by demonstrating that sublinear space truly perfect L_p samplers do exist in the insertion only streaming model. Specifically, we develop a framework for truly-perfect sampling in insertion only streams for *general* measure functions G. First, we formalize the appropriate generalization of Definition 3.1.2 to general measure functions.

Definition 3.1.3. *Fix any non-negative function $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ satisfying $G(0) = 0$, and fix values $\epsilon, \delta, \gamma \in (0, 1)$. Let $x \in \mathbb{R}^n$ be a non-zero vector. Then a $(\epsilon, \delta, \gamma)$-approximate G-sampler is an algorithm which, when run on x, outputs a random variable $Z \in [n] \cup \{\perp, \mathbf{ZERO}\}$ such that $\mathbf{Pr}[Z = \perp] \leq \delta$, $\mathbf{Pr}[Z = \mathbf{ZERO}] \leq \gamma$, and such that for every $i \in [n]$:*

$$\mathbf{Pr}[Z = i \mid Z \notin \{\perp, \mathbf{ZERO}\}] = (1 \pm \epsilon) \frac{G(x_i)}{\sum_{j \in [n]} G(x_j)} \pm \gamma$$

Moreover, when run on the zero vector $x = \vec{0} \in \mathbb{R}^n$, a $(\epsilon, \delta, \gamma)$-approximate G-sampler should output $Z = \mathbf{ZERO}$ with probability at least $1 - \gamma$.

Notice that setting $G(x) = |x|^p$, we recover the definition of an approximate L_p sampler.

We inherent the same terminology as in Definition 3.1.2 for (ϵ, δ)-approximate G-samplers, and perfect and truly perfect G-samplers.

Our framework for truly perfect sampling is developed in Section 7.2, wherein several instantiations of the framework are given for specific functions G. Our theorem in its most general form is as follows, although we remark that for several applications, such as for L_p estimation, significant additional work is needed to apply the theorem.

Theorem 22 *Let G be a function such that $G(x) - G(x-1) \leq \zeta$ for all $x \geq 1$. Given a lower bound $\widehat{F_G}$ on $F_G = \sum_{i \in [n]} G(x_i)$, then there exists a truly perfect G sampler for an insertion-only stream that outputs $\perp$ with probability at most δ and uses $O\left(\frac{\zeta m}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space. Further, the time to process each update is $O(1)$ in expectation.*

The main barrier to applying Theorem 22 to any arbitrary measure function G is obtaining a "good" lower bound $\widehat{F_G}$ to $F_G = \sum_{i \in [n]} G(f_i)$. Moreover, this lower bound must be obtained correctly with probability 1, as any possibility of failure of a randomized algorithm would necessarily contribute to additive error to the distribution of the samples, resulting in only a perfect, but not truly perfect, sampler

Interestingly, our samplers are based on a time-stamp based reservoir sampling scheme, as opposed to the common *precision sampling* framework utilized for other L_p samplers [AKO11, JST11, JW18b, JSTW19, CJLW20a]. This property of our samplers makes our framework particularly versatile, and for instance allows it to be applied to the sliding window model of streaming.

As specific applications of our framework, we obtain the first truly perfect samplers for many fundamental sampling problems, including L_p sampling, concave functions, and a large number of measure functions, including the $L_1 - L_2$, Fair, Huber, and Tukey estimators. For $p \geq 1$, our results for L_p sampling are as follows.

Theorem 24 *Fix any $p \in [1, 2]$ and value $\delta \in (0, 1)$. Then there exists a truly perfect L_p-sampling algorithm, namely a $(0, \delta, 0)$-L_p Sampler, which uses $O\left(n^{1-1/p} \log n \log \delta^{-1}\right)$ bits of space.*

The above theorems show a strong separation between turnstile and insertion-only truly perfect L_p samplers; surprisingly, for every $p > 1$, a truly perfect L_p sampler exists with $O\left(n^{1-1/p} \operatorname{polylog}(n)\right)$ bits of memory in the insertion-only model, while in the turnstile model this requires $\Omega(n)$ bits of memory. For $p < 1$, our bounds are as follows

Theorem 25 *Fix $p \in (0, 1]$ and $\delta \in (0, 1)$. Then there exists a truly perfect L_p-sampling algo-*

rithm, namely a $(0, \delta, 0)$*-*L_p *Sampler, which uses* $O\left(m^{1-p} \log n \log \delta^{-1}\right)$ *bits of space.*

Applications of Perfect Samplers

As discussed in Section 1.1.1, approximate L_p samplers can be used as a building block in algorithms for many important streaming problems, such as finding heavy hitters, L_p-norm estimation, cascaded norm estimation, and finding duplicates in data streams [AKO11, MW10, JST11, BOZ12]. L_p samplers, particularly for $p = 1$, are often used as a black-box subroutine to design representative histograms of x on which more complicated algorithms are run [GMP, GM98b, Olk93, GKMS02, HNG+07, CMR05]. For these black-box applications, the only property of the samplers needed is the distribution of their samples. Samplers with relative error are statistically biased and, in the analysis of more complicated algorithms built upon such samplers, this bias and its propagation over multiple samples must be accounted for and bounded. The analysis and development of such algorithms would be simplified dramatically, therefore, with the assumptions that the samples were truly uniform (i.e., from a perfect L_1 sampler). In this case, no error terms or variational distance need be accounted for. Our results show that such an assumption is possible without affecting the space complexity of the sampler.

Another important application of L_p sampling is for the analysis of massive graphs [AGM12a, AGM12c, Jin18, CLP18, KKP18, CKP+21, CFCHT20, MV20, KMM+20, FKN21, AD21, FKN21]. One particularly important example is the problem of simulating random walks on graphs [Jin18, CKP+21], which can be used for connectivity testing [Rei08], clustering [COP03, ACL07, AP09, ST13], sampling [JVV86b] generating random spanning tree [Sch18], and approximate counting [JS89]. An important component of the state of the art random walk simulation algorithms for graph streams [Jin18, CKP+21], is that they use the L_p sampling algorithm of Theorem 3 to obtain walks with small variation distance from a true random walk. Note, importantly, that using a (ϵ, δ)-approximate L_p sampler from prior works [MW10, AKO11, JST11] would result in at least ϵ-variational distance from the true random walk distribution with polynomial dependency in the space on ϵ, whereas usage of a perfect sampler results in only $\log(1/\epsilon)$ dependency in the space to obtain variation distance ϵ [CKP+21].

Another motivation for utilizing perfect L_p samplers comes from applications in privacy. Here $x \in \mathbb{R}^n$ is some underlying dataset, and we would like to reveal a sample $i \in [n]$ drawn from the L_p distribution over x to some external party without revealing too much global information about x itself. Using an approximate L_p sampler introduces a $(1 \pm \epsilon)$ multiplicative bias into the sampling probabilities, and this bias can depend on global properties of the data. For instance,

such a sampler might bias the sampling probabilities of a large set S of coordinates by a $(1+\epsilon)$ factor if a certain global property P holds for x, and may instead bias them by $(1-\epsilon)$ if a disjoint property P' holds. Using only a small number of samples, an adversary would then be able to distinguish whether P or P' holds by determining how these coordinates were biased. On the other hand, the bias in the samples produced by a perfect L_p sampler is polynomially small, and thus the leakage of global information could be substantially smaller when using one.

One formalization of what it means to "leak" global information comes from the literature in *private approximations* [FIM+01, Woo11], where given a non-negative real-valued function $g(x)$ for some input x, the goal is to output a $(1 \pm \epsilon)$ approximation R to $g(x)$, such that the value R reveals no more information about the input x than the actual value of $g(x)$ reveals about x. Specifically, to do this, one must show that the distribution of the value R can be simulated given only knowledge of the value $g(x)$. In our setting, $g(x) = |x_i|^p/\|x\|_p^p$ could be the probability that an exact L_p sampler should output a fixed coordinate $i \in [n]$ on input frequency vector $x \in \mathbb{R}^n$, and the corresponding goal of a private sampler would be to output i with probability $R = (1 \pm \epsilon)g(x)$ without reveling any more information about x than the value $g(x)$. In this setting, a perfect L_p sampler will allow for a simulation which has $1/\mathrm{poly}(n)$ variational distance to the true distribution of R resulting from the sampler (just by adding small $1/\mathrm{poly}(n)$-sized noise), whereas an approximate $(1+\epsilon)$ relative error sampler would only allow for a simulation that is correct up to variational distance ϵ. This results in approximate samplers being substantially less private than perfect samplers.

Roadmap of the Chapter

In Section 3.2, we introduce the *precision sampling framework*, as well as an overview of our techniques in developing the main results of the paper [JW18b]. In Section 3.3, we prove the correctness of the distributional portion of the perfect L_p sampling algorithm of Theorem 3, which is summarized in Theorem 14. In Section 3.4, we provide the space and runtime analysis of the algorithm, including the derandomization and the proof of Theorem 17. In Section 3.5, we prove the second component of Theorem 3, which gives an estimate of the sampled coordinate (Theorem 21). In Section 3.6, we introduce our framework for truly perfect G-samplers, for general measure functions G, in the insertion only model. Finally, in Section 3.7, we prove the lower bound from Theorem 27, and the truly perfect sampling lower bound from Theorem 29.

3.2 Primer on Precision Sampling and Overview

The perfect L_p sampler from Theorem 3 is based on a powerful framework known as the *precision sampling* framework, which we will now discuss in detail. This technique was introduced by Andoni, Krauthgamer, and Onak [AKO11, AKO10] for L_p sampling, among other applications. It was later refined by Jowhari, Sağlam, and Tardos specifically for the purpose of L_p sampling [JST11]. However, the precision sampling technique utilized in [AKO11, JST11] was only capable of developing *approximate* L_p samplers, which had polynomial space dependency on the error parameter ϵ. One of the main technical challenges in the proof of Theorem 3 is to further refine and adapt the precision sampling framework to produce *perfect* L_p samplers.

Precision Sampling At a high level, precision sampling is a technique which reduces the problem of sampling a coordinate x_i from a vector $x \in \mathbb{R}^n$ to the problem of finding the largest coordinate of another vector $z \in \mathbb{R}^n$. Since, in the streaming model, one cannot exactly determine the identity $\arg\max_i |z_i|$ of the largest coordinate in small space, one instead obtains an *estimate* $y \in \mathbb{R}^n$ of the vector z, and sets $i^* = \arg\max_i |y_i|$. Finally, one performs a statistical test to determine whether this estimate was good, in which case the algorithm returns the coordinate $i^* \in [n]$, or not good, in which case the failure symbol $\perp$ is returned. A high level template with these three main steps in the precision sampling framework is given in Figure 3.2.

The first step of transforming x into z in the precision sampling framework consists of scaling each of the coordinates x_i by a random variable α_i as the updates arrive. By setting $z_i = \alpha_i x_i$, we obtain a new stream vector $z \in \mathbb{R}^n$, simply by the transformation $(i, \Delta) \to (i, \alpha_i \Delta)$ to the updates.[3]

In the prior works [AKO11, JST11], the scaling variables were set to $\alpha_i = \frac{1}{t_i^{1/p}}$ where $t_i \sim [0, 1]$ is uniform random variable drawn from $[0, 1]$. Supposing that the algorithm was given a value $T > \|x\|_p$, then one could simply return any $i \in [n]$ with $|z_i| \geq T$. Observe that for a fixed $i \in [n]$, the probability this occurs is given by:

$$\begin{aligned} \mathbf{Pr}\left[|z_i| \geq T\right] &= \mathbf{Pr}\left[u_i < |x_i|^p / T^p\right] \\ &= |x_i|^p / T^p \propto |x_i| \end{aligned} \tag{3.1}$$

It follows that the probability that $|z_i| \geq T$ is proportional to $|x_i|^p$, which is what is desired

[3] In order to avoid the $O(n)$ space required to store the random variables $(\alpha_1, \ldots, \alpha_n)$, the algorithm must generate these random variables with *limited independence*. In prior works [AKO11, JST11], it sufficed to generate these variables from k-wise independent hash functions for small k. This will not be sufficient for the algorithm presented in this chapter, and a more involved derandomization procedure (discussed later) will be required.

Precision Sampling

`Input:` $x \in \mathbb{R}^n$
`Output:` a sampled index $i^* \in [n]$

1. Perform a transformation of the stream vector x into a new vector z.
2. Run instance $\mathbf{A}$ of count-sketch on z to obtain an estimate y of z.
3. Find $i^* = \arg\max_i |y_i|$. Then run a statistical test on y to decide whether to output i^* or $\perp$.

Figure 3.2: Precision Sampling Template

of an L_p sampler. The algorithms of [AKO11, JST11] then attempted to find any such coordinate $i \in [n]$ which passed this statistical test $|z_i| \geq T$, and return that coordinate. These heavy coordinates can be found using a *heavy-hitters* streaming algorithm, which produces the required estimate y of z. Specifically, one can use the *count-sketch* of Charikar, Chen, and Farach-Colton [CCFC02b], which we now introduce.

Primer on Count-Sketch To describe how we handle these technical challenge, we begin by describing the high-level precision sampling template that all L_p sampling algorithms since [AKO11] have adhered to (shown in Figure 3.2). This template employs the classic *count-sketch* algorithm of [CCFC02b] as a subroutine, which is introduced in the thesis preliminaries in Section 2.3.2. For clarity of exposition and completeness, we briefly reintroduce the count-sketch algorithm here.

Given a precision parameter η, count-sketch maintains a data structure $\mathbf{A} \in \mathbb{R}^{d \times k}$, where $d = \Theta(\log n)$ and $k = \Theta(1/\epsilon^2)$. The table is constructed as follows: first, one selects 4-wise independent hash functions $h_j : [n] \to [6/\eta^2]$ and $g_j : [n] \to \{1, -1\}$, for $j = 1, 2, \ldots, d$. Then for all $i \in [d],\ j \in [k]$, it computes the following linear function $\mathbf{A}_{i,j} = \sum_{k \in [n], h_i(k) = j} g_i(k) z_k$, where $z \in \mathbb{R}^n$ is the vector which count sketch is being applied to. Notice that, because the function is linear, count-sketch can be maintained in a stream of updates to z. Given the table $\mathbf{A}$ and the hash functions h_i, g_i, one can then construct an approximation $y \in \mathbb{R}^n$ of the vector x, given by:

$$y_v = \text{median}_{i \in [d]} \{ g_i(k) \mathbf{A}_{i, h_i(k)} \} \tag{3.2}$$

for any $v \in [n]$. The estimation guarantee of count-sketch is as follows:

Theorem 9 (Count-Sketch). *Fix $\eta > 0$, and let $d = \Theta(\log n)$ and $k = \Theta(1/\epsilon^2)$. Then if $y \in \mathbb{R}^n$*

is defined as in Equation 3.2, we have

$$\|y - z\|_\infty < \eta \|z_{\text{tail}(1/\eta^2)}\|_2$$

with probability $1 - n^{-c}$.

In the above, recall that for a vector x and real $t \geq 1$, the vector $x_{\text{tail}(t)}$ is the result of setting equal to zero the t largest coordinates (in magnitude) of x.

Prior Algorithms. In the following discussion we consider the algorithm of [JST11] (which refined several components of the sampler in [AKO11]). Recall that in this algorithm, the vector z is defined via $z_i = x_i / t_i^{1/p}$ where $t_i \sim [0, 1]$. Firstly, to obtain the threshold $T > \|x\|_p$, one can run an L_p estimation algorithm, such as the streaming algorithm of Indyk [Ind06], to obtain an estimate $R \in [\|x\|_p, 2\|x\|_p]$ with high probability. The sampler then sets the threshold to be $T = \epsilon^{-1} R$. By scaling up the threshold by a factor of ϵ^{-1}, this minimizes the chance that *two* or more coordinates i exceed the threshold T, which would contribute error to the distribution of the sampler.[4] The only remaining task of the algorithm is to find a coordinate $i \in [n]$ that crosses the threshold $|z_i| > T$. By using the count-sketch data structure, it obtains an estimate y of z satisfying $\|y - z\|_\infty < \eta \|x_{\text{tail}(1/\eta^2)}\|_2$ for some parameter η. It then finds the maximum coordinate $i^* = \arg\max_i |y_i|$, checks if $|y_{i^*}| > T$. If this is not the case, the algorithm returns $\perp$.

Now the algorithm runs into trouble when it incorrectly identifies z_{i^*} as crossing the threshold when it has not, or fails to identify any coordinate which crosses the threshold when one exists. However, if the count-sketch error $\eta \|z_{\text{tail}(1/\eta^2)}\|_2$ is at most $O(\|x\|_p)$, then since t_i is a uniform variable the probability that z_{i^*} is close enough to the threshold to be misidentified is $O(\epsilon)$, which results in at most $(1 \pm \epsilon)$ relative error in the sampling probabilities. Thus, it will suffice to prove that the event $\eta \|z_{\text{tail}(1/\eta^2)}\|_2 = O(\|x\|_p)$ occurs with probability $1 - \epsilon$.

To show that this is the case, consider the level sets

$$I_k = \left\{ z_i \;\middle|\; \frac{\|x\|_p}{2^{(k+1)/p}} \leq z_i \leq \frac{\|x\|_p}{2^{k/p}} \right\}$$

One can show that $\mathbf{E}[|I_k|] = 2^k$. Now for $p < 2$, the total contribution $\Theta\left(\frac{\|x\|_p^2}{2^{2k/p}} |I_k|\right)$ of the level sets to $\|z\|_2^2$ decreases geometrically with k, and so with constant probability we have $\|z\|_2 = O(\|x\|_p)$. Moreover, if one removes the top $\log(1/\epsilon)$ largest items, the contribution of

[4] Specifically, the probability that this occurs is at most ϵ, which is one of several sources of relative error in the approximate sampler of [JST11].

the remaining items to the L_2 is $O(\|x\|_p)$ with probability $1-\epsilon$. So taking $\eta = \sqrt{\log(1/\epsilon)}$, the tail error $\|z_{\text{tail}(1/\eta^2)}\|_2$ has the desired size. For $p = 2$, as we will later discuss, one can only show that $\|z_{\text{tail}(1/\eta^2)}\|_2 = O(\log n\|x\|_p)$, which will require one to take $\eta = \sqrt{\log n}$.

Now with probability ϵ the guarantee on the error from the prior paragraph does not hold, and in this case one *cannot* still output an index i, as this would result in a ϵ-*additive* error sampler. Thus, as in Step 3 of Figure 3.2, the algorithm must implement a statistical test to check that the guarantee holds. To do this, using the values of the largest $1/\eta^2$ coordinates of y, they produce an estimate of the tail-error and output $\perp$ if it is too large. Otherwise, the item $i^* = \arg\max_i |y_i|$ is output if $|y_{i^*}| > T$. The whole algorithm is repeated $O(\epsilon^{-1}\log\delta^{-1})$ times independently, so that an index is output with probability at least $1-\delta$.

Our Algorithm. Our first observation is that, in order to obtain a truly perfect sampler, one needs to use different scaling variables t_i. Notice that the approach of scaling by inverse uniform variables and returning a coordinate which reaches a certain threshold T faces the obvious issue of what to return when more than one of the variables $|z_i|$ crosses T. This is solved by simply outputting the maximum of all such coordinates. However, the probability of an index becoming the maximum *and* reaching a threshold is drawn from an entirely different distribution, and for uniform variables t_i this distribution does not appear to be the correct one.

To overcome this, we must use a distribution where the maximum index i of the variables $(|x_1 t_2^{-1/p}|, |x_2 t_2^{-1/p}|, \ldots, |x_n t_n^{-1/p}|)$ is drawn *exactly* according to the L_p distribution $|x_i|^p/\|x\|_p^p$. We observe that the distribution of exponential random variables has precisely this property, and thus to implement Step 1 of Figure 3.2 we set $z_i = x_i/t_i^{1/p}$ where t_i is an exponential random variable. We remark that exponential variables have been used in the past, such as for x_p moment estimation, $p > 2$, in [AKO11] and regression in [WZ13]. However it appears that their applicability to sampling has never before been exploited.

Next, we similarly run an instance $\mathbf{A}$ of count-sketch on the vector z. Because we are only interested in the maximizer of z, we develop a modified version of count-sketch, called *count-max*. Instead of producing an estimate y such that $\|y-z\|_\infty$ is small, count-max simply checks, for each $i \in [n]$, how many times z_i hashed into the largest bucket (in absolute value) of a row of $\mathbf{A}$. If this number is at least a $4/5$-fraction of the total number of rows, count-max declares that z_i is the maximizer of z. We show that with high probability, count-max never incorrectly declares an item to be the maximizer, and moreover if $z_i > 20(\sum_{j\neq i} z_j^2)^{1/2}$, then count-max will declare i to be the maximizer. Using the *min-stability* property of exponential random variables, we can show that the maximum item $|z_{i^*}| = \max\{|z_i|\}$ is distributed as $\|x\|_p/E^{1/p}$, where E is another

exponential random variable. Thus $|z_{i^*}| = \Omega(\|x\|_p)$ with constant probability. Using a more general analysis of the L_2 norm of the level sets I_k, we can show that $(\sum_{j \neq i^*} z_j^2)^{1/2} = O(\|x\|_p)$. If all these events occur together (with sufficiently large constants), count-max will correctly determine the coordinate $i^* = \arg\max_i\{|z_i|\}$. However, just as in [JST11], we cannot output an index anyway if these conditions do not hold, so we will need to run a statistical test to ensure that they do.

The Statistical Test. To implement Step 3 of the template, our algorithm tests whether count-max declares any coordinate $i \in [n]$ to be the maximizer, and we output $\perp$ if no coordinate is declared as the maximizer. This approach guarantees that we correctly output the maximizer conditioned on not failing to output a coordinate (i.e., outputting $\perp$) The primary technical challenge will be to show that, conditioned on $i = \arg\max_j\{|z_j|\}$ for some i, the probability of failing the statistical test *does not depend on* i. In other words, conditioning on $|z_i|$ being the maximum does not change the failure probability. Let $z_{D(k)}$ be the k-th order statistic of z (i.e., $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$). Here the $D(k)$'s are known as *anti-ranks* of the distribution $(z_1, \ldots, z_n)$. To analyze the conditional dependence, we must first obtain a closed form for $z_{D(k)}$ which separates the dependencies on k and $D(k)$. Hypothetically, if $z_{D(k)}$ depended only on k, then our statistical test would be completely independent of $D(1)$, in which case we could safely fail whenever such an event occurred.

Of course, in reality this is not the case. Consider the vector $x = (100n, 1, 1, 1, \ldots, 1) \in \mathbb{R}^n$ and $p = 1$. Clearly we expect z_1 to be the maximizer, and moreover we expect a gap of $\Theta(n)$ between z_1 and $z_{D(2)}$. On the other hand, if you were told that $D(1) \neq 1$, it is tempting to think that $z_{D(1)}$ just *barely* beat out z_1 for its spot as the max, and so z_1 would not be far behind. Indeed, this intuition would be correct, and one can show that the probability that $z_{D(1)} - z_{D(2)} > n$ conditioned on $D(1) = i$ changes by an additive constant depending on whether or not $i = 1$. Conditioned on this gap being smaller or larger, we are more or less likely (respectively) to output $\perp$. In this setting, the probability of conditional failure can change by an $\Omega(1)$ factor depending on the value of $D(1)$.

To handle scenarios of this form, our algorithm will utilize an additional linear transformation in Step 1 of the template. Instead of only scaling by the random coefficients $1/t_i^{1/p}$, our algorithm first *duplicates* the coordinates x_i to remove all heavy items from the stream. If x is the vector from the example above and F is the duplicated vector, then after $\mathrm{poly}(n)$ duplications all copies of the heavy item x_1 will have weight at most $|x_1|/\|F\|_1 < 1/\mathrm{poly}(n)$. By duplication, this washes out the dependency of $|z_{D(2)}|$ on $D(1)$, since $\|x_{-D(1)}\|_p^p = (1 \pm n^{-\Omega(c)})\|x_{-j}\|_p^p$ after n^c duplications, for any $j \in [n^c]$. Notice that this transformation blows-up the dimension of x by

a $\mathrm{poly}(n)$ factor. However, since our space usage is always $\mathrm{poly}\log(n)$, the result is only a constant factor increase in the complexity.

After duplication, we scale x by the coefficients $1/t_i^{1/p}$, and the rest of the algorithm proceeds as described above. Using expressions for the order statistics $z_{D(k)}$ which separate the dependence into the anti-ranks $D(j)$ and a set of exponentials $E_1, E_2, \ldots E_n$ *independent* of the anti-ranks, after duplication we can derive tight concentration of the $z_{D(k)}$'s conditioned on fixed values of the E_i's. Using this concentration result, we decompose our count-max data structure $\mathbf{A}$ into two component variables: one independent of the anti-ranks (the independent component), and a small adversarial noise of relative weight n^{-c}. In order to bound the effect of the adversarial noise on the outcome of our tests we must $\mathbf{1}$) randomize the threshold for our failure condition and $\mathbf{2}$) demonstrate the anti-concentration of the resulting distribution over the independent components of $\mathbf{A}$. This will demonstrate that with high probability, the result of the statistical test is completely determined by the value of the independent component, which allows us to fail without affecting the conditional probability of outputting $i \in [n]$.

Derandomization The correctness of our sampler crucially relies on the full independence of the t_i's to show that the variable $D(1)$ is drawn from precisely the correct distribution (namely, the L_p distribution $|x_i|^p/\|x\|_p^p$). Being the case, we cannot directly implement our algorithm using any method of limited independence. In order to derandomize the algorithm from requiring full-independence, we will use a combination of Nisan's pseudorandom generator (PRG) [Nis92], as well as an extension of the recent PRG of [GKM18] which fools certain classes of *Fourier transforms*. We first use a closer analysis of the seed length Nisan's generator requires to fool the randomness required for the count-max data structure, which avoids the standard $O(\log n)$-space blowup which would be incurred by using Nisan's as a black box. Once the count-max has been derandomized, we demonstrate how the PRG of [GKM18] can be used to fool *arbitrary* functions of d halfspaces, so long as each of the half-spaces can be specified by a normal vector with bounded bit-complexity. Specifically, we require that each coordinate v_i of the normal vector $v \in \mathbb{R}^m$ that specifies an m-dimensional halfspace has bounded bit complexity; for our application, each coordinate v_i is specified using $O(\log n)$ bits. We use this result to derandomize the exponential variables t_i with a seed of length $O(\log^2(n)(\log\log n)^2)$, which will allow for the total derandomization of our algorithm for $\delta = \Theta(1)$ and $p < 2$ in the same space.

Our derandomization technique is in fact fairly general, and can be applied to streaming algorithms beyond the sampler in this work. Namely, we demonstrate that *any* streaming algorithm which stores a linear sketch $\mathbf{A} \cdot x$, where the entries of $\mathbf{A}$ are independent and can be sam-

pled from with $O(\log n)$-bits, can be derandomized with only a $O((\log\log n)^2)$-factor increase in the space requirements (see Theorem 17). This improves the $O(\log n)$-blow up incurred from black-box usage of Nisan's PRG.

As an application of our technique, we derandomize the count-sketch variant of Minton and Price [MP14] to use $O(\epsilon^{-2}\log^2 n(\log\log n)^2)$-bits of space, which gives improved concentration results for count-sketch when the hash functions are fully-independent. The problem of improving the derandomization of [MP14] beyond the black-box application of Nisan's PRG was an open problem. We remark that using $O(\epsilon^{-2}\log^3 n)$-bits of space in the classic count sketch of [CCFC02b] has strictly better error guarantees that those obtained from derandomizing [MP14] with Nisan's PRG to run in the same space. Our derandomization, in contrast, demonstrates a strong improvement on this, obtaining the same bounds with an $O((\log\log n)^2)$ instead of an $O(\log n)$ factor blowup.

Case of $p = 2$. Recall for $p < 2$, we could show that the L_2 norm of the level sets I_k decays geometrically with k. More precisely, for any $\gamma \geq 1$ we have $\|z_{\text{tail}(\gamma)}\|_2 = O(\|F\|_p\gamma^{-1/p+1/2})$ with probability $1 - O(e^{-\gamma})$, where now z is a scaling of the duplicated vector F. Using this, we actually do not need the tight concentration of the $z_{D(k)}$'s, since we can show that the top $n^{c/10}$ coordinates change by at most $(1\pm n^{-\Omega(c)})$ depending on $D(1)$, and the L_2 norm of the remaining coordinates is only an $O(n^{-c/10(1/p-1/2)})$ fraction of the whole L_2, and can thus be absorbed into the adversarial noise. For $p = 2$ however, each level set I_k contributes weight $O(\|F\|_p^2)$ to $\|z\|_2^2$, so $\|z_{\text{tail}(\gamma)}\|_2 = O(\sqrt{\log n}\|F\|_p)$ even for $\gamma = \text{poly}(n)$. Therefore, for $p = 2$ it is essential that we show concentration of the $z_{D(k)}$'s for *nearly all* k. Since $\|z\|_2^2$ will now be larger than $\|F\|_2^2$ by a factor of $\log n$ with high probability, count-max will only succeed in outputting the largest coordinate when it is an $O(\sqrt{\log n})$ factor larger than expected. This event occurs with probability $1/\log n$, so we will need to run the algorithm $\log n$ times in parallel to get constant probability, for a total $O(\log^3 n)$-bits of space. Using the same $O(\log^3 n)$-bit Nisan PRG seed for all $O(\log n)$ repititions, we show that the entire algorithm for $p = 2$ can be derandomized to run in $O(\log^3 n \log 1/\delta)$-bits of space.

Optimizing the Runtime. In addition to our core sampling algorithm, we show how the linear transformation step to construct z can be implemented via a parameterized rounding scheme to improve the update time of the algorithm without affecting the space complexity, giving a run-time/relative sampling error trade-off. By rounding the scaling variables $1/t_i^{1/p}$ to powers of $(1+\epsilon)$, we discretize their support to have size $O(\epsilon^{-1}\log n)$. We then simulate the update procedure by sampling from the distribution over updates to our count-max data-structure $\mathbf{A}$ of

duplicating an update and hashing each duplicate independently into $\mathbf{A}$. Our simulation utilizes results on efficient generation of binomial random variables, through which we can iteratively reconstruct the updates to $\mathbf{A}$ bin-by-bin instead of duplicate-by-duplicate. In addition, by using an auxiliary heavy-hitter data structure, we can improve our query time from the naïve $O(n)$ to $O(\text{poly}\log n)$ without increasing the space complexity.

Estimating the Frequency. We show that allowing an additional additive $O(\min\{\epsilon^{-2}, \epsilon^{-p}\log(\frac{1}{\delta_2})\}\log n \log \delta_2^{-1})$ bits of space, we can provide an estimate $\tilde{x} = (1 \pm \epsilon)x_i$ of the outputted frequency x_i with probability $1 - \delta_2$ when $p < 2$. To achieve this, we use our more general analysis of the contribution of the level sets I_k to $\|z\|_2$, and give concentration bounds on the tail error when the top ϵ^{-p} items are removed. When $p = 2$, for similar reasons as described in the sampling algorithm, we require another $O(\log n)$ factor in the space complexity to obtain a $(1 \pm \epsilon)$ estimate. Finally, we demonstrate an $\Omega(\epsilon^{-p}\log n \log \delta_2^{-1})$ lower bound for this problem, which is nearly tight when $p < 2$. To do so, we adapt a communication problem introduced in [JW13], known as *Augmented-Indexing on Large Domains*. We weaken the problem so that it need only succeed with constant probability, and then show that the same lower bound still holds. Using a reduction from this problem, we show that our lower bound for L_p samplers holds even if the output index is from a distribution with constant *additive* error from the true L_p distribution $|x_i|^p/\|x\|_p^p$.

3.2.1 The Count-Max Data Structure

We now describe the count-max data structure, which is a slight variant of the count-sketch algorithm of [CCFC02b]. The count-sketch algorithm is formally described in the thesis preliminaries in Section 2.3.2, thus we direct the reader to this section for the definition of count-sketch and the relevant notation. Recall that, in the precision sampling framework, we are only interested in determining the index of the largest coordinate of x, that is $i^* = \arg\max_{v\in[n]} |x_v|$. This motivates the design of a simpler reporting algorithm based on the same count-sketch data structure $\mathbf{A}$ that tests whether a fixed $v \in [n]$, if $v = i^*$. Specifically, given a count-sketch table $\mathbf{A} \in \mathbb{R}^{d\times k}$, we will define a reporting produced $\text{Report}(\mathbf{A})$, and returns either a coordinate $i \in [n]$ or a symbol $\perp$.

We will make several small modifications to the construction of the table $\mathbf{A}$ and the reporting procedure of count-sketch as defined in Section 2.3.2. First, instead of having the hash functions g_i's map to $\{-1, 1\}$, we we instead draw them independently from the Gaussian distribution, namely $g_i(v) \sim \mathcal{N}(0, 1)$ for all $v \in [n]$ and $i \in [d]$. Thus, we can think of g_i as functions

$g_i : [n] \to \mathbb{R}$. An entry of count-max is given by:

$$\mathbf{A}_{i,j} = \sum_{v \in [n] \,:\, h_i(v) = j} g_i(v) \cdot x_v \tag{3.3}$$

for any $i \in [d]$ and $j \in [k]$. Notice that Equation 3.3 is the same as Equation 2.1 for count-sketch, however the $g_i(v)'s$ are different functions in the case of count-max.

The second fundamental change we make is to the reporting procedure. Firstly, before reporting a maximizer, we will we will scale each bucket $\mathbf{A}_{i,j}$ of count-max by a uniform random variable $\mu_{i,j} \sim \texttt{Uniform}(\frac{99}{100}, \frac{101}{100})$. This scaling step will be used for technical reasons in our analysis of Lemma 3.3.3. Namely, we will need it to ensure that our failure threshold of our algorithm is randomized, which will allow us to handle small adversarial error.

Formally, for all $(i, r) \in [d] \times [k]$ we draw i.i.d. variables $\mu_{i,j} \sim \texttt{Uniform}(\frac{99}{100}, \frac{101}{100})$. Next, for any $v \in [n]$, we set

$$\alpha_v = \left| \{ i \in [d] \mid h_i(v) = \arg\max_{r \in [k]} |\mu_{i,r} \cdot \mathbf{A}_{i,r}| \} \right|$$

Our goal will be to declare that $v = i^*$ to be the maximizer if $\alpha_v > \frac{4}{5}d$. Specifically, the algorithm computes α_v for all $v \in [n]$, and outputs $\text{Report}(\mathbf{A}) = v$, where $v \in [n]$ is the first index that satisfies $\alpha_v > \frac{4}{5}d$. We will show that there will only be at most one such coordinate with high probability. If no such v exists, the algorithm sets $\text{Report}(\mathbf{A}) = \perp$. To distinguish this modified querying protocol from the classic count-sketch, we refer to this algorithm as count-max. To refer to the data structure $\mathbf{A}$ itself, we will use the terms count-sketch and count-max interchangeably.

Lemma 3.2.1. *Let $c \geq 1$ be an arbitrarily large constant, set $d = \Theta(\log n)$ and $k = 2$, and let $\mathbf{A}$ be a $d \times k$ instance of count-max run on input $x \in \mathbb{R}^n$ using fully independent hash functions $h_i : [n] \to [k]$, Gaussian random variables $g_i(v) \sim \mathcal{N}(0, 1)$, and scalings $\mu_{i,j} \sim \texttt{Uniform}(\frac{99}{100}, \frac{101}{100})$ as described above. Then with probability $1 - n^{-c}$ the following holds:*

- *For every $i \in [n]$, if $|x_i| > 20\|x_{-i}\|_2$, then $\text{Report}(\mathbf{A}) = i$.*
- *If $|x_i| \leq \max_{j \in [n] \setminus \{i\}} |x_j|$, then $\text{Report}(\mathbf{A}) \neq i$.*

Proof. First suppose $|x_i| > 20\|x_{-i}\|_2$, and consider a fixed row j of $\mathbf{A}$. WLOG i hashes to $\mathbf{A}_{j,1}$, thus using the 2-stability of Gaussians (Definition 2.2.6), we have $\mathbf{A}_{j,1} = \mu_{j,1} g^1 \|x^1\|_2$ and $\mathbf{A}_{j,2} = \mu_{j,2} g^2 \|x^2\|_2$, where x^k is x restricted to the coordinates that hash to bucket $\mathbf{A}_{j,k}$, and $g^1, g^2 \sim \mathcal{N}(0, 1)$. Since $\|x^1\|_2 > 20|x^2\|_2$ and $\mu_{i,j} \sim \texttt{Uniform}(\frac{99}{100}, \frac{101}{100})$, the probability that

$|\mathbf{A}_{j,2}| > |\mathbf{A}_{j,1}|$ is less than probability that one $\mathcal{N}(0,1)$ Gaussian is 19 times larger than another, which can be bounded by $15/100$ by direct computation. Thus i hashes into the max bucket in a row of $\mathbf{A}$ with probability at least $85/100$, so by Chernoff bounds, taking $d = \Omega(c \log n)$, with probability $1 - n^{-2c}$ we have that x_i is in the largest bucket at least a $4/5$ fraction of the time, which completes the first claim.

Now suppose i is not a unique max, and let i^* be such that $|x_{i^*}|$ is maximal. Then conditioned on i, i^* not hashing to the same bucket, the probability that x_i hashes to a larger bucket than x_{i^*} is at most $1/2$. To see this, note that conditioned on this, one bucket is distributed as $\mu(g_j(i^*)x_{i^*} + G)$ and the other as $\mu'(g_j(i)x_i + G')$, where G, G' μ, μ', and $g_j(i^*)x_{i^*}, g_j(i)x_i$ are identically distributed random variables. Thus the probability that x_i is the in maximal bucket is at most $3/4$, and so by Chernoff bounds x_i will hash to strictly less than $(4d/5)$ of the maximal buckets with probability $1 - n^{-2c}$. Union bounding over all $j \in [n]$ gives the desired result. □

Corollary 3.2.2. *In the setting of Lemma 3.2.1, with probability* $1 - O(n^{-c})$*, if* $\text{Report}(\mathbf{A}) = i$ *for some* $i \in [n]$*, then* $|x_i| > \frac{1}{100}\|x\|_2$.

Proof. Suppose $|x_i| < \frac{1}{100}\|x\|_2$, and in a given row WLOG i hashes to $\mathbf{A}_{j,1}$. Let x_{-i} be x with the coordinate i set equal to 0. Then we have $\mathbf{A}_{j,1} = \mu_{j,1} g^1 \|x^1\|_2$ and $\mathbf{A}_{j,2} = \mu_{j,2} g^2 \|x^2\|_2$, where x^k is x restricted to the coordinates that hash to bucket $\mathbf{A}_{j,k}$, and $g^1, g^2 \sim \mathcal{N}(0,1)$. Since x^1_{-i}, and $x^2_{-i} = x^2$ are identically distributed, with probability $1/2$ we have $\|x^2_{-i}\|_2 \geq \|x^1_{-i}\|_2$. Conditioned on this, we have

$$\begin{aligned} \|x^2_{-i}\|_2^2 &\geq \|x^1\|_2^2 - |x_i^2|^2 \\ &\geq \|x^1\|_2^2 - \frac{\|x^2_{-i}\|_2^2}{50} \end{aligned} \tag{3.4}$$

Giving

$$\|x^2_{-i}\|_2 (1 + 1/50)^{1/2} \geq \|x^1\|_2^2$$

So conditioned on $\|x^2_{-i}\|_2 > \|x^1_{-i}\|_2$, we have $|\mathbf{A}_{j,1}| < |\mathbf{A}_{j,2}|$ whenever one Gaussian is $\frac{101}{100}(1 + \frac{1}{50})^{1/2}$ times larger than another in magnitude, which occurs with probability greater than $\frac{1}{2} - \frac{1}{25}$. So i hashes into the max bucket with probability at most $\frac{1}{2} + \frac{1}{2}(\frac{1}{2} + \frac{1}{25}) = \frac{77}{100}$, and thus by Chernoff bounds, taking c sufficiently large and union bounding over all $i \in [n]$, i will hash into the max bucket at most a $\frac{79}{100} < \frac{4}{5}$ fraction of the time with probability $1 - O(n^{-c})$, and thus will not be reported. □

3.2.2 Exponential Order Statistics

In this section, we discuss several useful properties of the order statistics of n independent non-identically distributed exponential random variables. Let $(t_1, \ldots, t_n)$ be independent exponential random variables where t_i has mean $1/\lambda_i$ (equivalently, t_i has rate λ_i). Recall that t_i is given by the cumulative distribution function $\mathbf{Pr}[t_i < x] = 1 - e^{-\lambda_i x}$. Our perfect L_p sampling algorithm will require a careful analysis of the distribution of values $(t_1, \ldots, t_n)$, which we will now describe. We begin by noting that constant factor scalings of an exponential variable result in another exponential variable.

Fact 3.2.3 (Scaling of exponentials)**.** *Let t be exponentially distributed with rate λ, and let $\alpha > 0$. Then αt is exponentially distributed with rate λ/α*

Proof. The cdf of αt is given by $\mathbf{Pr}[t < x/\alpha] = 1 - e^{-\lambda x/\alpha}$, which is the cdf of an exponential with rate λ/α. □

We would now like to study the order statistics of the variables $(t_1, \ldots, t_n)$, where t_i has rate λ_i. To do so, we introduce the *anti-rank vector* $(D(1), D(2), \ldots, D(n))$, where for $k \in [n]$, $D(k) \in [n]$ is a random variable which gives the index of the k-th smallest exponential.

Definition 3.2.4. *Let $(t_1, \ldots, t_n)$ be independent exponentials. For $k = 1, 2, \ldots, n$, we define the k-th anti-rank $D(k) \in [n]$ of $(t_1, \ldots, t_n)$ to be the values $D(k)$ such that $t_{D(1)} \leq t_{D(2)} \leq \cdots \leq t_{D(n)}$.*

Using the structure of the anti-rank vector, it has been observed [Nag06] that there is a simple form for describing the distribution of $t_{D(k)}$ as a function of $(\lambda_1, \ldots, \lambda_n)$ and the anti-rank vector.

Fact 3.2.5 ([Nag06])**.** *Let $(t_1, \ldots, t_n)$ be independently distributed exponentials, where t_i has rate $\lambda_i > 0$. Then for any $k = 1, 2, \ldots, n$, we have*

$$t_{D(k)} = \sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{n} \lambda_{D(j)}}$$

Where the $E_1, E_2, \ldots, E_n$'s are i.i.d. exponential variables with mean 1, and are independent of the anti-rank vector $(D(1), D(2), \ldots, D(n))$.

Fact 3.2.6 ([Nag06]). *For any $i = 1, 2, \ldots, n$, we have*

$$\mathbf{Pr}\left[D(1) = i\right] = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

We now describe how these properties will be useful to our sampler. Let $x \in \mathbb{R}^n$ be any vector presented in a turnstile stream (note that x must have integral valued coordinates by definition of the model). We can generate i.i.d. exponentials $(t_1, \ldots, t_n)$, each with rate 1, and construct the random variable $z_i = x_i / t_i^{1/p}$, which can be obtained in a stream by scaling updates to x_i by $1/t_i^{1/p}$ as they arrive. By Fact 3.2.3, the variable $|z_i|^{-p} = t_i / |x_i|^p$ is exponentially distributed with rate $\lambda_i = |x_i|^p$.

Now let $(D(1), \ldots, D(n))$ be the anti-rank vector of the exponentials $(t_1/|x_1|^p, \ldots, t_n/|x_n|^{1/p})$. By Fact 3.2.6, we have

$$\begin{aligned} \mathbf{Pr}\left[D(1) = i\right] &= \mathbf{Pr}\left[i = \arg\min\{|z_1|^{-p}, \ldots, |z_n|^{-p}\}\right] \\ &= \mathbf{Pr}\left[i = \arg\max\{|z_1|, \ldots, |z_n|\}\right] \\ &= \frac{\lambda_i}{\sum_j \lambda_j} = \frac{|x_i|^p}{\|f\|_p^p} \end{aligned} \tag{3.5}$$

In other words, the probability that $|z_i| = \arg\max_j\{|z_j|\}$ is precisely $|x_i|^p / \|x\|_p^p$, so for a perfect L_p sampler it suffices to return $i \in [n]$ with $|z_i|$ maximum. Now note $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$, and in this scenario the statement of Fact 3.2.5 becomes:

$$z_{D(k)} = \Big(\sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{N} \lambda_{D(j)}}\Big)^{-1/p} = \Big(\sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{N} x_{D(j)}^p}\Big)^{-1/p}$$

Where E_i's are i.i.d. exponential random variables with mean 1, and are independent of the anti-rank vector $(D(1), \ldots, D(n))$. We call the exponentials E_i the *hidden exponentials*, as they do not appear in the actual execution of the algorithm, and will be needed for analysis purposes only.

3.3 The Sampling Algorithm

We now provide intuition for the workings of the perfect L_p sampling algorithm. The algorithm scales the input stream by inverse exponentials to obtain a new vector z. Namely, we scale x_i

by $1/t_i^{1/p}$ to obtain z_i. Let $D = (D(1), \ldots, D(n))$ be the indices such that $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(n)}|$. Note that D is precisely the anti-ranks of the variables $(t_1, \ldots, t_n)$ where t_i is exponential with rate $|x_i|^p$, because we have $|z_i|^p = 1/t_i$ and $t_{D(1)} \leq t_{D(2)} \leq \cdots \leq t_{D(n)}$. Due to this correspondence, we will also refer to D as the anti-rank vector of z.

By the results of Section 3.2.2, we can write the order statistics $z_{D(k)}$ as a function of the anti-rank vector D and the hidden exponentials E_i, which describe the "scale" of the order statistics. Importantly, the hidden exponentials are independent of the anti-ranks. We would like to determine the index i for which $D(1) = i$, however this may not always be possible. This is the case when the largest element $|z_{D(1)}|$ is not sufficently larger than the remainig L_2 mass $\sum_{j>1} \left(|z_{D(j)}|^2\right)^{1/2}$. In such a case, count-max will not declare any index to be the largest, and we would therefore like to output the failure symbol $\perp$. Note that this event is more likely when there is another element $|z_{D(2)}|$ which is very close to $|z_{D(1)}|$ in size, as whenever the two elements do not collide in count-max, it is less likely that $|z_{D(1)}|$ will be in the max bucket.

Now consider the trivial situation where $x_1 = x_2 = \cdots = x_n$. Here the variables $z_{D(k)}$ have no dependence at all on the anti-rank vector D. In this case, the condition of failing is independent of $D(1)$, so we can safely fail whenever we cannot determine the maximum index. On the other hand, if the values $|x_i|$ vary wildly, the variables $z_{D(k)}$ will depend highly on the anti-ranks. In fact, if there exists x_i with $|x_i|^p \geq \epsilon \|x\|_p^p$, then the probability that $|z_{D(1)}| - |z_{D(2)}|$ is above a certain threshold can change by a $(1 \pm \epsilon)$ factor conditioned on $D(1) = i$, as opposed to $D(1) = j$ for a smaller $|x_j|$. Given this, the probability that we fail can change by a multiplicative $(1 \pm \epsilon)$ conditioned on $D(1) = i$ as opposed to $D(1) = j$. In this case, we cannot output $\perp$ when count-max does not report a maximizer, lest we suffer a $(1 \pm \epsilon)$ error in outputting an index with the correct probability.

To handle this, we must remove the heavy items from the stream to weaken the dependence of the values $z_{D(k)}$ on the anti-ranks, which we carry out by duplication of coordinates. For the purposes of efficiency, we carry out the duplication via a rounding scheme which will allow us to generate and quickly hash updates into our data-structures (Section 3.4). We will show that, conditioned on the fixed values of the E_i's, the variables $z_{D(k)}$ are highly concentrated, and therefore nearly independent of the anti-ranks ($z_{D(k)}$ depends only on k and not $D(k)$). By randomizing the failure threshold to be anti-concentrated, the small adversarial dependence of $z_{D(k)}$ on $D(k)$ cannot non-trivially affect the conditional probabilities of failure, leading to small relative error in the resulting output distribution.

The L_p Sampler. We now describe our sampling algorithm, $\mathrm{L_p}$-Sampler, as shown in Figure 3.3. Let $x \in \mathbb{R}^n$ be the input vector of the stream. As the stream arrives, we duplicate updates to each coordinate x_i a total of n^{c-1} times to obtain a new vector $F \in \mathbb{R}^{n^c}$. More precisely, for $i \in [n]$ we set $i_j = (i-1)n^{c-1} + j$ for $j = 1, 2, \ldots, n^{c-1}$, and then we will have $F_{i_j} = x_i$ for all $i \in [n]$ and $j \in [n^{c-1}]$. We call the coordinate F_{i_j} a duplicate of x_i. Whenever we use i_j as a subscript in this way it will refer to a duplicate of i, whereas a single subscript i will be used both to index into $[n]$ and $[n^c]$. Note that this duplication has the effect that $|F_i|^p \leq n^{-c+1}\|F\|_p^p$ for all $p > 0$ and $i \in [n^c]$.

We then generate independent exponential rate 1 random variables $(t_1, \ldots, t_{n^c})$, and define the vector $z \in \mathbb{R}^{n^c}$ by $z_i = F_i / t_i^{1/p}$. As shown in Section 3.2.2, we have

$$\mathbf{Pr}\left[i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}\right] = \frac{|F_{i_j}|^p}{\|F\|_p^p}$$

Since $\sum_{j \in [n^{c-1}]} \frac{|F_{i_j}|^p}{\|F\|_p^p} = \frac{|x_i|^p}{\|x\|_p^p}$, it will therefore suffice to find $i_j \in [n^c]$ for which $i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}$, and return the index $i \in [n]$. The assumption that the t_i's are independent will later be relaxed in Section 3.4 while derandomizing the algorithm. In Section 3.4, we also demonstrate that all relevant continuous distributions will be made discrete without affecting the perfect sampling guarantee.

Now fix any sufficiently large constant c, and fix some error parameter $\epsilon > n^{-c}$. To speed up the update time, instead of explicitly scaling F_i by $1/t_i^{1/p}$ to construct the stream z, our algorithm instead scales F_i by $\mathtt{rnd}_\epsilon(1/t_i^{1/p})$, where $\mathtt{rnd}_\epsilon(x)$ rounds $x > 0$ down to the nearest value in $\{\ldots, (1+\epsilon)^{-1}, 1, (1+\epsilon), (1+\epsilon)^2, \ldots\}$. In other words, $\mathtt{rnd}_\epsilon(x)$ rounds x down to the nearest power of $(1+\epsilon)^j$ (for $j \in \mathbb{Z}$). This results in a separate stream vector $\zeta \in \mathbb{R}^{n^c}$ where $\zeta_i = F_i \cdot \mathtt{rnd}_\epsilon(1/t_i^{1/p})$. Note $\zeta_i = (1 \pm O(\epsilon))z_i$ for all $i \in [n^c]$. Importantly, note that this rounding is order preserving. Thus, if ζ has a unique largest coordinate $|\zeta_{i^*}|$, then $|z_{i^*}|$ will be the unique largest coordinate of z.

We remark that this rounding procedure will introduce $(1 \pm \epsilon)$ relative error into the sampler, resulting in a $(\tilde{O}(\epsilon), \delta)$-approximate L_p sampler. However, as we will see, ϵ will not factor into the spatial complexity of the sampler (so long as $\epsilon > n^{-c}$ for some constant c), thus by setting $\epsilon = 1/\mathrm{poly}(n)$ and absorbing the $(1 \pm 1/\mathrm{poly}(n))$ relative error into the $\gamma = 1/\mathrm{poly}(n)$ additive error allow for in a perfect sampler, the result will be a perfect L_p sampler.

Having constructed the transformed stream ζ, we then instatiate a $d \times 2$ instance $\mathbf{A} \in \mathbb{R}^{d \times 2}$ of count-max (from Section 3.2.1), with $d = \Theta(\log(n))$, and run it on the input vector $\zeta \in \mathbb{R}^{n^c}$.Now

L_p-Sampler
Input: Error parameter $\epsilon \in (0,1)$. 1. Set $d = \Theta(\log n)$, and instantiate a $d \times 2$ count-max data structure $\mathbf{A}$ (Section 3.2.1). 2. Duplicate updates to x to obtain the vector $F \in \mathbb{R}^{n^c}$ so that $x_i = F_{i_j}$ for all $i \in [n]$ and $j = 1, 2, \ldots, n^{c-1}$, for some fixed constant c. 3. Choose i.i.d. exponential random variables $t = (t_1, t_2, \ldots, t_{n^c})$, and define the vector $\zeta \in \mathbb{R}^{n^c}$ by $\zeta_i = F_i \cdot \mathtt{rnd}_\epsilon(1/t_i^{1/p})$. 4. Run count-max $\mathbf{A}$ on the scaled stream vector ζ, and compute the output $\mathrm{Report}(\mathbf{A})$. 5. If $\mathrm{Report}(\mathbf{A}) = i_j \in [n^c]$ for some $j \in [n^{c-1}]$, output $i \in [n]$. If $\mathrm{Report}(\mathbf{A}) = \perp$, then return $\perp$.

Figure 3.3: Our main L_p sampling algorithm

recall that count-max will output as the maximizer $\mathrm{Report}(\mathbf{A}) = i_j$ for some index $i_j \in [n^c]$, or output the symbol $\mathrm{Report}(\mathbf{A}) = \perp$. If an index i_j is returned, where i_j is the j-th copy of index $i \in [n]$, then our overall sampling algorithm outputs the index i. If count-max outputs $\mathrm{Report}(\mathbf{A}) = \perp$, we return $\perp$.

Let $i^* = \arg\max_i |\zeta_i| = D(1)$ (where $D(1)$ is the first anti-rank as in Section 3.2.2). By the guarantee of Lemma 3.2.1, we know that if $|\zeta_{i^*}| \geq 20\|\zeta_{-i^*}\|_2$, then with probability $1 - n^{-c}$ count-max will report the index $i^* \in [n^c]$. Moreover, with the same probability, count-max will never report an index which is not the unique maximizer. To prove correctness, therefore, it suffices to analyze the conditional probability of failure given $D(1) = i$. Let $N = |\{i \in [n^c] \mid F_i \neq 0\}|$; in other words, N is the support size of F. We can assume that $N \neq 0$ (to check this one could run, for instance, the $O(\log^2 n)$-bit support sampler of [JST11]). Note that $n^{c-1} \leq N \leq n^c$. The following fact is straightforward.

Fact 3.3.1. *For $p \in (0,2]$, suppose that we choose the constant c such that $mM \leq n^{c/20}$, where note we have $|F_i| \leq mM$ for all $i \in [N]$. Then if $S \subset \{i \in [n^c] \mid F_i \neq 0\}$ is any subset, then $\sum_{i \in S} |F_i|^p \geq \frac{|S|}{N} n^{-c/10} \|F\|_p^p$*

Proof. We know that $|F_i|^p \leq (mM)^p \leq n^{c/10}$ using $p \leq 2$. Then each non-zero value $|F_i|^p$ is at most an $n^{-c/10}$ fraction of any other item $|F_j|^p$, and in particular of the average item weight. It follows that $|F_i|^p \geq n^{-c/10} \frac{\|F\|_p^p}{N}$ for all $i \in [N]$, which results in the stated fact. □

As in Section 3.2.2, we now use the anti-rank vector $D(k)$ to denote the index of the k-th largest value of z_i in absolute value. In other words, $D(k)$ is the index such that $|z_{D(k)}|$ is the k-th largest value in the set $\{|z_1|, |z_2|, \ldots, |z_{n^c}|\}$. Note that the $D(k)$'s are also the anti-ranks of the vector ζ, since rounding z into ζ preserves partial ordering. For the following lemma, it suffices to consider only the exponentials t_i with $F_i \neq 0$, and we thus consider only values of k between 1 and N. Thus $|z_{D(1)}| \geq |z_{D(2)}| \geq \cdots \geq |z_{D(N)}|$. Moreover, we have that $|z_{D(k)}|^{-p} = \frac{t_{D(k)}}{|F_{D(k)}|^p}$ is the k-th smallest of all the $\frac{t_i}{|F_i|^p}$'s, and by the results of Section 3.2.2 can be written as $|z_{D(k)}|^{-p} = \sum_{\tau=1}^{k} \frac{E_\tau}{\sum_{j=\tau}^{N} |F_{D(j)}|^p}$ where the E_τ are i.i.d. exponentials and independent of the anti-rank vector D. We will make use of this in the following lemma.

Lemma 3.3.2. *For every* $1 \leq k < N - n^{9c/10}$*, we have*

$$|z_{D(k)}| = \left[(1 \pm O(n^{-c/10})) \sum_{\tau=1}^{k} \frac{E_\tau}{\mathbf{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right]} \right]^{-1/p}$$

with probability $1 - O(e^{-n^{c/3}})$.

Proof. Let $\tau < N - n^{9c/10}$. We can write $\sum_{j=\tau}^{N} |F_{D(j)}|^p$ as a deterministic function $\psi(t_1, \ldots, t_N)$ of the random scaling exponentials $t_1, \ldots, t_N$ corresponding to $F_i \neq 0$. We first argue that

$$|\psi(t_1, \ldots, t_N) - \psi(t_1 \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_N)| < 2 \max_j \{F_j^p\} < 2n^{-c+1} \|F\|_p^p$$

This can be seen from the fact that changing a value of t_i can only have the effect of adding (or removing) $|F_i|^p$ to the sum $\sum_{j=\tau}^{N} |F_{D(j)}|^p$ and removing (or adding) a different $|F_l|$ from the sum. The resulting change in the sum is at most $2 \max_j \{|F_j|^p\}$, which is at most $2n^{-c+1}\|F\|_p^p$ by duplication. Set $T = N - \tau + 1$. Since the t_i's are independent, we apply McDiarmid's inequality (Proposition 2.2.3) to obtain that for any $\lambda > 0$:

$$\begin{aligned} \mathbf{Pr}\left[\left| \sum_{j=\tau}^{N} |F_{D(j)}|^p - \mathbf{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right] \right| > \lambda T n^{-c} \|F\|_p^p \right] &\leq 2\exp\left(\frac{-2\lambda^2 T^2 n^{-2c} \|F\|_p^{2p}}{n^c (2n^{-c+1}\|F\|_p^p)^2} \right) \\ &\leq 2\exp\left(-\frac{1}{2}\lambda^2 T^2 n^{-c-2} \right) \end{aligned} \tag{3.6}$$

Setting $\lambda = \Theta(n^{-c/5})$ and using $T > n^{9c/10}$, this is at most $2\exp(-\frac{1}{2}n^{2c/5-2})$. To show concentration up to a $(1 \pm O(n^{-c/10}))$ factor, it remains to show that $\mathbf{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right] =$

$\Omega(Tn^{-11c/10}\|F\|_p^p)$. This follows from the Fact 3.3.1, which gives

$$\sum_{j=0}^{T} |F_{D(-j)}|^p \geq n^{-c/10}(Tn^{-c}\|F\|_p^p)$$

Now recall $|z_{D(k)}| = [\sum_{\tau=1}^{k} \frac{E_\tau}{\sum_{j=\tau}^{N} |F_{D(-j)}|^p}]^{-1/p}$, and we have just shown $\sum_{j=\tau}^{N} |F_{D(j)}|^p = (1 \pm O(n^{-c/10}))\,\mathbf{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right]$, so we can union bound over all $\tau = 1, 2, \ldots, N - n^{9c/10}$ to obtain

$$|z_{D(k)}| = \left[(1 \pm O(n^{-c/10})) \sum_{\tau=1}^{k} \frac{E_\tau}{\mathbf{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right]}\right]^{-1/p}$$

for all $k \leq N - n^{9c/10}$ with probability $1 - O(n^c e^{-n^{2c/5-2}}) = 1 - O(e^{n^{c/3}})$.

□

We use this result to show that the event where our algorithm outputs the failure symbol $\perp$ is nearly-independent of the random variable $D(1)$. In what follows, let $\mathcal{E}_1$ be the event that Lemma 3.3.2 holds.

Lemma 3.3.3. *For $p \in (0, 2]$ a constant bounded away from 0 and any $\epsilon \geq n^{-c/60}$ and any index $\ell \in [n^c]$, we have:*

$$\mathbf{Pr}\left[\text{Report}(\mathbf{A}) \neq \perp \mid D(1) = \ell\right] = \mathbf{Pr}\left[\text{Report}(\mathbf{A}) \neq \perp\right] \pm \tilde{O}(\epsilon)$$

Proof. Let

$$U_{D(k)} = \left(\sum_{\tau=1}^{k} \frac{E_\tau}{\mathbf{E}\left[\sum_{j=\tau}^{N} |F_{D(j)}|^p\right]}\right)^{-1}$$

and note that $U_{D(k)}$ is independent of the anti-rank vector D; in fact, it is totally determined by k and the hidden exponentials E_i). Then by Lemma 3.3.2, conditioned on $\mathcal{E}_1$, for every $k < N - n^{9c/10}$ we have

$$|z_{D(k)}| = U_{D(k)}^{1/p}(1 \pm O(n^{-c/10}))^{1/p} = U_{D(k)}^{1/p}(1 \pm O(\frac{1}{p}n^{-c/10}))$$

where we used the identity $(1 + x) \leq e^x$ and the Taylor expansion of e^x. Then for c sufficiently large, we have $|\zeta_{D(k)}| = U_{D(k)}^{1/p}(1 \pm O(\epsilon))$, and so for all $p \in (0, 2]$ and $k < N - n^{9c/10}$, we have:

$$|\zeta_{D(k)}| = U_{D(k)}^{1/p} + U_{D(k)}^{1/p} V_{D(k)}$$

where $V_{D(k)}$ is some random variable that satisfies $|V_{D(k)}| = O(\epsilon)$.

Now consider a bucket $\mathbf{A}_{i,j}$ for some $(i,j) \in [d] \times [2]$. Let $\sigma_k = \texttt{sign}(z_k) = \texttt{sign}(\zeta_k)$ for $k \in [n^c]$. We can write

$$\frac{\mathbf{A}_{i,j}}{\mu_{i,j}} = \sum_{k \in B_{ij}} \sigma_{D(k)} \cdot |\zeta_{D(k)}| \cdot g_i(D(k)) + \sum_{k \in S_{ij}} \sigma_{D(k)} \cdot |\zeta_{D(k)}| \cdot g_i(D(k))$$

where

$$B_{ij} = \{k \leq N - n^{9c/10} \mid h_i(D(k)) = j\}, \quad \text{and} \quad S_{ij} = \{n^c \geq k > N - n^{9c/10} \mid h_i(D(k)) = j\}$$

Note that here we define $\{D(N+1), \ldots, D(n^c)\}$ to be the set of indices i with $F_i = 0$ (in any ordering, as they contribute nothing to the sum). Also recall that $g_i(D(k)) \sim \mathcal{N}(0,1)$ is the i.i.d. Gaussian coefficent associated to item $D(k)$ in row i of $\mathbf{A}$. So

$$\frac{\mathbf{A}_{i,j}}{\mu_{i,j}} = \sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} + \sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} V_{D(k)} + \sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)}$$

Importantly, observe that since the variables $h_i(D(k))$ are fully independent, the sets $B_{i,j}, S_{i,j}$ are independent of the anti-rank vector D. In other words, the values $h_i(D(k))$ are independent of the values $D(k)$ (and of the entire anti-rank vector). This follows from the fact that the hash values $\{h_i(1), \ldots, h_i(n^c)\} = \{h_i(D(1)), \ldots, h_i(D(n^c))\}$ are fully independent.[5] So we can condition on a fixed set of values $\{h_i(D(1)), \ldots, h_i(D(n^c))\}$ now, which fixes the sets $B_{i,j}, S_{i,j}$. Now let $U_{i,j}^* = |\sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p}|$. We claim the following:

Claim 3.3.4. *For all $i,j \in [d] \times [2]$ and $p \in (0,2]$, we have*

$$\left| \sum_{k \in B_{ij}} g_i(D(k)) \sigma_{D(k)} U_{D(k)}^{1/p} V_{D(k)} + \sum_{k \in S_{ij}} g_i(D(k)) \zeta_{D(k)} \right| = O\left(\epsilon(|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|)\right)$$

with probability $1 - O(\log(n) n^{-c/60})$.

[5] Note that this would not necessarily be the case if $\{h_i(1), \ldots, h_i(n^c)\}$ were only ℓ-wise independent for some $\ell = o(n^c)$.

Proof. By the 2-stability of Gaussians (Definition 2.2.6), we have

$$\left|\sum_{k\in S_{ij}} g_i(D(k))\zeta_{D(k)}\right| = O(\sqrt{\log(n)}(\sum_{k\in S_{i,j}} (2z_{D(k)})^2)^{1/2})$$

with probability at least $1 - n^{-c}$. This is a sum over a subset of the $n^{9c/10}$ smallest items $|z_i|$, so

$$\sum_{k\in S_{i,j}} z_{D(k)}^2 < \frac{n^{9c/10}}{N}\|z\|_2^2$$

giving

$$\left|\sum_{k\in S_{ij}} g_i(D(k))\zeta_{D(k)}\right| = O\left(\sqrt{\log(n)}n^{-c/30}\|z\|_2\right)$$

Now without loss of generality, the value $\mathbf{A}_{i,1}$ satisfies $\sum_{k\in B_{i,1}\cup S_{i,1}} \zeta_{D(k)}^2 > \frac{1}{2}\|\zeta\|_2^2$. Then we have $|\mathbf{A}_{i,1}| \geq |g|\|z\|_2^2/3$, where $g \sim \mathcal{N}(0,1)$. Via the cdf of a Gaussian, it follows with probability $1 - O(n^{-c/60})$ that

$$|\mathbf{A}_{i,1}| > n^{-c/60}\|z\|_2^2 = \Omega\left(\frac{n^{c/60}}{\sqrt{\log(n)}} \cdot \left|\sum_{k\in S_{ij}} g_i(D(k))\zeta_{D(k)}\right|\right)$$

Rescaling the parameter ϵ down by a $\log n$ factor yields $|\sum_{k\in S_{ij}} g_i(D(k))\zeta_{D(k)}| = O(\epsilon|\mathbf{A}_{i,1}|)$. Next, using that $|V_{D(k)}| = O(\epsilon)$, we have

$$\left|\sum_{k\in B_{ij}} g_i(D(k))\sigma_{D(k)}U_{D(k)}^{1/p}V_{D(k)}\right| = O\left(\epsilon \cdot \left|\sum_{k\in B_{ij}} g_i(D(k))\sigma_{D(k)}U_{D(k)}^{1/p}\right|\right) = O(\epsilon U_{i,j}^*)$$

Combined with the prior paragraph, we have $U_{i,j}^* = O(|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|)$. Note there are only $O(\log n)$ terms i, j to union bound over, and from which the claim follows. □

Call the event where the Claim 3.3.4 holds $\mathcal{E}_2$. Conditioned on $\mathcal{E}_2$, we can decompose $|\mathbf{A}_{i,j}|/\mu_{i,j}$ for all i, j into $U_{i,j}^* + \mathcal{V}_{ij}$ where $\mathcal{V}_{ij}$ is some random variable satisfying $|\mathcal{V}_{ij}| = O(\epsilon(|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|))$ and $U_{i,j}^*$ is independent of the anti-rank vector D (it depends only on the hidden exponentials E_k, and the uniformly random gaussians $g_i(D(k))$). Now fix any realization of the count-max randomness, Let $E = (E_1, \ldots, E_N)$ be the hidden exponential vector, $\mu = \{\mu_{i,1}, \mu_{i,2}\}_{i\in[d]}$, $D = (D(1), D(2), \ldots, D(N))$, and let $\neg\perp$ be shorthand for the event $\text{Report}(\mathbf{A}) \neq \perp$. Now observe that for any given value of $D(1) \in [n^c]$ that we could condition

on, we have

$$\mathbf{Pr}\left[\neg\perp \mid D(1)\right] = \sum_{E,\mu} \mathbf{Pr}\left[\neg\perp \mid D(1), E, \mu\right] \cdot \mathbf{Pr}\left[E, \mu\right]$$

Here we have used the fact that E, μ are independent of the anti-ranks D. Thus, it will suffice to bound the probability of obtaining E, μ such that the event of failure can be determined by the realization of D. So consider any row i, and consider the event $\mathcal{Q}_i$ that $|\mu_{i,1}U^*_{i,1} - \mu_{i,2}U^*_{i,2}| < 2(|\mathcal{V}^*_{i,1}| + |\mathcal{V}^*_{i,2}|) = O(\epsilon(|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|)$ (where here we have conditioned on the high probability event $\mathcal{E}_2$). WLOG, $U^*_{i,1} \geq U^*_{i,2}$, giving $U^*_{i,1} = \Theta(|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|)$. Since the $\mu_{i,j}$'s are uniform, we have

$$\mathbf{Pr}\left[\mathcal{Q}_i\right] = O\left(\epsilon \cdot \frac{|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|}{U^*_{i,1}}\right) = O(\epsilon)$$

and by a union bound $\mathbf{Pr}\left[\cup_{i\in[d]}\mathcal{Q}_i\right] = O(\epsilon \log n)$. Thus conditioned on $\mathcal{E}_1 \cap \mathcal{E}_2$ and $\neg(\cup_{i\in[d]}\mathcal{Q}_i)$, the event of failure is completely determined by the values E, μ, and in particular is independent of the anti-rank vector D. Thus

$$\mathbf{Pr}\left[\neg\perp | D(1), E, \mu, \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\right] = \mathbf{Pr}\left[\neg\perp | E, \mu, \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\right]$$

So averaging over all E, μ:

$$\begin{aligned}
\mathbf{Pr}\left[\neg\perp | D(1)\right] &= \mathbf{Pr}\left[\neg\perp | D(1), \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\right] + O(\log(n)\epsilon) \\
&= \mathbf{Pr}\left[\neg\perp | \neg(\cup_{i\in[d]}\mathcal{Q}_i), \mathcal{E}_1 \cap \mathcal{E}_2\right] + O(\log(n)\epsilon) \\
&= \mathbf{Pr}\left[\neg\perp\right] + O(\log(n)\epsilon)
\end{aligned}$$

as needed.

□

In Lemma 3.3.3, we demonstrated that the probability of failure can only change by an additive $\tilde{O}(\epsilon)$ term given that any one value of $i \in [N]$ achieved the maximum (i.e., $D(1) = i$). This property will translate into a $(1 \pm \tilde{O}(\epsilon))$-*relative error* in our sampler, where the space complexity is independent of ϵ. One can then set $\epsilon = 1/\mathrm{poly}(n)$ to obtain a perfect L_p sampler.

To complete the proof of correctness of our algorithm, we now need to bound the probability that we fail at all. To do so, we first prove the following fact about $\|z_{\text{tail(s)}}\|_2$, or the L_2 norm of z with the top s largest (in absolute value) elements removed.

Proposition 3.3.5. *For any $s = 2^j \leq n^{c-2}$ for some $j \in \mathbb{N}$, then with probability $1 - 3e^{-s}$, we*

have

$$\sum_{i=4s}^{N} z_{D(i)}^2 = O\left(\frac{\|F\|_p^2}{s^{2/p-1}}\right)$$

if $p \in (0,2)$ *is a constant bounded below* 2*, and if* $p = 2$ *we have*

$$\sum_{i=4s}^{N} z_{(i)}^2 = O\left(\log(n) \cdot \|F\|_p^2\right)$$

Proof. Let $I_k = \{i \in [N] \mid z_i \in \left(\frac{\|F\|_p}{2^{(k+1)/p}}, \frac{\|F\|_p}{2^{k/p}}\right)\}$ for $k = 0, 1, \ldots, p\log(\|F\|_p)$, where we have $\log(\|F\|_p^p) = O(\log(n))$. Note that

$$\mathbf{Pr}\left[i \in I_k\right] = \mathbf{Pr}\left[t_i \in (\frac{2^k F_i^p}{\|F\|_p^p}, \frac{2^{k+1}F_i^p}{\|F\|_p^p})\right] < \frac{2^k F_i^p}{\|F\|_p^p}$$

where we used fact that the pdf e^{-x} of the exponential distribution is upper bounded by 1, and then integrating the pdf over a interval of size at most $\frac{2^k F_i^p}{\|F\|_p^p}$. Thus $\mathbf{E}\left[|I_k|\right] < 2^k$, so for every $k \geq \log(s) = j$, we have $\mathbf{Pr}\left[|I_k| > 4(2^k)\right] < e^{-s2^{k-j}}$ by Chernoff bounds. By a union bound, the probability that $|I_k| > 4(2^k)$ for any $k \geq \log(s)$ is at most $e^{-s}\sum_{i=0}^{O(\log(n))} e^{2^i} \leq 2e^{-s}$.

Now observe $\mathbf{Pr}\left[z_i > \|F\|_p/s^{1/p}\right] < \frac{sF_i^p}{\|F\|_p^p}$, so $\mathbf{E}\left[|\{i|z_i > \|F\|_p/s^{1/p}\}|\right] < s$, and as before the number of such i with $z_i > \|F\|_p/s^{1/p}$ is at most $4s$ with probability $1 - e^{-s}$. Conditioning on this, $\sum_{i=4s}^{N} z_{(i)}^2$ does not include the weight of any of these items, so

$$\sum_{i=4s}^{N} z_{(i)}^2 \leq \sum_{k=\log(s)}^{O(\log(n))} |I_k| \left(\frac{\|F\|_p}{2^{k/p}}\right)^2 \leq 4 \sum_{k=0}^{O(\log(n))} \frac{\|F\|_p^2}{2^{(\log(s)+k)(2/p-1)}}$$

First, if $p < 2$, the above sum is geometric and converges to at most $4\frac{\|F\|_p^2}{1-2^{-2/p+1}}\frac{1}{s^{2/p-1}} = O(\|F\|_p^2/s^{2/p-1})$ for p a constant bounded below by 2. If $p = 2$ or is arbitrarily close to 2, then each term is at most $\|F\|_p^2$, and the sum is upper bounded by $O(\log(n)\|F\|_p^2)$ as stated. Altogether, the probability of failure is at most $1 - 3e^{-s}$ by a union bound. □

Lemma 3.3.6. *For* $0 < p < 2$ *a constant bounded away from* 0 *and* 2*, the probability that* L_p `Sampler` *outputs* $\perp$ *is at most* $1 - \Omega(1)$*, and for* $p = 2$ *is* $1 - \Omega(1/\log(n))$*.*

Proof. By Proposition 3.3.5, with probability $1 - 3e^{-4} > .9$ we have $\|z_{\text{tail}(16)}\|_2 = O(|F\|_p)$ for $p < 2$, and $\|z_{\text{tail}(16)}\| = O(\sqrt{\log(n)}\|F\|_p)$ when $p = 2$. Observe that for $t = 2, 3, \ldots, 16$, we have $|z_{D(t)}| < \|F\|_p(\frac{2}{\sum_{\tau=1}^{t} E_\tau})^{1/p}$, and with probability $99/100$ we have $E_t > 1/100$, which implies that $|z_{D(t)}| = O(\|F\|_p)$ for all $t \in [16]$. Conditioned on this, we have $\|z_{\text{tail}(2)}\|_2 < q\|F\|_p$

where q is a constant when $p < 2$, and $q = \Theta(\sqrt{\log(n)})$ when $p = 2$. Now $|z_{D(1)}| = \frac{\|F\|_p}{E_1^{1/p}}$, and using the fact that the pdf exponential random variables around 0 is bounded above by a constant, we will have $|z_{D(1)}| > 20\|z_{-D(1)}\|_2$ with probability $\Omega(1)$ when $p < 2$, and probability $\Omega(\frac{1}{\log(n)})$ when $p = 2$. Conditioned on this, by Lemma 3.2.1, count-max will return the index $D(1)$ with probability $1 - n^{-c}$, and thus the Sampling algorithm will not fail. □

Putting together the results of this section, we obtain the correctness of our algorithm as stated in Theorem 13. In Section 3.4, we will show that the algorithm can be implemented to have $\tilde{O}(\epsilon^{-1})$ update and $\tilde{O}(1)$ query time, and that the entire algorithm can be derandomized to use $O(\log^2 n(\log\log n)^2)$ bits of space for $p \in (0, 2)$ and $O(\log^3(n))$ bits for $p = 2$.

Theorem 13. *Fix any constant $c \geq 2$, and values $\epsilon \geq n^{-c}$, $\delta \in (0, 1)$, and $0 < p \leq 2$. Then there is a one-pass $(\tilde{O}(\epsilon), \delta, O(n^{-c}))$-approximate L_p sampler. The space used is:*

- $O(\log^2 n \log \delta^{-1}(\log\log n)^2)$ *bits for $p < 2$.*
- $O(\log^3 n)$ *bits when $p < 2$ and $\delta = 1/\mathrm{poly}(n)$.*
- $O(\log^3 n \log \delta^{-1})$ *bits for $p = 2$*

The update time is $\tilde{O}(\epsilon^{-1})$, and the query time is $\tilde{O}(1)$.

Proof. First note that if $x = \vec{0}$, we can estimate $\|x\|_1$ to error $(1 \pm 1/2)$ with probability $1 - n^{-c}$ using space $O(\log^2 n)$ via the norm estimation algorithm of [KNW10a]. This allows us to determine whether or not $x = \vec{0}$ correctly with probability $1 - n^{-c}$, and we can output **ZERO** whenever the algorithm gives an estimate of 0 for $\|x\|_1$. This satisfies the distributional assumptions on outputting **ZERO** from Definition 3.1.2. Thus, in what follows, we assume $x \neq \vec{0}$.

Conditioned on not reporting $\perp$, by Lemma 3.2.1, with probability $1 - n^{-c}$ we have that the coordinate $i_j \in [n^c]$ reported of count-max will in fact be equal to $\arg\max_i\{|\zeta_i|\}$. Recall that $\zeta_i = (1 \pm O(\epsilon))z_i$ for all $i \in [n^c]$ (and this rounding of z to ζ is order preserving). By Lemma 3.2.1 count-max only outputs a coordinate which is the *unique* maximizer of ζ. Now if there was *unique* maximizer of ζ, there must also be a unique maximizer in z, from which it follows that $i_j = \arg\max_i\{|z_i|\}$.

Lemma 3.3.3 states for any $i_j \in [n^c]$ that

$$\mathbf{Pr}\left[\neg\perp \mid i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}\right] = \mathbf{Pr}\left[\neg\perp\right] \pm \tilde{O}(\epsilon) = q \pm \tilde{O}(\epsilon)$$

where $q = \mathbf{Pr}[\neg\perp] = \Omega(1)$ for $p < 2$, and $q = \Omega(\frac{1}{\log(n)})$ for $p = 2$, both of which follow from Lemma 3.3.6, which does not depend on any of the randomness in the algorithm. Recall that $\neg\perp$ be shorthand for the event $\mathrm{Report}(\mathbf{A}) \neq \perp$. Since conditioned on $\neg\perp$, the coordinate i_j reported by count-max satisfies $i_j = \arg\max_i\{|z_i|\}$, the probability we output $i_j \in [n^c]$ is $\mathbf{Pr}[\neg\perp \cap i_j = \arg\max\{|z_i|\}]$, so the probability our final algorithm outputs $i \in [n]$ is

$$\sum_{j\in[n^{c-1}]} \mathbf{Pr}\left[\neg\perp \mid i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}\right]\mathbf{Pr}\left[i_j = \arg\max_{i',j'}\{|z_{i'_{j'}}|\}\right] = \sum_{j\in[n^{c-1}]} \frac{|x_i|^p}{\|F\|_p^p}(q \pm \tilde{O}(\epsilon))$$

$$= \frac{|x_i|^p}{\|x\|_p^p}(q \pm \tilde{O}(\epsilon))$$

Note that we can scale the c value used in the algorithm by a factor of 60, so that the statement of Lemma 3.3.3 holds for any $\epsilon \geq n^{-c}$. The potential of the failure of the various high probability events that we conditioned on only adds another additive $O(n^{-c})$ term to the error. Thus, conditioned on an index i being returned, we have $\mathbf{Pr}[i = j] = \frac{|x_j|^p}{\|f\|_p^p}(1 \pm \tilde{O}(\epsilon))) \pm n^{-c}$ for all $j \in [n]$, which is the desired result after scaling ϵ by a $\mathrm{poly}(\log(n))$ term. Running the algorithm $O(\log(\delta^{-1}))$ times in parallel for $p < 2$ and $O(\log(n)\log(\delta^{-1}))$ for $p = 2$, it follows that at least one index will be returned with probability $1 - \delta$.

For the complexity, the update time of count-max data structure $\mathbf{A}$ follows from the routine `Fast-Update` of Lemma 3.4.3, and the query time follows from Lemma 3.4.12. Theorem 19 shows that the entire algorithm can be derandomized to use a random seed with $O(\log^2 n(\log\log(n))^2)$ bits, so to complete the claim it suffices to note that using $O(\log n)$-bit precision as required by `Fast-Update` (Lemma 3.4.3), it follows that our whole data structure $\mathbf{A}$ can be stored with $O(\log^2 n)$ bits, which is dominated by the cost of storing the random seed. This gives the stated space after taking $O(\log\delta^{-1})$ parallel repetitions for $p < 2$. For $p = 2$, we only need a random seed of length $O(\log^3 n)$ for all $O(\log n\log\delta^{-1})$ repetitions by Corollary 3.4.11, which gives $O(\log^3 n\log\delta^{-1} + \log^3 n) = O(\log^3 n\log\delta^{-1})$ bits of space for $p = 2$ as stated. Similarly for the case of $p < 2$ and $\delta = 1/\mathrm{poly}(n)$, the stated space follows from Corollary 3.4.11. □

In particular, by setting $\epsilon = 1/\mathrm{poly}(n, M)$, we obtain the following:

Theorem 14. *Given $0 < p \leq 2$, for any constant $c \geq 2$ there is a one-pass perfect L_p sampler, i.e. a $(0, \delta, O(n^{-c}))$-approximate L_p sampler. The space used is:*

- $O(\log^2 n\log\delta^{-1}(\log\log n)^2)$ *bits for $p < 2$.*
- $O(\log^3 n)$ *bits when $p < 2$ and $\delta = 1/\mathrm{poly}(n)$.*

- $O(\log^3 n \log \delta^{-1})$ *bits for* $p = 2$

Proof. By Theorem 13 with $\epsilon = n^{-c}$, the output $X \in [n]$ of the sampler satisfies

$$\begin{aligned} \mathbf{Pr}[X = i] &= (1 \pm \tilde{O}(\epsilon))\frac{|x_i|^p}{\|x\|_p^p} \pm O(n^{-c}) \\ &= \frac{|x_i|^p}{\|x\|_p^p} \pm O(n^{-c} + \epsilon) \\ &= \frac{|x_i|^p}{\|x\|_p^p} \pm O(n^{-c}) \end{aligned} \tag{3.7}$$

Thus the algorithm is a $(0, \delta, O(n^{-c}))$-approximate sampler, making it a perfect L_p sampler via Definition 3.1.2. □

Finally, we note that the cause of having to pay an extra $(\log \log n)^2$ factor in the space complexity for $p < 2$ is only due to the derandomization. Thus, in the random oracle model where the algorithm has access to a $\mathrm{poly}(n)$-length random tape which does not count against its space requirement, the space is an optimal $O(\log^2(n) \log(1/\delta))$. We remark that the $\Omega(\log^2(n) \log(1/\delta))$ of [KNP+17] lower bound also holds in the random oracle model.

Corollary 3.3.7. *For $p \in (0, 2)$, in the random oracle model, there is a perfect L_p sampler which fails with probability $\delta > 0$ and uses $O(\log^2(n) \log(1/\delta))$ bits of space.*

Remark 15. Note that for p arbitrarily close to 2, the bound on $\|z\|_2$ of Proposition 3.3.5 as used in Lemma 3.3.6 degrades, as the sum of the L_2 norms of the level sets is no longer geometric, and must be bounded by $O(\sqrt{\log(n)}\|F\|_2)$. In this case, the failure probability from Lemma 3.3.6 goes to $\Theta(\frac{1}{\log(n)})$, and so we must use the upper bound for $p = 2$. Similarly, for p arbitrarily close to 0, the bound also degrades since the values $V_{D(k)}$ in Lemma 3.3.3 blow-up. For such non-constant p arbitrarily close to 0, we direct the reader to the $O(\log^2(n))$-bit perfect L_0 sampler of [JST11].

3.4 Time and Space Complexity

In this section, we will show how the L_p sampling algorithm $\mathrm{L_p\text{-}Sampler}$ from Section 3.3 can be implemented with the space and time complexity as stated in Theorem 13. First, in Section 3.4.1, we show how the sampler can be implemented with the update procedure $\mathrm{Fast\text{-}Update}$ to result in $\tilde{O}(\epsilon^{-1})$ update time. Next, in Section 3.4.2, we show that the algorithm $\mathrm{L_p\text{-}Sampler}$ with

Fast-Update can be derandomized to use a random seed of length $O(\log^2(n)(\log\log n)^2)$-bits, which will give the desired space complexity. Finally, in Section 3.4.3, we show how using an additional heavy-hitters data structure as a subroutine, we can obtain $\tilde{O}(1)$ update time as well. This additional data structure will not increase the space or update time complexity of the entire algorithm, and does not need to be derandomized.

3.4.1 Optimizing the Update Time

In this section we prove Lemma 3.4.3, which demonstrates that the sampling algorithm can be implemented with a fast update procedure. Our algorithm utilizes a single data structure run on the stream ζ, which is count-max matrix $\mathbf{A} \in \mathbb{R}^{d\times 2}$ where $d = \Theta(\log n)$. We will introduce an update procedure Fast-Update which updates the count-max data structure $\mathbf{A}$ of $\mathrm{L_p}$-Sampler in $\tilde{O}(\epsilon^{-1})$ time.

Throughout this section, we will refer to the *original algorithm* as the algorithm which implements $\mathrm{L_p}$-Sampler of section Section 3.3 by individually generating each scaling exponential t_i for $i \in [n^c]$, rounding them to create the vector ζ, and hashing them individually into $\mathbf{A}$ (naïvely taking n^c update time). Our procedure will utilize the following result about efficiently sampling binomial random variables which can be found in [BKP+14].

Proposition 3.4.1. *For any constant $c > 0$ and integer $n \geq 1$, there is an algorithm that can draw a sample $X \sim \mathtt{Bin}(n, 1/2)$ in expected $O(1)$ time in the unit cost RAM model. Moreover, it can be sampled in time $\tilde{O}(1)$ with probability $1 - n^{-c}$. The space required is $O(\log n)$-bits.*

Proof. The proof of the running time bounds and correctness can be found in [BKP+14]. Since they do not analyze the space complexity of their routine, we do so here. Their algorithm is as follows. We can assume n is even, otherwise we could sample $\mathtt{Bin}(n, q) \sim \mathtt{Bin}(n-1, q) + \mathtt{Bin}(1, q)$, where the latter can be sampled in constant time (ion the unit cost RAM model) and $O(\log(n))$-bits of space. The algorithm first computes $\Delta \in [\sqrt{n}, \sqrt{n}+3]$, which can be done via any rough approximation of the function $\sqrt{x}$, and requires only $O(\log(n))$-bits. Define the block $\mathcal{B}_k = \{km, km+1, \ldots, km+m-1\}$ for $k \in \mathbb{Z}$, and set

$$f(i) = \frac{4}{2^{\max\{k,-k-1\}}m} \quad \text{s.t. } i \in \mathcal{B}_k$$

$$p(i) = 2^{-n}\binom{n}{n/2+i}$$

Note that given i, $f(i)$ can be computed in constant time and $O(\log(n))$ bits of space. The algorithm then performs the following loop:

1. Sample i via the normalized probability distribution $\bar{f} = f/16$.
2. Return $n/2 + i$ with probability $p(i)/f(i)$
3. Else, reject i and return to Step 1.

To compute the first step, the symmetry around $n/2$ of f is utilized. We flip unbiased coins $C_1, C_2, \ldots$ until we obtain C_{t+1} which lands tails, and pick i uniformly from block $\mathcal{B}_t$ or $\mathcal{B}_{-t}$ (where the choice is decided by a single coin flip). The procedure requires at most $O(\log(n))$-bits to store the index t. Next, to perform the second step, we obtain 2^{-L} additive error approximations $\tilde{q}$ of $q = (p(i)/f(i))$ for $L = 1, 2, \ldots$, which (using the fact that $0 \leq q \leq 1$) can be done by obtaining a 2^{-L}-relative error approximation of q. Then we flip L random bits to obtain a uniform $\tilde{R} \in [0, 1]$, and check if $|\tilde{R} - \tilde{q}| > 2^{-L}$. If so, we can either accept or reject i based on whether $\tilde{R} > \tilde{q} + 2^{-L}$ or not, otherwise we repeat with $L \leftarrow L + 1$.

To obtain $\tilde{q}$, it suffices to obtain a 2^{-L-1} relative error approximation of the factorial function $x!$. To do so, the 2^{-L} approximation

$$x! \approx (x + L)^{x+1/2} e^{-(x+L)} \Big[\sqrt{2\pi} + \sum_{k=1}^{L-1} \frac{c_k}{x + k}\Big]$$

is used, where $c_k = \frac{(-1)^{k-1}}{(k-1)!}(L - k)^{k-1/2} e^{L-k}$. This requires estimating the functions e^x, $\sqrt{x}$ and π, all of which, as well as each term in the sum, need only be estimated to $O(L)$-bits of accuracy (as demonstrated in [BKP+14]). Thus the entire procedure is completed in $O(L) = O(\log(n))$-bits of space (L can never exceed $O(\log(n))$, as q is specified with at most $O(\log(n))$ bits), which completes the proof. □

We now utilize a straightforward reduction from the case of sampling from $\texttt{Bin}(n, q)$ for any $q \in [0, 1]$ to sampling several times from $\texttt{Bin}(n', 1/2)$ where $n' \leq n$. This reduction has been observed before [FCT15], however we will state it here to clearly demonstrate our desired space and time bounds.

Lemma 3.4.2. *For any constant $c > 0$, integer $n \geq 1$, and real $q \in [0, 1]$, there is an algorithm that can draw a sample $X \sim \texttt{Bin}(n, q)$ in expected $O(1)$ time in the unit cost RAM model. Moreover, it can be sampled in time $\tilde{O}(1)$ with probability $1 - n^{-c}$, and the space required is $O(\log n)$-bits.*

Proof. The reduction is as follows (for a more detailed proof of correctness, see [FCT15]). We sample $\texttt{Bin}(n, q)$ by determining how many of the n trials were successful. This can be done by generating variables $u_1, \ldots, u_n$ uniform on $[0, 1]$, and determining how many are less than q.

We do this without generating all the variables u_i explicitly as follows. First write q in binary as $q = (0.q_1q_2, \ldots)_2$. Set $b \leftarrow 0$, $j \leftarrow 1$, $n_j \leftarrow n$ and sample $b_j \sim \texttt{Bin}(n_j, 1/2)$. If $q_j = 1$, then set $b = b + b_j$, as these corresponding b_j trials u_i with the first bit set to 0 will all be successful trials given that $q_j = 1$. Then set $n_{j+1} \leftarrow n_j - b_j$ and repeat with $j \leftarrow j + 1$. Otherwise, if $q_j = 0$, then we set $n_{j+1} \leftarrow n_j - (n_j - b_j) = b_j$, since this represents the fact that $(n_j - b_j)$ of the variables u_i will be larger than q. With probability $1 - n^{-100c}$, we reach the point where $n_j = 0$ within $O(\log(n))$ iterations, and we return the value stored in b at this point. By Proposition 3.4.1, each iteration requires $\tilde{O}(1)$ time, and thus the entire procedure is $\tilde{O}(1)$.

For space, note that we need only store q to its first $O(\log(n))$ bits, since the procedure terminates with high probability within $O(\log(n))$ iterations. Then the entire procedure requires $O(\log(n))$ bits, since each sample of $\texttt{Bin}(n_j, 1/2)$ requires only $O(\log(n))$ space by Proposition 3.4.1.

□

The Fast-Update **procedure.** We are now ready to describe the implementation of our algorithms update procedure. Specifically, our goal is to show that the update time of the (ϵ, δ)-approximate sampling algorithm L_p-Sampler can be made $\tilde{O}(1/\epsilon)$. Recall that for our *perfect* L_p sampler we require $\epsilon = 1/\text{poly}(n)$, thus the Fast-Update procedure will not improve the update time of our perfect L_p sampler (beyond the naïve $\text{poly}(n)$). However, if one allows the relative error ϵ to be larger, then a $\tilde{O}(1/\epsilon)$ update time is now much faster. Thus, the Fast-Update procedure allows for a trade-off between the update time and the relative error of the sampler. Note that all prior works had a dependency on the relative error ϵ in *both* the spacial complexity and the update time of the sampler.[6]

Recall that our algorithm utilizes just a single data structure on the stream ζ: the $d \times 2$ count-max matrix A (where $d = \Theta(\log(n))$). Upon receiving an update (i, Δ) to a coordinate x_i for $i \in [n]$, we proceed as follows. Our goal is to compute the set

$$\left\{ \texttt{rnd}_\epsilon \left(\frac{1}{t_{i_1}^{1/p}} \right), \texttt{rnd}_\epsilon \left(\frac{1}{t_{i_2}^{1/p}} \right), \ldots, \texttt{rnd}_\epsilon \left(\frac{1}{t_{i_{n^{c-1}}}^{1/p}} \right) \right\}$$

[6]Note that our algorithm has no dependency on ϵ in the spacial complexity so long as $1/\epsilon = O(\text{poly}(n))$.

and update each row of $\mathbf{A}$ accordingly in $\tilde{O}(\epsilon^{-1})$ time, Naïvely, this could be done by computing each value individually, and then updating each row of $\mathbf{A}$ accordingly, however this would require $O(n^{c-1})$ time. To avoid this, and obtain speed-ups when the relative error ϵ is made larger than $1/\mathrm{poly}(n)$, we exploit the fact that the support size of $\mathtt{rnd}_\epsilon(x)$ for $1/\mathrm{poly}(n) \leq x \leq \mathrm{poly}(n)$ is $\tilde{O}(\epsilon^{-1})$, so it will suffice to determine how many variables $\mathtt{rnd}_\epsilon(1/t_{i_j}^{1/p})$ are equal to each value in the support of $\mathtt{rnd}_\epsilon(x)$.

Our update procedure is then as follows. Let $I_j = (1+\epsilon)^j$ for $j = -\Pi, -\Pi+1, \ldots, \Pi-1, \Pi$ where $\Pi = O(\log(n)\epsilon^{-1})$. We utilize the c.d.f. $\psi(x) = 1 - e^{-x^{-p}}$ of the $1/p$-th power of the inverse exponential distribution $t^{-1/p}$ (here t is exponentially distributed). Then beginning with $j = -\Pi, -\Pi+1, \ldots, \Pi$ we compute the probability $q_j = \psi(I_{j+1}) - \psi(I_j)$ that $\mathtt{rnd}_\epsilon(1/t^{1/p}) = I_j$, and then compute the number of values Q_j in $\{\mathtt{rnd}_\epsilon(1/t_{i_1}^{1/p}), \mathtt{rnd}_\epsilon(1/t_{i_2}^{1/p}), \ldots, \mathtt{rnd}_\epsilon(1/t_{i_{n^{c-1}}}^{1/p})\}$ which are equal to I_j. With probability $1 - n^{100c}$, we know that $1/\mathrm{poly}(n) \leq t_i \leq \mathrm{poly}(n)$ for all $i \in [N]$, and thus conditioned on this, we will have completely determined the values of the items in $\{\mathtt{rnd}_\epsilon(1/t_{i_1}^{1/p}), \mathtt{rnd}_\epsilon(1/t_{i_2}^{1/p}), \ldots, \mathtt{rnd}_\epsilon(1/t_{i_{n^{c-1}}}^{1/p})\}$ by looking at the number equal to I_j for $j = -\Pi, \ldots, \Pi$.

Now we know that there are Q_j updates which we need to hash into $\mathbf{A}$ (along with i.i.d. Gaussian scalings), each with the same value ΔI_j. This is done by the procedure Fast-Update-CS (Figure 3.5), which computes the number $b_{k,\theta}$ that hash to each bucket $\mathbf{A}_{k,\theta}$ by drawing binomial random variables. Once this is done, we know that the value of $\mathbf{A}_{k,\theta}$ should be updated by the value $\sum_{t=1}^{b_{k,\theta}} g_t \Delta I_j$, where each $g_t \sim \mathcal{N}(0,1)$. Naïvely, computing the value $\sum_{t=1}^{b_{k,\theta}} g_t \Delta I_j$ would involve generating $b_{k,\theta}$ random Gaussians. To avoid this, we utilize the 2-stability of Gaussians (Definition 2.2.6), which asserts that $\sum_{t=1}^{b_{k,\theta}} g_t \Delta I_j \sim g\sqrt{b_{k,\theta}}\Delta I_j$, where $g \sim \mathcal{N}(0,1)$. Thus we can simply generate and store the Gaussian g associated with the item $i \in [n]$, rounding I_j, and bucket $\mathbf{A}_{k,\theta}$, and on each update Δ to x_i we can update $\mathbf{A}_{k,\theta}$ by $g\sqrt{b_{k,\theta}}\Delta I_j$.

Finally, once the number of values in $\{\mathtt{rnd}_\epsilon(1/t_{i_1}^{1/p}), \mathtt{rnd}_\epsilon(1/t_{i_2}^{1/p}), \ldots, \mathtt{rnd}_\epsilon(1/t_{i_{n^{c-1}}}^{1/p})\}$ which are left to determine is less than K for some $K = \Theta(\log(n))$, we simply generate and hash each of the remaining variables individually. The generation process is the same as before, except that for each of these at most K remaining items we associate a fixed index i_j for $j \in [n^{c-1}]$, and store the relevant random variables $h_\ell(i_j), g_\ell(i_j)$ for $\ell \in [d]$. Since the value of j which is chosen for each of these coordinates does not affect the behavior of the algorithm – in other words the index of the duplicate which is among the K largest is irrelevant – we can simply choose these indices to be $i_1, i_2, \ldots, i_K \in [N]$ so that the first item hashed individually via step 3 corresponds to ζ_{i_1}, the second to ζ_{i_2}, and so on.

Note that the randomness used to process an update corresponding to a fixed $i \in [n]$ is stored

Fast-Update $(i, \Delta, \mathbf{A})$

Set $L = n^{c-1}$, and fix $K = \Theta(\log(n))$ with a large enough constant.
For $j = -\Pi, -\Pi + 1, \ldots, \Pi - 1, \Pi$:

1. Compute $q_j = \psi(I_{j+1}) - \psi(I_j)$.
2. Draw $Q_j \sim \mathtt{Bin}(L, q_j)$.
3. **If** $L < K$, hash the Q_j items individually into each row $\mathbf{A}_\ell$ using explicitly stored uniform i.i.d. random variables $h_\ell : [n^c] \to [2]$ and Gaussians $g_\ell(j)$ for $\ell \in [d]$.
4. **Else:** update count-max table $\mathbf{A}$ by via Fast-Update-CS$(\mathbf{A}, Q_j, I_j, \Delta, i)$
5. $L \leftarrow L - Q_j$.

Figure 3.4: Algorithm to Update count-max A

Fast-Update-CS $(\mathbf{A}, Q, I, \Delta, i)$

Set $W_k = Q$ for $k = 1, \ldots, d$
For $k = 1, \ldots, d$,

1. For $\theta = 1, 2$:
   (a) Draw $b_{k,\theta} \sim \mathtt{Bin}(W_k, \frac{1}{2-\theta+1})$.
   (b) Draw and store $g_{k,\theta,I,i} \sim \mathcal{N}(0, 1)$. Reuse on every call to Fast-Update-CS with the same parameters (k, θ, I, i).
   (c) Set $\mathbf{A}_{k,\theta} \leftarrow \mathbf{A}_{k,\theta} + g_{k,\theta,I,i}\sqrt{b_{k,\theta}}\Delta I$
   (d) $W_k \leftarrow W_k - b_{k,\theta}$.

Figure 3.5: Update A via updates to Q coordinates, each with a value of ΔI

so it can be reused to generate the same updates to $\mathbf{A}$ whenever an update to i is made. Thus, each time an update $+1$ is made to a coordinate $i \in [n]$, each bucket of count-max is updated by the same value. When an update of size Δ comes, this update to the count-max buckets is scaled by Δ. For each $i \in [n]$, let K_i denote the size of L when step 3 of Figure 3.4 was first executed while processing an update to i. In other words, the coordinates $\zeta_{i_1}, \ldots, \zeta_{i_{K_i}}$ were hashed into each row $\ell \in [d]$ of $\mathbf{A}$ using explicitly stored random variables $h_\ell(i_j), g_\ell(i_j)$. Let $\mathcal{K} = \cup_{i \in [n]} \cup_{j=1}^{K_i} \{i_j\}$. Then on the termination of the algorithm, to find the maximizer of ζ, the count-max algorithm checks for each $i \in \mathcal{K}$, whether i hashed to the largest bucket (in absolute value) in a row at least a $\frac{4}{5}$ fraction of the time. Count-max then returns the first i which satisfies this, or $\perp$ if no such coordinate exists. In other words, the count-max algorithm decides to output $\perp$ or output an index i based on computing the fraction of rows for which i hashes into the largest bucket, except now it only computes these values for $i \in \mathcal{K}$ instead of $i \in [n^c]$, thus count-max can only

return a value of $i \in \mathcal{K}$. We now argue that the distribution of our algorithm is not changed by using the update procedure Fast-Update. This will involve showing that $\arg\max\{|\zeta_i|\} \in \mathcal{K}$ if our algorithm was to return a coordinate originally.

Lemma 3.4.3. *Running the* $\mathrm{L_p}$-Sampler *with the update procedure given by* Fast-Update *results in the same distribution over the count-max table* $\mathbf{A}$ *as the original algorithm. Moreover, conditioned on a fixed realization of* $\mathbf{A}$*, the output of the original algorithm will be the same as the output of the algorithm using* Fast-Update*. For a given* $i \in [n]$*,* Fast-Update *requires* $\tilde{O}(\epsilon^{-1})$*-random bits, and runs in time* $\tilde{O}(\epsilon^{-1})$*.*

Proof. To hash an update Δ to a coordinate x_i, the procedure Fast-Update computes the number Q_j of variables in the set $\{\mathtt{rnd}_\epsilon(1/t_{i_1}^{1/p}), \mathtt{rnd}_\epsilon(1/t_{i_2}^{1/p}), \ldots, \mathtt{rnd}_\epsilon(1/t_{i_{n^{c-1}}}^{1/p})\}$ which are equal to I_j for each $j \in \{-\Pi, \ldots, \Pi\}$. Instead of computing Q_j by individually generating the variables and rounding them, we utilize a binomial random variable to determine Q_j, which results in the same distribution over $\{\mathtt{rnd}_\epsilon(1/t_{i_1}^{1/p}), \mathtt{rnd}_\epsilon(1/t_{i_2}^{1/p}), \ldots, \mathtt{rnd}_\epsilon(1/t_{i_{n^{c-1}}}^{1/p})\}$. As noted, with probability $1 - n^{100c}$ none of the variables $\mathtt{rnd}_\epsilon(1/t_{i_j}^{1/p})$ will be equal to I_k for $|k| > \Pi$, which follows from the fact that $n^{-101c} < t_i < O(\log(n))$ with probability $1 - n^{-101c}$ and then union bounding over all n^c exponential variables t_i. So we can safely ignore this low probability event.

Once computed, we can easily sample from the number of items of the Q_j that go into each bucket $\mathbf{A}_{k,\theta}$, which is the value $b_{k,\theta}$ in Fast-Update-CS (Figure 3.5). By 2-stability of Gaussians (Definition 2.2.6), we can update each bucket $\mathbf{A}_{k,\theta}$ by $g_{k,\theta,I_j,i}\sqrt{b_{k,\theta}}\Delta I_j$, which is distributed precisely the same as if we had individually generated each of the $b_{k,\theta}$ Gaussians, and taken their inner product with the vector $\Delta I_j \vec{1}$, where $\vec{1}$ is the all 1's vector. Storing the explicit values $h_\ell(i_j)$ for the top K largest values of $\mathtt{rnd}_\epsilon(1/t_{i_j}^{1/p})$ does not effect the distribution, but only allows the algorithm to determine the induces of the largest coordinates i_j corresponding to each $i \in [n]$ at the termination of the algorithm. Thus the distribution of updates to $\mathbf{A}$ is unchanged by the Fast-Update Procedure.

We now show that the output of the algorithm run with this update procedure is the same as it would have been had all the random variables been generated and hashed individually. First observe that for $\epsilon < 1/2$, no value $q_j = \psi(I_{j+1}) - \psi(I_j)$ is greater than $1/2$. Thus at any iteration, if $L > K$ then $L - \mathtt{Bin}(L, q_j) > L/3$ with probability $1 - n^{-100c}$ by Chernoff bounds (using that $K = \Omega(\log(n))$). Thus the first iteration at which L drops below K, we will have $L > K/3$. So for each $i \in [n]$ the top $K/3$ values ζ_{i_j} will be hashed into each row $\mathbf{A}_\ell$ using stored random variables $h_\ell(i_j)$, so $K_i > K/3 = \Omega(\log(n))$ for all $i \in [n]$. In particular, $K_i > 0$ for all $i \in [n]$.

Now the only difference between the output procedure of the original algorithm and that of

the efficient-update time algorithm is that in the latter we only compute the values of $\alpha_{i_j} = \left|\{t \in [d] \mid |\mathbf{A}_{t,h_t(i_j)}| = \max_{r\in\{1,2\}} |\mathbf{A}_{t,r}|\}\right|$ for the $i_j \in [n^c]$ corresponding to the K_i largest values $t_{i_j}^{-1/p}$ in the set $\{t_{i_1}^{-1/p}, \ldots, t_{i_{n^{c-1}}}^{-1/p}\}$, whereas in the former all values of α_{i_j} are computed to find a potential maximizer. In other words, count-max with Fast-Update only searches throguh the subset $\mathcal{K} \subset [n^c]$ for a maxmizer instead of searching through all of $[n^c]$ (here $\mathcal{K}$ is as defined earlier in this section). Since count-max never outputs a index i_j that is not a unique maximizer with high probability, we know that the output of the original algorithm, if it does not output $\perp$), must be i_j such that $j = \arg\max_{j'}\{t_{i_{j'}}\}$, and therefore $i_j \in \mathcal{K}$. Note the n^{-c} failure probability can be safely absorbed into the additive n^{-c} error of the perfect L_p sampler. Thus the new algorithm will also output i_j. Since the new algorithm with Fast-Update searches over the subset $\mathcal{K} \subset [n^c]$ for a maximier, if the original algorithm outputs $\perp$ then certainly so will Fast-Update. Thus the output of the algorithm using Fast-Update is distributed identically (up to n^{-c} additive error) as the output of the original algorithm, which completes the proof.

Runtime & Random Bits For the last claim, first note that it suffices to generate all continuous random varaibles used up to $(nmM)^{-c} = 1/\mathrm{poly}(n)$ precision, which is $1/\mathrm{poly}(n)$ additive error after conditioning on the event that all random variables are all at most $\mathrm{poly}(n)$ (which occurs with probability $1 - n^{-c}$), and recalling that the length of the stream m satisfies $m < \mathrm{poly}(n)$ for a suitably smaller $\mathrm{poly}(n)$ then as in the additive error. More formally, we truncate the binary representation of every continuous random variable (both the exponentials and Gaussians) after $O(\log(n))$-bits with a sufficiently large constant. This will result in at most an additive $1/\mathrm{poly}(n)$ error for each bucket $\mathbf{A}_{i,j}$ of A, which can be absorbed by the adversarial error $\mathcal{V}_{i,j}$ with $|\mathcal{V}_{i,j}| = O(\epsilon(|\mathbf{A}_{i,1}| + |\mathbf{A}_{i,2}|))$ that we incur in each of these buckets already in Lemma 3.3.3. Thus each random variable requires $O(\log(n))$ bits to specify. Similarly, a precision of at most $(nmM)^{-c}$ is needed in the computation of the q_j's in Figure 3.4 by Lemma 3.4.2, since the routine to compute $\texttt{Bin}(n, q_j)$ will terminate with probability $1 - n^{-100c}$ after querying at most $O(\log(n))$ bits of q_j. Now there are at most $2\Pi = O(\epsilon^{-1}\log(n))$ iterations of the loop in Fast-Update. Within each, our call to sample a binomial random variable is carried out in $\tilde{O}(1)$ time with high probability by Lemma 3.4.2 (and thus use at most $\tilde{O}(1)$ random bits), and there are $\tilde{O}(1)$ entries in $\mathbf{A}$ to update (which upper bounds the running time and randomness requirements of Fast-Update-CS).

Note that since the stream has length $m = \mathrm{poly}(n)$, and there are at most $\tilde{O}(\epsilon)$ calls made to sample binomial random variables in each, we can union bound over each call to guarantee that each returns in $\tilde{O}(1)$ time with probability $1 - n^{-100c}$. Since $K = \tilde{O}(1)$, we must store an additional $\tilde{O}(1)$ random bits to store the individual random variables $h_\ell(i_j)$ for $i_j \in \{i_1, \ldots, i_{K_i}\}$.

Similarly, we must store $\tilde{O}(\epsilon)$ independent Gaussians for the procedure Fast-Update-CS, which also terminates in $\tilde{O}(1)$ time, which completes the proof.

□

3.4.2 Derandomizing the Algorithm

We now show that the algorithm $\text{L}_\text{p}\text{-Sampler}$ with Fast-Update can be derandomized without affecting the space or time complexity. Recall that $\text{L}_\text{p}\text{-Sampler}$ utilizes two main sources of randomness. Firstly, it uses randomness to generate the exponential random scaling variables $(t_1, \ldots, t_{n^c})$ (the "exponential randomness"), and secondly, it uses randomness to generate the Gaussian coefficients $g_i(j)$ and fully random hash functions $h_i(j)$ needed for count-max (the "count-max randomness"). To derandomize both these sources of randomness, we will need to use a combination of Nisan's pseudorandom generator (PRG) [Nis92], and the PRG of Goplan, Kane, and Meka [GKM18]. Specifically, we will derandomize the exponential randomness with the PRG of Goplan, Kane, and Meka, and we will derandomize the count-max randomness with Nisan's PRG.

We begin by introducing Nisan's PRG, which is a deterministic map $G : \{0,1\}^\ell \to \{0,1\}^T$, where $T \gg \ell$ (for instance, one can think of $T = \text{poly}(n)$ and $\ell = O(\log^2(n))$). Let $\sigma : \{0,1\}^T \to \{0,1\}$ be a tester (a function computable under some specified restrictions). For the case of Nisan's PRG, σ must be a tester which reads its random T-bit input in a stream, left to right, and outputs either 0 or 1 at the end. Nisan's PRG can be used to fool *any* such tester, which means:

$$\left| \mathbf{Pr}\left[\sigma(U_T) = 1\right] - \mathbf{Pr}\left[\sigma(G(U_\ell)) = 1\right] \right| < \frac{1}{T^c}$$

Where U_t indicates t uniformly random bits for any t, and c is a sufficiently large constant. Here the probability is taken over the choice of the random bits U_T and U_ℓ. In other words, the probability that σ outputs 1 is nearly the same when it is given random input as opposed to input from Nisan's generator. Note that since $\sigma(U_T)$ is a $0, 1$ random variable, the left hand side of the above equation can be rewritten as $|\,\mathbf{E}\left[\sigma(U_T)\right] - \mathbf{E}\left[\sigma(G(U_\ell))\right]|$. Nisan's theorem states if σ can be computed by an algorithm with at most $\text{poly}(T)$ states and which uses a working memory tape of size at most $O(\log T)$, then a seed length of $\ell = O(\log^2 T)$ suffices for the above result [Nis92]. More generally, if σ can be computed by an algorithm with $\text{poly}(T)$ states and which uses a working memory of tape of size $S = \Omega(\log T)$, then a seed length of $\ell = O(S \log T)$ suffices for the above result. Thus Nisan's PRG fools space bounded testers σ that read their randomness in a stream.

Why Nisan's PRG alone is insufficient. We remark that it is possible to derandomize our entire algorithm with Nisan's PRG, albeit with suboptimal seed length. Since our algorithm is a linear sketch and is therefore independent of the ordering of the stream, one can assume for the sake of the derandomization that the stream is ordered so that all updates to a single coordinate occur consecutively (this is a fairly standard argument, e.g. [Ind06]). Reading the exponential and count-max randomness in a stream, one can then fully construct the state of the algorithm's data structure at the end of the stream, by adding the contribution of each coordinate to the whole data structure one by one. The space required to do this is the size of the data structure, which is $O(\log^2 n)$ bits. Then to derandomize with Nisan's PRG, we would require a seed length of $O(\log^3 n)$-bits, which does not match our desired space complexity. Thus to improve the seed length to $O(\log^2 n(\log\log n)^2)$, we will need the approach followed here.

We remark that the main difficulty in applying Nisan's PRG alone is that, for a given i, to test if i is returned by count-max, one must check for each pair of buckets in count-max whether i hashes to the larger bucket. Since each bucket depends on the same exponential randomness, one would either need to make multiple passes over the exponential randomness (which is not allowed by Nisan's), once for each bucket, or one would need to store all the buckets simultaneously. On the other hand, if the exponential randomness was *fixed*, and hard-coded into the tester σ, then one could construct the buckets one at a time, reading only the count-max randomness in a stream, and thus only using $O(\log n)$ bits of space. We make use of this latter fact, by generating the exponential randomness with a separate PRG from [GKM18], to obtain our main result.

Road-map for the Derandomization

We now briefly lay out the structure of this section. First, in Section 3.4.2, we introduce the PRG of Goplan, Kane, and Meka [GKM18], along with the notion of a half-space tester which will be crucial for us. We then demonstrate in Lemma 3.4.7 how the PRG of [GKM18] can be used to fool such half-space testers with a small seed length. Recall the PRG is just a function $G : \{0,1\}^\ell \to \{-M, \dots, M\}^n$ for some $M = \text{poly}(n)$. However, in order to use this PRG for our streaming algorithm, we must show not only that the seed length is small, but also that given a random seed r, one can compute each coordinate of $G(r)$ in small space *and* small runtime, which we do in Proposition 3.4.8. We then use this fact that G can fool half-space testers to prove a set of general results about derandomizing streaming algorithms, which is captured in Theorem 17. As a corollary, we obtain a novel and improved derandomization of the count-sketch variant of [MP14]. Finally, in Section 3.4.2, we present our main theorem, Theorem 19, which uses a

combination of the PRG of Goplan, Kane, and Meka as well as Nisan's PRG to derandomize our streaming algorithm.

Half Space Fooling PRG's.

Our derandomization crucially uses the PRG of Goplan, Kane, and Meka [GKM18], which fools a certain class of fourier transforms. Utilizing the results of [GKM18], we will design a PRG that can fool *arbitrary* functions of $\lambda = O(\log(n))$ halfspaces, using a seed of length $O(\log^2(n)(\log\log(n))^2)$. We remark that in [GKM18] it is shown how to obtain such a PRG for a function of a single half-space. Using extensions of the techniques in that paper, we demonstrate that the same PRG with a smaller precision ϵ can be used to fool functions of more half-spaces. We now introduce the main result of [GKM18]. Let $\mathbb{C}_1 = \{c \in \mathbb{C} \mid |c| \leq 1\}$.

Definition 3.4.4 (Definition 1 [GKM18])**.** *An (m, n)-Fourier shape $f : [m]^n \to \mathbb{C}_1$ is a function of the form $f(x_1, \ldots, x_n) = \prod_{j=1}^n f_j(x_j)$ where each $f_j : [m] \to \mathbb{C}_1$.*

Theorem 16 (Theorem 1.1 [GKM18])**.** *There is a PRG $G : \{0,1\}^\ell \to [m]^n$ that fools all (m, n)-Fourier shapes f with error ϵ using a seed of length $\ell = O(\log(mn/\epsilon)(\log\log(mn/\epsilon))^2)$, meaning:*

$$|\mathbf{E}[f(x)] - \mathbf{E}[f(G(y))]| \leq \epsilon$$

where x is uniformly chosen from $[m]^n$ and y from $\{0,1\}^\ell$.

For any $a^1, \ldots, a^\lambda \in \mathbb{Z}^n$ and $\theta_1, \ldots, \theta_\lambda \in \mathbb{Z}$, let $H_i : \mathbb{R}^n \to \{0,1\}$, be the function given by $H_i(X_1, \ldots, X_n) = \mathbf{1}[a_1^i X_1 + a_2^i X_2 + \cdots + a_n^i X_n > \theta_i]$, where $\mathbf{1}$ is the indicator function. We now define the notion of a λ-halfspace tester, and what it means to fool one.

Definition 3.4.5 (λ-halfspace tester)**.** *A λ-halfspace tester is any function $\sigma_H : \mathbb{R}^n \to \{0,1\}$ which, on input $X = (X_1, \ldots, X_n)$, outputs $\sigma'_H(H_1(X), \ldots, H_\lambda(X)) \in \{0,1\}$ where σ'_H is any fixed function $\sigma'_H : \{0,1\}^\lambda \to \{0,1\}$. In other words, the Boolean valued function $\sigma_H(X)$ only depends on the values $(H_1(X), \ldots, H_\lambda(X))$. A λ-halfspace tester is said to be M bounded if all the half-space coefficents a_j^i and θ_i are integers of magnitude at most M, and each X_i is drawn from a discrete distrubtion $\mathcal{D}$ with support contained in $\{-M, \ldots, M\} \subset \mathbb{Z}$.*

Definition 3.4.6 (Fooling a λ-halfspace tester)**.** *Fix any $\epsilon > 0$. A PRG $G : \{0,1\}^\ell \to \mathbb{R}^n$ is said to ϵ-fool the class of λ-halfspace testers under a distribution $\mathcal{D}$ over $\mathbb{R}^n$ if for every set of λ*

halfspaces $H = (H_1, \ldots, H_\lambda)$ and every λ-halfspace tester $\sigma_H : \mathbb{R}^n \to \{0,1\}$, we have:

$$\left|\mathbb{E}_{X\sim\mathcal{D}}\Big[\sigma_H(X) = 1\Big] - \mathbb{E}_{y\sim\{0,1\}^\ell}\Big[\sigma_H(G(y)) = 1\Big]\right| < \epsilon$$

Here ℓ is the seed length of G.

We will consider only product distributions $\mathcal{D}$. In other words, we assume that each coordiante X_i is drawn i.i.d. from a fixed distribution $\mathcal{D}$ over $\{-M, \ldots, M\} \subset \mathbb{Z}$. We consider PRG's $G : \{0,1\}^\ell \to \{-M, \ldots, M\}^n$ which take in a random seed of length ℓ and output a $X' \in \{-M, \ldots, M\}^n$ such that any M-bounded λ-halfspace tester will be unable to distinguish X' from $X \sim \mathcal{D}^n$ (where $\mathcal{D}^n$ is the product distribution of $\mathcal{D}$, such that each $X_i \sim \mathcal{D}$ independently). The following Lemma demonstrates that the PRG of [GKM18] can be used to fool M-bounded λ-halfspace testers. The authors would like to thank Raghu Meka for providing us a proof of Lemma 3.4.7.

Lemma 3.4.7. *Suppose $X_i \sim \mathcal{D}$ is a distribution on $\{-M, \ldots, M\}$ that can be sampled from with $\log(M') = O(\log(M))$ random bits. Then, for any $\epsilon > 0$ and constant $c \geq 1$, there is a PRG $G : \{0,1\}^\ell \to \{-M, \ldots, M\}^n$ which $\epsilon(nM)^{-c\lambda}$-fools the class of all M-bounded λ-halfspace testers on input $X \sim \mathcal{D}^n$ with a seed of length $\ell = O(\lambda \log(nM/\epsilon)(\log\log(nM/\epsilon))^2)$ (assuming $\lambda \leq n$). Moreover, if $G(y) = X' \in \{-M, \ldots, M\}^n$ is the output of G on random seed $y \in \{0,1\}^\ell$, then each coordinate X'_i can be computed in $O(\ell)$-space and in $\tilde{O}(1)$ time, where $\tilde{O}$ hides $\mathrm{poly}(\log(nM))$ factors.*

Proof. Let $X = (X_1, \ldots, X_n)$ be uniformly chosen from $[M']^n$ for some $M' = \mathrm{poly}(M)$, and let $Q : [M'] \to \{-M, \ldots, M\}$ be such that $Q(X_i) \sim \mathcal{D}^n$ for each $i \in [n]$. Let $a^1, \ldots, a^\lambda \in \mathbb{Z}^n$, $\theta_1, \ldots, \theta_\lambda \in \mathbb{Z}$ be $\log(M)$-bit integers, where $H_i(x) = \mathbf{1}[\langle a^i, x\rangle > \theta_i]$. Let $Y_i = \langle Q(X), a^i\rangle - \theta_i$. Note that $Y_i \in [-2M^2n, 2M^2n]$. So fix any $\alpha_i \in [-2M^2n, 2M^2n]$ for each $i \in [\lambda]$, and let $\alpha = (\alpha_1, \ldots, \alpha_\lambda)$. Let $h_\alpha(x) = \mathbf{1}(Y_1 = \alpha_1) \cdot \mathbf{1}(Y_2 = \alpha_2) \cdots \mathbf{1}(Y_\lambda = \alpha_\lambda)$, where $\mathbf{1}(\cdot)$ is the indicator function. Now define $f(x) = \sum_{j=1}^{\lambda} (2M^2n)^{j-1}\langle a^i, x\rangle$ for any $x \in \mathbb{Z}^n$. Note that $f(Q(X)) \in \{-(Mn)^{O(\lambda)}, \ldots, (Mn)^{O(\lambda)}\}$. We define the Kolmogorov distance between two integer valued random variables Z, Z' by $d_K(Z, Z') = \max_{k\in\mathbb{Z}}(|\mathbf{Pr}[Z \leq k] - \mathbf{Pr}[Z' \leq k]|)$. Let $X' \in [M']^n$ be generated via the (M', n)-fourier shape PRG of [GKM18] with error ϵ' (Theorem 1.1 [GKM18]). Observe $\mathbf{E}[h_\alpha(Q(X))] = \mathbf{Pr}\left[f(Q(X)) = \sum_{j=1}^{\lambda}(Mn)^{j-1}\alpha_j\right]$, so

$$|\mathbf{E}[h_\alpha(Q(X))] - \mathbf{E}[h_\alpha(Q(X')]| \leq d_K(f(Q(X)), f(Q(X')))$$

Now by Lemma 9.2 of [GKM18], we have

$$d_K(f(Q(X)), f(Q(X'))) = O\Big(\lambda \log(Mn) d_{FT}\Big(f(Q(X)), f(Q(X'))\Big)\Big)$$

where for integer valued Z, Z', we define

$$d_{FT}(Z, Z') = \max_{\beta \in [0,1]} |\mathbf{E}\left[\exp(2\pi i \beta Z)\right] - \mathbf{E}\left[\exp(2\pi i \beta Z')\right]|$$

Now $\exp(2\pi i \beta f(Q(X))) = \prod_{i=1}^{n}((\sum_{j=1}^{\lambda}(2M^2n)^{j-1}a_i^j)Q(X_i))$, which is a (M', n)-Fourier shape as in Definition 3.4.4. Thus by Theorem 16 (Theorem 1.1 of [GKM18]), we have

$$d_{FT}(f(Q(X)), f(Q(X'))) \leq \epsilon'$$

Thus

$$|\mathbf{E}\left[h_\alpha(Q(X))\right] - \mathbf{E}\left[h_\alpha(Q(X')\right]| = O(\lambda \log(Mn)\epsilon')$$

Now let $\sigma_H(x) = \sigma'_H(H_1(x), \ldots, H_\lambda(x))$ be any M-bounded λ-halfspace tester on $x \sim \mathcal{D}^n$. Since the inputs to the halfspaces H_i of σ'_H are all integers in $\{-2M^2n, 2M^2n\}$, let $A \subset \{-2M^2n, 2M^2n\}$ be the set of $\alpha \in A$ such that $Y = (Y_1, \ldots, Y_\lambda) = \alpha$ implies that $\sigma_H(Q(X)) = 1$, where $Q(X) \sim \mathcal{D}^n$ as above. Recall here that $Y_i = \langle Q(X), a^i \rangle - \theta_i$. Then we can think of a $\sigma_H(X) = \sigma''_H(Y_1, \ldots, Y_\lambda)$ for some function $\sigma''_H : \{-2M^2n, \ldots, 2M^2n\}^\lambda \to \{0, 1\}$, and in this case we have $A = \{\alpha \in \{-2M^2n, 2M^2n\} \mid \sigma''_H(\alpha) = 1\}$. Then

$$\begin{aligned}
|\mathbf{E}\left[\sigma_H(Q(X))\right] - \mathbf{E}\left[\sigma_H(Q(X'))\right]| &\leq \sum_{\alpha \in A} |\mathbf{E}\left[h_\alpha(Q(X))\right] - \mathbf{E}\left[h_\alpha(Q(X')\right]| \\
&\leq \sum_{\alpha \in A} O(\lambda \log(Mn)\epsilon')
\end{aligned}$$

Now note that $|A| = (nM)^{O(\lambda)}$, so setting $\epsilon' = \epsilon(nM)^{-O(\lambda)}$ with a suitably large constant, we obtain $|\mathbf{E}\left[\sigma_H(Q(X))\right] - \mathbf{E}\left[\sigma_H(Q(X'))\right]| \leq \epsilon(nM)^{-c\lambda}$ as needed. By Theorem 16, the seed required is $\ell = O(\lambda \log(nM/\epsilon)(\log\log(nM/\epsilon))^2)$ as needed. The space and time required to compute each coordinate follows from Proposition 3.4.8 below.

□

Proposition 3.4.8. *In the setting of Lemma 3.4.7, if $G(y) = X' \in \{-M, \ldots, M\}^n$ is the output of G on random seed $y \in \{0,1\}^\ell$, then each coordinate X'_i can be computed in $O(\ell)$-space and in $\tilde{O}(1)$ time, where $\tilde{O}$ hides $\mathrm{poly}(\log(nM))$ factors.*

Proof. In order to analyze the space complexity and runtime needed to compute a coordinate X'_i, we must describe the PRG of Theorem 16. The Goplan-Kane-Meka PRG has 3 main components, which themselves use other PRGs such as Nisan's PRG as sub-routines. Recall that the PRG generates a psuedo-uniform element from $X \sim [M]^n$ that fools a class of Fourier shapes $f : [M]^n \to \mathbb{C}$ on truly uniform input in $[M]^n$. Note that because of the definition of a Fourier shape, if we wish to sample from a distribution $X \sim \mathcal{D}$ over $\{-M, \ldots, M\}^n$ that is not uniform, but such that X_i can be sampled with $\log(M')$-bits, we can first fool Fourier shapes $f' : [M']^n \to \mathbb{C}$, and then use a function $Q : [M'] \to \{-M, \ldots, M\}$ which samples $X_i \sim \mathcal{D}$ given $\log(M')$ uniformly random bits. We then fool Fourier shapes $F = \prod_{j=1}^n f'_j(x) = \prod_{j=1}^n f_j(Q(y))$ where x, y are uniform, and thus $Q(y) \sim \mathcal{D}$. Thus it will suffice to fool (M', n)-Fourier shapes on uniform distributions. For simplicity, for the most part we will omit the parameter ϵ in this discussion.

The three components of the PRG appear in Sections 5,6, and 7 of [GKM18] respectively. In this proof, when we write Section x we are referring to the corresponding Section of [GKM18]. They consider two main cases: one where the function f has *high variance* (for some notion of variance), and one where it has low variance. The PRGs use two main pseudo-random primitives, δ-biased and k-wise independent hash function families. Formally, a family $\mathcal{H} = \{h : [n] \to [M]\}$ is said to be δ-biased if for all $r \leq n$ distinct indices $i_1, \ldots, i_r \in [n]$ and $j_1, \ldots, j_r \in [M]$ we have

$$\mathbf{Pr}_{h \sim \mathcal{H}}\left[h(i_1) = j_1 \wedge \cdots \wedge h(i_r) = j_r\right] = \frac{1}{M^r} \pm \delta$$

The function is said to be k-wise independent if it holds with $\delta = 0$ for all $r \leq k$. It is standard that k-wise independent families can be generated by taking a polynomial of degree k over a suitably large finite field (requiring space $O(k \log(Mn))$). Furthermore, a value $h(i)$ from a δ-biased family can be generated by taking products of two $O(\log(n/\delta))$-bit integers over a suitable finite field [Kop13] (requiring space $O(\log(n/\delta))$). So in both cases, computing a value $h(i)$ can be done in space and time that is linear in the space required to store the hash functions (or $O(\log(n/\delta))$-bit integers). Thus, any nested sequence of such hash functions used to compute a given coordinate X'_i can be carried out in space linear in the size required to store all the hash functions.

Now the first PRG (Section 5 [GKM18]) handles the high variance case. The PRG first sub-samples the n coordinates at $\log(n)$ levels using a pair-wise hash function (note that a 2-wise permutation is used in [GKM18], which reduces to computation of a 2-wise hash function). In each level S_j of sub-sampling, it uses $O(1)$-wise independent hash functions to generate the coordinates $X_i \in S_j$. So if we want to compute a value X_i, we can carry out one hash function

computation $h(i)$ to determine j such that $X_i \in S_j$, and then carry out another hash function computation $h_j(i) = X_i$. Instead of using $\log(n)$ independent hash functions h_j, each of size $O(\log(nM))$, for each of the buckets S_j, they derandomize this with the PRG of Nisan and Zuckerman [NZ96] to use a single seed of length $O(\log n)$. Now the PRG of Nisan and Zuckerman can be evaluated online, in the sense that it reads its random bits in a stream and writes its pseudo-random output on a one-way tape, and runs in space linear in the seed required to store the generator itself (see Definition 4 of [NZ96]). Such generators are composed to yield the final PRG of Theorem 2 [NZ96], however by Lemma 4 of the paper, such online generators are composable. Thus the entire generator of [NZ96] is online, and so any substring of the pseudorandom output can be computed in space linear in the seed of the generator by a single pass over the random input. Moreover, by Theorem 1 of [NZ96] in the setting of [GKM18], such a substring can be computed in $\tilde{O}(1)$ time, since it is only generating $\tilde{O}(1)$ random bits to begin with.

On top of this, the PRG of Section 5 [GKM18] first splits the coordinates $[n]$ via a limited independence hash function into $\text{poly}(\log(1/\epsilon))$ buckets, and applies the algorithm described above on each. To do this second layer of bucketing and not need fresh randomness for each bucket, they use Nisan's PRG [Nis92] with a seed of length $\log(n)\log\log(n)$. Now any bit of Nisan's PRG can be computed by several nested hash function computations, carried out in space linear in the seed required to store the PRG. Thus any substring of Nisan's can be computed in space linear in the seed and time $\tilde{O}(1)$. Thus to compute X_i', we first determine which bucket it hashes to, which involves computing random bits from Nisan's PRG. Then we determine a second partitioning, which is done via a 2-wise hash fucntion, and finally we compute the value of X_i' via an $O(1)$-wise hash function, where the randomness for this hash function is stored in a substring output by the PRG of [NZ96]. Altogether, we conclude that the PRG of Section 5 [GKM18] is such that value X_i' can be computed in space linear in the seed length and $\tilde{O}(1)$ time.

Next, in Section 6 of [GKM18], another PRG is introduced which reduces the problem to the case of $M \leq poly(n)$. Assuming a PRG G_1 is given which fools (M, n)-Fourier shapes, they design a PRG G_2 using G_1 which fools (M^2, n)-Fourier shapes. Applying this $O(\log\log(M))$ times reduces to the case of $m \leq n^4$. The PRG is as follows. Let $G_1, \ldots, G_t$ be the iteratively composed generators, where $t = O(\log\log(M))$. To compute the value of $(G_i)_j \in [M]$, where $(G_i)_j$ is the j-th coordinate of $G_i \in [M]^n$, the algorithm first implicitly generates a matrix $Z \in [M]^{\sqrt{M}\times M}$. An entry $Z_{p,q}$ is generated as follows. First one applies a k-wise hash function $h(q)$ (for some k), and uses the $O(\log M)$-bit value of $h(q)$ as a seed for a second 2-wise indepedent

hash function $h'_{h(q)}$. Then $Z_{p,q} = h'_{h(q)}(p)$. Thus within a column q of Z, the entries are 2-wise independent, and separate columns of Z are k-wise independent. This requires $O(k \log M)$-space to store, and the nested hash functions can be computed in $O(k \log M)$-space. Thus computing $Z_{i,j}$ is done in $\tilde{O}(1)$ time and space linear in the seed length. Then we set $(G_i)_j = Z_{(G_{i-1})_j, j}$ for each $j \in [n]$. Thus $(G_i)_j$ only depends on $(G_{i-1})_j$, and the random seeds stored for two hash functions to evaluate entries of Z. So altogether, the final output coordinate $(G_t)_j$ can be computed in space linear in the seed length required to store all required hash functions, and in time $\tilde{O}(1)$. Note importantly that the recursion is linear, in the sense that computing $(G_i)_j$ involves only one query to compute $(G_i)_{j'}$ for some j'.

Next, in Section 7 of [GKM18], another PRG is introduced for the *low-variance case*, which reduces the size of n to $\sqrt{n}$, but blows up m polynomially in the process. Formally, it shows given a PRG G'_1 that fools $(\mathrm{poly}(n), \sqrt{n})$ Fourier shapes, one can design a PRG G'_2 that fools $O(M, n)$-Fourier shapes with $M < n^4$ (here the $\mathrm{poly}(n)$ can be much larger than n^4). To do so, the PRG first hashes the n coordinates into $\sqrt{n}$ buckets k-wise independently, and then in each bucket uses k-wise independence to generate the value of the coordinate. A priori, this requires $\sqrt{n}$ independent seeds for the hash function in each of the buckets. To remove this requirement, it uses G'_1 to generate the $\sqrt{n}$ seeds required from a smaller seed. Thus to compute a coordinate i of G'_2, simply evaluate a k-wise independent hash function on i to determine which bucket $j \in [\sqrt{n}]$ the item i is hashed into. Then evaluate $G'_1(j)$ to obtain the seed required for the k-wise hash function h_j, and the final result is given by $h_j(i)$. Note that this procedure only requires one query to the prior generator G'_1. The space required to do so is linear in the space required to store the hash functions, and the space required to evaluate a coordinate of the output of G'_1, which will be linear in the size used to store G'_1 by induction.

Finally, the overall PRG composes the PRG from Section 6 and 7 to fool larger n, M in the case of low variance. Suppose we are given a PRG G_0 which fools $(M'', \sqrt{n'})$-Fourier shapes for some $M'' < (n')^2$. We show how to construct a PRG G_1 which fools (M', n')-Fourier shapes for any $M' \leq (n')^4$. Let G^{6+7} be the PRG obtained by first applying the PRG from Section 6 on G_0 as an initial point, which gives a PRG that fools $(\mathrm{poly}(n'), \sqrt{n'})$-Fourier shapes, and then applying the PRG from section 7 on top which now fools (M', n')-Fourier shapes (with low variance). Let G^5 be the generator from Section 5 which fools (M', n')-Fourier shapes with high variance. The final algorithm for fooling the class of all (M', n')-Fourier shapes given G_0 computes a generator G_1 such that the i-th coordinate is $(G_1)_i = (G^{6+7})_i \oplus (G^5)_i$, where $\oplus$ is addition mod M'. This allows one to simultaneously fool high and low variance Fourier shapes of the desired M', n'. If $M > (n')^4$, one can apply the PRG for Section 6 one last time on top of

G_1 to fool arbitrary M. Thus if for any i, the i-th coordinate of G_{6+7} and G_5 can be composed in $\tilde{O}(1)$ time and space linear in the size required to store the random seed, then so can G_i. Thus going from G_0 to G_1 takes a generator that fools $(M'', \sqrt{n'})$ to (M', n')-Fourier shapes, and similarly we can compose this to design a G_2 that fools $(M', (n')^2)$-Fourier shapes. Composing this $t = O(\log \log n)$-times, we obtain G_t which fools $O(M, n)$ Fourier shapes for any M, n. As a base case (to define the PRG G_0), the PRG of [NZ96] is used, which we have already discussed can be evaluated on-line in space linear in the seed required to store it and time polynomial in the length of the seed.

Now we observe an important property of this recursion. At every step of the recursion, one is tasked with computing the j-th coordinate output by some PRG for some j, and the result will depend *only* on a query for the j'-th coordinate of another PRG for some j' (as well as some additional values which are computed using the portion of the random seed dedicated to this step in the recursion). Thus at every step of the recursion, only one query is made for a coordinate to a PRG at a lower level of the recursion. Thus the recursion is linear, in the sense that the computation path has only L nodes instead of 2^L (which would occur if two queries to coordinate j', j'' were made to a PRG in a lower level). Since at each level of recursion, computing G^{6+7} itself uses $O(\log \log(nM))$ levels of recursion, and also has the property that each level queries the lower level at only one point, it follows that the total depth of the recursion is $O((\log \log(nM))^2)$. At each point, to store the information required for this recursion on the stack requires only $O(\log(nM))$-bits of space to store the relevant information identifying the instance of the PRG in the recursion, along with its associated portion of the random seed. Thus the total space required to compute a coordinate via these $O(\log \log(nM))^2)$ recursions is $O(\log(nM)(\log \log nM)^2)$, which is linear in the seed length. Moreover, the total time $\tilde{O}(1)$, since each step of the recursion requires $\tilde{O}(1)$. □

We use the prior technique to derandomize a wide class of linear sketches $\mathbf{A} \cdot x$ such that the entries of $\mathbf{A}$ are independent, and can be sampled using $O(\log(n))$-bit, and such that the behavior of the algorithm only depends on the sketch $\mathbf{A}x$. It is well known that there are strong connections between turnstile streaming algorithms and linear sketches, insofar as practically all turnstile streaming algorithms are in fact linear sketches. The equivalence of turnstile algorithms and linear sketches has even been formalized [LNW14b], with some restrictions. Our results show that all such sketches that use independent, efficiently sampled entries in their sketching matrix $\mathbf{A}$ can be derandomized with our techniques. As an application, we derandomize the count-sketch variant of Minton and Price [MP14], a problem which to the best of the authors knowledge was hitherto open.

Lemma 3.4.9. *Let* Alg *be any streaming algorithm which, on stream vector $x \in \{-M, \dots, M\}^n$ for some $M = \text{poly}(n)$, stores only a linear sketch $\mathbf{A} \cdot x$ such that the entries of the random matrix $\mathbf{A} \in \mathbb{R}^{k \times n}$ are i.i.d., and can be sampled using $O(\log(n))$-bits. Fix any constant $c \geq 1$. Then* Alg *can be implemented using a random matrix $\mathbf{A}'$ using $O(k \log(n)(\log\log n)^2)$ bits of space, such that for every vector $y \in \mathbb{R}^k$ with entry-wise bit-complexity of $O(\log(n))$,*

$$|\mathbf{Pr}\left[\mathbf{A}x = y\right] - \mathbf{Pr}\left[\mathbf{A}'x = y\right]| < n^{-ck}$$

Proof. We can first scale all entries of the algorithm by the bit complexity so that each entry in $\mathbf{A}$ is a $O(\log(n))$-bit integer. Then by Lemma 3.4.7, we can store the randomness needed to compute each entry of $\mathbf{A}'$ with $O(k \log(n)(\log\log n)^2)$-bits of space, such that $\mathbf{A}'$ n^{-ck}-fools the class of all $O(k)$-halfspace testers, in particular the one which checks, for each coordinate $i \in [k]$, whether both $(\mathbf{A}'x)_i < y + 1$ and $(\mathbf{A}'x)_i > y_i - 1$, and accepts only if both hold of all $i \in [k]$. By Proposition 3.4.8, the entries of $\mathbf{A}'$ can be computed in space linear in the size of the random seed required to store $\mathbf{A}'$. Since we have scaled all values to be integers, n^{-ck} fooling this tester is equivalent to the theorem statement. Note that the test $(\mathbf{A}'x)_i < y + 1$ can be made into a half-space test as follows. Let $X^i \in \mathbb{R}^{nk}$ be the vector such that $X^i_{j+(i-1)n} = x_j$ for all $j \in [n]$ and $X^i_j = 0$ otherwise. Let $\texttt{vec}(\mathbf{A}) \in \mathbb{R}^{nk}$ be the vectorization of $\mathbf{A}$. Then $(\mathbf{A}x)_i = \langle \texttt{vec}(\mathbf{A}), X^i \rangle$, and all the entries of $\texttt{vec}(\mathbf{A})$ are i.i.d., which allows us to make the stated constraints into the desired half-space constraints. □

Observe that the above Lemma derandomized the linear sketch $\mathbf{A}x$ by writing each coordinate $(\mathbf{A}x)_i$ as a linear combination of the random entries of $\texttt{vec}(\mathbf{A})$. Note, however, that the above proof would hold if we added the values of any $O(k)$ additional linear combinations $\langle X_j, \texttt{vec}(\mathbf{A})$ to the Lemma, where each $X_j \in \{-M, \dots, M\}^{kn}$. This will be useful, since the behavior of some algorithms, for instance count-sketch, may depend not only on the sketch $\mathbf{A}f$ but also on certain values or linear combinations of values within the sketch $\mathbf{A}$. This is formalized in the following Corollary.

Corollary 3.4.10. *Let the entries of $\mathbf{A} \in \mathbb{R}^{k \times n}$ be drawn i.i.d. from a distribution which can be sampled using $O(\log n)$-bits, and let $\texttt{vec}(\mathbf{A}) \in \mathbb{R}^{nk}$ be the vectorization of $\mathbf{A}$. Let $\mathbf{X} \in \mathbb{R}^{t \times nk}$ be any fixed matrix with entries contained within $\{-M, \dots, M\}$, where $M = \text{poly}(n)$. Then there is a distribution over random matrices $\mathbf{A}' \in \mathbb{R}^{k \times n}$ which can be generated and stored using $O(t \log(n)(\log\log n)^2)$ bits of space, such that for every vector $y \in \mathbb{R}^t$ with entry-wise*

bit-complexity of $O(\log(n))$*,*

$$|\mathbf{Pr}\left[\mathbf{X} \cdot \mathtt{vec}(\mathbf{A}) = y\right] - \mathbf{Pr}\left[\mathbf{X} \cdot \mathtt{vec}(\mathbf{A}') = y\right]| < n^{-ct}$$

Proof. The proof is nearly identical to Lemma 3.4.9, where we first scale entries to be $O(\log(n))$-bit integers, and then apply two half-space tests to each coordinate of $\mathbf{X} \cdot \mathtt{vec}(\mathbf{A}')$. □

Theorem 17. *Let* Alg *be any streaming algorithm which, on stream vector* $x \in \{-M, \ldots, M\}^n$ *and fixed matrix* $\mathbf{X} \in \mathbb{R}^{t \times nk}$ *with entries contained within* $\{-M, \ldots, M\}$*, for some* $M = \mathrm{poly}(n)$*, outputs a value that only depends on the sketches* $\mathbf{A} \cdot x$ *and* $\mathbf{X} \cdot \mathtt{vec}(\mathbf{A})$*. Assume that the entries of the random matrix* $\mathbf{A} \in \mathbb{R}^{k \times n}$ *are i.i.d. and can be sampled using* $O(\log(n))$*-bits. Let* $\sigma : \mathbb{R}^k \times \mathbb{R}^t \to \{0, 1\}$ *be any tester which measures the success of* Alg*, namely* $\sigma(\mathbf{A}x, \mathbf{X} \cdot \mathtt{vec}(\mathbf{A})) = 1$ *whenever* Alg *succeeds. Fix any constant* $c \geq 1$*. Then* Alg *can be implemented using a random matrix* $\mathbf{A}'$ *using a random seed of length* $O((k+t)\log(n)(\log\log n)^2)$*, such that:*

$$|\mathbf{Pr}\left[\sigma(\mathbf{A}x, \mathbf{X} \cdot \mathtt{vec}(\mathbf{A})) = 1\right] - \mathbf{Pr}\left[\sigma(\mathbf{A}'x, \mathbf{X} \cdot \mathtt{vec}(\mathbf{A}')) = 1\right]| < n^{-c(k+t)}$$

and such that each entry of $\mathbf{A}'$ *can be computed in time* $\tilde{O}(1)$ *and using working space linear in the seed length.*

Proof. As in the Lemma 3.4.9, we first scale all entries of the algorithm by the bit complexity so that each entry in $\mathbf{A}$ is $O(\log(n))$-bit integer. Then there is a $M' = \mathrm{poly}(n)$ such that each entry of $\mathbf{A} \cdot x$ and $\mathbf{X} \cdot \mathtt{vec}(\mathbf{A})$ will be a integer of magnitude at most M'. First note that the sketch $\mathbf{A} \cdot x$ and $\mathbf{X} \cdot \mathtt{vec}(\mathbf{A})$ can be written as one linear sketch $\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A})$ where $\mathbf{X}_0 \in \mathbb{R}^{k+t \times kn}$. Then σ can be written as a function $\sigma : \mathbb{R}^{k+t} \to \{0, 1\}$ evaluated on $\sigma(\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A}))$. Let $S = \{y \in \{-M', \ldots, M'\}^{k+t} \mid \sigma(y) = 1\}$. Then by Corollary 3.4.10, we have

$$|\mathbf{Pr}\left[\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A}) = y\right] - \mathbf{Pr}\left[\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A}') = y\right]| < n^{-c(k+t)}$$

for all $y \in S$. Taking c sufficiently large, and noting $|S| = n^{-O(k+t)}$, we have

$$\begin{aligned}
\mathbf{Pr}\left[\sigma(\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A})) = 1\right] &= \sum_{y \in S} \mathbf{Pr}\left[\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A}) = y\right] \\
&= \sum_{y \in S} (\mathbf{Pr}\left[\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A}') = y\right] \pm n^{-c(k+t)}) \qquad (3.8) \\
&= \mathbf{Pr}\left[\sigma(\mathbf{X}_0 \cdot \mathtt{vec}(\mathbf{A}')) = 1\right] + n^{-O(k+t)}
\end{aligned}$$

as desired. The final claim follows from Proposition 3.4.8. □

Derandomizing the Count-Sketch of Minton and Price

We now show how this general derandomization procedure can be used to derandomize the count-sketch variant of Minton and Price [MP14]. Our discussion will utilize the notation for count-sketch as defined in Section 3.2.1. Minton and Price's analysis shows improved concentration bounds for count-sketch when the random signs $g_i(k) \in \{1, -1\}$ are fully independent. They demonstrate that in this setting, if $y \in \mathbb{R}^n$ is the count-sketch estimate of a stream vector x, where the count-sketch table $\mathbf{A}$ has k columns and d rows, then for any $t \leq d$ and index $i \in [n]$ we have:

$$\mathbf{Pr}\left[(x_i - y_i)^2 > \frac{t}{d}\frac{\|x_{\text{tail}(k))}\|_2^2}{k}\right] \leq 2e^{-\Omega(t)}$$

Notice that by setting $t = d = \Theta(\log 1/\delta)$, one recovers the standard count-sketch result of Theorem 9. However, in order to apply this algorithm in $o(n)$ space, one must first derandomize it from using fully independent random signs, which are not required for the original count-sketch of [CCFC02b]. To the best of the authors' knowledge, the best known derandomization procedure was a black-box application of Nisan's PRG which results in $O(\epsilon^{-2}\log^3(n))$-bits of space when $k = O(1/\epsilon^2)$ and $d = O(\log n)$. Due to this $\log n$ blow-up in the space, the guarantees of this count-sketch variant, if derandomized with Nisan's PRG, are strictly worse than using the original count sketch of [CCFC02b]. Our derandomization, in contrast, demonstrates a strong improvement on this, obtaining the same bounds with an $(\log\log n)^2$ instead of an $\log n$ factor blowup. For the purposes of the theorem, we replace the notation $1/\epsilon^2$ with k (the number of columns of count-sketch up to a constant).

Theorem 18. *The count-sketch variant of [MP14] can be implemented so that if $\mathbf{A} \in \mathbb{R}^{d\times k}$ is a count-sketch table, then for any $t \leq d$ and index $i \in [n]$ we have:*

$$\mathbf{Pr}\left[(x_i - y_i)^2 > \frac{t}{d}\frac{\|f_{tail(k))}\|_2^2}{k}\right] \leq 2e^{-\Omega(t)}$$

and such that the total space required is $O(kd\log(n)(\log\log n)^2)$.

Proof. We first remark that the following modifcation to the count-sketch procedure does not effect the analysis of [MP14]. Let $\mathbf{A} \in \mathbb{R}^{d\times k}$ be a $d \times k$ count-sketch matrix. The modification is as follows: instead of each variable $h_i(\ell)$ being uniformly distributed in $\{1, 2, \ldots, k\}$, we replace them with variables $h_{i,j,\ell} \in \{0, 1\}$ for $(i, j, \ell) \in [d] \times [k] \times [n]$, such that $h_{i,j,\ell}$ are all i.i.d. and

equal to 1 with probability $1/k$. We also let $g_{i,h,\ell} \in \{1,-1\}$ be i.i.d. Rademacher variables (1 with probability $1/2$). Then $\mathbf{A}_{i,j} = \sum_{\ell=1}^{n} x_\ell g_{i,j,\ell} h_{i,j,\ell}$, and the estimate y_ℓ of f_ℓ for $\ell \in [n]$ is given by:

$$y_\ell = \text{median}\{g_{i,j,\ell}\mathbf{A}_{i,j} \mid h_{i,j,\ell} = 1\}$$

Thus the element x_ℓ can be hashed into multiple buckets in the same row of $\mathbf{A}$, or even be hashed into none of the buckets in a given row. By Chernoff bounds, $|\{g_{i,j,\ell}\mathbf{A}_{i,j} \mid h_{i,j,\ell} = 1\}| = \Theta(d)$ with high probability for all $\ell \in [n]$. Observe that the marginal distribution of each bucket is the same as the count-sketch used in [MP14], and moreover seperate buckets are fully independent. The key property used in the analysis of [MP14] is that the final estimator is a median over estimators whose error is independent and symmetric, and therefore the bounds stated in the theorem still hold after this modification [Pri18].

Given this, the entire sketch stored by the streaming algorithm is $\mathbf{B} \cdot x$, where

$$\mathbf{B}_j = \begin{cases} 1 & \text{with prob } \frac{1}{2k} \\ -1 & \text{with prob } \frac{1}{2k} \\ 0 & \text{otherwise} \end{cases}$$

Thus the entries of $\mathbf{B}$ are i.i.d., and can be sampled with $O(\log(k)) \leq O(\log(n))$ bits, and $\texttt{vec}(\mathbf{A}) = \mathbf{B} \cdot x$, where $\texttt{vec}(\mathbf{A})$ is the vectorization of the count-sketch table $\mathbf{A}$. Here $\mathbf{B} \in \mathbb{R}^{dk \times n}$.

Now note that for a fixed i, to test the statement that $(x_i - y_i)^2 > \frac{t}{d}\frac{\|x_{\text{tail}(k))}\|_2^2}{k}$, one needs to know both the value of the sketch $\mathbf{B}x$, in addition to the value of the i-th column of $\mathbf{B}$, since the estimate can be written as $y_i = \text{median}_{j \in [kd], \mathbf{B}_{j,i} \neq 0}\{\mathbf{B}_{j,i} \cdot (\mathbf{B}x)_j\}$. Note that the i-th column of $\mathbf{B}$ (which has kd entries) can simply be written as a sketch of the form $\mathbf{X} \cdot \texttt{vec}(\mathbf{B})$, where $\mathbf{X} \in \mathbb{R}^{kd \times dkn}$ is a fixed matrix such that $\mathbf{X} \cdot \texttt{vec}(\mathbf{B}) = \mathbf{B}_i$, so we also need to store $\mathbf{X} \cdot \texttt{vec}(\mathbf{B})$ Thus by Theorem 17, the algorithm can be derandomized to use $O(kd \log(n)(\log\log n)^2)$ bits of space, and such that for any $t \leq d$ and any $i \in [n]$ we have $\mathbf{Pr}\left[(x_i - y_i)^2 > \frac{t}{d}\frac{\|x_{\text{tail}(k))}\|_2^2}{k}\right] \leq 2e^{-\Omega(t)} \pm n^{-\Omega(dk)}$.

□

Derandomizing the L_p Sampling Algorithm

We now introduce the notation which will be used in our derandomization.The L_p sampler uses two sources of randomness which we must construct PRG's for. The first, r_e, is the randomness

needed to construct the exponential random variables t_i, and the second, r_c, is the randomness needed for the fully random hash functions and signs used in count-max. Note that r_e, r_c both require $\text{poly}(n)$ bits by Lemma 3.4.3. From here on, we will fix any index $i \in [n]$. In what follows, the symbols u, v will denote random seeds given as input to PRG's. The L_p sampler can then be thought of as a tester $\mathcal{A}(r_e, r_c) \in \{0, 1\}$, which tests on inputs r_e, r_c, whether the algorithm will output $i \in [n]$. Let $G_1(u)$ be Nisan's PRG, and let $G_2(v)$ be the half-space PRG. For two values $b, c \in \mathbb{R}$, we write $a \sim_\epsilon b$ to denote $|a - b| < \epsilon$. Our goal is to show that

$$\mathbf{Pr}_{r_e, r_c}\Big[\mathcal{A}(r_e, r_c)\Big] \sim_{n^{-c}} \mathbf{Pr}_{u,v}\Big[\mathcal{A}(G_2(v), G_1(u))\Big]$$

where u, v are seeds of length at most $O(\log^2 n(\log\log n)^2)$, and c is an arbitrarily large constant.

Theorem 19. *A single instance of the algorithm* L_p-Sampler *using* Fast-Update *as its update procedure can be derandomized using a random seed of length* $O(\log^2(n)(\log\log n)^2)$*, and thus can be implemented in this space. Moreover, this does not affect the time complexity as stated in Lemma 3.4.3.*

Proof. First note that by Lemma 3.4.3, we require $\tilde{O}(\epsilon^{-1})$ random bits for each $i \in [n]$, and thus we require a total of $\tilde{O}(n\epsilon^{-1}) = \text{poly}(n)$ random bits to be generated. Since Nisan's PRG requires the tester to read its random input in a stream, we can use a standard reordering trick of the elements of the stream, so that all the updates to a given coordinate $i \in [n]$ occur at the same time (see [Ind06]). This does not effect the output distribution of our algorithm, since linear sketches do not depend on the ordering of the stream. Now let c' be the constant such that the algorithm L_p-Sampler duplicates coordinates $n^{c'}$ times. In other words, the count-max is run on the stream vector $\zeta \in \mathbb{R}^{n^{c'}}$, and let $N = n^{c'}$. Now, as above, we fix any index $i \in [N]$, and attempt to fool the tester which checks if, on a given random string, our algorithm would output i. For any fixed randomness r_e for the exponentials, let $\mathcal{A}_{r_e}(r_c)$ be the tester which tests if our L_p sampler would output the index i, where now the bits r_e are hard-coded into the tester, and the random bits r_c are taken as input and read in a stream. We first claim that this tester can be implemented in $O(\log(n))$-space.

To see this, note that $\mathcal{A}_{r_e}(r_c)$ must simply count the number of rows of count-max such that item i is hashed into the largest bucket (in absolute value) of that row, and output 1 if this number is at least $\frac{4d}{5}$, where d is the number of rows in count-max. To do this, $\mathcal{A}_{r_e}(r_c)$ can break r_c into d blocks of randomness, where the j-th block is used only for the j-th row of count-max. It can then fully construct the values of the counters in a row, one row at a time, reading the bits of r_c in a stream. To build a bucket, it looks at the first element of the stream, uses r_c to find the

bucket it hashes to and the Gaussian scaling it gets, then adds this value to that bucket, and then continues with the next element. Note that since r_e is hardcoded into the tester, we can assume the entire stream vector ζ is hardcoded into the tester. Once it constructs a row of count-max, it checks if i is in the largest bucket by absolute value, and increments a $O(\log(d))$-bit counter if so. Note that it can determine which bucket i hashes to in this row while reading off the block of randomness corresponding to that row. Then, it throws out the values of this row and the index of the bucket i hashed to in this row, and builds the next row. Since each row has $O(1)$ buckets, $\mathcal{A}_{r_e}(r_c)$ only uses $O(\log(n))$-bits of space at a time. Then using $G_1(u)$ as Nisan's generator with a random seed u of length $O(\log^2(n))$-bits, we have $\mathbf{Pr}\left[\mathcal{A}_{r_e}(r_c)\right] \sim_{n^{-c_0}} \mathbf{Pr}\left[\mathcal{A}_{r_e}(G_1(u))\right]$, where the constant c_0 is chosen to be sufficiently larger than the constant c_1 in the n^{-c_1} additive error of our perfect sampler, as well as the constant c' Moreover:

$$\begin{aligned}
\mathbf{Pr}\left[\mathcal{A}(r_e, r_c)\right] &= \sum_{r_e} \mathbf{Pr}\left[\mathcal{A}_{r_e}(r_c)\right] \mathbf{Pr}\left[r_e\right] \\
&= \sum_{r_e} ((\mathbf{Pr}\left[\mathcal{A}_{r_e}(G_1(u))\right] \pm n^{-c_0}) \mathbf{Pr}\left[r_e\right] \\
&= \sum_{r_e} (\mathbf{Pr}\left[\mathcal{A}_{r_e}(G_1(u))\right] \mathbf{Pr}\left[r_e\right] \pm \sum_{r_e} n^{-c_0} \mathbf{Pr}\left[r_e\right] \\
&\sim_{n^{-c_0}} \mathbf{Pr}\left[\mathcal{A}(r_e, G_1(u))\right]
\end{aligned}$$

Now fix any random seed u for input to Nisan's PRG, and consider the tester $\mathcal{A}_{G_1(u)}(r_e)$, which on fixed count-max randomness $G_1(u)$, tests if the algorithm will output $i \in [n]$ on the random input r_e for the exponential variables.

We first observe that it seems unlikely that $\mathcal{A}_{G_1(u)}(r_e)$ can be implemented in $\log(n)$ space while reading its random bits r_e in a stream. This is because each row of count-max depends on the same random bits in r_e used to construct the exponentials t_i, thus it seems $\mathcal{A}_{G_1(u)}(r_e)$ would need to store all $\log^2(n)$ bits of count-max at once. However, we will now demonstrate that $\mathcal{A}_{G_1(u)}(r_e)$ is in fact a $\mathrm{poly}(n)$ bounded $O(d)$-halfspace tester (as defined earlier in this section) where d is the number of rows of count-max, and therefore can be derandomized with the PRG of [GKM18].

By the Runtime & Random bits analysis in Lemma 3.4.3, it suffices to take all random variables in the algorithm to be $O(\log(n))$-bit rational numbers. Scaling by a sufficiently large $\mathrm{poly}(n)$, we can assume that $1/t_j^{1/p}$ is a discrete distribution supported on $\{-T, \ldots, T\}$ where $T \leq \mathrm{poly}(n)$ for a sufficiently large $\mathrm{poly}(n)$. We can then remove all values in the support which

occur with probability less than $\mathrm{poly}(n)$, which only adds an a n^{-c_0} additive error to our sampler. After this, the distribution can be sampled from with $\mathrm{poly}(T) = \mathrm{poly}(n)$ random bits, which is as needed for the setting of Lemma 3.4.7. Note that we can also apply this scaling the Gaussians in count-max, so that they too are integers of magnitude at most $\mathrm{poly}(n)$.

Given this, the distribution of the variables $1/t_j^{1/p}$ satisfy the conditions of Lemma 3.4.7, in particular being $\mathrm{poly}(n)$-bounded, thus we must now show that $\mathcal{A}_{G_1(u)}(r_e)$ is indeed a $O(d)$-halfspace tester, with integer valued half-spaces bounded by $\mathrm{poly}(n)$. First consider a given row of count-max, and let the buckets be B_1, B_2. WLOG i hashes into B_1, and we must check if $|B_1| > |B_2|$. Let g_j be the random count-max signs (as specified by $G_1(u)$), and let $S_1, S_2 \subset [n]$ be the set of indices which hash to B_1 and B_t respectively. We can run the following 6 half-space tests to test if $|B_1| > |B_2|$:

$$\sum_{j \in S_1} g_j \frac{F_j}{t_j^{1/p}} > 0 \tag{3.9}$$

$$\sum_{j \in S_2} g_j \frac{F_j}{t_j^{1/p}} > 0 \tag{3.10}$$

$$a_1 \sum_{j \in S_1} g_j \frac{F_j}{t_j^{1/p}} + a_2 \sum_{j \in S_2} g_j \frac{F_j}{t_j^{1/p}} > 0 \tag{3.11}$$

where a_1, a_2 range over all values in $\{1, -1\}^2$, and recall that F is the duplicated vector constructed in the procedure $\mathrm{L_p}$-Sampler. The tester can decide whether $|B_1| > |B_2|$ by letting a_1 be the truth value (where -1 is taken as fail) of the first test 3.9 and a_2 the truth value of 3.10. It then lets $b_2 \in \{0, 1\}$ be the truth value of 3.11 on the resulting a_1, a_2 values, and it can correctly declare $|B_1| > |B_2|$ iff $b_2 = 1$. Thus for each pair of buckets, the tester uses 6 halfspace testers to determine if $|B_1| > |B_2|$, and so can determine if i hashed to the max bucket with $O(1)$ halfspace tests. So $\mathcal{A}_{G_1(u)}(r_e)$ can test if the algorithm will output i by testing if i hashed to the max bucket in a $4/5$ fraction of the d rows of count-max, using $O(d) = O(\log(n))$ halfspace tests.

Note that by the scaling performed in the prior paragraphs, all coefficents of these half-spaces are integers of magnitude at most $\mathrm{poly}(n)$. So by Lemma 3.4.7, the PRG $G_2(v)$ of [GKM18] fools $\mathcal{A}_{G_1(u)}(r_e)$ with a seed v of $O(\log^2(n)(\log\log n)^2)$-bits. So

$$\mathbf{Pr}\left[\mathcal{A}_{G_1(u)}(r_e)\right] \sim_{n^{-c_0}} \mathbf{Pr}\left[\mathcal{A}_{G_1(u)}(G_2(v))\right]$$

and so by the same averaging argument as used in for the Nisan PRG above, we have

$$\mathbf{Pr}\left[\mathcal{A}(r_e, G_1(u))\right] \sim_{n^{-c_0}} \mathbf{Pr}\left[\mathcal{A}(G_2(v), G_1(u))\right]$$

and so $\mathbf{Pr}\left[\mathcal{A}(r_e, r_c)\right] \sim_{n^{-c_0}} \mathbf{Pr}\left[\mathcal{A}(G_2(v), G_1(u))\right]$ as desired. Now fixing any $i \in [n]$, let $\mathcal{A}'(r_e, r_c)$ be the event that the overall algorithm outputs the index i. In other words, $\mathcal{A}'_i(r_e, r_c) = 1$ if $\mathcal{A}_{i_j}(r_e, r_c) = 1$ for some $j \in [n^{c'-1}]$, where $\mathcal{A}_{i_j}(r_e, r_c) = 1$ is the event that count-max declares that i_j is the maximum in Algorithm $\mathrm{L_p}$-Sampler. Thus, the probability that the algorithm outputs a non-duplicated coordinate $i \in [n]$ is given by:

$$\begin{aligned}
\mathbf{Pr}\left[\mathcal{A}'_i(r_e, r_c)\right] &= \sum_{j=1}^{n^{c'}} \mathbf{Pr}\left[\mathcal{A}_{i_j}(r_e, r_c)\right] \\
&= \sum_{j=1}^{n^{c'}} \mathbf{Pr}\left[\mathcal{A}_{i_j}(G_2(v), G_1(u))\right] \pm n^{-c_0} \\
&= \mathbf{Pr}\left[\mathcal{A}'_i(G_2(v), G_1(u))\right] \pm n^{-c_1}
\end{aligned} \tag{3.12}$$

where in the last line we set $c_0 > c' + c_1$, where recall c_1 is the desired additive error in our main sampler. In conclusion, replacing the count-max randomness with Nisan's PRG and the exponential random variable randomness with the half-space PRG $G_2(v)$, we can fool the algorithm which tests the output of our algorithm with a total seed length of $O(\log^2(n)(\log\log n)^2)$.

To show that the stated update time of Lemma 3.4.3 is not affected, we first remark that Nisan's PRG simply involves performing $O(\log(n))$ nested hash computations on a string of length $O(\log(n))$ in order to obtain any arbitrary substring of $O(\log(n))$ bits. Thus the runtime of such a procedure is $\tilde{O}(1)$ to obtain the randomness needed in each update of a coordinate $i \in [n^c]$. By Lemma 3.4.7, the PRG of [GKM18] requires $\tilde{O}(1)$ time to sample the $O(\log(n))$-bit string needed to generate an exponential, and moreover can be computed with working space linear in the size of the random seed (note that this is also true of Nisan's PRG, which just involves $O(\log(n))$-nested hash function computations). Thus the update time is only blown up by a $\tilde{O}(1)$ factor, which completes the proof. □

Corollary 3.4.11. *Given any $\epsilon > n^{-c}$, for $p = 2$, the entire (ϵ, δ)-approximate L_p sampling algorithm can be derandomized to run using $O(\log^3 n \log \delta^{-1})$ bits of space. For $p < 2$, the algorithm can be derandomized to run using $O(\log^3 n)$-bits of space with $\delta = 1/\mathrm{poly}(n)$.*

Proof. We can simply derandomize a single instance of our sampling algorithm using Nisan's PRG as in Theorem 19, except that we derandomize all the randomness in the algorithm at once. Since such an instance requires $O(\log^2 n)$-bits of space, using Nisan's blows up the complexity to $O(\log^3 n)$ (the tester can simply simulate our entire algorithm in $O(\log^2 n)$-bits of space, reading the randomness in a stream by the reordering trick of [Ind06]). Since the randomness for

separate parallel instances of the main sampling algorithm is disjoint and independent, this same $O(\log^2 n)$-bit tester can test the entire output of the algorithm by testing each parallel instance one by one, and terminating on the first instance that returns an index $i \in [n]$. Thus the same $O(\log^3 n)$-bit random seed can be used to randomize all parallel instances of our algorithm.

For $p < 2$, we can run $O(\log n)$ parallel instances to get $1/\text{poly}(n)$ failure probability in $O(\log^3 n)$-bits of space as stated. For $p = 2$, we can run $O(\log n \log \delta^{-1})$ parallel repetitions needed to get δ failure probability using the same random string, for a total space of $O(\log^3 n \log \delta^{-1} + \log^3 n) = O(\log^3 n \log \delta^{-1})$ as stated. As noted in the proof of Theorem 19, computing a substring of $O(\log n)$-bits from Nisan's PRG can be done in $\tilde{O}(1)$ time and using space linear in the seed length, which completes the proof. □

3.4.3 Query Time

We will now show the modifications to our algorithm necessary to obtain $\tilde{O}(1)$ query time. Recall that the algorithm must maintain a count-max data $\mathbf{A}$. Thealgorithm then searches over all indices $i \in \mathcal{K}$ to check if i hashed into the maximum bucket in a row of $\mathbf{A}$ at least a $4/5$ fraction of the time. Since $|\mathcal{K}| = \tilde{O}(n)$, running this procedure requires $\tilde{O}(n)$ time to produce an output on a given query. To avoid this, and obtain $\tilde{O}(1)$ running time, we will utilize the heavy hitters algorithm of [LNNT16], which has $\tilde{O}(1)$ update and query time, and which does not increase the complexity of our algorithm.

Theorem 20 ([LNNT16])**.** *For any precision parameter $0 < \nu < 1/2$, given a general turnstile stream $x \in \mathbb{R}^n$ there is an algorithm,* ExpanderSketch*, which with probability $1 - n^{-c}$ for any constant c, returns a set $S \subset [n]$ of size $S = O(\nu^{-2})$ which contains all indices i such that $|x_i| \geq \nu \|x\|_2$. The update time is $O(\log n)$, the query time is $\tilde{O}(\nu^{-2})$, and the space required is $O(\nu^{-2} \log^2 n)$-bits.*

Using `ExpanderSketch` to speed up query time. The modifications to our main algorithm L_p-Sampler with Fast-Update are as follows. We run our main algorithm as before, maintaining the same count-max data structures $\mathbf{A}$. Upon initialization of our algorithm, we also initialize an instance ExSk of ExpanderSketch as in Theorem 20, with the precision parameter $\nu = 1/100$.

Now recall in our Fast-Update procedure, for each $i \in [n]$ we hash the top $K_i = O(\log(n))$ largest duplicates ζ_{i_j} corresponding to x_i individually, and store the random variables $h_\ell(i_j)$ that determine which buckets in A they hash to. While processing updates to our algorithm at this point, we make the modification of *additionally* sending these top K_i items to ExSk to be

sketched. More formally, we run ExSk on the stream $\zeta_{\mathcal{K}}$, where $\zeta_{\mathcal{K}}$ is the vector ζ projected onto the coordinates of $\mathcal{K}$. Since $K_i = \tilde{O}(1)$, this requires making $\tilde{O}(1)$ calls to update ExSk on different coordinates, which only increases our update time by an $\tilde{O}(1)$ additive term.

On termination, we obtain the set S containing all items ζ_i such that $i \in \mathcal{K}$ and $\zeta_i \geq \frac{1}{100}\|\zeta_{\mathcal{K}}\|_2$. Instead of searching through all coordinates of $\mathcal{K}$ to find a maximizer, we simply search through the coordinates in S, which takes $\tilde{O}(|S|) = \tilde{O}(1)$ time. We now argue that the output of our algorithm does not change with these new modifications. We refer collectively to the new algorithm with these modifications as $\mathrm{L_p}$-Sampler with Fast-Update and ExSk, and the algorithm of Section 3.4.1 as simply $\mathrm{L_p}$-Sampler with Fast-Update.

Lemma 3.4.12. *For any constant $c > 0$, with probability $1 - n^{-100c}$ the algorithm $\mathrm{L_p}$-Sampler with Fast-Update and ExSk as described in this section returns the same output (an index $i \in [n]$ or $\perp$) as $\mathrm{L_p}$-Sampler using Fast-Update but without ExSk. The space and update time are not increased by using ExSk, and the query time is now $\tilde{O}(1)$.*

Proof. We condition on the event that S contains all items i such that $i \in \mathcal{K}$ and $|\zeta_i| \geq 1/100\|\zeta_{\mathcal{K}}\|_2$, which occurs with probability $1 - n^{-100c}$ by Theorem 20. Since $\mathrm{L_p}$-Sampler already uses at least $O(\log^2 n)$ bits of space, the additional $O(\log^2 n)$ bits of overhead required to run an instance ExSk of `ExpanderSketch` with sensitivity parameter $\nu = 1/100$ does not increase the space complexity. Furthermore, as mentioned above, the update time is blown-up by a factor of $\tilde{O}(1)$, since we make $K_i = \tilde{O}(1)$ calls to update ExSk, which has an update time of $\tilde{O}(1)$ by Theorem 20. Furthermore, our algorithm does not require any more random bits, as it only uses `ExpanderSketch` as a subroutine, and thus no further derandomization is required. Thus the complexity guarantees of Lemma 3.4.3 are unchanged.

For the query time, we note that obtaining S requires $\tilde{O}(1)$ time (again by Theorem 20), and querying each of the $|S| = O(1)$ items in our count-max $\mathbf{A}$ requires $\tilde{O}(1)$ time. To complete the proof, we now consider the output of our algorithm. Since we are searching through a strict subset $S \subset [n^c]$, it suffices to show that if the original algorithm output an $i_j \in [n^c]$, then so will we. As argued in Lemma 3.4.3, such a coordinate must be contained in $\mathcal{K}$. By Corollary 3.2.2, we must have

$$|\zeta_{i_j}| > \frac{1}{100}\|\zeta\|_2 \geq \frac{1}{100}\|\zeta_{\mathcal{K}}\|_2$$

with probability $1 - n^{-100c}$ (scaling c by 100 here), thus $i_j \in S$, which completes the proof.

□

3.5 Approximating the Sampled Coordinate

In this section, we will show how given a precision parameter $\nu > 0$, conditioned on our algorithm $\mathrm{L_p}$-Sampler returning a sampled index $i \in [n]$, we can obtain an estimate $\tilde{x}_i = (1 \pm \nu)x_i$ with probability $1 - \delta_2$. Given a failure parameter $\delta_2 > 0$, we set $d' = O(\log \delta_2^{-1})$ and

$$\gamma = O\left(\min\left\{\nu^{-2},\ \nu^{-p}\log\left(\frac{1}{\delta_2}\right)\right\}\right)$$

Then our algorithm, in addition to the count-max matrix $\mathbf{A}$ used by $\mathrm{L_p}$-Sampler, stores a count-sketch matrix $\mathbf{A}' \in \mathbb{R}^{d\times\gamma}$ with d rows and γ columns.

Recall in the Fast-Update procedure, for each $i \in [n]$ we hash the top $K_i = O(\log(n))$ largest duplicates ζ_{i_j} corresponding to x_i individually into A, and store the random variables $h_\ell(i_j)$ that determine which buckets in $\mathbf{A}$ they hash to. Thus if count-max outputs an $i_j \in [n^c]$ we know that $i_j \in \mathcal{K}$, where $\mathcal{K} = \cup_{i\in[n]}\cup_{j=1}^{K_i}\{i_j\}$ as in Section 3.4 (since our algorithm only searches through $\mathcal{K}$ to find a maximizer). Thus it suffices to run the count-sketch instance $\mathbf{A}'$ on the stream $\zeta_{\mathcal{K}}$, where $\zeta_{\mathcal{K}}$ is the vector ζ with the coordinates not in $\mathcal{K}$ set to 0. Since $K_i = \tilde{O}(1)$, we perform at most $\tilde{O}(1)$ updates to count-sketch at every step in the stream. This requires making $\tilde{O}(1)$ calls to update count-sketch on each stream update, which only increases our update time by an $\tilde{O}(1)$ additive term.

Now if $\mathrm{L_p}$-Sampler returns $i_j \in [n^c]$ (corresponding to some duplicate i_j of i), then we must have $i_j \in \mathcal{K}$. Thus we can query $\mathbf{A}'$ for a value $\tilde{y}_{i_j}$ such that

$$|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\mathrm{tail}(\gamma)}\|_2$$

with probability $1 - \delta_2$ by Theorem 9. Furthermore, since $i_j \in \mathcal{K}$, we can compute the value I_k such that $I_k = (\mathtt{rnd}_\epsilon(1/t_{i_j}))$ by simulating the Fast-Update procedure on an update to i. We will argue that the estimate $\tilde{x} = \tilde{y}_{i_j}(\mathtt{rnd}_\epsilon(1/t_{i_j}^{1/p}))^{-1}$ satisfies $\tilde{x} = (1 \pm \nu)x_i$. Putting this together with Theorem 13, we will obtain the following result.

Theorem 21. *Given parameters $\epsilon, \delta_1, \delta_2, \nu \in (0, 1)$ and any constant $c > 0$, there is an algorithm $\mathcal{A}$ which is a $(\epsilon, \delta_1, n^{-c})$-approximate L_p sampler. Conditioned on the sampler outputting some index $i \in [n]$ (and not $\perp$), the algorithm $\mathcal{A}$ will then output $\tilde{x}$ such that*

$$(1 - \nu)x_i < \tilde{x} < (1 + \nu)x_i$$

with probability $1 - \delta_2$. *The space required is*

$$O\left(\left(\log^2 n(\log\log n)^2 + \beta \log(n) \log \delta_2^{-1}\right) \log \delta_1^{-1}\right)$$

for $p \in (0, 2)$, *and*

$$O\left(\left(\log^3 n + \nu^{-2} \log^2(n) \log \delta_2^{-1}\right) \log \delta_1^{-1}\right)$$

for $p = 2$, *where*

$$\beta = \min\left\{\nu^{-2},\ \nu^{-p} \log\left(\frac{1}{\delta_2}\right)\}\right\}$$

The update time is $\tilde{O}(\nu^{-1})$ *and the query time is* $\tilde{O}(1)$

Proof. We first consider the complexity. The first term in each of the upper bounds follows from Theorem 13, as well as the $\log \delta_1^{-1}$ term which comes from repeating the entire algorithm $\log \delta_1^{-1}$ times for $p < 2$, and $\log(n) \log \delta_1^{-1}$ times for $p = 2$. The second term in the space bound results from storing the $d' \times \gamma$ count-sketch table $\mathbf{A}'$, which is $O(\gamma \log(n) \log \delta_2^{-1})$ as stated. Moreover, the update time for the new data structure is at most $\tilde{O}(1)$, since the only additional work we do on each update is to hash $K_i = O(\log(n))$ items into $d' = O(\log(n))$ rows of $\mathbf{A}'$. Furthermore, the query time just requires computing a median of $O(\log(n))$ entries of $\mathbf{A}'$. Each of these actions is $\tilde{O}(1)$ time in the unit cost RAM model, so the additional update and query time is $\tilde{O}(1)$. The remaining $\tilde{O}(\nu)$ update time follows from Lemma 3.4.3.

For correctness, note that if $\mathrm{L_p}$-Sampler does not fail and instead outputs $i_j \in [n^c]$, we know that $|\zeta_{i_j}| > 1/100\|\zeta\|_2$. Furthermore, we have

$$|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\text{tail}(\gamma)}\|_2 \leq \sqrt{1/\gamma}\|\zeta\|_2$$

with probability $1 - \delta_2$, so setting $\gamma = \Theta(1/\nu^2)$ sufficiently large, it follows that $\tilde{y}_{i_j} = (1 \pm O(\nu))\zeta_{i_j}$. Then $\tilde{y}_{i_j}(\mathtt{rnd}_\epsilon(1/t_{i_j}^{1/p}))^{-1} = (1 \pm \nu)x_i$ follows immediately from the fact that $x_i = \zeta_{i_j}(\mathtt{rnd}_\epsilon(1/t_{i_j}^{1/p}))^{-1}$ (and a rescaling of ν by a constant). This shows that $O(\nu^{-2})$ bits is always an upper bound for the value of $\gamma = \Theta(\beta)$ needed for $p \in (0, 2]$.

To show the other upper bound in the definition of β (for cases when $p < 2$), first define $T_\gamma \subset [n^c]$ as the set of $n^c - \gamma$ smallest coordinates (in absolute value) of z. In other words $z_{T_\gamma} = z_{\text{tail}(\gamma)}$, where for any set $S \subseteq [n^c]$ z_S denotes z projected onto the coordinates of S. Note that if S is any set of size $n^c - s$ and $v \in \mathbb{R}^{n^c}$ any vector, we have $\|v_{\text{tail}(s)}\|_2 \leq \|v_S\|_2$. Then by

Proposition 3.3.5, using the fact that $\zeta_i = (1 \pm O(\nu))z_i$ for all $i \in [n^c]$, we have

$$\begin{aligned} \|\zeta_{\text{tail}(\gamma)}\|_2 &\leq \|\zeta_{T_\gamma}\|_2 \\ &\leq 2\|z_{\text{tail}(\gamma)}\|_2 \\ &= O(\|F\|_p(\gamma)^{-1/p+1/2}) \end{aligned} \tag{3.13}$$

for $p < 2$ with probability $1-O(e^{-\gamma}) > 1-\delta_2$, where now we are setting $\gamma = \Theta(\max\{\nu^{-p}, \log(1/\delta_2)\})$. Condition on this now. Then we obtain error

$$\begin{aligned} |\tilde{y}_{i_j} - \zeta_{i_j}| &< \sqrt{1/\gamma}\|\zeta_{\text{tail}(\gamma)}\|_2 \\ &= O(\|F\|_p\gamma^{-1/p}) \\ &= O(\nu(\log \delta_2^{-1})^{-1/p}\|F\|_p) \end{aligned} \tag{3.14}$$

from our second count-sketch $\mathbf{A}'$. Now $z_{D(1)} = \|F\|_p/E_1^{1/p}$, which is at least $\Omega(\|F\|_p/(\log \delta_2^{-1})^{1/p})$ with probability greater than $1-\delta_2$ using the pdf of an exponential. Conditioned on this, the error from our second count-sketch $\mathbf{A}'$ gives, in fact, a $(1 \pm O(\nu))$ relative error approximation of ζ_{i_j}, which is the desired result. Note that we conditioned only on our count-sketch giving the desired $|\tilde{y}_{i_j} - \zeta_{i_j}| < \sqrt{1/\gamma}\|\zeta_{\text{tail}(\gamma)}\|_2$ error, on $\|z_{\text{tail}(\gamma)}\|_2 = O(\|F\|_p(\gamma)^{-1/p+1/2})$, and on $E_1 = O(\log \delta_2^{-1})$, each of which holds with probability at least $1 - O(\delta_2)$, so the Theorem follows after a union bound.

□

3.6 Truly Perfect Sampling for General Measures

In this section, we provide a framework for *truly perfect sampling* for some measure function $G : \mathbb{R} \to \mathbb{R}^{\geq 0}$ such that $G(x) = G(-x)$, $G(0) = 0$ and G is non-decreasing in $|x|$. As we show in Section 3.7, any such algorithm in the turnstile model needs linear space. Thus, in this section, we will focus on the insertion only model of streaming, and develop algorithms for this model. By duplicating updates, one can assume that each update in the stream is of the form $(i, 1)$ for some $i \in [n]$. Therefore, we will abbrivate notation below, and think of an update as simply being a unit increment to some coordinate $i \in [n]$.

If we define $F_G = \sum_{i=1}^n G(x_i)$, recall from Definition 3.1.3 that a truly perfect G sampler outputs index $i \in [n]$ with probability *exactly* $\frac{G(x_i)}{F_G}$, conditioned on not outputting the failure symbol $\perp$ (which it can do with probability at most δ). We then show how to apply the framework

to L_p sampling where $G(x) = |x|^p$ and to various M-estimators, such as the $L_1 - L_2$, Fair, Huber, and Tukey estimators.

3.6.1 Algorithmic Framework

Our algorithm is based on running parallel instances of a single sampler. Each instance uses $\log n$ bits of space, but only succeeds with small probability and thus we need to run multiple instances to ensure that some instance succeeds with sufficiently high probability. Each instance uses reservoir sampling to sample an item s and keeps a counter c of how many times s appears in the stream after it is sampled.

We first describe the Sampler algorithm. Given a stream of elements $u_1, \ldots, u_t$, where each $u_i \in [n]$, Sampler selects an index $j \in [t]$ uniformly at random and outputs u_j as well as the number of instances of u_j that appear after time j. The algorithm uses reservoir sampling to ensure that each item is selected with probability $\frac{1}{t}$. A counter is also maintained to track the number of instances of the sample. Each time a new sample replaces the existing sample in the reservoir sampling procedure, the counter is reset to zero.

Algorithm 1: Sampler: Reservoir sampling, counting number of times item has appeared afterwards.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector x.
Output: Sample each coordinate u_i with probability $\frac{1}{m}$ and output the number of occurrences that appears afterwards.
$s \leftarrow \emptyset$, $c \leftarrow 0$
for *each update* u_r **do**
    $s \leftarrow u_r$ with probability $\frac{1}{r}$
    if s *is updated to* u_r **then**
        $c \leftarrow 0$ //Reset counter
    if $u_r = s$ **then**
        $c \leftarrow c + 1$ //Increment counter
Sample s and counter c

By outputting s with probability $\frac{G(c)-G(c-1)}{\zeta}$, where ζ is a parameter such that $G(x) - G(x-1) \leq \zeta$ for all possible coordinates x in the y vector, i.e., $x \in \{x_1, \ldots, x_n\}$, then it can be shown by a telescoping argument that the probability of outputting each $i \in [n]$ is "corrected" to roughly $\frac{G(x_i)}{\zeta \cdot m}$, where m is the length of the stream. Hence if the sampler successfully outputs a coordinate, it follows the desired distribution.

Algorithm 2: Truly perfect G-sampler algorithm for insertion only streams.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n]$ represents a single update to a coordinate of the underlying vector f, a measure function G.
Initialize an instance of Sampler. Algorithm 1
for *each update* $u_t \in [n]$ **do**
    Update Sampler.
Let s be the sampled output of Sampler and let c be the number of times s has appeared afterwards.
Let ζ be a parameter such that $G(x) - G(x-1) \leq \zeta$ for all $x \geq 1$.
Return: s with probability $\frac{G(c+1)-G(c)}{\zeta}$.

Theorem 22. *Let G be a function such that $G(x) - G(x-1) \leq \zeta$ for all $x \geq 1$. Given a lower bound $\widehat{F_G}$ on $F_G = \sum_{i\in[n]} G(x_i)$, then there exists a truly perfect G sampler for an insertion-only stream that outputs $\perp$ with probability at most δ and uses $O\left(\frac{\zeta m}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space. Further, the time to process each update is $O(1)$ in expectation.*

Proof. The probability that s is the j-th particular instance of item i inside the stream is $\frac{1}{m}$. Since the number of instances of i appearing after j is $x_i - j$ then the probability that i is output is

$$\sum_{j=1}^{x_i} \frac{1}{m} \frac{G(x_i - j + 1) - G(x_i - j)}{\zeta} = \frac{G(x_i)}{\zeta m}.$$

We note that $G(x_i - j + 1) - G(x_i - j) \leq \zeta$ for all $j \in [x_i]$, so returning s with probability $\frac{G(c+1)-G(c)}{\zeta}$ is valid.

Hence the probability that some index is returned by Algorithm 2 is $\sum_{i\in[n]} \frac{G(x_i)}{\zeta m} = \frac{F_G}{\zeta m}$, where $F_G = \sum G(x_i)$. Thus by repeating the sampler $O\left(\frac{\zeta m}{F_G} \log \frac{1}{\delta}\right)$ times, the algorithm will output a sample s with probability at least $1 - \delta$. Although the algorithm does not actually have the value of F_G, given a lower bound $\widehat{F_G}$ on F_G, then it suffices to repeat the sampler $O\left(\frac{\zeta m}{\widehat{F_G}} \log \frac{1}{\delta}\right)$ times. Moreover, the sample s will output each index $i \in [n]$ with probability $\frac{G(x_i)}{F_G}$. Each instance only requires $O(\log n)$ bits to maintain the counter c, assuming $\log m = O(\log n)$. Thus the total space used is $O\left(\frac{\zeta m}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space. □

3.6.2 Applications to Specific Measures

In this section, we show various applications of Algorithm 2 in the streaming model. The main barrier to applying 22 to any arbitrary measure function G is obtaining a "good" lower bound $\widehat{F_G}$

to $F_G = \sum_{i\in[n]} G(x_i)$.

Truly Perfect L_p Sampling on Insertion-Only Streams

We first consider truly perfect L_p sampling, where $G(x) = |x|^p$, for $p \geq 1$. Note that reservoir sampling is already a perfect L_1 sampler for $p = 1$ and it uses $O(\log n)$ bits of space on a stream of length $m = \text{poly}(n)$. For $p \in (1, 2]$, we first require the following norm estimation algorithm on insertion-only streams.

We now introduce an algorithm that for truly perfect L_p sampling using the above framework. We first describe the case of $p = 2$. Before describing the algorithm, we first recall the MisraGries data structure for finding heavy-hitters.

Theorem 23 ([MG82])**.** *There exists a deterministic one-pass streaming algorithm* MisraGries *that uses $O\left(\frac{1}{\epsilon}\log m\right)$ space on a stream of length m and outputs a list L of size $\frac{1}{2\epsilon}$ that includes all items i such that $x_i > 2\epsilon m$. Moreover, the algorithm returns an estimate $\widehat{x_i}$ for each $i \in L$ such that $x_i - \epsilon m \leq \widehat{x_i} \leq x_i$.*

Although we could obtain a perfect L_p sampler using any L_p estimation algorithm that succeeds with high probability, by using the deterministic data structure from Theorem 23, we can further remove the additive $\gamma = \frac{1}{\text{poly}(n)}$ error in the sampler to obtain truly perfect samplers.

Theorem 24. *Fix any $p \in [1, 2]$ and value $\delta \in (0, 1)$. Then there exists a truly perfect L_p-sampling algorithm, namely a $(0, \delta, 0)$-L_p Sampler, which uses $O\left(n^{1-1/p}\log n \log \delta^{-1}\right)$ bits of space.*

Proof. By Theorem 23, using a single MisraGries data structure with $O\left(n^{1-1/p}\log n\right)$ allows us to obtain a number Z such that

$$\|x\|_\infty \leq Z \leq \|x\|_\infty + \frac{m}{n^{1-1/p}}.$$

Note that for $p \in [1, 2]$, we have $(x_i)^p - (x_i - 1)^p \leq 2\|x\|_\infty^{p-1}$ for any $i \in [n]$, so that $\zeta = 2Z^{p-1}$ induces a valid sampling procedure. Hence each instance outputs some index $i \in [n]$ with probability at least $\frac{F_p}{2Z^{p-1}m}$. If $\|x\|_\infty \geq \frac{m}{n^{1-1/p}}$, then we have $2Z \leq 4\|x\|_\infty \leq 4\|x\|_p$, so that

$$\frac{F_p}{2Z^{p-1}m} \geq \frac{F_p}{4L_p^{p-1}\cdot m} = \frac{L_p}{4F_1} \geq \frac{1}{4n^{1-1/p}}.$$

On the other hand, if $\|x\|_\infty \leq \frac{m}{n^{1-1/p}}$, then we have $2Z \leq \frac{4m}{n^{1-1/p}}$, so that

$$\frac{F_p}{2Z^{p-1}m} \geq \frac{F_p \cdot n^{(p-1)^2/p}}{4m^p} = \frac{F_p \cdot n^{(p-1)^2/p}}{4F_1^p} \geq \frac{F_p \cdot n^{(p-1)^2/p}}{4F_p \cdot n^{(p-1)}}$$
$$= \frac{1}{4n^{1-1/p}}$$

Therefore, the probability that an instance outputs some index $i \in [n]$ is at least $\frac{1}{4n^{1-1/p}}$, and it suffices to use $O\left(n^{1-1/p}\log(1/\delta)\right)$ such instances, with total space $O\left(n^{1-1/p}\log n\right)$ bits of space, so that a coordinate is returned with probability at least $1-\delta$. By Theorem 23, conditioned on an index being returned by the algorithm, the probability that each coordinate $i \in [n]$ is output is $\frac{x_i^p}{F_p}$.

Finally, we remark that the runtime of the algorithm can be optimized to constant time per update by storing a hash table containing a count and a list of offsets. Specifically, when item i is first sampled by some repetition of the algorithm, then we start counting the number of subsequent instances of i in the stream. If i is subsequently sampled by another independent instance of the reservoir sampling at some time t, then it suffices to store the value of the counter at the time t as an offset. This value does not change and can now be used to correctly recover the correct count of the number of instances of i after time t by subtracting this offset from the largest count. Finally, we can maintain a hash table with pointers to the head and tail of the list, so that when an item i is sampled, we can efficiently check whether that item is already being tracked by another repetition of the sampler. Hence the update time is $O(1)$ worst-case once the hash bucket for i is determined and $O(1)$ in expectation overall given the assignment of the bucket by the hash function. Finally, note that by design of the offsets, we can build the correct counters at the end of the stream to determine the corresponding sampling probabilities. □

Theorem 25. *Fix $p \in (0,1]$ and $\delta \in (0,1)$. Then there exists a truly perfect L_p-sampling algorithm, namely a $(0,\delta,0)$-L_p Sampler, which uses $O(m^{1-p}\log n \log \delta^{-1})$ bits of space.*

Proof. Note that for $p \in (0,1]$, we have $(x_i)^p - (x_i-1)^p \leq 1$ for any $i \in [n]$, so that $\zeta = 1$ induces a valid sampling procedure. Hence each instance outputs some index $i \in [n]$ with probability at least $\frac{F_p}{m} \geq \frac{1}{m^{1-p}}$. Therefore, it suffices to use $O(m^{1-p}\log \delta^{-1})$ such instances so that a coordinate is returned with probability at least $1-\delta$, with total space $O(m^{1-p}\log n \log \delta^{-1})$ bits of space. □

M-estimators on Insertion-Only Streams

We generalize the paradigm of Algorithm 2 to sampling from general statistical M-estimator distributions.

Corollary 3.6.1. *Fix any $\delta \in (0,1)$. Then there exist truly perfect G samplers for the insertion-only streaming model, namely $(0,\delta,0)$ G-Samplers, that use $O\left(\log n \log \frac{1}{\delta}\right)$ bits of space when G is the $L_1 - L_2$ estimator, the Fair estimator, or the Huber estimator.*

Proof. For the $L_1 - L_2$ estimator, we have $G(x) = 2\left(\sqrt{1+\frac{x^2}{2}} - 1\right)$ so that $G(x) - G(x-1) < 3$ for $x \geq 1$. Moreover, $G(x) > |x|$ so $F_G > m$. Hence by Theorem 22, there exists a perfect G sampler that uses $O(\log n)$ bits of space when G is the $L_1 - L_2$ estimator.

For the Fair estimator, we have $G(x) = \tau|x| - \tau^2 \log\left(1 + \frac{|x|}{\tau}\right)$ for some constant $\tau > 0$ so that $G(x) - G(x-1) < \tau$ for $x \geq 1$. Since $G(x) > \tau|x|$ and thus $F_G > \tau m$, then by Theorem 22, there exists a perfect G sampler that uses $O(\log n)$ bits of space when G is the Fair estimator.

For the Huber measure function, we have $G(x) = \frac{x^2}{2\tau}$ for $|x| \leq \tau$ and $G(x) = |x| - \frac{\tau}{2}$ otherwise, where $\tau > 0$ is some constant parameter. Hence, $G(x) - G(x-1) < 1$ and $G(x) > \frac{\tau}{2} \cdot m$, so there exists a perfect G sampler that uses $O(\log n)$ bits of space when G is the Huber estimator by Theorem 22. □

Matrix Norms on Insertion-Only Streams

We now consider the problem of sampling row $\mathbf{M}_{i,*}$ from a matrix $\mathbf{M} \in \mathbb{R}^{n \times d}$ with probability $\frac{G(\mathbf{M}_{i,*})}{F_G}$ for some non-negative function $G : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, where we define $F_G = \sum_{j \in [n]} G(\mathbf{M}_{j,*})$. We consider a insertion-only stream, where each update is a positive update to some coordinate of $\mathbf{M}$. Note that this is equivelant to the standard insertion only streaming model, thinking of $\mathbf{M}$ as a vector in $\mathbb{R}^{nd}$.

We generalize the approach of Algorithm 2 by first using reservoir sampling to sample an update to a coordinate c to a row r of $\mathbf{M}$. We then maintain a vector v that consists of all the updates to row r and choose to output r with probability $G(v + e_c) - G(v)$, where e_c represents the elementary vector in $\mathbb{R}^d$ with a single 1 in coordinate c and zeros elsewhere. The correctness of Algorithm 3 follows from a similar proof to that of Theorem 22.

Theorem 26. *Fix any non-negative function $G : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ satisfying $G(\vec{0}) = 0$. Let ζ be a parameter such that $G(x) - G(x - e_i) \leq \zeta$ for all $x \in \mathbb{R}^d_{\geq 0}$, $i \in [d]$. Given a lower bound $\widehat{F_G}$ on*

Algorithm 3: Truly perfect G-sampler algorithm for vectors and matrices in insertion only streams.

Input: A stream of updates $u_1, u_2, \ldots, u_m$, where each $u_i \in [n] \times [d]$ represents a single update to a coordinate of a underlying matrix $\mathbf{M}$, a measure function G.
Initialize an instance of Sampler. Algorithm 1
for *each update* $u_t \in [n] \times [d]$ **do**
    Update Sampler.
Let r be the row and c be the column sampled by Sampler and let v be the vector induced by the subsequent updates to row r.
Let ζ be a parameter such that $G(x) - G(x - e_i) \leq \zeta$ for all $x \geq \left(\mathbb{R}^{\geq 0}\right)^d$, $i \in [d]$.
Return: r with probability $\frac{G(v+e_c)-G(v)}{\zeta}$.

F_G, then there exists a truly perfect G sampler, which outputs $\perp$ with probability at most δ for an insertion-only streams. The space required is $O\left(\frac{\zeta dm}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits.

Proof. The probability that the update to (r, c) is the j-th particular update to row r inside the stream is $\frac{1}{m}$. Let x_j be the sum of the updates to row r after the j-th update and let e_{c_j} be the coordinate of row r incremented in the j-th update, so that the probability that i is output is

$$\begin{aligned}\sum_j \frac{1}{m} \frac{G(x_j + e_{c_j}) - G(x_j)}{\zeta} &= \sum_j \frac{1}{m} \frac{G(x_{j-1}) - G(x_j)}{\zeta} \\ &= \frac{G(\mathbf{M}_{i,*})}{\zeta m}\end{aligned}$$

where the final equality results from the observations that $x_0 = \mathbf{M}_{r,*}$ and that $G(\vec{0}) = 0$, since v must be the all zeros vector after the last update to row r. Thus conditioned on some row being output, the algorithm outputs each row $i \in [n]$ with probability $\frac{G(x_i)}{F_G}$. We again note that $G(x_j + e_{c_j}) - G(x_j) \leq \zeta$ for all $x \in \left(\mathbb{R}^{\geq 0}\right)^d$, $i \in [d]$, so returning r with probability $\frac{G(x_j+e_{c_j})-G(x_j)}{\zeta}$ is well-defined. Therefore, the probability that some row is returned by Algorithm 3 is $\sum_{i \in [n]} \frac{G(\mathbf{M}_{i,*})}{\zeta m} = \frac{F_G}{\zeta m}$, where $F_G = \sum G(\mathbf{M}_{i,*})$. By repeating the sampler $O\left(\frac{\zeta m}{F_G} \log \frac{1}{\delta}\right)$ times, the algorithm successfully outputs a sample s with probability at least $1 - \delta$.

We again note that although the algorithm does not know the value of F_G, it suffices to repeat the sampler $O\left(\frac{\zeta m}{\widehat{F_G}} \log \frac{1}{\delta}\right)$ times for some lower bound $\widehat{F_G}$ on F_G. Each instance only

requires $O(d \log n)$ bits to maintain the vector v, assuming $\log m = O(\log n)$ and each update is bounded by $\text{poly}(n)$, which can be expressed using $O(\log n)$ bits. Thus the total space used is $O\left(\frac{\zeta dm}{\widehat{F_G}} \log n \log \frac{1}{\delta}\right)$ bits of space. □

For example, when $G(x) = \sum_{i \in [d]} |x_i|$, then F_G is the $L_{1,1}$ norm. Then we have $F_G = m$, so that by Theorem 26, so we can sample a row $\mathbf{M}_{i,*}$ with probability proportional to its L_1 norm, using $O\left(d \log n \log \frac{1}{\delta}\right)$ bits of space. We can also apply Theorem 26 when $G(x) = \sqrt{\sum_{i \in [d]} x_i^2}$ is the L_2 norm of each row, so that F_G is the $L_{1,2}$ norm crucially used in many adaptive sampling techniques (see [MRWZ20b] and references therein).

3.7 Lower Bounds

In this section, we obtain a lower bound for providing relative error approximations $\tilde{x} = (1 \pm \nu) x_i$ of the frequency of a sampled item $i \in [n]$. Our lower bound is derived from one-way two-party communication complexity, which we now give a primer for.

Two-Party Communication. Let $\mathcal{X}, \mathcal{Y}$ be input domains to a two party communication complexity problem. Alice is given $x \in \mathcal{X}$ and Bob is given $y \in \mathcal{Y}$. Their goal is to solve some relational problem $Q \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, where for each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ the set $Q_{xy} = \{z | (x, y, z) \in Q\}$ represents the set of *correct* solutions to the communication problem.

In the one-way *communication protocol* $\mathcal{P}$, Alice must send a single message M to Bob (depending on her input X), from which Bob must output an answer in $o \in \mathcal{O}$ depending on his input Y and the message M. The maximum possible length (in bits) of M over all inputs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is the *communication cost* of the protocol $\mathcal{P}$. Communication protocols are allowed to be randomized, where each player has *private* access to an unlimited supply of random bits. The protocol $\mathcal{P}$ is said to *solve* the communication problem Q if Bob's output o belongs to Q_{xy} with failure probability at most $\delta < 1/2$. The *one-way communication complexity* of Q, denoted $R^{\rightarrow}_{\delta}(Q)$, is the minimum communication cost of a protocol which solves the protocol Q with failure probability δ.

Now a similar measure of complexity is the *distributional complexity* $D^{\rightarrow}_{\mu,\delta}(Q)$, where μ is a distribution over $\mathcal{X} \times \mathcal{Y}$, which denotes the minimum communication cost of the best deterministic protocol of Q with failure probability at most δ when the inputs $(x, y) \sim \mu$. By Yao's Lemma, we have that $R^{\rightarrow}_{\delta}(Q) = \max_{\mu} D^{\rightarrow}_{\mu,\delta}(Q)$. We first review some basic facts about entropy and mutual information (see Chapter 2 of [CT12] for proofs of these facts). Recall that for a

discrete random variable X supported on a finite domain Ω, the entropy $H(X)$ of X is given by $H(X) = -\sum_{a\in\Omega} \mathbf{Pr}[X = a] \log(\mathbf{Pr}[X = a])$.

Proposition 3.7.1.

1. *Entropy Span: if X takes on at most s values, then $0 \leq H(X) \leq \log s$*
2. *$I(X : Y) := H(X) - H(X|Y) \geq 0$, that is $H(X|Y) \leq H(X)$*
3. *Chain Rule: $I(X_1, X_2, \ldots, X_n : Y|Z) = \sum_{i=1}^{n} I(X_i : Y|X_1, \ldots, X_{i-1}, Z)$*
4. *Subadditivity: $H(X, Y|Z) \leq H(X|Z) + H(Y|Z)$ and equality holds if and only if X and Y are independent conditioned on Z*
5. *Fano's Inequality: Let M be a predictor of X. In other words, there exists a function g such that $\mathbf{Pr}[g(M) = X] > 1-\delta$ where $\delta < 1/2$. Let $\mathcal{U}$ denote the support of X, where $\mathcal{U} \geq 2$. Then $H(X|M) \leq \delta \log(|\mathcal{U}| - 1) + h_2(\delta)$, where $h_2(\delta) := \delta \log(\delta^{-1}) + (1 - \delta) \log(\frac{1}{1-\delta})$ is the binary entropy function.*

We now define the information cost of a protocol $\mathcal{P}$:

Definition 3.7.2. *Let μ be a distribution of the input domain $\mathcal{X} \times \mathcal{Y}$ to a communication problem Q. Suppose the inputs (X, Y) are chosen according to μ, and let M be Alice's message to Bob, interpreted as a random variable which is a function of X and Alice's private coins. Then the information cost of a protocol $\mathcal{P}$ for Q is defined as $I(X : M)$.*

The one-way information complexity of Q with respect to μ and δ, denoted by $IC^{\rightarrow}_{\mu,\delta}(Q)$, is the minimum information cost of a one-way protocol under μ that solves Q with failure probability at most δ.

Note that by Proposition 3.7.1, we have

$$I(X : M) = H(M) - H(M|X) \leq H(M) \leq |M|$$

where $|M|$ is the length of the message M in bits. This results in the following proposition.

Proposition 3.7.3. *For every probability distribution μ on inputs,*

$$R^{\rightarrow}_{\delta}(Q) \geq IC^{\rightarrow}_{\mu,\delta}(Q)$$

3.7.1 Lower Bound for Estimating Sampled Coordinate

We now introduce the following communication problem, known as Augmented Index on Large Domains. Our communication problem is derived from the communication problem (of the same name) introduced in [JW13], but we modify the guarantee of the output required so that constant probability of error is allowed. The problem is as follows.

Definition 3.7.4. *Let $\mathcal{U}$ be an alphabet with $|\mathcal{U}| = k \geq 2$. Alice is given a string $X = (X_1, \ldots, X_d) \in \mathcal{U}^d$, and Bob is given $i \in [d]$ along with the values $X_{i+1}, X_{i+2}, \ldots, X_d$. Alice must send a message M to Bob, and then Bob must output the value $X_i \in \mathcal{U}$ with probability $3/4$. We refer to this problem as the augmented-index problem on large domains, and denote it by $\text{IND}^d_{\mathcal{U}}$.*

Note that in [JW13], a correct protocol is only required to determine whether $X_i = a$ for some fixed input $a \in \mathcal{U}$ given only to Bob, however such a protocol must succeed with probability $1-\delta$. For the purposes of both problems, it is taken that $|\mathcal{U}| = \Theta(1/\delta)$. In this scenario, we note that the guarantee of our communication problem is strictly weaker, since if one had a protocol that determined whether $X_i = a$ for a given $a \in \mathcal{U}$ with probability $1 - \delta$, one could run it on all $a \in \mathcal{U}$ and union bound over all $|\mathcal{U}|$ trails, from which the exact value of x_i could be determined with probability $3/4$, thereby solving the form of the communication problem we have described. We show, nevertheless, that the same lower bound on the communication cost of our protocol holds as the lower bound in [JW13].

Let $\mathcal{X} = \mathcal{U}^d$ be the set of all possible inputs to Alice, let $\mathcal{Y} = [d]$, and define μ to be the uniform distribution over $\mathcal{X} \times \mathcal{Y}$.

Lemma 3.7.5. *Suppose $|\mathcal{U}| \geq c$ for some sufficiently large constant c. We have $IC^{\rightarrow}_{\mu,3/4}(\text{IND}^d_{\mathcal{U}}) \geq d\log(|\mathcal{U}|)/2$.*

Proof. Fix any protocol $\mathcal{P}$ for $\text{IND}^d_{\mathcal{U}}$ which fails with probability at most $1/4$. We now draw Alice's input $X = (X_1, X_2, \ldots, X_d) \sim \mathcal{X}$ via the uniform distribution μ, and let M be Alice's message to Bob given X. By Proposition 3.7.1

$$I(X : M) = \sum_{i=1}^{d} I(X_i : M | X_1, \ldots, X_{i-1})$$

$$= \sum_{i=1}^{d} \Big(H(X_i | X_1, \ldots, X_{i-1}) - H(X_i | M, X_1, \ldots, X_{i-1}) \Big)$$

First note that since X_i is independent of X_j for all $j \neq i$, we have $H(X_i|X_1, \ldots, X_{i-1}) = H(X_i) = \log(|\mathcal{U}|)$. Now since the protocol $\mathcal{P}$ is correct on $\text{IND}^d_{\mathcal{U}}$, then the variables $M, X_1, \ldots, X_{i-1}$ must be a predictor for X_i with failure probability $1/4$ (since Bob outputs X_i with probability $3/4$ given only $M, X_1, \ldots, X_{i-1}$ and his private, independent randomness). So by Fano's inequality (Proposition 3.7.1), we have

$$H(X_i|M, X_1, \ldots, X_{i-1}) \leq \frac{1}{4}\log(|\mathcal{U}| - 1) + h_2(\frac{1}{4})$$

$$\leq \frac{1}{2}\log(|\mathcal{U}|)$$

which holds when $|\mathcal{U}|$ is sufficiently large. Putting this together, we obtain

$$I(X : M) \geq \frac{d\log(|\mathcal{U}|)}{2}$$

□

Corollary 3.7.6. *We have* $R^{\rightarrow}_{3/4}(\text{IND}^d_{\mathcal{U}}) = \Omega(d\log(|\mathcal{U}|))$.

We now use this lower bound on $\text{IND}^d_{\mathcal{U}}$ to show that, even when the index output is from a distribution with constant *additive* error from the true L_p distribution, returning an estimate with probability $1 - \delta_2$ still requires $\Omega(\nu^{-p}\log(n)\log(1/\delta_2))$ bits of space.

Theorem 27. *Fix any $p > 0$ constant bounded away from 0, and let $\nu < 1/3$ with $\nu^{-p} = o(n)$. Then any L_p sampling algorithm that, on stream input $x \in \mathbb{R}^n$, outputs $\perp$ with probability at most $1/100$, and otherwise returns an item $\ell \in [n]$ such that $\mathbf{Pr}[\ell = l] = |x_l|^p/\|x\|_p^p \pm 1/50$ for all $l \in [n]$, along with an estimate $\tilde{x}_\ell$ such that $\tilde{x}_\ell = (1 \pm \nu)f_\ell$ with probability $1 - \delta_2$, requires $\Omega(\nu^{-p}\log(n)\log(1/\delta_2))$ bits of space.*

Proof. We reduce via $\text{IND}^d_{\mathcal{U}}$. Suppose we have a streaming algorithm $\mathcal{A}$ which satisfies all the properties stated in the theorem. Set $|\mathcal{U}| = 1/(10\delta_2)$, and let $X \in \mathcal{U}^d$ be Alice's input, where $d = rs$ where $r = \frac{1}{10^{p+1}\nu^p}$ and $s = \log(n)$. Alice conceptually divides X into s blocks $X^1, \ldots, X^s$, each containing r items $X^i = X^i_1, X^i_2, \ldots, X^i_r \in \mathcal{U}$. Fix some labeling $\mathcal{U} = \{\sigma_1, \ldots, \sigma_k\}$, and let $\pi(X^i_j) \in [k]$ be such that $X^i_j = \sigma_{\pi(X^i_j)}$. Then each X^i_j can be thought of naturally as a binary vector in $\mathbb{R}^{rsk}$ with support 1, where $(X^i_j)_t = 1$ when $t = (i-1)r + (j-1)k + \pi(X^i_j)$, and $(X^i_j)_t = 0$ otherwise. Set $n' = rsk < n$ for $\nu^{-p} = o(n)$. Using this interpretation of $X^i_j \in \mathbb{R}^{rsk}$,

we define the vector $x \in \mathbb{R}^{rsk}$ by

$$x = \sum_{i=1}^{s} \sum_{j=1}^{r} B^i X_j^i$$

Where $B = 10^{1/p}$. Alice can construct a stream with the frequency vector x by making the necessary insertions, and then send the state of the streaming algorithm $\mathcal{A}$ to Bob. Now Bob has some index $i^* \in [d] = [rs]$, and his goal is to output the value of $X_{j'}^{i'} = X_{i^*}$ such that $i^* = (i'-1)r + j'$. Since Bob knows X_i^j for all (i,j) with $i > i'$, he can delete off the corresponding values of $B^i X_j^i$ from the stream, leaving the vector x with the value

$$x = \sum_{i=1}^{i'} \sum_{j=1}^{r} B^i X_j^i$$

For $j \in [k]$, let $\gamma^j \in \mathbb{R}^{rsk}$ be the binary vector with $\gamma^j_{(i'-1)r+(j'-1)k+j} = B^{i'}/(10\nu)$ and $\gamma_t^j = 0$ at all other coordinates $t \neq (i'-1)r + (j'-1)k + j$. Bob then constructs the streams $x^j = x + \gamma^j$ for $j = 1, \ldots, k$ sequentially. After he constructs x^j, he runs $\mathcal{A}$ on x^j to obtain an output $(\ell_j, \tilde{x}^j_{\ell_j}) \in ([n'] \times \mathbb{R}) \cup (\{\perp\} \times \{\perp\})$ from the streaming algorithm, where if the algorithm did not fail we have that $\ell_j \in [n']$ is the index output and $\tilde{x}^j_{\ell_j}$ is the estimate of $x^j_{\ell_j}$. By union bounding over the guarantee of $\mathcal{A}$ we have that if $\ell_j \neq \perp$ then $\tilde{x}_{\ell_j} = (1 \pm \nu) x^j_{\ell_j}$ for all $j = 1, 2, \ldots, k$ with probability $1 - k\delta_2 > 9/10$. Call this event $\mathcal{E}_1$. Conditioned on $\mathcal{E}_1$, it follows that if for each ℓ_j with $\ell_j = (i'-1)r + (j'-1)k + j$, if $X_{j'}^{i'} = \sigma_j$ then

$$\tilde{x}^j_{\ell_j} > B^{i'}(1 + \frac{1}{10\nu})(1-\nu) > \frac{B^{i'}}{10\nu} + \frac{9}{10} B^{i'} - \nu B^{i'}$$

On the other hand, if $X_{j'}^{i'} \neq \sigma_j$, then we will have

$$\tilde{x}^j_{\ell_j} < (B^{i'}/(10\nu))(1+\nu) = \frac{B^{i'}}{10\nu} + \frac{B^{i'}}{10}$$

$$< \frac{B^{i'}}{10\nu} + \frac{9}{10} B^{i'} - \nu B^{i'}$$

using that $\nu < 1/3$. Thus if $\ell_j = (i'-1)r + (j'-1)k + j$, Bob can correctly determine whether or not $X_{j'}^{i'} = \sigma_j$. Now suppose that, in actuality, Alice's item was $X_{j'}^{i'} = \sigma_\tau \in \mathcal{U}$ for some $\tau \in [k]$. Set $\lambda = (i'-1)r + (j'-1)k + \tau$. To complete the proof, it suffices to lower bound the probability that $\ell_\tau \neq \lambda$.

Thus we consider only the event of running $\mathcal{A}$ on x^τ. We know that with probability $99/100$, $\ell_\tau \neq \perp$. We write $\mathcal{E}_2$ to denote the event that $\ell_\tau \neq \perp$. Let $x_{-\lambda}$ be equal to f everywhere except

with the coordinate λ set equal to 0. Then

$$\|x^{\tau}_{-\lambda}\|_p^p < \sum_{i=1}^{i'}\sum_{j=1}^{r}(B^p)^i$$

$$\leq r\sum_{i=1}^{i'} 10^i \leq (\frac{1}{10^{p+1}\nu^p})\frac{10^{i'+1}}{9}$$

So

$$\frac{|x^{\tau}_{\lambda}|^p}{\|x^{\tau}_{-\lambda}\|_p^p} \geq \frac{10^{i'}(\frac{1}{10\nu})^p}{(\frac{1}{10^{p+1}\nu^p})\frac{10^{i'+1}}{9}} \geq \frac{9(\frac{1}{10\nu})^p}{(\frac{1}{10\nu})^p} \geq 9$$

Since $\mathcal{A}$ has $1/50$-additive error, we conclude $\mathbf{Pr}\left[\ell_\tau = \lambda\right] > 9/10 - 1/50 = 22/25$, and call the event that this occurs $\mathcal{E}_3$. Then conditioned on $\mathcal{E} = \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3$ Bob sucsessfully recovers the value of $X^{i'}_{j'} = X_{i^*}$, and thus solves the communication problem. Note that the probability of success is $\mathbf{Pr}\left[\mathcal{E}\right] > 1 - (1/10 + 1/100 + 3/25) > 3/4$, and thus this protocol solves $\text{IND}^d_{\mathcal{U}}$. So by Corollary 3.7.6, it follows that any such streaming algorithm $\mathcal{A}$ requires $\Omega(rs\log(|\mathcal{U}|)) = \Omega(\nu^{-p}\log(n)\log(1/\delta_2))$ bits of space. Note that the stream x in question had length $n' < n$ for p constant bounded from 0, and no coordinate in the stream ever had a value greater than $\text{poly}(n)$, and thus the stream in question is valid in the given streaming model. □

3.7.2 Lower Bound for Truly Perfect Sampling

In this section, we demonstrate that the truly perfect G-samplers cannot exist in sublinear space in the turnstile model, for nearly any measure $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$. Specifically, we show that any perfect sampler with additive error $\gamma = n^{-c}$ requires space at least $\Omega(c\log n)$. This demonstrates that no sublinear space truly perfect sampler can exist in the turnstile model, and demonstrates the tightness (up to $\log n$ factors), of the previously known approximate L_p samplers [MW10, AKO11, JST11] as well as the perfect L_p described earlier in this chapter.

Our lower bound is based on the fine-grained hardness of the EQUALITY problem from two-party communication complexity [BCK+14]. Specifically, consider the boolean function $\text{EQ}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ given by $\text{EQ}_n(X,Y) = 1 \iff X = Y$. Recall in the two party, one way communication model, there are two parties: Alice and Bob. Alice is given a input string $X \in \{0,1\}^n$ and Bob is given $Y \in \{0,1\}^n$. Then Alice must send a single message M to Bob, who must then output whether $\text{EQ}_n(X,Y)$ correctly with some probability. A communication protocol $\mathcal{P}$ is a randomized two-party algorithm which takes the input (X,Y) and decides on a message M and output procedure for Bob given (M,Y). The communication cost of $\mathcal{P}$ is

denoted $\mathrm{cost}(\mathcal{P}, X, Y)$, and defined as the maximum number of bits sent in the message M over all coin flips of the algorithm, on inputs (X, Y).

We now define the randomized *refutation complexity* of a communication protocol for computing a Boolean function f. We define the *refutation cost*, *refutation error*, and *verification error* as:

$$\begin{aligned}
\mathrm{rcost}(\mathcal{P}) &= \max_{(X,Y)\in f^{-1}(0)} \mathrm{cost}(\mathcal{P}, X, Y) \\
\mathrm{rerr}(\mathcal{P}) &= \max_{(X,Y)\in f^{-1}(0)} \Pr \mathrm{out}(\mathcal{P}(X, Y)) = 1 \\
\mathrm{verr}(\mathcal{P}) &= \max_{(X,Y)\in f^{-1}(1)} \Pr \mathrm{out}(\mathcal{P}(X, Y)) = 0
\end{aligned}$$

We define the randomized refutation complexity of a function f for an integer $r \geq 1$ as

$$R^{(r),\mathrm{ref}}_{\epsilon,\delta}(f) = \min_{\mathcal{P}} \{\mathrm{rcost}(\mathcal{P}) : \mathrm{rerr}(\mathcal{P}) \leq \epsilon, \mathrm{verr}(\mathcal{P}) \leq \delta\}$$

where the minimum is restricted to r-round communication protocols $\mathcal{P}$. Finally, we define the effective instance size as

$$\hat{n} = \min \left\{ n + \log(1-\delta),\ \log\left(\frac{(1-\delta)^2}{\epsilon}\right) \right\}.$$

Theorem 28 (Theorem 44 [BCK+14])**.** *We have* $R^{(r),\mathrm{ref}}_{\epsilon,\delta}(\mathrm{EQ}_n) \geq \frac{1}{8}(1-\delta)^2(\hat{n} + \log(1-\delta) - 5)$.

Theorem 29. *Fix constant* $\epsilon_0 < 1$*, integer* $r \geq 1$*, and let* $2^{-n/2} \leq \gamma < \frac{1}{2}$*. Let* $G : \mathbb{R} \to \mathbb{R}_{\geq 0}$ *be any function satisfying* $G(x) > 0$ *for* $x \neq 0$*, and* $G(0) = 0$*. Then any* $(\epsilon_0, \frac{1}{2}, \gamma)$*-approximate* G*-sampler* $\mathcal{A}$ *in the* r*-pass turnstile streaming model must use* $\Omega\left(\min\left\{n, \log\frac{1}{\gamma}\right\}\right)$ *bits of space.*

Proof. Given such a sampler $\mathcal{A}$, we give an algorithm for the two-party, r-round equality problem as follows. Alice is given $X \in \{0,1\}^n$, and Bob is given $Y \in \{0,1\}^n$. Alice first creates a stream with frequency vector given by $x = X$, and sends the state of the algorithm to Bob. Bob then adds the vector $-Y$ into the stream so that the final state of the frequency vector induced by the stream is $x = X - Y$. Alice and Bob each run $\mathcal{A}$ on their stream and repeatedly pass the state of the algorithm between each other over the course of r rounds. Bob then finally obtains the output of the streaming algorithm $\mathcal{A}(x)$ after r rounds. If the output is $\perp$ or any coordinate $i \in [n]$, then Bob declares $\mathrm{EQ}_n(X, Y) = 0$. If the output is **ZERO**, Bob declares $\mathrm{EQ}_n(X, Y) = 1$. Notice by definition of a $(\epsilon_0, \frac{1}{2}, \gamma)$–$G$ sampler (Definition 3.1.3), if we actually had $X = Y$, then $x = \vec{0}$, so $\mathcal{A}$ must output **ZERO** with probability at least $1 - \gamma$. Moreover, if $X \neq Y$, then a correct

sampler can output $\perp$ with probability at most γ.

The above protocol therefore satisfies that if $\text{EQ}_n(X, Y) = 0$, Bob outputs 1 with probability at most γ, thus the refutation error is at most $\epsilon < \gamma$. Moreover, if $\text{EQ}_n(X, Y) = 1$, then $\mathcal{A}$ outputs 0 with probability at most γ. Thus, the verification error is at most $\gamma < 1/2$. Then we have $n - \log(1 - \delta) > n/2$, and $\log(\frac{(1-\delta)^2}{\epsilon}) > \log(\frac{1}{16\gamma})$. So the effective input size is given by:

$$\hat{n} > \min\left\{\frac{n}{2}, \log\frac{1}{4\gamma}\right\}$$

By Theorem 28:

$$\begin{aligned} R^{(r),\text{ref}}_{\epsilon,\delta}(\text{EQ}_n) &\geq \frac{1}{8}(1-\delta)^2(\hat{n} + \log(1-\delta) - 5) \\ &\geq \frac{1}{8 \cdot 4}(\hat{n} - 7) \\ &= \Omega(\hat{n}) \end{aligned} \tag{3.15}$$

which completes the proof of the lower bound.

□

Chapter 4

Moment Estimation in Streaming and Distributed Models

The study of streaming algorithms has often been closely linked to the field of distributed algorithms. Dual to the latter is the field of *communication complexity*, which studies of the amount of communication required to solve a distributed task. One significant component of the relationship between streaming and distributed algorithms is the fact that nearly all known lower bounds for the space complexity of streaming algorithms arise through the field of *communication complexity* [Woo04, WZ18, KNW10a, JST11, KNP+17, BCWY16, CKS03, WW15, LW13, MPTW16, JW09]. In this Chapter, we investigate the fundamental problem of F_p moment estimation in both streaming and distributed models.

The study of frequency moments in the streaming model was initiated by the seminal 1996 paper of Alon, Matias, and Szegedy [AMS96]. Since then, nearly two decades of research have been devoted to understanding the space and time complexity of this problem. An incomplete list of works which study frequency moments in data streams includes [CKS03, IW05, BYJKS04, Woo04, Ind06, KNW10a, BO10, KNPW11a, BKSV14, CCM16, BKSV14, BDN17, BVWY18]. For $p > 2$, it is known that polynomial in n (rather than logarithmic) space is required for F_p estimation [CKS03, IW05]. In the regime of $p \in (0, 2]$, the space complexity of F_p estimation in the turnstile model is now understood, with matching upper and lower bounds of $\Theta(\epsilon^{-2} \log n)$ bits to obtain a $(1 \pm \epsilon)$ approximation of F_p. For insertion only streams, however, the best known lower bound is $\Omega(\epsilon^{-2} + \log n)$ [Woo04]. Moreover, in the random oracle model, then the lower bound is only $\Omega(\epsilon^{-2})$. On the other hand, the best upper bound is to just run the turnstile $O(\epsilon^{-2} \log n)$-space algorithm.

This Chapter, based on our paper [JW19], makes progress towards resolving this fundamental problem. For $p < 1$, we resolve the space complexity by giving an $\tilde{O}(\epsilon^{-2} + \log n)$-bits of space upper bound. In the random oracle model, our upper bound is $\tilde{O}(\epsilon^{-2})$[1], which also matches the lower bound in this setting. Previously, an $\tilde{O}(\epsilon^{-2} + \log(n))$ upper bound for F_p estimation was only known in the restricted *random-order* model, where it is assumed that the stream updates are in a uniformly random ordering [BVWY18]. Additionally, we apply our techniques to obtain a space optimal algorithm for insertion-only entropy estimation.

For the range of $p \in (1, 2]$, we prove an $\tilde{O}(\epsilon^{-2})$-bits of max-communication upper bound in the distributed models most frequently used to prove *lower bounds* for streaming. This result rules out a large and very commonly used class of approaches for proving lower bounds against the space complexity of streaming algorithms for F_p estimation. Our approach is based on a new randomized rounding scheme for p-stable sketches. Along the way, we will prove some useful inequalities to bound the heavy-tailed error incurred from rounding non-i.i.d. p-stable variables for $p < 2$ (Lemmas 4.3.3 and 4.4.7). We show that our rounding scheme can be additionally applied to design improved protocols for the distributed heavy hitters and approximate matrix product problems.

Highlighted Contributions

The materials from this chapter are drawn from our paper [JW19]. The main contributions therein are as follows:

- We design improved F_p-moment estimation algorithm for $p \in (1, 2]$, using $\tilde{O}(\epsilon^{-2})$-bits of max-communication, in the distributed models most frequently used to prove *lower bounds* for streaming, ruling out a wide class of approaches for improving lower bounds for this range of p (Section 4.3).

- We resolve the space complexity of F_p-estimation in insertion-only streams for $p < 1$. Specifically, we give an algorithm that uses $\tilde{O}(\epsilon^{-2} + \log n)$ bits of space in general, and $\tilde{O}(\epsilon^{-2})$ bits in the random oracle model, both of which match their respective lower bounds up to $\log \log n, \log \epsilon^{-1}$ factors (Section 4.4).

[1]This space complexity is measured *between updates*. To read and process the $\Theta(\log(n))$-bit identity of an update, the algorithm will use an additional $O(\log(n))$-bit working memory tape during an update. Note that all lower bounds only apply to the space complexity between updates, and allow arbitrary space to process updates.

- We resolve the space complexity of entropy estimation in the random oracle insertion only model, by giving a $\tilde{O}(\frac{1}{\epsilon^2})$ upper bound (Section 4.5).

4.1 Background

In this Chapter, we study a more general model than streaming, known as the **message passing multi-party communication model**, which we now introduce. All of our upper bounds apply to this model, and our streaming algorithms are just the result of special cases of our communication protocols. In the message passing model, there are m players, each positioned at a unique vertex in a graph $G = (V, E)$. The i-th player is given as input an integer vector $X_i \in \mathbb{Z}^n$. The goal of the players is to work together to jointly approximate some function $f : \mathbb{R}^n \to \mathbb{R}$ of the aggregate vector $\mathcal{X} = \sum_{i=1}^n X_i$, such as the p-th moment $f(\mathcal{X}) = F_p = \|\mathcal{X}\|_p^p = \sum_{i=1}^n |\mathcal{X}_i|^p$. We use the notation $\mathcal{X}, X$ instead of x to distinguish the distributed from streaming models; in particular, in this Chapter the symbol x is not reserved for the frequency vector of a data stream, as the results primarily concern the more general message passing model.

In the message passing model, as opposed to the *broadcast* model of communication, the players are only allowed to communicate with each other over the edges of G. Thus player i can send a message to player j only if $(i, j) \in E$, and this message will only be received by player j (and no other). At the end of the protocol, it is assumed that at least one player holds the approximation to $f(\mathcal{X})$. The goal of multi-party communication is to solve the approximation problem using small total communication between all the players over the course of the execution. More specifically, the goal is to design protocols that use small *max-communication*, which is the total number of bits sent over any edge of G. The protocols described in this Chapter hold in an even more restricted setting, known as the *one-shot* setting, where each player is allowed to communicate exactly once over the course of the entire protocol.

The message passing setting is often used to model distributed computation, where there are m distributed machines, arranged in some network topology G, and their goal is to jointly compute some function of the aggregate data $\mathcal{X}$. Oftentimes, the predominant bottleneck in distributed data processing is the network bandwidth and the energy cost of communication [JOW+02, MFHH05], and so it is desirable to have algorithms that use as little communication over any edge as possible.

We now observe that data streams can be modeled as a special case of one-shot multi-party communication. Here, the graph G in question is the line graph on m vertices. If the updates

to the data stream vector are $(i_1, \Delta_1), \ldots, (i_m, \Delta_m)$, then the t-th player has input $X_t \in \mathbb{Z}^n$, where $(X_t)_{i_t} = \Delta_t$ and $(X_t)_j = 0$ for $j \neq i_t$. The aggregate vector $\mathcal{X} = \sum_{i=1}^m X_i$ is just the frequency vector x at the end of the stream, and the space complexity of any algorithm is just the max-communication used over any edge of the corresponding communication protocol. Since we are primarily interested in insertion only streams, in this Chapter we will consider the *non-negative data* model, where $X_i \in \{0, 1, \ldots, M\}^n$ for all input vectors X_i, for some $M > 0$ (as in streaming, we assume $M = \text{poly}(n, m)$ for simplicity). Note that an equivalent condition is that each $X_i \in \mathbb{R}^n_{\geq 0}$ such that the entries of X_i can be stored in $O(\log M)$-bits.

We are now ready to introduce our results for moment estimation in the message passing model. Let d be the *diameter* of the communication graph G. Our first result is a protocol for F_p estimation when $p \in (1, 2]$ which uses a max communication of $\tilde{O}(\epsilon^{-2} \log d)$ bits. Using similar techniques, we also obtain a (optimal for $d = \Theta(1)$) bound of $\tilde{O}(\epsilon^{-2} \log n \log d)$ for the heavy hitters problem, which is to find the coordinates of $\mathcal{X}$ which contribute at least an ϵ fraction of the total $\sqrt{F_2} = \|\mathcal{X}\|_2$ of $\mathcal{X}$. For $p \in (0, 1)$, we give an $\tilde{O}(\epsilon^{-2})$ upper bound for F_p estimation. Notice that this is independent of the graph topology, and thus holds for the line graph, where we derive our $\tilde{O}(\epsilon^{-2})$ upper bound for F_p estimation in the random oracle streaming model. We then show how the streaming algorithm can be derandomized to not require a random oracle, now using an optimal $\tilde{O}(\epsilon^{-2} + \log(n))$-bits of space. Our techniques also result in an $\tilde{O}(\epsilon^{-2})$ upper bound for additively approximating the empirical entropy of the vector $\mathcal{X}$.

Our results for $p \in (1, 2]$ have interesting implications for any attempts to prove *lower-bounds* for streaming algorithms that estimate F_p, which we now describe. The link between streaming and communication complexity is perhaps one of the most fruitful sources of space lower bounds for algorithms in computer science. Namely, nearly all lower bounds for the space complexity of randomized streaming algorithms are derived via reductions from communication problems. For an incomplete list of such reductions, see [Woo04, WZ18, KNW10a, JST11, KNP+17, BCWY16, CKS03, WW15, LW13, MPTW16, JW09] and the references therein. Now nearly all such lower bounds (and all of the ones that were just cited) hold in either the 2-party setting (G has 2 vertices), the coordinator model, or the black-board model. In the coordinator model there are m players, each with a single edge to a central coordinator (i.e., G is a star graph on $m + 1$ vertices). Note that the diameter d of the coordinator graph is 2. In the multi-player black-board model, every message that is sent is written to a shared blackboard that can be read by all players. Observe that any one-way protocol for the coordinator model immediately results in a protocol with the same communication for the blackboard model. Namely, each player simply writes what it would have sent to the coordinator on the blackboard, and at the end of

the protocol the blackboard contains all the information that the coordinator would have had. For these three settings, our protocol gives an $\tilde{O}(\epsilon^{-2})$ max-communication upper bound for F_p estimation, $p \in (1, 2]$. This completely rules out the approach for proving lower bounds against F_p estimation in a stream via any of these three techniques. In particular, it appears that any lower bound for F_p estimation via communication complexity in this regime of p will need to use a graph with $\Omega(n)$ diameter, such as the line graph, without a black-board.

The coordinator and black-board models have also been studied in many other settings than for proving lower bounds against streaming. For instance, in the *Distributed Functional Monitoring* literature [CMY11, YZ13, WZ12, HYZ12, TW11, JSTW19], each player is receiving a continuous stream of updates to their inputs X_i, and the coordinator must continuously update its approximation to $f(\mathcal{X})$. The black-board model is also considered frequently for designing communication upper bounds, such as those for set disjointness [BYJKS04, CKS03, Gro09]. Finally, there is substantial literature which considers numerical linear algebra and clustering problems in the coordinator model [WZ18, CSWZ16, BLS+16, WZ16]. Thus, our upper bounds can be seen as a new and useful contribution to these bodies of literature as well.

Numerical Linear Algebra The study of numerical linear algebra in the distributed model has become increasingly important [FSS13, LBKW14, Lib13, GP14, KVW14, BJS15, WZ18, BWZ16], especially as frameworks like Hadoop [had] and Spark [spa] become critical for analyzing massive datasets. These works prioritize designing protocols to compute linear algebraic primitives of matrices distributed over multiple servers, using as little communication as possible. Motivated by this, we apply the techniques developed for our F_p estimation problems to the linear algebraic primitive of approximate matrix product.

The setting is as follows. Instead of vector-valued inputs $X_i \in \mathbb{Z}^n_{\geq 0}$, the players are given $\mathbf{X}_i \in \mathbb{Z}^{n \times t_1}_{\geq 0}$, and they would like to compute statistics of the aggregate *data matrix* $\boldsymbol{\mathcal{X}} = \sum_{i=1}^m \mathbf{X}_i$. We remark that in the domain of application, we generally assume $n >> t_1, t_2$ (however this is not, strictly speaking, a requirement). So, for instance, they may want to estimate the $t_1 \times t_2$ dimensional product of $\boldsymbol{\mathcal{X}}^T$ with another matrix $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n \times t_2}$. In this case, each player also gets as input a $\mathbf{Y}_i \in \mathbb{Z}^{n \times t_2}$, and we set $\boldsymbol{\mathcal{Y}} = \sum_{i=1}^m \mathbf{Y}_i$. The goal of the approximate matrix product problem is for the players to jointly compute an approximation to $\boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{Y}}$. Specifically, they would like to obtain a $\mathbf{R} \in \mathbb{R}^{t_1 \times t_2}$ such that $\|\mathbf{R} - \boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{Y}}\|_F \leq \epsilon \|\boldsymbol{\mathcal{X}}\|_F \|\boldsymbol{\mathcal{Y}}\|_F$, where recall that the Frobenius norm $\|\mathcal{X}\|_F$ of a matrix $\mathcal{X}$ is just the square root of the sum of squares of entries in $\mathcal{X}$. We note that a special and important case of interest is covariance estimation, where $\mathbf{X}_i = \mathbf{Y}_i$. Here, each player i has some subset of a positive data matrix $\mathbf{X}$, and the goal is to compute the empirical covariance $\boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{X}}$ of the data.

Using similar techniques as our F_p estimation algorithm for $p > 1$, we design an $\tilde{O}((t_1 + t_2)\epsilon^{-2} \log d)$-bit max communication protocol for the problem. We note here that a $O((t_1 + t_2)\epsilon^{-2} \log n)$ upper bound is a standard and well-known result from sketching, and our main contribution here is to reduce the $\log n$ to a $\log d$. Since the majority of works in the distributed linear algebra literature consider the coordinator model, our results demonstrate an improvement of a $\log(n)$ factor in the communication for approximate matrix product in this setting.

Linear Sketches Our results are closely related to the theory of linear sketching, which is the standard approach used to solve many problems which involve approximating functions of extremely high dimensional data $\mathcal{X}$. Formally, a linear sketch is a randomized matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$, where $k << n$, such that given $\mathbf{S}\mathcal{X} \in \mathbb{R}^k$, one can approximate the desired function of $\mathcal{X}$. To estimate F_p, $p \in (0, 2]$, such a sketch $\mathbf{S}$ exists with $k = \Theta(\epsilon^{-2})$ rows ([Ind06, KNW10a]). If $p = 2$, then $\mathbf{S}$ is a matrix of i.i.d. Gaussian random variables, and for general $p \in (0, 2]$, so called p-stable distributions, $\mathcal{D}_p$, are used (Definition 2.2.6). These distributions have the property that if $Z_1, \ldots, Z_m \sim \mathcal{D}_p$, then $\sum_{i=1}^m Z_i \mathcal{X}_i \sim Z \|\mathcal{X}\|_p$, where $Z \sim \mathcal{D}_p$ again.

To solve the multi-party F_p estimation problem using sketches, each player can locally compute $\mathbf{S}X_i$ and pass it along to some centralized vertex, who can then compute $\mathbf{S}\mathcal{X} = \sum_i \mathbf{S}X_i$. Given $\mathbf{S}\mathcal{X}$, the central vertex can then compute the desired approximation. If the players sum up their sketches along the way, as they travel up to the central vertex, the per-player communication of this protocol is $O(\epsilon^{-2} \log n)$, where the $\log n$ factor is required to store each coordinate of $\mathbf{S}\mathcal{X}$. As noted, however, the extra $\log n$ factor does not match the $\Omega(\epsilon^{-2})$ lower bound. Thus, the main challenge for each of the protocols in this paper will be to avoid sending the entire $O(\log n)$-bit counter needed to store a coordinate of the sketch $\mathbf{S}X_i$.

4.1.1 Contributions

As noted, the upper bounds in this Chapter all hold in the general non-negative data multi-party message passing model, over an arbitrary topology G. The algorithms also have the additional property that they are *one-shot*, meaning that each player is allowed to communicate exactly once. Our protocols pre-specify a central vertex $\mathcal{C} \in V$ of G. Specifically, $\mathcal{C}$ will be a *center* of G, which is a vertex with minimal max-distance to any other vertex. Our protocols then proceed in d rounds, where d is the diameter of G. Upon termination of the protocols, the central vertex $\mathcal{C}$ will hold the estimate of the protocol. We note that $\mathcal{C}$ can be replaced by any other vertex v, and d will then be replaced by the max distance of any other vertex to v. A summary of the result in this Chapter is given in Figure 4.1.

Problem	Prior best upper bound	Upper bound (this work)	Notes
F_p, $1 < p \leq 2$	$O(\epsilon^{-2}\log(n))$ [KNW10a]	$\tilde{O}(\epsilon^{-2}\log(d))$	
F_p, $p < 1$	$O(\epsilon^{-2}\log(n))$[KNW10a]	$\tilde{O}(\epsilon^{-2})$	
F_p Streaming, $p < 1$	$O(\epsilon^{-2}\log(n))$[KNW10a]	$\tilde{O}(\epsilon^{-2})$	
Entropy	–	$\tilde{O}(\epsilon^{-2})$	
Entropy Streaming	$O(\epsilon^{-2}\log^2(n))$ [CC13]	$\tilde{O}(\epsilon^{-2})$	random oracle
Point Estimation	$O(\epsilon^{-2}\log^2(n))$ [CCFC02b]	$\tilde{O}(\epsilon^{-2}\log(d)\log(n))$	
Approx Matrix Prod.	–	$\tilde{O}(1)$	per coordinate of sketch

Figure 4.1: For the communication problems above, the bounds are for the max-communication (in bits) across any edge. For the streaming problems, the bounds are for the space requirements of the algorithm. Here, d is the diameter of the communication network G. For all problems except point estimation, there is a matching $\Omega(\epsilon^{-2})$ lower bound. The problem of point estimation itself has a matching $\Omega(\epsilon^{-2}\log n)$ lower bound for graphs with constant d.

We first formally state our general result for F_p estimation, $1 < p \leq 2$. Note that, while we state all our results for constant probability of success, by repeating $\log(1/\delta)$ times and taking the median of the estimates, this is boosted to $1-\delta$ in the standard way.

Theorem 30 *For $p \in (1,2]$, there is a protocol for $(1 \pm \epsilon)$ approximating F_p which succeeds with probability $3/4$ in the message passing model. The protocol uses a max communication of $O(\frac{1}{\epsilon^2}(\log\log n + \log d + \log 1/\epsilon))$ bits, where d is the diameter of G.*

For graphs with constant diameter, such as the coordinator model, our max communication bound of $\tilde{O}(\epsilon^{-2})$ matches the $\Omega(\epsilon^{-2})$ lower bound [Woo04, CR12], which follows from a 2-player reduction from the Gap-Hamming Distance problem. For $p = 2$, our *total communication* in the coordinator model matches the $\Omega(m^{p-1}/\epsilon^2)$ total communication lower bound (up to $\log\log(n)$ and $\log(1/\epsilon)$ terms) for non-one shot protocols [WZ12]. For one shot protocols, we remark that there is an $\Omega(m/\epsilon^2)$ total communication lower bound for any $p \in (0,2] \setminus \{1\}$ (see Section 4.7.1). As discussed previously, our result also has strong implications for streaming algorithms, demonstrating that no $\Omega(\epsilon^{-2}\log n)$ lower bound for F_p estimation, $p \in (1,2]$, can be derived via the common settings of 2-party, coordinator, or blackboard communication complexity.

Our main technique used to obtain Theorem 30 is a new randomized rounding scheme for p-stable sketches. Suppose we are in the coordinator model, $Z = (Z_1, \ldots, Z_n)$ are i.i.d. p-stable, and the players want to jointly compute $\langle Z, \mathcal{X}\rangle = \sum_{i=1}^m \langle Z, X_i\rangle$. Then, roughly speaking, they

could each round their values $\langle Z, X_i \rangle$ to $(1 \pm \gamma)$ relative error, using $O(\log \log n + \log 1/\gamma)$ bits of communication, and send it to the coordinator. The error for each player would be at most $\gamma |\langle Z, X_i \rangle|$, for a total error of $\gamma \sum_{i=1}^m |\langle Z, X_i \rangle|$. For $p > 1$, however, the final counter value $\langle Z, \mathcal{X} \rangle$ will be much smaller than $\gamma \sum_{i=1}^m |\langle Z, X_i \rangle|$ – in fact it will by polynomial smaller. To see this, note that $\langle Z, \mathcal{X} \rangle \sim z \|\mathcal{X}\|_p$, where z is p-stable, thus $\langle Z, \mathcal{X} \rangle = O(\|\mathcal{X}\|_p)$ with good probability. On the other hand, each $|\langle Z, X_i \rangle|$ will be $\Omega(\|X_i\|_p)$ with good probability, and so $\sum_{i=1}^m |\langle Z, X_i \rangle|$ can be as large as $\Omega(m^{1-1/p} \|\mathcal{X}\|_p)$ (and in fact larger due to the heavy tails), so the error is too great.

Our solution is to randomly round the values $\langle Z, X_i \rangle$ to relative error $(1 \pm \gamma)$ so that the error is zero mean. The total error can then be bounded via the variance. This is proportional to $\sum_{i=1}^m |\langle Z, X_i \rangle|^2$, which behaves like $\sum_{i=1}^m z_i^2 \|X_i\|_p^2$, where the z_i's are non-i.i.d. p-stable. Note that if each z_i^2 was constant, by the positivity of the vectors X_i, we have $\sum_{i=1}^m z_i^2 \|X_i\|_p^2 = O(\|\mathcal{X}\|_p^2)$ for any $1 \leq p \leq 2$. However, since p-stables have heavy tails, many of the z_i^2's will be super-constant, and in fact many will be polynomially sized in m. By a careful analysis of the tail behavior of this sum, we are able to still bound the variance by (roughly) $\gamma \|\mathcal{X}\|_p^2$, where γ is the precision to which the randomized rounding is carried out, which will give us our protocol for the coordinator model. Our general protocol is the result of iterative applying this rounding and merging scheme until all sketches reach a central vertex. By a judicious analysis of how the variance propagates, we can demonstrate that γ need only be $O(1/d)$, where d is the diameter of G, to sufficiently bound the overall error of the protocol.

We then show that this randomized rounding protocol can be applied to give improved communication upper bounds for the *point-estimation* problem. Here, the goal is to output a vector $\tilde{X} \in \mathbb{R}^n$ that approximates $\mathcal{X}$ well coordinate-wise. The result is formally given below in Theorem 31.

Theorem 31 *Consider a message passing topology $G = (V, E)$ with diameter d, where the i-th player is given as input $X^i \in \mathbb{Z}_{\geq 0}^n$ and $\mathcal{X} = \sum_{i=1}^m X^i$. Then there is a communication protocol which outputs an estimate $\tilde{\mathcal{X}} \in \mathbb{R}^n$ of $\mathcal{X}$ such that*

$$\|\tilde{\mathcal{X}} - \mathcal{X}\|_\infty \leq \epsilon \|\mathcal{X}_{tail(\epsilon^{-2})}\|_2$$

with probability $1 - 1/n^c$ for any constant $c \geq 1$. Here $\mathcal{X}_{tail(\epsilon^{-2})}$ is $\mathcal{X}$ with the ϵ^{-2} largest (in absolute value) coordinates set equal to 0. The protocol uses a max communication of $O(\frac{1}{\epsilon^2} \log(n)(\log \log n + \log d + \log 1/\epsilon))$.

For graphs with small diameter, our protocols demonstrate an improvement over the previously best known sketching algorithms, which use space $O(\epsilon^{-2}\log^2(n))$ to solve the point estimation problem [CCFC02b]. Note that there is an $\Omega(\epsilon^{-2}\log n)$-max communication lower bound for the problem. This follows from the fact that point-estimation also solves the L_2 *heavy-hitters* problem. Here the goal is to output a set $S \subset [n]$ of size at most $|S| = O(\epsilon^{-2})$ which contains all $i \in [n]$ with $|\mathcal{X}_i| \geq \epsilon\|\mathcal{X}\|_2$ (such coordinates are called heavy hitters). The lower bound for heavy hitters is simply the result of the space required to store the $\log(n)$-bit identities of all possible ϵ^{-2} heavy hitters. Note that for the heavy hitters problem alone, there is an optimal streaming $O(\epsilon^{-2}\log(n))$-bits of space upper bound called BPTree [BCI+16]. However, BPTree cannot be used in the general distributed setting, since it crucially relies on the sequential natural of a stream.

Next, we demonstrate that F_p estimation for $p < 1$ is in fact possible with max communication independent of the graph topology. After derandomizing our protocol, this results in a optimal streaming algorithm for F_p estimation, $p < 1$, which closes a long line of research on the problem for this particular range of p [Woo04, Ind06, KNW10a, KNPW11a, CCM16, BVWY18].

Theorem 32 *For $p \in (0,1)$, there is a protocol for F_p estimation in the message passing model which succeeds with probability $2/3$ and has max-communication of $O(\frac{1}{\epsilon^2}(\log\log n + \log 1/\epsilon))$.*

Theorem 33 *There is an insertion only streaming algorithm for F_p estimation, $p \in (0,1)$, which outputs a $(1\pm\epsilon)$ approximation to $\|x\|_p$ with probability at least $2/3$. The algorithm uses $O((\frac{1}{\epsilon^2}(\log\log n + \log 1/\epsilon) + \frac{\log 1/\epsilon}{\log\log 1/\epsilon}\log n)$-bits of space. In the random oracle model, the space is $O(\frac{1}{\epsilon^2}(\log\log n + \log 1/\epsilon))$.*

The above bound matches the $\Omega(\epsilon^{-2})$ max communication lower bound of [Woo04] in the shared randomness model, which comes from 2-party communication complexity. Moreover, our streaming algorithm matches the $\Omega(\log n)$ lower bound for streaming when a random oracle is not allowed. Our results are derived from observations about the behavior of p-stable distributions for $p < 1$, followed by a careful analysis of their tail-behavior. Namely, if $Z_1, \ldots, Z_n \sim \mathcal{D}_p$ are p-stable for $p \leq 1$, and if $X_1, \ldots, X_m \in \mathbb{R}^n$ are non-negative, then $\sum_{i=1}^m |Z_i X_i|$ is roughly on the same order as $\sum_{i=1}^m Z_i X_i$, which allows us to approximate the latter in small space via

approximate counters.

We also point out an alternative approach for deriving the upper bound of Theorem 32, which is to use *maximally-skewed p-stable distributions* [Nol, Li09, LZ11]. These distributions have the property for that they are always positive (or negative) for $p < 1$, and one could then follow a similar analysis to that provided in this Chapter. We chose instead to consider general p-stable variables, and not specifically maximally skewed variables, for several reasons. Firstly, for the purposes of streaming, our derandomization utilizes ideas from [KNW10a], which only consider un-skewed symmetric stable variables. Thus, to utilize maximally skewed distributions, one would need to adapt the arguments of [KNW10a]. Moreover, our arguments give general results for the behavior of p-stables, and therefore also apply to the maximally-skewed case.

We remark that F_p estimation in the range $p \in (0, 1)$ is useful for several reasons. Firstly, for p near 1, F_p estimation is often used as a subroutine for estimating the empirical entropy of a stream, which itself is useful for network anomaly detection ([LZ11], also see [HNO08a] and the references therein). Moment estimation is also used in weighted sampling algorithms for data streams [MW10, JST11, JW18b] (see [CJ19] for a survey of such samplers and their applications). Which can be used as subroutines for other tasks, such as finding heavy-hitters in the stream, estimating cascaded norms [AKO11, MW10], and designing representative histograms of the data on which more complicated algorithms are run [GMP, GM98b, Olk93, GKMS02, HNG+07, CMR05]. Furthermore, moment estimation for fractional p, such as $p = .5$ and $p = .25$, has been shown to be useful for data mining [CIKM02].

As another application of our protocol for F_p estimation, $p < 1$, we demonstrate a communication optimal protocol for additive approximation of the empirical *Shannon entropy* $H(\mathcal{X})$ of the aggregate vector $\mathcal{X}$. Here, $H = H(\mathcal{X})$ is defined by $H = \sum_{i=1}^{n} p_i \log(1/p_i)$ where $p_i = |\mathcal{X}_i|/\|\mathcal{X}\|_1$ for $i \in [n]$. The goal of our protocols is to produce an estimate $\tilde{H} \in \mathbb{R}$ of H such that $|\tilde{H} - H| \leq \epsilon$. Our result is as follows.

Theorem 35 *There is a multi-party communication protocol in the message passing model that outputs a ϵ-additive error of the Shannon entropy H. The protocol uses a max-communication of $O(\frac{1}{\epsilon^2}(\log\log(n) + \log(1/\epsilon))$-bits.*

Note that for a *multiplicative* approximation of the Shannon entropy, there is a $\tilde{\Omega}(\epsilon^{-2})$ lower bound [CCM10]. For additive estimation, [KN14] gives a $\Omega(\epsilon^{-2}\log(n))$ lower bound in the turnstile model. Using a similar reduction, we prove a matching $\Omega(\epsilon^{-2})$ lower bound for additive ϵ

approximation in the insertion only model (see Section 4.7.2 for the proof). Furthermore, our protocol directly results in an $\tilde{O}(\epsilon^{-2})$-bits of space, insertion only *streaming* algorithm for entropy estimation in the random oracle model. We note that most lower bounds in communication complexity (and all of the bounds discussed in this paper except for the $\Omega(\log n)$ term in the lower bound for F_p estimation) also apply to the random oracle model. Previously, the best known algorithm for the insertion only random oracle model used $O(\epsilon^{-2}\log n)$-bits [LZ11, CC13], whereas the best known algorithm for the non-random oracle model uses $O(\epsilon^{-2}\log^2 n)$-bits (the extra factor of $\log n$ comes from a standard application of Nisan's pseudo-random generator [Nis92]).

Theorem 36 *There is an insertion only streaming algorithm for ϵ-additive approximation of the empirical Shannon entropy of an insertion only stream in the random oracle model, which succeeds with probability $3/4$. The space required by the algorithm is $O(\frac{1}{\epsilon^2}(\log\log(n)+\log(1/\epsilon))$ bits.*

Finally, we show how our techniques can be applied to the important numerical linear algebraic primitive of *approximate matrix product*, which we now define.

Definition 4.1.1. *The multi-party approximate matrix product problem is defined as follows. Instead of vector valued inputs, each player is given $\mathbf{X}_i \in \{0,1,\ldots,M\}^{n\times t_1}$ and $\mathbf{Y}_i \in \{0,1,\ldots,M\}^{n\times t_2}$, where $\boldsymbol{\mathcal{X}} = \sum_i \mathbf{X}_i$ and $\boldsymbol{\mathcal{Y}} = \sum_i \mathbf{Y}_i$. Here, it is generally assumed that $n >> t_1, t_2$ (but not required). The players must work together to jointly compute a matrix $\mathbf{R} \in \mathbb{R}^{t_1\times t_2}$ such that*

$$\|\mathbf{R} - \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{Y}}\|_F \leq \epsilon\|\boldsymbol{\mathcal{X}}\|_F\|\boldsymbol{\mathcal{Y}}\|_F$$

Where for a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, $\|\mathbf{A}\|_F = (\sum_{i=1}^n\sum_{j=1}^m \mathbf{A}_{i,j}^2)^{1/2}$ is the Frobenius norm of $\mathbf{A}$.

Theorem 37 *There is a protocol which outputs, at the central vertex $\mathcal{C}$, a matrix $\mathbf{R} \in \mathbb{R}^{t_1\times t_2}$ which solves the approximate communication protocol with probability $3/4$ [2]. The max communication required by the protocol is $O\left(\epsilon^{-2}(t_1+t_2)(\log\log n + \log 1/\epsilon + \log d)\right)$, where d is the diameter of the communication topology G.*

We remark that an upper bound of $O\left(\epsilon^{-2}(t_1+t_2)\log n\right)$ was already well-known from sketching theory [W+14], and our main improvement is removing the $\log n$ factor for small diameter

[2] We remark that there are standard techniques to boost the probability of the matrix sketching results to $1-\delta$, using a blow-up of $\log(\delta)$ in the communication. See e.g. Section 2.3 of [W+14]

graphs, such as the coordinator model where distributed numerical linear algebra is usually considered.

4.1.2 Related Work for Moment Estimation in Streaming and Distributed Models

As mentioned, a closely related line of work is in the *distributed functional monitoring model*. Here, there are m machines connected to a central coordinator (the coordinator topology). Each machine then receives a stream of updates, and the coordinator must maintain at all time steps an approximation of some function, such as a moment estimation or a uniform sample, of the union of all streams. We note that there are two slightly different models here. One model is where the items (coordinates) being updated in the separate streams are considered disjoint, and each time an insertion is seen it is to a unique item. This model is considered especially for the problem of maintaining a uniform sample of the items in the streams [CMY11, HYZ12, TW11, JSTW19]. The other model, which is more related to ours, is where each player is receiving a stream of updates to a *shared* overall data vector $\mathcal{X} \in \mathbb{R}^n$. This can be seen as a distributed streaming setting, where the updates to a centralized stream are split over m servers, and is considered in [WZ12, CMY11, ABC09a]. For the restricted setting of *one-way* algorithms, which only transmit messages from the sites to the coordinators, any such algorithm can be made into a one-shot protocol for the multi-party message passing model. Here, each machine just simulates a stream on their fixed input vectors X_i, and sends all the messages that would have been sent by the functional monitoring protocol.

Perhaps the most directly related result to our upper bound for for F_p estimation, $p \in (1, 2]$, is in the distributed functional monitoring model, where Woodruff and Zhang [WZ12] show a $O(m^{p-1}\text{poly}(\log(n), 1/\epsilon) + m\epsilon^{-1}\log(n)\log(\log(n)/\epsilon))$[3] *total communication* upper bound. We remark here, however, that the result of [WZ12] is incomparable to ours for several reasons. Firstly, their bounds are only for total communication, whereas their max communication can be substantially larger than $O(1/\epsilon^2)$. Secondly, while it is claimed in the introduction that the protocols are one way (i.e., only the players speak to the coordinator, and not vice versa), this is for their threshold problem and not for F_p estimation[4]. As remarked before, there is an $\Omega(m/\epsilon^2)$ to-

[3] We remark that the $\text{poly}(\log(n), 1/\epsilon)$ terms here are rather large, and not specified in the analysis of [WZ12].

[4] The reason for this is as follows. Their algorithm reduces F_p estimation to the threshold problem, where for a threshold τ, the coordinator outputs 1 when the F_p first exceeds $\tau(1+\epsilon)$, and outputs 0 whenever the F_p is below $\tau(1-\epsilon)$. To solve F_p estimation, one then runs this threshold procedure for the $\log(mMn)/\epsilon$ thresholds $\tau = (1+\epsilon), (1+\epsilon)^2, \ldots, (mMn)^2$ in parallel. However, the analysis from [WZ12] only demonstrates a total communication

tal communication lower bound for one-way protocols, which demonstrates that their complexity could not hold in our setting (we sketch a proof of this in Section 4.7.1).

The message passing model itself has been the subject of significant research interest over the past two decades. The majority of this Chapter is concerned with *exact* computation of Boolean functions of the inputs. Perhaps the canonical multi-party problem, and one which has strong applications to streaming, is set disjointness, where each player has a subset $S_i \subset [n]$ and the players want to know if $\cap_{i=1}^{m} S_i$ is empty. Bar-Yossef et al. [BYJKS04] demonstrated strong bounds for this problem in the black-board model. This lower bound resulted in improved (polynomially sized) lower bounds for streaming F_p estimation for $p > 2$. These results for disjointness have since been generalized and improved using new techniques [CKS03, Gro09, Jay09, BEO+13]. Finally, we remark that while most results in the multi-party message passing model are not topology dependent, Chattopadhyay, Radhakrishnan, and Rudra have demonstrated that tighter topology-dependent lower bounds are indeed possible in the message passing model [CRR14].

Road Map for the Chapter. In Section 4.2, we formally introduce the message passing model. In Section 4.3, we give our main F_p estimation algorithm in the message passing model for $p > 1$. In Section 4.3.2, we provide our algorithm for point-estimation and heavy hitters in the message passing model. In Section 4.4 we give our F_p estimation algorithm for $p < 1$ in the message passing and streaming model. In Section 4.5, we give our algorithm for entropy estimation, and in Section 4.6 we give our algorithm for approximate matrix product.

4.2 Communication Complexity and the Message Passing Model

In this section we introduce the relevant preliminaries on Communication Complexity and the Message Passing Model. Let f be a function $f : \mathbb{R}^n \to \mathbb{R}$. Let $G = (V, E)$ be a connected undirected graph with m vertices, i.e. $V = \{1, \ldots, m\}$. In the message passing model on the graph topology G, there are m players, each placed at a unique vertex of G, with unbounded computational power. Player i is given as input only a vector $X_i \in \mathbb{Z}^n$, which is known as the Number in Hand (NIH) model of communication. Let $\mathcal{X} = \sum_{i=1}^{n} X_i$ be the aggregate vector of the players inputs. The goal of the players is to jointly compute or approximate the function $f(\mathcal{X})$ by carrying out some previously unanimously agreed upon communication protocol. It is

of $O(k^{1-p}\text{poly}(\log(n), \epsilon^{-1}))$ for the time steps *before* the threshold τ is reached. Once the threshold is reached, the communication would increase significantly, thus the coordinator must inform all players when a threshold τ is reached so that they stop sending messages for τ, violating the one-way property. This step also requires an additive k messages for each of the $O(\epsilon^{-1}\log(n))$ thresholds, which results in the $O(m\epsilon^{-1}\log(n)\log(\log(n)\epsilon)))$ term.

assumed that the graph topology of G is known to all players.

In this paper, we are concerned with the non-negative input model. Namely, the inputs X_i satisfy $X_i \in \{0, 1, \ldots, M\}^n$ for all players i. Note an equivalent assumption to is that $(X_i)_j \geq 0$ for all i, and that the $(X_i)_j$'s can be specified in $O(\log(M))$ bits. As in the streaming model, we assume that $m, M = O(n^c)$ for some constant c, which allows us to simplify complexity bounds and write $\log(nmM) = O(\log n)$.

During execution of the protocol, a player $i \in V$ is only allowed to send a message to a player j if $(i, j) \in E$. Thus, players may only communicate directly with their neighbors in the graph G. In contrast to the *broadcast* and *blackboard* models of communication, in the message passing model the message sent by player i to player j is only received by player j, and no other player. Upon termination of the protocol, at least one player must hold an approximation of the value $f(\mathcal{X})$. For the protocols considered in this paper, this player will be fixed and specified by the protocol beforehand. We use $\mathcal{C} \in V$ to denote the distinguished player specified by the protocol to store the approximation at the end of the execution.

Every such communication protocol in this model can be divided into rounds, where on the j-th round some subset $S_j \subseteq V$ of the players simultaneously send a message across one of their edges. Although it is not a restriction in the message passing model, our protocols satisfy the additional property that each player communicates *exactly once*, across one of its edges, and that each player will receive messages from its neighbors in exactly one round. Specifically, for each player i, there will be exactly one round j where some subset of its neighbors send player i a message, and then player i will send a single message in round $j + 1$, and never again communicate. Such protocols are called *one-shot* protocols.

The *total communication* cost of a protocol is the total number of bits sent in all the messages during its execution. The *max-communication* of a protocol is the maximum number of bits sent across any edge over the execution of the protocol. Communication protocols can be either deterministic or randomized. In this paper we consider the standard *public-coin* model of communication, where each player is given shared access to an arbitrarily long string of random bits. This allows players to jointly utilize the same source of randomness without having to communicate it.

4.3 Message Passing F_p Estimation for $p > 1$

In this section, we provide our algorithm for F_p estimation, $1 \leq p \leq 2$, in the message passing model with max communication $O(\frac{1}{\epsilon^2}(\log d + \log\log + \log 1/\epsilon))$, where d is the diameter of the graph G. We begin by specifying the distinguished vertex $\mathcal{C} \in V$ which will hold and output the F_p approximation at the end of the protocol. For a vertex $v \in G$, define its eccentricity $\mathtt{ecc}(v) = \max_{u \in V} d(v,u)$, where $d(v,u)$ is the graph distance between v, u. We then set $\mathcal{C} \in V$ to be any vertex with minimal eccentricity. Such a vertex is known as a center of G. We now fix a shortest path spanning tree T for G, rooted at the distinguished player $\mathcal{C}$. The spanning tree T has the property that the path between $\mathcal{C}$ and any vertex $v \in V$ in the tree T is also a shortest path between $\mathcal{C}$ and v in G. Thus the distance between $\mathcal{C}$ and any vertex $v \in V$ is the same in T as it is in G. The fact that the depth of T is at most d, where d is the diameter of G, now follows naturally. Such a shortest path spanning tree T can be easily obtained via a breath first search.

Our algorithms for F_p estimation and heavy hitters are based on a sketching step, followed by a randomized rounding procedure. Specifically, the players jointly generate a randomized matrix $S \in \mathbb{R}^{k \times n}$, such that one can first compute $\sum_i SX_i = S\mathcal{X} \in \mathbb{R}^k$, where $k << n$, and deduce the desired properties of $\mathcal{X}$ from this small sketch. Thus, for each coordinate $j \in [k]$, the players could all sketch their data SX_i, and send it up the tree T to the distinguish vertex $\mathcal{C}$. To improve the communication complexity required to send each coordinate of SX_i, the players randomly round the entries in SX_i and send these instead, using only $O(\log\log(mM) + \log(1/\epsilon) + \log(d))$ bits of space.

Before we introduce the randomized rounding protocol, we will need a technical Lemma about the behavior of p-stable random variables (see Definition 2.2.6). To prove it, we first use the following fact about the tails of p stable distributions, which can be found in [Nol].

Proposition 4.3.1. *If $Z \sim \mathcal{D}_p$ for $0 < p < 2$, then* $\mathbf{Pr}[|Z| \geq \lambda] \leq O(\frac{1}{\lambda^p})$.

Also, we use the straightforward fact that $\|X_i\|_p^p \leq \|\sum_{i=1}^m X_i\|_p^p$ for non-negative vectors X_i and $p \geq 1$.

Fact 4.3.2. *If $X_1, \ldots, X_m \in \mathbb{R}^n$ are entry-wise non-negative vectors and $1 \leq p \leq 2$, then $\sum_{i=1}^m \|X_i\|_p^p \leq \|\sum_{i=1}^m X_i\|_p^p$.*

Proof. It suffices to consider the values coordinate-wise. If $x, y \geq 0$ and $1 \leq p \leq 2$, then $x^{p-1} \leq (x+y)^{p-1}$, so $x^p + y^p \leq x(x+y)^{p-1} + y(x+y)^{p-1} = (x+y)^p$, and the general result

then follows from induction on the number of vectors m. $\square$

This following lemma will be fundamental to our analysis. Recall by the p-stability of the distribution $\mathcal{D}_p$, if $Z \sim \mathcal{D}_p^n$ and $X_i \in \mathbb{R}^n$ is a vector, then we have $\langle Z, X_i \rangle \sim z\|X_i\|_p$, where $z \sim \mathcal{D}_p$. Thus, we have then that $\sum_{i=1}^m |\langle Z, X_i \rangle|^q \sim \sum_{i=1}^m z_i^q \|X_i\|_p^q$ for any $q > 0$ and vectors $X_1, \ldots, X_m \in \mathbb{R}^n$. We would like to use the fact that for $1 \leq p \leq q$, we have $\sum_{i=1}^m \|X_i\|_p^q \leq \|\sum_{i=1}^m X_i\|_p^q = \|\mathcal{X}\|_p^q$ if the X_i's are non-negative, and then bound the whole sum by $O(\|\mathcal{X}\|_p^q)$ with good probability. However, there are several complications. First, note that the z_i's are not independent, since they are all generated by the same Z. Second, note that the z_i's have heavy tails, so $\mathbf{E}[z_i^q]$ will generally be infinite (e.g., if $p < 2$ and $q = 2$). Thus, we must be careful when attempting to bound the overall probability that this sum is large. To do so, we use a level-set analysis for variables with power-law tails.

Lemma 4.3.3. *Fix $1 \leq p \leq q \leq 2$, and let $Z = (Z_1, Z_2, \ldots, Z_n) \sim \mathcal{D}_p^m$. Suppose $X_1, \ldots, X_m \in \mathbb{R}^n$ are non-negative vectors, with $\mathcal{X} = \sum_j X_j$. Then for any $\lambda \geq 1$, if either $q - p \geq c > 0$ for some constant c independent of m, or if $p = 2$, we have*

$$\mathbf{Pr}\left[\sum_{j=1}^m |\langle Z, X_j \rangle|^q \geq C\lambda^q \|\mathcal{X}\|_p^q\right] \leq \frac{1}{\lambda^p}$$

Otherwise, we have

$$\mathbf{Pr}\left[\sum_{j=1}^m |\langle Z, X_j \rangle|^q \geq C\log(\lambda m)\lambda^q \|\mathcal{X}\|_p^q\right] \leq \frac{1}{\lambda^p}$$

where C is some constant (depending only on c in the first case)[5].

Proof. First note, for $p = 2$, each term $\sum_{j=1}^m |\langle Z, X_j \rangle|^2$ is distributed as $g^2\|X_j\|_2^2$, where g is Gaussian, thus g^2 is χ^2 distributed with $\mathbf{E}[g^2] = 1$. It follows that $\mathbf{E}\left[\sum_{j=1}^m |\langle Z, X_j \rangle|^2\right] = \sum_{j=1}^m \|X_j\|_2^2 \leq \|\mathcal{X}\|_2^2$, where the inequality follows from the Fact 4.3.2, and the result then follows from Markov's inequality.

Now suppose $p \in [1, 2)$. Again, by p-stability we have $\sum_{j=1}^m |\langle Z, X_j \rangle|^q = \sum_{j=1}^m \|X_j\|_p^q \hat{Z}_j^q$, where $\hat{Z}_j \sim \mathcal{D}_p$. Note though that the $\hat{Z}_j$'s are not independent. Define $I_k = \{j \in [m] \mid 2^{-k}\|\mathcal{X}\|_p^q \leq$

[5]Observe that the extra $\log(m)$ factor is necessary in general when $p = q$. Note that if the support of all the $X_i's$ are distinct indicator vectors e_i, then the sum is the sum of the p-th powers of independent p-stables. If $p = 1$, this is a sum of m Cauchy random variables, which we expect to be $\Omega(m\log(m))$. For $1 \neq p < 2$ and $p = q$, the tail behavior of p-stables raised to the p-th power is asymptotically similar to that of Cauchy's, so the result is the same here.

$\hat{Z}_j^q \|X_j\|_p^q \leq 2^{-k+1}\|\mathcal{X}\|_p^q\}$. So if $j \in I_k$, then we have $\hat{Z}_j^q \geq 2^{-k}\frac{\|X\|_p^q}{|X_j|_p^q}$. For each k, the goal is to bound the contribution of the terms in the set I_k to the overall sum. By Proposition 4.3.1, there is some constant $c \geq 1$ such that

$$\begin{aligned}\mathbf{Pr}\left[j \in I_k\right] &\leq c\left(2^k \frac{\|X_j\|_p^q}{\|\mathcal{X}\|_p^q}\right)^{p/q} \\ &= c2^{pk/q}\frac{\|X_j\|_p^p}{\|\mathcal{X}\|_p^p}\end{aligned} \tag{4.1}$$

By Fact 4.3.2, we have $\sum_j \|X_j\|_p^p \leq \|\mathcal{X}\|_p^p$. So we obtain $\mathbf{E}\left[|I_k|\right] \leq c2^{pk/q}\sum_j \frac{\|X_j\|_p^p}{\|\mathcal{X}\|_p^p} \leq c2^{pk/q}$ for any $k \in \mathbb{Z}$.

Now consider the event $\mathcal{E}_0$ that $I_k = \emptyset$ for all $k \leq -\frac{q}{p}\log(10C'\lambda)$. We first analyze $\mathbf{Pr}\left[\mathcal{E}_0\right]$. Note that for a fixed j, the probability that $j \in I_k$ for some $k \leq -\frac{q}{p}\log(10C'\lambda)$ is at most $\lambda^{-1}\frac{\|X_j\|_p^p}{\|\mathcal{X}\|_p^p}/10$. So the expected number of $j \in [n]$ with $j \in I_k$ for some $k \leq -\frac{q}{p}\log(10C'\lambda)$ is at most $\lambda^{-1}\sum_j \frac{\|X_j\|_p^p}{\|\mathcal{X}\|_p^p}/10 \leq \lambda^{-1}/10$. So by Markov's inequality, we conclude that $\mathbf{Pr}\left[\mathcal{E}_0\right] \leq \frac{1}{10\lambda}$. Now note that for any $k \in \mathbb{Z}$, the contribution of the items $j \in I_k$ is at most $|I_k|2^{-k+1}\|\mathcal{X}\|_p^q$. Thus, an upper bound on the expected contribution of the items in sets I_k for $-\frac{q}{p}\log(10C'\lambda) \leq k \leq 4\log(m\lambda)$ is given by

$$\sum_{j=-\frac{q}{p}\log(10C'\lambda)}^{4\log(m\lambda)} \mathbf{E}\left[|I_k|2^{-k+1}\|\mathcal{X}\|_p^q\right] \leq \sum_{j=-\frac{q}{p}\log(10C'\lambda)}^{4\log(m\lambda)} c2^{-k(1-p/q)}\|\mathcal{X}\|_p^q$$

Now if $q - p > c'$ for some constant c' independent of m, then the above sum is geometric, and at most $O(2^{(\frac{q}{p}-1)\log(10C'\lambda)}\|\mathcal{X}\|_p^q) = O((10C'\lambda)^{q/p-1}\|\mathcal{X}\|_p^q)$. If p/q is arbitrarily small, the above can be bounded by $O(\log(m\lambda)(10C'\lambda)^{q/p-1}\|\mathcal{X}\|_p^q)$. Setting $\delta = O(\log(m\lambda))$ if this is the case, and $\delta = 1$ otherwise, we can apply Markov's to obtain:

$$\mathbf{Pr}\left[\sum_{j=-\frac{q}{p}\log(10C'\lambda)}^{4\log(m\lambda)} |I_k|2^{-k+1}\|\mathcal{X}\|_p^q > \delta(10C'\lambda)^{q/p}\|\mathcal{X}\|_p^q\right] < \frac{1}{10C'\lambda}$$

Call the above event $\mathcal{E}_1$. Conditioned on $\mathcal{E}_0 \cup \mathcal{E}_1$ not occurring, which occurs with probability at least $1 - \frac{1}{\lambda}$, it follows that the contribution of the items in level sets I_k for $k \leq 4\log(m\lambda)$ is at most $O(\delta\lambda^{q/p}\|\mathcal{X}\|_p^q)$. Now for $k \geq 4\log(m\lambda)$, note that the contribution of any term $j \in I_k$ is at most $\frac{1}{m^2}\|\mathcal{X}\|_p^2$, and so the contribution of all such items to the total sum is at most $\frac{1}{m}\|\mathcal{X}\|_p^2$. Thus $\sum_{j=1}^m |\langle Z, X_j\rangle|^q = O(\delta\lambda^{q/p}\|\mathcal{X}\|_p^q)$ with probability at least $1/\lambda$, which is the desired result after

replacing λ with λ^p as needed.

□

While the above result was described to specifically hold only for p-stable random variables, it is straightforward to show that the result for $p = 2$ holds when Gaussian random variables are replaced with Rademacher random variables: i.e., variables Z_i that are uniform over $\{1, -1\}$.

Corollary 4.3.4. *Suppose $Z = (Z_1, \ldots, Z_m)$ where the Z_i's are uniform over $\{1, -1\}$ and pairwise independent, and let $X_1, \ldots, X_m$ be non-negative vectors with $\mathcal{X} = \sum_j X_j$. Then for any $\lambda \geq 1$, we have*

$$\mathbf{Pr}\left[\sum_{j=1}^{m} |\langle Z, X_j\rangle|^2 \geq \lambda \|\mathcal{X}\|_2^2\right] \leq \frac{1}{\lambda}$$

Proof. We have $\mathbf{E}\left[|\langle Z, X_j\rangle|^2\right] = \|X_j\|_2^2$, so the result follows from an application of Markov's inequality and using the fact that $\sum_j \|X_j\|_2^2 \leq \|\mathcal{X}\|_2^2$. □

We now note that for $p = 2$, we obtain much stronger concentration. We will need this for the approximate matrix product section.

Corollary 4.3.5. *Let $Z = (Z_1, Z_2, \ldots, Z_n) \sim \mathcal{D}_2^m$ be i.i.d. Gaussian. Suppose $X_1, \ldots, X_m \in \mathbb{R}^n$ are non-negative vectors, with $\mathcal{X} = \sum_j X_j$. Then for any $\lambda \geq c \log(m)$ for some sufficiently large constant c, we have*

$$\mathbf{Pr}\left[\sum_{j=1}^{m} |\langle Z, X_j\rangle| \geq \lambda \|\mathcal{X}\|_2^2\right] \leq \exp(-C\lambda)$$

where C is some universal constant.

Proof. Recall for $j \in [m]$, $|\langle Z, X_j\rangle|^2$ is distributed as $g_j^2 \|X_j\|_2^2$, where g_j is Gaussian, thus g_j^2 is χ^2 distributed with $\mathbf{E}\left[g_j^2\right] = 1$. Now if z is χ^2 distributed, we have $\mathbf{Pr}\left[z > \lambda\right] < e^{-C\lambda}$ for some absolute constant C. So $\mathbf{Pr}\left[|\langle Z, X_j\rangle|^2 > \log(m)\lambda\right] < e^{-C\lambda}/m$ for a slightly different constant C. We can union bound over all m terms, so we have $\sum_{j=1}^m |\langle Z, X_j\rangle|^2 \leq \sum_{j=1}^m \lambda \|X_j\|_2^2 \leq \lambda \|\mathcal{X}\|_2^2$ with probability $1 - \exp(-C\lambda)$ as needed. □

4.3.1 Randomized Rounding of Sketches

We now introduce our randomized rounding protocol. Consider non-negative integral vectors $X_1, X_2, \ldots, X_m \in \mathbb{Z}^n_{\geq 0}$, with $\mathcal{X} = \sum_{i=1}^n X_i$. Fix a message passing topology $G = (V, E)$, where each player $i \in V$ is given as input X_i. Fix any vertex $\mathcal{C}$ that is a center of G, and let T be a shortest path spanning tree of G rooted at $\mathcal{C}$ as described at the beginning of the section. Let d be the depth of T. The players use shared randomness to choose a random vector $Z \in \mathbb{R}^n$, and their goal is to approximately compute $\langle Z, \mathcal{X} \rangle = \langle Z, \sum_{i=1}^m X_i \rangle$. The goal of this section is to develop a d-round randomized rounding protocol, so that at the end of the protocol the approximation to $\langle Z, \mathcal{X} \rangle$ is stored at the vertex $\mathcal{C}$.

We begin by introducing the rounding primitive which we use in the protocol. Fix $\epsilon > 0$, and let $\gamma = (\epsilon\delta / \log(nm))^C$, for a sufficiently large constant $C > 1$. For any real value $r \in \mathbb{R}$, let $i_r \in \mathbb{Z}$ and $\alpha_i \in \{1, -1\}$ be such that $(1+\gamma)^{i_r} \leq \alpha_i r \leq (1+\gamma)^{i_r+1}$. Now fix p_r such that:

$$\alpha_i r = p_r(1+\gamma)^{i_r+1} + (1-p_r)(1+\gamma)^{i_r}$$

We then define the rounding random variable $\Gamma(r)$ by

$$\Gamma(r) = \begin{cases} 0 & \text{if } r = 0 \\ \alpha_i(1+\gamma)^{i_r+1} & \text{with probability } p_r \\ \alpha_i(1+\gamma)^{i_r} & \text{with probability } 1 - p_r \end{cases}$$

The following proposition is clear from the construction of p_r and the fact that the error is deterministically bounded by $\gamma|r|$.

Proposition 4.3.6. *For any $r \in R$, We have $\mathbf{E}[\Gamma(r)] = r$ and Var$(\Gamma(r)) \leq r^2\gamma^2$*

We will now describe our rounding protocol. We partition T into d layers, so that all nodes at distance $d - t$ from $\mathcal{C}$ in T are put in layer t. Define $L_t \subset [n]$ to be the set of players at layer t in the tree. For any vertex $u \in G$, let T_u be the subtree of T rooted at u (including the vertex u). For any player i, let $C_i \subset [n]$ be the set of children of i in the tree T. The procedure for all players $j \in V$ is then given in Figure 4.2.

For each player i in layer 0, they take their input X_i, and compute $\langle Z, X_i \rangle$. They then round their values as $r_i = \Gamma(\langle Z, X_i \rangle)$, where the randomness used for the rounding function Γ is drawn independently for each call to Γ. Then player i sends r_i to their parent in T. In general,

Procedure for node j in layer i

1. Choose random vector $Z \in \mathbb{R}^n$ using shared randomness.
2. Receive rounded sketches $r_{j_1}, r_{j_2}, \ldots, r_{j_{t_j}} \in \mathbb{R}$ from the t_j children of node j in the prior layer (if any such children exist).
3. Compute $x_j = \langle X_j, Z \rangle + r_{j_1} + r_{j_2} + \cdots + r_{j_t} \in \mathbb{R}$.
4. Compute $r_j = \Gamma(x_j)$. If player $j \neq \mathcal{C}$, then send r_j it to the parent node of j in T. If $j = \mathcal{C}$, then output r_j as the approximation to $\langle Z, \mathcal{X} \rangle$.

Figure 4.2: Recursive Randomized Rounding

consider any player i at depth $j > 0$ of T. At the end of the j-th round, player i will receive a rounded value r_ℓ for every child vertex $\ell \in C_i$. They then compute $x_i = \langle Z, X_i \rangle + \sum_{\ell \in C_i} r_\ell$, and $r_i = \Gamma(x_i)$, and send r_i to their parent in T. This continues until, on round d, the center vertex $\mathcal{C}$ receives r_ℓ for all children $\ell \in C_{\mathcal{C}}$. The center $\mathcal{C}$ then outputs $r_{\mathcal{C}} = \langle Z, X_{\mathcal{C}} \rangle + \sum_{\ell \in C_{\mathcal{C}}} r_\ell$ as the approximation.

We will now demonstrate that if the entries Z_i of Z are drawn independently from a p-stable distribution, we obtain a good estimate of the product $\langle Z, \mathcal{X} \rangle$. For any player i, let $Q_i = \sum_{u \in T_i} X_u$, and $y_i = \langle Z, Q_i \rangle$. Then define the error e_i at player i as $e_i = y_i - r_i$. We first prove a proposition that states the expectation of the error e_i for any player i is zero, and then the main lemma which bounds the variance of e_i. The error bound of the protocol at $\mathcal{C}$ then results from an application of Chebyshev's inequality.

Proposition 4.3.7. *For any player i, we have $\mathbf{E}[e_i] = 0$. Moreover, for any players i, j such that $i \notin T_j$ and $j \notin T_i$, the variables e_i and e_j are statistically independent.*

Proof. We prove the first claim by induction on the layer j such that $i \in L_j$. For the base case of $j = 0$, player i has no children, so $e_i = r - \Gamma(r)$ for some $r \in \mathbb{R}$, from which the result follows from the fact that $\Gamma(r)$ being an unbiased estimate of r. Now supposing the result holds for all players in L_j, and let $i \in L_{j+1}$ with children $i_1, \ldots, i_k \in [n]$. Then

$$\begin{aligned}\mathbf{E}\left[e_i\right] &= \mathbf{E}\left[y_i - \Gamma(y_i + e_{i_1} + \cdots + e_{i_k})\right] \\ &= \mathbb{E}_{e_{i_1},\ldots,e_{i_k}}\left[\mathbf{E}\left[y_i - \Gamma(y_i + e_{i_1} + \cdots + e_{i_k}) \;\middle|\; e_{i_1},\ldots,e_{i_k}\right]\right] \\ &= \mathbb{E}_{e_{i_1},\ldots,e_{i_k}}\left[e_{i_1} + \cdots + e_{i_k}\right] \\ &= 0\end{aligned} \tag{4.2}$$

which completes the first claim. The second claim follows from the fact that the randomness used to generate e_i and e_j is distinct if the subtrees of player i and j are disjoint. □

Lemma 4.3.8. *Fix $p \in [1,2]$, and let $Z = (Z_1, Z_2, \ldots, Z_n) \sim \mathcal{D}_p^n$. Then the above procedure when run on $\gamma = (\epsilon\delta/(d\log(nm)))^C$ for a sufficiently large constant C, produces an estimate $r_{\mathcal{C}}$ of $\langle Z, \mathcal{X}\rangle$, held at the center vertex $\mathcal{C}$, such that $\mathbf{E}\left[r_{\mathcal{C}}\right] = \langle Z, \mathcal{X}\rangle$. Moreover, over the randomness used to draw Z, with probability $1-\delta$ for $p < 2$, and with probability $1 - e^{-1/\delta}$ for Gaussian Z, we have $\mathbf{E}\left[(r_{\mathcal{C}} - \langle Z, \mathcal{X}\rangle)^2\right] \leq (\epsilon/\delta)^2\|\mathcal{X}\|_p$. Thus, with probability at least $1 - O(\delta)$, we have*

$$|r_{\mathcal{C}} - \langle Z, \mathcal{X}\rangle| \leq \epsilon\|\mathcal{X}\|_p$$

Moreover, if $Z = (Z_1, Z_2, \ldots, Z_n) \in \mathbb{R}^n$ where each $Z_i \in \{1, -1\}$ is a 4-wise independent Rademacher variable, then the above bound holds with $p = 2$ (and with probability $1 - \delta$).

Proof. Set $N = \mathrm{poly}(n,m)^{\mathrm{poly}(d/(\delta\epsilon))}$ to be sufficiently large, and let $\gamma = (\epsilon\delta/(d\log(nm)))^C$ so that $1/\gamma$ is a sufficiently large polynomial in $\log(N)$. Let $\gamma_0 = \gamma(\frac{\log(N)}{\epsilon})^c$ for a constant $c < C$ that we will later choose. For the remainder of the proof, we condition on the event $\mathcal{E}_i$ for $i \in [d]$ that $\sum_{i\in L_t} y_i^2 \leq \log^2(N)\|\sum_{i\in L_t} Q_i\|_p^2$. By Lemma 4.3.3 (or Corollary 4.3.4 for Rademacher variables) and a union bound, we have $\mathbf{Pr}\left[\cup_{j\in[d]}\mathcal{E}_j\right] \leq \log^{-1}(N)$. For Gaussian Z, we have $\mathbf{Pr}\left[\cup_{j\in[d]}\mathcal{E}_j\right] \leq 1/N \leq \exp(-1/\delta)$ by Corollay 4.3.5. Note that $\mathcal{E}_i$ depends only on the randomness used to sample Z, and not on the randomnesses used in the rounding protocol.

We now prove by induction that for any player $i \in L_j$, we have:

$$\mathbf{E}\left[e_i^2\right] \leq (j+1)\gamma_0^2 \sum_{v\in T_i} |\langle Q_v, Z\rangle|^2$$

For the base case, if $i \in L_0$ by definition of the rounding procedure we have $e_i^2 \leq \gamma^2|\langle Q_i, Z\rangle|^2 \leq \gamma_0^2|\langle Q_i, Z\rangle|^2$. Now suppose the result holds for layer j, and let $i \in L_{j+1}$, and let $C_i \subset [n]$ be the children of player i. Then $e_i = \eta_i + \sum_{v\in C_i} e_v$, where η_i is the error induced by the rounding carried out at the player i. Then η_i is obtained from rounding the value $\left(\langle Q_i, Z\rangle + \sum_{v\in C_i} e_v\right)$,

thus

$$\begin{aligned}\mathbf{E}\left[\eta_i^2\right] &\leq \gamma^2\,\mathbf{E}\left[\left(\langle Q_i, Z\rangle + \sum_{v\in C_i} e_v\right)^2\right] \\ &= \gamma^2\left(|\langle Q_i, Z\rangle| + \mathbf{E}\left[\sum_{v\in C_i} e_v^2\right]\right) \\ &\leq \gamma^2\left(|\langle Q_i, Z\rangle| + (j+1)\gamma_0^2 \sum_{v\in C_i}\sum_{u\in T_v} |\langle Q_u, Z\rangle|^2\right)\end{aligned} \tag{4.3}$$

Where the first equality holds by the independence and mean 0 properties of the errors e_v, and the last inequality holds by induction. Thus

$$\begin{aligned}\mathbf{E}\left[e_i^2\right] &= \mathbf{E}\left[(\eta_i + \sum_{v\in C_i} e_v)^2)\right] \\ &= \mathbf{E}\left[\eta_i^2\right] + \mathbf{E}\left[\sum_{v\in C_i} e_v^2\right] \\ &\leq \gamma^2\left(|\langle Q_i, Z\rangle| + (j+1)\gamma_0^2 \sum_{v\in C_i}\sum_{u\in T_v} |\langle Q_u, Z\rangle|^2\right) + \mathbf{E}\left[\sum_{v\in C_i} e_v^2\right] \\ &\leq \gamma_0^2\,|\langle Q_i, Z\rangle| + \gamma_0^2 \sum_{v\in C_i}\sum_{u\in T_v} |\langle Q_u, Z\rangle|^2 + (j+1)\gamma_0^2 \sum_{v\in C_i}\sum_{u\in T_v} |\langle Q_u, Z\rangle|^2 \\ &\leq (j+2)\gamma_0^2\,|\langle Q_i, Z\rangle| + (j+2)\gamma_0^2 \sum_{v\in C_i}\sum_{u\in T_v} |\langle Q_u, Z\rangle|^2 \\ &= (j+2)\gamma_0^2 \sum_{u\in T_i} |\langle Q_u, Z\rangle|^2\end{aligned} \tag{4.4}$$

which is the desired result. The variance bound then follows after conditioning on $\cap_j \mathcal{E}_j$. It follows that:

$$|r_{\mathcal{C}} - \langle Z, \mathcal{X}\rangle|^2 \leq \frac{2}{\delta}\gamma_0^2(d+1)\sum_{u\in T} |\langle Q_u, Z\rangle|^2$$

with probability at least $1 - \delta/2$ by Chebyshev's inequality. We now condition on this and $\cap_j \mathcal{E}_j$, which hold together with probability $1 - \delta$ by a union bound. Since we conditioned on $\cap_j \mathcal{E}_j$, we

have

$$
\begin{aligned}
|r_{\mathcal{C}} - \langle Z, \mathcal{X}\rangle|^2 &\leq \frac{1}{\delta}\gamma_0^2(d+1)\sum_{u\in T}|\langle Q_u, Z\rangle|^2 \\
&= \frac{1}{\delta}\gamma_0^2(d+1)\sum_{i=1}^{d}\sum_{u\in L_i} y_u \\
&\leq \frac{1}{\delta}\gamma_0^2(d+1)\sum_{i=1}^{d}\log^2(N)\|\sum_{u\in L_i} Q_u\|_p^2 \\
&\leq \frac{1}{\delta}\gamma_0^2(d+1)\sum_{i=1}^{d}\log^2(N)\|\mathcal{X}\|_p^2 \\
&\leq \epsilon^2\|\mathcal{X}\|_p^2
\end{aligned}
\tag{4.5}
$$

which completes the proof. Note here, in the second to last inequality, we used the fact that since Q_u are positive vectors, for $p \leq 2$ we have $\sum_{i=1}^{d}\|\sum_{u\in L_i} Q_u\|_p^2 \leq \|\sum_{i=1}^{d}\sum_{u\in L_i} Q_u\|_p^2 \leq \|\mathcal{X}\|_p^2$.

□

Theorem 30. *For $p \in (1, 2]$, there is a protocol for F_p estimation which succeeds with probability $3/4$ in the non-negative data message passing model, which uses a total of $O(\frac{m}{\epsilon^2}(\log(\log(n)) + \log(d) + \log(1/\epsilon)))$ communication, and a max communication of $O(\frac{1}{\epsilon^2}(\log(\log(n)) + \log(d) + \log(1/\epsilon)))$, where d is the diameter of the communication network.*

Proof. We use Indyk's classic p-stable sketch $S \in \mathbb{R}^{k\times n}$, where $k = \Theta(1/\epsilon^2)$, and each entry $S_{i,j} \sim \mathcal{D}_p$. It is standard that computing $\textbf{median}_i\{|(S\mathcal{X})_i|\}$ gives a $(1 \pm \epsilon)$ approximation to $\|\mathcal{X}\|_p$ with probability $3/4$ [Ind06]. By running the randomized rounding protocol on each inner product $\langle S_j, \mathcal{X}\rangle$ independently with error parameter $\epsilon' = \text{poly}(\epsilon)$ and failure parameter $\delta' = \text{poly}(\epsilon)$ (rounding to powers of $(1+\gamma)$ where γ is as defined in this section in terms of ϵ', δ', d), it follows that the central vertex $\mathcal{C}$ recovers $S\mathcal{X}$ up to entry-wise additive $\epsilon'\|\mathcal{X}\|_p$ error with probability $1 - O(\epsilon)$, after union bounding over all k repetitions using Lemma 4.3.8. This changes the median by at most $\epsilon'\|X\|_p$, which results in an $\epsilon'\|\mathcal{X}\|_p$ additive approximation of $\|\mathcal{X}\|_p$, which can be made into a $\epsilon\|\mathcal{X}\|_p$ additive approximation after a rescaling of ϵ.

For the communication bound, note that for each coordinate of $S\mathcal{X}$, exactly one message is sent by each player. Set $K = Mnm/\gamma$, where γ is as in the rounding procedure. By Proposition 4.3.1, we can condition on the fact that $|Z_i| \leq cK^3$ for all $i \in [n]$ and for some constant $c > 0$, which occurs with probability at least $1 - 1/K^2$. Now by the proof of Lemma 4.3.8, we have

that

$$\mathbf{E}\left[e_i^2\right] \leq (j+1)\gamma_0^2 \sum_{v \in T_i} |\langle Q_v, Z\rangle|^2 \leq K^5$$

where $e_i = \sum_{u \in T_i} X_u - r_i$, where r_i is the message sent by the i-th player for a single coordinate of SX_i. By Markov's with probability $1 - 1/K^2$ we have $|e_i| < K^4$, and thus $|r_i| \leq K^6$ for all i.

Now for any r_i for player i in layer ℓ with $|r_i| < 1/(mK)^{d+3-\ell}$, we can simply send 0 instead of r_i. Taken over all the potential children of a player j in layer $\ell + 1$, this introduces a total additive error of $1/K^{d+3-\ell}$ in x_j (where x_j is as in Figure 4.2). Now if x_j originally satisfies $|x_j| > 1/K^{d+2-\ell}$, then the probability that this additive error of $1/K^{d+3-\ell}$ changes the rounding result $r_j = \Gamma(x_j)$ is $O(1/K)$, and we can union bound over all m vertices that this never occurs. Thus, the resulting r_j is unchanged even though x_j incurs additive error. Otherwise, if $|x_j| \leq 1/K^{d+2-\ell}$, then since Player j is in layer $(\ell + 1)$, we round their sketch x_j down to 0 anyway. The final result is an additive error of at most $1/K^2$ to $r_{\mathcal{C}}$. Note that we can tolerate this error, as it is only greater than $\gamma\|\mathcal{X}\|_p$ when $\mathcal{X} = 0$, which is a case that can be detected with $O(1)$ bits of communication per player (just forwarding whether their input is equal to 0). With these changes, it follows that $1/(mK)^{d+3} \leq r_j \leq K^6$ for all players j. Thus each message r_j can be sent in $O(\log(\log((mK)^{d+3}))) = O(\log\log(n) + \log(d) + \log(1/\epsilon))$ as needed.

□

4.3.2 Heavy Hitters and Point Estimation

In this section, we show how our randomized rounding protocol can be used to solve the L_2 heavy hitters problem. For a vector $\mathcal{X} \in \mathbb{R}^n$, let $\mathcal{X}_{\text{tail}(k)}$ be $\mathcal{X}$ with the k largest (in absolute value) entries set equal to 0. Formally, given a vector $\mathcal{X} \in \mathbb{R}^n$, the heavy hitters problem is to output a set of coordinates $H \subset [n]$ of size at most $|H| = O(\epsilon^{-2})$ that contains all $i \in [n]$ with $|\mathcal{X}_i| \geq \epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$. Our protocols solve the strictly harder problem of *point-estimation*. The point estimation problem is to output a $\tilde{\mathcal{X}} \in \mathbb{R}^n$ such that $\|\tilde{\mathcal{X}} - \mathcal{X}\|_\infty \leq \epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$.

To solve this task, we will use the *count-sketch* matrix $\mathbf{S} \in \mathbb{R}^{t \times n}$ [CCFC02b], introduced in Section 2.3.2. By Theorem 9, given the matrix vector product $\mathbf{S}\mathcal{X} \in \mathbb{R}^t$ with $t = \Theta(\frac{1}{\epsilon^2}\log n)$, one can solve the point-estimation problem. Thus, it will suffice to maintain the product $\mathbf{S}\mathcal{X}$ in the message passing model.

We remark that there is a $O(\frac{1}{\epsilon^2}\log(n))$-space streaming algorithm for the heavy hitters problem in insertion only streams, known as BPTree [BCI+16]. However, this streaming algorithm

does not produce a linear sketch, and is not mergeable. The BPTtree algorithm crucially relies on sequentially learning one bit of the identity of each heavy hitter at a time. However, in the message passing model, the data cannot be sequentially accessed unless a path P was fixed running through all the players in G. Such a path may cross the same edge more than once, thus cannot be one-way, and also will require as many as m rounds (instead of the d given by our algorithm). Thus constructions such as BPTtree cannot be used as is to solve the heavy hitters problem in the message passing model. Moreover, BPTtree does not solve the frequency estimation problem, which our protocol does in fact accomplish.

Theorem 31. *Consider a message passing topology $G = (V, E)$ with diameter d, where the i-th player is given as input $X_i \in \mathbb{Z}^n_{\geq 0}$ and $\mathcal{X} = \sum_{i=1}^m X_i$. Then there is a communication protocol which outputs an estimate $\tilde{\mathcal{X}} \in \mathbb{R}^n$ of $\mathcal{X}$ such that*

$$\|\tilde{\mathcal{X}} - \mathcal{X}\|_\infty \leq \epsilon \|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$$

with probability $1 - 1/n^c$ for any constant $c \geq 1$. The protocol uses $O(\frac{m}{\epsilon^2}\log(n)(\log(\log(n)) + \log(d) + \log(1/\epsilon)))$ total communication, and a max communication of $O(\frac{1}{\epsilon^2}\log(n)(\log(\log(n)) + \log(d) + \log(1/\epsilon)))$.

Proof. Note that, by construction, the non-zero entries of each row of $\mathbf{S}$ are 4-wise independent Rademacher variables. So by Lemma 4.3.8, for each entry $(\mathbf{S}\mathcal{X})_j$ of $\mathbf{S}\mathcal{X}$, we can obtain an estimate $r_{\mathcal{C}}$ at the central vertex $\mathcal{C}$ such that $|r_{\mathcal{C}} - (\mathbf{S}\mathcal{X})_j| \leq \epsilon \|\mathcal{X}_{\text{supp}(j)}\|_2$, where $\text{supp}(j) \subset [n]$ is the support of the j-th row of $\mathbf{S}$.

We will now utilize the count-sketch table $\mathbf{A}$ notation as introduced in Section 2.3.2, where we reshape the matrix-vector product $\mathbf{S}\mathcal{X} \in \mathbb{R}^t$ into a table $\mathbf{A} \in \mathbb{R}^{d \times 6/\epsilon^2}$ with d rows and $6/\epsilon^2$ columns, where $t = d \cdot (6/\epsilon^2)$. We first go through the standard analysis of count-sketch which results in the statement of Theorem 9. Consider the estimate of a fixed entry $k \in [n]$. Consider a given entry $g_i(k)\mathbf{A}_{i,h_i(k)}$ for a given row i of $\mathbf{A}$. With probability $5/6$, none of the top $1/\epsilon^2$ largest items in $\mathcal{X}$ (in absolute value) collide with $k \in [n]$ in this entry. Call this event $\mathcal{E}$, and condition on it now. Now note that $\mathbf{E}\left[g_i(k)\mathbf{A}_{i,h_i(k)}\right] = \mathcal{X}_k$, and

$$\begin{aligned}\mathbf{E}\left[(g_i(k)\mathbf{A}_{i,h_i(k)})^2\right] &= \mathbf{E}\left[\sum_{\ell:h_i(\ell)=h_i(k)} \mathcal{X}_\ell^2\right] + \mathbf{E}\left[\sum_{u\neq v:h_i(u)=h_i(v)=h_i(k)} \mathcal{X}_u\mathcal{X}_v g_i(u)g_i(v)\right] \\ &= \mathbf{E}\left[\sum_{\ell:h_i(\ell)=h_i(k)} \mathcal{X}_\ell^2\right]\end{aligned} \tag{4.6}$$

which is at most $\epsilon^2\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2^2/6$ after conditioning on $\mathcal{E}$. Thus with probability $5/6$, we have $|g_i(k)\mathbf{A}_{i,h_i(k)} - \mathcal{X}_k| \leq \epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$. Note that we conditioned on $\mathcal{E}$, so altogether we have that $|g_i(k)\mathbf{A}_{i,h_i(k)} - \mathcal{X}_k| \leq \epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$ with probability at least $2/3$. Thus with probability at least $1 - 1/n^2$, the above bound holds for the median of the estimates $\text{median}_{i\in[d]}\{g_i(k)\mathbf{A}_{i,h_i(k)}\}$, and we can then union bound over all $k \in [n]$ to obtain the desired result.

Now consider the changes that occur to this argument when we add an additional $\epsilon\|\mathcal{X}_{\text{supp}(j)}\|_2$ error to each $\mathbf{A}_{i,h_i(k)}$. Let $r_{\mathcal{C}}^{i,h_i(k)}$ be the estimate held by the central vertex of $\mathbf{A}_{i,h_i(k)}$. As noted above, after conditioning on $\mathcal{E}$, we have $\mathbf{E}\left[\|\mathcal{X}_{\text{supp}(j)}\|_2^2\right] \leq \epsilon^2\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2^2$, so with probability $15/16$ we have $\|\mathcal{X}_{\text{supp}(j)}\|_2 \leq 4\epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$. Taken together with the event that $|g_i(k)\mathbf{A}_{i,h_i(k)} - \mathcal{X}_k| \leq \epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$, which occurred with probability $5/6$ after conditioning on $\mathcal{E}$, it follows that $|g_i(k)r_{\mathcal{C}}^{i,h_i(k)} - \mathcal{X}_k| \leq \epsilon\|\mathcal{X}_{\text{tail}(1/\epsilon^2)}\|_2$ with probability $1 - (1/6 + 1/6 + 1/16) > 3/5$. The result now follows now by the same argument as above (note that it only required this probability to be at least $1/2 + \Omega(1)$).

The message complexity analysis is identical to that of Theorem 30, except instead of applying Proposition 4.3.1 to bound the probability that $|Z_i| \leq c(Mnm/\gamma)^3$ for all $i \in [n]$ (Where Z is the non-zero entries in some row of $\mathbf{S}$), we can use the deterministic bound that $|Z_i| \leq 1$. By the same argument as in Theorem 30, each rounded message r_j requires $O(\log\log(n) + \log(d) + \log(1/\epsilon))$ bits to send with high probability. Since each player sends $6d/\epsilon^2 = O(\log(n)/\epsilon^2)$ messages (one for each row of $\mathbf{S}$), the claimed communication follows.

□

4.4 F_p Estimation for $p < 1$

In this section, we develop algorithms for F_p estimation for $p < 1$ in the message passing model, and in the process obtain improved algorithms for entropy estimation. Our algorithms require a max communication of $O(\frac{1}{\epsilon^2}(\log\log n + \log 1/\epsilon))$, which is independent of the diameter of the graph topology. In particular, these results hold for the directed line graph, where communication is only allowed in one direction. As a consequence, we improve upon the best known algorithms for the space complexity of insertion only streaming algorithms.

We begin by reviewing the fundamental sketching procedure used in our estimation protocol. The sketch is known as a Morris counter. We point out that, in fact, our rounding algorithm from Section 4.3.1 reduces to being a Morris counter when run on insertion only streams.

4.4.1 Morris Counters

We begin by describing the well known approximate counting algorithm, known as a Morris Counter [Mor78, Fla85]. The algorithm first picks a base $1 < b \leq 2$, and initalizes a counter $C \leftarrow 0$. Then, every time it sees an insertion, it increments the counter $C \leftarrow C + \delta$, where $\delta = 1$ with probability b^{-C}, and $\delta = 0$ otherwise (in which case the counter remains unchanged). After n insertions, the value n can be estimated by $\tilde{n} = (b^C - b)/(b-1) + 1$.

Definition 4.4.1. *The approximate counting problem is defined as follows. Each player i is given a positive integer value $X_i \in \mathbb{Z}_{\geq 0}$, and the goal is for some player at the end to hold an estimate of $\mathcal{X} = \sum_i X_i$.*

Proposition 4.4.2 (Proposition 5 [Fla85])**.** *If C_n is the value of the Morris counter after n updates, then $\mathbf{E}[\tilde{n}] = n$, and Var$(\tilde{n}) = (b-1)n(n+1)/2$.*

Corollary 4.4.3. *If C_n is the value of a Morris counter run on a stream of n insertions with base $b = (1 + (\epsilon\delta)^2)$, then with probability at least $1 - \delta$, we have $\tilde{n} = (1 \pm \epsilon)n$ with probability at least $1 - \delta$. Moreover, with probability at least $1 - \delta$, the counter C_n requires $O(\log\log(n) + \log(1/\epsilon) + \log(1/\delta))$-bits to store.*

Proof. The first claim follows immediately from Proposition 4.4.2 and Chebyshev's inequality. Conditioned on $\tilde{n} = (1 \pm \epsilon)n$, which occurs with probability at least $1 - \delta$, we have that $\tilde{n} = (b^C - b)/(b-1) < 2n$, so $C = O(\frac{1}{1-b}\log(n)) = O(\frac{\log(n)}{(\epsilon\delta)^2})$, which can be stored in the stated space. □

Next, we demonstrate that Morris counters can be *merged*. That is, given one Morris counter X run on a stream of length n_1 (of only insertions or deletions), and a Morris counter Y run on a stream of length n_2, we can produce a Morris counter Z which is distributed identically to a Morris counter run on a stream of $n_1 + n_2$ updates. The procedure for doing so is given below.

A sketch of the proof of the correctness of the above merging procedure is given by Cohen [Coh13]. For completeness, we provide a proof here as well.

Lemma 4.4.4. *Given Morris counters X, Y run on streams of length n_1, n_2 respectively, The above merging procedure produces a Morris counter Z which is distributed identically to a Morris counter that was run on a stream of $n_1 + n_2$ insertions.*

Proof. Let $u_1, \ldots, u_{n_1}$ be the updates that X is run on, and let $v_1, \ldots, v_{n_2}$ be the updates that

Merging Morris Counters

`Input` : Morris counters X, Y with base b.

- Set $Z \leftarrow X$.
- for $i = 1, 2, \ldots, Y$, set:

$$Z \leftarrow \begin{cases} Z+1 & \text{with probability } b^{-Z+i-1} \\ Z & \text{with probability} \end{cases}$$

`Output` : Morris counter Z

Y is run on. Consider the distribution of a random variable Z' run on $u_1, \ldots, u_{n_1}, v_1, \ldots, v_{n_2}$. We show that Z', Z have the same distribution. We associate with each u_i, v_i a uniform random variable $p_i, q_i \in [0, 1]$ respectively. First, fix p_i exactly for all $i \in [n_1]$, and now condition on the set of v_i which increment Y during the execution. We now consider the condition distribution of the q_i's given the updates v_i which incremented Y. We show that the conditional distribution of executions of Z' is the same as the merging procedure.

To see this, conditioned on some counter value y, items v_i that incremented y has $q_i < b^{-y}$, otherwise they had $q_i \geq b^{-y}$. We will prove via induction that the items that: 1) the items v_i that did not increment Y do not increment Z', 2) the item v_j that had the i-th increment of Y, conditioned on the value of Z' so far, increments Z' with probability b^{-Z+i-1}, and 3) at any point during the processing of the v_i's we have $Z' \geq Y$.

Note that at the beginning, we have $Z' \geq 0$ and $Y = 0$, so $Z' \geq Y$. Moreover, v_1 increments Y with probability 1, so conditioning on this does not affect the distribution of q_1. So the conditional probability that Z' is incremented is just $b^{-Z'}$ as needed. If $Z' = 0$ initially, then $b^{-Z'} = 1$, so we maintain $Z' \geq 1 = Y$ at this point.

Now consider any update v_j that did not cause an increment to Y. We must have had $q_j \in [b^{-Y}, 1]$, and since we maintained $Z' \geq Y$ by induction, it follows that v_j did not cause an update to Y, and neither Z' or Y was affected by the update v_j, so we maintain $Z' \geq Y$. Now consider the item v_j that caused the i-th increment to Y. This conditioning implies $q_j \in [0, b^{-(i-1)}]$. Conditioned on the value of Z' so far, the probability v_j increments Z' is the probability that $q_j \in [0, b^{-Z'}]$ conditioned on $q_j \in [0, b^{-(i-1)}]$, which is $b^{-Z'+i-1}$ as desired. Moreover, if we had that $Z' = Y$ at this time step, then $Z' = Y = (i-1)$, so $b^{-Z'+i-1} = 1$ and Z' is incremented with probability 1 conditioned on the fact that Y was incremented. So we maintain the property that $Z' \geq Y$ at all steps, which completes the induction proof. $\square$

Corollary 4.4.5. *There is a protocol for F_1 estimation of non-negative vectors, equivalently for the approximate counting problem, in the message passing model which succeeds with probability $1-\delta$ and uses a max-communication of $O((\log\log(n)+\log(1/\epsilon)+\log(1/\delta))$-bits.*

Proof. First note that the F_1 of $\mathcal{X}$ is simply $\sum_i \|X_i\|_1$, so each player can run a Morris counter on $\|X_i\|_1$ and send them up a tree to a center vertex $\mathcal{C}$ as in Section 4.3.1. Since Morris counters can be merged without affecting the resulting distribution, the center player $\mathcal{C}$ at the end holds a Morris counter Z which is distributed identically to a Morris counter with base b run on $\|\mathcal{X}\|_1 = \sum_i \|X_i\|_1$ updates. Then by Corollary 4.4.3, we have that $\tilde{n} = (b^Z - b)/(b-1)+1 = (1\pm\epsilon)\|\mathcal{X}\|_1$ with probability $1-\delta$, and moreover that Z is at most $O(\log\log(n)+\log(1/\epsilon)+\log(1/\delta)$ bits long. Since Z is at least as large as the size of any other counter sent across an edge, it follows that the max communication in the protocol was at most $O(\log\log(n)+\log(1/\epsilon)+\log(1/\delta)$ bits, and the proof of the total communication concludes from the fact that each player communicates exactly once. □

We now note that Morris counters can easily used as approximate counters for streams with both insertions and deletions (positive and negative updates), by just storing a separate Morris counter for the insertions and deletions, and subtracting the estimate given by one from the other at the end. The error is then proportional to the total number of updates made.

Corollary 4.4.6. *Using two Morris counters separately for insertions and deletions, on a stream of I insertions and D deletions, there is an algorithm, called a signed Morris counter, which produces $\tilde{n}$ with $|\tilde{n}-n| \leq \epsilon(I+D)$, where $n = I - D$, with probability $1-\delta$, using space $O(\log\log(I+D)+\log(1/\epsilon)+\log(1/\delta))$.*

Hereafter, when we refer to a Morris counter that is run on a stream which contains both positive and negative updates as a *signed* Morris counter. Therefore, the guarantee of Corollary 4.4.6 apply to such signed Morris counters, and moreover such signed Morris counters can be Merged as in Lemma 4.4.4 with the same guarantee, but just Merged separately the counters for insertions and deletions.

4.4.2 The F_p Estimation Protocol, $p < 1$

We now provide our algorithm for F_p estimation in the message passing model with $p \leq 1$. Our protocol is similar to our algorithm for $p \geq 1$. We fix a vertex $\mathcal{C}$ which is a center of the communication topology: i.e., the distance of the farthest vertex from $\mathcal{C}$ in G is minimal

among all such vertices. We then consider the shortest path tree T rooted at $\mathcal{C}$, which has depth at most d, where d is the diameter of G. The players then choose random p-stable vectors $\mathbf{S}_{i,*} \in \mathbb{R}^n$ for $i \in [k]$, where $k = \Theta(1/\epsilon^2)$ and the j-th player computes $\langle \mathbf{S}_{i,*}, X_j \rangle$, and adds this value to a Morris counter. Each player receives Morris counters from their children in T, and thereafter merges these Morris counters with its own. Finally, it sends this merged Morris counter, containing updates from all players in the subtree rooted at j, to the parent of j in T. At the end, the center $\mathcal{C}$ holds a Morris counter C_i which approximates $\sum_j \langle \mathbf{S}_{i,*}, X_j \rangle$. The main algorithm for each player j is given formally in Figure 4.3.

We first prove a technical lemma which bounds the total number of updates made to the Morris counters. Since the error of the signed Morris counters is proportional to the number of updates made, and not to the actual value that is being approximated, it is important that the number of updates does not exceed the desired quantity of estimation by too much.

Lemma 4.4.7. *Let $Z_1, \ldots, Z_n$ each be distributed via some distribution $\mathcal{D}$, but not necessarily independently, such that $\mathcal{D}$ has the property that if $Z \sim \mathcal{D}$ then there is a universal constant $c > 0$ such that for every $\lambda > 1$ we have $\mathbf{Pr}[|Z| > c\lambda] < \lambda^{-p}$ for some $0 < p \leq 1$. Note that this is true of the p-stable distribution $\mathcal{D}_p$. Fix any positive vector $\mathcal{X} \in \mathbb{R}^n$, and let $Y_i = Z_i \mathcal{X}_i$ for $i \in [n]$. Then for any $\lambda \geq 1$, if $1 - p > \theta > 0$ for some constant θ, then*

$$\mathbf{Pr}\left[\|Y\|_1 \geq C\lambda^{1/p}\|\mathcal{X}\|_p\right] \leq \frac{1}{\lambda}$$

otherwise

$$\mathbf{Pr}\left[\|Y\|_1 \geq C\lambda^{1/p}\log(n\lambda)\|\mathcal{X}\|_p\right] \leq \frac{1}{\lambda}$$

Where $C > 0$ is a fixed constant independent of λ, n, p.

Proof. We proceed similarly as in Lemma 4.3.3. For $k \in \mathbb{Z}$, let $I_k = \{i \in [n] \;\middle|\; 2^{-k}\|\mathcal{X}\|_p \leq Y_i \leq 2^{-k+1}\|\mathcal{X}\|_p\}$. Note that for any $i \in [n]$

$$\mathbf{Pr}\left[i \in \bigcup_{k' \geq k} I_{k'}\right] \leq \mathbf{Pr}\left[Z_i \geq 2^{-k}\|\mathcal{X}\|_p/\mathcal{X}_i\right]$$

$$\leq c2^{kp}\frac{\mathcal{X}_i^p}{\|\mathcal{X}\|_p^p}$$

For some constant $c > 0$, where in the last inequality we used Proposition 4.3.1 for $\mathcal{D}_p$ (or the fact that $\mathcal{D}$ has tails of order λ^{-p} by assumption). Thus $\mathbf{E}\left[\sum_{k' \geq k} |I_{k'}|\right] \leq c2^{pk}$. Let $\mathcal{E}_0$ be the

Procedure for player j

$k \leftarrow \Theta(1/\epsilon^2)$
$\epsilon' \leftarrow \Theta(\epsilon \frac{\delta^{1/p}}{\log(n/\delta)})$
$\delta \leftarrow 1/(200k)$

1. Using shared randomness, choose sketching matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$ of i.i.d. p-stable random variables. Generate $\mathbf{S}$ up to precision $\eta = \text{poly}(1/(n, m, M))$, so that $\eta^{-1}\mathbf{S}$ has integral entries.

2. For each $i \in [k]$, receive signed Morris counters $y_{j_1,i}, y_{j_2,i}, \ldots, y_{j_t,i}$ from the $t \in \{0, \ldots, m\}$ children of node j in the prior layer.

3. Compute $\eta^{-1}\langle \mathbf{S}_{i,*}, X_j \rangle \in \mathbb{Z}$, where $\mathbf{S}_{i,*}$ is the i-th row of $\mathbf{S}$, and run a new signed Morris counter C on $\eta^{-1}\langle \mathbf{S}_{i,*}, X_j \rangle$ with parameters (ϵ', δ').

4. Merge the signed Morris counters $y_{j_1,i}, y_{j_2,i}, \ldots, y_{j_t,i}, C$ into a counter $y_{j,i}$.

5. Send the merged signed Morris counter $y_{j,i}$ to the parent of player j. If player j is the root node $\mathcal{C}$, then set C_i to be the estimate of the signed Morris counter $y_{j,i}$, and return the estimate
$$\eta \cdot \text{median} \left\{ \frac{|C_1|}{\theta_p}, \ldots, \frac{|C_k|}{\theta_p} \right\}$$
Where θ_p is the median of the distribution $\mathcal{D}_p$.

Figure 4.3: Multi-party F_p estimation protocol, $p < 1$

event that $I_k = \emptyset$ for all $k \leq -\frac{1}{p}\log(10c\lambda)$ for some sufficiently large constant c. By Markov's inequality, we have $\mathbf{Pr}[\mathcal{E}_0] \geq 1 - \frac{1}{10\lambda}$. Now for any $k \geq \frac{1}{p}\log(n^2/\lambda)$, the contribution of any coordinate of Y_i to $\|Y\|_1$ with $i \in I_k$ is at most $\frac{\lambda}{n^2}\|\mathcal{X}\|_p$, thus the total contribution of all such items is at most $\frac{\lambda}{n}\|\mathcal{X}\|_p$, which we can safely disregard. Finally, the expected contribution of the coordinates Y_i for $i \in I_k$ with $-\lceil \frac{1}{p}\log(10c\lambda)\rceil \leq k \leq \lceil \frac{1}{p}\log(n^2/\lambda)\rceil$ is at most

$$\mathbf{E}\left[\sum_{-\lceil \frac{1}{p}\log(10c\lambda)\rceil \leq k \leq \lceil \frac{1}{p}\log(n^2/\lambda)\rceil} 2^{-k+1}\|\mathcal{X}\|_p |I_k| \right] \leq c \sum_{-\lceil \frac{1}{p}\log(10c\lambda)\rceil \leq k \leq \lceil \frac{1}{p}\log(n^2/\lambda)\rceil} 2^{-k(1-p)+1}\|\mathcal{X}\|_p$$

If p is constant bounded from 1, this sum is geometric and bounded above by

$$O\left(2^{\log(10c\lambda)(1/p-1)+1}\|\mathcal{X}\|_p\right) = O\left((10c\lambda)^{1/p-1}\|\mathcal{X}\|_p\right)$$

otherwise we can upper bound each term by this quantity, giving a total bound of

$$O\left((10c\lambda)^{1/p-1}\frac{\log(n\lambda)}{p}\|\mathcal{X}\|_p\right) = O\left((10c\lambda)^{1/p-1}\log(n\lambda)\|\mathcal{X}\|_p\right)$$

where the equality holds because $1/p = \Theta(1)$ if p is close to 1. In the first case, by Markov's inequality, we have

$$\mathbf{Pr}\left[\sum_{-\lceil\frac{1}{p}\log(10c\lambda)\rceil\le k\le\lceil\frac{1}{p}\log(n^2/\lambda)\rceil} 2^{-k+1}\|\mathcal{X}\|_p|I_k| \ge 2(10c\lambda)^{1/p}\|\mathcal{X}\|_p\right] \le \frac{1}{2\lambda}$$

Union bounding over the above event and $\mathcal{E}_0$, we obtain the desired result with probability at least $1-(\frac{1}{2\lambda}+\frac{1}{10\lambda}) > 1-\frac{1}{\lambda}$ in the case that p is constant bounded from 1. In the event that it is not, Markov's inequality gives:

$$\mathbf{Pr}\left[\sum_{-\lceil\frac{1}{p}\log(10c\lambda)\rceil\le k\le\lceil\frac{1}{p}\log(n^2/\lambda)\rceil} 2^{-k+1}\|\mathcal{X}\|_p|I_k| \ge 2(10c\lambda)^{1/p}\log(n\lambda)\|\mathcal{X}\|_p\right] \le \frac{1}{2\lambda}$$

as needed. Applying the same union bound gives the desired result for p arbitrarily close or equal to 1.

□

Theorem 32. *For $p \in (0,1)$, there is a protocol for F_p estimation in the non-negative data message passing model which succeeds with probability $2/3$ and uses a total communication of $O(\frac{m}{\epsilon^2}(\log\log(n)+\log(1/\epsilon))$-bits, and a max-communication of $O(\frac{1}{\epsilon^2}(\log\log(n)+\log(1/\epsilon))$-bits. The protocol requires a total of at most d rounds, where d is the diameter of the communication topology G.*

Proof. By Lemma 4.4.4, the Merging procedure has the effect that at the end, the player $\mathcal{C}$ has a signed Morris counter $I, D \in \mathbb{Z}_{\ge 1}$, such that I is distributed as a Morris counter run on the updates $\eta^{-1}\sum_{j:\langle\mathbf{S}_{i,*},X_j\rangle\ge 0}\langle\mathbf{S}_{i,*},X_j\rangle$, and D is distributed as a Morris counter run on the updates $\eta^{-1}\sum_{j:\langle\mathbf{S}_{i,*},X_j\rangle<0}\langle\mathbf{S}_{i,*},X_j\rangle$. Then By Corollary 4.4.6, the estimate of this signed Morris counter is a value $\tilde{I}-\tilde{D}$ such that

$$\begin{aligned}|(\tilde{I}-\tilde{D}) - \eta^{-1}\sum_j\langle\mathbf{S}_{i,*},X_j\rangle| &\le \epsilon'\eta^{-1}\sum_j|\langle\mathbf{S}_{i,*},X_j\rangle| \\ &\le \epsilon'\eta^{-1}\sum_{j=1}^{n}|\mathbf{S}_{i,j}\mathcal{X}_j|\end{aligned} \tag{4.7}$$

holds with probability $1-\delta$. Call this event E_i^1. Now by Lemma 4.4.7, with probability at least $1-\delta$, we have that

$$\sum_{j=1}^{n} |\mathbf{S}_{i,j}\mathcal{X}_j| \leq C\delta^{-1/p}\log(n\delta)\|\mathcal{X}\|_p$$

for some constant C. Note that while Lemma 4.4.7 held for random variables $\mathbf{S}_{i,j}$ that were not generated to finite precision, note that generating up to precision η only changes each term in the sum by at most ηM, and since $\|\mathcal{X}\|_p \geq 1$ (because $\mathcal{X} \neq 0$, and if it was it could be tested with $O(1)$ bits of communication per player), this additive error can be absorbed into the constant C. So call the event that this inequality holds E_i^2, and let E_i be the event that both E_i^1 and E_i^2 hold. Note $\mathbf{Pr}[E_i] > 1-2\delta$. We now condition on $\cap_{i=1}^k E_i$, which occurs with probability $1-2k\delta > 99/100$.

Now, conditioned on $\cap_{i=1}^k E_i$, it follows that for each $i \in [k]$, the center $\mathcal{C}$ has an estimate C_i of $\eta^{-1}\langle \mathbf{S}_{i,*}, \mathcal{X}\rangle$ such that

$$\begin{aligned} |\eta C_i - \langle \mathbf{S}_{i,*}, \mathcal{X}\rangle| &\leq \epsilon' \sum_{j=1}^{n} |\mathbf{S}_{i,j}\mathcal{X}_j| \\ &\leq C\epsilon'\delta^{-1/p}\log(n/\delta)\|\mathcal{X}\|_p \\ &\leq \epsilon\|\mathcal{X}\|_p/4 \end{aligned} \tag{4.8}$$

Now by [Ind06], setting $k = \Theta(1/\epsilon^2)$, if we define

$$\tilde{F}_p = \text{median}_{i\in[k]}\left\{\frac{|\langle \mathbf{S}_i, \mathcal{X}\rangle|}{\theta_p}\right\}$$

then the bound

$$|\tilde{F}_p - \|\mathcal{X}\|_p| < (\epsilon/2)\|\mathcal{X}\|_p$$

holds with probability at least $4/5$. Since $|\eta C_i - \langle \mathbf{S}_{i,*}, \mathcal{X}\rangle| < (\epsilon/4)\|\mathcal{X}\|_p$ for all $i \in [k]$, it follows that

$$\left|\eta \cdot \text{median}_{i\in[k]}\left\{\frac{|C_i|}{\theta_p}\right\} - \|\mathcal{X}\|_p\right| \leq \epsilon\|\mathcal{X}\|_p$$

with probability at least $1-(1/100+1/5) > 3/4$, which completes the proof of correctness.

For message complexity, note that conditioned on $\cap_{i=1}^k E_i$, every Morris counter in question is a $(1+\epsilon)$ relative error approximation of the quantity (insertions or deletions) that it is estimating. Thus by Corollary 4.4.6, noting that at most $\text{poly}(n, m, M)$ updates are every made to any Morris counter, the space to store any Morris counter is $O(\log\log(n) + \log(1/\epsilon))$, which is an upper bound on the size of any message in the protocol. Note that we can safely fail if any message

becomes larger than this threshold, since this would mean that $\cap_{i=1}^{k} E_i$ failed to hold. The total message complexity follows from the fact that each player sends at most one message during the entire protocol, and the round complexity follows from the depth of the shortest path tree rooted at the center $\mathcal{C}$.

□

4.4.3 The Streaming Algorithm for F_p Estimation, $p < 1$

As discussed earlier, the insertion-only streaming model of computation is a special case of the above communication setting, where the graph in question is the line graph, and each player receives vector $X_i \in \mathbb{R}^n$ which is the standard basis vector $e_j \in \mathbb{R}^n$ for some $j \in [n]$. This immediately implies an improved streaming algorithm for insertion only F_p estimation in the random oracle model. The only step remaining to fully generalize the result to the non-random oracle streaming setting is an adequate derandomization of the randomness required to generate the matrix $\mathbf{S}$.

Our derandomization will follow from the results of [KNW10a], which demonstrate that, using a slightly different estimator known as the log-cosine estimator (discussed below), the entries of each row $\mathbf{S}_{i,*}$ can be generated with only $\Theta(\log(1/\epsilon)/\log\log(1/\epsilon))$-wise independence, and the seeds used to generate separate rows of $\mathbf{S}_{i,*}$ need only be pairwise independent. Thus, storing the randomness used to generate $\mathbf{S}$ requires only $O(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\log(n))$-bits of space.

We now discuss the estimator of [KNW10a] precisely. The algorithm generates a matrices $\mathbf{S} \in \mathbb{R}^{k\times n}$ and $\mathbf{S}' \in \mathbb{R}^{k'\times n}$ with $k = \Theta(1/\epsilon^2)$ and $k' = \Theta(1)$, where each entry of $\mathbf{S}, \mathbf{S}'$ is drawn from the p-stable distribution $\mathcal{D}_p$. For a given row i of $\mathbf{S}$, the entries $\mathbf{S}_{i,j}$ are $\Theta(\log(1/\epsilon)/\log\log(1/\epsilon))$-wise independent, and for $i \neq i'$, the seeds used to generate $\{\mathbf{S}_{i,j}\}_{j=1}^n$ and $\{S_{i',j}\}_{j=1}^n$ are pairwise independent. $\mathbf{S}'$ is generated with only $\Theta(1)$-wise independence between the entries in a given row in $\mathbf{S}'$, and pairwise independence between rows. The algorithm then maintains the vectors $y = \mathbf{S}\mathcal{X}$ and $y' = \mathbf{S}'\mathcal{X}$ throughout the stream, where $\mathcal{X} \in \mathbb{Z}^n_{\geq 0}$ is the stream vector. Define $y'_{med} = \text{median}\{|y'_i|\}_{i=1}^{k'}/\theta_p$, where θ_p is the median of the distribution $\mathcal{D}_p$ ([KNW10a] discusses how this can be approximated to $(1 \pm \epsilon)$ efficiently). The log-cosine estimator R of $\|\mathcal{X}\|_p$ is then given by

$$R = y'_{med} \cdot \left(-\ln\left(\frac{1}{k}\sum_{i=1}^{k}\cos\left(\frac{y_i}{y'_{med}}\right)\right)\right)$$

Theorem 33. *There is a streaming algorithm for insertion only F_p estimation for any $p \in (0,1)$, which outputs a value $\tilde{R}$ such that with probability at least $2/3$, we have that*

$$|\tilde{R} - \|x\|_p| \leq \epsilon \|x\|_p$$

where $x \in \mathbb{R}^n$ is the state of the stream vector at the end of the stream. The algorithm uses $O((\frac{1}{\epsilon^2}(\log\log(n) + \log(1/\epsilon)) + \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\log(n))$-bits of space.

Proof. In [KNW10a], they first condition on the fact that y'_{med} is a constant factor approximation of $\|x\|_p$. Namely, they condition on the event that $|y'_{med} - \|x\|_p| \leq \|x\|_p/10$, which occurs with probability $7/8$ for sufficiently large $k' = \Theta(1)$ (this is just the standard p-stable median estimator of Indyk [Ind06]). Note that y' can be computed exactly by our algorithm using only $O(\log(n))$ bits of space. Using our protocol from Theorem 32, applied to the line graph on m updates (corresponding to a streaming algorithm), it follows that we can approximate y by a vector $\tilde{y} \in \mathbb{R}^k$ such that $|\tilde{y} - y|_\infty < \epsilon' \|x\|_p$ with probability $99/100$ (here we have boosted the probability by a constant by running $\Theta(1)$ copies in parallel and taking the median for each coordinate), for some sufficiently small $\epsilon' = \Theta(\epsilon)$, and such that computing and storing $\tilde{y}$ requires only $O(k(\log\log(n) + \log(1/\epsilon)))$-bits of space. Since cosine has bounded derivatives everywhere, we have

$$\left|\frac{\tilde{y}_i}{y'_{med}} - \frac{y_i}{y'_{med}}\right| \leq \frac{\epsilon' \|x\|_p}{y'_{med}} = O(\epsilon')$$

so

$$\left|\cos\left(\frac{\tilde{y}_i}{y'_{med}}\right) - \cos\left(\frac{y_i}{y'_{med}}\right)\right| < O(\epsilon')$$

for all $i \in [k]$. This gives

$$\left|\frac{1}{k}\sum_{i=1}^{k}\cos\left(\frac{\tilde{y}_i}{y'_{med}}\right) - \frac{1}{k}\sum_{i=1}^{k}\cos\left(\frac{y_i}{y'_{med}}\right)\right| = O(\epsilon')$$

Furthermore, conditioned on the success of the estimator of [KNW10a] (which occurs with probability $3/4$, and includes the conditioning on the event that $y'_{med} = \Theta(\|x\|_p)$), we have $|\frac{1}{k}\sum_{i=1}^{k}\cos\left(\frac{y_i}{y'_{med}}\right)| = \Theta(1)$ (see Lemma 2.6 in [KNW10a]), and since $\ln(\cdot)$ has bounded derivatives for values $\Theta(1)$, it follows that if

$$\tilde{R} = y'_{med} \cdot \left(-\ln\left(\frac{1}{k}\sum_{i=1}^{k}\cos\left(\frac{\tilde{y}_i}{y'_{med}}\right)\right)\right)$$

then $|\tilde{R} - R| < O(\epsilon' y'_{med}) = O(\epsilon' \|x\|_p)$. Taking $k = \Theta(1/\epsilon^2)$ with a small enough constant, we have $|R - \|x\|_p| < \epsilon \|x\|_p/2$, and setting $\epsilon' = \Theta(\epsilon)$ small enough, we obtain $|\tilde{R} - \|x\|_p| < \epsilon \|x\|_p$ as needed.

□

4.5 Entropy Estimation

In this section, we show how our results imply improved algorithms for entropy estimation in the message-passing model. Recall that, for a vector $\mathcal{X} \in \mathbb{R}^n$, the Shannon entropy is given by $H = \sum_{i=1}^n \frac{|\mathcal{X}_i|}{\|\mathcal{X}\|_1} \log(\frac{\|\mathcal{X}\|_1}{|\mathcal{X}_i|})$.

While we primarily have focused on the standard p-stable distribution $\mathcal{D}_p$ from Definition 2.2.6 in this thesis, which is the distribution with characteristic function $\mathbf{E}\left[e^{itZ}\right] = e^{-|t|^p}$, in this section we will utilize a different family of p-stable distributions. Specifically, we follow the approach taken by [CC13, LZ11, HNO08a, HNO08b] for entropy estimation in data streams, which is to use *maximally-skewed* stable distributions. Formally, a maximally skewed stable distribution is a stable distribution with skewness parameter $\beta \in \{1, -1\}$ in the more general parameterization of stable distributions from Definition 2.2.5. For convince, we restate this more general definition of stable distributions below.

Definition 2.2.5 (Stable distribution, general). *A random variable X is said to be p-stable if it is drawn from the distribution $S(p, \beta, \gamma, \delta)$ for some $p \in (0, 2]$, $\beta \in [-1, 1]$, $\gamma > 0$, and $\delta \in \mathbb{R}$, where $S(p, \beta, \gamma, \delta)$ is defined via its characteristic function:*

$$\mathop{\mathbf{E}}_{Z \sim S(p,\beta,\gamma,\delta)} \left[e^{-itZ}\right] = \begin{cases} \exp\left(-\gamma^p |t|^p \left[1 - i\beta \tan(\frac{\pi p}{2}) \textit{sign}(t)\right] + i\delta t\right) & \textit{if } p \neq 1 \\ \exp\left(-\gamma |t| \left[1 + i\beta \frac{2}{\pi} \textit{sign}(t) \log(|t|)\right] + i\delta t\right) & \textit{if } p = 1 \end{cases}$$

where $\textit{sign}(t) \in \{1, -1\}$ is the sign of a real $t \in \mathbb{R}$.

In what follows, we will utilize the maximally-skewed 1-stable distribution $S(1, -1, \pi/2, 0)$, along with the algorithm of [CC13], which is given formally in Figure 4.4. The guarantee of the algorithm is given in Theorem 34.

Theorem 34 ([CC13])**.** *The above estimate $\tilde{H}$ satisfies $|\tilde{H} - H| < \epsilon$ with probability at least $9/10$.*

Sketching Algorithm for Entropy Estimation

Input: $\mathcal{X} \in \mathbb{R}^n$

1. Generate $\mathbf{S} \in \mathbb{R}^{k \times n}$ for $k = \Theta(1/\epsilon^2)$ of i.i.d. $S(1, -1, \pi/2, 0)$ random variables to precision $\eta = 1/\text{poly}(M, n)$.

2. Compute $\mathbf{S}\mathcal{X} \in \mathbb{R}^k$.

3. Set $y_i \leftarrow (\mathbf{S}\mathcal{X})_i / \|\mathcal{X}\|_1$ for $i \in [k]$

4. Return
$$\tilde{H} = -\log\left(\frac{1}{k}\sum_{i=1}^{k} e^{y_i}\right)$$

Figure 4.4: Entropy Estimation algorithm of [CC13]

Lemma 4.5.1. *Fix $0 < \epsilon_0 < \epsilon$. Let $\mathbf{S} \in \mathbb{R}^{k \times n}$ with $k = \Theta(1/\epsilon^2)$ be a matrix of i.i.d. $S(1, -1, \pi/2, 0)$ random variables to precision $\eta = 1/\text{poly}(M, n)$. Then there is a protocol in the message passing model that outputs $Y \in \mathbb{R}^k$ at a centralized vertex with $\|Y - \mathbf{S}\mathcal{X}\|_\infty \leq \epsilon_0 \|\mathcal{X}\|_1$ with probability $9/10$. The protocol uses a total communication of $O(\frac{m}{\epsilon^2}(\log\log(n) + \log(1/\epsilon_0))$-bits, and a max-communication of $O(\frac{1}{\epsilon^2}(\log\log(n) + \log(1/\epsilon_0)))$-bits.*

Proof. Note that in the proof of Theorem 32, the only property of the distribution of $\mathbf{S}$ that was needed to obtain a $\epsilon_0\|\mathcal{X}\|_1$ additive error estimate of $\langle \mathbf{S}_{i,*}, \mathcal{X} \rangle = (\mathbf{S}\mathcal{X})_i$ for every $i \in [k]$ was that Lemma 4.4.7 hold for this distribution. Namely, we need the property that if $Z \sim S(1, -1, \pi/2, 0)$, then there is a constant c such that for every $\lambda \geq 1$ we have $\mathbf{Pr}[|Z| \geq c\lambda] < \lambda^{-1}$. This tail bound is indeed true for 1-stable distributions of any skewness (Theorem 1.12 [Nol]). Thus the conditions of Lemma 4.4.7 hold for $S(1, -1, \pi/2, 0)$, and the result follows from the proof of Theorem 32. □

Theorem 35. *There is a multi-party communication protocol in the non-negative data message passing model that outputs a ϵ-additive error of the Shannon entropy H. The protocol uses a max-communication of $O(\frac{1}{\epsilon^2}(\log\log(n) + \log(1/\epsilon))$-bits.*

Proof. By Lemma 4.5.1, the central vertex $\mathcal{C}$ (as in Section 4.3.1) can obtain a vector $Y \in \mathbb{R}^k$ with $\|Y - \mathbf{S}\mathcal{X}\|_\infty \leq \epsilon_0\|\mathcal{X}\|_1$ with probability $9/10$. By Corollary 4.4.5, there is also a multi-party protocol which gives an estimate $R \in \mathbb{R}$ such that $R = (1 \pm \epsilon_0)\|\mathcal{X}\|_1$ with probability $9/10$, where each player communicates once, sending at most $O(\log\log(n) + \log(1/\epsilon)))$ bits, and the estimate is held at $\mathcal{C}$ at the end. Now note that each $(\mathbf{S}\mathcal{X})_i / \|\mathcal{X}\|_1$ is distributed as

$S(1, -1, \pi/2, H)$, where H is the Shannon entropy of $\mathcal{X}$ ([CC13] Lemma 2.2 or [Nol] Proposition 1.17).

By anti-concentration of all stable random variables, we have that if $Z \sim S(p, \beta, \gamma, \delta)$ then $\mathbf{Pr}\left[|Z| < \epsilon_0 \gamma\right] < C\epsilon_0$ for some constant C. Thus

$$\mathbf{Pr}\left[|(\mathbf{S}\mathcal{X})_i| < \epsilon_0 \|\mathcal{X}\|_1\right] \leq C\epsilon_0$$

for some constant C, so we can condition on the event $\mathcal{E}$ that $|(\mathbf{S}\mathcal{X})_i| > \epsilon_0/\epsilon^3 \|\mathcal{X}\|_1$ for all $i \in [k]$, which occurs with probability at least $99/100$ after a union bound and setting $\epsilon_0 = O(\epsilon^6)$ sufficiently small. Given this, we have $Y_i = (1 \pm \epsilon^3)(\mathbf{S}\mathcal{X})_i$ for all $i \in [k]$. Setting $\overline{Y_i} = Y_i/R$ for $i \in [k]$, we obtain $\overline{Y_i} = (1 \pm O(\epsilon^3))y_i$, where $y_i = (\mathbf{S}\mathcal{X})_i/\|\mathcal{X}\|_1$ as in Figure 4.4. Moreover, we can condition on the event that $|(\mathbf{S}\mathcal{X})_i| < C'/(\epsilon^2)\|\mathcal{X}\|_1$ for all $i \in [k]$, which occurs with probability at least $99/100$ after a union bound and applying the tail bounds for 1-stable random variables (Theorem 1.12 [Nol]). Given this, we have $|y_i| = O(1/\epsilon^2)$ for all $i \in [k]$, so $e^{\overline{Y}_i} = e^{y_i \pm O(\epsilon^3) y_i} = e^{y_i \pm O(\epsilon)} = (1 \pm O(\epsilon))e^{y_i}$. Thus

$$\begin{aligned} -\log\left(\frac{1}{k}\sum_{i=1}^{k} e^{\overline{Y_i}}\right) &= -\log\left((1 \pm O(\epsilon))\frac{1}{k}\sum_{i=1}^{k} e^{y_i}\right) \\ &= -\log\left(\frac{1}{k}\sum_{i=1}^{k} e^{y_i}\right) + O(\epsilon) \\ &= H + O(\epsilon) \end{aligned} \tag{4.9}$$

Where in the first equality, we use the fact that each summand $\sum_{i=1}^{k} e^{\overline{Y_i}}$ is non-negative, so $\sum_{i=1}^{k}(1 \pm O(\epsilon))e^{y_i} = (1 \pm O(\epsilon)) \sum_{i=1}^{k} e^{y_i}$, and the last inequality follows from applying Theorem 34. The central vertex $\mathcal{C}$ can then output $-\log\left(\frac{1}{k}\sum_{i=1}^{k} e^{\overline{Y_i}}\right)$ as the estimate, which completes the proof. □

Since our protocol does not depend on the topology of G, a direct corollary is that we obtain a $\tilde{O}(\epsilon^{-2})$-bits of space *streaming* algorithm for entropy estimation in the random oracle model. Recall that the random oracle model allows the streaming algorithm query access to an arbitrarily long tape of random bits. This fact is used to store the random sketching matrix $\mathbf{S}$.

Theorem 36. *There is a streaming algorithm for ϵ-additive approximation of the empirical Shannon entropy $H(x)$ of a vector $x \in \mathbb{R}^n$ presented in an insertion only stream in the random oracle model. The algorithm succeeds with probability $3/4$, and the space required is $O(\frac{1}{\epsilon^2}(\log\log(n) + \log(1/\epsilon))$ bits.*

4.6 Approximate Matrix Product in the Message Passing Model

In this section, we consider the approximate regression problem in the message passing model over a topology $G = (V, E)$. Here, instead of vector valued inputs, each player is given as input two integral matrices $\mathbf{X}_i \in \{0, 1, 2, \ldots, M\}^{n \times t_1}$, $\mathbf{Y}_i \in \{0, 1, 2, \ldots, M\}^{n \times t_2}$. It is generally assumed that $n >> t_1, t_2$, so the matrices X_i, Y_i are rectangular. Let $\boldsymbol{\mathcal{X}} = \sum_{i=1}^{m} \mathbf{X}_i$ and $\boldsymbol{\mathcal{Y}} = \sum_i \mathbf{Y}_i$. The goal of the players is to approximate the matrix product $\boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{t_1 \times t_2}$. Specifically, at the end of the protocol one player must output a matrix $\mathbf{R} \in \mathbb{R}^{t_1 \times t_2}$ such that

$$\|\mathbf{R} - \boldsymbol{\mathcal{X}}^T \mathcal{Y}\|_F \leq \epsilon \|\boldsymbol{\mathcal{X}}\|_F \|\boldsymbol{\mathcal{Y}}\|_F$$

We now describe a classic sketching algorithm which can be used to solve the approximate regression problem. The algorithm is relatively straightforward: it picks a random matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$. For instance, $\mathbf{S}$ can be a matrix of i.i.d. Gaussian variables with variance $1/k$, or even the count-sketch matrix from Section 4.3.2. It then computes $\mathbf{S}\boldsymbol{\mathcal{X}}$ and $\mathbf{S}\mathcal{Y}$, and outputs $(\mathbf{S}\boldsymbol{\mathcal{X}})^T \mathbf{S}\boldsymbol{\mathcal{Y}}$. In this work, we will use a Gaussian sketch $\mathbf{S}$. The following fact about such sketches will demonstrate correctness.

Lemma 4.6.1 ([KN14]). *Fix matrices $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n \times t_1}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n \times t_2}$ and $0 < \epsilon_0$. Let $\mathbf{S} \in \mathbb{R}^{k \times n}$ be a matrix of i.i.d. Gaussian random variables with variance $1/k$, for $k = \Theta(1/(\delta \epsilon_0^2))$. Then we have*

$$\mathbf{Pr}\left[\|\boldsymbol{\mathcal{X}}^T \mathbf{S}^T \mathbf{S} \mathcal{Y} - \boldsymbol{\mathcal{X}}^T \mathcal{Y}\|_F \leq \epsilon_0 \|\boldsymbol{\mathcal{X}}\|_F \|\boldsymbol{\mathcal{Y}}\|_F\right] \geq 1 - \delta$$

Moreover, with the same probability we have $\|\mathbf{S}\boldsymbol{\mathcal{X}}\|_F = (1 \pm \epsilon_0)\|\boldsymbol{\mathcal{X}}\|_F$ and $\|\mathbf{S}\boldsymbol{\mathcal{Y}}\|_F = (1 \pm \epsilon_0)\|\mathcal{Y}\|_F$

Proof. We first claim that dense Gaussian matrices $\mathbf{S} \in \mathbb{R}^{k \times n}$ satisfy the $(\epsilon, \delta, 2)$-JL moment property (Definition 20 [KN14]) with $k = \Theta(1/(\delta \epsilon^2))$. To see this, note that

$$\|\mathbf{S}x\|_2^2 = \sum_{i=1}^{k} g_i^2 \|x\|_2^2 / k$$

by 2-stability of Gaussians, where g_i's are i.i.d. Gaussian. Thus $k\|\mathbf{S}x\|_2^2$ is the sum of squares of k i.i.d. normal Gaussians, and therefore has variance $3k$ (since Gaussians have 4-th moment equal to 3). It follows that the variance of $\|\mathbf{S}x\|_2^2$, which is precisely $\mathbf{E}\left[(\|\mathbf{S}x\|_2^2 - 1)^2\right]$, is $\Theta(1/k)$. Setting $k = 1/(\delta\epsilon^2)$, it follows that $\mathbf{S}$ has the $(\epsilon, \delta, 2)$-JL moment property. So by Theorem 21 of [KN14], we obtain the approximate matrix product result. □

Recall now by Lemma 4.3.8, with probability $1 - \exp(-1/\delta)$ for a fixed i, j, taken over the randomness used to draw a row $\mathbf{S}_{i,*}$ of $\mathbf{S}$, we have that the central vertex $\mathcal{C}$ can recover a value $r_{\mathcal{C}}^{i,j}$ such that $\mathbf{E}\left[r_{\mathcal{C}}^{i,j}\right] = (\mathbf{S}\boldsymbol{\mathcal{X}})_{i,j}$ and $\text{Var}(r_{\mathcal{C}}^{i,j}) \leq (\epsilon/\delta)^2 \|\boldsymbol{\mathcal{X}}_{*,j}\|_2$, where $\boldsymbol{\mathcal{X}}_{*,j}$ is the j-th *column* of $\boldsymbol{\mathcal{X}}$. Setting $\delta = \Omega(\log n)$, we can union bound over this variance holding for all i, j. Recall the δ here just goes into the $\log(1/\delta)$ communication bound of the algorithm. Altogether, the central vertex obtains a random matrix $\mathbf{R}^T\boldsymbol{\mathcal{X}} \in \mathbb{R}^{k \times t_1}$ such that $\mathbf{E}\left[\mathbf{R}^T\boldsymbol{\mathcal{X}}\right] = (\mathbf{S}\boldsymbol{\mathcal{X}})$ and

$$\mathbf{E}\left[\|\mathbf{R}^T\boldsymbol{\mathcal{X}} - \mathbf{S}\boldsymbol{\mathcal{X}}\|_F^2\right] \leq k(\epsilon/\delta)^2 \sum_{j=1}^{t_1} \|\boldsymbol{\mathcal{X}}_{*,j}\|_2$$

Setting $\epsilon = \text{poly}(1/k) = \text{poly}(1/\epsilon_0)$ small enough, we obtain

$$\mathbf{E}\left[\|\mathbf{R}^T\boldsymbol{\mathcal{X}} - \mathbf{S}\boldsymbol{\mathcal{X}}\|_F^2\right] \leq (\epsilon_0/\delta)^2 \|\boldsymbol{\mathcal{X}}\|_F$$

Similarly, we can obtain a $\mathbf{R}^T\boldsymbol{\mathcal{Y}}$ at the central vertex $\mathcal{C}$, such that

$$\mathbf{E}\left[\|\mathbf{R}^T\boldsymbol{\mathcal{Y}} - \mathbf{S}\boldsymbol{\mathcal{Y}}\|_F^2\right] \leq (\epsilon_0/\delta)^2 \|\boldsymbol{\mathcal{Y}}\|_F$$

Let $\boldsymbol{\Delta}^T\boldsymbol{\mathcal{X}} = \mathbf{R}^T\boldsymbol{\mathcal{X}} - S\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\Delta}^T\boldsymbol{\mathcal{Y}} = \mathbf{R}^T\boldsymbol{\mathcal{Y}} - \mathbf{S}\boldsymbol{\mathcal{Y}}$. By Chebyshev's inequality, we have $\|\boldsymbol{\Delta}^T\boldsymbol{\mathcal{X}}\|_F \leq \epsilon_0^2\|\boldsymbol{\mathcal{X}}\|_F$ and $\|\boldsymbol{\Delta}^T\boldsymbol{\mathcal{Y}}\|_F \leq \epsilon_0^2\|\boldsymbol{\mathcal{Y}}\|_F$ with probability $1 - \delta$, so

$$\begin{aligned}
\|(\mathbf{R}^T\boldsymbol{\mathcal{X}})^T\mathbf{R}^T\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{Y}}\|_F &= \|\boldsymbol{\mathcal{X}}^T\mathbf{S}^T\mathbf{S}\boldsymbol{\mathcal{Y}} + (\boldsymbol{\Delta}^T\boldsymbol{\mathcal{X}})^T\mathbf{S}\boldsymbol{\mathcal{Y}} + \boldsymbol{\mathcal{X}}^T\mathbf{S}^T\boldsymbol{\Delta}^T\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{Y}}\|_F \\
&\leq \|\boldsymbol{\mathcal{X}}^T\mathbf{S}^T\mathbf{S}\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{Y}}\|_F + \|\boldsymbol{\mathcal{X}}^T\mathbf{S}^T\boldsymbol{\Delta}^T\boldsymbol{\mathcal{Y}}\|_F + \|(\boldsymbol{\Delta}^T\boldsymbol{\mathcal{X}})^T\mathbf{S}\boldsymbol{\mathcal{Y}}\|_F \\
&\leq \epsilon_0\|\boldsymbol{\mathcal{X}}\|_F\|\boldsymbol{\mathcal{Y}}\|_F + \|\boldsymbol{\mathcal{X}}^T\mathbf{S}^T\|_F\|\boldsymbol{\Delta}^T\boldsymbol{\mathcal{Y}}\|_F + \|(\boldsymbol{\Delta}^T\boldsymbol{\mathcal{X}})^T\|_F\|\mathbf{S}\boldsymbol{\mathcal{Y}}\|_F \\
&\leq \epsilon_0\|\boldsymbol{\mathcal{X}}\|_F\|\boldsymbol{\mathcal{Y}}\|_F + 2\epsilon_0^2\|\boldsymbol{\mathcal{Y}}\|_F\|\boldsymbol{\mathcal{X}}\|_F \\
&\leq O(\epsilon_0)\|\boldsymbol{\mathcal{X}}\|_F\|\boldsymbol{\mathcal{Y}}\|_F
\end{aligned} \tag{4.10}$$

where we applied Cauchy-Schwartz in the second inequality, giving the desired result. Taken together, this gives the following theorem.

Theorem 37. *Given inputs $\boldsymbol{\mathcal{X}} = \sum_{i=1}^m \mathbf{X}_i, \boldsymbol{\mathcal{Y}} = \sum_{i=1}^m \mathbf{Y}_i$ as described above, there is a protocol which outputs, at the central vertex $\mathcal{C}$, a matrix $\mathbf{R} \in \mathbb{R}^{t_1 \times t_2}$ such that with probability $3/4$ we have*

$$\|\mathbf{R} - \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{Y}}\|_F \leq \epsilon\|\boldsymbol{\mathcal{X}}\|_F\|\boldsymbol{\mathcal{Y}}\|_F$$

The max communication required by the protocol is $O\left(\epsilon^{-2}(t_1+t_2)(\log\log n+\log 1/\epsilon+\log d)\right)$*, where* d *is the diameter of the communication topology* G*.*

Proof. Correctness follows from the above discussion. The communication complexity bound is nearly identical to the proof of Theorem 30. First note that for each coordinate of $\mathbf{S}\boldsymbol{\mathcal{X}}$, exactly one message is sent by each player. Set $K = (Mnm)^2/\gamma$, where $\gamma = (\epsilon\delta/(d\log(nm)))^C$ is the rounding parameter as in Lemma 4.3.8. We can condition on the fact that $|\mathbf{S}_{i,j}| \leq cK^3$ for all i,j and for some constant $c > 0$, which occurs with probability at least $1 - 1/K^2$. Now by Lemma 4.3.8, using the notation of Section 4.3.1, for a fixed coordinate of $\mathbf{S}\boldsymbol{\mathcal{X}}$, we have that $\mathbf{E}\left[e_i^2\right] \leq (j+1)\gamma_0^2 \sum_{v\in T_i} |\langle Q_v, Z\rangle|^2 \leq K^5$, where $e_i = \sum_{u\in T_i} \mathbf{X}_u - r_i$, where r_i is the message sent by the i-th player that coordinate of $\mathbf{S}\mathbf{X}_i$. By Markov's inequality with probability $1-1/K^2$ we have $|e_i| < K^4$. Thus $|r_i| \leq K^6$ for all i.

Now for any r_i for player i in layer ℓ with $|r_i| < 1/(mK)^{d+3-\ell}$, we can simply send 0 instead of r_i. Taken over all the potential children of a player j in layer $\ell+1$, this introduces a total additive error of $1/K^{d+3-\ell}$ in x_j (where x_j is as in Figure 4.2). Now if x_j originally satisfies $|x_j| > 1/K^{d+2-\ell}$, then the probability that this additive error of $1/K^{d+3-\ell}$ changes the rounding result $r_j = \Gamma(x_j)$ is $O(1/K)$, and we can union bound over all m vertices that this never occurs, and then union bound over all $t_1 k < n^2$ coordinates of $\mathbf{S}\boldsymbol{\mathcal{X}}$. Thus, the resulting r_j is unchanged even though x_j incurs additive error. Otherwise, if $|x_j| \leq 1/K^{d+2-\ell}$, then since Player j is in layer $(\ell+1)$, we round their sketch x_j down to 0 anyway. The final result is an additive error of at most $1/K^2$ to $r_{\mathcal{C}}$. Note that we can tolerate this error, as it is only greater than $\gamma\|\boldsymbol{\mathcal{X}}\|_p$ when $\boldsymbol{\mathcal{X}} = 0$, which is a case that can be detected with $O(1)$ bits of communication per player (just forwarding whether their input is equal to 0). With these changes, it follows that $1/(mK)^{d+3} \leq r_j \leq K^6$ for all players j. Thus each message r_j can be sent in $O(\log(\log((mK)^{d+3}))) = O(\log\log(n)+\log(d)+\log(1/\epsilon))$ as needed, and the same bound holds for estimating $\mathbf{S}\boldsymbol{\mathcal{Y}}$.

□

4.7 Lower Bounds

4.7.1 Sketch of the Lower Bound for F_p Estimation in the One-Way Coordinator Model

We now sketch the proof of the $\Omega(m/\epsilon^2)$ lower bound that was remarked upon at the beginning of the Chapter. First, consider the following problem Alice is given a vector $x \in \mathbb{R}^t$, and bob $y \in \mathbb{R}^t$, such that $x_i \geq 0, y_i \geq 0$ for all $i \in [t]$. Alice and Bob both send a message to Eve, who must then output a $(1 \pm \epsilon)$ approximation to $\|x+y\|_p$, for $p \in (0,2] \setminus \{1\}$. Via a reduction from the Gap-Hamming communication problem, there is an $\Omega(1/\epsilon^2)$-bit communication lower bound for this problem [Woo04]. More specifically, there is a distribution $\mathcal{D}$ over inputs $(x,y) \in \mathbb{R}^t \times \mathbb{R}^t$, such that any communication protocol that solves the above problem on these inputs correctly with probability $3/4$ must send $\Omega(1/\epsilon^2)$ bits.

Now consider the one-way coordinator model, where there are m players connected via an edge to a central coordinator. They are given inputs $x_1, \ldots, x_m$, and must each send a single message to the coordinator, who then must estimate $\|x\|_p = \|x_1 + x_2 + \cdots + x_m\|_p$. Consider two distributions, P_1, P_2 over the inputs $(x_1, \ldots, x_m)$. In the first, two players i, j are chosen uniformly at random, and given as inputs $(x,y) \sim \mathcal{D}$, and the rest of the players are given the 0 vector. In P_2, we draw $(x,y) \sim \mathcal{D}$, and every player is given either x or y at random. The players are then either given input from P_1 or P_2, with probability $1/2$ for each. In the first case, if the two players with the input do not send $\Omega(1/\epsilon^2)$ bits, then they will not be able to solve the estimation problem via the 2-party lower bound. However, given only their input, the distributions P_1 and P_2 are indistinguishable to a given player. So the players cannot tell if the input is from P_1 or P_2, so any player that gets an non-zero input must assume they are in case P_1 if they want to solve the communication problem with sufficiently high constant probability, and send $\Omega(1/\epsilon^2)$ bits of communication. This results in $\Omega(m/\epsilon^2)$ total communication when the input is from P_2, which is the desired lower bound.

4.7.2 Lower Bound for Entropy Approximation in Insertion-Only Streams

We now prove the $\Omega(1/\epsilon^2)$-bits of space lower bound for any streaming algorithm that produces an approximation $\tilde{H}$ such that $|\tilde{H} - H| < \epsilon$ with probability $3/4$. Here H is the empirical entropy of the stream vector $\mathcal{X}$, namely $H = H(\mathcal{X}) = -\sum_{i=1}^{n} \frac{|\mathcal{X}_i|}{F_1} \log \frac{|\mathcal{X}_i|}{F_1}$. To prove the lower bound, we must first introduce the GAP-HAMDIST problem. Here, there are two players, Alice and

Bob. Alice is given $x \in \{0,1\}^t$ and Bob receives $y \in \{0,1\}^t$. Let $\Delta(x,y) = |\{i \mid x_i \neq y_i\}|$ be the Hamming distance between two binary strings x, y. Bob is promised that either $\Delta(x,y) \leq t/2 - \sqrt{t}$ (NO instance) or $\Delta(x,y) \geq t/2 + \sqrt{t}$ (YES instance), and must decide which holds. Alice must send a single message to Bob, from which he must decide which case the inputs are in. It is known that any protocol which solves this problem with constant probability must send $\Omega(t)$-bits in the worst case (i.e., the maximum number of bits sent, taken over all inputs and random bits used by the protocol).

Proposition 4.7.1 ([Woo04, JKS08])**.** *Any protocol which solves the* GAP-HAMDIST *problem with probability at least* $2/3$ *must send* $\Omega(t)$*-bits of communication in the worst case.*

We remark that while a $\Omega(1/\epsilon^2)$ lower bound is known for *multiplicative-approximation* of the entropy, to the best of our knowledge there is no similar lower bound written in the literature for additive approximation in the insertion only model. We note that for the turnstile model, a lower bound of $\tilde{\Omega}(1/\epsilon^2 \log(n))$ for additive approximation is given in [KNW10a]. The proof of the following theorem is an adaptation of the proof in [KNW10a], where we restrict the hard instance to have no deletions.

Theorem 38. *Any algorithm for* ϵ*-additive approximation of the entropy* H *of a stream, in the insertion-only model, which succeeds with probability at least* $2/3$*, requires space* $\Omega(\epsilon^{-2})$

Proof. Given a $x, y \in \{0,1\}^t$ instance of GAP-HAMDIST, for $t = \Theta(1/\epsilon^2)$, Alice constructs a stream on $2t$ items. Let x' be the result of flipping all the bits of x, and let $x'' = x \circ 0^t + 0^t \circ x' \in \{0,1\}^{2t}$ where $\circ$ denotes concatenation. Define y', y'' similarly. Alice then inserts updates so that the stream vector $\mathcal{X} = x''$, and then sends the state of the streaming algorithm to Bob, who inserts his vector, so that now $\mathcal{X} = x'' + y''$. We demonstrate that the entropy of H differs by an additive term of at least ϵ between the two cases. In all cases case, we have

$$\begin{aligned} H &= \frac{t-\Delta}{t}\log(t) + \frac{\Delta}{2t}\log(2t) \\ &= \log(t) + \Delta\left(\frac{2\log(t) - \log 2t}{2t}\right) \end{aligned} \tag{4.11}$$

We can assume $t \geq 4$, and then $2\log(t) - \log(2t) = C > 0$, where C is some fixed value known to both players that is bounded away from 0. So as Δ increases, the entropy increases.

Thus in a YES instance, the entropy is at least

$$\begin{aligned} H &\geq \log(t) + (t/2 + \sqrt{t})\frac{C}{2t} \\ &= \log(t) + (1/4 + 1/2\sqrt{t})C \\ &= \log(t) + C/4 + \Theta(\epsilon) \end{aligned} \tag{4.12}$$

In addition, in the NO instance, the entropy is maximized when $\Delta = t/2 - \sqrt{T}$. so we have

$$\begin{aligned} H &\leq \log(t) + (t/2 - \sqrt{t})\frac{C}{2t} \\ &= \log(t) + C/4 - \Theta(\epsilon) \end{aligned} \tag{4.13}$$

Therefore, the entropy differs between YES and NO instances by at least an additive $\Theta(\epsilon)$ term. After sufficient rescaling of ϵ by a constant, we obtain our $\Omega(t) = \Omega(1/\epsilon^2)$ lower bound for additive entropy estimation via the linear lower bound for GAP-HAMDIST from Proposition 4.7.1. □

Chapter 5

Sampling from Distributed Streams

In this chapter, we consider the fundamental problem of maintaining a weighted random sample from a data stream that is partitioned into multiple physically distributed streams. This task generalizes the classic *reservoir sampling* problem [Knu97b, Vit85a] from a centralized to a distributed setting. The results of this chapter are drawn from a joint work with Gokarna Sharma, Srikanta Tirthapura, and David P. Woodruff [JSTW19].

Specifically, we consider the *continuous, distributed, streaming model*, also known as the *distributed functional monitoring model*, first introduced in [CMYZ12]. In this model, there are k physically distributed sites, numbered 1 through k. Each site i receives a local stream of data $\mathcal{S}_i$. The sites are connected to a central coordinator, to which they can send and recieve messages. Queries are posed to the coordinator, asking for an aggregate over $\mathcal{S} = \cup_{i=1}^{k} \mathcal{S}_i$, the union of all streams observed so far. We assume that there is a total ordering over the updates in $\mathcal{S}$, which agrees with the ordering of each individual sequence of updates $\mathcal{S}_i$. In other words, all updates (e, w) which arrives at one of the sites can be though of as arriving at a distinct moment in time, which induces the ordering over $\mathcal{S}$.

Since, oftentimes, the predominant bottleneck in distributed data processing is the network bandwidth, it is highly desirable to have algorithms that communicate as few messages as possible. Thus, the goal of the distributed streaming model is to minimize the *message complexity*, which is the total number of messages sent over the network over the execution of the protocol.[1]

As we have already seen, many streaming tasks can be reduced to sampling from the underlying data. For instance, a search engine that uses multiple distributed servers can maintain

[1]Note that we do not assume a bound on the size (bit complexity) of any given message.

the set of "typical" queries posed to it through continuously maintaining a random sample of all the queries seen thus far. Another application is network monitoring, one of the primary motivations for the streaming model, which deploys multiple monitoring devices within a network. Each device receives extremely high rate data streams, and one of the most commonly desired aggregates is a random sample over all data received so far [DLT04, DLT03b]. It follows that having extremely efficient sampling algorithms is a invaluable primitive for more complicated data mining tasks.

The goal of a distributed weighted sampling protocol is for the coordinator to continuously maintain a weighted random sample of size s from $\mathcal{S}$. Here, a single weighted random sample is item (e, w) sampled with probability proportional to w. There are two important variations of weighted sampling: **sampling with replacement** (SWR) and **sampling without replacement** (SWOR). In the latter, once a item is sampled, it is removed from the set and cannot be sampled again. Thus, for streams containing a skewed distribution of weights, such as a stream with several *heavy hitters*, the resulting samples can be drastically different.

The problem of unweighted distributed sampling, where all weights are equal, is well studied and admits tight upper and lower bounds on message complexity [TW11, CMYZ12, CTW16]. Moreover, for weighted SWR, there is a simple (and tight) reduction to the unweighted case. Thus, for distributed SWR, the complexity of the problem is understood both for the weighted and unweighted cases. However, the same is not true of weighted SWOR. Moreover, in many applications the stream has only a few heavy items which may dominate a random sample when chosen with replacement. Weighted sampling *without replacement* (weighted SWOR) eludes this issue, since such heavy items can be sampled at most once. The design of efficient, distributed weighted SWOR algorithms is therefore a critical task for many data processing applications.

The main result of this chapter is the development of the first message-optimal algorithm for weighted SWOR from a distributed stream, which additionally has optimal space and time complexity. As an application of our sampler, we provide the first distributed streaming algorithm with small message complexity for continuously monitoring heavy hitters with a *residual* error guarantee, and prove a nearly matching lower bound (tight up to $\log(1/\epsilon)$ factors). Here, the residual error guarantee allows one identify heavy hitters in the residual stream after the largest elements are removed, which is especially useful for streams where the weight of items is highly skewed [BICS09]. Finally, we consider the well-studied problem of L_1 tracking (or count tracking) in distributed streams,which requires the coordinator to maintain a $(1 \pm \epsilon)$-approximation of the total weight of all updates seen so far. We introduce an improved algorithm and lower bound for the problem, which resolves the space complexity of this fundamental problem

Highlighted Contributions

The materials from this chapter are drawn from our paper [JSTW19]. The main contributions therein are as follows:

- We design the the first message-optimal algorithm for weighted SWOR from a distributed stream, which uses an optimal expected $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ messages between the sites and the coordinator, and continuously maintains at every point in the stream a weighted SWOR of size s from the items seen so far (Section 5.4).

- We design the first algorithm for distributed monitoring of heavy hitters with residual error, which uses an expected $O\left(\frac{k\log(W)}{\log(k)}+\frac{\log(\epsilon^{-1})\log(W)}{\epsilon}\right)$ messages. We also prove that our algorithm is nearly optimal, by deriving a $\Omega\left(\frac{k\log(W)}{\log(k)}+\frac{\log(W)}{\epsilon}\right)$ lower bound (Section 5.5).

- We resolve the message complexity of L_1 tracking in distributed streams, by providing both improved upper and lower bounds (Section 5.6).

5.1 Background

We begin by formalizing the distributed weighted random sampling problem.

Definition 5.1.1. *Fix k streams $\mathcal{S}_1,\ldots,\mathcal{S}_k$, each consisting of a sequence of items of the form (e,w) where e is an item identifier and $w\in\mathbb{R}_{>0}$ is a positive weight. Let $\mathcal{S}=\cup_{i=1}^{k}\mathcal{S}_i$, let $n=|\mathcal{S}|$, and fix a sample size $s\leq n$. In the distributed weighted random sampling problem, the task is for the coordinator to continuously maintain, at every time step τ, a weighted random sample of size $\min\{s,\tau\}$ from $\mathcal{S}$. Note that the same identifier e can occur multiple times, perhaps in different streams and with different weights, and each such occurrence is to be sampled as if it were a different item.*

We consider two variations of weighted random sampling – sampling with replacement and sampling without replacement. In what follows, a single weighted random sample from $\mathcal{S}$ is defined to be an item chosen from $\mathcal{S}$ where *the probability of choosing item (e,w) is proportional to w, i.e., equal to $\frac{w}{\sum_{(e',w')\in\mathcal{S}}w'}$.*

Definition 5.1.2. *A weighted random sample without replacement (weighted SWOR) from $\mathcal{S}$ is a set S generated according to the following process. Initially S is empty. For i from 1 to s, a*

single weighted random sample is chosen from $(\mathcal{S} \setminus S)$ *and added to* S.

Definition 5.1.3. *A weighted random sample with replacement (weighted SWR) from* $\mathcal{S}$ *is a set* S *generated according to the following process. Initially* S *is empty. For* i *from* 1 *to* s*, a single weighted random sample is chosen from* $\mathcal{S}$ *and added to* S.

Definition 5.1.4. *A distributed streaming algorithm* $\mathcal{P}$ *is a weighted sampler without (with) replacement if for each* $t > 0$*, the coordinator maintains a set* S *of size* $\min\{t, s\}$ *such that* S *is a weighted random sample chosen without (with) replacement from all items seen so far,* $\{(e_1, w_1), \ldots, (e_t, w_t)\}$.

Distributed random sampling generalizes the classic *reservoir sampling* problem [Knu97b, Vit85a] from a centralized to a distributed setting. Random sampling also serves as a building block in other aggregation tasks, such as estimation of the number of distinct elements [CG05, CMY11] and identifying heavy hitters [BO03, KCR06, MSDO05, YZ13]. Distributed random sampling has also been used in approximate query processing in big data systems such as BlinkDB [AMP+13]. The problem of unweighted distributed sampling, where all weights are equal, is well studied and admits tight upper and lower bounds on message complexity [TW11, CMYZ12, CTW16]. However, prior to our work [JSTW19], the problem of weighted SWOR fromk distributed streams had not been studied, though there is prior work in the centralized setting [ES06, BOV15].

Challenges. We remark that designing a message-efficient distributed sampler is a non-trivial task. One challenge is that the state of the system is distributed across multiple sites. It is not possible to keep the distributed state tightly synchronized, since this requires a significant message overhead. Since the states of the sites are not synchronized with the coordinator, sites may send updates to the coordinator even though the sample at the coordinator does not change. A second challenge is that the traditional metrics for a centralized sampling algorithm, such as space complexity and time per item, do not affect the message complexity of a distributed algorithm. For instance, a centralized algorithm that uses $O(1)$ update time and optimal $O(s)$ space need not lead to a distributed algorithm with optimal messages, since the number of messages sent depends on how many times the random sample changes according to the views of the sites and the coordinator. Finally, we emphasize the fact that, in Definition 5.1.4, the protocol must maintain a weighted SWOR *at all times* in the stream. There is no notion of failure in the definition of a weighted sampler – the protocol can never fail to maintain the sample S. These two features make the problem substantially more challenging.

Contributions of this Chapter

The main contribution of this Chapter is the design of the first message-optimal algorithm for weighted SWOR from a distributed stream, which additionally has optimal space and time complexity. In what follows, recall that k is the number of sites (and thus the number of streams $\mathcal{S}_i$), s is the desired sample size, and define $W = \sum_{i=1}^{k} \sum_{(e,w)\in\mathcal{S}_i} w$ to be the total weight received across all sites.

• We present an algorithm for weighted SWOR that achieves an optimal expected message complexity of $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$. The algorithm uses an optimal $\Theta(1)$ space at each site, and an optimal $\Theta(s)$ space at the coordinator. The update time of our algorithm is also optimal: $\Theta(1)$ for a site to process an update, and $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ total runtime at the coordinator. This algorithm is the first message-optimal algorithm for distributed weighted SWOR. We note that a message-efficient algorithm for weighted SWR follows via a reduction to the unweighted case, and obtaining an algorithm for SWOR is significantly harder, as in Section 5.2. A crucial technique used in the development of this algorithm is the *precision sampling framework* [AKO11, JW18b], which is further developed and central to the perfect L_p sampler developed in Chapter 3.

• As an application of our weighted SWOR, we provide the first distributed streaming algorithm with small message complexity for continuously monitoring heavy hitters with a *residual* error guarantee. This allows us to identify heavy hitters in the residual stream after extremely heavy elements are removed. A residual error guarantee is stronger than the guarantee provided by ℓ_1 heavy hitters, and is especially useful for streams where the weight of items is highly skewed [BICS09]. The expected message complexity of our algorithm is:

$$O\left(\frac{k\log(W)}{\log(k)} + \frac{\log(\epsilon^{-1})\log(W)}{\epsilon}\right)$$

We prove that our algorithm is nearly optimal, by also giving a

$$\Omega\left(\frac{k\log(W)}{\log(k)} + \frac{\log(W)}{\epsilon})\right)$$

lower bound, which is tight up to a $\log(1/\epsilon)$ factor in the second term.

• We demonstrate another application of our sampling algorithms to the well-studied problem of L_1 tracking (or count tracking) in distributed streams, which requires the coordinator to maintain a $(1 \pm \epsilon)$ relative error approximation of the total weight seen so that at any given time, with

constant probability, the estimate is correct.

For the case $k \geq 1/\epsilon^2$, the best known upper bound was $O(k \log W)$ messages in expectation [HYZ12]. Our algorithm for L_1 tracking uses

$$O\left(\frac{k \log(W)}{\log(k)} + \frac{\log(\epsilon W)}{\epsilon^2}\right)$$

messages in expectation, which improves on the best known upper bound when $k \geq 1/\epsilon^2$. In this setting, we also improve the lower bound from $\Omega(k)$ to $\Omega(k\frac{\log(W)}{\log(k)})$.

For the case $k \leq 1/\epsilon^2$, matching upper and lower bounds of $\Theta(\frac{\sqrt{k}}{\epsilon} \log W)$ were known [HYZ12]. When $k \geq 1/\epsilon^2$, the lower bound of [HYZ12] becomes $\Omega(\log(W)/\epsilon^2)$. So if $k \geq 1/\epsilon^2$ and $k\frac{\log(W)}{\log(k)} < \log(W)/\epsilon^2$, our upper bound is $O(\log(W)/\epsilon^2)$, which is tight, and otherwise our upper bound is $O(k \log(W)/\log(k))$, which is also tight. Thus, combined with the upper and lower bounds of [HYZ12], our results close the complexity of the distributed L_1 tracking problem.

Roadmap. We present additional preliminaries and basic results in Section 5.3, followed by an optimal algorithm for weighted SWOR in Section 5.4, applications to residual heavy hitters and lower bounds in Section 5.5, and applications to L_1 tracking and lower bounds in Section 5.6.

Other Work on Basic Random Sampling in Streams

Random sampling from a stream is a fundamental problem and there has been substantial prior work on it. The reservoir sampling algorithm (attributed to Waterman [Knu97b]) has been known since the 1960s. There has been much follow-up work on reservoir sampling including methods for speeding up reservoir sampling [Vit85a], sampling over a sliding window [BOZ12, GT02, XTB08, BDM02, GL08], and sampling from distinct elements in data [GT01, Gib01].

The sequential version of weighted reservoir sampling was considered by Efraimidis and Spirakis [ES06], who presented a one-pass $O(s)$ algorithm for weighted SWOR. Braverman et al. [BOV15] presented another sequential algorithm for weighted SWOR, using a reduction to sampling with replacement through a "cascade sampling" algorithm. Unweighted random sampling from distributed streams has been considered in prior works [CMYZ12, CTW16, TW11], which have yielded matching upper and lower bounds on sampling without replacement. Continuous random sampling for distinct elements from a data stream in a distributed setting has been considered in [CT15].

There has been a significant body of research on algorithms and lower bounds in the continuous distributed streaming model. This includes algorithms for frequency moments, [CMY11, CG05], entropy estimation [ABC09b, CZ17], heavy hitters and quantiles [YZ13], distributed counts [HYZ12], and lower bounds on various statistical and graph aggregates [WZ17].

5.2 Overview of the Sampling Algorithm and Techniques

For the problem of sampling *with* replacement, there is a relatively straightforward reduction from the weighted to the unweighted case, which we elaborate on in Section 5.3.2. The reduction involves duplicating a item (e, w) a total of w times into *unweighted* updates, and does not require an increase in message complexity. On the other hand, there are inherent difficulties in attempting to carry out a similar reduction for sampling *without* replacement, which we will now discuss.

On the Difficulty of a Reduction from Weighted SWOR to Unweighted SWOR: We now examine the difficulties which arise when attempting to reduce the problem of weighted SWOR to unweighted SWOR. We consider the following natural candidate reduction. Given a weighted stream of items $\mathcal{S} = \{(e_i, w_i) | i = 1 \dots n\}$, consider an unweighted stream $\mathcal{S}'$ where for each $(e_i, w_i) \in \mathcal{S}$, there are w_i copies of e_i in $\mathcal{S}'$. Note that $\mathcal{S}'$ can be constructed in a streaming manner as items of $\mathcal{S}$ are seen.

Let S' be an unweighted SWOR of s items from $\mathcal{S}'$. We remark that *if S' consists of s distinct identifiers*, then those distinct identifiers are in fact a weighted SWOR of $\mathcal{S}$. The difficulty of using this reduction is that of ensuring s distinct items within S'. One could consider a method that maintains an unweighted SWOR of size greater than s in S', expecting to get at least s distinct identifiers among them. However, this is not straightforward either, due to the presence of heavy-hitters (items with very large weight) which may contribute to a large fraction of the total weight in $\mathcal{S}$. For instance, if there are $s/2$ items that contribute to a more than $1 - 1/(100s)$ fraction of the total weight within $\mathcal{S}$, then S' is likely to contain only identifiers corresponding to these items. This makes it very unlikely that S' has s distinct identifiers, even if the size of S' is much larger than s. If the number of distinct items in S' falls below s, then one could invoke a "recovery" procedure that samples further items from the stream to bring the sample size back to s, but this itself will be a non-trivial distributed algorithm. Moreover, re-initializing or recovering the algorithm would be costly in terms of message complexity, and introduce unintended conditional dependencies into the distribution of the output. Note that one cannot apply distinct sampling [GT01, Gib01] to maintain s distinct items from $\mathcal{S}'$, since distinct

sampling completely ignores the frequency of identifiers in $\mathcal{S}'$ (which correspond to weights in $\mathcal{S}$), while we do not want this behavior – items with a greater weight should be chosen with a higher probability.

Thus, one of the primary difficulties with an algorithm for weighted SWOR is to handle heavy hitters. One could next consider a method that explicitly maintains heavy hitters within the stream (using a streaming algorithm). On the remainder of the stream excluding the heavy hitters, one could attempt to apply the reduction to unweighted SWOR as described above. However, this does not quite work either, since after removing the heavy hitters from the stream, the remaining weight of the stream may still be dominated by just a few items, and the same difficulty persists. One has only swapped one set of heavy hitters for another. Such "residual heavy hitters" may not be heavy hitters in the original stream, and may have evaded capture when the first heavy hitters were identified. This may proceed recursively, where the weight of the stream is still dominated by only a few items after removing the heavy hitters and the residual heavy hitters. Therefore, heavy hitter identification does not solve our problem and further ideas are needed.

Algorithmic Ideas for Weighted SWOR: Our main algorithm for weighted SWOR combines two key ideas. Our first key technical contribution is to divide the items into multiple "level sets" according to the magnitude of their weights. All items within a single "level set" have weights that are close to each other. Our algorithm withholds an item from being considered by the sampler until the level set that the item belongs to has a (pre-specified) minimum number of items. This property ensures that when an item is considered for sampling, there are at least a minimum number of items of the same (or similar) weight, so that the difficulty that we faced with extreme heavy hitters does not surface. When a level set reaches a certain size, all items within the level set are released for further sampling. By choosing the weights of different levels to increase geometrically, we ensure that the number of level sets remains small, and the overhead of additional messages for filling up the level sets is small. While an item is withheld, we also run a procedure so that, at every time step, it will still be output in the sample at that time with the correct probability. Thus, the notion of withholding an item applies means only that it is withheld from an internal sampling algorithm (a subroutine), and not the overall sampler. Note that, for a distributed sampler to be correct, it cannot entirely withhold an item from being sampled, even for a single time step.

Precision Sampling The second idea is the precision sampling framework, which was discussed at length in Section 3.2. Originally developed by Andoni, Krauthgamer, and Onak [AKO11] for sampling in non-distributed data streams, and then extended by [JST11, JW18b], the idea

of precision sampling is to scale each weight w_i by an i.i.d. random variable x_i, which generates a "key" $v_i = w_i x_i$ for each item (e_i, w_i). One then looks at the resulting vector $v = (v_1, v_2, \ldots, v_n)$ of keys, and returns the largest coordinates in v. In particular, the results of our work [JW18b], as discussed in Chapter 3, use the random variables $x_i = 1/t_i^{1/p}$ where t_i is *exponentially distributed*, to develop perfect L_p samplers. A similar idea is also used in "priority sampling" [DLT07] which was developed in the context of network monitoring to estimate subset sums. We use precision sampling to generate these keys for each item, such that the items with the s largest keys form the weighted SWOR of the stream.

It is not difficult to see that if each site independently ran such a sampler on its input–storing the items with the s largest keys–and sent each new sample to the coordinator, who then stores the items with the overall s largest keys, one would have a correct protocol with $O(ks\log(W))$ expected communication. This could be carried out by generating the keys w_i/t_i with exponential variables t_i, or also by using the keys u_i^{1/w_i} with u_i uniform on $(0,1)$, as used for non-distributed weighted sampling without replacement in [ES06]. Thus, a key challenge addressed by our work is to improve the naïve multiplicative bound of $\tilde{O}(ks)$ to an additive bound of $\tilde{O}(k+s)$.

We also remark that by duplicating weighted items *in combination* with our new level set technique, it may be possible to adapt the message-optimal *unweighted* sampling algorithms of [TW11, CMYZ12] to yield a weighted SWOR. The approach would nonetheless require the use of level sets, along with a similar analysis as provided in this work, to remove extremely heavy items from the stream which would cause issues with these samplers. We believe that our approach by scaling an entire weight by a random variable, rather than manually duplicating weighted items (e, w) into w unweighted items, is more natural and simpler to understand. Moreover, it is likely that we obtain improved runtime bounds at the sites by using the algorithm presented in this paper (which are runtime optimal).

Residual Heavy Hitters: For the problem of monitoring heavy hitters, the technique of sampling *with replacement* has been frequently applied. By standard coupon collector arguments, taking $O(\log(1/\epsilon)/\epsilon)$ samples with replacement is enough to find all items which have weight within an ϵ fraction of the total. On the other hand, it is possible and frequently the case that there are very few items which contain nearly all the weight of the stream. This is precisely the domain where SWOR achieves remarkably better results, since a with replacement sampler would only ever see the heavy items. Using this same number of samples *without replacement*, we demonstrate that we can recover all items which have weight within an ϵ fraction of the total *after* the top $1/\epsilon$ largest items are removed. This is *residual error* guarantee is much stronger in the case of skewed distributions of weights, and is a important application of SWOR.

L_1 **Tracking:** Finally, we observe that the following desirable property of our weighted SWOR: namely that, once the heavy hitters are withheld, the values of the keys stored by the algorithm at any time provide good estimates of the total L_1 (the sum of all the weights seen so far). By taking enough samples, we can show that the s-th order statistic, that is the s-largest key overall, concentrates around the value W^t/s, up to a $(1 \pm \epsilon)$ factor, where W^t is the sum of all the weights up to a given time t. Unfortunately, withholding heavy items alone is not sufficient for good message complexity, as one must naively withhold an extra $\Theta(1/\epsilon)$ factor more heavy items than the size of s to obtain good concentration. To avoid a $\Theta(1/\epsilon)$ blow-up in the complexity, we must remove heavy hitters in another way. To do this, we duplicate updates instead of withholding them, a trick used in [JW18b] for a similar sampler. This observation yields an optimal algorithm for distributed L_1 tracking for $k \geq 1/\epsilon^2$, by using our weighted SWOR as a subroutine. Previously an optimal algorithm for L_1 tracking was known only for $k < 1/\epsilon^2$.

5.3 Basic Results on Distributed Sampling

5.3.1 The Continuous Distributed Streaming Model

As in prior work in the continuous distributed streaming model, we assume a *synchronous* communication model, where the system operates in *rounds*, and in each round, each site can observe (at most) one item, and send a message to the coordinator, and receive a response from the coordinator. We assume that the messages are delivered in FIFO order (no overtaking of messages), there is no message loss, and the sites and the coordinator do not crash. We make no assumption on the sizes of the different local streams received by different sites, the order of arrival, and the interleaving of the streams at different sites. The only assumption we have is a global ordering of the stream items by their time of arrival onto one of the sites. If n is the total number of items observed in the system, then for $j = 1 \ldots n$, $o^j = (e_j, w_j)$ is the jth item observed in the total order. The partitioning of the items across different processors is carried out by an adversary.

Our space bounds are in terms of machine words, which we assume are of size $\Theta(\log(nW))$ bits, and that arithmetic operations on machine words can be performed in $O(1)$ time. We also assume that an element identifier and weight fits in a constant number of words. In our algorithms, the length of a message is a constant number of words, so that the number of messages is of the same order as the number of words communicated over the network. For simplicity (but without loss of generality), in the following sections we assume each weight w_j satisfies $w_j \geq 1$. Since each weight w_j can be written in a constant number of machine words by assumption, it

follows that we could always scale the weights by a polynomial factor to ensure $w_j \geq 1$, which blows up the complexity of our algorithms only by a constant, since we only have logarithmic dependency on the total weight.

5.3.2 Basic Results

We first present some basic results for weighted SWR and weighted SWOR. Let n denote the number of items received in the stream.

Algorithm for Weighted SWR: We first derive a message-efficient algorithm for weighted SWR using a reduction to unweighted SWR. For this reduction, we assume that the weights w_i are integers. We first note the following result, from [CMYZ12].

Theorem 39 ([CMYZ12])**.** *There is a distributed algorithm for unweighted SWR with message complexity $O((k + s \log s)\frac{\log n}{\log(2+k/s)})$. The coordinator needs $O(s)$ space and $O((k + s \log s)\frac{\log n}{\log(2+k/s)})$ total time; each site needs $O(1)$ space and $O(1 + \frac{s}{n} \log s \frac{\log n}{\log(2+k/s)})$ time per item amortized.*

Corollary 5.3.1. *There is a distributed algorithm for weighted SWR with message complexity $O((k+s \log s)\frac{\log W}{\log(2+k/s)})$ where W is the total weight received so far. The coordinator needs $O(s)$ space and $O((k + s \log s)\frac{\log W}{\log(2+k/s)})$ total time. Each site needs $O(1)$ space and the amortized processing time per item at a site is $O(1 + \frac{1}{n}(k + s \log s)\frac{\log W}{\log(2+k/s)})$.*

Proof. We reduce weighted SWR to unweighted SWR as follows. Given stream of items with weights $\mathcal{S} = (e_i, w_i), i = 1 \ldots n$, consider an unweighted stream $\mathcal{S}'$ where for each $(e_i, w_i) \in \mathcal{S}$, there are w_i copies of e_i in $\mathcal{S}'$. Note that $\mathcal{S}'$ can be constructed in a streaming manner as items of $\mathcal{S}$ are seen.

Note that an unweighted SWR of s items chosen from $\mathcal{S}'$ is a weighted SWR of s items chosen from $\mathcal{S}$. To prove this, let S' denote an unweighted SWR of size s from $\mathcal{S}'$. Consider an arbitrary item in S'. Since there are w_i copies of e_i in $\mathcal{S}'$, this item is likely to be e_i with probability $w_i / \sum_j w_j$, and hence obeys the distribution we desire of a single item in a weighted SWR of $\mathcal{S}$. Since the items of S' are chosen independent of each other, the entire sample S' has the same distribution as a weighted SWR of s items from $\mathcal{S}$. The number of items in $\mathcal{S}'$ is equal to W, the total weight of $\mathcal{S}$. The message complexity, as well as the space complexity at the sites and the coordinator follow from Theorem 39.

The processing time per item at the site needs additional work. An unweighted SWR sampler simulates the execution of s independent copies of a single element sampler. Each element (e, w) in the weighted input stream leads to w elements into each of the s samplers. Naively done, this reduction leads to a total runtime of $O(sw)$, which can be very large. To improve on this, we speed up the sampling as follows. We note that in the algorithm in Section 3.2 of [CMYZ12] for a single element unweighted SWR, in round j, an element is sent to the coordinator with probability 2^{-j} [2] When w elements are inserted, as in our reduction, the probability that at least one of them is sent to the coordinator is $\alpha(w, j) = 1-(1-2^{-j})^w$. The site can simply send the element (e, w) to the coordinator with probability $\alpha(w, j)$. Repeating this s times leads to a runtime of $O(s)$ per element. To further improve on this, note that across all the s single element samplers, the number of samplers that send a message to the coordinator is a binomial random variable $B(s, \alpha(w, j))$. In the improved algorithm, the site samples a number X from this distribution. If $X > 0$, it chooses a random size X subset of the s samplers, and send (e, w) to the coordinator for each chosen sampler – as also noted in [CMYZ12], this leads to the same distribution as making an independent decision for each sampler. The total time taken at the sites is now of the same order as the number of messages sent to the coordinator, which is $O\left((k + s \log s)\frac{\log W}{\log(2+k/s)}\right)$. The amortized time per element at the site is thus $O\left(1 + \frac{1}{n}(k + s \log s)\frac{\log W}{\log(2+k/s)}\right)$. □

Lower Bound for Weighted SWOR: We next present a lower bound on the message complexity of weighted SWOR, which follows from prior work on the message complexity of unweighted sampling without replacement. Let s denote the desired sample size and k the number of sites.

Theorem 40 ([TW11])**.** *For a constant $q, 0 < q < 1$, any correct algorithm that continuously maintains an unweighted random sample without replacement from a distributed stream must send $\Omega\left(\frac{k \log(n/s)}{\log(1+(k/s))}\right)$ messages with probability at least $1 - q$ and where the probability is taken over the algorithm's internal randomness, and where n is the number of items.*

Corollary 5.3.2. *For a constant $q, 0 < q < 1$, any correct algorithm that continuously maintains a weighted random sample without replacement from $\mathcal{S}$ must send $\Omega\left(\frac{k \log(W/s)}{\log(1+(k/s))}\right)$ messages with probability at least $1-q$, where the probability is taken over the algorithm's internal randomness, and W is the total weight of all items so far.*

Proof. This lower bound follows since unweighted SWOR is a special case of weighted SWOR. We consider an input stream of W items, each with a weight of 1, and apply Theorem 40. □

[2] The algorithm in [CMYZ12] divides execution into rounds; the exact definition of a round does not impact our analysis here, and is hence not provided.

5.4 Weighted SWOR via Precision Sampling

We now present an algorithm for weighted SWOR on distributed streams. Our algorithm utilizes the general algorithmic framework of *precision sampling*, as discussed in Section 3.2 (and more generally used throughout Chapter 3). We recall the basic components of this framework, as it related to sampling from distributed streams, now. When an update (e_i, w_i) is received at a site, the site generates a "key" $v_i = w_i/t_i$, where t_i is a generated exponential random variable. The coordinator then keeps the stream items (e_i, w_i) with the top s largest keys v_i. We first state a result on exponential scaling that allows for the basic correctness of our sampling algorithm.

Proposition 5.4.1 ([Nag06], Equation 11.7 and Remark 1)**.** *Let $\mathcal{S} = \{(e_1, w_1), \ldots, (e_n, w_n)\}$ be a set of items, where each item has an identifier e_i and a weight w_i. Suppose for each $i \in [n]$ we generate a key $v_i = w_i/t_i$, where the t_is are i.i.d. exponential random variables with rate 1 (i.e., with pdf $p(x) = e^{-x}$ for $x \geq 0$). For $k \in [n]$, let the anti-ranks $D(k)$ be defined as the random variable indices such that $v_{D(1)} \geq v_{D(2)} \geq \cdots \geq v_{D(n)}$. For $s \leq n$, let $S(s) \subset [n]$ be the set of items (e_i, w_i) such that $i = D(k)$ for some $k \in [s]$ (i.e. v_i is among the top s largest keys). Then,*

- *$S(s)$ is a weighted SWOR from the set of items $\mathcal{S}$.*
- *We have the distributional equality:*

$$v_{D(k)} = \left(\sum_{j=1}^{k} \frac{E_j}{\sum_{q=j}^{n} w_{D(q)}} \right)^{-1}$$

where the random variables $E_1, \ldots, E_n$ are i.i.d. exponential random variables with rate 1, and are independent of the anti-rank vector $(D(1), D(2), \ldots, D(n))$.

The above remark demonstrates that taking the items with the top s largest keys indeed gives a weighted sample without replacement. The distributional equality given in the second part of Proposition 5.4.1 will be used soon in our analysis. Note that while the above results require continuous random variables to be used, we show that our results are not effected by limiting ourselves to a machine word of precision when generating the exponentials (Proposition 5.4.11). Thus our expected message complexity and runtime take into account the complexity required to generate the exponentials to the precision needed by our algorithm.

To reduce the number of messages sent from the sites to the coordinator, each site locally filters out items whose keys it is sure will not belong to the globally s largest keys. In order to achieve this, our algorithm divides the stream into *epochs*. The coordinator continuously

maintains a threshold u, equal to the value of the smallest key v of an item (e, w, v) held in its sample set S, where S always holds the items (e_i, w_i, v_i) with the s largest values of v_i. For $r = \max\{2, k/s\}$, whenever $u \in [r^j, r^{j+1})$, the coordinator declares to all sites that the algorithm is in epoch i (at the beginning, the epoch is 0 until u first becomes equal to or larger than r). Note that this announcement requires k messages to be sent from the coordinator, and must be done once per epoch.

If the algorithm is in epoch j, and a site receives an update (e_i, w_i), it generates key v_i and sends (e_i, w_i, v_i) to the coordinator if and only if $v_i > r^j$. The coordinator will add the update (e_i, w_i, v_i) to the set S, and if this causes $|S| > s$, it removes the item (e, w, v) from S which has the smallest key v of all items in S. This way, at any time step t, the coordinator holds the items with the s-largest key values $v_i = w_i/t_i$, and outputs the set S as its weighted sample.

A complication in analyzing this algorithm is that it is not possible to directly connect the total weight received so far to the number of epochs that have progressed. For instance, it is not always true that the larger the total weight, the more epochs have elapsed (in expectation). For instance, if a few (fewer than s) items with a very large weight are received, then the total weight received can be large, but the sth largest key is still 0, so that we are still in the zeroth epoch.

To handle this situation, we introduce a *level set* procedure. The idea is to *withhold* heavy items seen in the stream from being sent to the sampler until they are no longer heavy. Here, by releasing an update (e_i, w_i) to the sampler we mean generating a key v_i for the item (e_i, w_i) and deciding whether to accept (e_i, w_i, v_i) into the sample set S. Specifically, we only release a heavy item to the sampler when its weight is no more than a $1/4s$ fraction of the total weight released to the sampler so far. We now defined the *level* of an update (e, w) below (Definition 5.4.2).

Definition 5.4.2. *The level of an item (e, w) is the integer $j \geq 0$ such that $w \in [r^j, r^{j+1})$. If $w \in [0, r)$, we set $j = 0$.*

For $j > 0$, the *level set* D_j will consist of the first $4rs$ items in the stream that are in level j. We do this for all $j \geq 0$, but note that we can clearly stop at $j = \log(W)/\log(r)$, and we do not need to explicitly initialize a level set until at least one item is sent to the set. The coordinator stores the set D_j. As long as $|D_j| < 4rs$, if a site received an item (e, w) that is in level j, the item is sent directly to the coordinator without any filtering at the site, and then placed into D_j. We call such a message an "early" message, as the item (e, w) is withheld from the sampling procedure. We call all other messages which send an item (e, w, v) from the site to the coordinator "regular" messages. Similarly, we call an update that resulted in an early message as an "early update",

and all other updates as "regular updates". Note that a regular update may not result in a regular message, if the key of the regular update is smaller than the threshold for the current epoch.

We call the level set D_j *unsaturated* if $|D_j| < 4rs$ at a given time, and *saturated* otherwise. Each site stores a binary value $\texttt{saturated}_j$, which is initialized to `false`, indicating that $|D_j| < 4rs$. Once $|D_j| = 4rs$, the level set is saturated, and the coordinator generates keys $v_i = w_i/t_i$ for all items $(e_i, w_i) \in D_j$. For each new item-key pair, it then adds (e_i, w_i, v_i) to S if v_i is in the top s largest keys in $S \cup \{(e_i, w_i, v_i)\}$. At this point, the coordinator announces to all the sites that the level set D_j has been saturated (who then set $\texttt{saturated}_j = \texttt{true}$), and thereafter no early updates will be sent for this level set.

Note that in this procedure, the set S will only be a weighted sample over the updates in level sets that are saturated. Since our algorithm must always maintain a weighted sample over *all* stream items which have been received so far, we can simply simulate generating the keys for all items in the unsaturated level sets D_j, and take the items with the top s largest keys in $S \cup (\cup_{j \geq 0} D_j)$ as the weighted sample (see the description of this in the proof of Theorem 41). The algorithm for weighted SWOR is described in Algorithm 5.1 (algorithm at the site), and Algorithms 5.2, 5.3 (algorithm at the coordinator).

We now observe the following result of this level-set procedure.

Lemma 5.4.3. *At any time step, let $(e_1, w_1), \ldots, (e_t, w_t)$ be all items so far that are in saturated level sets. For any $i \in [t]$, $w_i \leq \frac{1}{4s} \sum_{p \in [t]} w_p$.*

Proof. The reason is that for each item (e_i, w_i) in a saturated level set, there are at least $4rs$ items in the same level set, whose weights are within a factor of r of w_i. Thus w_i can be no more than $\frac{1}{4s}$ of the total weight of this level set, and hence no more than $\frac{1}{4s}$ of the total weight. □

5.4.1 Analysis of the Sampling Protocol

We now analyze the message complexity of the algorithm. Let $r = \max\{2, k/s\}$, and let u be the s-th largest value of the keys v_j's given to the coordinator. As described, we define an epoch i as the sequence of updates such that $u \in [r^i, r^{i+1})$. Set $W = \sum_{i=1}^{n} w_i$, where n is the total number of updates at the end of the stream.

Let $(e_1, w_1), \ldots, (e_n, w_n)$ be the set of all stream items. For the sake of analysis, we consider a new ordering of the stream items, which corresponds to the order in which the keys v_i are

Weighted SWOR: Algorithm at Site i

//Initialization
For each level $j \geq 0$:

- Set $\texttt{saturated}_j \leftarrow \texttt{false}$.

  //we will have $u_i = r^j$ whenever the algorithm is in epoch j

$u_i \leftarrow 0$, $r \leftarrow \max\{2, k/s\}$
//End Initialization

If *receive update* (e, w):

1. Let j be the level set of (e, w), i.e. the integer j such that $w \in [r^j, r^{j+1})$. If $w \in (0, 1)$, we set $j = 0$.
2. If $\texttt{saturated}_j = \texttt{false}$, send $(\texttt{early}, e, w)$ to the coordinator.
3. Otherwise, we have $\texttt{saturated}_j = \texttt{true}$, and we generate an exponential variable t. Set $v \leftarrow w/t$.
4. If $v > u_i$, Send $(\texttt{regular}, e, w, v)$ to the coordinator

If *receive ("level saturated", j) from coordinator*:

- Set $\texttt{saturated}_j \leftarrow \texttt{true}$

If *receive ("update epoch", r^j) from coordinator*:

- Set $u \leftarrow r^j$

Figure 5.1: Description of the Protocol at Site i

Weighted SWOR: Algorithm at Coordinator

//Initialization
For each level $j \geq 0$: s:

1. Set $\texttt{saturated}_j \leftarrow \texttt{false}$.
2. $D_j \leftarrow \emptyset$

$u \leftarrow 0$ //sth max of keys of regular items
$S \leftarrow \emptyset$ //the current sample at the coordinator
$r \leftarrow \max\{2, k/s\}$
//End Initialization

If *receive* $(\texttt{early}, e, w)$ *from site* i:

1. Let j be the level set of (e, w), i.e. the integer j such that $w \in [r^j, r^{j+1})$. If $w \in (0, 1)$, we set $j = 0$.
2. Generate key $v = w/i$, and add (e, w, v) to D_j.
3. $|D_j| \geq 4sr$:
   (a) For each $(e', w', v') \in D_j$: call $\texttt{Add-to-Sample}(S, (e', w', v'))$.
   (b) $D_j \leftarrow \emptyset$, $\texttt{saturated}_j \leftarrow \texttt{true}$
   (c) Broadcast ("level saturated", j) to all sites

If *receive* $(\texttt{regular}, e, w, v)$ *from site* i:

- If $v > u$, call $\texttt{Add-to-Sample}(S, (e, w, v))$

If *receive query for sample of* s *items*:

- Return items with the s largest keys in $S \cup (\cup_j D_j)$

Figure 5.2: Algorithm at Coordinator

Weighted SWOR: Add-to-Sample$(S, (e_i, w_i, v_i))$

Input: S is the current sample at the coordinator, and (e_i, w_i, v_i) the item to be inserted
$S \leftarrow S \cup \{(e_i, w_i, v_i)\}$
If $|S| > s$:

1. Let $v_{j_{\min}} = \arg\min_{v_t}\{v_t \mid (e_t, w_t, v_t) \in S\}$
2. $S \leftarrow S \setminus \{(e_{j_{\min}}, w_{j_{\min}}, v_{j_{\min}})\}$

$u_{old} \leftarrow u$
$u \leftarrow \min_{(e,w,v)\in S} v$
If $u \in [r^j, r^{j+1})$ **and** $u_{old} \notin [r^j, r^{j+1})$ *for some* j:

- Broadcast ("update epoch", r^j) to all sites.

Figure 5.3: The Add-to-Sample routine

generated for the items e_i in our algorithm. In other words, we assume the items are ordered $(e_1, w_1), \ldots, (e_{n'}, w_{n'})$, such that the key v_i for e_i was generated before the key for e_{i+1}, and so on. This reordering is simply accomplished by adding each regular item to the ordering as they arrive, but withholding each early item, and only adding an early item to the ordering once the level set D_j that it was sent to is saturated. Note that at the end of the stream, some level sets may not be saturated, thus $n' \leq n$. The remaining $n - n'$ items will have already been sent to the coordinator as early messages, and since no exponentials t_i will have been generated for them when the stream ends, we can ignore them when analyzing the message complexity of the regular stream items. Note that this ordering is a deterministic function of the original ordering and weights of the stream, and has no dependency on the randomness of our algorithm.

Now let $(e_{p_1}, w_{p_1}), (e_{p_2}, w_{p_2}), \ldots, (e_{p_\tau}, w_{p_\tau})$ be the subsequence of the regular items in the sequence $(e_1, w_1), \ldots, (e_{n'}, w_{n'})$, so that $p_1 \leq p_2 \leq \cdots \leq p_\tau$. We group the regular items $e_{p_1}, e_{p_2}, \ldots, e_{p_\tau}$ into sets Ω_i^j as follows. For any $t \in [n']$, let W_t be the total weight seen in the stream up to time t under the new ordering. Thus $W_t = \sum_{i=1}^{t} w_i$. We know by construction of the level sets, at any point in time that if e_{p_i} is a regular item then $w_{p_i} \leq \epsilon_0 W_{p_i - 1}$, where $\epsilon_0 = \frac{1}{4s}$. For any $i \geq 1$, let Ω_j be the set of all regular items (e_{p_i}, w_{p_i}) such that $W_{p_i} \in (sr^{j-1}, sr^j)$. Note that by definition, $\Omega_j = \emptyset$ for all $j > z$, where $z = \lceil \log(W/s)/\log(r) \rceil$. We now further break Ω_i into blocks $\Omega_i^1, \Omega_i^2, \ldots, \Omega_i^{q_i}$, such that Ω_i^t consists of all regular items (e_{p_i}, w_{p_i}) such that $W_{p_i} \in (sr^{i-1}(1+\epsilon)^{t-1}, sr^{i-1}(1+\epsilon)^t)$, for some value $\epsilon = \Theta(1)$ which we will later fix. Note that $q_i < \epsilon^{-1} \log(r)$.

Let Y_i indicate the event that the i-th regular item caused a message to be sent, where we order the items via the new ordering as above. If $\mathfrak{p}$ is the total number of regular items, then we would like to bound $\mathbf{E}\left[\sum_{i=1}^{\mathfrak{p}} Y_i\right]$. Note that the property of an item being regular is independent of the randomness in the algorithm, and is defined deterministically as a function of the ordering and weights of the stream. Thus $\mathfrak{p}$ is a fixed value depending on the stream itself. Then $\mathbf{E}[Y_i] = \sum_{j=1}^{\infty} \mathbf{E}[Y_i | \mathcal{E}_{i,j}] \mathbf{Pr}[\mathcal{E}_{i,j}]$ where $\mathcal{E}_{i,j}$ is the event that we are in epoch j when the i-th update is seen. Note that since $\mathcal{E}_{i,j}$ only depends on the values $t_1, \ldots, t_{i-1}$, it follows that $\mathbf{E}[Y_i | \mathcal{E}_{i,j}] = \mathbf{Pr}[t_i < w_i/r^j] \leq w_i/r^j$. Thus

$$\mathbf{E}[Y_i] \leq w_i \sum_{j=1}^{\infty} r^{-j} \mathbf{Pr}[\mathcal{E}_{i,j}]$$

Our goal will now be to bound the above quantity. We will only concern ourselves with $\mathcal{E}_{i,j}$ when e_i is a regular message, since the early messages will be sent to the coordinator deterministically.

To bound $\mathbf{Pr}[\mathcal{E}_{i,j}]$, we must bound the probability that we are in an epoch j much longer

than expected. To do this, we develop a tail bound on the probability that the s-th largest key $v_i = w_i/t_i$ is much smaller than expected. To do so, we will first need the following standard tail bound on sums of exponential random variables.

Proposition 5.4.4. *Let $E_1, \ldots, E_k$ be i.i.d. mean 1 exponential random variables, and let $E = \sum_{i=1}^{k} E_i$. Then for any $c \geq 1/2$,*

$$\mathbf{Pr}\left[E > ck\right] < \lambda e^{-Cc}$$

where $\lambda, C > 0$ are fixed, absolute constants.

Proof. The moment generating function of an exponential E_i with mean 1 is given by $\mathbf{E}\left[e^{tE_i}\right] = \frac{1}{1-t}$ for $t < 1$. Thus $\mathbf{E}\left[e^{tE}\right] = \mathbf{E}\left[e^{t\sum_{i=1}^{k} E_i}\right] = \prod_{i=1}^{k} \mathbf{E}\left[e^{tE_i}\right] = (\frac{1}{1-t})^k$. Setting $t = 1/2$, then by Markov's inequality, $\mathbf{Pr}\left[E > ck\right] < \mathbf{Pr}\left[e^{tE} > e^{tck}\right] \leq \frac{2^k}{e^{1/2ck}} \leq e^{-1/2ck+k} = \lambda e^{-\Omega(c)}$ for any $c \geq 1/2$ and a fixed constant $\lambda > 0$ as needed. □

We now introduce our main tail bound on the behaviour of the s-th largest key v_i.

Proposition 5.4.5. *Let $w_1, \ldots, w_t$ be any fixed set of weights, and let $W = \sum_{i=1}^{t} w_i$. Fix $\ell \in [t]$, and suppose that $w_i \leq \frac{1}{2\ell} W$ for all $i \in [t]$. Let $v_i = w_i/t_i$ where the t_i's are i.i.d. exponential random variables. Define the anti-ranks $D(k)$ for $k \in [t]$ as the random variables such that $v_{D(1)} \geq v_{D(2)} \geq \cdots \geq v_{D(t)}$. Then for any $c \geq 1/2$, we have*

$$\mathbf{Pr}\left[v_{D(\ell)} \leq \frac{W}{c\ell}\right] \leq O(e^{-Cc})$$

where $C > 0$ is a fixed constant.

Proof. We note that $1/v_i$ is distributed as an exponential with rate w_i. The exponential order statistics of independent non-identical exponential random variables were studied by Nagaraja [Nag06], who demonstrates that the ℓ-th largest value of v_k in $\{v_1, v_2, \ldots, v_t\}$ is distributed as (see Proposition 5.4.1, or equation 11.7 in [Nag06]):

$$v_{D(\ell)} = \left(\sum_{j=1}^{\ell} \frac{E_j}{W - \sum_{q=1}^{j-1} w_{D(q)}}\right)^{-1} \geq \frac{W}{2}\left(\sum_{j=1}^{\ell} E_j\right)^{-1}$$

where the E_i's are i.i.d. exponential random variables with mean 1 that are independent of the anti-ranks $D(1), \ldots, D(t)$. Here, for the inequality, we used the fact that each regular item is at most a $1/(2\ell)$ heavy hitter at every intermediary step in the stream. It follows that if $v_{D(\ell)} \leq$

$W/(c\ell)$, then $\sum_{j=1}^{\ell} E_j \geq \ell c/2$ Which occurs with probability $O(e^{-Cc})$ for some fixed constant $C > 0$ by Proposition 5.4.4. □

Proposition 5.4.6. *Let (e_i, w_i) be any regular stream item, and let a_i, b_i be such that $e_i \in \Omega_{a_i}^{b_i}$. Then we have:*

$$\sum_{j=1}^{\infty} r^{-j} \mathbf{Pr}\left[\mathcal{E}_{i,j}\right] \leq O\left(r^{-a_i+2} e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}} + r^{-a_i+1}\right)$$

where $C > 0$ is some absolute constant.

Proof. First note for $j \geq a_i - 1$, we have $\sum_{j \geq k_i}^{\infty} r^{-j} \mathbf{Pr}\left[\mathcal{E}_{i,j}\right] \leq 2r^{-a_i+1}$. So we now bound $\mathbf{Pr}\left[\mathcal{E}_{i,j}\right]$ for $j < a_i$. Let $v_q = w_q/t_q$ for $q \in [i-1]$, and let $D(p)$ be the anti-ranks of the set $\{v_1, \ldots, v_{i-1}\}$. We note then that if $\mathcal{E}_{i,j}$ holds, then $v_{D(s)} < r^{j+1}$ by definition. Thus $\mathbf{Pr}\left[\mathcal{E}_{i,j}\right] \leq \mathbf{Pr}\left[v_{D(s)} \leq r^{j+1}\right]$. Note that by definition of the level sets $\Omega_{a_i}^{b_i}$, the total weight of all items seen up to time $i-1$ is $W_{i-1} > W_i \frac{1}{(1+\epsilon_0)} > s r^{a_i-1} \frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}$. Here we have used the fact that, by construction of the ordering, no item w_i has weight greater than $\epsilon_0 W_i$ where $\epsilon_0 = \frac{1}{4s}$. This fact will also be needed to apply Proposition 5.4.5.

It then suffices to bound $\mathbf{Pr}\left[v_{D(s)} < r^{j+1}\right] = \mathbf{Pr}\left[v_{D(s)} < \frac{1}{c} W_{i-1}/s\right]$, where

$$c = \frac{W_{i-1}}{r^{j+1} s} \geq \frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)} r^{a_i-j-2}$$

Note that since $j \leq i-2, b_i \geq 1$ and $\epsilon_0 = \frac{1}{4s} < 1/2$, we have $c > 1/2$. So by Proposition 5.4.5, we have

$$\begin{aligned} \mathbf{Pr}\left[\mathcal{E}_{i,j}\right] &\leq \mathbf{Pr}\left[v_{D(s)} < \frac{W_{i-1}}{c \cdot s}\right] \\ &= O(e^{-Cc}) \\ &= O\left(e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)} r^{a_i-j-2}}\right) \end{aligned} \tag{5.1}$$

where $C > 0$ is some absolute constant. Thus

$$r^{-j} \mathbf{Pr}\left[\mathcal{E}_{i,j}\right] \leq O\left(e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)} r^{a_i-j-2}} r^{-j}\right).$$

If $j = a_i - 2$, this is $O(r^{-a_i+2} e^{-\frac{C(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}})$. In general, if $j = a_i - 2 - \ell$ for any $\ell \geq 1$, then this

bound becomes:

$$O\left(r^{-a_i+2+\ell}e^{-\frac{C(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}}r^{\ell}\right) = O\left(r^{-a_i+2-\ell}e^{-\frac{C(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}}\right)$$

where here we used the fact that $(e^{-x})^y = O(e^{-y}x^{-2})$ for any $x \geq 2$ and $y \geq \tau$ where $\tau > 0$ is some constant. Thus $\sum_{\ell=0}^{a_i-1} r^{-a_i+2-\ell}\,\mathbf{Pr}\left[\mathcal{E}_{i,j}\right] = O(r^{-a_i+2}e^{-C\frac{(1+\epsilon)^{b_i-1}}{(1+\epsilon_0)}})$ since $r \geq 2$, which completes the proof. □

Lemma 5.4.7. *Let $\mathfrak{p}$ be the number of regular items in the stream. If $Y_1, Y_2, \ldots$ are such that Y_i indicates the event that the i-th regular stream item (in the ordering defined above) causes a message to be sent, then:*

$$\mathbf{E}\left[\sum_{i=1}^{\mathfrak{p}} Y_i\right] = O\left(sr\frac{\log(W/s)}{\log(r)}\right).$$

Proof. Let $\mathcal{W}_i^j$ be the weight in set Ω_i^j. In other words, $\mathcal{W}_i^j = \sum_{w_{p_t}\in\Omega_i^j} w_{p_t}$. Recall we set $z = \lceil \log(W/s)/\log(r)\rceil$, and note that $\Omega_i = \emptyset$ for $i > z$. Also note that by construction we have $W_i^j < sr^{i-1}(1+\epsilon)^j$. Recall q_i is the number of sets of the form Ω_i^j for some j, and that $q_i \leq O(\epsilon^{-1}\log(r))$ by construction. Using Proposition 5.4.6, we have:

$$\begin{aligned}
\mathbf{E}\left[\sum_i Y_i\right] &\leq \sum_{i=1}^{z}\sum_{j=1}^{q_i}\sum_{e_{p_i}\in\Omega_i^j} w_{p_i} O\left(r^{-i+2}e^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} + r^{-i+1}\right)\\
&\leq \sum_{i=1}^{z}\sum_{j=1}^{q_i}\mathcal{W}_i^j O\left(r^{-i+2}e^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} + r^{-i+1}\right)\\
&\leq O(s)\sum_{i=1}^{z}\sum_{j=1}^{q_i}(1+\epsilon)^j\left(re^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}} + 1\right)\\
&= O(rs)\sum_{i=1}^{z}\sum_{j=1}^{q_i}(1+\epsilon)^j e^{-\frac{C(1+\epsilon)^{j-1}}{(1+\epsilon_0)}}\\
&+ O(s)\left(\sum_{i=1}^{z}\sum_{j=1}^{q_i}(1+\epsilon)^j\right)
\end{aligned}$$

Setting $\epsilon = .5$, we have $(1+\epsilon_0) < (1+\epsilon)$, this is

$$\leq O(sr)\sum_{i=1}^{z}\sum_{j=1}^{q_i}\frac{1.5^j}{\exp\left(C(1.5)^{j-2}\right)} + s\left(\sum_{i=1}^{z}O(r)\right)$$

Now by the ratio test $\sum_{j=1}^{\infty}\frac{(1.5)^j}{\exp(C(1.5)^{j-2})}$ converges absolutely to a constant (recalling $C > 0$ is

just some absolute constant), so the whole sum is:

$$\leq sr\sum_{i=1}^{z} O(1) + sr\left(\sum_{i=1}^{z} O(1)\right) = O\left(sr\frac{\log(W/s)}{\log(r)}\right)$$

where we used $z = \lceil \log(W/s)/\log(r) \rceil$. □

We now bound the number of epochs used in the algorithm. We let the random variable ζ denote the total number of epochs in the algorithm.

Proposition 5.4.8. *If ζ is the number of epochs in the algorithm, then if $z = \lceil \frac{\log(W/s)}{\log(r)} \rceil$, then*

$$\mathbf{E}[\zeta] \leq 3\left(\frac{\log(W/s)}{\log(r)} + 1\right).$$

Proof. After epoch $z + \ell$ for any $\ell \geq 0$, we have that $u > r^\ell W$, where u is the value of the s-th largest key at the coordinator. Let $Y = |\{i \in [n] \mid v_i \geq r^\ell W\}|$. Since the pdf of an exponential random variable is upper bounded by 1, it follows that $\mathbf{Pr}\left[v_i \geq r^\ell W/s\right] = \mathbf{Pr}\left[t_j \leq sr^{-\ell}\frac{w_j}{W}\right] \leq sr^{-\ell}\frac{w_j}{W}$, thus $\mathbf{E}[Y] \leq sr^{-\ell}$. Thus $\mathbf{Pr}[\zeta \geq z + \ell] \leq \mathbf{Pr}[Y \geq s] \leq r^{-\ell}$, where the last inequality follows by a Markov bound. Thus

$$\begin{aligned}\mathbf{E}[\zeta] &\leq z + \sum_{\ell \geq 1}(z+\ell)\,\mathbf{Pr}[\zeta \geq z + \ell] \\ &\leq z + \sum_{\ell \geq 1}(z+\ell)r^{-\ell} \leq 3z.\end{aligned}$$

□

Lemma 5.4.9. *The total expected number of messages sent by the algorithm is $O\left(sr\frac{\log(W/s)}{\log(r)}\right)$, where $r = \max\{2, k/s\}$. Thus this can be rewritten as*

$$O\left(k\frac{\log(W/s)}{\log(1 + k/s)}\right).$$

Proof. For each level set B_t for $t < \log(W/s)/\log(r)$, at most $4rs + k$ messages are sent, corresponding to the $4rs$ messages sent by sites to saturate the set, and then k messages coming from the reply from the coordinator to all k sites announcing that the set B_t is full. This gives a total of $(4rs + k)\log(W/s)/\log(r)$ messages. Finally, there are at most s items with weight greater than W/s, and thus we pay one message for each of these when they arrive at a site. The

level sets corresponding to values greater than W/s will never be saturated, so the total message complexity to handle the early messages is $O(sr \log(W/s)/\log(r))$ as needed.

Next, we pay k messages at the end of every epoch. Thus if ζ is the number of epochs in the algorithm, the expected number of messages sent due to the end of epochs is

$$\mathbf{E}\left[\sum_{i=1}^{\zeta} k\right] = k\,\mathbf{E}\left[\zeta\right] < k\frac{\log(W/s)}{\log(r)} = O(sr\frac{\log(W/s)}{\log(r)})$$

by Proposition 5.4.8. Finally, the expected number of messages sent due to regular stream items is $O(sr \log(W/s)/\log(r))$ due to Lemma 5.4.7, which completes the proof of the message complexity. □

Proposition 5.4.10. *The algorithm described in this section can be implemented, without changing its output behavior, to use $O(s)$ memory and $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ expected total runtime at the coordinator, and $O(1)$ memory and $O(1)$ processing time per update at each site. Note that all four of these bounds are optimal.*

Proof. We first consider the sites. Note that each site only generates a random variable, checks a bit to see if a level set is saturated, and then makes a single comparison to decide whether to send an item, all in $O(1)$ time. For space, note that each site only needs to store a bit that determines whether level set D_j is full for $j \leq \log(W)/\log(r)$. For all level sets j with $j \geq \log(W)/\log(r)$, no item will ever arrive in this level since no item has weight more than W. Thus, to store the bits $\texttt{saturated}_j$, it suffices to store a bit string of length $\log(W)/\log(r)$, which is at most $O(1)$ machine words.

We now consider the coordinator. For space, we demonstrate how the level-set procedure can be carried out using less than $O(sr \log(W)/\log(r))$ space (which is the total numer of items that can be sent to level sets). For each item that arrives at a level set, we generate its key right away. Now note that the items in the level sets with keys that are not among the s largest keys in all of the level sets will never be sampled, and thus never change the set S or the behavior of the algorithm, thus there is no point of keeping them. So at any time, we only store the identities of the items in the level sets with the top s keys (and which level they were in). Call this set S_{level} To determine when a level set becomes saturated, we keep an $O(\log(rs))$-bit counter for each level set D_j, which stores $|D_j|$. Note that we only need to store this for the levels $j \leq \log(W/s)/\log(r)$, since at most s items have weight more than W/s, and such level sets will never be saturated. When a level set D_j becomes saturated, for each item $(e, w, v) \in S_{level}$ such that (e, w) is in level j, we send this item to the main sampler (i.e., we decide whether or not to include it in S based

on the size of its key). The result is the same had the entire level set been stored at all times, and only $O(s)$ machine words are required to store S_{level}, and $O(\log(sr)\log(W)/\log(r)) = O(\log(s)\log(W))$-bits for the counters, which is $O(\log(s)) < O(s)$ machine words, so the total space is $O(s)$ machine words.

For the runtime of the coordinator, each early item and regular item is processed in $O(1)$ time. When a set is saturated, it takes at most $O(s)$ work to send these new items into the main sampler, and this occurs at most $O(\log(W/s)/\log(r))$ times, for a total of $O(s\log(W/s)/\log(r))$ work. Since every other message the coordinator recieves requires $O(1)$ work, the result follows from Lemma 5.4.9. □

We now remark that, up until this point, we have not considered the bit complexity of our messages, or the runtime required to generate the exponentials to the required precision. We address this issue now.

Proposition 5.4.11. *The algorithm for weighted SWOR can be implemented so that each message requires an expected $O(1)$ machine words, and moreover, for any constant $c \geq 1$ uses $O(1)$ machine words with probability at least $1 - W^{-c}$. Each exponential can be generated in time $O(1)$ in expectation, and, for any constant $c \geq 1$, time $O(1)$ with probability at least $1 - W^{-c}$.*

Proof. To see this, we note that a site only needs to generate enough bits to determine whether a given key is large enough to be sent. Recall that the quantile function for the exponential distribution is given by $F^{-1}(p) = -\ln(1-p)$ and, since $1-p$ and p are distributed the same for a uniform variable, an exponential variable t_i can be generated by first generating a uniform random variable U, and outputting $-\ln(U)$ [Knu97a]. Thus, if the algorithm is in epoch j, one can simply generate U bit by bit, until one can determine whether $w_i/t_i > r^j$ or not. So each bit of U generated cuts the remaining probability space by a factor of 2. So for any threshold τ, it requires only $O(1)$ bits to be generated in expectation to determine whether $-\ln(U) < \tau$, and since the probability space is cut by a constant factor with each bit, only $O(\log(W))$ bits (or $O(1)$ machine words) are needed with high probability in W. Thus the coordinator generates the message in expected $O(1)$ time and with high probability, and the size of the message is an expected $O(1)$ machine words with high probability. Similarly, when the coordinator decides whether to accept a sample into the set, it again needs only to check the exponential against a threshold, which requires generating at most an additional expected $O(1)$ bits and with high probability, which completes the proof. □

Theorem 41. *The algorithm of this section for weighted sampling without replacement uses an*

expected $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$ *messages and maintains continuously at every point in the stream a uniform weighted sample size* s *of the items seen so far in the stream.*

The space required for the coordinator is $O(s)$ *machine words, and the space required for each site is* $O(1)$ *words. Each site requires* $O(1)$ *processing time per update, and the total runtime of the coordinator is* $O\left(k\frac{\log(W/s)}{\log(1+k/s)}\right)$.

Proof. The message complexity follows from Lemma 5.4.9, and the space and time complexity follow from Proposition 5.4.10. The bit complexity issues which arise from dealing with exponentials are dealt with in Proposition 5.4.11. For correctness, note that at any point in time, the coordinator maintains two kinds of sets: the set S which consists of the top s samples, and the level sets $D = \cup_{j\geq 0} D_j$. To obtain a true weighted SWOR of the stream up to any point i, the coordinator can simply generate an exponential t_j for each $(e_j, w_j) \in D$, and set $v_j = w_j/t_j$. It then constructs $D' = \{(e_j, w_j, v_j) | (e_j, w_j) \in D\}$, and returns the identifiers e_j in $D' \cup S$ with the s largest keys v_j. The result is that at any point in time, the coordinator can output the set S of the entries (e_j, w_j, v_j) with the top s largest values of v_j.

Let Δ_ℓ be the identifiers e_j with the top ℓ values of v_j. Since $1/v_j$ is exponentially distributed with rate w_j, for any $\ell \geq 1$, if $v_j \notin \Delta_{\ell-1}$ then it follows that the probability that $1/v_j$ is the ℓ-th smallest (or v_j is the ℓ-th largest) is $\frac{w_j}{\sum_{e_t \notin \Delta_{\ell-1}} w_t}$ (via Proposition 5.4.1). Thus the coordinator indeed holds a uniform weighted sample of the stream continuously at all points in time. Note that this sample may contain items in a level set D_j which has not yet been filled at the time of query. However, this does not affect the behavior of the algorithm, since an exponential can be generated for such an item early and used as the sample (see proof of Proposition 5.4.10). By this, we mean that when the algorithm is queried for a weighted sample S over *all* stream items seen so far, it can simply generate a key for each item that is held in an unsaturated level set D_j, and keep the items with the top s largest keys in $S \cup (\cup_{j\geq 0} D_j)$. Note, however, that the true set S will not be modified by this procedure, and the actual items will not be sent into the set S to potentially be sampled until the level set D_j is full. □

5.5 Tracking Heavy Hitters with Residual Error

In this section, we demonstrate that the sampling algorithm developed in Section 5.4 can be used to obtain a new algorithm for continuously monitoring heavy hitters with a *residual* error

guarantee. This is also known as the heavy hitters tracking problem. We show an

$$O\left(\left(\frac{k}{\log(k)}+\frac{\log(1/(\epsilon\delta))}{\epsilon}\right)\log(\epsilon W)\right)$$

upper bound for the message complexity of the problem, along with a nearly tight lower bound $\Omega\left(\left(\frac{k}{\log(k)}+\frac{1}{\epsilon}\right)\log(\epsilon W)\right)$.

While the monitoring of heavy hitters has been studied substantially in the streaming and distributed streaming literature, to date no distributed algorithm has been designed which obtains heavy hitters with a residual error guarantee. We first formalize these variations of the heavy hitters problem. For a vector $x \in \mathbb{R}^n$, and any $t \in [n]$, recall that we define $x_{\text{tail}(t)} \in \mathbb{R}^n$ to be the vector which is equal to x except that the top t largest coordinates $|x_i|$ of x are set to 0. The following is the standard notion of the heavy hitters tracking problem.

Definition 5.5.1. *Let $\mathcal{S} = (e_1, w_1), \ldots, (e_n, e_n)$, and let $x^t \in \mathbb{R}^n$ be the vector with $x_i = w_i$ for $i \leq t$, and $x_i = 0$ otherwise. Then an algorithm $\mathcal{P}$ solves the (ϵ, δ) heavy hitters tracking problem if, for any fixed $t \in [n]$, with probability $1 - \delta$, it returns a set S with $|S| = O(1/\epsilon)$ such that for every $i \in [t]$ with $x_i \geq \epsilon\|x^t\|_1$, we have $x_i \in S$.*

Now, we introduce the variant of the heavy hitters tracking problem that requires residual error.

Definition 5.5.2. *Let $\mathcal{S} = (e_1, w_1), \ldots, (e_n, e_n)$, and let $x^t \in \mathbb{R}^n$ be the vector with $x_i = w_i$ for $i \leq t$, and $x_i = 0$ otherwise. Then an algorithm $\mathcal{P}$ solves the (ϵ, δ) heavy hitters tracking problem with residual error if, for any fixed $t \in [n]$, with probability $1 - \delta$ it returns a set S with $|S| = O(1/\epsilon)$ such that for every $i \in [t]$ with $x_i \geq \epsilon\|x^t_{\text{tail}(1/\epsilon)}\|_1$, we have $x_i \in S$.*

Note that the residual error guarantee is strictly stronger, and captures substantially more information about the data set when there are very large heavy items.

Theorem 42. *There is a distributed streaming algorithm $\mathcal{P}$ which solves the (ϵ, δ) heavy hitters problem with residual error, and sends an expected $O\left(\left(\frac{k}{\log(k)}+\frac{\log(1/(\epsilon\delta))}{\epsilon}\right)\log(\epsilon W)\right)$ messages. The algorithm uses $O(1)$ space per site, $O(1)$ update time per site, $O(\frac{\log(1/(\delta\epsilon))}{\epsilon})$ space at the coordinator, and $O\left(\left(\frac{k}{\log(k)}+\frac{\log(1/(\epsilon\delta))}{\epsilon}\right)\log(\epsilon W)\right)$ overall runtime at the coordinator.*

Proof. To obtain the bound, we run our weighted SWOR of Theorem 41 with $s = 6\frac{\log(1/(\delta\epsilon))}{\epsilon}$, where $C > 0$ is a sufficently large constant. Fix a time step t, and let S^t be the sample

obtained by the weighted SWOR at that time. We know that S^t is a weighted SWOR from $(e_1, w_1), \ldots, (e_t, w_t)$. Let x^t be the vector corresponding to these weights, and let $T \subset [t]$ be the set of coordinates such that $x_i \geq \epsilon \|x^t_{\text{tail}(1/\epsilon)}\|_1$. Let $T_0 \subset T$ be the subset of i such the x^t_i is one of the $1/\epsilon$ largest values in x^t. Now fix any $i \in T$, and break S into blocks $S^1, S^2, \ldots, S^{2\log(1/(\delta\epsilon))}$, each of size $3/\epsilon$. Now for each $j \in [2\log(1/(\delta\epsilon))]$, there are at least $2/\epsilon$ items which are not in T_0 (which follows since $|T_0| \leq 1/\epsilon$). Conditioned on a sample $s \in S^j$ not being in T_0, since $x_j \geq \epsilon \|x^t_{\text{tail}(1/\epsilon)}\|_1$ by assumption, the probability that $s = i$ is at least ϵ. Thus the probability that $i \notin S^j \leq (1-\epsilon)^{2/\epsilon} < 1/2$. Repeating $2\log(1/\epsilon)$ times, the probability that $i \in S$ is at least $1 - (\frac{1}{2})^{2\log(1/(\delta\epsilon))} < 1 - (\epsilon\delta)^2$. We can then union bound over all $|T| < 2/\epsilon$ items, so that $T \subset S$ with probability at least $1 - \delta$. To restrict the set S to $O(1/\epsilon)$ items, we simply order the items in S by weight and output the top $2/\epsilon$, which will contain T if S does, which completes the proof of correctness. The remainder of the Theorem then follows from the complexity bounds of Theorem 41. □

5.5.1 Lower Bound for Tracking Heavy Hitters

We now demonstrate an $\Omega(k\log(W)/\log(k)+\log(W)/\epsilon))$ lower bound on the expected message complexity of any distributed algorithm that monitors the heavy hitters in a weighted stream (Definition 5.5.1). Since the residual error guarantee is strictly stronger, this lower bound extends to monitoring heavy hitters with residual error.

Theorem 43. *Fix any constant $0 < q < 1$, and let $\epsilon \in (0, 1/2)$. Then any algorithm which $(\epsilon/2, \delta) = (\epsilon, q^2/64)$ solves the heavy hitters tracking problem (Definition 5.5.1), must send at least $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$ messages with probability at least $1 - q$ (assuming $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$). In particular, the expected message complexity of such an algorithm with $\delta = \Theta(1)$ is $\Omega(k\log(W)/\log(k) + \epsilon^{-1}\log(W))$.*

Proof. Create $s = \Theta(\frac{1}{\epsilon}\log(W))$ global stream updates (e_i, w_i), such that $w_i = (1+\epsilon)^i\epsilon$ and $w_0 = 1$. Then note for each $i \geq 0$, we have that w_i is an $\epsilon/(1+\epsilon) > \epsilon/2$ heavy hitter in $w_1, w_2, \ldots, w_i$. Note that the total weight of the stream is $\sum_{i=1}^{s} w_i = W$.

Now consider any algorithm $\mathcal{P}$ for $(\epsilon/2, q^2/64)$ heavy hitter tracking, and at any time step t let S denote the set of heavy hitters held by $\mathcal{P}$,). On a given time step $t \in [n]$, let S_t denote the value of S at time t. and let R denote the concatenation of all random coin flips used by the algorithm $\mathcal{P}$ for both the sites and the coordinator. We first claim the following:

Claim 5.5.3. *Let $0 < q < 1$ be any constant. Suppose $\mathcal{P}$ is a randomized algorithm which $(\epsilon/2, q/64)$-solves the heavy hitter tacking problem. Suppose the set of heavy hitters it maintains at all times is called S. Then there is a constant $C' = C'(q) > 0$ such that:*

- *The set S changes at least $\frac{C'}{\epsilon}\log(n)$ times with probability at least $1 - q$.*

Proof. Consider the above stream instance. Note that on each time step, the new update (e_i, w_i) becomes an $\epsilon/2$ heavy hitter, and therefore should be accepted into the set S. Note moreover that at any time step t, the total weight is $(1+\epsilon)^t$. Suppose there is a randomized algorithm $\mathcal{P}$ which for each $t \in T$, succeeds in outputting all the ϵ heavy hitters in a set S at time t with probability $1 - q/64$. Let X_t be a random variable that indicates that $\mathcal{P}$ is correct on t. We have $\mathbf{E}[X_t] > 1 - q/64$, and clearly $\mathbf{E}[X_t X_{t'}] \leq 1$ for any t, t'. Thus

$$\begin{aligned} \text{Var}\left(\sum_{i=1}^{\frac{1}{\epsilon}\log(n)} X_i\right) &\leq \frac{1}{\epsilon}\log(n)(1-q/64) + \frac{1}{\epsilon^2}\log^2(n) - \frac{1}{\epsilon^2}\log^2(n)(1-q/64)^2 \\ &\leq (q/30)\frac{1}{\epsilon^2}\log^2(n) \end{aligned} \tag{5.2}$$

for n larger than some constant. By Chebyshev's inequality:

$$\mathbf{Pr}\left[\left|\frac{1}{\epsilon}\log(n)(1-q/64) - \sum_{i \leq \epsilon^{-1}\log(n)} X_i\right| > \frac{1}{q}\sqrt{q/30}\frac{\log(n)}{\epsilon}\right] < q$$

Thus with probability at least $1 - q$, $\mathcal{P}$ is correct on at least a $\frac{C'}{\epsilon}\log(n)$ number of the points in T, for $C' = (1 - q/64 - \frac{1}{\sqrt{30}} -) > 1/2$. Now suppose that, conditioned on this, the algorithm changed S fewer than $\frac{C'}{\epsilon}\log(n)$ times. Note that every time a new item (e_i, w_i) arrives, if $\mathcal{P}$ is correct on time i, it must be the case that $(e_i, w_i) \in S$ once update i is done processing. Thus if S did not change, but $(e_i, w_i) \in S_t$, this means $(e_i, w_i) \in S_{t-1}$ – the coordinator contained e_i in its set before the item arrived! By having e_i's being $O(\log(W))$ bit identifiers and randomizing the identities in the stream, it follows that the probability that this could occur at any time step t is less than $(W/\epsilon)(1/\text{poly}(W)) < W^{-100}$ for e_i's with $O(\log(W))$ bits and a sufficiently large constant. So we can safely condition on this event not occurring without affecting any of the claims of the theorem. It follows that S must change on every time step t on which it is correct, which completes the first claim that S must change at least $\frac{C'}{\epsilon}\log(n)$ times with probability at least $1 - q$. □

Observe that the lower bound of of $\Omega(\log(W)/\epsilon)$ messages being sent with probability at

least $1-q$ follows directly from the last claim. For the $\Omega(k\log(W)/\log(k))$ lower bound, we construct a new weighted stream. Define $\eta = \Theta(\frac{\log(W)}{\log(k)})$ *epochs*, as follows. In the i-th epoch, each site j receives *one* weighted update (e_i^j, k^i). Note then at the very beginning of the i-th epoch (for any i), the total weight seen in the stream so far is at most $2k^i$. Thus the first update (e_i^j, k^i) that arrives in the i-th epoch will be a $1/2$ heavy hitter, and thus must be accepted into S for the protocol to be correct on that time step.

Now consider any epoch i, and let X_i be the number of sites which sent or received a message from the coordinator in epoch i. Note that X_i is a lower bound on the number of messages sent in epoch i. Let X_i^j indicate that site j sent or recieved a message from the coordinator in epoch i, so that $X_i = \sum_{j\in[k]} X_i^j$. We claim $\mathbf{E}[X_i] = \Omega(k)$. To demonstrate this, consider the first site j^* which recieves an update in epoch i. Since item $(e_i^{j^*}, k^i)$ is immediately a $1/2$ heavy hitter, it must be sent to be correct. Since the protocol is correct on the $ki+1$'st time step with probability at least $1-q/64$, it follows that site j^* must send a message after receiving its update in epoch i with probability $1-q/64$, so $\mathbf{E}\left[X_i^{j^*}\right] > 1-q/64$.

But note that when a site j receives its update in the i-th epoch, if it has not communicated to the coordinator since the start of the epoch, it does not know the order in which it recieved the item. In particular, it does not know whether it was the first site to receive an item. Since the randomized protocol must be correct on any advesarial fixing of the ordering, it follows that if site j does not know if it recieved the first update, it must nevertheless send it with probability at least $1-q/64$ to be a correct protocol. In other words $\mathbf{E}\left[X_i^j\right] > 1-q/64$ *for all* $j\in[k]$, from which $\mathbf{E}[X_i] > (1-q/64)k$, and therefore $\mathbf{E}[\sum_{i=1}^{\eta} X_i] > (1-q/64)k$ follows. Thus $\mathbf{E}[k\eta - \sum_{i=1}^{\eta} X_i] < k\eta q/64$, and so by a Markov bound over the last expectation, $\mathbf{Pr}[\sum_{i=1}^{\eta} X_i < (1-1/64)k\eta] < q$. Since X_i lower bounds the number of messages sent in the i-th epoch, this completes the proof. □

5.6 L_1 tracking

In this section, we demonstrate how our sampling algorithm from Section 5.4 can be used to design a message-optimal L_1 tracking algorithm, which will close the complexity of this problem. We first recall the definition of an L_1 tracking algorithm.

Definition 5.6.1. *Given a distributed weighted data stream $\mathcal{S} = (e_1, w_1), (e_2, w_2), \ldots, (e_n, w_n)$ and parameters $\epsilon, \delta > 0$, a distributed streaming algorithm (ϵ, δ) solves the L_1 tracking problem if the coordinator continuously maintains a value $\widetilde{W}$ such that, at any fixed time $t\in[n]$, we have*

$\widetilde{W} = (1 \pm \epsilon)W_t$ *with probability* $1 - \delta$*, where* $W_t = \sum_{i=1}^{t} w_i$.

In [CMYZ12], an L_1 tracker was given that had $O(k \log(W) \log(1/\delta))$ expected messages. An improved bound of $O((k + \sqrt{k} \log(1/\delta)/\epsilon) \log(W))$ expected messages was then provided in [HYZ12], along with a lower bound of $\Omega(\frac{\sqrt{k}}{\epsilon} \log(W))$ for all $k \leq \frac{1}{\epsilon^2}$. We remark that while the bounds of [HYZ12] are stated as $O(\sqrt{k}/\epsilon \log(W))$ (for constant δ), the actual expected message complexity of their algorithm is $O((k+\sqrt{k} \log(1/\delta)/\epsilon) \log(W))$ (in [HYZ12] the authors assume that $k < 1/\epsilon^2$). Nevertheless, the aforementioned bound is, up to this point, the current best known distributed algorithm for L_1 tracking.

In this section, we give tight bounds for the case $k > 1/\epsilon^2$. More specifically, we prove an L_1 tracking algorithm with expected message complexity:

$$O\left(\frac{k \log(\epsilon W)}{\log(k)} + \frac{\log(\epsilon W) \log(1/\delta)}{\epsilon^2}\right)$$

In the following section, we also provide an $\Omega(k \frac{\log(W)}{\log(k)})$ lower bound for the problem. We remark that the results of [HYZ12] show an $\Omega(\frac{\sqrt{k}}{\epsilon} \log(W))$ lower bound assuming that $k < \frac{1}{\epsilon^2}$, and they give a matching upper bound in this regime. When $k > \frac{1}{\epsilon^2}$, the lower bound becomes $\Omega(\frac{1}{\epsilon^2} \log(W))$, which can be seen by simply applying the same lower bound on a subset of $1/\epsilon^2$ of the sites. Then when $k > 1/\epsilon^2$, if $k \log(W)/\log(k) < \log(W)/\epsilon^2$, our upper bound is $O(\log(W)/\epsilon^2)$, which is tight. Otherwise, our upper bound is $O(k \log(W)/\log(k))$, which matches our lower bound of $\Omega(k \log(W)/\log(k))$. Thus our protocol is optimal whenever $k > 1/\epsilon^2$. Since tight upper and lower bounds were known for the case of $k < 1/\epsilon^2$ by [HYZ12], this closes the complexity of the problem.

We also note that our lower bound assumes $1/\epsilon < W^{1-\xi}$ for some fixed constant $\xi > 0$. Thus our lower bound does not contradict our upper bound of $\Omega(k \log(\epsilon W)/\log(k) + \log(\epsilon W)/\epsilon^2)$ (for $\delta = \Theta(1)$). Observe that for $1/\epsilon = \Theta(W)$, one can simply send every stream update for $O(W)$ communication, which is also clearly a lower bound since any stream update not immediately sent would cause the coordinator to no longer have a $(1 \pm \epsilon)$ approximation.

Note that to obtain an L_1 tracking algorithm, where the coordinator has a value $\widetilde{W}$ such that $(1-\epsilon)W_t \leq \widetilde{W} \leq (1+\epsilon)W_T$ for *every* step $t = 1, 2, \ldots, n$ with probability $1 - \delta$, our algorithm uses

$$O\left(\frac{k \log(\epsilon W)}{\log(k)} + \frac{\log(W/\epsilon) \log(\frac{\log(W)}{\epsilon\delta})}{\epsilon^2}\right)$$

expected message complexity. This fact simply follows from setting δ to be small enough to

union bound over the $\frac{\log(W)}{\epsilon\delta}$ points in the stream where the L_1 increases by a factor of $(1+\epsilon)$.

The known upper and lower bounds for L_1 tracking on a single time step with failure probability $\delta = \Theta(1)$ are summarized below. Note again, constant L_1 tracking at all points in the stream is accomplished in all cases by setting $\delta = \log(W)/\epsilon$, and note that the $\log(1/\delta)$ does not multiply the $O(k\log(W)/\log(k))$ term in our upper bound. With the combination of the upper and lower bounds of [HYZ12], this completely closes the complexity of the problem.

Citation	Message Complexity (upper or lower bound)
[CMYZ12] + folklore	$O\left(\frac{k}{\epsilon}\log(W)\right)$
[HYZ12]	$O\left(k\log(W) + \frac{\sqrt{k}\log(W)\log(1/\delta)}{\epsilon}\right)$
This work	$O\left(\frac{k\log(\epsilon W)}{\log(k)} + \frac{\log(\epsilon W)}{\epsilon^2}\right)$
[HYZ12]	$\Omega\left(\frac{\sqrt{\min\{k,1/\epsilon^2\}}}{\epsilon}\log(W)\right)$
This work	$\Omega\left(k\frac{\log(W)}{\log(k)}\right)$

Our algorithm for L_1 tracking is described formally in Algorithm 16 below. To show correctness, we first prove Proposition 5.6.2, which is a standard tail bound for sums of exponential random variables. We utilize the techniques from our weighted SWOR algorithm to assign an key to each item (e, w) in the stream. Then, by the concentration of sums of exponential random variables, we will demonstrate that the s-th largest key can be used to obtain a good approximation to the current L_1 of the stream, where $s = \Theta(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$

Algorithm 16: Tracking L_1

`Input:` Distributed stream $\mathcal{S}$

`Output:` a value $\tilde{W}$ such that $\tilde{W} = (1 \pm \epsilon)W_t$ at any step time t with probability $1 - \delta$.

Initalization:

Instantiate weighted SWOR algorithm $\mathcal{P}$ of Theorem 41 with $s = \Theta(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$

On Stream item (e, w)

1. Duplicate (e, w), a total of $\ell = \frac{s}{2\epsilon}$ times, to yield $(e^1, w), (e^2, w), \ldots, (e^\ell, w)$. Insert each into the algorithm $\mathcal{P}$.

On Query for L_1 at time t:

1. Let u be the value stored at the coordinator of the s-th largest key held in the sample set S.

2. Output $\tilde{W} = \frac{su}{\ell}$.

Proposition 5.6.2. *Let $E_1, E_2, \ldots, E_s$ be i.i.d. exponential random variables with mean 1. Then for any $\epsilon > 0$, we have:*

$$\mathbf{Pr}\left[\left|\sum_{j=1}^{s} E_j - s\right| > \epsilon s\right] < 2e^{-\epsilon^2 s/5}$$

Proof. The moment generating function of $\sum_{j=1}^{s} E_j$ is given by $(\frac{1}{1-t})^s$ for $t < 1$,

$$\begin{aligned}\mathbf{Pr}\left[\sum_{j=1}^{s} E_j > (1+\epsilon)s\right] &< \frac{(\frac{1}{1-t})^s}{e^{t(1+\epsilon)s}} \\ &\leq \exp\left(-t(1+\epsilon)s + s\log(\frac{1}{1-t})\right)\end{aligned}$$

Setting $t = \epsilon$ and taking $\epsilon < 1/2$ we obtain

$$\begin{aligned}&\leq \exp\left(-\epsilon^2 s - \epsilon s - s\log(1-\epsilon)\right) \\ &\leq \exp\left(-\epsilon^2 s - \epsilon s + s(\epsilon + \epsilon^2/2 + \epsilon^3/3 + \ldots)\right) \\ &\leq \exp\left(-\epsilon^2 s/5\right)\end{aligned}$$

as needed. Next, we have

$$\begin{aligned}\mathbf{Pr}\left[\sum_{j=1}^{s} E_j < (1-\epsilon)s\right] &< \mathbf{Pr}\left[e^{-t\sum_{j=1}^{s} E_j} > e^{-t(1-\epsilon)s}\right] \\ &\leq \frac{(\frac{1}{1+t})^s}{\exp(-t(1-\epsilon)s)} \\ &\leq \exp\left(t(1-\epsilon)s - s\log(1+t)\right)\end{aligned}$$

setting $t = \epsilon$:

$$\begin{aligned}&\leq \exp\left(-\epsilon^2 s + \epsilon s - s(\epsilon + \epsilon^2/2 + \epsilon^3/3 + \ldots)\right) \\ &\leq \exp\left(-\epsilon^2 s/5\right)\end{aligned}$$

and union bounding over the upper and lower tail bound gives the desired result. □

Theorem 44. *There is a distributed algorithm (Algorithm 16), where at every time step the*

coordinator maintains a value $\widetilde{W}$ such that at any fixed time $\tau \in [n]$, we have

$$(1-\epsilon)W_\tau \leq \widetilde{W} \leq (1+\epsilon)W_\tau$$

with probability $1-\delta$, where $W_\tau = \sum_{i=1}^{\tau} w_i$ is the total weight seen so far at step τ. The expected message complexity of the algorithm is $O\left(\left(\frac{k}{\log(k)} + \epsilon^{-2}\log(1/\delta)\right)\log\left(\frac{\epsilon W}{\log(\delta^{-1})}\right)\right)$

Proof. If W_τ is the total weight seen so far in the stream, then after duplication the weight of the new stream $\mathcal{S}'$ that is fed into $\mathcal{P}$ is $W_\tau \frac{2s}{\epsilon}$. Call the total weight seen up to time τ of the new stream $\overline{W}_\tau$. The duplication has the effect that at any time step τ in the original stream, every item $(e_j^i, w_j) \in \mathcal{S}'$, corresponding to a duplicate of $(e_j^i, w_j) \in \mathcal{S}$, is such that $w_j \leq \frac{\epsilon}{2s}\overline{W}_\tau$. Thus once an update to $\mathcal{S}$ is finished being processed, no item in $\mathcal{S}'$ is more than an $\frac{\epsilon}{2s}$ heavy hitter. Since there are $\ell > s$ duplications made to each update, the level sets used in $\mathcal{P}$ will be immediately saturated the moment the first update in $\mathcal{S}$ with a weight in that level set arrives. Thus there is never any intermediate point in the stream $\mathcal{S}$ where some of the weight of the stream is being withheld in the level sets used by $\mathcal{P}$. Thus the total weight of the stream seen so far at any given time has already been sent to the sample, so each item (e_i, w_i) seen so far in the stream has had a key v_i generated for it.

Now the coordinator holds the value u such that u is the s-th largest value in $\{v_1, v_2, \ldots, v_{\ell\tau}\}$, where ℓ was the number of duplications. Here $v_i = w_i/t_i$, where t_i is an exponetial variable and w_i is a weight of one of the duplicates sent to the stream $\mathcal{S}'$. By results on the order statistics of collections of exponential variables ([Nag06]), we have the distributional equality

$$u = \left(\sum_{j=1}^{s} \frac{E_j}{\overline{W}_\tau - \sum_{q=1}^{j-1} w_{D(q)}}\right)^{-1}$$

where $E_1, E_2, \ldots, E_s$ are i.i.d. exponential random variables, and $D(q)$ are the random variable indices such that $v_{D(1)} \geq V_{D(2)} \geq \cdots \geq V_{D(\tau\ell)}$ (the $D(q)$'s are known as anti-ranks). Since $w_{D(q)} \leq \frac{\epsilon}{2s}\overline{W}_\tau$, it follows that

$$
\begin{aligned}
u &= \left((1 \pm \epsilon) \sum_{j=1}^{s} \frac{E_j}{\overline{W}_\tau}\right)^{-1} \\
&= (1 \pm O(\epsilon))\overline{W}_\tau \left(\sum_{j=1}^{s} E_j\right)^{-1} \\
&= (1 \pm O(\epsilon))\ell W_\tau \left(\sum_{j=1}^{s} E_j\right)^{-1}
\end{aligned}
$$

Now by Proposition 5.6.2, we have $\mathbf{Pr}\left[|\sum_{j=1}^{s} E_j - s| > \epsilon s\right] < 2e^{-\epsilon^2 s/5} < \delta$, where here we set $s = 10\log(\delta^{-1})/\epsilon^2$. Conditioned on this not occurring, we have $u = (1 \pm O(\epsilon))\ell W_\tau \frac{1}{s}$, thus $u\frac{s}{\ell} = (1 \pm O(\epsilon))W_\tau$ as required, and the expected message complexity follows by Theorem 41 setting $s = \Theta(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$. □

Corollary 5.6.3. *There is a distributed algorithm where at every time step the coordinator maintains a value $\widetilde{W}$ such that*

$$(1-\epsilon)W_\tau \leq \widetilde{W} \leq (1+\epsilon)W_\tau$$

for every $\tau \in [n]$ with probability $1-\delta$. The expected message complexity of the algorithm is

$$O\left(\left(\frac{k}{\log(k)} + \epsilon^{-2}\log(\frac{\log(W)}{\delta\epsilon})\right)\log\left(\frac{\epsilon W}{\log(\delta^{-1})}\right)\right).$$

Proof. The result follows from running Theorem 44 with $\delta' = \log(W)/(\delta\epsilon)$, and union bounding over the $\log(W)/\epsilon$ time steps t in the stream where W_t increases by a factor of $(1+\epsilon)$. □

5.6.1 Lower Bound for L_1 Tracking

We now demonstrate an $\Omega(k\log(W)/\log(k) + \log(W)/\epsilon))$ lower bound on the expected message complexity of any distributed L_1 tracker. We remark again that our lower bound does not contradict out upper bound of $O(k\log(\epsilon W)/\log(k) + \log(\epsilon W)/\epsilon^2)$, since it requires $1/\epsilon < W^{1-\xi}$ for some constant ξ.

Note that the lower bound holds for both weighted and unweighted streams, as the hard example in the proof is a stream where all weights are equal to 1. We remark that the $\Omega(\log(W)/\epsilon)$ portion of the bound is only interesting when $k < \frac{1}{\epsilon}$, and in this regime a better lower bound of $\Omega(\frac{\sqrt{k}}{\epsilon}\log(W))$ is given by [HYZ12]. We include this portion of the bound in the Theorem

for completeness, but remark that the main contribution of the Theorem is the lower bound of $\Omega(k \log(W)/\log(k))$.

Theorem 45. *Fix any constant $0 < q < 1$. Then any algorithm which $(\epsilon, \delta) = (\epsilon, q^2/64)$ solves the L_1 tracking problem (Definition 5.6.1), must send at least $\Omega(k \log(W)/\log(k) + \epsilon^{-1} \log(W))$ messages with probability at least $1 - q$ (assuming $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$). In particular, the expected message complexity of such an algorithm with $\delta = \Theta(1)$ is $\Omega(k \log(W)/\log(k) + \epsilon^{-1} \log(W))$.*

Proof. We proceed much in the way of Theorem 43. We define $\eta = \Theta(\frac{\log(W)}{\log(k)})$ epochs as follows: at the end of the the $i-th$ epoch, for $i = 0, 1, 2, \ldots, \eta - 1$, there will have been exactly k^i global stream updates processed since the start of the stream, each with weight 1. Thus each epoch i is deterministically defined, and contains $k^{i+1} - k^i$ global updates. Let the unweighted updates be $e_1, e_2, \ldots, e_n$ (we can make these weighted by simply adding a weight of 1. Here $n = W$, so the stream is both weighted and unweighted. In each epoch, we partition the updates arbitrarily over the sites k.

Now consider any algorithm $\mathcal{P}$ for L_1 tracking, and at any time step t let u denote the value held by $\mathcal{P}$ which tracks the L_1 (total weight seen so far in the stream). On a given time step $t \in [n]$, let u_t denote the value of u at time t, and let R denote the concatination of all random coin flips used by the algorithm $\mathcal{P}$ for both the sites and the coordinator. We first claim the following:

Claim 5.6.4. *Let $0 < q < 1$ be any constant. Suppose $\mathcal{P}$ is a randomized algorithm which $(\epsilon, q/64)$-solves the L_1 tracking problem. Suppose the value it maintains for the L_1 approximation at all times is called u. Then there is a constant $C' = C'(q) > 0$ such that the following holds:*

- *The value u changes at least $\frac{C'}{\epsilon} \log(W)$ times with probability at least $1 - q$.*

Proof. Consider the time steps $T = \{1, \texttt{rnd}(1+\epsilon), \texttt{rnd}((1+\epsilon)^2), \ldots, \ldots, n\}$, where the L_1 changes be a factor of $(1+\epsilon)$, where `rnd` rounds the vaue to the nearest integer (note we can assume n is a power of $(1+\epsilon)$ by construction). Suppose there is a randomized algorithm $\mathcal{P}$ which for each $t \in T$, succeeds in outputting a $(1 \pm \epsilon)$ approximation to t at time t with probability $1 - q^2/64$. Assuming that $1/\epsilon < W^{1-\xi}$ for some constant $\xi > 0$, we have $|T| > \frac{1}{\epsilon} \log(W)$ distinct such points. Let X_t be a random variable that indicates that $\mathcal{P}$ is correct on t. We have

$\mathbf{E}[X_t] > 1 - q/64$, and clearly $\mathbf{E}[X_t X_{t'}] \leq 1$ for any t, t'. Thus

$$\begin{aligned}\text{Var}\left(\sum_{i=1}^{\frac{1}{\epsilon}\log(W)} X_i\right) &\leq \frac{1}{\epsilon}\log(W)(1-q/64) + \frac{1}{\epsilon^2}\log^2(W) - \frac{1}{\epsilon^2}\log^2(W)(1-q/64)^2 \\ &\leq (q/30)\frac{1}{\epsilon^2}\log^2(n)\end{aligned} \tag{5.3}$$

for n larger than some constant. By Chebyshev's inequality:

$$\mathbf{Pr}\left[\left|\frac{1}{\epsilon}\log(W)(1-q/64) - \sum_{i \leq \epsilon^{-1}\log(W)} X_i\right| > \frac{1}{\sqrt{q}}\sqrt{q/30}\frac{\log(W)}{\epsilon}\right] < q$$

Thus with probability at least $1-q$, $\mathcal{P}$ is correct on at least a $\frac{C'}{\epsilon}\log(W)$ number of the points in T, for $C' = (1 - q/64 - \frac{1}{\sqrt{30}} -) > 1/2$. Now suppose that, conditioned on this, the algorithm changed u less than $\frac{C'}{\epsilon}\log(W)$ times. But each time u changes, it can be correct for at most one of the values in T, which contradicts the fact that $\mathcal{P}$ is correct on at least a $\frac{C'}{\epsilon}\log(W)$ of these points. So with probability at least $1-q$, the value of u must change at least $\frac{C'}{\epsilon}\log(W)$ times, as needed.

□

Note that the lower bound of $\Omega(\log(W)/\epsilon)$ messages being sent with probability at least $1-q$ follows directly from the last claim. Note moreover, that our lower bound reduces to $\Omega(\log(W)/\epsilon)$ when $k < 8/\epsilon$, so we can now assume that $k > 8/\epsilon$.

Now consider any epoch i, and let X_i be the number of sites which sent or received a message from the coordinator in epoch i. Let X_i^j indicate that site j sent or recieved a message from the coordinator in epoch i, so that $X_i = \sum_{j \in [k]} X_i^j$. We claim $\mathbf{E}[X_i] = \Omega(k)$. To demonstrate this, first note that at the beginning of an epoch, the current L_1 of the stream is k^i. First condition on the event $\mathcal{E}_i$ that the estimate of the algorithm is correct at the beginning of the epoch. Note $\mathbf{Pr}[\mathcal{E}_i] > 1 - q/64$. By the law of total expectation: $\mathbf{E}[X_i] \geq \mathbf{E}[X_i \mid \mathcal{E}_i](1 - q/64)$.

Now consider the stream of updates σ_i^j in epoch i which send exactly $2k^i$ consecutive updates to each site j. Consider any stream σ_i constructed by composing σ_i^j's for any arbitrary permutation of the j's. Note that if site j^* recieved the first set of updates in σ_i and does not send a message after these updates, the algorithm will have an incorrect L_1 estimate at time $3k^i$. Since this does not occur with probability at least $1 - q/64$, it follows that $\mathbf{E}\left[X_i^{j^*}\right] \geq (1 - q/64)$. But note that this must hold for all sites j, and once a site j receives its set of $2k^i$ updates,

since it could have been the first one to recieve the updates (unless it hears otherwise from the coordinator). In other words, since the randomized protocol must maintain the same guarantee for any ordering of the stream, it follows that every site j must also send a message to the coordinator with probability at least $1 - q/64$ after it sees its $2k^i$ updates (unless the coordinator sends a message to it first). Thus $\mathbf{E}\left[X_i^j\right] \geq (1 - q/64)$ for all j, which completes the claim that $\mathbf{E}[X_i] \geq (1 - q/64)k$, giving $\mathbf{E}[\sum_{i=1}^{\eta} X_i] > k\eta(1 - q/64)$. Thus $\mathbf{E}[k\eta - \sum_{i=1}^{\eta} X_i] < k\eta q/64$, and so by a Markov bound over the last expectation, $\mathbf{Pr}[\sum_{i=1}^{\eta} X_i < (1 - 1/64)k\eta] < q$. Since X_i lower bounds the number of messages sent in the i-th epoch, this completes the proof.

□

Chapter 6

Data Streams with Bounded Deletions

As introduced in Section 1.1, the two prevalent models in the data stream literature are the insertion-only and turnstile models. The turnstile model, being the more general of the two, captures a significantly wider array of applications. Unfortunately, many important streaming problems require a $\Theta(\log n)$ multiplicative factor more space for turnstile streams than for insertion-only streams. For instance, identifying a coordinate $i \in [n]$ for which $|x_i| > \frac{1}{10}\sum_{j=1}^{n} |x_j|$ can be accomplished using only $O(\log n)$-bits of space in the insertion-only model [BDW16], whereas the same task requires $\Omega(\log^2 n)$ bits in the turnstile model [JST11]. This "complexity gap" naturally motivates the study of an intermediate streaming model which avoids the extra $\log n$ factor provably required for turnstile algorithms, but still maintains sufficient expressivity to model many real-world data streams.

In this Chapter, we introduce such an intermediary model, known as the *bounded deletion model*, which was first proposed in our work [JW18a]. The bounded deletion model avoids the lower bounds for turnstile streams by bounding the number of deletions that are allowed. While similar notions of bounded deletion streams have been appeared in the past for their practical applicability [GGR16a],[1] prior to our work, no comprehensive theoretical study of data stream algorithms in this setting existed.

Specifically, in the bounded deletion model we are given a parameter $\alpha \geq 1$, and promised that, at the end of the stream, the norm $\|x\|_p$ is at least a $1/\alpha$-fraction of the L_p norm of the hypothetical stream had all updates been positive. Here, the value of p we choose depends on the application. This gives a fluid medium between insertion only streams (with $\alpha = 1$), and

[1] See also [CJMM16], where a bound on the maximum number of edges that could be deleted in a graph stream was useful.

turnstile streams (with $\alpha = \text{poly}(n)$).

We show that for streams with this α-property, for many fundamental streaming problems we can replace a $O(\log n)$ factor in the space usage for algorithms in the turnstile model with a $O(\log \alpha)$ factor. This is true for identifying heavy hitters, inner product estimation, L_0 estimation, L_1 estimation, L_1 sampling, and support (L_0) sampling. For each problem, we will provide matching or nearly matching lower bounds for α-property streams. We note that in practice, many important turnstile data streams are in fact α-property streams for small values of α. For such applications, our results represent significant improvements in efficiency for all the aforementioned problems.

Highlighted Contributions

The materials from this chapter are drawn from our paper [JW18a]. The main contributions therein are as follows:

- We introduce the *bounded deletion model*, which is an intermediate model between the insertion-only and turnstile model (Section 6.1).
- We give improved algorithms for many fundamental streaming problems in the bounded deletion model, with space complexity usually a $\log \alpha$ instead of a $\log n$ factor larger than the corresponding required complexity in the insertion-only model. This continuously links the complexity of the insertion-only and turnstile models (Sections 6.2-6.7).
- For each of the streaming problems we consider, we prove matching or nearly-matching lower bounds in the bounded deletion model (Section 6.8).

6.1 Background

We begin by formally introducing the bounded deletion model. First, we define the *insertion vector* $I \in \mathbb{R}^n$ to be the frequency vector of the substream of positive updates ($\Delta_t \geq 0$), and the *deletion vector* $D \in \mathbb{R}^n$ to be the entry-wise absolute value of the frequency vector of the

substream of negative updates. In other words, we have:

$$I_j = \sum_{\substack{t \,:\, i_t = j \\ \Delta_t \geq 0}} \Delta_t \qquad \text{and} \qquad D_j = \sum_{\substack{t \,:\, i_t = j \\ \Delta_t < 0}} |\Delta_t|$$

for each $j \in [n]$ on a stream with updates $(i_1, \Delta_1), (i_2, \Delta_2), \ldots, (i_m, \Delta_m)$. Notice that $x = I - D$ by definition. We then define the bounded deletion model as follows:

Definition 6.1.1 (α-Bounded Deletion Streams)**.** *For $\alpha \geq 1$ and $p \geq 0$, a data stream satisfies the L_p α-property if*

$$\|I + D\|_p \leq \alpha \|x\|_p$$

For $p = 1$, the definition simply asserts that the final L_1 norm of x must be no less than a $1/\alpha$ fraction of the total weight of updates in the stream $\sum_{t=1}^{m} |\Delta_t|$. For strict turnstile streams, this is equivalent to the number of deletions being less than a $(1 - 1/\alpha)$ fraction of the number of insertions, hence a bounded deletion stream.

For $p = 0$, the α-property simply states that $\|x\|_0$, the number of non-zero coordinates at the end of the stream, is no smaller than a $1/\alpha$ fraction of the number of distinct elements seen in the stream. Importantly, note that for both cases this constraint need only hold at the time of query, and not necessarily at every point in the stream.

Observe for $\alpha = 1$, we recover the insertion-only model, whereas for $\alpha = mM$ in the L_1 case or $\alpha = n$ in the L_0 case we recover the turnstile model (with the minor exception of streams with $\|x\|_p = 0$). So α-property streams are a natural parameterized intermediate model between the insertion-only and turnstile models. For clarity, in this Chapter we use the term *unbounded deletion stream* to refer to a (general or strict) turnstile stream which do not satisfy the α property for any $\alpha = o(n)$.

For many applications of turnstile data streaming algorithms, the streams in question are in fact α-property streams for small values of α. For instance, in network traffic monitoring it is useful to estimate differences between network traffic patterns across distinct time intervals or routers [M+05]. If x_i^1, x_i^2 represent the number of packets sent between the i-th [source, destination] IP address pair in the first and second intervals (or routers), then the stream in question is $x^1 - x^2$. In realistic systems, the traffic behavior will not be identical across days or routers, and even differences as small as 0.1% in overall traffic behavior (i.e. $\|x^1 - x^2\|_1 > .001\|x^1 + x^2\|_1$) will result in $\alpha < 1000$ (which is significantly smaller than the theoretical universe size of $n \approx 2^{256}$ potential IP addresses pairs in IPv6).

A similar case for small α can be made for differences between streams whenever these differences are not arbitrarily small. This includes applications in streams of financial transactions, sensor network data, and telecommunication call records [Ind04, CJK+04], as well as for identifying newly trending search terms, detecting DDoS attacks, and estimating the spread of malicious worms [PDGQ05, EVF03, Moo01, LCD05, WP05].

A setting in which α is likely even smaller is database analytics. For instance, an important tool for database synchronization is Remote Differential Compression (RDC)[TBG+06, ABFS02], which allows similar files to be compared between a client and server by transmitting only the differences between them. For files given by large data streams, one can feed these differences back into sketches of the file to complete the synchronization. Even if as much as a half of the file must be resynchronized between client and sever (an inordinately large fraction for typical RDC applications), streaming algorithms with $\alpha = 2$ would suffice to recover the data.

For the L_0-norm, there are important applications of streams with bounded value of α. For example, L_0 estimation is applicable to networks of cheap moving sensors, such as those monitoring wildlife movement or water-flow patterns [Ind04]. In such networks, some degree of clustering is expected (in regions of abundant food, points of water-flow accumulation), and these clusters will be consistently occupied by sensors, resulting in a bounded ratio of inactive to active regions. Furthermore, in networking one often needs to estimate the number of distinct IP addresses with active network connections at a given time [GGR16b, M+05]. Here we also observe some degree of clustering on high-activity IP's, with persistently active IP's likely resulting in an value pf α much larger than $1/n$ (where n is the universe size of IP addresses).

In many of the above applications, α can even be regarded as a constant when compared with n. For such applications, the space improvements detailed in Figure 6.1 are considerable, and reduce the space complexity of the problems nearly or exactly to known upper bounds for insertion-only streams [BDW16, Mor78, KNW10b, JST11].

Finally, we remark that in some cases it is not unreasonable to assume that the magnitude of *every* coordinate would be bounded by some fraction of the updates to it. For instance, in the case of RDC it is seems likely that none of the files would be totally removed. We summarize this stronger guarantee as the *strong α-property*.

Definition 6.1.2. *For $\alpha \geq 1$, a data stream satisfies the strong α-property if*

$$I_i + D_i \leq \alpha |x_i|$$

for all coordinates $i \in [n]$.

Note that this property is strictly stronger that the L_p α-property for any $p \geq 0$. In particular, it forces $x_i \neq 0$ if i is updated in the stream. In this Chapter, however, we focus primarily on the more general α-property of Definition 6.1.1, and use α-property to refer to Definition 6.1.1 unless otherwise explicitly stated. Nevertheless, we show that our lower bounds for L_p heavy hitters, L_1 estimation, L_1 sampling, and inner product estimation, all hold even for the more restricted strong α-property streams.

Contributions for Bounded Deletion Streams

As promised, we demonstrate that for many well-studied streaming problems, we can replace a $\log(n)$ factor in algorithms for general turnstile streams with a $\log(\alpha)$ factor for α-property streams. This is a significant improvement for small values of α. Our upper bound results, along with the lower bounds for the unbounded deletion case, are given in Figure 6.1. Several of our results come from the introduction of a new data structure, CSSamplingSimulator (Section 6.2), introduced in Section 6.2, which produces estimates of the frequencies x_i with small additive error. The data structure CSSamplingSimulator, roughly, is the result of first subsampling updates to the data stream, and then applying the classic count-sketch algorithm (Section 2.3.2).

Importantly, our improvements from CSSamplingSimulator and other L_1 problems are the result of applying sampling techniques, where we sample *individual updates*, to α-property streams While sampling individual updates to data streams has been studied in many papers, most have been in the context of insertion only streams (see, e.g., [Coh15, CDK+09, CDK+14, CCD11, EV03, GM98a, Knu98, MM12, Vit85b]). Notable examples of the use of sampling in the presence of deletions in a stream are [CCD12, GLH08, Haa16, GLH06, GLH07]. We note that these works are concerned with unbiased estimators and do not provide the $(1 \pm \epsilon)$-approximate relative error guarantees with small space that we obtain. They are also concerned with unbounded deletion streams, whereas our algorithms exploit the α-property of the underlying stream to obtain considerable savings.

In addition to upper bounds, we give matching or nearly matching lower bounds in the α-property setting for all the problems we consider. In particular, for the L_1-related problems (heavy hitters, L_1 estimation, L_1 sampling, and inner product estimation), we show that these lower bounds hold even for the stricter case of strong α-property streams. The statements and proofs of these lower bounds can be found in Section 6.8.

Problem	Turnstile L.B.	α-Property U.B.	Citation	Notes
ϵ-Heavy Hitters	$\Omega(\epsilon^{-1}\log^2(n))$	$O(\epsilon^{-1}\log(n)\log(\alpha))$	[JST11]	Strict-turnstile succeeds w.h.p.
ϵ-Heavy Hitters	$\Omega(\epsilon^{-1}\log^2(n))$	$O(\epsilon^{-1}\log(n)\log(\alpha))$	[JST11]	Gen.-turnstile $\delta = O(1)$
Inner Product	$\Omega(\epsilon^{-1}\log(n))$	$O(\epsilon^{-1}\log(\alpha))$	Theorem 69	Gen.-turnstile
L_1 Estimation	$\Omega(\log(n))$	$O(\log(\alpha))$	Theorem 64	Strict-turnstile
L_1 Estimation	$\Omega(\epsilon^{-2}\log(n))$	$O(\epsilon^{-2}\log(\alpha))$ $+\log(n))$	[KNW10a]	Gen.-turnstile
L_0 Estimation	$\Omega(\epsilon^{-2}\log(n))$	$O(\epsilon^{-2}\log(\alpha))$ $+\log(n))$	[KNW10a]	Gen.-turnstile
L_1 Sampling	$\Omega(\log^2(n))$	$O(\log(n)\log(\alpha))$	[JST11]	Gen.-turnstile $(*)$
L_0 Sampling	$\Omega(\log^2(n))$	$O(\log(n)\log(\alpha))$	[KNP+17]	Strict-turnstile

Figure 6.1: The best known lower bounds (L.B.) for classic data stream problems in the turnstile model, along with the upper bounds (U.B.) for α-property streams from this paper. The notes specify whether an U.B./L.B. pair applies to the strict or general turnstile model. For simplicity, we have suppressed $\log\log(n)$ and $\log(1/\epsilon)$ terms, and all results are for $\delta = O(1)$ failure probability, unless otherwise stated. $(*)$ L_1 sampling note: strong α-property, with $\epsilon = \Theta(1)$ for both U.B. & L.B.

We also demonstrate that for general turnstile streams, obtaining a constant approximation of the L_1 still requires $\Omega(\log(n))$-bits for α-property streams. For streams with unbounded deletions, there is an $\Omega(\epsilon^{-2}\log(n))$ lower bound for $(1 \pm \epsilon)$-approximation [KNW10a]. Although we cannot remove this $\log n$ factor for α-property streams, we are able to show an upper bound of $\tilde{O}(\epsilon^{-2}\log\alpha + \log n)$ bits of space for strong α-property streams, where the $\tilde{O}$ notation hides $\log(1/\epsilon) + \log\log n$ factors. We thus separate the dependence of ϵ^{-2} and $\log n$ in the space complexity, illustrating an additional benefit of α-property streams, and show a matching lower bound for strong α-property streams.

Overview of Techniques

Our results for L_1 streaming problems in Sections 6.2 to 6.5 are built off of the observation that for α property streams, the number of insertions and deletions made to any given coordinate $i \in [n]$ is upper bounded by $\alpha\|x\|_1$. This simply follows from the fact that $|I_i + D_i| \leq \|I + D\|_1 \leq \alpha\|x\|_1$ by definition of the α-property. Thus, if we were to uniformly sample $\text{poly}(\alpha/\epsilon)$ updates to such a coordinate i, we would obtain an unbiased estimator of x_i with standard deviation at most $\epsilon\|x\|_1$. On the other hand, the new, sub sampled, coordinate only requires $\log(\alpha/\epsilon)$-bits of

space to store, instead of $\log n$.

To exploit this fact, in Section 6.2 we introduce a data structure $\mathrm{CSSamplingSimulator}$ inspired by the classic Countsketch of [CCFC02b], which simulates running each row of Countsketch on a small uniform sample of stream updates. Our data structure does not correspond to a valid instantiation of Countsketch on any stream since we sample different stream updates for different rows of Countsketch. Nevertheless, we show via a Bernstein inequality that our data structure obtains the Countsketch guarantee plus an $\epsilon\|x\|_1$ additive error, with only a logarithmic dependence on ϵ in the space. This results in more efficient algorithms for the L_1 heavy hitters problem (Section 6.3), and is also used in our L_1 sampling algorithm (Section 6.4). We are able to argue that the counters used in our algorithms can be represented with much fewer than $\log n$ bits because we sample a very small number of stream updates.

Additionally, we demonstrate that sampling $\mathrm{poly}(\alpha/\epsilon)$ updates preserves the inner product between two α-property streams $f, g \in \mathbb{R}^n$ up to an additive $\epsilon\|f\|_1\|g\|_1$ error. Then by hashing the sampled universe down modulo a sufficiently large prime, we show that the inner product remains preserved, allowing us to estimate it in $O(\epsilon^{-1}\log(\alpha))$ space (Section 6.2.2).

Our algorithm for L_1 estimation (Section 6.5) utilizes our biased coin observation to show that sampling will recover the L_1 of a strict turnstile α-property stream. To carry out the sampling in $o(\log(n))$ space, give a alternate analysis of the well known Morris counting algorithm, giving better space but worse error bounds. This allows us to obtain a rough estimate of the position in the stream so that elements can be sampled with the correct probability. For L_1 estimation in general turnstile streams, we analyze a virtual stream which corresponds to scaling our input stream by Cauchy random variables, argue it still has the α-property, and apply our sampling analysis for L_1 estimation on it.

Our results for the L_0 streaming problems in Sections 6.6 and 6.7 mainly exploit the α-property in sub-sampling algorithms. Namely, many data structure for L_0 streaming problems subsample the universe $[n]$ at $\log(n)$ levels, corresponding to $\log(n)$ possible thresholds which could be $O(1)$-approximations of the L_0. If, however, an $O(1)$ approximation were known in advance, we could immediately subsample to this level and remove the $\log(n)$ factor from the space bound. For α property streams, we note that at any given point in time, the total number of non-zero coordinates seen after t updates, given by $\|I^{(t)} + D^{(t)}\|_0$, must be bounded in the interval $[\|x^{(t)}\|_0, \alpha \cdot \|x\|_0]$. Thus, by employing an $O(1)$ estimator R^t of $\|I^{(t)} + D^{(t)}\|_0$, we show that it suffices to sub-sample to only the $O(\log(\alpha/\epsilon))$ levels which are closest to $\log(R^t)$ at time t, from which our space improvements follows.

Transformation to Unit Update Streams

Let $(i_1, \Delta_1), \ldots, (i_t, \Delta_t)$ be the updates to the stream. For Sections 6.2 to 6.5, we now remark that it will suffice to assume that each update Δ_t satisfies $\Delta_t \in \{-1, 1\}$. For general updates, we can implicitly consider them to be several consecutive updates in $\{-1, 1\}$, and our analysis will hold in this expanded stream. This implicit expanding of updates is only necessary for our algorithms which sample updates with some probability p. If an update $|\Delta_t| > 1$ arrives, we update our data structures with the value $\text{sign}(\Delta_t) \cdot \text{Bin}(|\Delta_t|, p)$, where $\text{Bin}(n, p)$ is the binomial random variable on n trials with success probability p, which has the same effect.

In this unit-update setting, the L_1 α property simply reduces to the statement $m \leq \alpha \|x\|_1$. Thus, in the remainder of the Chapter, this is the definition which we will use for the L_1 α property.

Roadmap of the Chapter

In Section 6.2, we introduce the data-structure CSSamplingSimulator, which will be used centrally for several streaming tasks in this chapter. In Section 6.2.2, we show how the data structure CSSamplingSimulator can be used to solve the inner product estimation task, and in Section 6.3 we show how the data structure CSSamplingSimulator can be used to solve the heavy hitters problem. In Section 6.4, we consider the L_1 sampling problem, and in Section 6.5 we consider the problem of L_1 Estimation. For these Sections 6.2 to 6.5, which all concern the L_1 norm, we will consider only the L_1 α-property, and thus drop the L_1 in these sections for simplicity.

Next, in Section 6.6, we consider the problem of L_0, or distinct elements estimation, and in Section 6.7 we consider the problem of L_0, or support, sampling. For these latter two sections, namely Sections 6.6 and 6.7, we will consider only the L_0 α-property, and similarly drop the prefix of L_0 there when referring to the α property.

6.2 Frequency Estimation via Sampling

In this section, we will develop many of the tools needed for answering approximate queries about α property streams. Primarily, we develop a data structure CSSamplingSimulator, inspired by the classic count-sketch of [CCFC02b] (Section 2.3.2), that computes frequency estimates of items in the stream by sampling. This data structure will immediately result in an

improved heavy hitters algorithm in Section 6.3, and is at the heart of our L_1 sampler in Section 6.4. Recall that in this section, we will write α-property to refer to the L_1 α-property.

Firstly, for α-property streams, the following observation is crucial to many of our results. Given a fixed item $i \in [n]$, by sampling at least $\mathrm{poly}(\alpha/\epsilon)$ stream updates we can preserve x_i (after scaling) up to an additive $\epsilon\|x\|_1$ error.

Lemma 6.2.1 (Sampling Lemma)**.** *Let x be the frequency vector of a general turnstile stream with the α-property, and let x^* be the frequency vector of a uniformly sampled substream scaled up by $\frac{1}{p}$, where each update is sampled uniformly with probability $p > \alpha^2\epsilon^{-3}\log(\delta^{-1})/m$. Then with probability at least $1-\delta$ for $i \in [n]$, we have*

$$|x_i^* - x_i| < \epsilon\|x\|_1$$

Moreover, we have $\sum_{i=1}^n x_i^* = \sum_{i=1}^n x_i \pm \epsilon\|x\|_1$.

Proof. Assume we sample each update to the stream independently with probability p. Let x_i^+, x_i^- be the number of insertions and deletions of element i respectively, so $x_i = x_i^+ - x_i^-$. Let X_j^+ indicate that the j-th insertion to item i is sampled. First, if $\epsilon\|x\|_1 < x_i^+$ then by Chernoff bounds:

$$\begin{aligned}\mathbf{Pr}\left[\left|\frac{1}{p}\sum_{j=1}^{x_i^+} X_j^+ - x_i^+\right| \geq \epsilon\|x\|_1\right] &\leq 2\exp\Big(\frac{-px_i^+(\epsilon\|x\|_1)^2}{3(x_i^+)^2}\Big) \\ &\leq \exp\Big(-p\epsilon^3 m/\alpha^2\Big)\end{aligned} \tag{6.1}$$

where the last inequality holds because $x_i^+ \leq m \leq \alpha\|x\|_1$. Taking $p \geq \alpha^2\log(1/\delta)/(\epsilon^3 m)$ gives the desired probability δ. Now if $\epsilon\|x\|_1 \geq x_i^+$, then

$$\begin{aligned}\mathbf{Pr}\left[\frac{1}{p}\sum_{j=1}^{x_i^+} X_j^+ \geq x_i^+ + \epsilon\|x\|_1\right] &\leq \exp\Big(-\frac{px_i^+\epsilon\|x\|_1}{x_i^+}\Big) \\ &\leq \exp\Big(-\frac{p\epsilon m}{\alpha}\Big) \\ &\leq \delta\end{aligned} \tag{6.2}$$

for the same value of p. Applying the same argument to x_i^-, we obtain

$$x_i^* = x_i^+ - x_i^- \pm 2\epsilon\|x\|_1 = x_i \pm 2\epsilon\|x\|_1$$

as needed after rescaling ϵ. For the final statement, we can consider all updates to the stream as being made to a single element i, and then simply apply the same argument given above. □

6.2.1 Count-Sketch Sampling Simulator

In this section, we will extensively use concepts relating to the count-sketch data structure $\mathbf{A}$, see Section 2.3.2 for an introduction. Our algorithm $\mathrm{CSSamplingSimulator}$ will attempt to simulate running count-sketch on a uniform sample of the stream of size $\mathrm{poly}(\alpha \log(n)/\epsilon)$. The full data structure is given in Figure 6.2. Note that for a fixed row of the count-sketch table $\mathbf{A}$, each update is chosen with probability at least

$$2^{-p} \geq \frac{S}{2m} = \Omega\left(\frac{\alpha^2 T^2 \log(n)}{\epsilon^2 m}\right)$$

where $S = \Theta(\frac{\alpha^2}{\epsilon^2} T^2 \log(n))$, is as defined in Figure 6.2. We will use this fact to apply Lemma 6.2.1 with $\epsilon' = (\epsilon/T)$ and $\delta = 1/\mathrm{poly}(n)$. The parameter T will be $\mathrm{poly}(\log(n)/\epsilon)$, and we introduce it as a new symbol purely for clarity.

Now the updates to each row in $\mathrm{CSSamplingSimulator}$ are sampled independently from the other rows, thus $\mathrm{CSSamplingSimulator}$ may not represent running count-sketch on a single valid sample. However, each row *independently* contains the result of running a row of count-sketch on a valid sample of the stream. Since the count-sketch guarantee from Theorem 9 holds with probability $2/3$ for each row, and we simply take the median of $O(\log n)$ rows to obtain high probability, it follows that the output of $\mathrm{CSSamplingSimulator}$ will still satisfy an additive error bound w.h.p. if each row also satisfies that error bound with probability $2/3$.

By Lemma 6.2.1 with sensitivity parameter (ϵ/T), we know that we preserve the weight of all items x_i with $|x_i| \geq 1/T\|x\|_1$ up to a $(1 \pm \epsilon)$ factor w.h.p. after sampling and rescaling. For all smaller elements, however, we obtain error additive in $\epsilon\|x\|_1/T$. This gives rise to the natural division of the coordinates of x. Let $big \subset [n]$ be the set of i with $|x_i| \geq 1/T\|x\|_1$, and let $small \subset [n]$ be the complement. Let $\mathcal{E} \subset [n]$ be the top k heaviest coordinates in f, and let $s \in \mathbb{R}^n$ be a fixed sample vector of f after rescaling by p^{-1}.We have

$$\mathbf{Err}\ s2 \leq \|s_{big \setminus \mathcal{E}}\|_2 + \|s_{small}\|_2$$

CSSamplingSimulator

Input: sensitivity parameters $k \geq 1,\ \epsilon \in (0,1)$.

1. Set $S = \Theta(\frac{\alpha^2}{\epsilon^2} T^2 \log(n))$, where $T \geq 4/\epsilon^2 + \log(n)$.

2. Instantiate $d \times 6k$ count-sketch table $\mathbf{A}$ (Section 2.3.2), for $d = O(\log n)$. For each table entry $\mathbf{A}_{ij} \in A$, store two values $\mathbf{A}^+_{ij}$ and $\mathbf{A}^-_{ij}$, both initialized to 0.

3. Select 4-wise independent hash functions $h_i : [n] \to [6k],\ g_i : [n] \to \{1, -1\}$, for $i \in [d]$ as needed for count-sketch

4. Set $p \leftarrow 0$, and start $\log(n)$ bit counter to store the position in the stream.

5. **On Update** (i_t, Δ_t)**:**
   (a) **if** $t = 2^r \log(S) + 1$ for any $r \geq 1$, then for every entry $\mathbf{A}_{ij}$ set
   $$\mathbf{A}^+_{ij} \leftarrow \text{Bin}(\mathbf{A}^+_{ij}, 1/2), \quad \mathbf{A}^-_{ij} \leftarrow \text{Bin}(\mathbf{A}^-_{ij}, 1/2)$$
   and $p \leftarrow p + 1$

   (b) **On Update** (i_t, Δ_t)**:** Sample (i_t, Δ_t) with probability 2^{-p}. If sampled, then for $i \in [d]$
      i. **if** $\Delta_t g_i(i_t) > 0$, set $\mathbf{A}^+_{i,h_i(i_t)} \leftarrow \mathbf{A}^+_{i,h_i(i_t)} + \Delta_t g_i(i_t)$.

      ii. **else** set $\mathbf{A}^-_{i,h_i(i_t)} \leftarrow \mathbf{A}^-_{ih_i(i_t)} + |\Delta_t g_i(i_t)|$

6. **On Query for** x_j**:** return
   $$y^*_j = \underset{i \in [d]}{\text{median}} \left\{ 2^p \cdot g_i(j) \cdot (\mathbf{A}^+_{i,h_i(j)} - \mathbf{A}^-_{i,h_i(j)}) \mid i \in [d] \right\}$$

Figure 6.2: Our data structure to simulate running count-sketch on a uniform sample of the stream.

Furthermore, by Lemma 6.2.1, we may bound

$$\|s_{big\setminus\mathcal{E}}\|_2 \leq (1+\epsilon)\|x_{big\setminus\mathcal{E}}\|_2 \leq (1+\epsilon)\,\mathbf{Err}\,x2$$

So it remains to upper bound $\|s_{small}\|_2^2$, which we do in the following technical lemma.

The intuition for the Lemma is that $\|x_{small}\|_2$ is maximized when all the L_1 weight is concentrated in T elements, thus $\|x_{small}\|_2 \leq (T(\|x\|_1/T)^2)^{1/2} = \|x\|_1/T^{1/2}$. By the α property, we know that the number of insertions made to the elements of $small$ is bounded by $\alpha\|x\|_1$. Thus, computing the variance of $\|s_{small}\|_2^2$ and applying Bernstein's inequality, we obtain a similar upper bound for $\|s_{small}\|_2$.

Lemma 6.2.2. *If s is the rescaled frequency vector resulting from uniformly sampling with probability $p \geq S/(2m)$, where $S = \Omega(\alpha^2 T^2\ \log(n))$ for $T = \Omega(\log(n))$, of a general turnstile stream f with the α property, then we have $\|s_{small}\|_2 \leq 2T^{-1/2}\|x\|_1$ with high probability.*

Proof. Fix $x \in \mathbb{R}^n$, and assume that $\|x\|_1 > S$ (otherwise we could just sample all the updates and our counters in count-sketch would never exceed $\log(\alpha S)$). For any $i \in [n]$, let $x_i = x_i^+ - x_i^-$, so that $x = x^+ - x^-$, and let s_i be our rescaled sample of x_i. By definition, for each $i \in small$ we have $x_i < \frac{1}{T}\|x\|_1$. Thus the quantity $\|x_{small}\|_2^2$ is maximized by having T coordinates equal to $\frac{1}{T}\|x\|_1$. Thus $\|x_{small}\|_2^2 \leq T(\frac{\|x\|_1}{T})^2 \leq \frac{\|x\|_1^2}{T}$. Now note that if we condition on the fact that $s_i \leq 2/T\|x\|$ for all $i \in small$, which occurs with probability greater than $1 - n^{-5}$ by Lemma 6.2.1, then since $\mathbf{E}\left[\sum_i |s_i|^4\right] = O(n^4)$, all of the following expectations change by a factor of at most $(1 \pm 1/n)$ by conditioning on this fact. Thus we can safely ignore this conditioning and proceed by analyzing $\mathbf{E}\left[\|s_{small}\|_2^2\right]$ and $\mathbf{E}\left[\|s_{small}\|_4^4\right]$ without this condition, but use the conditioning when applying Bernstein's inequality later.

We have that $|s_i| = |\frac{1}{p}\sum_{j=1}^{x_i^+ + x_i^-} X_{ij}|$, where X_{ij} is an indicator random variable that is ± 1 if the j-th update to x_i is sampled, and 0 otherwise, where the sign depends on whether or not the update was an insertion or deletion and $p \geq S/(2m)$ is the probability of sampling an update.

Then $\mathbf{E}\left[|s_i|\right] = \mathbf{E}\left[|\frac{1}{p}\sum_{j=1}^{x_i^+ + x_i^-} X_{ij}|\right] = |x_i|$. Furthermore, we have

$$\begin{aligned}\mathbf{E}\left[s_i^2\right] &= \frac{1}{p^2}\,\mathbf{E}\left[(\sum_{j=1}^{x_i^+ + x_i^-} X_{ij})^2\right] \\ &= \frac{1}{p^2}(\sum_{j=1}^{x_i^+ + x_i^-} \mathbf{E}\left[X_{ij}^2\right] + \sum_{j_1 \neq j_2} \mathbf{E}\left[X_{ij_1}X_{ij_2}\right]) \\ &= \frac{1}{p}(x_i^+ + x_i^-) + ((x_i^+)(x_i^+ - 1) + (x_i^-)(x_i^- - 1) - 2x_i^+x_i^-)\end{aligned} \tag{6.3}$$

Substituting $x_i^- = x_i^+ - x_i$ in part of the above equation gives

$$\begin{aligned}\mathbf{E}\left[s_i^2\right] &= \frac{1}{p}(x_i^+ + x_i^-) + x_i^2 + x_i - 2x_i^+ \\ &\leq \frac{1}{p}(x_i^+ + x_i^-) + 2x_i^2\end{aligned} \tag{6.4}$$

So

$$\mathbf{E}\left[\sum_{i \in small} s_i^2\right] \leq \frac{1}{p}(\|x_{small}^+\|_1 + \|x_{small}^-\|_1) + 2\|x_{small}\|_2^2$$

which is at most $\frac{\alpha}{p}\|x\|_1 + 2\frac{\|x\|_1^2}{T}$ by the α-property of x and the upper bound on $\|x_{small}\|_2^2$. Now $\mathbf{E}\left[s_i^4\right] = \frac{1}{p^4}\,\mathbf{E}\left[(\sum_{j=1}^{x_i^+ + x_i^-} X_{ij})^4\right]$, which we can write as

$$\mathbf{E}\left[\sum_j X_{ij}^4 + 4\sum_{j_1 \neq j_2} X_{ij_1}^3 X_{ij_2} + 12\sum_{\substack{j_1,j_2,j_3 \\ \text{distinct}}} X_{ij_1}^2 X_{ij_2}X_{ij_3} + \sum_{\substack{j_1,j_2,j_3,j_4 \\ \text{distinct}}} X_{ij_1}X_{ij_2}X_{ij_3}X_{ij_4}\right]\frac{1}{p^4} \tag{6.5}$$

We analyze Equation 6.5 term by term. First note that

$$\mathbf{E}\left[\sum_j X_{ij}^4\right] = p(x_i^+ + x_i^-)$$

and

$$\mathbf{E}\left[\sum_{j_1 \neq j_2} X_{ij_1}^3 X_{ij_2}\right] = p^2\Big((x_i^+)(x_i^+ - 1) + (x_i^-)(x_i^- - 1) - 2(x_i^+x_i^-)\Big)$$

Substituting $x_i^- = x_i^+ - x_i$, we obtain $\mathbf{E}\left[\sum_{j_1 \neq j_2} X_{ij_1}^3 X_{ij_2}\right] \leq 2p^2x_i^2$. Now for the third term we

have

$$\mathbf{E}\left[\sum_{j_1\neq j_2\neq j_3} X_{ij_1}^2 X_{ij_2} X_{ij_3}\right] = p^3\Big((x_i^+)(x_i^+-1)(x_i^+-2) + (x_i^-)(x_i^--1)(x_i^--2) + x_i^+(x_i^-)(x_i^--1) + x_i^-(x_i^+)(x_i^+-1) - 2(x_i^+x_i^-)(x_i^+-1) - 2(x_i^+x_i^-)(x_i^--1)\Big) \tag{6.6}$$

which after the same substitution is upper bounded by

$$10p^3 \max\{x_i^+, |x_i|\}x_i^2 \leq 10p^3\alpha\|x\|_1 x_i^2$$

where the last inequality follows from the α-property of f. Finally, the last term is

$$\mathbf{E}\left[\sum_{j_1\neq j_2\neq j_3\neq j_4} X_{ij_1}X_{ij_2}X_{ij_3}X_{ij_4}\right] = p^4\Big(x_i^+(x_i^+-1)(x_i^+-2)(x_i^+-3) + x_i^-(x_i^--1)(x_i^--2)(x_i^--3) + 6(x_i^+(x_i^+-1))(x_i^-(x_i^--1)) - 4\Big(x_i^+(x_i^+-1)(x_i^+-2)x_i^- + x_i^-(x_i^--1)(x_i^--2)x_i^+\Big)\Big) \tag{6.7}$$

Making the same substitution allows us to bound this above by

$$p^4(24x_i^4 - 12x_ix_i^+ + 12(x_i^+)^2) \leq p^4(36(x_i)^4 + 24(x_i^+)^2)$$

Now x satisfies the α property. Thus $\|x_{small}^+\|_1 + |x_{small}^-\|_1 \leq \alpha\|x\|_1$, so summing the above bounds over all $i \in small$ we obtain

$$\mathbf{E}\left[\frac{1}{p^4}\|s_{small}\|_4^4\right] \leq \frac{1}{p^3}\alpha\|x\|_1 + \frac{8}{p^2}\|x_{small}\|_2^2$$

$$+\frac{120\alpha}{p}\|x_{small}\|_2^2\,|f\|_1 + 36\|x_{small}\|_4^4 + 24\alpha^2\|x\|_1^2$$

Now $1/p \leq \frac{2m}{S} \leq \frac{2\alpha}{S}\|x\|_1$ by the α-property. Applying this with the fact that $\|x_{small}\|_2^2$ is maximized at $\frac{\|x\|_1^2}{T}$, and similarly $\|x_{small}\|_2^2$ is maximized at $T(\|x\|_1/T)^4 = \|x\|_1^4/T^3$ we have that $\mathbf{E}\left[\frac{1}{p^4}\|s_{small}\|_4^4\right]$ is at most

$$\frac{8\alpha^4}{S^3}\|x\|_1^4 + \frac{36\alpha^2}{S^2T}\|x\|_1^4 + \frac{240\alpha^2}{ST}\|x\|_1^4 + \frac{36}{T^3}\|x\|_1^4 + 24\alpha^2\|x\|_1^2$$

Since we have $\|x\|_1 > S > \alpha^2 T^2$, the above expectation is upper bounded by $\frac{300}{T^3}\|x\|_1^4$. We now apply Bernstein's inequality. Using the fact that we conditioned on earlier, we have the upper bound $\frac{1}{p}s_i \leq 2\|x\|_1/T$, so plugging this into Bernstein's inequality yields:

$$\begin{aligned}\mathbf{Pr}\left[\left|\|s_{small}\|_2^2 - \mathbf{E}\left[\|s_{small}\|_2^2\right]\right| > \frac{\|x\|_1^2}{T}\right] &\leq \exp\Big(-\frac{\|x\|_1^4/(2T^2)}{300\|x\|_1^4/T^3 + 2\|x\|_1^3/(3T^2)}\Big) \\ &\leq \exp\Big(-T/604\Big)\end{aligned} \tag{6.8}$$

Finally, $T = \Omega(\log(n))$, so the above probability is poly$(\frac{1}{n})$ for $T = c\log(n)$ and a sufficiently large constant c. Since the expectation $\mathbf{E}\left[\|s_{small}\|_2^2\right]$ is at most

$$\frac{\alpha}{p}\|x\|_1 + 2\|x\|_1^2/T \leq 3\|x\|_1^2/T$$

it follows that $\|s_{small}\|_2 \leq \frac{2\|x\|_1}{\sqrt{T}}$ with high probability, which is the desired result. □

Applying the result of Lemma 6.2.2, along with the bound on $\mathbf{Err}\, s2$ from the previous paragraphs, we obtain the following corollary.

Corollary 6.2.3. *With high probability, if s is as in Lemma 6.2.2, then*

$$\mathbf{Err}\, s2 \leq (1+\epsilon)\,\mathbf{Err}\, x2 + 2T^{-1/2}\|x\|_1$$

Now we analyze the error from CSSamplingSimulator. Observe that each row in $\mathbf{A}$ contains the result of hashing a uniform sample into $6k$ buckets. Let $s^i \in \mathbb{R}^n$ be the frequency vector, *after scaling* by $1/p$, of the sample hashed into the i-th row of CSSamplingSimulator, and let $y^i \in \mathbb{R}^n$ be the estimate of s^i taken from the i-th row of CSSamplingSimulator. Let $\sigma(i) : n \to [O(\log(n))]$ be the row from which count-sketch returns its estimate for x_i, meaning $y_i^* = y_i^{\sigma(i)}$.

Theorem 46. *For $\epsilon > 0, k > 1$, with high probability, when run on a general turnstile stream $x \in \mathbb{R}^n$ with the α property,* CSSamplingSimulator *with $6k$ columns, $O(\log(n))$ rows, returns y^* such that, for every $i \in [n]$ we have*

$$|y_i^* - x_i| \leq 2\left(\frac{1}{k^{1/2}}\,\mathbf{Err}\, x2 + \epsilon\|x\|_1\right)$$

It follows that if $\hat{y} = y^* - y^*_{\text{tail}(k)}$*, then*

$$\mathbf{Err}\, x2 \leq \|x - \hat{y}\|_2 \leq 5(k^{1/2}\epsilon\|x\|_1 + \mathbf{Err}\, x2)$$

with high probability. The space required is $O\left(k \log n \log(\frac{\alpha \log n}{\epsilon}\right)$.

Proof. Set $S = \Theta(\frac{\alpha^2}{\epsilon^2}T^2 \log(n))$ as in Figure 6.2, and set $T = 4/\epsilon^2 + O(\log(n))$. Fix any $i \in [n]$. Now CSSamplingSimulator samples updates uniformly with probability $p > S/(2m)$, so applying Lemma 6.2.1 to our sample s_i^j of x_i for each row j and union bounding, with high probability we have $s_i^j = x_i \pm \frac{\epsilon}{T}\|x\|_1 = s_i^q$ for all rows j, q. Then by the count-sketch guarantee (Theorem 9), for each row q of the table $\mathbf{A}$, we have

$$y_i^q = x_i \pm \left(\frac{\epsilon}{T}\|x\|_1 + k^{-1/2} \max_j \mathbf{Err}\, s^j 2\right)$$

with probability $2/3$. Thus

$$y_i^* = y_i^{\sigma(i)} = x_i \pm \left(\frac{\epsilon}{T}\|x\|_1 + k^{-1/2} \max_j \mathbf{Err}\, s^j 2\right)$$

with high probability. Now noting that $\sqrt{T} \geq 2/\epsilon$, we apply Corollary 6.2.3 to $\mathbf{Err}\, s^j 2$ and union bound over all $j \in [O(\log(n))]$ to obtain

$$\max_j \mathbf{Err}\, s^j 2 \leq (1 + \epsilon)\, \mathbf{Err}\, x2 + \epsilon\|x\|_1$$

w.h.p., and union bounding over all $i \in [n]$ gives

$$|y_i^* - x_i| \leq 2\left(\frac{1}{k^{1/2}}\, \mathbf{Err}\, x2 + \epsilon\|x\|_1\right)$$

for all $i \in [n]$, again with high probability.

For the second claim, note that $\mathbf{Err}\, x2 \leq \|x - z\|_2$ for any k-sparse vector z, from which the lower bound on $\|x - \hat{y}\|_2$ follows. For the upper bound, note that if the top k largest coordinates are the same in y^* as they are in x, then $\|x - \hat{y}\|_2$ is at most $\mathbf{Err}\, x2$ plus the L_2 error from estimating the top k elements, which is at most

$$\left(4k(\epsilon\|x\|_1 + k^{-1/2}\, \mathbf{Err}\, x2)^2\right)^{1/2} \leq 2(k^{1/2}\epsilon\|x\|_1 + \mathbf{Err}\, x2)$$

In the worst case the top k coordinates of y^* are disjoint from the top k in x. Applying the

triangle inequality, the error is at most the error on the top k coordinates of y^* plus the error on the top k in x. Thus $\|x - \hat{y}\|_2 \leq 5(k^{1/2}\epsilon\|x\|_1 + \mathbf{Err}\, x2)$ as required.

For the space bound, note that the count-sketch table $\mathbf{A}$ has $O(k\log(n))$ entries, each of which stores two counters which together hold $O(S)$ samples in expectation. So the magnitude of the counters never exceed $\text{poly}(S) = \text{poly}(\frac{\alpha}{\epsilon}\log(n))$ w.h.p. by Chernoff bounds, and so can be stored using $O(\log(\frac{\alpha \log n}{\epsilon})$ bits each (we can simply terminate if a counter gets too large). □

We now address how the error term of Theorem 46 can be estimated so as to bound the potential error. This will be necessary for our L_1 sampling algorithm. We first state the following well known fact about norm estimation [TZ], and give a proof for completeness.

Lemma 6.2.4. *Let $R \in \mathbb{R}^k$ be any row $\mathbf{A}_{i,*}$ of a count-sketch matrix $\mathbf{A}$ with k columns, run on a stream with frequency vector x. Then with probability $99/100$, we have*

$$\sum_{i=1}^{k} R_i^2 = (1 \pm O(k^{-1/2}))\|x\|_2^2$$

Proof. Let $\mathbf{1}(E)$ be the indicator function that is 1 if the event E is true, and 0 otherwise. Let $h : [n] \to [k]$ and $g : [n] \to \{0, 1\}$ be 4-wise independent hash functions which specify the row of count-sketch. Then

$$\begin{aligned}\mathbf{E}\left[\sum_{i=1}^{k} R_i^2\right] &= \mathbf{E}\left[\sum_{j=1}^{k}\left(\sum_{i=1}^{n}\mathbf{1}(h(i)=j)g(i)x_i\right)^2\right] \\ &= \mathbf{E}\left[\sum_{i=1}^{n} x_i^2\right] + \mathbf{E}\left[\sum_{j=1}^{k}\left(\sum_{i_1\neq i_2}\mathbf{1}(h(i_1)=j)\mathbf{1}(h(i_2)=j)g(i_1)g(i_2)x_{i_1}x_{i_2}\right)^2\right]\end{aligned} \tag{6.9}$$

By the 2-wise independence of g, the second quantity is 0, and we are left with $\|x\|_2^2$. By a similar computation using the full 4-wise independence, we can show that

$$\mathbf{Var}\left(\sum_{i=1}^{k} R_i^2\right) = 2\frac{\|x\|_2^4 - \|x\|_4^4}{k} \leq \frac{2}{k}\|x\|_2^4$$

Then by Chebyshev's inequality, we obtain

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{k} R_i^2 - \|x\|_2^2\right| > 10\sqrt{\frac{2}{k}}\|x\|_2^2\right] < \frac{1}{100}$$

as needed. □

Lemma 6.2.5. *For $k > 1, \epsilon \in (0,1)$, given a α-property stream x, there is an algorithm that can produce an estimate v such that*

$$\mathbf{Err}\, x2 \leq v \leq 45k^{1/2}\epsilon\|x\|_1 + 20\,\mathbf{Err}\, x2$$

with high probability. The space required is the space needed to run two instances of the data structure CSSamplingSimulator *with parameters k, ϵ.*

Proof. By Lemma 6.2.4, the L_2 of row i of CSSamplingSimulator with a constant number of columns will be a $(1 \pm 1/2)$ approximation of $\|s^i\|_2$ with probability $99/100$, where s^i is the scaled up sampled vector corresponding to row i. Our algorithm is to run two copies of CSSamplingSimulator on the entire stream, say $\texttt{CSSS}_1$ and $\texttt{CSSS}_2$, with k columns and sensitivity ϵ. At the end of the stream we compute y^* and $\hat{y}$ from $\texttt{CSSS}_1$ where $\hat{y} = y^* - y^*_{\text{tail}(k)}$ is the best k-sparse approximation to y^*. We then feed $-\hat{y}$ into $\texttt{CSSS}_2$. The resulting L_2 norm of the i-th row of $\texttt{CSSS}_2$ is $(1 \pm 1/2)\|s^i_2 - \hat{y}\|_2$ (after rescaling of s^i) with probability $99/100$, where s^i_2 is the sample corresponding to the i-th row of $\texttt{CSSS}_2$.

Now let $T = 4/\epsilon^2$ as in Theorem 46, and let s^i be the sample corresponding to the i-th row of $\texttt{CSSS}_1$. Then for any i, $\|s^i_{small}\|_2 \leq 2T^{-1/2}\|x\|_1$ w.h.p. by Lemma 6.2.2, and the fact that $\|x_{small}\|_2 \leq T^{-1/2}\|x\|_1$ follows from the definition of $small$.

Furthermore, by Lemma 6.2.1, we know w.h.p. that $|s^i_j - x_j| < \epsilon|x_j|$ for all $j \in big$, and thus $\|s^i_{big} - x_{big}\|_2 \leq \epsilon\|x\|_1$. Then by the reverse triangle inequality we have

$$\begin{aligned} \left|\|s^i - \hat{y}\|_2 - \|x - \hat{y}\|_2\right| &\leq \|s^i - f\|_2 \\ &\leq \|s^i_{small} - x_{small}\|_2 + \|s^i_{big} - x_{big}\|_2 \\ &\leq 5\epsilon\|x\|_1 \end{aligned} \tag{6.10}$$

Thus

$$\|s^i - \hat{y}\|_2 = \|x - \hat{y}\|_2 \pm 5\epsilon\|x\|_1$$

for all rows i w.h.p, so the L_2 of the i-th row of $\texttt{CSSS}_2$ at the end of the algorithm is the value v_i such that

$$\frac{1}{2}(\|x - \hat{y}\|_2 - 5\epsilon\|x\|_1) \leq v_i \leq 2(\|x - \hat{y}\|_2 + 5\epsilon\|x\|_1)$$

with probability $9/10$ (note that the bounds do not depend on i). Taking $v = 2\cdot\texttt{median}(v_1, \ldots, v_{O(\log(n))}) + 5\epsilon\|x\|_1$, it follows that

$$\|x - \hat{y}\|_2 \leq v \leq 4\|x - \hat{y}\|_2 + 25\epsilon\|x\|_1$$

with high probability. Applying the upper and lower bounds on $\|x - \hat{y}\|_2$ given in Theorem 46 yields the desired result. The space required is the space needed to run two instances of CSSamplingSimulator, as stated. □

6.2.2 Inner Products

Given two vectors $f, g \in \mathbb{R}^n$ which are each updated in a stream, the *inner-product estimation problem* is to estimate their product

$$\langle f, g \rangle = \sum_{i=1}^{n} f_i g_i$$

Inner product estimation has been studied for estimating the size of join and self-join relations for databases [AGMS99, RD08, RD07]. For unbounded deletion streams, to obtain an $\epsilon \|f\|_1 \|g\|_1$-additive error approximation the best known result requires $O(\epsilon^{-1} \log(n))$ bits with a constant probability of success [CM05]. We show in Theorem 69 that $\Omega(\epsilon^{-1} \log(\alpha))$ bits are required even when f, g are *strong* α-property streams. This also gives a matching $\Omega(\epsilon^{-1} \log(n))$ lower bound for the unbounded deletion case

In Theorem 47, we give a matching upper bound for α-property streams up to $\log \log(n)$ and $\log(1/\epsilon)$ terms. We first prove the following technical Lemma, which shows that inner products are preserved under a sufficiently large sample.

Lemma 6.2.6. *Let $f, g \in \mathbb{R}^n$ be two α-property streams with lengths m_f, m_g respectively. Let $f', g' \in \mathbb{R}^n$ be unscaled uniform samples of f and g, sampled with probability $p_f \geq s/m_f$ and $p_g \geq s/m_g$ respectively, where $s = \Omega(\alpha^2/\epsilon^2)$. Then with probability $99/100$, we have*

$$\langle p_f^{-1} f', p_g^{-1} g' \rangle = \langle f, g, \rangle \pm \epsilon \|f\|_1 \|g\|_1$$

Proof. We have

$$\mathbf{E}\left[\langle p_f^{-1} f', p_g^{-1} g' \rangle\right] = \sum_i \mathbf{E}\left[p_f^{-1} f_i'\right] \mathbf{E}\left[p_g^{-1} g_i'\right] = \langle f, g \rangle$$

Now the random variables $\{f_i' g_i'\}_{i \in [n]}$'s are independent, so it suffices to compute the variance $\mathbf{Var}\left(p_f^{-1} p_g^{-1} f_i' g_i'\right)$ of a single coordinate. We can write

$$\mathbf{Var}\left(p_f^{-1} p_g^{-1} f_i' g_i'\right) = (p_f^{-1} p_g^{-1})^2 \mathbf{E}\left[(f_i')^2\right] \mathbf{E}\left[(g_i')^2\right] - (f_i g_i)^2$$

Let $X_{i,j} \in \{1, -1, 0\}$ be a signed indicator variable, indicating if the j-th update to f_i is sampled,

where the sign indicates whether the update was an insertion or deletion. Let f^+, f^- be the insertion and deletion vectors of f respectively (see Definition 6.1.1), and let $F = f^+ + f_i$. Then $f = f^+ - f^-$, and $f_i' = \sum_{j=1}^{f_i^+ + f_i^-} X_{ij}$. Similarlly define the vectors g^+, g^- and $G = g^+ + g^-$. We have

$$\begin{aligned}\mathbf{E}\left[p_f^{-2}(f_i')^2\right] &= \mathbf{E}\left[p_f^{-2}\left(\sum_{j=1}^{f_i^+ + f_i^-} X_{ij}^2 + \sum_{j_1 \neq j_2} X_{ij_1}X_{ij_2}\right)\right] \\ &\leq p_f^{-1}F_i + (f_i^+)^2 + (f_i^-)^2 - 2f_i^+ f_i^- \\ &= p_f^{-1}F_i + f_i^2\end{aligned} \tag{6.11}$$

Note that the α-property states that $m_f = \|F\|_1 \leq \alpha\|f\|_1$ and $m_g = \|G\|_1 \leq \alpha\|g\|_1$. Moreover, it gives

$$p_f^{-1} \leq \frac{m_f}{s} \leq \frac{\epsilon^2\|f\|_1}{\alpha}, \qquad \text{and} \qquad p_g^{-1} \leq \frac{m_g}{s} \leq \frac{\epsilon^2\|g\|_1}{\alpha}$$

Then

$$\begin{aligned}\mathbf{Var}\left(\langle p_f^{-1}f', p_g^{-1}g'\rangle\right) &= \sum_{i=1}^{n} \mathbf{Var}\left(p_f^{-1}p_g^{-1}f_i'g_i'\right) \\ &\leq \sum_{i=1}^{n}\left(p_g^{-1}f_i^2G_i + p_f^{-1}g_i^2F_i + p_f^{-1}p_g^{-1}F_iG_i\right)]\end{aligned} \tag{6.12}$$

We can bound the first term via:

$$\sum_{i=1}^{n} p_g^{-1}f_i^2G_i \leq p_g^{-1}\|G\|_1\|f\|_1^2 \leq \epsilon^2\|g\|_1^2\|f\|_1^2 \tag{6.13}$$

and similarly

$$\sum_{i=1}^{n} p_f^{-1}g_i^2\ F_i \leq \epsilon^2\|g\|_1^2\|f\|_1^2 \tag{6.14}$$

Finally, we can bound the last term as follows:

$$\begin{aligned}\sum_{i=1}^{n} p_f^{-1}p_g^{-1}F_iG_i &\leq p_f^{-1}p_g^{-1}\|F\|_1\|G\|_1 \\ &\leq \epsilon^4\|f\|_1^2\|g\|_1^2\end{aligned} \tag{6.15}$$

Putting together equations 6.12, 6.13, 6.14, and 6.15, we can bound the variance by $3\epsilon^2\|f\|_1^2\|g\|_1^2$.

Applying Chebyshev's inequality:

$$\mathbf{Pr}\left[\left|\langle p_f^{-1}f', p_g^{-1}g'\rangle - \langle f, g\rangle\right| > 30\epsilon\|f\|_1\|g\|_1\right] \leq 1/100$$

and rescaling ϵ by a constant gives the desired result. □

Our algorithm obtains such a sample as needed for Lemma 6.2.6 by sampling in exponentially increasing intervals. Next, we hash the universe down by a sufficiently large prime to avoid collisions in the samples, and then run a inner product estimator in the smaller universe. Note that this hashing is not pairwise independent, as that would require the space to be at least $\log n$ bits; rather the hashing just has the property that it preserves distinctness with good probability. We now prove a fact that we will need for our desired complexity.

Lemma 6.2.7. *Given a $\log(n)$ bit integer x, the value $x \pmod p$ can be computed using only $\log\log(n) + \log(p)$ bits of space.*

Proof. We initialize a counter $c \leftarrow 0$. Let $x_1, x_2, \ldots, x_{\log(n)}$ be the bits of x, where x_1 the least significant bit. Then we set $y_1 = 1$, and at every step $t \in [\log(n)]$ of our algorithm, we store y_t, y_{t-1}, where we compute y_t as $y_t = 2y_{t-1} \pmod p$. This can be done in $O(\log(p))$ space. Our algorithm then takes $\log(n)$ steps, where on the t-th step it checks if $x_t = 1$, and if so it updates $c \leftarrow c + y_t \pmod p$, and otherwise sets $c \leftarrow c$. At the end we have $c = \sum_{i=0}^{\log(n)} 2^i x_i \pmod p = x \pmod p$ as desired, and c never exceeds $2p$, and can be stored using $\log(p)$ bits of space. The only other value stored is the index $t \in [\log(n)]$, which can be stored using $\log\log(n)$ bits as stated. □

We now observe how the count-sketch data structure (Section 2.3.2) can be used to estimate inner products. To do this, we instantiate two instances of count-sketch, whose tables we will denote $\mathbf{A}, \mathbf{B}$, each with only a single row with k buckets; thus, we can think of $\mathbf{A}, \mathbf{B} \in \mathbb{R}^k$. Next, we run the first count-sketch $\mathbf{A}$ on the vector f, and run the second count-sketch on the vector g, and report as an estimate the value $\langle \mathbf{A}, \mathbf{B}\rangle$ of $\langle f, g\rangle$. In the following Lemma, we show that if we first subsample the vectors f, g of two α-property streams, before running count-sketch, then $\langle \mathbf{A}, \mathbf{B}\rangle$ is still a good approximation of $\langle f, g\rangle$ after rescaling $\mathbf{A}$ and $\mathbf{B}$ by the inverse of the sampling probability.

Lemma 6.2.8. *Let f', g' be uniformly sampled rescaled vectors of general-turnstile α-property streams f, g with lengths m_f, m_g and sampling probabilities p_f, p_g respectively. Suppose that $p_f \geq s/m_f$ and $p_g \geq s/m_g$, where $s = \Omega(\alpha^2 \log^7(n) T^2 \epsilon^{-10})$. Then let $\mathbf{A} \in \mathbb{R}^k$ be a single row*

of count-sketch run on f', and let $\mathbf{B} \in \mathbb{R}^k$ be a single row of count-sketch run on g', where $\mathbf{A}, \mathbf{B}$ share the same hash function h and $k = \Theta(1/\epsilon)$ Then

$$\sum_{i=1}^{k} \mathbf{A}_i \mathbf{B}_i = \langle f, g \rangle \pm \epsilon \|f\|_1 \|g\|_1$$

with probability $11/13$.

Proof. Set $k = 100^2/\epsilon$. Let Y_{ij} indicate $h(i) = h(j)$, and let $X_{ij} = \sigma_+ i \sigma_+ j f'_i g'_j Y_{ij}$. We have

$$\sum_{i=1}^{k} \mathbf{A}_i \mathbf{B}_i = \langle f', g' \rangle + \sum_{i \neq j} X_{ij}$$

Let $\epsilon_0 = \Theta(\log^2(n)/\epsilon^3)$ and let $T = \Theta(\log(n)/\epsilon^2)$, so we can write $s = \Omega(\alpha^2 \log(n) T^2/\epsilon_0^2)$ (this will align our notation with that of Lemma 6.2.2). Define the sets

$$\mathcal{F} = \{i \in [n] \mid |f_i| \geq \|f\|_1/T\}, \qquad \text{and} \qquad \mathcal{G} = \{i \in [n] \mid |g_i| \geq \|g\|_1/T\}$$

We first bound the term

$$|\sum_{i \neq j, (i,j) \in \mathcal{F} \times \mathcal{G}} X_{ij}| \leq \sum_{(i,j) \in \mathcal{F} \times \mathcal{G}} Y_{ij} |f'_i g'_j|$$

Now by Lemma 6.2.1 and union bounding over all $i \in [n]$, we have

$$\|f' - f\|_\infty \leq \epsilon_0 \|f\|_1/T, \qquad \text{and} \qquad \|g' - g\|_\infty \leq \epsilon_0 \|g\|_1/T$$

with high probability, and we condition on this now. Thus for every $(i,j) \in \mathcal{F} \times \mathcal{G}$, we have $f'_i = (1 \pm \epsilon_0) f_i$ and $g'_j = (1 \pm \epsilon_0) g_j$. It follows that

$$\sum_{(i,j) \in \mathcal{F} \times \mathcal{G}} Y_{ij} |f'_i g'_j| \leq 2 \sum_{(i,j) \in \mathcal{F} \times \mathcal{G}} Y_{ij} |f_i| |g_j|$$

where we used that $\epsilon_0 < 1/4$. Since $\mathbf{E}[Y_{ij}] = 1/k$, we have

$$\mathbf{E}\left[\sum_{(i,j) \in \mathcal{F} \times \mathcal{G}} Y_{ij} |f_i| |g_j|\right] \leq \frac{1}{k} \|f\|_1 \|g\|_1$$

By Markov's inequality with $k = 100^2/\epsilon$, it follows that

$$\left| \sum_{i \neq j, (i,j) \in \mathcal{F} \times \mathcal{G}} X_{ij} \right| \leq 2 \sum_{(i,j) \in \mathcal{F} \times \mathcal{G}} Y_{ij} |f_i||g_j| \leq \epsilon \|f\|_1 \|g\|_1$$

with probability greater than $1 - (1/1000 + O(n^{-c})) > 99/100$, where the $O(n^{-c})$ comes from conditioning on the high probability events from Lemma 6.2.1. Call this event $\mathcal{E}_1$.

Now define the complement sets $\mathcal{F}^C = [n] \setminus \mathcal{F}$ and $\mathcal{G}^C = [n] \setminus \mathcal{G}$, and let $\mathbf{A} \subset [n]^2$ be the set of all (i, j) with $i \neq j$ and such that either $i \in \mathcal{F}^C$ or $j \in \mathcal{G}^C$. We now consider the variables $\{|X_{ij}|\}_{i<j}$, and let X_{ij}, X_{pq} be two such distinct variables. Then

$$\mathbf{E}\left[X_{ij} X_{pq}\right] = \mathbf{E}\left[f_i' g_i' f_p' g_q' Y_{ij} Y_{pq} \sigma_+ i \sigma_+ j \sigma_+ p \sigma_+ q\right]$$

Note that the variables σ are independent from $Y_{ij} Y_{pq}$, which are determined by h. Since $i < j$ and $p < q$ and $(i, j) \neq (p, q)$, it follows that one of i, j, p, q is unique among them. WLOG it is i, so by 4-wise independence of σ we have

$$\begin{aligned}
\mathbf{E}\left[f_i' g_i' f_p' g_q' Y_{ij} Y_{pq} \sigma_+ i \sigma_+ j \sigma_+ p \sigma_+ q\right] &= \mathbf{E}\left[f_i' g_i' f_p' g_q' Y_{ij} Y_{pq} \sigma_+ j \sigma_+ p \sigma_+ q\right] \mathbf{E}\left[\sigma_+ i\right] \\
&= 0 \\
&= \mathbf{E}\left[X_{ij}\right] \mathbf{E}\left[X_{pq}\right]
\end{aligned}$$

Thus the variables $\{|X_{ij}|\}_{i<j}$ (and symmetrically $\{|X_{ij}|\}_{i>j}$) are uncorrelated so

$$\mathbf{Var}\left(\sum_{i<j, (i,j) \in \mathcal{A}} X_{ij} \right) = \sum_{i<j, (i,j) \in \mathcal{A}} \mathbf{Var}\left(X_{ij}\right) \leq \sum_{(i,j) \in \mathcal{A}} (f_i' g_j')^2 / k$$

Since $\mathbf{E}\left[\sum_{i<j, (i,j) \in \mathcal{A}} X_{ij}\right] = 0$, by Chebyshev's inequality with $k = 100^2/\epsilon$, we have

$$\left| \sum_{i<j, (i,j) \in \mathcal{A}} X_{ij} \right| \leq (\epsilon \sum_{(i,j) \in \mathcal{A}} (f_i' g_j')^2)^{1/2}$$

with probability $99/100$. So by the union bound and a symmetric arguement for $j > i$, we have

$$\begin{aligned}
\left| \sum_{(i,j) \in \mathcal{A}} X_{ij} \right| &\leq \left| \sum_{i<j, (i,j) \in \mathcal{A}} X_{ij} \right| + \left| \sum_{i>j, (i,j) \in \mathcal{A}} X_{ij} \right| \\
&\leq 2(\epsilon \sum_{(i,j) \in \mathcal{A}} (f_i' g_j')^2)^{1/2}
\end{aligned}$$

with probability $1 - (2/100) = 49/50$. Call this event $\mathcal{E}_2$. We have

$$\sum_{(i,j)\in\mathcal{A}} (f'_i g'_j)^2 = \sum_{\substack{(i,j)\in\mathcal{F}\times\mathcal{G}^C \\ i\neq j}} (f'_i g'_j)^2 + \sum_{\substack{(i,j)\in\mathcal{F}^C\times\mathcal{G} \\ i\neq j}} (f'_i g'_j)^2 + \sum_{\substack{(i,j)\in\mathcal{F}^C\times\mathcal{G}^C \\ i\neq j}} (f'_i g'_j)^2$$

Now the last term is at most $\|f'_{\mathcal{F}^C}\|_2^2 \|g'_{\mathcal{G}^C}\|_2^2$, which is at most $16\|f\|_1^2\|g\|_1^2/T^2 \leq \epsilon^4\|f\|_1^2\|g\|_1^2$ w.h.p. by Lemma 6.2.2 applied to both f and g and union bounding (note that $\mathcal{F}^C, \mathcal{G}^C$ are exactly the set $small$ in Lemma 6.2.2 for their respective vectors). We hereafter condition on this w.h.p. event that $\|f'_{\mathcal{F}^C}\|_2^2 \leq 4\|f\|_1^2/T$ and $\|g'_{\mathcal{G}^C}\|_2^2 \leq 4\|g\|_1^2/T$ (note $T > 1/\epsilon^2$).

Now as noted earlier, w.h.p. we have $f'_i = (1 \pm \epsilon_0) f_i$ and $f'_j = (1 \pm \epsilon_0) f_j$ for $i \in \mathcal{F}$ and $j \in \mathcal{G}$. Thus $\|f'_{\mathcal{F}}\|_2^2 = (1 \pm O(\epsilon_0))\|f_{\mathcal{F}}\|_2^2 < 2\|f\|_1^2$ and $\|g'_{\mathcal{G}}\|_2^2 \leq 2\|g\|_1^2$. We have

$$\begin{aligned}\sum_{\substack{(i,j)\in\mathcal{F}\times\mathcal{G}^C \\ i\neq j}} (f'_i g'_j)^2 &\leq \sum_{i\in\mathcal{F}} (f'_i)^2 \|g'_{\mathcal{G}^c}\|_2^2 \\ &\leq O(\epsilon^2)\|f'_{\mathcal{F}}\|_2^2\|g\|_1^2 \\ &\leq O(\epsilon^2)\|f\|_1^2\|g\|_1^2\end{aligned}$$

Applying a symmetric argument, we obtain $\sum_{i\neq j,(i,j)\in\mathcal{F}^C\times\mathcal{G}} (f'_i g'_j)^2 \leq O(\epsilon^2)\|f\|_1^2\|g\|_1^2$ with high probability. Thus each of the three terms is $O(\epsilon^2)\|f\|_1^2\|g\|_1^2$, so

$$\sum_{(i,j)\in\mathcal{A}} (f'_i g'_j)^2 = O(\epsilon^2)\|f\|_1^2\|g\|_1^2$$

with high probability. Call this event $\mathcal{E}_3$. Now by the union bound, $\mathbf{Pr}\left[\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3\right] > 1 -$

$(1/50 + 1/100 + O(n^{-c})) > 24/25$. Conditioned on this, we can bound the error term via:

$$\begin{aligned}\left|\sum_{i\neq j} X_{ij}\right| &\leq \left|\sum_{i\neq j,(i,j)\in\mathcal{F}\times\mathcal{G}} X_{ij}\right| + \left|\sum_{(i,j)\in\mathcal{A}} X_{ij}\right| \\ &\leq \epsilon\|f\|_1\|g\|_1 + 2\epsilon\sqrt{\sum_{(i,j)\in\mathcal{A}}(f_i'g_j')^2} \\ &= \epsilon\|f\|_1\|g\|_1 + 2\sqrt{O(\epsilon^3)\|f\|_1^2\|g\|_1^2} \\ &= O(\epsilon)\|f\|_1\|g|_1\end{aligned}$$

as desired. Thus with probability $24/25$ we have

$$\sum_{i=1}^{k} \mathbf{A}_i\mathbf{B}_i = \langle f', g'\rangle \pm O(\epsilon)\|f\|_1\|g\|_1$$

By Lemma 6.2.6, noting that in the notation of this Lemma the vectors f', g' have already been scaled by p_f^{-1} and p_g^{-1}, we have $\langle f', g'\rangle = \langle f, g\rangle \pm \epsilon\|f\|_1\|g\|_1$ with probability $99/100$. Altogether, with probability $1 - (1/25 + 1/100) > 11/13$ we have $\sum_{i=1}^{k} \mathbf{A}_i\mathbf{B}_i = \langle f, g\rangle \pm O(\epsilon)\|f\|_1\|g\|_1$, which is the desired result after a constant rescaling of ϵ. □

We will apply Lemma 6.2.8 to complete our proof of our main Theorem.

Theorem 47. *Given two general-turnstile stream vectors f, g with the α-property, there is a one-pass algorithm which with probability $11/13$ produces an estimate $\texttt{IP}(f, g)$ such that*

$$\texttt{IP}(f, g) = \langle f, g\rangle \pm O(\epsilon)\|f\|_1\|g\|_1$$

using $O(\epsilon^{-1}\log(\frac{\alpha\log(n)}{\epsilon}))$ bits of space.

Proof. Let $s = \Theta(\alpha^2\log(n)^7/\epsilon^{10})$, and let $I_r = [s^r, s^{r+2}]$. Then for every $r = 1, 2, \ldots, \log_s(n)$, we choose a random prime $P = [D, D^3]$ where $D = 100s^4$. We then do the following for the stream f (and apply the same following procedure for g). For every interval I_r and time step $t \in I_r$, we sample the t-th update (i_t, Δ_t) to f with probability s^{-r} (we assume $\Delta_t \in \{1, -1\}$, for bigger updates we expand them into multiple such unit updates.). If sampled, we let i_t' be the $\log(|P|)$ bit identity obtained by taking i_t modulo P. We then feed the unscaled update (i_t', Δ_t) into an instance of count-sketch with $k = O(1/\epsilon)$ buckets. Call this instance $\texttt{CS}_r^f$. At any time step t, we only store the instance $\texttt{CS}_r^f$ and $\texttt{CS}_{r+1}^f$ such that $t \in I_r \cap I_{r+1}$. At the end of the stream it is time step m^f, and fix r such that $m^f \in I_r \cap I_{r+1}$.

Now let $f' \in \mathbb{R}^n$ be the (scaled up by s^r) sample of f taken in I_r. Let $f'' \in \mathbb{R}^p$ be the (unscaled) stream on $|P|$ items that is actually fed into $\texttt{CS}_r^f$. Then $\texttt{CS}_r^f$ is run on the stream f'' which has a universe of $|P|$ items. Let $F \in \mathbb{R}^k$ be the count-sketch vector from the instance $\texttt{CS}_r^f$.

Let $\hat{f}$ be the frequency vector f restricted to the suffix of the stream $s^r, s^r+1, \ldots, m^f$ (these were the updates that were being sampled from while running $\texttt{CS}_r$). Since $m^f \in I_{r+1}$, we have $m^f \geq s^{r+1}$, so $\|f^{(s^r)}\|_1 \leq m/s < \epsilon\|f\|_1$ (by the α property), meaning the L_1 mass of the prefix of the stream $1, 2, \ldots, s^r$ is an ϵ fraction of the whole stream L_1, so removing it changes the L_1 by at most $\epsilon\|f\|_1$. It follows that $\|\hat{f} - f\|_1 \leq O(\epsilon)\|f\|_1$ and thus $\|\hat{f}\|_1 = (1 \pm O(\epsilon))\|f\|_1$. If we let $\hat{g}$ be defined analogously by replacing f with g in the previous paragraphs, we obtain $\|\hat{g} - g\|_1 \leq O(\epsilon)\|g\|_1$ and $\|\hat{g}\|_1 = (1 \pm O(\epsilon))\|g\|_1$ as well.

Now with high probability we sampled fewer than $2s^{j+2}/s^j = 2s^2$ distinct identities when creating f', so f' is $2s^2$-sparse. Let $J \subset [n] \times [n]$ be the set of pairs pf indices i, j with f'_i, f'_j non-zero. Then $|J| < 2s^4$. Let $Q_{i,j}$ be the event that $i - j = 0 \pmod P$. For this to happen, P must divide the difference. By standard results on the density of primes, there are s^8 primes in $[D, D^3]$, and since $|i - j| \leq n$ it follows that $|i - j|$ has at most $\log(n)$ prime factors. So $\mathbf{Pr}[Q_{i,j}] < \log(n)/s^8$, Let $Q = \cup_{(i,j)\in J} Q_{i,j}$, then by the union bound $\mathbf{Pr}[Q] < s^{-3}$. It follows that no two sampled identities collide when being hashed to the universe of p elements with probability $1 - 1/s^3$.

Let $\text{supp}(f') \subset [n]$ be the support of f' (non-zero indices of f'). Conditioned on Q, we have $p_f^{-1} f''_{i \pmod p} = f'_i$ for all $i \in \text{supp}(f')$, and $f''_i = 0$ if $i \neq j \pmod p$ for any $j \in \text{supp}(f')$. Thus there is a bijection between $\text{supp}(f')$ and $\text{supp}(f'')$, and the values of the respective coordinates of f', f'' are equal under this bijection. Let $g, g'', \hat{g}, m_g, p_g$ be defined analogously to f, f'' by replacing f with g in the past paragraphs, and let $G \in \mathbb{R}^k$ be the count-sketch vector obtained by running count-sketch on f just as we did to obtain $F \in \mathbb{R}^k$.

Conditioned on Q occurring (no collisions in samples) for both f'' and g'' (call these Q_f and Q_g respectively), which together hold with probability $1 - O(s^{-3})$ by a union bound, the non-zero entries of $p_f^{-1} f''$ and f' and of $p_g^{-1} g''$ and g' are identical. Thus the scaled up count-sketch vectors $p_f^{-1} F$ and $p_g^{-1} G$ of F, G obtained from running count-sketch on f'', g'' are identical in distribution to running count-sketch on f', g'. This holds because count-sketch hashes the non-zero coordinates 4-wise independently into k buckets $\mathbf{A}, \mathbf{B} \in \mathbb{R}^k$. Thus conditioned on Q_f, Q_g, we can assume that $p_f^{-1} F$ and $p_g^{-1} G$ are the result of running count-sketch on f' and g'. We claim that $p_f^{-1} p_g^{-1} \langle F, G \rangle$ is the desired estimator.

Now recall that f' is a uniform sample of $\hat{f}$, as is g' of $\hat{g}$. Then applying the count-sketch

error of Lemma 6.2.8, we have

$$\begin{aligned} p_f^{-1}p_g^{-1}\langle F, G\rangle &= \langle \hat{f}, \hat{g}\rangle + \epsilon\|\hat{f}\|_1\|\hat{g}\|_1 \\ &= \langle \hat{f}, \hat{g}\rangle + O(\epsilon)\|f\|_1\|g\|_1 \end{aligned}$$

with probability $11/13$. Now since $\|\hat{f} - f\|_1 \leq O(\epsilon)\|f\|_1$ and $\|\hat{g} - g\|_1 \leq O(\epsilon)\|g\|_1$, we have

$$\begin{aligned} \langle \hat{f}, \hat{g}\rangle &= \langle f, g\rangle \pm \sum_{i=1}^{n} (\epsilon g_i\|f\|_1 + \epsilon f_i\|g\|_1) + \epsilon^2\|f\|_1\|g\|_1 \\ &= \langle f, g\rangle \pm O(\epsilon)\|f\|_1\|g\|_1 \end{aligned}$$

Thus

$$p_f^{-1}p_g^{-1}\langle F, G\rangle = \langle f, g\rangle \pm O(\epsilon)\|f\|_1\|g\|_1$$

as required. This last fact holds deterministically using only the α-property, so the probability of success is $11/13$ as stated.

For the space, at most $2s^2$ samples were sent to each of f'', g'' with high probability. Thus the length of each stream was at most $\text{poly}(\alpha \log(n)/\epsilon)$, and each stream had $P = \text{poly}(\alpha \log(n)/\epsilon)$ items. Thus each counter in the count-sketch data structures $\mathbf{A}, \mathbf{B} \in \mathbb{R}^k$ can be stored with $O(\log(\alpha \log(n)/\epsilon))$ bits. So storing $\mathbf{A}, \mathbf{B}$ requires $O(\epsilon^{-1} \log(\alpha \log(n)/\epsilon))$ bits. Note we can safely terminate if too many samples are sent and the counters become too large, as this happens with probability $O(\text{poly}(1/n))$. The 4-wise independent hash function $h : [P] \to [k]$ used to create $\mathbf{A}, \mathbf{B}$ requires $O(\log(\alpha \log(n)/\epsilon))$ bits.

Next, by Lemma 6.2.7, the space required to hash the $\log(n)$-bit identities down to $[P]$ is $\log(P) + \log\log(n)$, which is dominated by the space for count-sketch. Finally, we can assume that s is a power of two so $p_f^{-1}, p_g^{-1} = \text{poly}(s) = 2^q$ can be stored by just storing the exponent q, which takes $\log\log(n)$ bits. To sample we then flip $\log(n)$ coins sequentially, and keep a counter to check if the number of heads reaches q before a tail is seen. □

6.3 L_1 Heavy Hitters

As an application of the CSSamplingSimulator algorithm presented in the last section, we give an improved upper bound for the classic L_1 ϵ-heavy hitters problem in the α-property setting. Formally, given $\epsilon \in (0, 1)$, the L_1 ϵ-heavy hitters problem asks to return a subset of $[n]$ that contains all items i such that $|x_i| \geq \epsilon\|x\|_1$, and no items j such that $|x_j| < (\epsilon/2)\|x\|_1$.

The heavy hitters problem is one of the most well-studied problems in the data stream literature. For general turnstile unbounded deletion streams, there is a known lower bound of $\Omega(\epsilon^{-1}\log(n)\log(\epsilon n))$ (see [BIPW10], in the language of compressed sensing, and [JST11]), and the Countsketch of [CCFC02b] gives a matching upper bound (assuming $\epsilon^{-1} = o(n)$). In the insertion only case, however, the problem can be solved using $O(\epsilon^{-1}\log(n))$ bits [BDW16], and for the strictly harder L_2 heavy hitters problem (where $\|x\|_1$ is replaced with $\|x\|_2$ in the problem definition), there is an $O(\epsilon^{-2}\log(1/\epsilon)\log(n))$-bit algorithm [BCI+16]. In this section, we beat the lower bounds for unbounded deletion streams in the α-property case. We first run a subroutine to obtain a value $R = (1 \pm 1/8)\|x\|_1$ with probability $1 - \delta$. To do this, we use the following algorithm from [KNW10a].

Fact 6.3.1 ([KNW10a])**.** *There is an algorithm which gives a* $(1\pm\epsilon)$ *multiplicative approximation with probability* $1 - \delta$ *of the value* $\|x\|_1$ *using space* $O(\epsilon^{-2}\log(n)\log(1/\delta))$.

Next, we run an instance of CSSamplingSimulator with parameters $k = 32/\epsilon$ and $\epsilon/32$ to obtain our estimate y^* of f. This requires space $O(\epsilon^{-1}\log(n)\log(\frac{\alpha\log(n)}{\epsilon}))$, and by Theorem 46 gives an estimate $y^* \in \mathbb{R}^n$ such that

$$|y_i^* - x_i| < 2\left(\sqrt{\epsilon/32}\,\mathbf{Err}\,x2 + \epsilon\|x\|_1/32\right)$$

for all $i \in [n]$ with high probability. We then return all items i with $|y_i^*| \geq 3\epsilon R/4$. Since the top $1/\epsilon$ elements do not contribute to $\mathbf{Err}\,x2$, the quantity is maximized by having k elements with weight $\|x\|_1/k$, so $\mathbf{Err}\,x2 \leq k^{-1/2}\|x\|_1$. Thus

$$\|y_i^* - x\|_\infty < (\epsilon/8)\|x\|_1$$

Given this, it follows that for any $i \in [n]$ if $|x_i| \geq \epsilon\|x\|_1$, then

$$|y_i^*| > (7\epsilon/8)\|x\|_1 > (3\epsilon/4)R$$

Similarly if $|x_i| < (\epsilon/2)\|x\|_1$, then

$$|y_i^*| < (5\epsilon/8)\|x\|_1 < (3\epsilon/4)R$$

So our algorithm correctly distinguishes ϵ heavy hitters from items with weight less than $\epsilon/2$. The probability of failure is $O(n^{-c})$ from CSSamplingSimulator and δ for estimating R, and the space required is $O(\epsilon^{-1}\log(n)\log(\frac{\alpha\log(n)}{\epsilon}))$ for running CSSamplingSimulator and another

$O(\log(n)\log(1/\delta))$ to obtain the estimate R. This gives the following theorem.

Theorem 48. *Given $\epsilon \in (0,1)$, there is an algorithm that solves the ϵ-heavy hitters problem for general turnstile α-property streams with probability $1-\delta$ using space $O(\epsilon^{-1}\log(n)\log(\frac{\alpha\log(n)}{\epsilon})+\log(n)\log(1/\delta))$.*

Now note for strict turnstile streams, we can compute $R = \|x\|_1$ exactly with probability 1 using an $O(\log(n))$-bit counter. Since the error bounds from CSSamplingSimulator holds with high probability, we obtain the following result.

Theorem 49. *Given $\epsilon \in (0,1)$, there is an algorithm that solves the ϵ-heavy hitters problem for strict turnstile α-property streams with high probability using space $O(\epsilon^{-1}\log(n)\log(\alpha\ \log(n)/\epsilon))$.*

6.4 L_1 Sampling

In this section, we consider the problem of designing approximate L_p samplers (see Chapter 3 for further introduction to this history of this problem). We begin by recalling the definition of a L_p Sampler here.

Definition 3.1.2 (L_p sampler). *Fix $p \geq 0$, values $\epsilon, \delta, \gamma \in (0,1)$, and let $x \in \mathbb{R}^n$ be a non-zero vector. Then a (ϵ,δ,γ)-approximate L_p sampler is an algorithm which outputs a random variable $Z \in [n] \cup \{\perp\}$ such that $\mathbf{Pr}[X=\perp] \leq \delta$, and such that for every $i \in [n]$:*

$$\mathbf{Pr}[X = i \mid X \neq \perp] = (1 \pm \epsilon)\frac{|x_i|^p}{\|x\|_p^p} \pm \gamma$$

We say that an algorithm is an (ϵ,δ)-approximate L_p sampler if, for any constant $c \geq 1$, it is a (ϵ,δ,n^{-c})-approximate L_p sampler.

In this section, using the data structure CSSamplingSimulator of Section 6.2, we will design an L_1 sampler for strict-turnstile strong L_1 α-property streams using

$$O\left(\epsilon^{-1}\log(\epsilon^{-1})\log(n)\log(\frac{\alpha\log(n)}{\epsilon})\log(\delta^{-1})\right)$$

bits of space. Recall that throughout the section we use *α-property* to refer to the L_1 α-property.

6.4.1 The L_1 Sampler

Our algorithm employs the precision sampling framework, as described in Section 3.2, although with several simplifications since we are only interested in approximate rather than perfect samplers. In particular, we will scale every item x_i by $1/t_i$ where $t_i \in [0,1]$ is a uniform random variable, and return any index i such that $z_i = |x_i|/t_i > \frac{1}{\epsilon}\|x\|_1$, since this occurs with probability exactly $\epsilon\frac{|x_i|}{\|x\|_1}$. One can then run a traditional count-sketch on the scaled stream z to determine when an element passes this threshold.

In this section, we will adapt this idea to *strong* α-property streams (Definition 6.1.2). The necessity of the strong α-property arises from the fact that if x has the strong α-property, then any coordinate-wise scaling z of x still has the α-property. Thus the stream z given by $z_i = x_i/t_i$ has the α-property (in fact, it again has the strong α-property, but we will only need the fact that z has the α-property). Our full L_1 sampler is given in Figure 6.3.

By running CSSamplingSimulator to find the heavy hitters of z, we introduce error additive in $O(\epsilon'\|z\|_1) = O(\epsilon^3/\log^2(n)\|z\|_1)$, but as we will see the heaviest item in z is an $\Omega(\epsilon^2/\log^2(n))$ heavy hitter with probability $1 - O(\epsilon)$ conditioned on an arbitrary value of t_i, so this error will only be an $O(\epsilon)$ fraction of the weight of the maximum weight element. Note that we use the term c-heavy hitter for $c \in (0,1)$ to denote an item with weight at least $c\|z\|_1$. Our algorithm then attempts to return an item z_i which crosses the threshold $\|x\|_1/\epsilon$, and we will be correct in doing so if the tail error $\mathbf{Err}\ z2$ from CSSamplingSimulator is not too great.

To determine if this is the case, since we are in the strict turnstile case we can compute $r = \|x\|_1$ and $q = \|z\|_1$ exactly by keeping a $\log(n)$-bit counter (note however that we will only need constant factor approximations for these). Next, using the result of Lemma 6.2.5 we can accurately estimate $\mathbf{Err}\ z2$, and abort if it is too large in Recovery Step 4 of Figure 6.3. If the conditions of this step hold, we will be guaranteed that if i is the maximal element, then $y_i^* = (1 \pm O(\epsilon))z_i$. This allows us to sample ϵ-approximately, as well as guarantee that our estimate of z_i has relative error ϵ. We now begin our analysis our L_1 sampler. First, the proof of the following fact can be found in [JST11].

Lemma 6.4.1. *Given that the values of t_i are $k = \log(1/\epsilon)$-wise independent, then conditioned on an arbitrary fixed value $t = t_l \in [0,1]$ for a single $l \in [n]$, we have*

$$\mathbf{Pr}\left[20\,\mathbf{Err}\ z2 > k^{1/2}\|x\|_1\right] = O(\epsilon + n^{-c})$$

α`L1Sampler`: L_1 sampling algorithm for strict-turnstile strong α-property streams:

Initialization

1. Instantiate CSSamplingSimulator with $k = O(\log(\epsilon^{-1}))$ columns and parameter $\epsilon' = \epsilon^3/\log^2(n)$.
2. Select $k = O(\log(\frac{1}{\epsilon}))$-wise independent uniform scaling factors $t_i \in [0, 1]$ for $i \in [n]$.
3. Run CSSamplingSimulator on scaled input z where $z_i = x_i/t_i$.
4. Keep $\log(n)$-bit counters r, q to store $r = \|x\|_1$ and $q = \|z\|_1$.

Recovery

1. Compute estimate y^* via CSSamplingSimulator.
2. Via algorithm of Lemma 6.2.5, compute v such that
$$\textbf{Err}\, z2 \leq v \leq 45\frac{k^{1/2}\epsilon^3}{\log^2(n)}\|z\|_1 + 20\,\textbf{Err}\, z2$$
3. Find i with $|y_i^*|$ maximal.
4. If $v > k^{1/2}r + 45\frac{k^{1/2}\epsilon^3}{\log^2(n)}q$, or $|y_i^*| < \max\{\frac{1}{\epsilon}r, \frac{(c/2)\epsilon^2}{\log^2(n)}q\}$ where the constant $c > 0$ is as in Proposition 6.4.2, output $\perp$. Otherwise output i and $t_i y_i^*$ as the estimate for x_i.

Figure 6.3: Our L_1 sampling algorithm with sucsess probability $\Theta(\epsilon)$

The following proposition shows that the $\epsilon/\log^2(n)$ term in the additive error of the data structure $\mathrm{CSSamplingSimulator}$ will be an ϵ fraction of the maximal element with high probability.

Proposition 6.4.2. *There exists some constant $c > 0$ such that conditioned on an arbitrary fixed value $t = t_l \in [0, 1]$ for a single $l \in [n]$, if j is such that $|z_j|$ is maximal, then with probability $1 - O(\epsilon)$ we have $|z_j| \geq c\epsilon^2/\log^2(n)\|z\|_1$.*

Proof. Let $x'_i = x_i, z'_i = z_i$ for all $i \neq l$, and let $x'_l = 0 = z'_l$. Define the level setts

$$I_t = \{i \in [n] \setminus l \mid z_i \in [\frac{\|x'\|_1}{2^{t+1}}, \frac{\|x'\|_1}{2^t})\}$$

Then

$$\mathbf{Pr}\left[i \in I_t, i \neq l\right] = 2^t \frac{|x_i|}{\|x'\|_1}$$

so $\mathbf{E}\left[|I_t|\right] = 2^t$ and by Markov's inequality $\mathbf{Pr}\left[|I_t| > \log(n)\epsilon^{-1}2^t\right] < \epsilon/\log(n)$. By the union bound:

$$\sum_{t\in[\log(n)]} \sum_{i\in I_t} |z_i| \leq \log^2(n)\|x'\|_1/\epsilon$$

with probability $1 - \epsilon$. Call this event $\mathcal{E}$. Now for $i \neq l$ let X_i indicate $z_i \geq \epsilon\|x'\|/2$. Then

$$\begin{aligned}\mathbf{Var}\left(X_i\right) &= (2/\epsilon)|x_i|/\|x'\|_1 - ((2/\epsilon)|x_i|/\|x'\|_1)^2 \\ &< \mathbf{E}\left[X_i\right]\end{aligned}$$

and so by pairwise independence of the t_i is enough to conclude that $\mathbf{Var}\left(\sum_{i\neq l} X_i\right) < \mathbf{E}\left[\sum_{i\neq l} X_i\right] = 2/\epsilon$. So by Chebyshev's inequality,

$$\mathbf{Pr}\left[\left|\sum_{i\neq l} X_i - 2\epsilon^{-1}\right| > \epsilon^{-1}\right] < 2\epsilon$$

so there is at least one and at most $3/\epsilon$ items in z' with weight $\epsilon\|x'\|_1/2$ with probability $1-O(\epsilon)$. Call these "heavy items". By the union bound, both this and $\mathcal{E}$ occur with probability $1 - O(\epsilon)$. Now the largest heavy item will be greater than the average of them, and thus it will be an $\epsilon/3$-heavy hitter among the heavy items. Moreover, we have shown that the non-heavy items in z' have weight at most $\log^2(n)\|x'\|_1/\epsilon$ with probability $1 - \epsilon$, so it follows that the maximum item in z' will have weight at least $\frac{\epsilon^2}{2\log^2 n}\|z\|_1$ with probability $1 - O(\epsilon)$.

Now if z_l is less than the heaviest item in z', then that item will still be an $\epsilon^2/(4\log^2(n))$

heavy hitter. If z_l is greater, then z_l will be an $\epsilon^2/(4\log^2(n))$ heavy hitter, which completes the proof with $c = 1/4$. □

Lemma 6.4.3. *The probability that* α`L1Sampler` *outputs the index* $i \in [n]$ *is*

$$(\epsilon \pm O(\epsilon^2))\frac{|x_i|}{\|x\|_1} + O(n^{-c})$$

The relative error of the estimate of x_i *is* $O(\epsilon)$ *with high probability.*

Proof. Ideally, we would like to output $i \in [n]$ with $|z_i| \geq \epsilon^{-1}r$, as this happens if $t_i \leq \epsilon|x_i|/r$, which occurs with probability precisely $\epsilon|x_i|/r$. We now examine what could go wrong and cause us to output i when this condition is not met or vice-versa. We condition first on the fact that v satisfies $\mathbf{Err}\, z2 \leq v \leq (45k^{1/2}\epsilon^3/\log^2(n))\|z\|_1 + 20\,\mathbf{Err}\, z2$ as in Lemma 6.2.5 with parameter $\epsilon' = \epsilon^3/\log^2(n)$, and on the fact that $|y_j^* - z_j| \leq 2(k^{-1/2}\,\mathbf{Err}\, z2 + (\epsilon^3/\log^2(n))\|z\|_1)$ for all $j \in [n]$ as detailed in Theorem 46, each of which occur with high probability.

The first type of error is if $z_i \geq \epsilon^{-1}\|x\|_1$ but the algorithm outputs $\perp$ instead of i. First, condition on the fact that $20\,\mathbf{Err}\, z2 < k^{1/2}\|x\|_1$ and that $|z_{j^*}| \geq c\epsilon^2/\log^2(n)\|z\|_1$ where $j^* \in [n]$ is such that $|z_{j^*}|$ is maximal, which by the union bound using Lemma 6.4.1 and Proposition 6.4.2 together hold with probability $1 - O(\epsilon)$ conditioned on any value of t_i. Call this conditional event $\mathcal{E}$. Since $v \leq 20\,\mathbf{Err}\, z2 + (45k^{1/2}\epsilon^3/\log^2(n))\|z\|_1$ w.h.p., it follows that conditioned on $\mathcal{E}$ we have $v \leq k^{1/2}\|x\|_1 + (45k^{1/2}\epsilon^3/\log^2(n))\|z\|_1$. So the algorithm does not fail and outbut $\perp$ due to the first condition in Recovery Step 4. of Figure 6.3. Since $v \geq \mathbf{Err}\, z2$ w.h.p, we now have $\frac{1}{k^{1/2}}\,\mathbf{Err}\, z2 \leq \|x\|_1 + 45\frac{\epsilon^3}{\log^2(n)}\|z\|_1$, and so

$$\begin{aligned}|y_j^* - z_j| &\leq 2(\frac{1}{k^{1/2}}\,\mathbf{Err}\, z2 + \frac{\epsilon^3}{\log^2(n)}\|z\|_1)\\ &\leq 2\|x\|_1 + 92\frac{\epsilon^3}{\log^2(n)}\|z\|_1\end{aligned}$$

for all $j \in [n]$.

The second way we output $\perp$ when we should not have is if $|y_i^*| < ((c/2)\epsilon^2/\log^2(n))\|z\|_1$ but $|z_i| \geq \epsilon^{-1}\|x\|_1$. Now $\mathcal{E}$ gives us that $|z_{j^*}| \geq c\epsilon^2/\log^2(n)\|z\|_1$ where $|z_{j^*}|$ is maximal in z, and since $|y_i^*|$ was maximal in y^*, it follows that $|y_i^*| \geq |y_{j^*}^*| > |z_{j^*}| - (2\|x\|_1 + 92\frac{\epsilon^3}{\log^2(n)}\|z\|_1)$.

But $|z_{j^*}|$ is maximal, so $|z_{j^*}| \geq |z_i| \geq \epsilon^{-1}\|x\|_1$. The two lower bounds on $|z_{j^*}|$ give us

$$(2\|x\|_1 + 92\frac{\epsilon^3}{\log^2(n)}\|z\|_1) = O(\epsilon)|z_{j^*}| < |z_{j^*}|/2$$

so $|y_i^*| \geq |z_{j^*}|/2 \geq ((c/2)\epsilon^2/\log^2(n))\|z\|_1$ So conditioned on $\mathcal{E}$, this type of failure can never occur. Thus the probability that we output $\perp$ for either of the last two reasons when $|z_i| \geq \epsilon^{-1}\|x\|_1$ is $O(\epsilon^2\frac{|x_i|}{\|x\|_1})$ as needed. So we can now assume that $y_i^* > ((c/2)\epsilon^2/\log^2(n))\|z\|_1$.

Given this, if an index i was returned we must have $y_i^* > \frac{1}{\epsilon}r = \frac{1}{\epsilon}\|x\|$ and $y_i^* > \frac{c\epsilon}{2\log^2 n}\|z\|_1$. These two facts together imply that our additive error from CSSamplingSimulator is at most $O(\epsilon)|y_i^*|$, and thus at most $O(\epsilon)|z_i|$, so $|y_i^* - z_i| \leq O(\epsilon)|z_i|$.

With this in mind, another source of error is if we output i with $y_i^* \geq \epsilon^{-1}r$ but $z_i < \epsilon^{-1}r$, so we should not have output i. This can only happen if z_i is close to the threshold $r\epsilon^{-1}$. Since the additive error from our count-sketch is $O(\epsilon)|z_i|$, it must be that t_i lies in the interval $\frac{|x_i|}{r}(1/\epsilon + O(1))^{-1} \leq t_i \leq \frac{|x_i|}{r}(1/\epsilon - O(1))^{-1}$, which occurs with probability $\frac{O(1)}{1/\epsilon^2 - O(1)}\frac{|x_i|}{r} = O(\epsilon^2\frac{|x_i|}{\|x\|_1})$ as needed.

Finally, an error can occur if we should have output i because $z_i \geq \epsilon^{-1}\|x\|_1$, but we output another index $i' \neq i$ because $y_{i'}^* > y_i^*$. This can only occur if $t_{i'} < (1/\epsilon - O(1))^{-1}\frac{|x_{i'}|}{r}$, which occurs with probability $O(\epsilon\frac{|x_{i'}|}{\|x\|_1})$. By the union bound, the probability that such an i' exists is $O(\epsilon)$, and by pairwise independence this bound holds conditioned on the fact that $z_i > \epsilon^{-1}r$. So the probability of this type of error is $O(\epsilon^2\frac{|x_i|}{\|x\|_1})$ as needed.

Altogether, this gives the stated $\epsilon\frac{|x_i|}{\|x\|_1}(1 \pm O(\epsilon)) + O(n^{-c})$ probability of outputting i, where the $O(n^{-c})$ comes from conditioning on the high probability events. For the $O(\epsilon)$ relative error estimate, if we return an index i we have shown that our additive error from CSSamplingSimulator was at most $O(\epsilon)|z_i|$, thus $t_i y_i^* = (1 \pm O(\epsilon))t_i z_i = (1 \pm O(\epsilon))x_i$ as needed. □

Theorem 50. *For $\epsilon, \delta > 0$, there is an $O(\epsilon)$-relative error one-pass L_1 sampler for α-property streams which also returns an $O(\epsilon)$-relative error approximation of the returned item. The algorithm outputs $\perp$ with probability at most δ, and the space is $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon})\log(n)\log(\alpha \log(n)/\epsilon)\log(\frac{1}{\delta}))$.*

Proof. By the last lemma, it follows that the prior algorithm fails with probability at most $1 - \epsilon + O(n^{-c})$. Conditioned on the fact that an index i is output, the probability that $i = j$ is $(1 \pm O(\epsilon))\frac{|x_i|}{\|x\|_1} + O(n^{-c})$. By running $O(1/\epsilon\log(1/\delta))$ copies of this algorithm in parallel and returning the first index returned by the copies, we obtain an $O(\epsilon)$ relative error sampler with

αL_1 Estimator

Input: (ϵ, δ) to estimate L_1 of an α-property strict turnstile stream.

1. **Initialization:** Set $s \leftarrow O(\alpha^2 \delta^{-1} \log^3(n)/\epsilon^2)$, and initialize `Morris-Counter` v_t with parameter δ'. Define $I_j = [s^j, s^{j+2}]$.

2. **Processing:** on update u_t, for each j such that $v_t \in I_j$, sample u_t with probability s^{-j}.

3. For each update u_t sampled while $v_t \in I_j$, keep counters c_j^+, c_j^- initialized to 0. Store all positive updates sampled in c_j^+, and (the absolute value of) all negative updates sampled in c_j^-.

4. If $v_t \notin I_j$ for any j, delete the counters c_j^+, c_j^-.

5. **Return:** $s^{-j^*}(c_{j^*}^+ - c_{j^*}^-)$ for which j^* is such that $c_{j^*}^+, c_{j^*}^-$ have existed the longest (the stored counters which have been receiving updates for the most time steps).

Figure 6.4: L_1 Estimator for strict turnstile α-property streams.

failure probability at most δ. The $O(\epsilon)$ relative error estimation of x_i follows from Lemma 6.4.3.

For space, note that CSSamplingSimulator requires

$$O\left(k \log(n) \log\left(\frac{\alpha \log n}{\epsilon}\right)\right) = O\left(\log(n) \log(1/\epsilon) \log\left(\frac{\alpha \log(n)}{\epsilon}\right)\right)$$

bits of space, which dominates the cost of storing r, q and the cost of computing v via Lemma 6.2.5, as well as the cost of storing the randomness to compute k-wise independent scaling factors t_i. Running $O(1/\epsilon \log(1/\delta))$ copies in parallel gives the stated space bound. □

Remark 51. Note that the only action taken by our algorithm which requires more space in the general turnstile case is the L_1 estimation step, obtaining r, q in step 2 of the Recovery in Figure 6.3. Note that r, q, need only be constant factor approximations in our proof, and such constant factor approximations can be obtained with high probability using $O(\log^2(n))$ bits (see Fact 6.3.1). This gives an $O(\frac{1}{\epsilon} \log(\frac{1}{\epsilon}) \log(n) \log(\frac{\alpha \log(n)}{\epsilon}) + \log^2(n))$-bit algorithm for the general turnstile case.

6.5 L_1 estimation

We now consider the L_1 estimation problem in the α-property setting (in this section we write α-property to refer to the L_1 α-property). We remark that in the general turnstile unbounded deletion setting, an $O(1)$ estimation of $\|x\|_1$ can be accomplished in $O(\log n)$ space [KNW10a]. We show in Section 6.8, however, that even for α as small as $3/2$, estimating $\|x\|_1$ in general turnstile α-property streams still requires $\Omega(\log n)$-bits of space. Nevertheless, in 6.5.2 we show that for α-property general turnstile streams there is a $\tilde{O}(\epsilon^{-2}\log(\alpha) + \log(n))$ bits of space algorithm, where $\tilde{O}$ hides $\log(1/\epsilon)$ and $\log\log(n)$ terms, thereby separating the ϵ^{-2} and $\log n$ factors. Furthermore, we show a nearly matching lower bound of $\Omega(\frac{1}{\epsilon^2}\log(\epsilon^2\alpha))$ for the problem (Theorem 62).

6.5.1 Strict Turnstile L_1 Estimation

Now for strict-turnstile α-property streams, we show that the problem can be solved with $\tilde{O}(\log(\alpha))$-bits. Ideally, to do so we would sample $\text{poly}(\alpha\log(n)/\epsilon)$ updates uniformly from the stream and apply Lemma 6.2.1. To do this without knowing the length of the stream in advance, we sample in exponentially increasing intervals, throwing away a prefix of the stream. At any given time, we will sample at two different rates in two overlapping intervals, and we will return the estimate given by the sample corresponding to the interval from which we have sampled from the longest upon termination. We first give a looser analysis of the well known Morris counting algorithm [Mor78].

Lemma 6.5.1. *There is an algorithm, `Morris-Counter`, that given $\delta \in (0,1)$ and a sequence of m events, produces non-decreasing estimates v_t of t such that*

$$\delta/(12\log(m))t \leq v_t \leq 1/\delta t$$

for a fixed $t \in [m]$ with probability $1-\delta$. The algorithm uses $O(\log\log(m))$ bits of space.

Proof. The well known Morris Counter algorithm is as follows. We initialize a counter $v_0 = 0$, and on each update t we set $v_t = v_{t-1} + 1$ with probability $1/2^{v_t}$, otherwise $v_t = v_{t-1}$. The estimate of t at time t is $2^{v_t} - 1$. It is easily shown that $\mathbf{E}[2^{v_t}] = t+1$, and thus by Markov's bound, $\mathbf{Pr}[2^{v_t} - 1 > t/\delta] < \delta$.

For the lower bound consider any interval $E_i = [2^i, 2^{i+1}]$. Now suppose our estimate of $t = 2^i$ is less than $6(2^i\delta/\log(n))$. Then we expect to sample at least $3\log(n)/\delta$ updates in E_i at $1/2$ the current rate of sampling (note that the sampling rate would decrease below this if we did

sample more than 2 updates). Then by Chernoff bounds, with high probability w.r.t. n and δ we sample at least two updates. Thus our relative error decreases by a factor of at least 2 by the time we reach the interval E_{i+1}. Union bounding over all $\log(m) = O(\log(n))$ intervals, the estimate never drops below $\delta/12\log(n)t$ for all $t \in [m]$ w.h.p. in n and δ. The counter is $O(\log\log(n))$ bits with the same probability, which gives the stated space bound. □

Our full L_1 estimation algorithm is given in Figure 6.4. Note that the value s^{-j^*} can be returned symbolically by storing s and j^*, without explicitly computing the entire value. Also observe that we can assume that s is a power of 2 by rescaling, and sample with probability s^{-i} by flipping $\log(s)i$ fair coins sequentially and sampling only if all are heads, which requires $O(\log\log(n))$ bits of space.

Theorem 52. *The algorithm αL_1`Estimator` gives a $(1 \pm \epsilon)$ approximation of the value $\|x\|_1$ of a strict turnstile stream with the α-property with probability $1 - \delta$ using*

$$O\left(\log\frac{\alpha}{\epsilon} + \log\delta^{-1} + \log\log n\right)$$

bits of space.

Proof. Let $\psi = 12\log^2(m)/\delta$. By the union bound on Lemma 6.5.1, with probability $1 - \delta$ the Morris counter v_t will produce estimates v_t such that $t/\psi \leq v_t \leq \psi t$ for all points $t = s^i/\psi$ and $t = \psi s^i$ for $i = 1, \ldots, \log(m)/\log(s)$. Conditioned on this, I_j will be initialized by time ψs^j and not deleted before s^{j+2}/ψ for every $j = 1, 2, \ldots, \log(n)/\log(s)$. Thus, upon termination, the oldest set of counters $c^+_{j^*}, c^-_{j^*}$ must have been receiving samples from an interval of size at least $m - 2\psi m/s$, with sampling probability $s^{-j^*} \geq s/(2\psi m)$. Since $s \geq 2\psi\alpha^2/\epsilon^2$, it follows by Lemma 6.2.1 that $s^{-j^*}(c^+_{j^*} - c^-_{j^*}) = \sum_{i=1}^n \hat{x}_i \pm \epsilon\|\hat{x}\|_1$ w.h.p., where $\hat{x}$ is the frequency vector of all updates after time t^* and t^* is the time step where $c^+_{j^*}, c^-_{j^*}$ started receiving updates. By correctness of our Morris counter, we know that $t^* < 2\psi m/s < \epsilon\|x\|_1$, where the last inequality follows from the size of s and the the α-property, so the number of updates we missed before initializing $c^+_{j^*}, c^-_{j^*}$ is at most $\epsilon\|x\|_1$. Since x is a strict turnstile stream, $\sum_{i=1}^n \hat{x}_i = \|x\|_1 \pm t^* = (1 \pm O(\epsilon))\|x\|_1$ and $\|\hat{x}\|_1 = (1 \pm O(\epsilon))\|x\|_1$. After rescaling of ϵ we obtain $s^{-j^*}(c^+_{j^*} - c^-_{j^*}) = (1 \pm \epsilon)\|x\|_1$ as needed.

For space, conditioned on the success of the Morris counter, which requires $O(\log\log(n))$-bits of space, we never sample from an interval I_j for more than ψs^{j+2} steps, and thus the maximum expected number of samples is ψs^2, and is at most s^3 with high probability by Chernoff bounds. Union bounding over all intervals, we never have more than s^2 samples in any interval

with high probability. At any given time we store counters for at most 2 intervals, so the space required is $O(\log(s) + \log\log(m)) = O(\log(\alpha/\epsilon) + \log(1/\delta) + \log\log(n))$ as stated. □

Remark 53. Note that if an update Δ_t to some coordinate i_t arrives with $|\Delta_t| > 1$, our algorithm must implicitly expand Δ_t to updates in $\{-1, 1\}$ by updating the counters by $\text{Sign}(\Delta_t)\cdot\text{Bin}(|\Delta_t|, s^{-j})$ for some j. Note that computing this requires $O(\log(|\Delta_t|))$ bits of working memory, which is potentially larger than $O(\log(\alpha\log(n)/\epsilon))$. However, if the updates are streamed to the algorithm using $O(\log(|\Delta_t|))$ bits then it is reasonable to allow the algorithm to have at least this much working memory. Once computed, this working memory is no longer needed and does not factor into the space complexity of maintaining the sketch of $\alpha L_1\texttt{Estimator}$.

6.5.2 General Turnstile L_1 Estimator

In [KNW10a], an $O(\epsilon^{-2}\log n)$-bit algorithm is given for general turnstile L_1 estimation. We show how modifications to this algorithm can result in improved algorithms for α-property streams. We state their algorithm in Figure 6.5, along with the results given in [KNW10a]. Let $\mathcal{D}_1$ be the standard 1-stable distribution (Definition 2.2.6).

Lemma 6.5.2 (A.6 [KNW10a]). *The entries of $\mathbf{A}, \mathbf{A}'$ can be generated to precision $\delta = \Theta(\epsilon/m)$ using $O(k\log(n/\epsilon))$ bits.*

Theorem 54 (Theorem 2.2 [KNW10a]). *The algorithm above can be implemented using precision δ in the variables $\mathbf{A}_{i,j}, \mathbf{A}'_{i,j}$, and thus precision δ in the entries y_i, y'_i, such that the output $\tilde{L}$ satisfies $\tilde{L} = (1 \pm \epsilon)\|x\|_1$ with probability $3/4$, where $\delta = \Theta(\epsilon/m)$. In this setting, we have $y'_{med} = \Theta(1)\|x\|_1$, and*

$$\left|\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{y_i}{y'_{med}})\right) - e^{-(\frac{\|x\|_1}{y'_{med}})}\right| \leq O(\epsilon)$$

.

We demonstrate that this algorithm can be implemented with reduced space complexity for α-property streams by sampling to estimate the values y_i, y'_i. We first prove an alternative version of Lemma 6.2.1.

Lemma 6.5.3. *Suppose a sequence of I insertions and D deletions are made to a single item, and let $m = I + D$ be the total number of updates. Then if X is the result of sampling updates*

General Turnstile L_1 Estimator

1. **Initialization:** Generate random matrices $\mathbf{A} \in \mathbb{R}^{r \times n}$ and $\mathbf{A} \in \mathbb{R}^{r' \times n}$ of variables drawn from $\mathcal{D}_1$, where $r = \Theta(1/\epsilon^2)$ and $r' = \Theta(1)$. The variables $\mathbf{A}_{ij}$ are k-wise independent, for $k = \Theta(\log(1/\epsilon)/\log\log(1/\epsilon))$, and the variables $\mathbf{A}'_{ij}$ are k'-wise independent for $k' = \Theta(1)$. For $i \neq i'$, the seeds used to generate the variables $\{\mathbf{A}_{i,j}\}_{j=1}^n$ and $\{\mathbf{A}_{i',j}\}_{j=1}^n$ are pairwise independent

2. **Processing:** Maintain vectors $y = \mathbf{A}x$ and $y' = \mathbf{A}'x$.

3. **Return:** Let $y'_{med} = \text{median}\{|y'_i|\}_{i=1}^{r'}$. Output $\tilde{L} = y'_{med}\Big(- \ln\Big(\frac{1}{r}\sum_{i=1}^r \cos(\frac{y_i}{y'_{med}})\Big)\Big)$

Figure 6.5: L_1 estimator of [KNW10a] for general turnstile unbounded deletion streams.

with probability $p = \Omega(\gamma^{-3}\log(n)/m)$, then with high probability

$$X = (I - D) \pm \gamma m$$

Proof. Let X^+ be the positive samples and X^- be the negative samples. First suppose $I > \epsilon m$, then $\mathbf{Pr}\left[|p^{-1}X^- - I| > \gamma m\right] < 2\exp\Big(-\frac{\gamma^2 pI}{3}\Big) < \exp\Big(-\frac{\gamma^3 pm}{3}\Big) = 1/\text{poly}(n)$. Next, if $I < \gamma m$, we have $\mathbf{Pr}\left[p^{-1}X^+ > 2\gamma m\right] < \exp\Big(-(\gamma m p/3\Big) < 1/\text{poly}(n)$ as needed. A similar bound shows that $X^- = D \pm O(\gamma)m$, thus $X = X^+ - X^- = I - D \pm O(\gamma m)$ as desired after rescaling γ. □

Theorem 55. *There is an algorithm that, given a general turnstile α-property stream x, produces an estimate $\tilde{L} = (1 \pm O(\epsilon))\|x\|_1$ with probability $2/3$ using*

$$O\left(\epsilon^{-2}\log(\frac{\alpha\log n}{\epsilon}) + \frac{\log(\frac{1}{\epsilon})\log(n)}{\log\log(\frac{1}{\epsilon})}\right)$$

bits of space.

Proof. Using Lemma 6.5.2, we can generate the matrices $\mathbf{A}, \mathbf{A}'$ using

$$O\left(\frac{k\log n}{\epsilon}\right) = O\left(\frac{\log\epsilon^{-1}\log(n/\epsilon)}{\log\log\epsilon^{-1}}\right)$$

bits of space, with precision $\delta = \Theta(\epsilon/m)$. Every time an update (i_t, Δ_t) arrives, we compute the update $\eta_i = \Delta_t \mathbf{A}_{i,i_t}$ to y_i for $i \in [r]$, and the update $\eta'_{i'} = \Delta_t \mathbf{A}'_{i',i_t}$ to $y'_{i'}$ for $i' \in [r']$.

Let $X \sim \mathcal{D}_1$ be drawn from the standard 1-stable distribution. We think of y_i and $y'_{i'}$ as streams on one variable, and we will sample from them and apply Lemma 6.5.3. We condition on the success of the algorithm of Theorem 54, which occurs with probability $3/4$. Conditioned on this, this estimator $\tilde{L}$ of Figure 6.5 is a $(1 \pm \epsilon)$ approximation, so we need only show we can estimate $\tilde{L}$ in small space.

Now the number of updates to y_i is $\sum_{q=1}^{n} |\mathbf{A}_{iq}| F_q$, where $F_q = I_q + D_q$ is the number of updates to x_q. Conditioned on $\max_{j \in [n]} |\mathbf{A}_{ij}| = O(n^2)$, which occurs with probability $1 - O(1/n)$ by a union bound, $\mathbf{E}\left[|\mathbf{A}_{iq}|\right] = \Theta(\log(n))$ (see, e.g., [Ind06]). Then

$$\mathbf{E}\left[\sum_{q=1}^{n} |\mathbf{A}_{iq}| F_q\right] = O(\log(n))\|F\|_1$$

is the expected number of updates to y_i, so by Markov's inequality

$$\sum_{q=1}^{n} |\mathbf{A}_{iq}| F_q = O(\log(n)/\epsilon^2 \|F\|_1)$$

with probability $1 - 1/(100(r + r'))$, and so with probability $99/100$, by a union bound this holds for all $i \in [r], i' \in [r']$. We condition on this now.

Our algorithm then is as follows. We then scale Δ_i up by δ^{-1} to make each update η_i, η'_i an integer, and sample updates to each y_i with probability $p = \Omega(\epsilon_0^{-3} \log(n)/m)$ and store the result in a counter c_i. Note that scaling by δ^{-1} only blows up the size of the stream by a factor of m/ϵ. Furthermore, if we can $(1 \pm \epsilon)$ approximation the L_1 of this stream, scaling our estimate down by a factor of δ gives a $(1 \pm \epsilon)$ approximation of the actual L_1, so we can assume from now on that all updates are integers.

Let $\tilde{y}_i = p^{-1} c_i$. We know by Lemma 6.5.3 that

$$\begin{aligned} \tilde{y}_i &= y_i \pm \epsilon_0 (\sum_{q=1}^{n} |\mathbf{A}_{iq}| F_q) \\ &= y_i \pm O(\epsilon_0 \log(n) \|F\|_1 / \epsilon^2) \end{aligned}$$

with high probability. Setting $\epsilon_0 = \epsilon^3/(\alpha \log(n))$, we have $|\tilde{y}_i - y_i| < \epsilon/\alpha \|F\|_1 \leq \epsilon \|x\|_1$ by the α-property for all $i \in [r]$. Note that we can deal with the issue of not knowing the length of the stream by sampling in exponentially increasing intervals $[s^1, s^3], [s^2, s^4], \ldots$ of size $s = \text{poly}(\alpha/\epsilon_0)$ as in Figure 6.4, throwing out a ϵ_0 fraction of the stream. Since our error is an

additive ϵ_0 fraction of the length of the stream already, our error does not change. We run the same routine to obtain estimates $\tilde{y'}_i$ of y'_i with the same error guarantee, and output

$$L' = \tilde{y}'_{med}\left(-\ln\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{\tilde{y}_i}{\tilde{y}'_{med}})\right)\right)$$

where $\tilde{y}'_{med} = \text{median}(\tilde{y_i}')$. By Theorem 54, we have that $y'_{med} = \Theta(\|x\|_1)$, thus $\tilde{y}'_{med} = y'_{med} \pm \epsilon\|x\|_1$ since $|\tilde{y}'_i - y'_i| < \epsilon\|x\|_1$ for all i. Using the fact that $y'_{med} = \Theta(\|x\|_1)$ by Theorem 54, we have:

$$\begin{aligned} L' &= (y'_{med} \pm \epsilon\|x\|_1)\left(-\ln\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{y_i}{y'_{med}(1\pm O(\epsilon))} \pm \frac{\epsilon\|x\|_1}{y'_{med}(1\pm O(\epsilon))})\right)\right) \\ &= (y'_{med} \pm \epsilon\|x\|_1)\left(-\ln\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{y_i}{y'_{med}}) \pm O(\epsilon)\right)\right) \end{aligned}$$

Where the last equation follows from the angle formula $\cos(\nu+\beta) = \cos(\nu)\cos(\beta) - \sin(\nu)\sin(\beta)$ and the Taylor series expansion of $\sin$ and $\cos$. Next, since

$$\left|\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{y_i}{y'_{med}})\right) - e^{-(\frac{\|x\|_1}{y'_{med}})}\right| < O(\epsilon)$$

and $y'_{med} = \Theta(\|x\|_1)$ by Theorem 54, it follows that $\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{y_i}{y'_{med}})\right) = \Theta(1)$, so using the fact that $\ln(1\pm O(\epsilon)) = \Theta(\epsilon)$, this is $(y'_{med}\pm\epsilon\|x\|_1)\left(-\ln\left(\frac{1}{r}\sum_{i=1}^{r}\cos(\frac{y_i}{y'_{med}})\right)\pm O(\epsilon)\right)$, which is $\tilde{L}\pm O(\epsilon)\|x\|_1$, where $\tilde{L}$ is the output of Figure 6.5, which satisfies $\tilde{L} = (1\pm\epsilon)\|x\|_1$ by Theorem 54. It follows that our estimate L' satisfies $L = (1\pm O(\epsilon))\|x\|_1$, which is the desired result. Note that we only conditioned on the success of Figure 6.5, which occurs with probability $3/4$, and the bound on the number of updates to every y_i, y'_i, which occurs with probability $99/100$, and high probability events, by the union bound our result holds with probability $2/3$ as needed.

For space, generating the entries of $\mathbf{A}, \mathbf{A}'$ requires $O(\log(\frac{1}{\epsilon})\log(n/\epsilon)/\log\log(\frac{1}{\epsilon}))$ bits as noted, which dominates the cost of storing δ. Moreover, every counter $\tilde{y}_i, \tilde{y}'_i$ is at most

$$\text{poly}(p\sum_{q=1}^{n}|\mathbf{A}_{iq}|F_q) = \text{poly}(\alpha\log(n)/\epsilon)$$

with high probability, and can thus be stored with $O(\log(\alpha\log(n)/\epsilon))$ bits each by storing a counter and separately storing p (which is the same for every counter). As there are $O(1/\epsilon^2)$

counters, the total space is as stated. □

6.6 L_0 Estimation

Recall that the problem of estimating the support size of a stream is known as L_0 estimation, i.e. $L_0 = |\{i \in [n] \mid x_i \neq 0\}|$. L_0 estimation is a fundamental problem for network traffic monitoring, query optimization, and database analytics [SD, AGPR99, FST88]. The problem also has applications in detecting DDoS attacks [ABRS] and port scans [EVF03].

For general turnstile streams, Kane, Nelson, and Woodruff gave an $O(\epsilon^{-2}\log(n)(\log(\epsilon^{-1}) + \log\log(n)))$-bit algorithm with constant probability of success [KNW10b], which nearly matches the known lower bound of $\Omega(\epsilon^{-2}\log(\epsilon^2 n))$ [KNW10a]. For insertion only streams, they also demonstrated an $O(\epsilon^{-2} + \log(n))$ upper bound. In this section we show that the ideas of [KNW10b] can be adapted to yield more efficient algorithms for general turnstile L_0 α-property streams. Recall that in this section, we will write α-*property* to refer to the L_0 α-property.

The idea of the algorithm stems from the observation that if $A = \Theta(K)$, then the number of non-empty bins after hashing A balls into K bins is well concentrated around its expectation. Treating this expectation as a function of A and inverting it, one can then recover A with good probability. By treating the (non-zero) elements of the stream as balls, we can hash the universe down into $K = 1/\epsilon^2$ bins and recover L_0 if $L_0 = \Theta(K)$. The primary challenge will be to ensure this last condition. In order to do so, we subsample the elements of the stream at $\log(n)$ levels, and simultaneously run an $O(1)$ estimator R of the L_0. To recover a $(1 \pm \epsilon)$ approximation, we use R to index into the level of subsampling corresponding to a substream with $\Theta(K)$ non-zero elements. We then invert the number of non-empty bins and scale up by a factor to account for the degree of subsampling.

6.6.1 Review of Unbounded Deletion Case

For sets U, V and integer k, let $\mathcal{H}_k(U, V)$ denote some k-wise independent hash family of functions mapping U into V. Assuming that $|U|, |V|$ are powers of 2, such hash functions can be represented using $O(k\log(|U| + |V|))$ bits [CW79] (without loss of generality we assume n, ϵ^{-1} are powers of 2 for the remainder of the section). For $z \in \mathbb{Z}_{\geq 0}$, we write $\text{lsb}(z)$ to denote the (0-based index of) the least significant bit of z written in binary. For instance, $\text{lsb}(6) = 1$ and $\text{lsb}(5) = 0$. We set $\text{lsb}(0) = \log(n)$. In order to fulfill the algorithmic template outlined above,

`L0Estimator`: L_0 estimation algorithm

1. Set $K = 1/\epsilon^2$, and instantiate $\log(n) \times K$ matrix $\mathbf{B}$ to 0.

2. Fix random $h_1 \in \mathcal{H}_2([n], \{0, \ldots, n-1\})$, $h_2 \in \mathcal{H}_2([n], [K^3])$, $h_3 \in \mathcal{H}_k([K^3], [K])$, and $h_4 \in \mathcal{H}_2([K^3], [K])$, for $k = \Omega(\log(\epsilon^{-1})/\log\log(\epsilon^{-1}))$.

3. Randomly choose prime $p \in [D, D^3]$, for $D = 100K\log(mM)$, and vector $\vec{u} \in \mathbb{F}_p^K$.

4. **On Update** (i, Δ): set

$$\mathbf{B}_{\text{lsb}(h_1(i)),h_3(h_2(i))} \leftarrow \left(\mathbf{B}_{\text{lsb}(h_1(i)),h_3(h_2(i))} + \Delta \cdot \vec{u}_{h_4(h_2(i))}\right) \pmod{p}$$

5. **Return**: run `RoughL0Estimator` to obtain $R \in [L_0, 110L_0]$. Set $T = |\{j \in [K] \mid \mathbf{B}_{i^*,j} \neq 0\}|$, where $i^* = \max\{0, \log(16R/K)\}$. Return estimate:

$$\tilde{L}_0 = \frac{32R}{K}\frac{\ln(1 - T/K)}{\ln(1 - 1/K)}$$

Figure 6.6: L_0 Estimation Algorithm of [KNW10b]

we need to obtain a constant factor approximation R to L_0. This is done using the following result which can be found in [KNW10b].

Lemma 6.6.1. *Given a fixed constant $\delta > 0$, there is an algorithm, `RoughL0Estimator`, that with probability $1 - \delta$ outputs a value $R = \tilde{L}_0$ satisfying $L_0 \leq R \leq 110L_0$, using space $O(\log(n)\log\log(n))$.*

The main algorithm then subsamples the stream at $\log(n)$ levels. This is accomplished by choosing a hash function $h_1 : [n] \to \{0, \ldots, n-1\}$, and subsampling an item i at level $\text{lsb}(h_1(i))$. Then at each level of subsampling, the updates to the subsampled items are hashed into $K = \frac{1}{\epsilon^2}$ bins $k = \Omega(\log(\epsilon^{-1}/\log(\log(\epsilon^{-1}))))$-wise independently. The entire data structure is then represented by a $\log(n) \times K$ matrix $\mathbf{B}$. The matrix $\mathbf{B}$ is stored modulo a sufficiently large prime p, and the updates to the rows are scaled via a random linear function to reduce the probability that deletions to one item cancel with insertions to another, resulting in false negatives in the number of buckets hit by items from the support. At the termination of the algorithm, we count the number T of non-empty bins in the i^*-th row of $\mathbf{B}$, where $i^* = \max\{0, \log(\frac{16R}{K})\}$. We then return the value $\tilde{L}_0 = \frac{32R}{K}\frac{\ln(1-T/K)}{\ln(1-1/K)}$. The full algorithm is given in Figure 6.6. First, the following Lemma can be found in [KNW10b].

Lemma 6.6.2. *There exists a constant ϵ_0 such that the following holds. Let A balls be mapped into $K = 1/\epsilon^2$ bins using a random $h \in \mathcal{H}_k([A],[K])$ for $k = c\log(1/\epsilon)/\log\log(1/\epsilon)$ for a sufficiently large constant c. Let X be a random variable which counts the number of non-empty bins after all balls are mapped, and assume $100 \leq A \leq K/20$ and $\epsilon \leq \epsilon_0$. Then $\mathbf{E}[X] = K(1-(1-K^{-1})^A)$ and*

$$\mathbf{Pr}\left[\left|X - \mathbf{E}[X]\right| \leq 8\epsilon\,\mathbf{E}[X]\right] \geq 4/5$$

Let $\mathbf{A}$ be the $\log(n) \times K$ bit matrix such that $\mathbf{A}_{i,j} = 1$ iff there is at least one $v \in [n]$ with $x_v \neq 0$ such that $\mathrm{lsb}(h_1(v)) = i$ and $h_3(h_2(v)) = j$. In other words, $\mathbf{A}_{i,j}$ is an indicator bit which is 1 if an element from the support of x is hashed to the entry $\mathbf{B}_{i,j}$ in the above algorithm. Clearly if $\mathbf{B}_{i,j} \neq 0$, then $\mathbf{A}_{i,j} \neq 0$. However, the other direction may not always hold. The proofs of the following facts and lemmas can be found in [KNW10b]. However we give them here for completeness.

Fact 6.6.3. *Let $t, r > 0$ be integers. Pick $h \in \mathcal{H}_2([r],[t])$. For any $S \subset [r]$, $\mathbf{E}\left[\sum_{i=1}^{s}\binom{|h^{-1}(i)\cap S|}{2}\right] \leq |S|^2/(2t)$.*

Proof. Let $X_{i,j}$ be an indicator variable which is 1 if $h(i) = j$. Utilizing linearity of expectation, the desired expectation is then $t\sum_{i<i'}\mathbf{E}[X_{i,1}]\,\mathbf{E}[X_{i',1}] = t\binom{|S|}{2}\frac{1}{t^2} \leq \frac{|S|^2}{2t}$. □

Fact 6.6.4. *Let $\mathbb{F}_q$ be a finite field and $v \in \mathbb{F}_q^d$ be a non-zero vector. Then, if $w \in \mathbb{F}_q^d$ is selected randomly, we have $\mathbf{Pr}[v \cdot w = 0] = 1/q$ where $v \cdot w$ is the inner product over $\mathbb{F}_q$.*

Proof. The set of vectors orthogonal to v is a linear subspace $V \subset \mathbb{F}_q^d$ of dimension $d-1$, and therefore contains q^{d-1} points. Thus $\mathbf{Pr}[w \in V] = 1/q$ as needed. □

Lemma 6.6.5 (Lemma 6 of [KNW10b])**.** *Assuming that $L_0 \geq K/32$, with probability $3/4$, for all $j \in [K]$ we have $\mathbf{A}_{i^*,j} = 0$ if and only if $\mathbf{B}_{i^*,j} = 0$. Moreover, the space required to store each $\mathbf{B}_{i,j}$ is $O(\log\log(n) + \log(1/\epsilon))$.*

Proof. The space follows by the choice of $p \in O(D^3)$, and thus it suffices to bound the probability that $\mathbf{B}_{i^*,j} = 0$ when $\mathbf{A}_{i^*,j} \neq 0$. Define $I_{i^*} = \{j \in [n] \mid \mathrm{lsb}(h_1(j)) = i^*, x_j \neq 0\}$. This is the set of non-zero coordinates of x which are subsampled to row i^* of $\mathbf{B}$. Now conditioned on $R \in [L_0, 110L_0]$, which occurs with arbitrarily large constant probability $\delta = \Theta(1)$, we have $\mathbf{E}[|I_{i^*}|] \leq K/32$, and using the pairwise independence of h_1 we have that $\mathrm{Var}(|I_{i^*}|) < \mathbf{E}[|I_{i^*}|]$. So by Chebyshev's inequality $\mathbf{Pr}[|I_{i^*}| \leq K/20] = 1 - O(1/K)$, which we now condition on.

Given this, since the range of h_2 has size K^3, the indices of I_{i^*} are perfectly hashed by h_2 with probability $1 - O(1/K) = 1 - o(1)$, an event we call Q and condition on occurring.

Since we choose a prime $p \in [D, D^3]$, with $D = 100K \log(mM)$, for mM larger than some constant, by standard results on the density of primes there are at least $(K^2 \log^2(mM))$ primes in the interval $[D, D^3]$. Since each x_j has magnitude at most mM and thus has at most $\log(mM)$ prime factors, it follows that $x_j \neq 0 \pmod p$ with probability $1 - O(1/K^2) = 1 - o(1)$. Union bounding over all $j \in I_{i^*}$, it follows that p does not divide $|x_j|$ for any $j \in I_{i^*}$ with probability $1 - o(1/K) = 1 - o(1)$. Call this event Q' and condition on it occurring. Also let Q'' be the event that $h_4(h_2(j)) \neq h_4(h_2(j'))$ for any distinct $j, j' \in I_{i^*}$ such that $h_3(h_2(j)) = h_3(h_2(j'))$.

To bound $\mathbf{Pr}[\neg Q'']$, let $X_{j,j'}$ indicate $h_3(h_2(j)) = h_3(h_2(j'))$, and let $X = \sum_{j<j'} X_{j,j'}$. By Fact 6.6.3 with $r = K^3, t = K$ and $|S| = |I_{i^*}| < K/20$, we have that $\mathbf{E}[X] \leq K/800$. Define the set

$$Z = \{(j, j') \mid h_3(h_2(j)) = h_3(h_2(j'))\}$$

For $(j, j') \in Z$, let $Y_{j,j'}$ indicate $h_4(h_2(j)) = h_4(h_2(j'))$, and let $Y = \sum_{(j,j')\in Z} Y_{j,j'}$. By the pairwise independence of h_4 and our conditioning on Q, we have

$$\mathbf{E}[Y] = \sum_{(j,j')\in Z} \mathbf{Pr}[h_4(h_2(j)) = h_4(h_2(j'))] = |Z|/K = X/K$$

Now conditioned on $X < 20\,\mathbf{E}[X] = K/40$ which occurs with probability $19/20$ by Markov's inequality, we have $\mathbf{E}[Y] \leq 1/40$, so $\mathbf{Pr}[Y \geq 1] \leq 1/40$. So we have shown that Q'' holds with probability $(19/20)(39/40) \geq 7/8$.

Now for each $j \in [K]$ such that $\mathbf{A}_{i^*,j} = 1$, we can view $\mathbf{B}_{i^*,j}$ as the dot product of the vector v, which is x restricted to the coordinates of I_{i^*} that hashed to j, with a random vector $w \in \mathbb{F}_p$ which is u restricted to coordinates of I_{i^*} that hashed to j. Conditioned on Q' it follows that v is non-zero, and conditioned on Q'' it follows that w is indeed random. So by Fact 6.6.4 with $q = p$, union bounding over all K counters $\mathbf{B}_{i^*,j}$, we have that $\mathbf{B}_{i^*,j} \neq 0$ whenever $\mathbf{A}_{i^*,j} \neq 0$ with probability $1 - K/p \geq 99/100$. Altogether, the success probability is then $(7/8)(99/100) - o(1) \geq 3/4$ as desired. □

Theorem 56. *Assuming that $L_0 > K/32$, the value returned by* `L0Estimator` *is a $(1 \pm \epsilon)$ approximation of the L_0 using space $O(\epsilon^{-2}\log(n)(\log(\frac{1}{\epsilon}) + \log(\log(n))\log(\frac{1}{\delta}))$, with $3/4$ success probability.*

Proof. By Lemma 6.6.5, we have shown that $T_{\mathbf{A}} = |\{j \in [K] \mid \mathbf{A}_{i^*,j} \neq 0\}| = T$ with probability

$3/4$, where T is as in Figure 6.6. So it suffices to show that $\tilde{L}_0^{\mathbf{A}} = \frac{32R}{K}\frac{\ln(1-T_{\mathbf{A}}/K)}{\ln(1-1/K)}$ is a $(1 \pm \epsilon)$ approximation.

Condition on the event $\mathcal{E}$ where $R \in [L_0, 110L_0]$, which occurs with large constant probability $\delta = \Theta(1)$. Define the set

$$I_{i^*} = \{j \in [n] \mid \text{lsb}(h_1(j)) = i^*, x_j \neq 0\}$$

Then $\mathbf{E}[|I_{i^*}|] = L_0/2^{i^*+1} = L_0K/(32R)$ (assuming $L_0 > K/32$) and $\text{Var}(|I_{i^*}|) < \mathbf{E}[|I_{i^*}|]$ by the pairwise independence of h_1. Then $K/3520 \leq \mathbf{E}[|I_{i^*}|] \leq K/32$ by $\mathcal{E}$, and by Chebyshev's inequality $K/4224 \leq |I_{i^*}| \leq K/20$ with probability $1 - O(1/K) = 1 - o(1)$. Call this event $\mathcal{E}'$, and condition on it. We then condition on the event $\mathcal{E}''$ that the indices of I_{i^*} are perfectly hashed, meaning they do not collide with each other in any bucket, by h_2. Given $\mathcal{E}'$, then by the pairwise independence of h_2 the event $\mathcal{E}''$ occurs with probability $1 - O(1/K)$ as well.

Conditioned on $\mathcal{E}' \wedge \mathcal{E}''$, it follows that $T_{\mathbf{A}}$ is a random variable counting the number of bins hit by at least one ball under a k-wise independent hash function, where there are $C = |I_{i^*}|$ balls, K bins, and $k = \Omega(\log(K/\epsilon)/\log\log(K/\epsilon))$. Then by Lemma 6.6.2, we have

$$T_{\mathbf{A}} = (1 \pm 8\epsilon)K(1 - (1 - 1/K)^C)$$

with probability $4/5$. So

$$\ln(1 - T_{\mathbf{A}}/K) = \ln\left((1 - 1/K)^C \pm 8\epsilon(1 - (1 - 1/K)^C)\right)$$

Since we condition on the fact that $K/4244 \leq C \leq K/32$, it follows that $(1 - 1/K)^C = \Theta(1)$, so the above is

$$\ln\left((1 \pm O(\epsilon))(1 - 1/K)^C\right) = C\ln(1 - 1/K) \pm O(\epsilon)$$

and since $\ln(1 + z) = O(|z|)$ for any real $z \in \mathbb{R}$ with $|z| < 1/2$, we have $\tilde{L}_0^{\mathbf{A}} = \frac{32RC}{K} + O(\epsilon R)$. Now the latter term is $O(\epsilon L_0)$, since $R = \Theta(L_0)$, so it suffices to show the concentration of C. Now since $\text{Var}(C) \leq \mathbf{E}[C]$ by pairwise independence of h_1, so Chebyshev's inequality gives

$$\begin{aligned}\mathbf{Pr}\left[|C - L_0K/(32R)| \geq c/\sqrt{K}\right] &< \frac{\mathbf{E}[C]}{(c^2/K)\,\mathbf{E}[C]^2} \\ &\leq \left(\frac{16}{c}\right)^2\end{aligned}$$

and this probability can be made arbitrarily small big increasing c, so set c such that the probabil-

ity is $1/100$. Note that $1/\sqrt{K} = \epsilon$, so it follows that $C = (1 \pm O(\epsilon))L_0 K/(32R)$. From this we conclude that $\tilde{L}_0^{\mathbf{A}} = (1 \pm O(\epsilon))L_0$. By the union bound the events $\mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{E}''$ occur with arbitrarily large constant probability, say $99/100$, and conditioned on this we showed that $\tilde{L}_0^{\mathbf{A}} = (1 \pm \epsilon)L_0$ with probability $4/5 - 1/100 = 79/100$. Finally, $\tilde{L}_0^{\mathbf{A}} = \tilde{L}_0$ with probability $3/4$, and so together we obtain the desired result with probability $1 - 21/100 - 1/100 - 1/4 = 53/100$. Running this algorithm $O(1)$ times in parallel and outputting the median gives the desired probability. □

6.6.2 Dealing With Small L_0

In the prior section it was assumed that $L_0 \geq K/32 = \epsilon^{-2}/32$. We handle the estimation when this is not the case the same way as [KNW10b]. We consider two cases. First, if $L_0 \leq 100$ we can perfectly hash the elements into $O(1)$ buckets and recovery the L_0 exactly with large constant probability by counting the number of nonzero buckets, as each non-zero item will be hashed to its own bucket with good probability (see Lemma 6.6.11).

Now for $K/32 > L_0 > 100$, a similar algorithm as in the last section is used, except we use only one row of $\mathbf{B}$ and no subsampling. In this case, we set $K' = 2K$, and create a vector $\mathbf{B}'$ of length K'. We then run the algorithm of the last section, but update $\mathbf{B}_j$ instead of $\mathbf{B}_{i,j}$ every time $\mathbf{B}_{i,j}$ is updated. In other words, $\mathbf{B}'_j$ is the j-th column of $\mathbf{B}$ collapsed, so the updates to all items in $[n]$ are hashed into a bucket of $\mathbf{B}'$. Let $I = \{i \in [n] \mid x_i \neq 0\}$. Note that the only fact about i^* that the proof of Lemma 6.6.5 uses was that $\mathbf{E}\left[|I_{i^*}|\right] < K/32$, and since $I = L_0 < K/32$, this is still the case. Thus by the the same argument given in Lemma 6.6.5, with probability $3/4$ we can recover a bitvector A from $\mathbf{B}$ satisfying $A_j = 1$ iff there is some $v \in [n]$ with $x_v \neq 0$ and $h_3(h_2(v)) = j$. Then if T_A is the number of non-zero bits of A, it follows by a similar argument as in Theorem 56 that

$$\tilde{L}'_0 = \ln(1 - T_A/K')/\ln(1 - 1/K') = (1 \pm \epsilon)L_0$$

for $100 < L_0 < K/32$. So if $\tilde{L}'_0 > K'/32 = K/16$, we return the output of the algorithm from the last section, otherwise we return $\tilde{L}'_0$. The space required to store $\mathbf{B}$ is $O(\epsilon^{-2}(\log\log(n) + \log(1/\epsilon)))$, giving the following Lemma.

Lemma 6.6.6. *Let $\epsilon > 0$ be given and let $\delta > 0$ be a fixed constant. Then there is a subroutine using $O(\epsilon^{-2}(\log(\epsilon^{-1}) + \log\log(n)) + \log(n))$ bits of space which with probability $1 - \delta$ either returns a $(1 \pm \epsilon)$ approximation to L_0, or returns LARGE, with the guarantee that $L_0 > \epsilon^{-2}/16$.*

α`L0Estimator`: L_0 estimator for α-property streams.

1. Initialize instance of `L0Estimator`, constructing only the top $2\log(4\alpha/\epsilon)$ rows of $\mathbf{B}$. Let all parameters and hash functions be as in `L0Estimator`.

2. Initialize α`StreamRoughL0Est` to obtain a value $\tilde{L_0^{(t)}} \in [L_0^{(t)}, 8\alpha L_0]$ for all $t \in [m]$, and set $\overline{L_0^{(t)}} = \max\{\tilde{L_0^{(t)}}, 8\log(n)/\log\log(n)\}$

3. Update the matrix $\mathbf{B}$ as in Figure 6.6, but only store the rows with index i such that $i = \log(16\overline{L_0^{(t)}}/K) \pm 2\log(4\alpha/\epsilon)$.

4. **Return**: run α`StreamConstL0Est` to obtain $R \in [L_0, 100L_0]$, and set

$$T = |\{j \in [K] \mid \mathbf{B}_{i^*,j} \neq 0\}|$$

where $i^* = \log(16R/K)$. Return the estimate

$$\tilde{L}_0 = \frac{32R}{K}\frac{\ln(1 - T/K)}{\ln(1 - 1/K)}$$

Figure 6.7: Our L_0 estimation algorithm for α-property streams with $L_0 > K/32$

6.6.3 The Algorithm for α-Property Streams

We will give a modified version of the algorithm in Figure 6.6 for L_0 α property streams. Our algorithm is given in Figure 6.7. We note first that the return value of the unbounded deletion algorithm only depends on the row $i^* = \log(16R/K)$, and so we need only ensure that this row is stored. Our L_0 α-property implies that if $L_0^{(t)}$ is the L_0 value at time t, then we must have $L_0^m = L_0 \geq 1/\alpha L_0^{(t)}$. So if we can obtain an $O(\alpha)$ approximation $R^{(t)}$ to $L_0^{(t)}$, then at time t we need only maintain and sample the rows of the matrix with index within $c\log(\alpha/\epsilon)$ distance of $i^{(t)} = \log(16R^{(t)}/K)$, for some small constant c.

By doing this, the output of our algorithm will then be the same as the output of `L0Estimator` when run on the suffix of the stream beginning at the time when we first begin sampling to the row i^*. Since we begin sampling to this row when the current $L_0^{(t)}$ is less than an ϵ fraction of the final L_0, it will follow that the L_0 of this suffix will be an ϵ-relative error approximation of the L_0 of the entire stream. Thus by the correctness of `L0Estimator`, the output of our algorithm will be a $(1 \pm \epsilon)^2$ approximation. For the remainder of the Chapter, we define

$$F_0 = \|I + D\|_0 \tag{6.16}$$

and similarly define $F_0^{(t)}$ via $F_0^{(t)} = \|I^{(t)} + D^{(t)}\|_0$, where $I^{(t)}, D^{(t)}$ are the insertion and deletion vectors after the first t updates. We remark that this notation is slightly different from its usage as the F_0-moment of the stream vector (as defined in Chapter 2), and instead aligns with the alternative usage of F_0 in the literature as referring to insertion only streams. Notice then, that F_0 is the number of distinct elements had all updates to the stream been positive.

To obtain an $O(\alpha)$ approximation to $L_0^{(t)}$ at all points t in the stream, we employ another algorithm of [KNW10b], which gives an $O(1)$ estimation of the F_0. By definition for any time $t \in [m]$ we have

$$F_0^{(t)} \leq F_0 = \|I + D\|_0 \leq \alpha \|x\|_0 = \alpha L_0$$

by the α-property, and also by definition $F_0^{(t)} \geq L_0^{(t)}$ at all times t. These two facts together imply that $[F_0^{(t)}, 8F_0^{(t)}] \subseteq [L_0^{(t)}, 8\alpha L_0]$.

Lemma 6.6.7 ([KNW10b])**.** *There is an algorithm, `RoughF0Est`, that with probability $1 - \delta$ outputs non decreasing estimates $\tilde{F}_0{}^{(t)}$ such that $\tilde{F}_0{}^{(t)} \in [F_0^{(t)}, 8F_0^{(t)}]$ for all $t \in [m]$ such that $F_0^{(t)} \geq \max\{8, \log(n)/\log\log(n)\}$, where m is the length of the stream. The space required is $O(\log(n)\log(\frac{1}{\delta}))$-bits.*

Corollary 6.6.8. *There is an algorithm, α`StreamRoughL0 Est`, that with probability $1 - \delta$ on an α-deletion stream outputs non-decreasing estimates $\tilde{L_0}{}^{(t)}$ such that $\tilde{L_0}{}^{(t)} \in [L_0^{(t)}, 8\alpha L_0]$ for all $t \in [m]$ such that $F_0^{(t)} \geq \max\{8, \log(n)/\log\log(n)\}$, where m is the length of the stream. The space required is $O(\log(n)\log(\frac{1}{\delta}))$ bits.*

Note that the approximation is only promised for t such that $F_0^{(t)} \geq \max\{8, \log(n)/\log\log(n)\}$. To handle this, we give a subroutine which produces the L_0 exactly for $F_0 < 8\log(n)/\log\log(n)$ using $O(\log(n))$ bits. Our main algorithm will assume that $F_0 \geq 8\log(n)/\log\log(n)$, and initialize its estimate of L_0^0 to be $\overline{L_0^0} = 8\log(n)/\log\log(n)$, where $\tilde{L_0^{(t)}} \in [L_0^{(t)}.8\alpha L_0]$ is the estimate produced by α`StreamRoughL0 Est`.

Lemma 6.6.9. *Given $c \geq 1$, there is an algorithm that with probability $49/50$ returns the L_0 exactly if $F_0 \leq c$, and returns LARGE if $F_0 > c$. The space required is $O(c\log(c) + c\log\log(n) + \log(n))$ bits*

Proof. The algorithm chooses a random hash function $h \in \mathcal{H}_2([n], [C])$ for some $C = \Theta(c^2)$. Every time an update (i_t, Δ_t) arrives, the algorithm hashes the identity i_t and keeps a counter, initialized to 0, for each identity $h(i_t)$ seen. The counter for $h(i_t)$ is incremented by all updates Δ_τ such that $h(i_\tau) = h(i_t)$. Furthermore, all counters are stored mod p, where p is a random

prime picked in the interval $[P, P^3]$ for $P = 100^2 c \log(mM)$. Finally, if at any time the algorithm has more than c counters stored, it returns LARGE. Otherwise, the algorithm reports the number of non-zero counters at the end of the stream.

To prove correctness, first note that at most F_0 items will ever be seen in the stream by definition. Suppose $F_0 \leq c$. By the pairwise independence of h, and scaling C by a sufficiently large constant factor, with probability $99/100$ none of the $F_0 \leq c$ identities will be hashed to the same bucket. Condition on this now. Let $I \subset [n]$ be the set of non-zero indices of x. Our algorithm will correctly report $|I|$ if p does not divide x_i for any $i \in I$. Now for mM larger then some constant, by standard results on the density of primes there are at least $100c^2 \log^2(mM)$ primes in the interval $[P, P^3]$. Since each x_i has magnitude at most mM, and thus at most $\log(mM)$ prime factors, it follows that p does not divide x_i with probability $1 - 1/(100c^2)$. Since $|I| = L_0 < F_0 \leq c$, union bounding over all $i \in I$, it follows that $p \nmid x_i$ for all $i \in I$ with probability $99/100$. Thus our algorithm succeeds with probability $1-(1/100+1/100) > 49/50$.

If $F_0 > c$ then conditioned on no collisions for the first $c+1$ distinct items seen in the stream, which again occurs with probability $99/100$ for sufficiently large $C = \Theta(c^2)$, the algorithm will necessarily see $c + 1$ distinct hashed identities once the $(c + 1)$-st item arrives, and correctly return LARGE.

For the space, each hashed identity requires $O(\log(c))$ bits to store, and each counter requires $O(\log(P)) = \log(c \log(n))$ bits to store. There are at most c pairs of identities and counters, and the hash function h can be stored using $O(\log(n))$ bits, giving the stated bound. □

Finally, to remove the $O(\log(n) \log\log(n))$ overhead of running `RoughL0Estimator` to determine the row i^*, we show that the exact same $O(1)$ approximation of the final L_0 can be obtained using $O(\log(\alpha \log(n)) \log(\log(n)) + \log(n))$ bits of space for α-property streams. We defer the proof of the following Lemma to Section 6.6.4

Lemma 6.6.10. *Given a fixed constant δ, there is an algorithm, α`StreamConstL0Est` that with probability $1 - \delta$ when run on an α property stream outputs a value $R = \hat{L}_0$ satisfying $L_0 \leq R \leq 100L_0$, using space $O(\log(\alpha) \log\log(n) + \log(n))$.*

Theorem 57. *There is an algorithm that gives a $(1\pm\epsilon)$ approximation of the L_0 value of a general turnstile stream with the α-property, using space $O(\frac{1}{\epsilon^2} \log(\frac{\alpha}{\epsilon})(\log(\frac{1}{\epsilon}) + \log\log(n)) + \log(n))$, with $2/3$ success probability.*

Proof. The case of $L_0 < K/32$ can be handled by Lemma 6.6.6 with probability $49/50$, thus

we can assume $L_0 \geq K/32$. Now if $F_0 < 8\log(n)/\log\log(n)$, we can use Lemma 6.6.9 with $c = 8\log(n)/\log\log(n)$ to compute the L_0 exactly. Conditioning on the success of Lemma 6.6.9, which occurs with probability $49/50$, we will know whether or not $F_0 < 8\log(n)/\log\log(n)$, and can correctly output the result corresponding to the correct algorithm. So we can assume that $F_0 > 8\log(n)/\log\log(n)$. Then by the α-property, it follows that $L_0 > 8\log(n)/(\alpha\log\log(n))$.

Let t^* be the time step when our algorithm initialized and began sampling to row i^*. Conditioned on the success of α`StreamRoughL0Est` and α`StreamConstL0Est`, which by the union bound together occur with probability $49/50$ for constant δ, we argue that t^* exists

First, we know that

$$\begin{aligned} i^* &= \log(16R/K) \\ &> \log(16L_0/K) \\ &> \log(16 \cdot (8\log(n)/(\log\log(n)))/K) - \log(\alpha) \end{aligned}$$

by the success of α`StreamConstL0Est`. Moreover, at the start of the algorithm we initialize all rows with indices

$$i = \log(16 \cdot (8\log\log(n)/\log(n))/K) \pm 2\log(4\alpha/\epsilon)$$

so if $i^* < \log(16\cdot(8\log\log(n)/\log(n))/K)+2\log(4\alpha/\epsilon)$ then we initialize i^* at the very beginning (time $t^* = 0$). Next, if $i^* > \log(16\cdot(8\log\log(n)/\log(n))/L)+2\log(4\alpha/\epsilon)$, then we initialize i^* at the first time t when $\overline{L_0^{(t)}} \geq R(\epsilon/(4\alpha))^2$. We know by termination that $\tilde{L_0}^m \in [L_0, 8\alpha L_0]$ since $F_0 > 8\log(n)/\log\log(n)$ and therefore by the end of the stream α`StreamRoughL0Est` will give its promised approximation. So our final estimate satisfies

$$\overline{L_0}^m \geq \tilde{L_0}^m \geq L_0 \geq R/8 > R(\epsilon/(4\alpha))^2$$

. Thus i^* will always be initialized at some time t^*.

Now because the estimates $\tilde{L_0^{(t)}}$ are non-decreasing and we have $\tilde{L_0}^m \in [L_0, 8\alpha L_0]$, it follows that $\tilde{L_0}^{(t)} < 8\alpha L_0$ for all t. Then, since

$$\overline{L_0^{(t)}} < \max\{8\alpha L_0, 8\log(n)/\log\log(n)\} < L_0(4\alpha/\epsilon)^2$$

it follows that at the termination of our algorithm the row i^* was not deleted, and will therefore be stored at the end of the stream.

Now, at time $t^* - 1$ right before row i^* was initialized we have $L_0^{t^*-1} \leq \overline{L_0^{t^*-1}} < R(\epsilon/(4\alpha))^2$, and since $R < 110L_0$ we have $L_0^{t^*-1}/L_0 \leq O(\epsilon^2)$. It follows that the L_0 value of the stream suffix starting at the time step t^* is a value $\hat{L}_0$ such that $\hat{L}_0 = (1 \pm O(\epsilon^2))L_0^m$. Since our algorithm produces the same output as running `L0Estimator` on this suffix, we obtain a $(1 \pm \epsilon)$ approximation of $\hat{L}_0$ by the proof of Theorem 56 with probability $3/4$, which in turn is a $(1 \pm \epsilon)^2$ approximation of the actual L_0, so the desired result follows after rescaling ϵ. Thus the probability of success is $1 - (3/50 + 1/4) > 2/3$.

For space, note that we only ever store $O(\log(\alpha/\epsilon))$ rows of the matrix $\mathbf{B}$, each with entries of value at most the prime $p \in O((K\log(n))^3)$, and thus storing all rows of the matrix requires $O(1/\epsilon^2 \log(\alpha/\epsilon)(\log(1/\epsilon)+\log\log(n)))$ bits. The space required to run α`StreamConstL0Est` is an additional additive $O(\log(\alpha)\log\log(n)+\log(n))$ bits. The cost of storing the hash functions h_1, h_2, h_3, h_4 is $O(\log(n)+\log^2(1/\epsilon))$ which dominates the cost of running α`StreamRoughL0Est`. Along with the cost of storing the matrix, this dominates the space required to run the small L_0 algorithm of Lemma 6.6.6 and the small F_0 algorithm of Lemma 6.6.9 with c on the order of $O(\log(n)/\log\log(n))$. Putting these together yields the stated bound. □

6.6.4 Our Constant Factor L_0 Estimator for α-Property Streams

In this section we prove Lemma 6.6.10. Our algorithm α`StreamConstL0Est` is a modification of the `RoughL0Estimator` of [KNW10b], which gives the same approximation for turnstile streams. Their algorithm subsamples the stream at $\log(n)$ levels, and our improvement comes from the observation that for α-property streams we need only consider $O(\log(\alpha))$ levels at a time. Both algorithms utilize the following lemma, which states that if the L_0 is at most some small constant c, then it can be computed exactly using $O(c^2\log\log(mM))$ space. The lemma follows from picking a random prime $p = \Theta(\log(mM)\log\log(mM))$ and pairwise independently hashing the universe into $[\Theta(c^2)]$ buckets. Each bucket is a counter which contains the sum of frequencies modulo p of updates to the universe which land in that bucket. The L_0 estimate of the algorithm is the total number of non-zero counters. The maximum estimate is returned after $O(\log(1/\eta))$ trials.

Lemma 6.6.11 (Lemma 8, [KNW10b]). *There is an algorithm which, given the promise that $L_0 \leq c$, outputs L_0 exactly with probability at least $1 - \eta$ using $O(c^2\log\log(n))$ space, in addition to needing to store $O(\log(1/\eta))$ pairwise independent hash functions mapping $[n]$ onto $[c^2]$.*

We now describe the whole algorithm `RoughL0Estimator` along with our modifications to it. The algorithm is very similar to the main algorithm of Figure 6.7. First, a random hash function $h : [n] \to [n]$ is chosen from a pairwise independent family. For each $0 \le j \le \log(n)$, a substream $\mathcal{S}_j$ is created which consists of the indices $i \in [n]$ with $\text{lsb}(h(i)) = j$. For any $t \in [m]$, let $\mathcal{S}_j^{t\to}$ denote the substream of $\mathcal{S}$ restricted to the updates $t, t+1, \ldots, m$, and similarly let $L_0^{t\to}$ denote the L_0 of the stream suffix $t, t+1, \ldots, m$. Let $L_0(\mathcal{S})$ denote the L_0 of the substream $\mathcal{S}$.

We then initialize the algorithm `Rough`α`StreamL0-Estimator`, which by Corollary 6.6.8 gives non-decreasing estimates $\tilde{L}_0^{(t)} \in [L_0^{(t)}, 8\alpha L_0]$ at all times t such that $F_0 > \frac{8\log(n)}{\log\log(n)}$ with probability $99/100$. Let

$$\overline{L_0^{(t)}} = \max\left\{\tilde{L_0^{(t)}}, \frac{8\log(n)}{\log\log(n)}\right\}$$

and let $U_t \subset [\log(n)]$ denote the set of indices i such that $i = \log(\overline{L_0^{(t)}}) \pm 2\log(\alpha/\epsilon)$, for some constant ϵ later specified.

Then at time t, for each $\mathcal{S}_j$ with $j \in U_t$, we run an instantiation of Lemma 6.6.11 with $c = 132$ and $\eta = 1/16$ on $\mathcal{S}_j$, and all instantiations share the same $O(\log(1/\eta))$ hash functions $h_1, \ldots, h_{\Theta(\log(1/\eta))}$. If $j \in U_t$ but $j \notin U_{t+1}$, then we throw away all data structures related to $\mathcal{S}_j$ at time $t+1$. Similarly, if j enters U_t at time t, we initialize a new instantiation of Lemma 6.6.11 for $\mathcal{S}_j$ at time t.

To obtain the final L_0 estimate for the entire stream, the largest value $j \in U_m$ with $j < 2\overline{L_0^m}$ such that $\mathbf{B}_j$ declares $L_0(\mathcal{S}_j) > 8$ is found. Then the L_0 estimate is $\hat{L}_0 = 20000/99 \cdot 2^j$, and if no such j exists the estimate is $\hat{L}_0 = 50$. Note that the main difference between our algorithm and `RoughL0Estimator` is that `RoughL0Estimator` sets $U_t = [\log(n)]$ for all $t \in [m]$, so our proof of Lemma 6.6.10 will follow along the lines of [KNW10b].

Proof of Lemma 6.6.10 . The space required to store the hash function h is $O(\log(n))$ and each of the $O(\log(1/\eta)) = O(1)$ hash functions h_i takes $\log(n)$ bits to store. The remaining space to store a single $\mathbf{B}^j$ is $O(\log\log(n))$ by Lemma 6.6.11, and thus storing all $\mathbf{B}^j$s for $j \in U_t$ at any time t requires at most $O(|U_t|\log\log(n)) = O(\log(\alpha)\log\log(n))$ bits (since $\epsilon = O(1)$), giving the stated bound.

We now argue correctness. First, for $F_0 \le 8\log(n)\log\log(n)$, we can run the algorithm of Lemma 6.6.9 to produce the L_0 exactly using less space than stated above. So we condition on the success of this algorithm, which occurs with probability $49/50$, and assume $F_0 >$

$8\log(n)/\log\log(n)$. This gives $L_0 > 8\log(n)/(\alpha\log\log(n))$ by the α-property, and it follows that $\overline{L_0^m} \leq 8\alpha L_0$.

Now for any $t \in [m]$, we have $\mathbf{E}\left[L_0(\mathcal{S}_j^{t\rightarrow})\right] = L_0^{t\rightarrow}/2^{j+1}$ if $j < \log(n)$, and $\mathbf{E}\left[L_0(\mathcal{S}_j^{t\rightarrow})\right] = L_0^{t\rightarrow}/n$ if $j = \log(n)$. At the end of the algorithm we have all data structures stored for $\mathbf{B}_j$'s with $j \in U_m$. Now let $j \in U_m$ be such that $j < \log(2\overline{L_0^m})$, and observe that $\mathbf{B}_j$ will be initialized at time t_j such that $\overline{L_0^{t_j}} > 2\overline{L_0^m}(\epsilon/\alpha)^2$, which clearly occurs before the algorithm terminates. If $j = \log(8\log(n)/\log\log(n)) \pm 2\log(\alpha/\epsilon)$, then $j \in U_0$ so $t_j = 0$, and otherwise t_j is such that

$$\begin{aligned} L_0^{t_j} &\leq \overline{L_0^{t_j}} \\ &\leq (\epsilon/\alpha)^2 2^j \\ &\leq (\epsilon/\alpha)^2 \overline{L_0^m} \\ &< 8(\epsilon^2/\alpha)L_0 \end{aligned}$$

So $L_0^{t_j}/L_0 = O(\epsilon^2)$. This means that when $\mathbf{B}_j$ was initialized, the value $L_0^{t_j}$ was at most an $O(\epsilon^2)$ fraction of the final L_0, from which it follows that $L_0^{t_j\rightarrow} = (1 \pm \epsilon)L_0$ after rescaling ϵ. Thus the expected output of $\mathbf{B}_j$ (if it has not been deleted by termination) for $j < \log(2L_0)$ is $\mathbf{E}\left[L_0(\mathcal{S}_j^{t_j\rightarrow})\right] = (1 \pm \epsilon)L_0/2^{j+1}$.

Now let j^* be the largest j satisfying $\mathbf{E}\left[L_0(\mathcal{S}_j^{t_j\rightarrow})\right] \geq 1$. Then $j^* < \log(2L_0) < \log(2\overline{L_0^m})$, and observe that $1 \leq \mathbf{E}\left[L_0(\mathcal{S}_{j^*}^{t_{j^*}\rightarrow})\right] \leq 2(1+\epsilon)$ (since the expectations decrease geometrically with constant $(1 \pm \epsilon)/2$). Then for any $\log(2\overline{L_0^m}) > j > j^*$, by Markov's inequality we have

$$\mathbf{Pr}\left[L_0(\mathcal{S}_j^{t_j\rightarrow}) > 8\right] < (1+\epsilon)1/(8 \cdot 2^{j-j^*-1})$$

By the union bound, the probability that any such $j \in (j^*, \log(2\overline{L_0^m}))$ has $L_0(\mathcal{S}_j^{t_j\rightarrow}) > 8$ is at most $\frac{(1+\epsilon)}{8}\sum_{j=j^*+1}^{\log(2\overline{L_0^m})} 2^{-(j-j^*-1)} \leq (1+\epsilon)/4$. Now let $j^{**} < j^*$ be the largest j such that $\mathbf{E}\left[L_0(\mathcal{S}_j^{t_j\rightarrow})\right] \geq 50$. Since the expectations are geometrically decreasing by a factor of 2 (up to a factor of $(1 \pm \epsilon)$), we have $100(1+\epsilon) \geq \mathbf{E}\left[L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow})\right] \geq 50$, and by the pairwise independence of h we have $\mathrm{Var}[L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow})] \leq \mathbf{E}\left[L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow})\right]$, so by Chebyshev's inequality we have

$$\mathbf{Pr}\left[\left|L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow}) - \mathbf{E}\left[L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow})\right]\right| < 3\sqrt{\mathbf{E}\left[L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow})\right]}\right]$$

$$> 8/9$$

Then assuming this holds and setting $\epsilon = 1/100$, we have

$$L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow}) > 50 - 3\sqrt{50} > 28$$

$$L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow}) < 100(1+\epsilon) + 3\sqrt{100(1+\epsilon)} < 132$$

What we have shown is that for every $\log(2\overline{L_0^m}) > j > j^*$, with probability at least $3/4(1-\epsilon/3)$ we will have $L_0(\mathcal{S}_j^{t_j\rightarrow}) \leq 8$. Since we only consider returning 2^j for $j \in U_m$ with $j < \log(2\overline{L_0^m})$, it follows that we will not return $\hat{L}_0 = 2^j$ for any $j > j^*$. In addition, we have shown that with probability $8/9$ we will have $28 < L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow}) < 132$, and by our choice of $c = 132$ and $\eta = 1/16$, it follows that $\mathbf{B}^{j^{**}}$ will output the exact value $L_0(\mathcal{S}_{j^{**}}^{t_{j^{**}}\rightarrow}) > 8$ with probability at least $1 - (1/9 + 1/16) > 13/16$ by Lemma 6.6.11. Hence, noting that $\epsilon = 1/100$, with probability $1 - (3/16 + 1/4(1+\epsilon)) < 14/25$, we output 2^j for some $j^{**} \leq j \leq j^*$ for which $j \in U_m$. Observe that since U_m contains all indices $i = \log(\overline{L_0^m}) \pm 2\log(\alpha/\epsilon)$, and along with the fact that $L_0 < \overline{L_0^m} < 8\alpha L_0$, it follows that all $j \in [j^{**}, j^*]$ will be in U_m at termination for sufficiently large $\epsilon \in O(1)$.

Now since $(1+1/100)L_0/2 > 2^{j^*} > (1-1/100)L_0/4$, and $(1+1/100)L_0/100 > 2^{j^{**}} > (1-1/100)L_0/200$, it follows that $(99/100)L_0/200 < 2^j < 99L_0/200$, and thus $20000/99 \cdot 2^j \in [L_0, 100L_0]$ as desired. If such a j^{**} does not exist then $L_0 < 50$ and $50 \in [L_0, 100L_0]$. Note that because of the α property, unless the stream is empty ($m = 0$), then we must have $L_0 \geq 1$, and our our approximation is always within the correct range. Finally, if $F_0 \leq 8\log(n)\log\log(n)$ then with probability $49/50$ Lemma 6.6.9 produces the L_0 exactly, and for larger F_0 we output the result of the algorithm just stated. This brings the overall success probability to $14/25 - 49/50 > 13/25$. Running $O(\log(1/\delta)) = O(1)$ copies of the algorithm and returning the median, we can amplify the probability $13/25$ to $1-\delta$. □

6.7 Support Sampling

The problem of support sampling asks, given a stream vector $x \in \mathbb{R}^n$ and a parameter $k \geq 1$, return a set $U \subset [n]$ of size at least $\min\{k, \|x\|_0\}$ such that for every $i \in U$ we have $x_i \neq 0$. Support samplers are needed crucially as subroutines for many dynamic graph streaming algorithms, such as connectivity, bipartitness, minimum spanning trees, min-cut, cut sparsifiers, spanners, and spectral sparsifiers [AGM12b]. They have also been applied to solve maximum matching [Kon15], as well as hyperedge connectivity [GMT15]. A more comprehensive study of their usefulness in dynamic graph applications can be found in [KNP+17].

α-SupportSampler

Initialization:

1. Set $s \leftarrow 205k$, and initialize linear sketch function $J : \mathbb{R}^n \rightarrow \mathbb{R}^q$ for $q = O(s)$ via Lemma 6.7.1.

2. Select random $h \in \mathcal{H}_2([n], [n])$, set $I_j = \{i \in [n] \mid h(i) \leq 2^j\}$, and set $\epsilon = 1/48$.

Processing:

1. Run αStreamRoughL0Est of Corollary 6.6.8 with $\delta = 1/12$ to obtain non-decreasing $R^{(t)} \in [L_0^{(t)}, 8\alpha L_0]$.

2. Define

$$B_t = \Big\{ j \in [\log(n)] \;\Big|\; j = \log(ns/3R^{(t)}) \pm 2\log(\alpha/\epsilon) \text{ OR } j \geq \log\left(\frac{ns\log\log(n)}{24\log(n)}\right) \Big\}$$

3. Let $t_j \in [m]$ be the first time t such that $j \in B_t$ (if one exists). Let t'_j be the first time step $t' > t_j$ such that $j \notin B_{t'}$ (if one exists).

4. For all $j \in B_t$, maintain linear sketch $y_j = J(x^{t_j:t'_j}|_{I_j})$.

Recovery:

1. For $j \in B_m$ at the end of the stream, attempt to invert y_j into $x^{t_j:m}|_{I_j}$ via Lemma 6.7.1. Return all strictly positive coordinates of all successfully returned $x^{t_j:m}|_{I_j}$'s.

Figure 6.8: Our support sampling algorithm for α-property streams.

For strict turnstile streams, an $\Omega(k \log^2(n/k))$ lower bound is known [KNP+17], and for general turnstile streams there is an $O(k \log^2(n))$ algorithm [JST11]. In this section we demonstrate that for L_0 α-property streams in the strict-turnstile case, more efficient support samplers exist. For the rest of the section, we write *α-property* to refer to the L_0 α-property, and we use the notation defined at the beginning of Section 6.6.1. Also recall that we define The F_0 value of the stream via $F_0 = \|I + D\|_0$.

First consider the following template for the unbounded deletion case (as in [JST11]). First, we subsample the set of items $[n]$ at $\log(n)$ levels, where at level j, the set $I_j \subseteq [n]$ is subsampled with expected size $|I_j| = 2^j$. Let $x|_{I_j}$ be the vector x restricted to the coordinates of I_j (and 0 elsewhere). Then for each I_j, the algorithm creates a small sketch y_j of the vector $x|_{I_j}$. If $x|_{I_j}$ is sparse, we can use techniques from sparse recovery to recover $x|_{I_j}$ and report all the non-zero coordinates. We first state the following well known result which we utilize for this recovery.

Lemma 6.7.1 ([JST11])**.** *Given $1 \leq s \leq n$, there is a linear sketch and a recovery algorithm which, given $x \in \mathbb{R}^n$, constructs a linear sketch $J(x) : \mathbb{R}^n \to \mathbb{R}^q$ for $q = O(s)$ such that if x is s-sparse then the recovery algorithm returns x on input $J(x)$, otherwise it returns `DENSE` with high probability. The space required is $O(s \log(n))$ bits.*

Next, observe that for L_0 α-deletion streams, the value $F_0^{(t)}$ is at least $L_0^{(t)}$ and at most αL_0 for every $t \in [m]$. Therefore, if we are given an estimate of $F_0^{(t)}$, we show it will suffice to only subsample at $O(\log(\alpha))$-levels at a time. In order to estimate $F_0^{(t)}$ we utilize the estimator α`StreamRoughL0Est` from Corollary 6.6.8 of Section 6.6. For $t' \geq t$, let $x^{t:t'} \in \mathbb{R}^n$ be the frequency vector of the stream of updates t to t'. We use the notation given in our full algorithm in Figure 6.8. Notice that since $R^{(t)}$ is non-decreasing, once j is removed from B_t at time t'_j it will never enter again. So at the time of termination, we have $y_j = J(x^{t_j:m}|_{I_j})$ for all $j \in B_m$.

Theorem 58. *Given a strict turnstile stream x with the L_0 α-property and $k \geq 1$, the algorithm α-`SupportSampler` outputs a set $U \subset [n]$ such that $x_i \neq 0$ for every $i \in U$, and such that with probability $1 - \delta$ we have $|U| \geq \min\{k, \|x\|_0\}$. The space required is $O(k \log(n) \log(\delta^{-1})$ $(\log(\alpha) + \log\log(n)))$ bits.*

Proof. First condition on the success of α`StreamRoughL0Est`, which occurs with probability $11/12$. Set $i^* = \min(\lceil(\log(\frac{ns}{3L_0}))\rceil, \log(n))$. We first argue that t_{i^*} exists. Now y_{i^*} would be initialized as soon as $R^{(t)} \geq L_0(\epsilon/\alpha)^2$, but $R^{(t)} \geq L_0^{(t)}$, so this holds before termination of the algorithm. Furthermore, for y_{i^*} to have been deleted by the end of the algorithm we would need $R^{(t)} > (\alpha/\epsilon)^2 L_0$, but we know $R^{(t)} < 8\alpha L_0$, so this can never happen. Finally, if

$L_0 \leq F_0 < 8\log(n)/\log\log(n)$ and our α`StreamRoughL0Est` fails, then note $i^* \geq \lceil \log(ns\log\log(n)/(24\log(n)))\rceil$, so we store i^* for the entire algorithm.

Now we have $L_0^{t_{i^*}} \leq R^{t_{i^*}} < (\epsilon/\alpha)L_0$, and thus $L_0^{t_{i^*}}/L_0 < \epsilon$. Since x is a turnstile stream, it follows that the number of strictly positive coordinates in $x^{t_{i^*}:m}$ is at least $L_0 - L_0^{t_{i^*}}$ and at most L_0. Thus there are $(1 \pm \epsilon)L_0$ strictly positive coordinates in $x^{t_{i^*}:m}$. By same argument, we have $\|x^{t_{i^*}:m}\|_0 = (1 \pm \epsilon)L_0$.

Let X_i indicate the event that $x_i^{t_{i^*}:m}|_{I_{i^*}} \neq 0$, and $X = \sum_i X_i$. Using the pairwise independence of h, the X_i's with $x_i^{t_{i^*}:m} \neq 0$ are pairwise independent, so we obtain

$$\mathbf{Var}(X) < \mathbf{E}[X] = \|x^{t_{i^*}:m}\|_0 \, \mathbf{E}[|I_{i^*}|]/n$$

First assume $L_0 > s$. Then $ns/(3L_0) \leq \mathbf{E}[|I_{i^*}|] < 2ns/(3L_0)$, so for $\epsilon < 1/48$, we have $\mathbf{E}[X] \in [15s/48, 33s/48]$. Then $\sqrt{\mathbf{E}[X]} < 1/8\,\mathbf{E}[X]$, so by Chebyshev's inequality we have

$$\mathbf{Pr}[|X - \mathbf{E}[X]| > 1/4\,\mathbf{E}[X]] < 1/4$$

and thus

$$\|x_i^{t_{i^*}:m}|_{I_{i^*}}\|_0 \leq 15/16s$$

with probability $3/4$. In this case, $x_i^{t_{i^*}:m}|_{I_{i^*}}$ is s-sparse, so we recover the vector w.h.p. by Lemma 6.7.1. Now if $L_0 \leq s$ then $F_0 < \alpha s$, so the index $i' = \log(n)$ will be stored for the entire algorithm. Thus $y_{i'} = J(f)$ and since x is s sparse we recover x exactly w.h.p., and return all non-zero elements. So we can assume that $L_0 > s$.

It suffices now to show that there are at least k strictly positive coordinates in $x_i^{t_{i^*}:m}|_{I_{i^*}}$. Since this number is also $(1 \pm L_0)$, letting X_i' indicate $x_i^{t_{i^*}:m}|_{I_{i^*}} > 0$ and using the same inequalities as in the last paragraph, it follows that there are at least $s/15 > k$ strictly positive coordinates with probability $3/4$. Since the stream is a strict-turnstile stream, every strictly positive coordinate of a suffix of the stream must be in the support of x, so we successfully return at least k coordinates from the support with probability at least $1 - (1/4 + 1/4 + 1/12 + 1/12) = 1/3$. Running $O(\log(\delta^{-1}))$ copies in parallel and setting U to be the union of all coordinates returned, it follows that with probability $1 - \delta$ at least $\min\{k, \|x\|_0\}$ distinct coordinates will be returned.

For memory, for each of the $O(\log(\delta^{-1}))$ copies we subsample at $O(\log(\alpha) + \log\log(n))$ different levels, and each to a vector of size $O(k)$ (and each coordinate of each vector takes $\log(n)$ bits to store) which gives our desired bound. This dominates the additive $O(\log(n))$ bits needed to run α`StreamRoughL0Est`. □

6.8 Lower Bounds

In this section, we show matching or nearly matching lower bounds for all problems considered in this Chapter. Our lower bounds all follow via reductions from one-way randomized communication complexity. We consider both the public coin model, where Alice and Bob are given access to infinitely many shared random bits, as well as the private coin model, where they do not have shared randomness.

Remark 59 (Notation for Frequency Vector)**.** Since the notation x, y are often reserved to denote the input to the communication protocol, in this section only we will use the notation $f \in \mathbb{R}^n$ to represent the frequency vector of a data stream.

Communication Protocols

We first state the communication complexity problems we will be reducing from. The first such problem we use is the Augmented-Indexing problem (IND), which is defined as follows. Alice is given a vector $y \in \{0,1\}^n$, and Bob is given a index $i^* \in [n]$, and the values $y_{i^*+1}, \ldots, y_n$. Alice then sends a message M to Bob, from which Bob must output the bit y_{i^*} correctly. A correct protocol for IND is one for which Bob correctly outputs y_{i^*} with probability at least $2/3$. The communication cost of a correct protocol is the maximum size of the message M that the protocol specifies to deliver. This problem has a well known lower bound of $\Omega(n)$ (see [MNSW95], or [KNW10a]).

Lemma 6.8.1 (Miltersen et al. [MNSW95])**.** *The one-way communication cost of any protocol for Augmented Indexing (*IND*) in the public coin model that succeeds with probability at least* $2/3$ *is* $\Omega(n)$.

We present the second communication complexity result which we will use for our reductions. We define the problem EQUALITY as follows. Alice is given $y \in \{0,1\}^n$ and Bob is given $x \in \{0,1\}^n$, and are required to decide whether $x = y$. This problem has a well known $\Omega(\log(n))$-bit lower bound when shared randomness is not allowed (see e.g., [AMS96] where it is used).

Lemma 6.8.2. *The one way communication complexity in the private coin model with* $2/3$ *success probability of* EQUALITY *is* $\Omega(\log(n))$.

Heavy Hitters Lower Bound

We begin with the hardness of the heavy hitters problem in the strict-turnstile setting. Our hardness result holds not just for α-property streams, but even for the special case of strong α-property streams (Definition 6.1.2). The result matches our upper bound for normal α-property streams from Theorem 49 up to $\log\log(n)$ and $\log(\epsilon^{-1})$ terms.

Theorem 60. *For $p \geq 1$ and $\epsilon \in (0,1)$, any one-pass L_p heavy hitters algorithm for strong L_1 α-property streams $x \in \mathbb{R}^n$ in the strict turnstile model which returns a set containing all $i \in [n]$ such that $|x_i| \geq \epsilon \|x\|_p$ and no $i \in [n]$ such that $|x_i| < (\epsilon/2)\|x\|_p$ with probability at least $2/3$ requires $\Omega(\epsilon^{-p}\log(n\epsilon^p)\log(\alpha))$ bits.*

Proof. Suppose there is such an algorithm which succeeds with probability at least $2/3$, and consider an instance of augmented indexing. Alice receives $y \in \{0,1\}^d$, and Bob gets $i^* \in [d]$ and y_j for $j > i^*$. Set $D = 6$, and let X be the set of all subsets of $[n]$ with $\lfloor 1/(2\epsilon)^p \rfloor$ elements, and set $d = \log_6(\alpha/4)\lfloor \log(|X|) \rfloor$. Alice divides y into $r = \log_6(\alpha/4)$ contiguous chunks $y^1, y^2, \ldots, y^r$ each containing $\lfloor \log(|X|) \rfloor$ bits. She uses y^j as an index into X to determine a subset $U_j \subset [n]$ with $|U_j| = \lfloor 1/(2\epsilon)^p \rfloor$. Thinking of U_j as a binary vector in $\mathbb{R}^n$, Alice defines the vector $v \in \mathbb{R}^n$ by.

$$v = (\alpha D + 1)U_1 + (\alpha D^2 + 1)U_2 + \cdots + (\alpha D^r + 1)U_r$$

She then creates a stream and inserts the necessary items so that the current frequency vector of the stream is v. She then sends the state of her heavy hitters algorithm to Bob, who wants to know $y_{i^*} \in y^j$ for some $j = j(i^*)$. Knowing $y_{i^*+1}, \ldots, y_d$ already, he can compute $u = \alpha D^{j+1}U_{j+1} + \alpha D^{j+2}U_{j+2} + \cdots + \alpha D^r U_r$. Bob then runs the stream which subtracts off u from the current stream, resulting in a final stream frequency vector of $f = v - u$. He then runs his heavy hitters algorithm to obtain a set $S \subset [n]$ of heavy hitters.

We now argue that if correct, his algorithm must produce $S = U_j$. Note that some items in $[n]$ may belong to multiple sets U_i, and would then be inserted as part of multiple U_i's, so we must deal with this possibility. For $p \geq 1$, the weight $\|f\|_p^p$ is maximized by having $U_1 = U_2 = \cdots = U_j$, and thus

$$\|f\|_p^p \leq 1/(2\epsilon)^p(\sum_{i=1}^{j} \alpha D^i + 1)^p$$

$$\leq \epsilon^{-p}(\alpha D^{j+1}/10 + 1)^p$$

$$\leq \epsilon^{-p}\alpha^p D^{jp}$$

and for every $k \in U_j$ we have $|f_k|^p \geq (\alpha D^j + 1)^p \geq \epsilon^p \|f\|_p^p$ as desired. Furthermore, $\|f\|_p^p \geq |U_j| \alpha^p D^{jp} = \lfloor 1/(2\epsilon)^p \rfloor \alpha^p D^{jp}$, and the weight of any element $k' \in [n] \setminus U_j$ is at most

$$\left(\sum_{i=1}^{j-1} \alpha D^i + 1 \right)^p \leq (\alpha D^j/5 + (j-1))^p < D^{jp}/4^p$$

for $\alpha \geq 3$, so $f_{k'}$ will not be a $\epsilon/2$ heavy hitter. Thus, if correct, Bob's heavy hitter algorithm will return $S = U_j$. So Bob obtains $S = U_j$, and thus recovers y^j which indexed U_j, and can compute the relevant bit $y_{i^*} \in y^j$ and return this value successfully. Hence Bob solves IND with probability at least $2/3$.

Now observe that at the end of the stream each coordinate has frequency at least 1 and received fewer than $3\alpha^2$ updates (assuming updates have magnitude 1). Thus this stream on $[n]$ items has the strong $(3\alpha^2)$-property. Additionally, the frequencies of the stream are always non-negative, so the stream is a strict turnstile stream. It follows by Lemma 6.8.1 that any heavy hitters algorithm for strict turnstile strong α-property streams requires

$$\Omega(d) = \Omega(\log(\sqrt{\alpha/3}) \log(|X|)) = \Omega(\epsilon^{-p} \log(\alpha) \log(n\epsilon^p))$$

bits as needed. □

L_1 Norm Estimation Lower Bounds

Next, we demonstrate the hardness of estimating the L_1 norm in the α-property setting. First, we show that the problem of L_1 estimation in the general turnstile model requires $\Omega(\log(n))$-bits even for α property streams with $\alpha = O(1)$. We also give a lower bound of $\Omega(1/\epsilon^2 \log(\alpha))$ bits for general turnstile L_1 estimation for strong α-property streams.

Theorem 61. *For any $\alpha \geq 3/2$, any algorithm that produces an estimate $\tilde{L_1} \in (1 \pm 1/16)\|f\|_1$ of a general turnstile stream with frequency vector $f \in \mathbb{R}^n$ with the L_1 α property with probability $2/3$ requires $\Omega(\log(n))$ bits of space.*

Proof. Let $\mathcal{G}$ be a family of $t = 2^{\Omega(n/2)} = 2^{n'}$ subsets of $[n/2]$, each of size $n/8$ such that no two sets have more than $n/16$ elements in common. As noted in [AMS96], the existence of $\mathcal{G}$ follows from standard results in coding theory, and can be derived via a simple counting argument. We now reduce from EQUALITY, where Alice has $y \in \{0,1\}^{n'}$ and Bob has $x \in \{0,1\}^{n'}$. Alice can use y to index into $\mathcal{G}$ to obtain a subset $s_y \subset [n/2]$, and similarly Bob obtains $s_x \subset [n/2]$ via x.

Let $y', x' \in \{0,1\}^n$ be the characteristic vectors of s_y, s_x respectively, padded with $n/2$ 0's at the end.

Now Alice creates a stream $f \in \mathbb{R}^n$ on n elements by first inserting y', and then inserting the vector v where $v_i = 1$ for $i > n/2$ and 0 otherwise. She then sends the state of her streaming algorithm to Bob, who deletes x' from the stream. Now if $x = y$, then $x' = y'$ and $\|f\|_1 = \|y' + v - x'\|_1 = n/2$. On the other hand, if $x \neq y$ then each of s_x, s_y have at least $n/16$ elements in one and not in the other. Thus $\|y' - x'\|_1 \geq n/8$, so $\|f\|_1 \geq 5n/8$. Thus a streaming algorithm that produces $\tilde{L_1} \in (1 \pm 1/16)\|f\|_1$ with probability $2/3$ can distinguish between the two cases, and therefore solve EQUALITY, giving an $\Omega(\log(n'))$ lower bound. Since $n' = \Omega(n)$, it follows that such an L_1 estimator requires $\Omega(\log(n))$ bits as stated.

Finally, note that at most $3n/4$ unit increments were made to f, and $\|f\|_1 \geq n/2$, so f indeed has the $\alpha = 3/2$ property. □

Theorem 62. *Any algorithm that produces an estimate $\tilde{L} \in (1 \pm \epsilon)\|f\|_1$ with probability $11/12$ of a general turnstile stream $f \in \mathbb{R}^n$ with the strong L_1 α property requires $\Omega(\frac{1}{\epsilon^2}\log(\epsilon^2\alpha))$ bits of space.*

To prove Theorem 62, we first define the following communication complexity problem.

Definition 6.8.3. *In the* GAP-HAM *communication complexity problem, Alice is given $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$. Bob is promised that either $\|x - y\|_1 < n/2 - \sqrt{n}$ (NO instance) or that $\|x - y\|_1 > n/2 + \sqrt{n}$ (YES instance), and must decide which instance holds.*

Our proof of Theorem 62 will use the following reduction. Our proof is similar to the lower bound proof for unbounded deletion streams in [KNW10a].

Theorem 63 ([JKS08], [Woo07] Section 4.3)**.** *There is a reduction from* IND *to* GAP-HAM *such that deciding* GAP-HAM *with probability at least $11/12$ implies a solution to* IND *with probability at least $2/3$. Furthermore, in this reduction the parameter n in* IND *is within a constant factor of that for the reduced* GAP-HAM *instance.*

We are now ready to prove Theorem 62.

Proof. Set $k = \lfloor 1/\epsilon^2 \rfloor$, and let $t = \lfloor \log(\alpha\epsilon^2) \rfloor$. The reduction is from IND. Alice receives $x \in \{0,1\}^{kt}$, and Bob obtains $i^* \in [kt]$ and x_j for $j > i^*$. Alice conceptually breaks her string x

up into t contiguous blocks b_i of size k. Bob's index i^* lies in some block b_{j^*} for $j^* = j^*(i^*)$, and Bob knows all the bits of the blocks b_i for $i > j^*$. Alice then applies the reduction of Theorem 63 on each block b_i separately to obtain new vectors y_i of length ck for $i \in [t]$, where $c \geq 1$ is some small constant. Let $\beta = c^2\epsilon^{-2}\alpha$. Alice then creates a stream with frequency vector $f \in \mathbb{R}^{ckt}$ on ckt items by inserting the update $((i,j), \beta 2^i + 1)$ for all (i,j) such that $(y_i)_j = 1$. Here we are using $(i,j) \in [t] \times [ck]$ to index into $[ckt]$. Alice then computes the value $v_i = \|y_i\|_1$ for $i \in [t]$, and sends $\{v_1, \ldots, v_t\}$ to Bob, along with the state of the streaming algorithm run on the frequency vector f.

Upon receiving this, since Bob knows the bits y_z for $z \geq i^*$, Bob can run the same reductions on the blocks b_i as Alice did to obtain y_i for $i > j^*$. He then can make the deletions $((i,j), -\beta 2^i)$ for all (i,j) such that $i > j^*$ and $(y_i)_j = 1$, leaving these coordinate to be $f_{(i,j)} = 1$. Bob then performs the reduction from IND to GAP-HAM specifically on the block b_{j^*} to obtain a vector $y(B)$ of length ck, such that deciding whether $\|y(B) - y_{j^*}\|_1 > ck/2 + \sqrt{ck}$ or $\|y(B) - y_{j^*}\|_1 < ck/2 - \sqrt{ck}$ with probability $11/12$ will allow Bob to solve the instance of IND on block b_{j^*} with index i^* in b_{j^*}. Then for each i such that $y(B)_i = 1$, Bob makes the update $((j^*, ji), -\beta 2^{j^*})$ to the stream f. He then runs an L_1 approximation algorithm to obtain $\tilde{L} = (1 \pm \epsilon)\|f\|_1$ with probability $11/12$. Let A be the number of indices such that $y(B)_i > (y_{j^*})_i$. Let B be the number of indices such that $y(B)_i < (y_{j^*})_i$. Let C be the number of indices such that $y(B)_i = 1 = (y_{j^*})_i$, and let D be the number of indices (i,j) with $i > j^*$ such that $(y_i)_j = 1$. Then we have

$$\|f\|_1 = \beta 2^{j^*} A + (\beta 2^{j^*} + 1)B + C + D + \sum_{i<j^*} v_i(\beta 2^i + 1)$$

Let $Z = C + D + B$, and note that $Z < ckt < \beta$. Let $\eta = \sum_{i<j^*} v_i(\beta 2^i + 1)$. Bob can compute η exactly knowing the values $\{v_1, \ldots, v_t\}$. Rearranging terms, we have

$$\|y(B) - y_{j^*}\|_1 = (\|f\|_1 - Z - \eta)/(\beta 2^{j^*}) = (\|f\|_1 - \eta)/(\beta 2^{j^*}) \pm 1$$

Recall that Bob must decide whether $\|y(B) - y_{j^*}\|_1 > ck/2 + \sqrt{ck}$ or $\|y(B) - y_{j^*}\|_1 < ck/2 - \sqrt{ck}$. Thus, in order to solve this instance of GAP-HAM it suffices to obtain an additive $\sqrt{ck}/8$ approximation of $\|f\|_1/(\beta 2^{j^*})$. Now note that

$$\|f\|_1/(\beta 2^{j^*}) \leq ckt(\beta 2^{j^*})^{-1} + (\beta 2^{j^*})^{-1} \sum_{i=1}^{j^*} \beta 2^j \cdot ck$$

$$\leq 1 + 4ck$$

where the $ckt < \beta$ term comes from the fact that every coordinate that is ever inserted will have magnitude at least 1 at the end of the stream. Taking $\epsilon' = \sqrt{ck}/(40ck) = 1/(40\sqrt{ck}) = O(1/\epsilon)$, it follows that a $(1 \pm \epsilon')$ approximation of $\|f\|_1$ gives a $\sqrt{ck}/8$ additive approximation of $\|f\|_1/(\beta 2^{j^*})$ as required. Thus suppose there is an algorithm A that can obtain such a $(1 \pm O(1/\epsilon^2))$ approximation with probability $11/12$. By the reduction of Theorem 63 and the hardness of IND in Lemma 6.8.1, it follows that this protocol just described requires $\Omega(kt)$ bits of space. Since the only information communicated between Alice and Bob other than the state of the streaming algorithm was the set $\{v_1, \ldots, v_t\}$, which can be sent with $O(t \log(k)) = o(kt)$ bits, it follows that A must have used $\Omega(kt) = \Omega(\epsilon^{-2} \log(\alpha\epsilon))$ bits of space, which is the desired lower bound.

Now note that every coordinate i that was updated in the stream had final magnitude $|f_i| \geq 1$. Furthermore, no item was inserted more than $\beta 2^t + 1 < \alpha^2 c^2 + 1$ times, thus the stream has the strong $O(\alpha^2)$ property. We have proven that any algorithm that gives a $(1 \pm \epsilon)$ approximation of $\|f\|_1$ where f is a strong α-property stream with probability $11/12$ requires $\Omega(\epsilon^{-2} \log(\epsilon^2\sqrt{\alpha}))$ $= \Omega(\epsilon^{-2} \log(\epsilon^2\alpha))$ bits, which completes the proof. □

We now give a matching lower bound for L_1 estimation of strict-turnstile strong α-property streams. This exactly matches our upper bound of Theorem 52, which is for the more general α-property setting.

Theorem 64. *For $\epsilon \in (0, 1/2)$ and $\alpha < n$, any algorithm which gives an ϵ-relative error approximation of the L_1 of a strong L_1 α property stream in the strict turnstile setting with probability at least $2/3$ must use $\Omega(\log(\alpha) + \log(1/\epsilon) + \log\log(n))$ bits.*

Proof. The reduction is from IND. Alice, given $x \in \{0, 1\}^t$ where $t = \log_{10}(\alpha/4)$, constructs a vector $u \in \mathbb{R}^t$ such that $u_i = \alpha 10^i x_i + 1$. She then inserts the vector u into the stream so that the frequency vector is give nby $f = u$, and sends the state of the stream f to Bob who, given $j \in [n]$ and $x_{j+1}, \ldots, x_t$, subtracts off v where $v_i = \alpha 2^i x_i$ for $i \geq j + 1$, and 0 otherwise. He then runs the L_1 estimation algorithm on the final frequency vector $f = u - v$, and obtains the value L such that $L = (1 \pm \epsilon)\|f\|_1$ with probability $2/3$. We argue that if $L = (1 \pm \epsilon)\|f\|_1$ (for $\epsilon < 1/2$) then $L > (1 - \epsilon)(\alpha 10^j) > \alpha 10^j/2$ iff $x_j = 1$. If $x_j = 1$ then $(u_j - v_j) = \alpha 10^j + 1$, if $L > (1 - \epsilon)\|f\|_1$ the result follows. If $x_j = 0$, then the total L_1 is at most $\alpha/4 + \alpha \sum_{i=1}^{j-1} 10^i < \alpha \frac{10^i}{9} + \alpha/4 < \alpha 10^i/3$, so $L < (1 + \epsilon)\|f\|_1 < \alpha 10^j/2$ as needed to solve IND. Note that each coordinate has frequency at least 1 at the end of the stream, and no coordinate received more than α^2 updates. Thus the stream has the strong α^2-property. By

Lemma 6.8.1, it follows that any one pass algorithm for constant factor L_1 estimation of a strict turnstile strong α-property stream requires $\Omega(\log(\sqrt{\alpha})) = \Omega(\log(\alpha))$ bits.

Finally, note that in the restricted insertion only case (i.e. $\alpha = 1$), estimating the L_1 norm means estimating the value m given only the promise that $m \leq \mathbb{M}$ for some value $\mathbb{M} = \text{poly}(n)$. There are $\log(\mathbb{M})/\epsilon$ powers of $(1+\epsilon)$ that could potentially be a $(1 \pm \epsilon)$ approximation of m, so to represent the solution requires requires $\log(\log(\mathbb{M})/\epsilon) = O(\log\log(n) + \log(1/\epsilon))$ bits of space, which gives the rest of the stated lower bound. □

L_0 Norm Estimation Lower Bound

We now prove a lower bound on L_0 estimation. Our lower bound matches our upper bound of Theorem 57 up to $\log\log(n)$ and $\log(1/\epsilon)$ multiplicative factors, and a $\log(n)$ additive term. To do so, we use the following Theorem of [KNW10a], which uses a one way two party communication complexity lower bound.

Theorem 65 (Theorem A.2. [KNW10a])**.** *Any one pass algorithm that gives a $(1\pm\epsilon)$ multiplicative approximation of the L_0 of a strict turnstile stream with probability at least $11/12$ requires $\Omega(\epsilon^{-2}\log(\epsilon^2 n))$ bits of space.*

Setting $n = O(\alpha)$ will give us:

Theorem 66. *For $\epsilon \in (0, 1/2)$ and $\alpha < n/\log(n)$, any one pass algorithm that gives a $(1 \pm \epsilon)$ multiplicative approximation of the L_0 of a L_0 α-property stream in the strict turnstile setting with probability at least $11/12$ must use $\Omega(\epsilon^{-2}\log(\epsilon^2\alpha) + \log\log(n))$ bits of space.*

Proof. We can first construct the same stream used in the communication complexity lower bound of Theorem 65 on $n = \alpha - 1$ elements, and then allow Bob to insert a final dummy element at the end of the stream with frequency 1. The stream now has α elements, and the L_0 of the resulting stream, call it R, is exactly 1 larger than the initial L_0 (which we will now refer to as L_0). Moreover, this stream has the L_0 α property since the final frequency vector is non-zero and there α items in the stream. If we then obtained an estimate $\tilde{R} = (1\pm\epsilon)R = (1\pm\epsilon)(L_0+1)$, then the original $L_0 = 0$ if $\tilde{R} < (1+\epsilon)$. Otherwise $\tilde{R} - 1 = (1 \pm O(\epsilon)L_0$, and a constant factor rescaling of ϵ gives the desired approximation of the initial L_0. By Theorem 65, such an approximation requires $\Omega(\epsilon^{-2}\log(\epsilon^2\alpha))$ bits of space, as needed. The $\Omega(\log\log(n))$ lower bound follows from the proof of Lemma 64, replacing the upper bound $\mathbb{M} \geq m$ with n. □

Sampling Lower Bounds

Next, we give lower bounds for L_1 and support sampling. Our lower bound for L_1 samplers holds in the more restricted strong α-property setting, and for such streams we show that even those which return an index from a distribution with variation distance at most $1/6$ from the L_1 distribution $|f_i|/\|f\|_1$ of a frequency vector $f \in \mathbb{R}^n$, requires $\Omega(\log(n)\log(\alpha))$ bits. In this setting, taking $\epsilon = o(1)$, this bound matches our upper bound from Theorem 50 up to $\log\log(n)$ terms. For $\alpha = o(n)$, our support sampling lower bound matches our upper bound in Theorem 58.

Theorem 67. *Any one pass L_1 sampler of a strong L_1 α-property stream f in the strict turnstile model with an output distribution that has variation distance at most $1/6$ from the L_1 distribution $|f_i|/\|f\|_1$ and succeeds with probability $2/3$ requires $\Omega(\log(n)\log(\alpha))$ bits of space.*

Proof. Consider the same strict turnstile strong $O(\alpha^2)$-property stream constructed by Alice and Bob in Theorem 60, with $\epsilon = 1/2$. Then $X = [n]$ is the set of all subsets of $[n]$ with 1 item. If $i^* \in [n]$ is Bob's index, then let $j = j(i^*)$ be the block y^j such that $y_{i^*} \in y^j$. The block y^j has $\log(|X|) = \log(n)$ bits, and Bob's goal is to determine the set $x_j \subset [n]$ of exactly one item which is indexed by y^j. Then for the one sole item $k \in x_j$ will be a $1/2$-heavy hitter, and no other item will be a $1/4$-heavy hitter, so Bob can run $O(1)$ parallel L_1 samplers and find the item k' that is returned the most number times by his samplers. If his sampler functions as specified, having at most $1/6$ variational distance from the L_1 distribution, then $k' = k$ with large constant probability and Bob can recovery the bit representation x_j of k, from which he recovers the bit $y_{i^*} \in y_j$ as needed. Since Alice's string had length $\Omega(\log(\alpha)\log(|X|)) = \Omega(\log(\alpha)\log(n))$, we obtain the stated lower bound. □

Theorem 68. *Any one pass support sampler that outputs an arbitrary $i \in [n]$ such that $f_i \neq 0$, of an L_0 α-property stream with probability of outputting $\perp$ at most $1/3$, requires $\Omega(\log(n/\alpha)\log(\alpha))$ bits of space.*

Proof. The reduction is again from IND. Alice receives $y \in \{0,1\}^d$, for $d = \lfloor \log(n/\alpha)\log(\alpha/4) \rfloor$, and breaks it into blocks $y^1, \ldots, y^{\log(\alpha/4)}$, each of size $\lfloor \log(n/\alpha) \rfloor$. She then initializes a stream vector $f \in \mathbb{R}^n$, and breaks f into $\lfloor 4n/\alpha \rfloor$ blocks of size $\alpha/4$, say $B_1, \ldots, B_{\lfloor 4n/\alpha \rfloor}$. She uses y_i as an index into a block B_j for $j = j(i)$, and then inserts 2^i distinct items into block B_j, each exactly once, and sends the state of her algorithm over to Bob. Bob wants to determine y_{i^*} for his fixed $i^* \in [n]$. Let k be such that $y_{i^*} \in y^j$ and B_k be the block indexed by y^j. He knows

$y^{j+1}, \ldots, y^{\log(\alpha/4)}$, and can delete the corresponding items from f that were inserted by Alice. At the end, the block B_k has 2^j items in it, and the total number of distinct items in the stream is less than 2^{j+1}. Moreover no other block has more than 2^{j-1} items.

Now suppose Bob had access to an algorithm that would produce a uniformly random non-zero item at the end of the stream, and that would report $\perp$ with probability at most $1/3$. He could then run $O(1)$ such algorithms, and pick the block $B_{k'}$ such that more than $4/10$ of the returned indices are in $B_{k'}$. If his algorithms are correct, we then must have $B_{k'} = B_k$ with large constant probability, from which he can recover y^j and his bit y_{j^*}, thus solving IND and giving the $\Omega(d) = \Omega(\log(n/\alpha)\log(\alpha))$ lower bound by Lemma 6.8.1.

We now show how such a random index from the support of f can be obtained using only a support sampler. Alice and Bob can used public randomness to agree on a uniformly random permutation $\pi : [n] \to [n]$, which gives a random relabeling of the items in the stream. Then, Alice creates the same stream as before, but using the relabeling and inserting the items in a randomized order into the stream, using separate randomness from that used by the streaming algorithm. In other words, instead of inserting $i \in [n]$ if i was inserted before, Alice inserts $\pi(i)$ at a random position in the stream. Bob the receives the state of the streaming algorithm, and then similarly deletes the items he would have before, but under the relabeling π and in a randomized order instead.

Let $i_1, \ldots, i_r \in [n]$ be the items inserted by Alice that were not deleted by Bob, ordered by the order in which they were inserted into the stream. If Bob were then to run a support sampler on this stream, he would obtain an arbitrary $i = g(i_1, \ldots, i_r) \in \{i_1, \ldots, i_r\}$, where g is a (possibly randomized) function of the ordering of the sequence $i_1, \ldots, i_r$. The randomness used by the streaming algorithm is separate from the randomness which generated the relabeling π and the randomness which determined the ordering of the items inserted and deleted from the stream. Thus, even conditioned on the randomness of the streaming algorithm, any ordering and labeling of the surviving items $i_1, \ldots, i_r$ is equally likely. In particular, i is equally likely to be any of $i_1, \ldots, i_r$. It follows that $\pi^{-1}(i)$ is a uniformly random element of the support of f, which is precisely what Bob needed to solve IND, completing the proof. □

Inner Product Estimation Lower Bound

Finally, we show that estimating inner products even for *strong* α-property streams requires $\Omega(\epsilon^{-1}\log(\alpha))$ bits of space. Setting $\alpha = n$, we obtain an $\Omega(\epsilon^{-1}\log(n))$ lower bound for unbounded deletion streams, which our upper bound beats for small α.

Theorem 69. *Any one pass algorithm that runs on two strong L_1 α-property streams f, g in the strict turnstile setting and computes a value $\texttt{IP}(f,g)$ such that $\texttt{IP}(f,g) = \langle f, g\rangle + \epsilon\|f\|_1\|g\|_1$ with probability $2/3$ requires $\Omega(\epsilon^{-1}\log(\alpha))$ bits of space.*

Proof. The reduction is from IND. Alice has $y \in \{0,1\}^d$ where y is broken up into $\log_{10}(\alpha)/4$ blocks of size $1/(8\epsilon)$, where the indices of the i-th block are called B_i. Bob wants to learn y_{i^*} and is given y_j for $j \geq i^*$, and let j^* be such that $i^* \in B_{j^*}$. Alice creates a stream f on d items, and if $i \in B_j$, then Alice inserts the items to make $f_i = b_i 10^j + 1$, where $b_i = \alpha$ if $y_i = 0$, and $b_i = 2\alpha$ otherwise. She creates a second stream $g = \vec{0} \in \mathbb{R}^d$, and sends the state of her streaming algorithm over to Bob. For every $y_i \in B_j = B_{j(i)}$ that Bob knows, he subtracts off $b_i 10^j$, leaving $f_i = 1$. He then sets $g_{i^*} = 1$, and obtains $\texttt{IP}(f,g)$ via his estimator. We argue that with probability $2/3$, $\texttt{IP}(f,g) \geq 3\alpha 10^{j^*}/2$ iff $y_{i^*} = 1$. Note that the error is always at most

$$\begin{aligned}
\epsilon\|f\|_1\|g\|_1 &= \epsilon\|f\|_1 \\
&\leq \epsilon\left(d + (8\epsilon)^{-1}(2\alpha 10^{j^*}) + \sum_{j<j^*}\sum_{i\in B_j}(2\alpha 10^j)\right) \\
&\leq \epsilon\left(\frac{1}{8\epsilon}\cdot\frac{\log(\alpha)}{4} + (4\epsilon)^{-1}\alpha 10^{j^*} + \epsilon^{-1}\alpha 10^{j^*}/32\right) \\
&< \alpha 10^{j*}/3
\end{aligned}$$

Now since $\langle f, g\rangle = (y_{i^*} + 1)\alpha 10^{j^*} + 1$, if $y_{i^*} = 0$ then if the inner product algorithm succeeds with probability $2/3$ we must have $\texttt{IP}(f,g) \leq \alpha 10^j + 1 + \alpha 10^j/3 < 3\alpha 10^{j^*}/2$, and similarly if $y_{i^*} = 1$ we have $\texttt{IP}(f,g) > 2\alpha 10^j + 1 - \alpha 10^j/3 > 3\alpha 10^j/2$ as needed. So Bob can recover y_{i^*} with probability $2/3$ and solve IND, giving an $\Omega(d)$ lower bound via Lemma 6.8.1. Since each item in f received at most $5(\alpha^2)$ updates and had final frequency 1, this stream has the strong $5\alpha^2$-property, and g was insertion only. Thus obtaining such an estimate of the inner product between strong α-property streams requires $\Omega(\epsilon^{-1}\log(\sqrt{\alpha})) = \Omega(\epsilon^{-1}\log(\alpha)$ bits, as stated. □

Chapter 7

Adversarially Robust Streaming

Our focus in Part I of this thesis has been on the design of *randomized* algorithms for streaming and distributed data. This preoccupation with randomization is not a coincidence. Specifically, many central problems in the streaming literature provably do not admit sublinear-space deterministic algorithms, so for such problems randomization is necessary. On the other hand, even for problems where sublinear space deterministic algorithms exists, randomized solutions are often more efficient and simpler to implement than their deterministic counterparts. Thus, randomization is a core component of sketching both in theory and practice.

While randomized streaming algorithms are very well-studied, the vast majority of them are defined and analyzed in the *static* setting, where the stream is first fixed in advance, and only then the randomness of the algorithm chosen. This has been the case for all sketching results described in this thesis so far. However, assuming that the stream sequence is independent of the randomness of the sketch, and in particular that future elements of the stream do not depend on previous outputs of the streaming algorithm, may not be realistic [MNS11, GHR+12, GHS+12, HW13, NY15a, BY20], even in non-adversarial settings. The failure to handle any amount of adaptivity is a major limitation of many known randomized sketching algorithms.

Scenarios where future inputs to an algorithm adaptively depend on past decisions of the algorithm are very common, and arise in many settings. For example, an adversary uses maliciously chosen examples to fool a trained machine learning model [SZS+14, MHS19]. Another example comes from recommendation systems, where an online store suggests recommended items based on a sample of previous purchases, which in turn influences future sales [SS+11, GHR+12]. In networking, a device routes traffic according to statistics pulled from a sampled substream of packets [DLT03a], and an adversary that observes the network's traffic learns the device's routing

choices might cause a denial-of-service attack by generating a small amount of adversarial traffic [NY15a]. Furthermore, in the field of autonomous vehicles, a vehicle receives physical signals from its immediate environment (which might be adversarial [SBM+18]) and has to decide on a suitable course of action, which in turn influences the future environment of the vehicle.

A streaming algorithm that works even when the stream is adaptively chosen by an adversary is said to be *adversarially robust*. Deterministic algorithms are inherently adversarially robust, since they are guaranteed to be correct on all possible inputs. However, the large gap in performance between deterministic and randomized streaming algorithms for many problems motivates the need for designing adversarially robust randomized algorithms, if they even exist. In particular, we would like to design adversarially robust randomized algorithms which are as space and time efficient as their static counterparts, and yet as robust as deterministic algorithms.

The focus of this chapter is precisely the study of such adversarially robust algorithm. In particular, we develop a generic framework for transforming any non-robust algorithm for a streaming problem into a robust one, with only a small space overhead. We apply this transformation to obtain the first adversarially robust algorithms for many fundamental streaming problems, including distinct element (L_0) estimation, L_p-norm estimation, heavy hitters, entropy estimation, and several others. In each case, the space complexity of all our algorithms is only a small factor larger than that of the best non-robust (static) algorithm for the same problem. Specifically, for a problem with precision parameter $\epsilon > 0$, the complexity of our robust algorithm is usually at most a $\frac{1}{\epsilon} \log n$ factor larger than that of the best possible static algorithm for the problem, and we often reduce this blow-up to only $\frac{1}{\epsilon} \log \frac{1}{\epsilon}$. Lastly, we demonstrate the necessity of such a framework by designing an adversary which breaks one of the most well-known streaming algorithms from the literature: the AMS sketch for L_2 estimation.

Highlighted Contributions

The materials from this chapter are drawn from a joint work with Omri Ben-Eliezer, David Woodruff, and Eylon Yogev [JSTW19]. The main contributions therein are as follows:

- We introduce a generic framework for robustification, which transforms any non-robust algorithm for a streaming problem into a robust one, with small space overhead. Our framework consists of two distinct and incomparable transformations: the *sketch-switching* technique and the *computation paths* technique (Section 7.2).

- We utilize this framework, along with several additional techniques and modifications, to obtain the first adversarially robust streaming algorithms for a wide array of fundemental streaming problems, each requiring only a small factor more in their space complexity than the best non-robust algorithms (Sections 7.3 to 7.7).

- We demonstrate the necessity of designing new algorithms by proving that the classical AMS sketch is *not* adversarially robust (Section 7.8).

7.1 Background

There are several ways to define the adversarial streaming setting, which depend on how much information the adversary (who chooses the stream) can observe from the streaming algorithm, and on any restrictions imposed on the computational resources of the adversary. We consider perhaps the most general non-trivial model, where the adversary is allowed unbounded computational power and resources, though we do discuss the case later when the adversary is computationally bounded. At each point in time, the streaming algorithm publishes its output to a query for the stream. The adversary observes these outputs one-by-one, and can choose the next update to the stream adaptively, depending on the full history of the outputs and stream updates. The goal of the adversary is to force the streaming algorithm to eventually produce an *incorrect* output to the query, as defined by the specific streaming problem in question.[1]

Specifically, let $x \in \mathbb{R}^n$ be the frequency vector of a data-stream (see Chapter 2 for streaming preliminaries, and Section 1.1 for further introduction to the streaming model). The general task posed to a streaming algorithm is to correctly respond to some query $\mathcal{Q}$ about the frequency vector $x^{(t)}$ at each point in time $t \in [m]$. For instance, this query could be to approximate some function $g : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ of $x^{(t)}$,, such as counting the number of distinct elements in a data stream, i.e. $g(x^{(t)}) = \|x^{(t)}\|_0 = |\{i \in [n] \mid x_i^{(t)} \neq 0\}|$.

Formally, the adversarial setting is modeled by a two-player game between a (randomized) StreamingAlgorithm and an Adversary. At the beginning, a query $\mathcal{Q}$ is fixed, which the StreamingAlgorithm must continually reply to. The game proceeds in rounds, where in the t-th round:

[1] In the streaming literature, and in the majority of this thesis, an algorithm is often required to be correct on a query made only *once*, at the end of the stream. This is a *one-shot* guarantee, as opposed to the *tracking* guarantee as defined here. However, the two settings are nearly equivalent. Indeed, for almost all streaming problems, a one-shot algorithm can be made into a tracking algorithm with at most an $O(\log n)$ blow-up in space, by simply setting the failure probability small enough to union bound over all points in the stream.

Round t in game between STREAMINGALGORITHM and ADVERSARY

1. ADVERSARY chooses an update $u_t = (a_t, \Delta_t)$ for the stream, which can depend on all previous stream updates and outputs of STREAMINGALGORITHM.

2. STREAMINGALGORITHM processes the new update u_t and outputs its current response R^t to the query $\mathcal{Q}$.

3. ADVERSARY observes R^t (stores it) and proceeds to the next round.

The goal of the ADVERSARY is to make the STREAMINGALGORITHM output an incorrect response R^t to $\mathcal{Q}$ at some point t in the stream. For example, in the distinct elements problem, the adversary's goal is that on some step t, the estimate R^t will fail to be a $(1+\epsilon)$-approximation of the true current number of distinct elements $|\{i \in [n] : x_i^{(t)} \neq 0\}|$.

Streaming algorithms in the adversarial setting It was shown by Hardt and Woodruff [HW13] that linear sketches are inherently *non-robust* in adversarial settings for a large family of problems, thus demonstrating a major limitation of such sketches. In particular, their results imply that no linear sketch can approximate the Euclidean norm of its input to within a polynomial multiplicative factor in the adversarial (turnstile) setting. Here, a linear sketch is an algorithm whose output depends only on values $\mathbf{A}x$ and $\mathbf{A}$, for some (usually randomized) sketching matrix $\mathbf{A} \in \mathbb{R}^{k \times n}$. This is quite unfortunate, as the vast majority of turnstile streaming algorithms are in fact linear sketches.

On the positive side, a recent work of Ben-Eliezer and Yogev [BY20] showed that *random sampling* is quite robust in the adaptive adversarial setting, albeit with a slightly larger sample size. While uniform sampling is a rather generic and important tool, it is not sufficient for solving many important streaming tasks, such as estimating frequency moments (F_p-estimation), finding F_2 heavy hitters, and various other central data analysis problems. This raises the natural question of whether there exist efficient adversarially robust randomized streaming algorithms for these problems and others, which is the main focus of this work. Perhaps even more importantly, we ask the following.

Is there a generic technique to transform a static streaming algorithm into an adversarially robust streaming algorithm?

The focus on this chapter is to answers the above question affirmatively for a large class of algorithms.

7.1.1 Results and Techniques

We devise adversarially robust algorithms for various fundamental insertion-only streaming problems, including distinct element estimation, F_p moment estimation, heavy hitters, entropy estimation, and several others. In addition, we give adversarially robust streaming algorithms which can handle a bounded number of deletions as well. The required space of our adversarially robust algorithms matches that of the best known non-robust ones up to a small multiplicative factor. Our new algorithmic results are summarized in Table 7.1. In contrast, we demonstrate that some classical randomized algorithms for streaming problems in the static setting, such as the celebrated Alon-Matias-Szegedy (AMS) sketch [AMS96] for F_2-estimation, are inherently non-robust to adaptive adversarial attacks in a strong sense.

The adversarially robust algorithms introduced in this chapter make use of two generic robustification frameworks that we develop, allowing one to efficiently transform a non-robust streaming algorithm into a robust one in various settings. Both of the robustification methods rely on the fact that functions of interest do not drastically change their value too many times along the stream. Specifically, the transformed algorithms have space dependency on the *flip-number* of the stream, which is a bound on the number of times the function $g(x^{(t)})$ can change by a factor of $(1 \pm \epsilon)$ in the stream (see Section 7.2).

The first method, called *sketch switching*, maintains multiple instances of the non-robust algorithm and switches between them in a way that cannot be exploited by the adversary. The second technique bounds the number of *computation paths* possible in the two-player adversarial game. This technique maintains only one copy of a non-robust algorithm, albeit with an extremely small probability of error δ. We show that a carefully rounded sequence of outputs generates only a small number of possible computation paths, which can then be used to ensure robustness by union bounding over these paths. The framework is described in Section 7.2.

The two above methods are incomparable: for some streaming problems the former is more efficient, while for others, the latter performs better, and we show examples of each. Specifically, sketch switching can exploit efficiency gains of *strong-tracking*, resulting in particularly good performance for static algorithms that can respond correctly to queries at each step without having to union bound over all m steps. In contrast, the computation paths technique can exploit an algorithm with good dependency on δ (the failure probability). Namely, algorithms that have small dependency in update-time or space on δ will benefit from the computation paths technique.

For each of the problems considered in this chapter, we show how to use the framework, in

Problem	Static Rand.	Deter.	Adversarial	Comments
Distinct elem. (F_0 est.)	$\tilde{O}(\epsilon^{-2}+\log n)$	$\Omega(n)$	$\boldsymbol{\tilde{O}(\epsilon^{-3}+\epsilon^{-1}\log n)}$	
			$\boldsymbol{\tilde{O}(\epsilon^{-2}+\log n)}$	crypto/rand. oracle
F_p estimation, $p\in(0,2]\setminus\{1\}$	$O(\epsilon^{-2}\log n)$	$\tilde{\Omega}(c_p n)$	$\boldsymbol{\tilde{O}(\epsilon^{-3}\log n)}$	
	$O(\epsilon^{-3}\log^2 n)$		$\boldsymbol{\tilde{O}(\epsilon^{-3}\log^3 n)}$	$\delta=\Theta(n^{-\frac{1}{\epsilon}}\log n)$
F_p estimation, $p>2$	$O(n^{1-\frac{2}{p}}(\epsilon^{-3}\log^2 n +\epsilon^{-\frac{6}{p}}\log^{\frac{4}{p}+1} n))$	$\Omega(n)$	$\boldsymbol{O(n^{1-\frac{2}{p}}(\epsilon^{-3}\log^2 n +\epsilon^{-\frac{6}{p}}\log^{\frac{4}{p}+1} n))}$	$\delta=\Theta(n^{-\frac{1}{\epsilon}}\log n)$
ℓ_2 Heavy Hit.	$O(\epsilon^{-2}\log^2 n)$	$\Omega(\sqrt{n})$	$\boldsymbol{\tilde{O}(\epsilon^{-3}\log^2 n)}$	
Entropy estimation	$O(\epsilon^{-2}\log^3 n)$	$\tilde{\Omega}(n)$	$\boldsymbol{O(\epsilon^{-5}\log^6 n)}$	
	$\tilde{O}(\epsilon^{-2})$		$\boldsymbol{O(\epsilon^{-5}\log^4 n)}$	crypto/rand. oracle
Turnstile F_p, $p\in(0,2]$	$O(\epsilon^{-2}\lambda\log^2 n)$	$\Omega(n)$	$\boldsymbol{O(\epsilon^{-2}\lambda\log^2 n)}$	λ-bounded F_p flip num., $\delta=\Theta(n^{-\lambda})$
F_p, $p\in[1,2]$ α-bounded del.	$\tilde{O}(\log^2 n+$ $\epsilon^{-2}\log\alpha\log n)$	$\tilde{\Omega}(c_p n)$	$\boldsymbol{O(\alpha\epsilon^{-(2+p)}\log^3 n)}$	static only for $p=1$

Table 7.1: A summary of the adversarially robust algorithms (in bold) introduced in our paper [BJWY20], as compared to the best known upper bounds for randomized algorithms in the static setting and lower bounds for deterministic algorithms. Note that all stated algorithms provide tracking. All results except for the last two (which hold in restricted versions of the turnstile model) are for insertion only streams. We write $\tilde{O},\tilde{\Omega}$ to hide $\log\epsilon^{-1}$ and $\log\log n$ factors. The static randomized upper bounds are proved, respectively, in [Bła18], [BDN17], [KNW10a], [GW18], [BCI+16], [CC13], [JW19], [KNW10a], and [JW18a]. All lower bounds for F_p-estimation are proved in [CK16], except for the turnstile bound, proved in [AMS96]; the lower bound for heavy hitters is from [KPW20]. Finally, the lower bound for deterministic entropy estimation follows from a reduction from estimating F_p for $p=1+\tilde{\Theta}(\frac{\epsilon}{\log^2 n})$ to entropy estimation [HNO08a].

addition to some further techniques which we develop along the way, to solve it. Interestingly, we also demonstrate how cryptographic assumptions (which previously were not commonly used in a streaming context) can be applied to obtain an adversarially robust algorithm against computationally bounded adversaries for the distinct elements problem at essentially no extra cost over the space optimal non-robust one. See Table 7.1 for a summary of our results in the adversarial setting compared to the state-of-the-art in the static setting, as well as to deterministic algorithms.

Distinct elements and F_p-estimation Our first suite of results provides robust streaming algorithms for estimating F_p, the p^{th} frequency moment of the frequency vector, which recall is defined as $F_p = \|x\|_p^p = \sum_{i=1}^n |x_i|^p$, where we interpret $0^0 = 0$. Estimating frequency moments has a myriad of applications in databases, computer networks, data mining, and other contexts. Efficient algorithms for estimating distinct elements (i.e., estimating F_0) are important for databases, since query optimizers can use them to find the number of unique values of an attribute without having to perform an expensive sort on the values. Efficient algorithms for F_2 are useful for determining the output size of self-joins in databases, and for computing the surprise index of a data sequence [Goo89]. Higher frequency moments are used to determine data skewness, which is important in parallel database applications [DNSS92].

We remark that for any fixed $p \neq 1$,[2] including $p = 0$, any deterministic insertion-only algorithm for F_p-estimation requires $\Omega(n)$ space [AMS96, CK16]. In contrast, we will show that randomized adversarially robust algorithms exist for all p, whose space complexity either matches or has a small multiplicative overhead over the best static randomized algorithms.

We begin with several results on the problem of estimating distinct elements, or F_0 estimation. The first of them utilizes an optimized version of the sketch switching method to derive an upper bound. The result is an adversarially robust F_0 estimation algorithm whose complexity is only a $\Theta(\frac{1}{\epsilon}\log\epsilon^{-1})$ factor larger than the optimal static (non-robust) algorithm [Bła18].

Theorem 70 (Robust Distinct Elements by Sketch Switch; see Theorem 86)**.** *There is an algorithm which, when run on an adversarial insertion only stream, with probability at least $1-\delta$ produces at every step $t \in [m]$ an estimate R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_0$. The space used by the algorithm is*

$$O\left(\frac{\log(1/\epsilon)}{\epsilon}\left(\frac{\log\epsilon^{-1} + \log\delta^{-1} + \log\log n}{\epsilon^2} + \log n\right)\right).$$

The second result utilizes a different approach, by applying the computation paths method. The space complexity is slightly worse, which is a result of setting the failure probability $\delta <$

[2]Note that there is a trivial $O(\log n)$-bit insertion only F_1 estimation algorithm: keeping a counter for $\sum_t \Delta_t$.

$n^{-\frac{1}{\epsilon}\log n}$ for any given static algorithm. However, we introduce a new *static* algorithm for F_0 estimation which has very small update-time dependency on δ, and nearly optimal space complexity. As a result, by applying our computation paths method to this new static algorithm, we obtain an adversarially robust F_0 estimation algorithm with extremely fast update time (note that the update time of the above sketch switching algorithm would be $O(\epsilon^{-1}\log n)$ to obtain the same result, even for constant δ).

Theorem 71 (Fast Robust Distinct Elements; see Theorem 87)**.** *There exists a streaming algorithm which, with probability $1-n^{-(C/\epsilon)\log n}$ for any constant $C \geq 1$, when run on an adversarial chosen insertion-only data stream, returns a $(1\pm\epsilon)$ multiplicative estimate of the number of distinct elements at every step in the stream. The space required is $O(\frac{1}{\epsilon^3}\log^3 n)$, and the algorithm runs in $O((\log^2 \log \frac{\log n}{\epsilon}) \cdot (\log\log\log \frac{\log n}{\epsilon}))$ worst case time per update.*

The third result takes a different approach: it shows that under certain standard cryptographic assumptions, there exists an adversarially robust algorithm which asymptotically matches the space complexity of the best non-robust tracking algorithm for distinct elements. The cryptographic assumption is that an exponentially secure pseudorandom function exists (in practice one can take, for instance, AES as such a function). While our other algorithms in this Chapter hold even against an adversary which is unbounded computationally, in this particular result we assume that the adversary runs in polynomial time. See Section 7.9 for more details.

Theorem 72 (Distinct Elements by Crypto Assumptions; see Theorem 93)**.** *In the random oracle model, there is an F_0-estimation (tracking) streaming algorithm in the adversarial setting, that for an approximation parameter ε uses $O(\varepsilon^{-2}(\log 1/\varepsilon + \log\log n) + \log n)$ bits of memory, and succeeds with probability $3/4$.*

Moreover, under a suitable cryptographic assumption, assuming the adversary has bounded running time of n^c, where m is the stream length and c is a constant, the random oracle can be replaced with a concrete function and the total memory is $O(\varepsilon^{-2}(\log 1/\varepsilon + \log\log n) + c\log n)$.

Here, the random oracle model means that the algorithm is given read access to an arbitrarily long string of random bits.

Our next set of results provides adversarially robust algorithms for F_p-estimation with $p > 0$. The following result concerns the case $0 < p \leq 2$. It was previously shown that for p bounded away from one, $\Omega(n)$ space is required to deterministically estimate $\|x\|_p^p$, even in the insertion only model [AMS96, CK16]. On the other hand, space-efficient non-robust randomized algorithms for F_p-estimation exist. We leverage these, along with an optimized version of the

sketch switching technique to save a $\log n$ factor, and obtain the following.

Theorem 73 (Robust F_p-estimation for $0 < p \leq 2$; see Theorem 82)**.** *Fix $0 < \epsilon, \delta \leq 1$ and $0 < p \leq 2$. There is a streaming algorithm in the insertion-only adversarial model which outputs at each step a value R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p$ at every step $t \in [m]$, and succeeds with probability $1 - \delta$. The algorithm uses $O(\epsilon^{-3} \log n \log \epsilon^{-1}(\log \epsilon^{-1} + \log \delta^{-1} + \log \log n))$ bits of space.*

We remark that the space complexity of Theorem 73 is within a $\Theta(\epsilon^{-1} \log \epsilon^{-1})$ factor of the best known static (non-robust) algorithm [BDN17] . While for most values of δ, the above theorem using sketch switching has better space complexity than the computation paths reduction, for the regime of very small failure probability δ it is actually preferable to use the latter, as we now state.

Theorem 74 (Robust F_p-estimation for small δ; see Theorem 83)**.** *Fix any $0 < \epsilon < 1$, $0 < p \leq 2$, and $\delta < n^{-C\frac{1}{\epsilon}\log n}$ for a sufficiently large constant $C > 1$. There is a streaming algorithm for the insertion-only adversarial model which, with probability $1 - \delta$, successfully outputs at each step $t \in [m]$ a value R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p$. The space used by the algorithm is $O\left(\frac{1}{\epsilon^2} \log n \log \delta^{-1}\right)$ bits.*

In addition, we show that for turnstile streams with bounded F_p *flip number* (defined formally in Section 7.2), efficient adversarially robust algorithms exist. Roughly speaking, the F_p flip number is the number of times that the F_p moment changes by a factor of $(1+\epsilon)$. Our algorithms have extremely small failure probability of $\delta = n^{-\lambda}$, and have optimal space among turnstile algorithms with this value of δ [JW13].

Theorem 75 (Robust F_p-Estimation in turnstile streams; See Theorem 84)**.** *Let $\mathcal{S}_\lambda$ be the set of all turnstile streams with F_p flip number at most $\lambda \geq \lambda_{\epsilon,m}(\|\cdot\|_p^p)$ for any $0 < p \leq 2$. Then there is an adversarially robust streaming algorithm for the class $\mathcal{S}_\lambda$ of streams that, with probability $1-n^{-C\lambda}$ for any constant $C > 0$, outputs at each time step a value R^t such that $R^t = (1\pm\epsilon)\|x\|_p^p$. The space used by the algorithm is $O(\epsilon^{-2}\lambda \log^2 n)$.*

The next result concerns F_p-estimation for $p > 2$. Here again, we provide an adversarially robust algorithm which is optimal up to a small multiplicative factor. This result applies the computation paths robustification method as a black box. Notably, a classic lower bound of [BYJKS04] shows that for $p > 2$, $\Omega(n^{1-2/p})$ space is required to estimate $\|x\|_p^p$ up to a constant factor (improved lower bounds have been provided since, e.g., [LW13, GW18]). By using our computation paths technique, we obtain adversarially robust F_p moment estimation algorithms

as well for $p > 2$.

Theorem 76 (Robust F_p-estimation for $p > 2$; see Theorem 85)**.** *Fix any $\epsilon > 0$, and fix any $p > 2$. There is a streaming algorithm for the insertion-only adversarial model which, with probability $1 - n^{-(c \log n)/\epsilon}$ for any constant $c > 1$, successfully outputs at each step a value R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p$ at every step $t \in [m]$. The space used by the algorithm is*

$$O\left(n^{1-2/p}\left(\epsilon^{-3}\log^2 n + \epsilon^{-6/p}\left(\log^2 n\right)^{2/p}\log n\right)\right)$$

Attack on AMS On the negative side, we demonstrate that the classic Alon-Matias-Szegedy sketch (AMS sketch) [AMS96], the first and perhaps most well-known F_2 estimation algorithm (which uses sub-polynomial space), is *not* adversarially robust. Specifically, we demonstrate an adversary which, when run against the AMS sketch, fools the sketch into outputting a value which is not a $(1 \pm \epsilon)$ estimate of the F_2. The non-robustness of standard static streaming algorithms, even under simple attacks, is a further motivation to design adversarially robust algorithms.

In what follows, recall that the AMS sketch computes $\mathbf{S}x$ throughout the stream, where $\mathbf{S} \in \mathbb{R}^{t \times n}$ is a matrix of uniform $\{t^{-1/2}, -t^{-1/2}\}$ random variables. The estimate of the F_2 is then the value $\|\mathbf{S}x\|_2^2$.

Theorem 77 (Attack on AMS sketch; see Theorem 92)**.** *Let $\mathbf{S} \in \mathbb{R}^{t \times n}$ be the AMS sketch, $1 \leq t \leq n/c$ for some constant $c > 1$. There is an adversary which, with probability $99/100$, succeeds in forcing the estimate $\|\mathbf{S}x\|_2^2$ of the AMS sketch to not be a $(1 \pm 1/2)$ approximation of the true norm $\|x\|_2^2$. Moreover, the adversary needs to only make $O(t)$ stream updates before this occurs.*

Heavy Hitters We also show how our techniques can be used to solve the popular *heavy-hitters* problem. Recall that the heavy-hitters problem tasks the streaming algorithm with returning a set S containing all coordinates i such that $|x_i| \geq \tau$, and containing no coordinates j such that $|x_j| < \tau/2$, for some threshold τ. Generally, the threshold τ is set to $\tau = \epsilon\|x\|_p$, which is known as the F_p heavy hitters guarantee.

For F_1 heavy hitters in insertion-only streams, a deterministic $O(\frac{1}{\epsilon}\log n)$ space algorithm exists [MG82]. However, for $p > 1$, specifically for the highly popular $p = 2$, things become more complicated. Note that since we can have $\|x\|_2 \ll \|x\|_1$, the F_2 guarantee is substantially stronger. For sketching-based turnstile algorithms, an $\Omega(n)$ lower bound for deterministic algorithms was previously known [Gan09]. Since $\|x\|_1 \leq \sqrt{n}\|x\|_2$, by setting $\epsilon = n^{-1/2}$, one can

obtain a deterministic $O(\sqrt{n}\log n)$ space insertion only F_2 heavy hitters algorithm. Recently, a lower bound of $\Omega(\sqrt{n})$ for deterministic insertion only algorithms was given, demonstrating the near tightness of this result [KPW20]. Thus, to develop a more efficient adversarially robust F_2 heavy hitters algorithm, we must employ randomness.

Indeed, by utilizing our sketch switching techniques, we demonstrate an adversarially robust F_2 heavy hitters (tracking) algorithm which uses only an $O(\epsilon^{-1}\log\epsilon^{-1})$ factor more space than the best known static F_2 heavy hitters tracking algorithm [BCI+17].

Theorem 78 (Robust F_2 heavy hitters: see Theorem 88)**.** *Fix any $\epsilon > 0$. There is a streaming algorithm in the adversarial insertion only model which solves the F_2 heavy hitters problem at every step $t \in [m]$ with probability $1 - n^{-C}$ (for any constant $C > 1$). The algorithm uses $O(\frac{\log\epsilon^{-1}}{\epsilon^3}\log^2 n)$ bits of space.*

Entropy Estimation Additionally, we demonstrate how our sketch switching techniques can be used to obtain robust algorithms for *empirical Shannon Entropy estimation*. Here, the Shannon Entropy $H(x)$ of the stream is defined via $H(x) = -\sum_{i=1}^{n}\frac{|x_i|}{\|x\|_1}\log\left(\frac{|x_i|}{\|x\|_1}\right)$. Our results follow from an analysis of the exponential of α-Renyi Entropy, which closely approximates the Shannon entropy, showing that the former cannot rapidly change too often within the stream. Our result is an adversarially robust algorithm with space complexity only a small polylogarithmic factor larger than the best known static algorithms [CC13, JW19].

Theorem 79 (Robust Entropy Estimation; see Theorem 90)**.** *There is an algorithm for ϵ-additive approximation of the Shannon entropy in the insertion-only adversarial streaming model using $O(\frac{1}{\epsilon^5}\log^4 n)$-bits of space in the random oracle model, and $O(\frac{1}{\epsilon^5}\log^6 n)$-bits of space in the general insertion only model.*

We remark that by making the same cryptographic assumption as in Theorem 72, we can remove the random oracle assumption in [JW19] for correctness of the entropy algorithm in the static case. Then, by applying the same techniques which resulted in Theorem 79, we can obtain the same stated bound for entropy with a cryptographic assumption instead of a random oracle assumption.

Bounded Deletion Streams Lastly, we show that our techniques for F_p moment estimation can be extended to the bounded deletion model of streaming (See Chapter 6, based on the model introduced in our paper [JW18a]). Rcall that, given some $\alpha \geq 1$, the model enforces the restriction that at all points $t \in [m]$ in the stream, we have $\|x^{(t)}\|_p^p \geq \frac{1}{\alpha}\|y^{(t)}\|_p^p$, where y is the frequency

vector of the stream with updates $u'_i = (a_i, \Delta'_i)$ where $\Delta'_i = |\Delta_i|$ (i.e., the absolute value stream). In other words, the stream does not delete off an arbitrary amount of the F_p weight that it adds over the course of the stream.

We demonstrate that bounded deletion streams have the desirable property of having a small flip number, which, as noted earlier, is a measurement of how often the F_p can change substantially (see Section 7.2 for a formal definition). Using this property and our sketch switching technique, we obtain the following.

Theorem 80 (F_p-estimation for bounded deletion; see Theorem 91)**.** *Fix $p \in [1, 2]$ and any constant $C > 1$. Then there is an adversarially robust F_p estimation algorithm which, with probability $1 - n^{-C}$, returns at each time step $t \in [m]$ an estimate R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p^p$. The space used by the algorithm is $O(\alpha\epsilon^{-(2+p)} \log^3 n)$.*

7.1.2 Prior Work on Adversarial Sketching

The need for studying adversarially robust streaming and sketching algorithms has been noted before in the literature. In particular, [GHR+12, GHS+12] motivate the adversarial model by giving applications and settings where it is impossible to assume that the queries made to a sketching algorithm are independent of the prior outputs of the algorithm, and the randomness used by the algorithm. One particularly important setting noted in [GHS+12] is when the *privacy* of the underlying data-set is a concern.

In response to this, in [HW13] the notion of adversarial robustness for *linear* sketching algorithms is studied. Namely, it is shown how any function $g : \mathbb{R}^n \to \mathbb{R}$, defined by $g(x) = f(\mathbf{A}x)$ for some $\mathbf{A} \in \mathbb{R}^{k\times n}$ and arbitrary $f : \mathbb{R}^k \to \mathbb{R}$ cannot approximate the F_2 moment $\|x\|_2^2$ of its input to an arbitrary polynomial factor in the presence of an adversary who is allowed to query $g(x_i)$ at polynomial many points (unless k is large). Since one can insert and delete off each x_i in a turnstile stream, this demonstrates a strong lower bound for adversarially robust turnstile linear sketching algorithms, at least when the stream updates are allowed to be real numbers.

We remark that other work has observed the danger inherent in allowing adversarial queries to a randomized sketch with only a static guarantee [AGM12a, AGM12c]. However, the motivation of these works is slightly different, and their setting not fully adversarial. In [MNS11], adversarial robustness of sketching in a distributed, *multi-player* model is considered, which is incomparable to the centralized streaming problem considered in this work. Finally, in [GGMW20], it was asked if there are randomized streaming algorithms whose output is independent of its ran-

domness, making such algorithms natural candidates for adversarial robustness; unfortunately a number of their results are negative, while their upper bounds do not apply to the problems studied here.

7.1.3 Subsequent Work and Open Questions

Since the introduction of the adversarial streaming model in our paper [BJWY20], several follow-up works have made progress in improving the complexity of several adversarially robust streaming problems, and answering several important questions about the model. Hassidim, Kaplan, Mansour, Matias, and Stemmer [HKM+20a] use techniques from differential privacy to obtain a generic robustification framework in the same mold as ours, where the dependency on the flip number is the improved $\sqrt{\lambda}$ as opposed to linear in λ, however this improvement is at the cost of additional $\mathrm{poly}((\log n)/\epsilon)$ factors. Similar to our construction, they run multiple independent copies of the static algorithm with independent randomness and feed the input stream to all of the copies. Unlike our construction, when a query comes, they aggregate the responses from the copies in a way that protects the internal randomness of each of the copies using differential privacy. Using their framework, one may construct an adversarially robust algorithm for F_p-moment estimation that uses $\widetilde{O}(\frac{\log^4 n}{\epsilon^{2.5}})$ bits of memory for any $p \in [0, 2]$. This improves over our $\widetilde{O}(\frac{\log n}{\epsilon^3})$ bound for some parameter regimes.

Woodruff and Zhang [WZ20b] obtain further improvements for specific problems which in some cases are (almost) optimal even for the static case. For example, they give an adversarially robust algorithm for F_p-moment estimation that uses $\widetilde{O}(\frac{\log n}{\epsilon^2})$ bits of memory for any $p \in [0, 2]$. This improves upon both our work and [HKM+20a]. Interestingly, the way they achieve this leads them to a new class of (classical) streaming algorithms they call difference estimators, which turn out to be useful also in the sliding windows (classical) model. Kaplan, Mansour, Nissim, and Stemmer [KMNS21] demonstrate the first separation between the static and adversarial model of streaming. Specifically, they show the existence of a (albeit highly-engineered) insertion-onlystreaming problem for which a polylogarithmic space static algorithm exists, but such that polynomial space is required for any adversarially robust algorithm.

Despite considerable progress having been made in a small amount of time, many problems remain open. Firstly, an important question is to achieve optimal bounds for important streaming problems in the insertion-only adversarial setting, such as moment estimation and heavy hitters. Furthermore, thus far nearly nothing is known about the turnstile model. Specifically, there are no known sublinear space streaming algorithms for any important turnstile streaming problem, such

as F_p estimation. On the other hand, proving lower bounds against the robustness of sublinear space algorithms is a non-trivial task, motivating the following question:

Do sublinear space turnstile streaming algorithms exist, such as for L_p estimation or finding the heavy hitters?

A negative answer to the above would represent a strong extension of the lower bound of [HW13], which held only against the restricted class of linear sketches. Moreover, it would represent a strong limitation on the robustness of sketching general data streams, and potentially motivate new models or additional restrictions on an adversary.

7.1.4 Tracking Algorithms

The robust streaming algorithms we design in this Chapter satisfy the *tracking* guarantee. Namely, they must output a response to a query at every step in time $t \in [m]$. For the case of estimation queries, this tracking guarantee is known as strong tracking.

Definition 7.1.1 (Strong tracking)**.** *Let $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ be the frequency vectors of a stream $(a_1, \Delta_1), \ldots, (a_m, \Delta_m)$, and let $g : \mathbb{R}^n \to \mathbb{R}$ be a function on frequency vectors. A randomized algorithm $\mathcal{A}$ is said to provide (ϵ, δ)-strong g-tracking if at each time step $t \in [m]$ it outputs an estimate R_t such that*

$$|R_t - g(x^{(t)})| \leq \epsilon |g(x^{(t)})|$$

for all $t \in [m]$ with probability at least $1 - \delta$.

In contrast, *weak tracking* replaces the error term $\epsilon|g(x^{(t)})|$ by $\max_{t' \in [m]} \epsilon \cdot |g(x^{(t')})|$. However, for the purposes of this paper, we will not need to consider weak tracking. We now state two results for strong tracking of F_p moments for $p \in [0, 2]$. Both results are for the static setting, i.e., for a stream fixed in advance (and not for the adaptive adversarial setting that we consider).

Lemma 7.1.2 ([BDN17])**.** *For $0 < p \leq 2$, there is an insertion only streaming algorithm which provides (ϵ, δ)-strong F_p-tracking using $O(\frac{\log n}{\epsilon^2}(\log \epsilon^{-1} + \log \delta^{-1} + \log \log n))$ bits of space.*

Lemma 7.1.3 ([Bła18])**.** *There is an insertion-only streaming algorithm which provides (ϵ, δ)-strong F_0-tracking using $O(\frac{\log \log n + \log \delta^{-1}}{\epsilon^2} + \log n)$ bits of space.*

7.1.5 Roadmap

In Section 7.2, we introduce our two general techniques for transforming static streaming algorithms into adversarially robust algorithms. In Section 7.3, we give our results on estimation of F_p moments, and in Section 7.4 we give our algorithms for adversarially robust distinct elements estimation. Next, in Section 7.5, we introduce our robust L_2 heavy hitters algorithm, and in Section 7.6 we give our entropy estimation algorithm. In Section 7.7, we provide our algorithms for F_p moment estimation in the bounded deletion model. In Section 7.8, we give our adversarial attack on the AMS sketch. Finally, in Section 7.9, we give our algorithm for optimal space distinct elements estimation under cryptographic assumptions.

7.2 Tools for Robustness

In this section, we establish two methods, *sketch switching* and *computation paths*, allowing one to convert an approximation algorithm for any sufficiently well-behaved streaming problem to an adversarially robust one for the same problem. The central definition of a *flip number*, bounds the number of major (multiplicative) changes in the algorithm's output along the stream. As we shall see, a small flip number allows for efficient transformation of non-robust algorithms into robust ones.[3]

7.2.1 Flip Number

Definition 7.2.1 (flip number)**.** *Let $\epsilon \geq 0$ and $m \in \mathbb{N}$, and let $\bar{y} = (y_0, y_1, \ldots, y_m)$ be any sequence of real numbers. The* ϵ-flip number $\lambda_\epsilon(\bar{y})$ *of $\bar{y}$ is the maximum $k \in \mathbb{N}$ for which there exist $0 \leq i_1 < \ldots < i_k \leq m$ so that $y_{i_{j-1}} \notin (1 \pm \epsilon) y_{i_j}$ for every $j = 2, 3, \ldots, k$.*

Fix a function $g \colon \mathbb{R}^n \to \mathbb{R}$ and a class $\mathcal{C} \subseteq ([n] \times \mathbb{Z})^m$ of stream updates. The (ϵ, m)-flip number $\lambda_{\epsilon,m}(g)$ of g over $\mathcal{C}$ is the maximum, over all sequences $((a_1, \Delta_1), \ldots, (a_m, \Delta_m)) \in \mathcal{C}$, of the ϵ-flip number of the sequence $=(y_0, y_1, \ldots, y_m)$ defined by $y_i = g(x^{(i)})$ for any $0 \leq i \leq m$, where as usual $x^{(i)}$ is the frequency vector after stream updates $(a_1, \Delta_1), \ldots, (a_i, \Delta_i)$ (and $x^{(0)}$ is the n-dimensional zeros vector).

[3]The notion of flip number we define here also plays a central role in subsequent works ([HKM+20a], [WZ20b]); for example, the main contribution of the former is a generic robustification technique with an improved (square root type instead of linear) dependence in the flip number. The latter improves the $\mathrm{poly}(1/\epsilon)$ dependence on the flip number.

The class $\mathcal{C}$ may represent, for instance, the subset of all insertion only streams, or bounded-deletion streams. For the rest of this section, we shall assume $\mathcal{C}$ to be fixed, and consider the flip number of g with respect to this choice of $\mathcal{C}$.[4]

Note that the flip number is clearly monotone in ϵ: namely $\lambda_{\epsilon',m}(g) \geq \lambda_{\epsilon,m}(g)$ if $\epsilon' < \epsilon$. One useful property of the flip number is that it is nicely preserved under approximations. As we show, this can be used to effectively construct approximating sequences whose 0-*flip number* is bounded as a function of the ϵ-flip number of the original sequence. This is summarized in the following lemma.

Lemma 7.2.2. *Fix $0 < \epsilon < 1$. Suppose that $\bar{u} = (u_0, \ldots, u_m)$, $\bar{v} = (v_0, \ldots, v_m)$, $\bar{w} = (w_0, \ldots, w_m)$ are three sequences of real numbers, satisfying the following:*

- *For any $0 \leq i \leq m$, $v_i = (1 \pm \epsilon/8)u_i$.*
- *$w_0 = v_0$, and for any $i > 0$, if $w_{i-1} = (1 \pm \epsilon/2)v_i$ then $w_i = w_{i-1}$, and otherwise $w_i = v_i$.*

Then $w_i = (1 \pm \epsilon)u_i$ for any $0 \leq i \leq m$, and moreover, $\lambda_0(\bar{w}) \leq \lambda_{\epsilon/8}(\bar{u})$.

In particular, if (in the language of Definition 7.2.1) $u_0 = g(x^{(0)}), u_1 = g(x^{(1)}), \ldots, u_m = g(x^{(m)})$ for a sequence of updates $((a_1, \Delta_1), \ldots, (a_m, \Delta_m)) \in \mathcal{C}$, then $\lambda_0(\bar{w}) \leq \lambda_{\epsilon/8,m}(g)$.

Proof. The first statement, that $w_i = (1 \pm \epsilon)u_i$ for any i, follows immediately since $v_i = (1 \pm \epsilon/8)u_i$ and $w_i = (1 \pm \epsilon/2)v_i$ and since $\epsilon < 1$. The third statement follows by definition from the second one. It thus remains to prove that $\lambda_0(\bar{w}) \leq \lambda_{\epsilon/8}(\bar{u})$.

Let $i_1 = 0$ and let $i_2, i_3, \ldots, i_k$ be the collection of all values $i \in [m]$ for which $w_{i-1} \neq w_i$. Note that $k = \lambda_0(\bar{w})$ and that $v_{i_{j-1}} = w_{i_{j-1}} = w_{i_{j-1}+1} = \cdots = w_{i_j - 1} \neq v_{i_j}$ for any $j = 2, \ldots, k$. We now claim that for every j in this range, $u_{i_{j-1}} \notin (1 \pm \epsilon/8)u_{i_j}$. This would show that $k \leq \lambda_{\epsilon/8}(\bar{u})$ and conclude the proof.

Indeed, fixing any such j, we either have $v_{i_{j-1}} > (1 + \epsilon/2)v_{i_j}$, or $w_{i_{j-1}} < (1 - \epsilon/2)v_{i_j}$. In the first case (assuming $u_{i_j} \neq 0$, as the case $u_{i_j} = 0$ is trivial),

$$\frac{u_{i_{j-1}}}{u_{i_j}} \geq \frac{v_{i_j}/(1 + \frac{\epsilon}{8})}{v_{i_{j-1}}/(1 - \frac{\epsilon}{8})} \geq \left(1 + \frac{\epsilon}{2}\right) \cdot \frac{1 - \frac{\epsilon}{8}}{1 + \frac{\epsilon}{8}} > 1 + \frac{\epsilon}{8}\,.$$

[4] A somewhat reminiscent definition, of an *unvarying algorithm*, was studied by [DNPR10] (see Definition 5.2 there) in the context of differential privacy. While their definition also refers to a situation where the output undergoes major changes only a few times, both the motivation and the precise technical details of their definition are different from ours.

In the second case, an analogous computation gives $u_{i_{j-1}}/u_{i_j} < 1 - \epsilon/8$. □

Note that the flip number of a function g critically depends on the model in which we work, as the maximum is taken over all sequences of *possible stream updates*; for insertion-only streams, the set of all such sequences is more limited than in the general turnstile model, and correspondingly many streaming problems have much smaller flip number when restricted to the insertion only model. We now give an example of a class of functions with bounded flip number.

Proposition 7.2.3. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be any monotone function, meaning that $g(x) \geq g(y)$ if $x_i \geq y_i$ for each $i \in [n]$. Assume further that $g(x) \geq T^{-1}$ for all $x > 0$, and $g(M \cdot \vec{1}) \leq T$, where M is a bound on the entries of the frequency vector and $\vec{1}$ is the all 1's vector. Then the flip number of g in the insertion only streaming model is $\lambda_{\epsilon,m}(g) = O(\frac{1}{\epsilon}\log T)$.*

Proof. To see this, note that $g(x^{(0)}) = 0$, $g(x^{(1)}) \geq T^{-1}$, and $g(x^{(m)}) \leq g(\vec{1} \cdot M) \leq T$. Since the stream has only positive updates, $g(x^{(0)}) \leq g(x^{(1)}) \leq \cdots \leq g(x^{(m)})$. Let $y_1, \ldots, y_k \in [m]$ be any set of points such that $g(x^{(y_i)}) < (1+\epsilon)g(x^{(y_{i+1})})$ for each i. Since there are at most $O(\frac{1}{\epsilon}\log T)$ powers of $(1+\epsilon)$ between T^{-1} and T, by the pigeonhole principle if $k > \frac{C}{\epsilon}\log(T)$ for a sufficiently large constant C, then at least two values must satisfy $(1+\epsilon)^j \leq g(x^{(y_i)}) \leq g(x^{(y_{i+1})}) \leq (1+\epsilon)^{j+1}$ for some j, which is a contradiction. □

Note that a special case of the above are the F_p moments of a data stream. Recall here $\|x\|_0 = |\{i : x_i \neq 0\}|$ is the number of non-zero elements in a vector x.

Corollary 7.2.4. *Let $p > 0$. Then the (ϵ, m)-flip number of $\|x\|_p^p$ in the insertion only streaming model is $\lambda_{\epsilon,m}(\|\cdot\|_p^p) = O(\frac{1}{\epsilon}\log m)$ for $p \leq 2$, and $\lambda_{\epsilon,m}(\|\cdot\|_p^p) = O(\frac{p}{\epsilon}\log m)$ for $p > 2$. For $p = 0$, we also have $\lambda_{\epsilon,m}(\|\cdot\|_0) = O(\frac{1}{\epsilon}\log m)$*

Proof. We have $\|\vec{0}\|_p^p = 0$, $\|z\|_p^p \geq 1$ for any non-zero $z \in \mathbb{Z}$, and $\|x^{(m)}\|_p^p \leq (Mm)^p n \leq n^{cp}$ for some constant c, where the second to last inequality holds because $\|x\|_\infty \leq Mm$ for where $M = \text{poly}(n)$ is the bound on the size of updates by assumption in the streaming model. Moreover, for $p = 0$ we have $\|x^{(m)}\|_0 \leq n$. The result then follows from applying Proposition 7.2.3 with $T = n^{c\cdot\max\{p,1\}}$. □

Another special case of Proposition 7.2.3 concerns the *cascaded norms* of insertion-only data streams [JW09]. Here, the frequency vector f is replaced with a matrix $\mathbf{A} \in \mathbb{Z}^{n\times d}$, which receives coordinate-wise updates in the same fashion, and the (p,k) cascaded norm of $\mathbf{A}$ is given by $\|\mathbf{A}\|_{(p,k)} = (\sum_i(\sum_j |\mathbf{A}_{i,j}|^k)^{p/k})^{1/p}$. In other words, $\|\mathbf{A}\|_{(p,k)}$ is the result of first taking the

L_k norm of the rows of $\mathbf{A}$, and then taking the F_p norm of the result. Proposition 7.2.3 similarly holds with $T = \text{poly}(n)$ in the insertion only model, and therefore the black-box reduction techniques introduced in the following sections are also applicable to these norms (using e.g., the cascaded algorithms of [JW09]).

Having a small flip number is very useful for robustness, as our next two robustification techniques demonstrate.

7.2.2 The Sketch Switching Technique

Our first technique is called *sketch switching*, and is described in Algorithm 15. The technique maintains multiple instances of a static strong tracking algorithm, where each time step only one of the instances is "active". The idea is to change the current output of the algorithm very rarely. Specifically, as long as the current output is a good enough multiplicative approximation of the estimate of the active instance, the estimate we give to the adversary does not change, and the current instance remains active. As soon as this approximation guarantee is not satisfied, we update the output given to the adversary, deactivate our current instance, and activate the next one in line. By carefully exposing the randomness of our multiple instances, we show that the strong tracking guarantee (which a priori holds only in the static setting) can be carried into the robust setting. By Lemma 7.2.2, the required number of instances, which corresponds to the 0-flip number of the outputs provided to the adversary, is controlled by the $(\Theta(\epsilon), m)$-flip number of the problem.

Algorithm 4: Adversarially Robust g-estimation by Sketch Switching

$\lambda \leftarrow \lambda_{\epsilon/8,m}(g)$
Initialize independent instances $A_1, \ldots, A_\lambda$ of $(\frac{\epsilon}{8}, \frac{\delta}{\lambda})$-strong g-tracking algorithm
$\rho \leftarrow 1$
$\tilde{g} \leftarrow g(\vec{0})$
while *new stream update* (a_k, Δ_k) **do**
    Insert update (a_k, Δ_k) into each algorithm $A_1, \ldots, A_\lambda$
    $y \leftarrow$ current output of A_ρ
    if $\tilde{g} \notin (1 \pm \epsilon/2)y$ **then**
        $\tilde{g} \leftarrow y$
        $\rho \leftarrow \rho + 1$
    Output estimate $\tilde{g}$

Lemma 7.2.5 (Sketch Switching)**.** *Fix any function $g : \mathbb{R}^n \to \mathbb{R}$ and let A be a streaming*

algorithm that for any $0 < \epsilon < 1$ and $\delta > 0$ uses space $L(\epsilon, \delta)$, and satisfies the $(\epsilon/8, \delta)$-strong g-tracking property on the frequency vectors $x^{(1)}, \ldots, x^{(m)}$ of any particular fixed stream. Then Algorithm 15 is an adversarially robust algorithm for $(1+\epsilon)$-approximating $g(x^{(t)})$ at every step $t \in [m]$ with success probability $1-\delta$, whose space is $O\left(L(\epsilon/8, \delta/\lambda) \cdot \lambda\right)$, where $\lambda = \lambda_{\epsilon/8,m}(g)$.

Proof. Note that for a fixed randomized algorithm $\mathcal{A}$ we can assume the adversary against $\mathcal{A}$ is deterministic without loss of generality (in our case, $\mathcal{A}$ refers to Algorithm 15). This is because given a randomized adversary and algorithm, if the adversary succeeds with probability greater than δ in fooling the algorithm, then by a simple averaging argument, there must exist a fixing of the random bits of the adversary which fools $\mathcal{A}$ with probability greater than δ over the coin flips of $\mathcal{A}$. Note also here that conditioned on a fixing of the randomness for both the algorithm and adversary, the entire stream and behavior of both parties is fixed.

We thus start by fixing such a string of randomness for the adversary, which makes it deterministic. As a result, suppose that y_i is the output of the streaming algorithm on step i. Then given $y_1, y_2, \ldots, y_k$ and the stream updates $(a_1, \Delta_1), \ldots, (a_k, \Delta_k)$ so far, the next stream update (a_{k+1}, Δ_{k+1}) is deterministically fixed. We stress that the randomness of the algorithm is not fixed at this point; we will gradually reveal it along the proof.

Let $\lambda = \lambda_{\epsilon/8,m}(g)$ and let $A_1, \ldots, A_\lambda$ be the λ independent instances of an $(\epsilon/8, \delta/\lambda)$-strong tracking algorithm for g. Since $\delta_0 = \delta/\lambda$, later on we will be able to union bound over the assumption that for all $\rho \in [\lambda]$, A_i satisfies strong tracking on some fixed stream (to be revealed along the proof); the stream corresponding to A_ρ will generally be different than that corresponding to ρ' for $\rho \neq \rho'$.

First, let us fix the randomness of the first instance, A_1. Let $u_1^1, u_2^1, \ldots, u_m^1$ be the updates $u_j^1 = (a_j, \Delta_j)$ that the adversary would make if $\mathcal{A}$ were to output $y_0 = g(\vec{0})$ at every time step, and let $x^{(t),1}$ be the stream vector after updates $u_1^1, \ldots, u_t^1$. Let $A_1(t)$ be the output of algorithm A_1 at time t of the stream $u_1^1, u_2^1, \ldots, u_t^1$. Let $t_1 \in [m]$ be the first time step such that $y_0 \notin (1 \pm \epsilon/2)A_1(t_1)$, if exists (if not we can set, say, $t_1 = m+1$). At time $t = t_1$, we change our output to $y_1 = A_1(t_1)$. Assuming that A_1 satisfies strong tracking for g with approximation parameter $\epsilon/8$ with respect to the fixed stream of updates $u_1^1, \ldots, u_m^1$ (which holds with probability at least $1 - \delta/\lambda$), we know that $A_1(t) = (1 \pm \epsilon/8)g(x^{(t)})$ for each $t < t_1$ and that $y_0 = (1 \pm \epsilon/2)A_1(t)$. Thus, by the first part of Lemma 7.2.2, $y_0 = (1 \pm \epsilon)g(x^{(t)})$ for any $0 \leq t < t_1$. Furthermore, by the strong tracking, at time $t = t_1$ the output we provide $y_1 = A_1(t_1)$ is a $(1 \pm \epsilon/8)$-approximation of the desired value $g(x^{(t_1)})$.

At this point, $\mathcal{A}$ "switches" to the instance A_2, and presents y_1 as its output as long as $y_1 =$

$(1 \pm \epsilon/2)A_2(t)$. Recall that randomness of the adversary is already fixed, and consider the sequence of updates obtained by concatenating $u_1^1, \ldots, u_{t_1}^1$ as defined above (these are the updates already sent by the adversary) with the sequence $u_{t_1+1}^2, \ldots, u_m^2$ to be sent by the adversary if the output from time $t = t_1$ onwards would always be y_1. We condition on the $\epsilon/8$-strong g-tracking guarantee on A_2 holding for this fixed sequence of updates, noting that this is the point where the randomness of A_2 is revealed. Set $t = t_2$ as the first value of t (if exists) for which $A_2(t) = (1 \pm \epsilon/2)y_1$ does not hold. We now have, similarly to above, $y_1 = (1 \pm \epsilon)g(x^{(t)})$ for any $t_1 \leq t < t_2$, and $y_2 = (1 \pm \epsilon/8)g(x^{(t_2)})$.

The same reasoning can be applied inductively for A_ρ, for any $\rho \in [\lambda]$, to get that (provided $\epsilon/8$-strong g-tracking holds for A_ρ) at any given time, the current output we provide to the adversary y_ρ is within a $(1 \pm \epsilon)$-multiplicative factor of the correct output for any of the time steps $t = t_\rho, t_\rho + 1, \ldots, \min\{t_{\rho+1} - 1, m\}$. Taking a union bound, we get that with probability at least $1 - \delta$, all instances provide $\epsilon/8$-tracking (each for its respective fixed sequence), yielding the desired $(1 \pm \epsilon)$-approximation of our algorithm.

It remains to verify that this strategy will succeed in handling all m elements of the stream (and will not exhaust its pool of algorithm instances before then). Indeed, this follows immediately from Lemma 7.2.2 applied with $\bar{u} = ((g(x^{(0)}), \ldots, g(x^{(m)}))$, $\bar{v} = (g(x^{(0)}), A_1(1), \ldots, A_1(t_1), A_2(t_1+1), \ldots, A_2(t_2), \ldots)$, and $\bar{w}$ being the output that our algorithm $\mathcal{A}$ provides ($y_0 = g(x^{(0)})$ until time $t_1 - 1$, then y_1 until time $t_2 - 1$, and so on). Observe that indeed $\bar{w}$ was generated from v exactly as described in the statement of Lemma 7.2.2. □

7.2.3 The Bounded Computation Paths Technique

With our sketch switching technique, we showed that maintaining multiple instances of a non-robust algorithm to estimate a function g, and switching between them when the rounded output changes, is a recipe for a robust algorithm to estimate g. We next provide another recipe, which keeps only one instance, whose success probability for any fixed stream is very high; it relies on the fact that if the flip number is small, then the total number of fixed streams that we should need to handle is also relatively small, and we will be able to union bound over all of them. Specifically, we show that any non-robust algorithm for a function with bounded flip number can be modified into an adversarially robust one by setting the failure probability δ small enough.

Lemma 7.2.6 (Computation Paths). *Fix $g \colon \mathbb{R}^n \to \mathbb{R}$ and suppose that the output of g uses $\log T$ bits of precision (see Remark 81). Let A be a streaming algorithm that for any $\epsilon, \delta > 0$ satisfies*

the (ϵ, δ)-strong g-tracking property on the frequency vectors $x^{(1)}, \ldots, x^{(m)}$ of any particular fixed stream. Then there is a streaming algorithm A' satisfying the following.

1. *A' is an adversarially robust algorithm for $(1+\epsilon)$-approximating $g(x^{(t)})$ in all steps $t \in [m]$, with success probability $1-\delta$.*
2. *The space complexity and running time of A' as above (with parameters ϵ and δ) are of the same order as the space and time of running A in the static setting with parameters $\epsilon/8$ and $\delta_0 = \delta / \left(\binom{m}{\lambda} T^{O(\lambda)}\right)$, where $\lambda = \lambda_{\epsilon/8,m}(g)$.*

Remark 81 (Bit precision of output)**.** For the purposes of this paper, we typically think of the bit precision as $O(\log n)$ (for example, in F_p-estimation, there are $\mathrm{poly}(n)$ possible outputs). Since we also generally assume that $m = \mathrm{poly}(n)$, the expression for δ_0 is of the form $\delta_0 = \delta / n^{\Theta(\lambda)}$ in this case. We note that while reducing the bit precision of the output slightly improves the bound on δ_0, this improvement becomes negligible for any streaming algorithm whose dependence in the error probability δ is logarithmic or better; this covers all situations where we apply Lemma 7.2.6 in this paper.

Proof. The algorithm A' that we construct runs by emulating A with the above parameters, and assuming that the output sequence of the emulated A up to the current time t is $v_0, \ldots, v_t$, it generates w_t in exactly the way described in Lemma 7.2.2: set $w_0 = v_0$, and for any $i > 0$, if $w_{i-1} \in (1 \pm \epsilon/2) v_i$ then $w_i = w_{i-1}$, and otherwise $w_i = v_i$. The output provided to the adversary at time t would then be w_t.

As in the proof of Lemma 7.2.5, we may assume the adversary to be deterministic. This means, in particular, that the output sequence we provide to the adversary fully determines its stream of updates $(a_1, \Delta_1), \ldots, (a_m, \Delta_m)$. Take $\lambda = \lambda_{\epsilon/8,m}(g)$. Consider the collection of all possible output sequences (with $\log T$ bits of precision) whose 0-flip number is at most λ, and note that the number of such sequences is at most $\binom{m}{\lambda} T^{O(\lambda)}$. Each output sequence as above uniquely determines a corresponding stream of updates for the deterministic adversary; let $\mathcal{S}$ be the collection of all such streams.

Pick $\delta_0 = \delta / |\mathcal{S}|$. Taking a union bound, we conclude that with probability $1-\delta$, A (instantiated with parameters $\epsilon/8$ and δ_0) provides an $\epsilon/8$-strong g-tracking guarantee for all streams in $\mathcal{S}$. We fix the randomness of A, and assume this event holds.

At this point, the randomness of both parties has been revealed, which determines an output sequence $v_0, \ldots, v_m$ for the emulated A and the edited output, $w_0, \ldots, w_m$, that our algorithm A' provided to the adversary. The proof now follows by induction over the number t of stream

updates that have been seen. The inductive statement is the following:

1. The sequence of outputs that the emulated algorithm A generates in response to the stream updates up to time t, $v_0, \ldots, v_t$, is a $(1 \pm \epsilon/8)$-approximation of g over the stream up to that time.
2. The sequence of outputs that the adversary receives from A' until time t, $(w_0, \ldots, w_t)$, has 0-flip number at most λ (and in particular, is a prefix of a sequence in $\mathcal{S}$).

The base case, $t = 0$, is obvious; and the induction step follows immediately from Lemma 7.2.2. □

7.3 Robust F_p-Estimation

In this section, we introduce our adversarially robust F_p moment estimation algorithms. Recall that F_p is given by $\|x\|_p^p = \sum_i |x_i|^p$ for $p > 0$. For $p = 0$, the F_0 norm, or the number of distinct elements, is the number of non-zero coordinates in f, that is, $\|x\|_0 = |\{i \in [n] : x_i \neq 0\}|$. Recall that in Corollary 7.2.4, we bounded the flip number of the F_p moment in insertion only streams for any fixed $p > 0$ by $O(p\epsilon^{-1} \log n)$. By using our sketch switching argument, the strong F_p tracking guarantees of [BDN17] as stated in Lemma 7.1.2, we obtain our first result for $0 < p \leq 2$.

Theorem 82 (F_p-estimation by sketch switching)**.** *Fix any $0 < \epsilon, \delta \leq 1$ and $0 < p \leq 2$. There is a streaming algorithm for the insertion-only adversarial model which, with probability $1 - \delta$, successfully outputs at each step $t \in [m]$ a value R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p$. The space used by the algorithm is*

$$O\left(\frac{1}{\epsilon^3} \log n \log \epsilon^{-1} (\log \epsilon^{-1} + \log \delta^{-1} + \log \log n)\right)$$

Proof. By an application of Lemma 7.2.5 along with the flip number bound of Corollary 7.2.4 and the strong tracking algorithm of Lemma 7.1.2, we immediately obtain a space complexity of

$$O\left(\frac{1}{\epsilon^3} \log^2 n (\log \epsilon^{-1} + \log \delta^{-1} + \log \log n)\right)$$

We now describe how the factor of $\frac{1}{\epsilon} \log n$, coming from running $\lambda_{\epsilon,m} = \Theta(\frac{1}{\epsilon} \log n)$ independent sketches in Lemma 7.2.5, can be improved to $\frac{1}{\epsilon} \log \epsilon^{-1}$.

To see this, we change Algorithm 15 in the following way. Instead of $\Theta(\frac{1}{\epsilon}\log n)$ independent sketches, we use $\lambda \leftarrow \Theta(\frac{1}{\epsilon}\log \epsilon^{-1})$ independent sketches, and change line 10 to state $\rho \leftarrow \rho + 1 \pmod{\lambda}$. Each time we change ρ to $\rho + 1$ and begin using the new sketch $A_{\rho+1}$, we *completely restart* the algorithm A_ρ with new randomness, and run it on the remainder of the stream. The proof of correctness in Lemma 7.2.5 is completely unchanged, except for the fact that now A_ρ is run only on a *suffix* $a_j, a_{j+1}, \ldots,$ of the stream, if j is the time step where A_ρ is reinitialized. Specifically, at each time step $t \geq j$, A_ρ will produce a $(1 \pm \epsilon)$ estimate of $\|x^{(t)} - x^{(j-1)}\|_p$ instead of $\|x^{(t)}\|_p$. However, since the sketch will not be used again until a time step t' where $\|x^{(t)}\|_p \geq (1+\epsilon)^\lambda \|x^{(j)}\|_p = \frac{100}{\epsilon}\|x^{(j)}\|_p$, it follows that only an ϵ fraction of the ℓ_p mass was missed by A_ρ. In particular, $\|x^{(t')} - x^{(j-1)}\|_p = (1 \pm \epsilon/100)\|x^{(t')}\|_p$, and thus by giving a $(1 \pm \epsilon/10)$ approximation of $\|x^{(t')} - x^{(j-1)}\|_p$, the algorithm A_ρ gives the desired $(1 \pm \epsilon)$ approximation of the underlying F_p moment, which is the desired result after a constant factor rescaling of ϵ. Note that this argument could be used for the F_0 moment, or any p-th moment for $p \geq 0$, using a F_p strong tracking algorithm for the relevant p. □

While for most values of δ, the above theorem has better space complexity than the computation paths reduction, for the regime of very small failure probability it is actually preferable to use the latter, as we now state.

Theorem 83 (F_p-estimation for small δ)**.** *Fix any $0 < \epsilon < 1$, $0 < p \leq 2$, and $\delta < n^{-C\frac{1}{\epsilon}\log n}$ for a sufficiently large constant $C > 1$. There is a streaming algorithm for the insertion-only adversarial model which, with probability $1 - \delta$, successfully outputs at each step $t \in [m]$ a value R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p$. The required space is $O\left(\frac{1}{\epsilon^2}\log n \log \delta^{-1}\right)$ bits.*

The proof is a direct application of Lemma 7.2.6, along with the flip number bound of Corollary 7.2.4, and the $O(\epsilon^{-2}\log n \log \delta^{-1})$ static F_p estimation algorithm of [KNW10a]. Indeed, note that the flip number is $\lambda = O(\log n/\epsilon)$ and that for small enough values of δ as in the lemma, one has $\log(m^\lambda/\delta) = \Theta(\log(1/\delta))$.

We remark that the below complexity is optimal up to a $\log n$ term for insertion only streams. The reason is that the $O(1/\epsilon^2 \log n \log 1/\delta)$ lower bound of [JW13] degrades to $O(1/\epsilon^2 \log 1/\delta)$ when deletions are not allowed.[5]

Next, we show that for turnstile streams with F_p flip number λ, we can estimate F_p with error probability $\delta = n^{-\lambda}$. The space requirement of the algorithm is optimal for algorithms with

[5]Specifically, one may not apply the augmented indexing step in the reduction of [JW13] to delete off coordinates in $\log n$ levels, which loses this factor in the lower bound.

such failure probability δ, which follows by an $\Omega(\epsilon^{-2}\log n\log\delta^{-1})$ lower bound for turnstile algorithms [JW13], where the hard instance in question has small F_p flip number.[6]

Theorem 84 (F_p-estimation for λ-flip number turnstile streams)**.** *Let $\mathcal{S}_\lambda$ be the set of all turnstile streams with F_p flip number at most $\lambda \geq \lambda_{\epsilon,m}(\|\cdot\|_p^p)$ for any $0 < p \leq 2$. Then there is an adversarially robust streaming algorithm for the class $\mathcal{S}_\lambda$ of streams that, with probability $1-n^{-C\lambda}$ for any constant $C > 0$, outputs at each time step a value R^t such that $R^t = (1\pm\epsilon)\|x\|_p^p$. The space used by the algorithm is $O(\epsilon^{-2}\lambda\log^2 n)$.*

Proof. The proof follows by simply applying Lemma 7.2.6, along with the $O(\epsilon^{-2}\log n\log\delta^{-1})$ bit turnstile algorithm of [KNW10a]. □

In addition, we show that the F_p moment can also be robustly estimated for $p > 2$. In this case, it is preferable to use our computation paths reduction, because the upper bounds for F_p moment estimation for large p yield efficiency gains when setting δ to be small.

Theorem 85 (F_p-estimation, $p > 2$, by Computation Paths)**.** *Fix any $\epsilon, \delta > 0$, and any constant $p > 2$. Then there is a streaming algorithm for the insertion-only adversarial model which, with probability $1 - n^{-(c\log n)/\epsilon}$ for any constant $c > 1$, successfully outputs at every step $t \in [m]$ a value R^t such that $R^t = (1\pm\epsilon)\|x^{(t)}\|_p$. The space used by the algorithm is $O(n^{1-2/p}(\epsilon^{-3}\log^2 n + \epsilon^{-6/p}(\log^2 n)^{2/p}\log n))$.*

Proof. We use the insertion only F_p estimation algorithm of [GW18], which achieves

$$\left(n^{1-2/p}\left(\epsilon^{-2}\log\delta^{-1} + \epsilon^{-4/p}\log^{2/p}\delta^{-1}\log n\right)\right)$$

bits of space in the turnstile (and therefore insertion only) model. We can set $\delta = \delta/m$ to union bound over all steps, making it a strong F_p tracking algorithm with

$$O\left(n^{1-2/p}\left(\epsilon^{-2}\log(n\delta^{-1}) + \epsilon^{-4/p}\log^{2/p}(n\delta^{-1})\log n\right)\right)$$

bits of space. Then by Lemma 7.2.6 along with the flip number bound of Corollary 7.2.4, the claimed space complexity follows. □

[6]The hard instance in [JW13] is a stream where $O(n)$ updates are first inserted and then deleted, thus the flip number is at most twice the F_p flip number of an insertion only stream.

7.4 Robust Distinct Elements Estimation

We now demonstrate how our sketch switching technique can be used to estimate the number of *distinct elements*, also known as F_0 estimation, in an adversarial stream. In this case, since there exist static F_0 strong tracking algorithms [Bła18] which are more efficient than repeating the sketch $\log \delta^{-1}$ times, it will be preferable to use our sketch switching technique.

Theorem 86 (Robust Distinct Elements by Sketch Switching)**.** *There is an algorithm which, when run on an adversarial insertion only stream, produces at each step $t \in [m]$ an estimate R^t such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_0$ with probability at least $1 - \delta$. The space used by the algorithm is $O(\frac{\log \epsilon^{-1}}{\epsilon}(\frac{\log \epsilon^{-1} + \log \delta^{-1} + \log\log n}{\epsilon^2} + \log n))$ bits.*

Proof. We use the insertion only distinct elements *strong* tracking algorithm of [Bła18]. Specifically, the algorithm of [Bła18] uses space $O(\frac{\log \delta_0^{-1} + \log\log n}{\epsilon^2} + \log n)$, and with probability $1 - \delta_0$, successfully returns an estimate R^t for every step $t \in [m]$ such that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_0$ in the non-adversarial setting. Then by application of Lemma 7.2.5, along with the flip number bound of $O(\log n/\epsilon)$ from Corollary 7.2.4, we obtain the space complexity with a factor of $\frac{\log n}{\epsilon}$ blow-up after setting $\delta_0 = \Theta(\delta \frac{\epsilon}{\log n})$. This gives a complexity of $O(\frac{\log n}{\epsilon}(\frac{\log \epsilon^{-1} + \log \delta^{-1} + \log\log n}{\epsilon^2} + \log n))$. To reduce the extra $\log n$-factor to a $\log \epsilon^{-1}$ factor, we just apply the same argument used in the proof of Theorem 82, which shows that by restarting sketches it suffices to keep only $O(\epsilon^{-1} \log \epsilon^{-1})$ copies. □

7.4.1 Fast and Robust Distinct Elements Estimation

As noted earlier, there are many reasons why one may prefer one of the reductions from Section 7.2 to the other. In this section, we will see such a motivation. Specifically, we show that adversarially robust F_0 estimation can be accomplished with extremely fast update time using the computation paths reduction of Lemma 7.2.6.

First note that the standard approach to obtaining failure probability δ is to repeat the estimation algorithm $\log \delta^{-1}$ times independently, and take the median output. However, this blows up the update time by a factor of $\log \delta^{-1}$. Thus black-box applying Lemma 7.2.6 by setting δ to be small can result in a larger update time. To improve upon this, we will introduce an insertion only distinct elements estimation algorithm, with the property that the runtime dependency on δ^{-1} is very small (roughly $\log^2 \log \delta^{-1}$). Thus applying Lemma 7.2.6 on this algorithm results in a very fast robust streaming algorithm.

Lemma 7.4.1. *There is a streaming algorithm which, with probability $1-\delta$, returns a $(1\pm\epsilon)$ multiplicative estimate of the number of distinct elements in an insertion only data stream. The space required is $O(\frac{1}{\epsilon^2}\log n(\log\log n+\log\delta^{-1}))$,[7] and the worst case running time per update is $O\left(\left(\log^2\log\frac{\log n}{\delta}\right)\cdot\left(\log\log\log\frac{\log n}{\delta}\right)\right)$.*

Before presenting our proof of Lemma 7.4.1, we state the following proposition which will allow for the fast evaluation of d-wise independent hash functions.

Proposition 7.4.2 ([vzGG13], Ch. 10)**.** *Let R be a ring, and let $p\in R[x]$ be a degree d univariate polynomial over R. Then given distinct $x_1,x_2,\ldots,x_d\in R$, all the values $p(x_1),p(x_2),\ldots,p(x_d)$ can be computed using $O(d\log^2 d\log\log d)$ operations over R.*

of Lemma 7.4.1. We describe the algorithm here, as stated in Algorithm 5.

Algorithm 5: Fast non-adversarial distinct elements estimation.

```
Initialize Lists L_0, L_1, ..., L_t ← ∅, for t = Θ(log n)
B ← Θ(ε^{-2} log δ^{-1}),  d ← Θ(log log n + log δ^{-1})
Initialize d-wise independent hash function H : [n] → [2^ℓ] such that n^2 ≤ 2^ℓ ≤ n^3.
while Receive update (a_t, Δ_t) ∈ [n] × ℤ do
    Let j be such that 2^{ℓ-j-1} ≤ H(a_t) < 2^{ℓ-j}
    if L_j has not been deleted then
        Add a_t to the list L_j if it is not already present.
    If |L_j| > B for any j, delete the list L_j, and never add any items to it again.
Let i be the largest index such that |L_i| ≥ (1/5)B.
Return 2^{i+1}|L_i| as the estimate of ||x||_0
```

We initialize lists $L_0,L_1,\ldots,L_t\leftarrow\emptyset$, for $t=\Theta(\log n)$, and hash functions $H:[n]\to[2^\ell]$, where ℓ is set so that $n^2\le 2^\ell\le n^3$. The lists L_i will store a set of identities $L_i\subset[n]$ which have occurred in the stream. We also set $B\leftarrow\Theta(\frac{1}{\epsilon^2}(\log\log n+\log\delta^{-1}))$. For now, assume that H is fully independent.

At each step when we see an update $a_i\in[n]$ (corresponding to an update which increments the value of x_i by one), we compute j such that $2^{\ell-j-1}\le H(a_i)\le 2^{\ell-j}$. Note that this event occurs with probability $2^{-(j+1)}$. Then we add the $O(\log n)$-bit identity a_i to the list L_j if $|L_j|<B$. Once $|L_k|=B$ for any $k\in[t]$, we delete the entire list L_k, and never add an item to L_k

[7] We remark that it is possible to optimize the $\log n$ factor to $O(\log\delta^{-1}+\log\epsilon^{-1}+\log\log n)$ by hashing the identities stored in the lists of the algorithm to a domain of size $\mathrm{poly}(\delta^{-1},\epsilon^{-1},\log n)$. However, in our application we will be setting $\delta\ll 1/n$, and so the resulting adversarially robust algorithm would actually be *less* space efficient.

again. We call such a list L_k *saturated*. At the end of the stream, we find the largest value i such that $\frac{1}{5}B \leq |L_i|$, and output $2^{i+1}|L_i|$ as our estimate of $\|x\|_0$.

We now analyze the above algorithm. Let i_0 be the smallest index such that $\mathbf{E}\left[|L_{i_0}|\right] \leq \|x\|_0 2^{-(i_0+1)} < \frac{1}{5(1+\epsilon)}B$. Note here that $\mathbf{E}\left[|L_k|\right] = 2^{-(k+1)}\|x\|_0$ for any $k \in [t]$. By a Chernoff bound, with probability $1 - \exp(-\Omega(-\epsilon^2 B)) < 1 - \delta^2/\log(n)$ we have that $|L_{i_0}| < \frac{1}{5}B$. We can then union bound over all such indices $i \geq i_0$. This means that we will not output the estimate used from any index $i \geq i_0$. Similarly, by a Chernoff bound we have that $|L_{i_0-1}| = (1 \pm \epsilon)\|x\|_0 2^{-i_0} < \frac{2}{5}B$ and $|L_{i_0-2}| = (1 \pm \epsilon)\|x\|_0 2^{-i_0+1}$, and moreover we have $\frac{2}{5(1+\epsilon)}B \leq \|x\|_0 2^{-i_0+1} \leq \frac{4}{5}B$, meaning that the output of our algorithm will be either $|L_{i_0-1}|2^{i_0}$ or $|L_{i_0-2}|2^{i_0-1}$, each of which yields a $(1 \pm \epsilon)$ estimate. Now note that we cannot store a fully independent hash function H, but since we only needed all events to hold with probability $1 - \Theta(\delta^2/\log(n))$, it suffices to choose H to be a d-wise independent hash function for $d = O(\log\log n + \log\delta^{-1})$, which yields Chernoff-style tail inequalities with a decay rate of $\exp(-\Omega(d))$ (see e.g. Theorem 5 of [SSS95]).

Next we analyze the space bound. Trivially, we store at most $O(\log n)$ lists L_i, each of which stores at most B identities which require $O(\log n)$ bits each to store, yielding a total complexity of $O(\frac{1}{\epsilon^2}\log^2 n(\log\log n + \log\delta^{-1}))$. We now show however that at any given step, there are at most $O(B\log\log n)$ many identities stored in all of the active lists. To see this, let $i_0 < i_1 < \cdots < i_s$ be the time steps such that $\|x^{(i_j)}\|_0 = 2^{j+1} \cdot B$, and note that $s \leq \log(n) + 1$. Note that before time i_0, at most B identities are stored in the union of the lists. First, on time step i_j for any $j \in [s]$, the expected size of $|L_{j-2}|$ is at least $2|B|$ (had we never deleted saturated lists), and, with probability $1 - (\delta/\log n)^{10}$ after a union bound, it holds that $|L_{j'}|$ is saturated for all $j' \leq j - 2$. Moreover, note that the expected number of identities written to lists $L_{j'}$ with $j' \geq j - 1$ is $\|x^{(i_j)}\|_0 \sum_{\nu \geq 1} 2^{-j+1+\nu} \leq 2B$, and is at most $4B$ with probability at least $1 - (\delta/\log n)^{10}$ (using the d-wise independence of H). We conclude that on time step t_j, the total space being used is $O(B\log n)$ with probability at least $1 - (\delta/\log n)^{10}$, so we can union bound over all such steps i_j for $j \in [s]$.

Next, we must analyze the space usage at steps τ for $i_j < \tau < i_{j+1}$. Note that the number of new distinct items which occur over all such time steps τ is at most $2^{j+1} \cdot B$ by definition. Since we already conditioned on the fat that $|L_{j'}|$ is saturated for all $j' \leq j - 2$, it follows that each new item is written into a list with probability at most 2^{-j}. Thus the expected number of items which are written into lists within times τ satisfying $i_j < \tau < i_{j+1}$ is $2^j \cdot B \cdot 2^{-j} = B$ in expectation, and at most $4B$ with probability $1 - (\delta/\log n)^{10}$ (again using the d-wise Independence of H). Conditioned on this, the total space used in these steps is at most

$O(B \log n) = O(\frac{1}{\epsilon^2} \log n(\log \log n + \log \delta))$ in this interval, and we then can union bound over all such $O(\log n)$ intervals, which yields the desired space.

Finally, for the update time, note that at each stream update $a_i \in [n]$ the first step of the algorithm. Naïvely, computing a d-wise independent hash function requires $O(d)$ arithmetic operations (in the standard RAM model), because H in this case is just a polynomial of degree d over $\mathbb{Z}$. On the other hand, we can batch sequences of $d = O(\log \log n + \log \delta^{-1})$ computations together, which require an additive $O(d \log n) = O(\log n(\log \log n + \log \delta^{-1}))$ bits of space at any given time step to store (which is dominated by the prior space complexity). Then by Proposition 7.4.2, all d hash function evaluations can be carried out in $O(d \log^2 d \log \log d) = O(d \log^2(\log \frac{\log n}{\delta}) \log \log \log \log \frac{n}{\delta})$ time. The work can then be evenly distributed over the following d steps, giving a worst case update time of $O(\log^2(\log \frac{\log n}{\delta}) \log \log \log \log \frac{n}{\delta})$. Note that this delays the reporting of the algorithm for the contribution of updates by a total of d steps, causing an additive d error. However, this is only an issue if $d \geq \epsilon \|x\|_0$, which occurs only when $\|x\|_0 \geq \frac{1}{\epsilon} d$. Thus for the first $D = O(\epsilon^{-1} d)$ distinct items, we can store the non-zero items exactly (and deterministically), and use the output of this deterministic algorithm. The space required for this is $O(\epsilon^{-1} \log(n)(\log \log n + \log \delta^{-1})$, which is dominated by the space usage of the algorithm overall. After D distinct items have been seen, we switch over to using the output of the randomized algorithm described here. Finally, the only other operation involves adding an identity to at most one list per update, which is $O(1)$ time, which completes the proof. □

We can use the prior result of Lemma 7.4.1, along with our argument for union bounding over adversarial computation paths of Lemma 7.2.6 and the flip number bound of Corollary 7.2.4, which results in an adversarially robust streaming algorithm for distinct elements estimation with extremely fast update time.

Theorem 87. *There is a streaming algorithm which, with probability $1 - n^{-(C/\epsilon) \log n}$ for any constant $C \geq 1$, when run on an adversarial chosen insertion-only data stream, returns a $(1 \pm \epsilon)$ multiplicative estimate of the number of distinct elements at every step in the stream. The space required is $O(\frac{1}{\epsilon^3} \log^3 n)$, and the worst case running time is $O\left(\left(\log^2 \log \frac{\log n}{\epsilon}\right) \cdot \left(\log \log \log \frac{\log n}{\epsilon}\right)\right)$ per update.*

7.5 Robust Heavy Hitters

In this section, we study robust algorithms for the heavy-hitters problem. Recall that the heavy hitters problem tasks the algorithm with recovering the most frequent items in a data-set. Stated simply, the goal is to report a list S of items x_i that appear least τ times, meaning $x_i \geq \tau$, for a given threshold τ. Generally, τ is parameterized in terms of the L_p norm of the frequency vector x, so that $\tau = \epsilon\|x\|_p$. For $p > 2$, this problem is known to take polynomial space [AMS96, BYJKS04]. Thus, the strongest such guarantee that can be given in sub-polynomial space is known as the L_2 guarantee:

Definition 7.5.1. *A streaming algorithm is said to solve the (ϵ, δ)-heavy hitters problem with the L_2 guarantee if the algorithm, when run on a stream with frequency vector $x \in \mathbb{R}^n$, outputs a set $S \subset [n]$ such that with probability $1 - \delta$ the following holds: for every $i \in [n]$ if $|x_i| \geq \epsilon\|x\|_2$ then $i \in S$, and if $|x_i| \leq (\epsilon/2)\|x\|_2$ then $i \notin S$.*

We also introduce the related task of (ϵ, δ)-point queries.

Definition 7.5.2. *A streaming algorithm is said to solve the (ϵ, δ) point query problem with the L_2 guarantee if with probability $1 - \delta$, at every time step $t \in [m]$, for each coordinate $i \in [n]$ it can output an estimate $\widehat{x}_i^t$ such that $|\widehat{x}_i^t - x_i^{(t)}| \leq \epsilon\|x^{(t)}\|_2$. Equivalently, it outputs a vector $\widehat{x}^t \in \mathbb{R}^n$ such that $\|x^{(t)} - \widehat{x}^t\|_\infty \leq \epsilon\|x^{(t)}\|_2$.*[8]

Notice that for any algorithm that solves the (ϵ, δ)-point query problem, if it also has estimates $R^t = (1 \pm \epsilon/10)\|x^{(t)}\|_2$ at each time step $t \in [m]$, then it immediately gives a solution to the (ϵ, δ)-heavy hitters problem by just outputting all $i \in [n]$ with $\tilde{x}_i^t > (3/4)\epsilon R^t$. Thus solving (ϵ, δ)-point queries, together with F_2 tracking, is a stronger property. In the following, we say that $\widehat{x}^t$ is ϵ-correct at time t if $\|x^{(t)} - \widehat{x}^t\|_\infty \leq \epsilon\|x^{(t)}\|_2$.

In this section, we demonstrate how this fundamental task of point query estimation can be accomplished robustly in the adversarial setting. Note that we have already shown how F_2 tracking can be accomplished in the adversarial model, so our focus will be on point queries. Our algorithm relies on a similar sketch switching technique as used in Lemma 7.2.5, which systematically hides randomness from the adversary by only publishing a new estimate $\widehat{x}^t$ when absolutely necessary. To define what is meant by "absolutely necessary", we will first need the following proposition.

[8] We note that a stronger form of error is possible, called the *tail guarantee*, which does not count the contribution of the top $1/\epsilon^2$ largest coordinates to the error $\epsilon\|x\|_2$. We restrict to the simpler version of the L_2 guarantee.

Proposition 7.5.3. *Suppose that $\widehat{x}^t \in \mathbb{R}^n$ is ϵ-correct at time t on an insertion only stream, and let $t_1 > t$ be any time step such that $\|x^{(t_1)} - x^{(t)}\|_2 \leq \epsilon \|x^{(t)}\|_\infty$. Then $\widehat{x}^t$ is 2ϵ-correct at time t_1.*

Proof. $\|\widehat{x}^t - x^{(t_1)}\|_\infty \leq \|\widehat{x}^t - x^{(t)}\|_\infty + \|x^{(t_1)} - x^{(t)}\|_\infty \leq \epsilon \|x^{(t)}\|_2 + \epsilon \|x^{(t)}\|_2 \leq 2\epsilon \|x^{(t_1)}\|_2$. □

To prove the main theorem of Section 4.3.2, we will need the classic *count-sketch* algorithm for finding L_2 heavy hitters [CCFC02b] (see Section 2.3.2), which solves the more general point query problem in the static setting with high probability.

Lemma 7.5.4 ([CCFC02b])**.** *There is a streaming algorithm in the non-adversarial insertion only model which solves the (ϵ, δ)-point query problem, using $O(\frac{1}{\epsilon^2} \log n \log \frac{n}{\delta})$ bits of space.*

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

Theorem 88 (L_2 point query and heavy hitters)**.** *Fix any $\epsilon, \delta > 0$. There is a streaming algorithm in the adversarial insertion only model which solves the (ϵ, n^{-C}) point query problem, and also the $O(\epsilon, n^{-C})$-heavy hitters problem, for any constant $C > 1$. The algorithm uses $O(\frac{\log \epsilon^{-1}}{\epsilon^3} \log^2 n)$ bits of space.*

Proof. Since we already know how to obtain estimates $R^t = (1 \pm \epsilon/100)\|x^{(t)}\|_2$ at each time step $t \in [m]$ in the adversarial insertion only model within the required space, it will suffice to show that we can obtain estimates $\widehat{x}^t$ which are ϵ-correct at each time step t (i.e., it will suffice to solve the point query problem).

Let $1 = t_1, t_2, \ldots, t_T = m$ for $T = \Theta(\epsilon^{-1} \log n)$ be any set of time steps such that $\|x^{(a_{t+1})} - x^{(a_t)}\|_2 \leq \epsilon \|x^{(a_t)}\|_2$ for each $i \in [T-1]$. Then by Proposition 7.5.3, we know that if we output an estimate $\widehat{x}^i$ on time t_i which is ϵ-correct for time t_i, then $\widehat{x}^i$ will still be 2ϵ correct at time t_{i+1}. Thus our approach will be to output vectors $\widehat{x}^1, \widehat{x}^2, \ldots, \widehat{x}^T$, such that we output the estimate $\widehat{x}^i \in \mathbb{R}^n$ at all times τ such that $t_i \leq \tau < t_{i+1}$, and such that $\widehat{x}^i$ is ϵ-correct for time t_i.

First, to find the time steps t_i, we run the adversarially robust F_2 estimator of Theorem 82, which gives an estimate $R^t = (1 \pm \epsilon/100)\|x^{(t)}\|_2$ at each time step $t \in [m]$ with probability $1 - n^{-C}$ for any constant $C > 1$, and uses space $O(\epsilon^{-3} \log^2 n \log \epsilon^{-1})$. Notice that this also gives the required estimates R^t as stated above. By rounding down the outputs R^t of this F_2 estimation algorithm to the nearest power of $(1 + \epsilon/2)$, we obtain our desired points t_i. Notice that this also gives $T = \Theta(\epsilon^{-1} \log n)$ as needed, by the flip number bound of Corollary 7.2.4.

Next, to obtain the desired ϵ point query estimators at each time step t_i, we run T independent copies of the point query estimation algorithm of Lemma 7.5.4. At time t_i, we use the output

vector of the i-th copy as our estimate $\widehat{x}^i$, which will also be used without any modification on all times τ with $t_i \leq \tau < t_{i+1}$. Since each copy of the algorithm only reveals any of its randomness at time t_i, at which point it is never used again, by the same argument as Lemma 7.2.5 it follows that each $\widehat{x}^i$ will be ϵ-correct for time t_i. Namely, since the set of stream updates on times $1, 2, \ldots, t_i$ are independent of the randomness used in the i-th copy of point-estimation algorithm, we can deterministically fix the updates on these time steps, and condition on the i-th copy of the non-adversarial streaming algorithm being correct on these updates. Therefore this algorithm correctly solves the 2ϵ point query problem on an adversarial stream. The total space used is

$$O\left(\epsilon^{-3}\log^2 n \log \epsilon^{-1} + T\epsilon^{-2}\log^2 n\right) .$$

We now note that we can improve the space by instead running only $T' = O(\epsilon^{-1}\log\epsilon^{-1})$ independent copies of the algorithm of Lemma 7.5.4. Each time we use one of the copies to output the desired estimate $\widehat{x}^i$, we completely restart that algorithm on the remaining suffix of the stream, and we loop modularly through all T' copies of the algorithm, at each step using the copy that was least recently restarted to output an estimate vector. More formally, we keep copies $\mathcal{A}_1, \ldots, \mathcal{A}_{T'}$ of the algorithm of Lemma 7.5.4. Each time we arrive at a new step t_i and must produce a new estimate $\widehat{x}^i$, we query the algorithm $\mathcal{A}_j$ that was *least recently restarted*, and use the estimate obtained by that algorithm, along with the estimates R^t.

The same correctness argument will hold as given above, except now each algorithm, when used after being restarted at least once, will only be ϵ-correct for the frequency vector defined by a *suffix* of the stream. However, by the same argument used in Theorem 82, we can safely disregard the prefix that was missed by this copy of the algorithm, because it contains only an $\epsilon/100$-fraction of the total L_p mass of the current frequency vector when it is applied again. Formally, if an algorithm is used again at time t_i, and it was last restarted at time τ, then by the correctness of our estimates R^t, the L_2 norm must have gone up by a factor of $(1+\epsilon)^{T'} = \frac{100}{\epsilon}$, so $\|x^{(\tau)}\|_2 \leq \epsilon/100\|x^{(t_i)}\|_2$. Moreover, we have that the estimate $\widehat{x}^i$ produced by this copy satisfies $\|\widehat{x}^i - (x^{(t_i)} - x^{(\tau)})\|_\infty \leq \epsilon\|x^{(t_i)} - x^{(\tau)}\|_2$. But then

$$\begin{aligned}
\|\widehat{x}^i - x^{(t_i)}\|_\infty &\leq \|\widehat{x}^i - (x^{(t_i)} - x^{(\tau)})\|_\infty + \|x^{(\tau)}\|_\infty \\
&\leq \epsilon\|x^{(t_i)} - x^{(\tau)}\|_2 + \|x^{(\tau)}\|_2 \\
&\leq \epsilon\left(\|x^{(t_i)}\|_2 + \|x^{(\tau)}\|_2\right) + \epsilon/100\|x^{(t_i)}\|_2 \\
&\leq \epsilon\|x^{(t_i)}\|_2(1+\epsilon) + \epsilon/100\|x^{(t_i)}\|_2 \\
&\leq 2\epsilon\|x^{(t_i)}\|_2
\end{aligned} \tag{7.1}$$

Thus $\widehat{x}^i$ is still 2ϵ-correct at time t_i for the full stream vector $x^{(t_i)}$. So by the same argument as above using Proposition 7.5.3, it follows that the output of the overall algorithm is always 4ϵ-correct for all time steps $\tau \in [m]$, and we can then re-scale ϵ by a factor of $1/4$. Substituting the new number T' of copies used into the above equation, we obtain the desired complexity. $\square$

7.6 Robust Entropy Estimation

We now show how our general techniques developed in Section 7.2 can be used to approximate the empirical Shannon entropy $H(x)$ of an adversarial stream. Recall that for a non-zero vector f, we have that $H(x) = -\sum_{i,(x)i\neq 0} p_i \log(p_i)$, where $p_i = \frac{|x_i|}{\|x\|_1}$. For $\alpha > 0$, we define the *α-Renyi Entropy* $H_\alpha(x)$ of x is given by

$$H_\alpha(x) = \frac{1}{1-\alpha} \log\left(\frac{\|x\|_\alpha^\alpha}{\|x\|_1^\alpha}\right)$$

We begin with the following observation, which will allow us to consider multiplicative approximation of $2^{H(x)}$. Then, by carefully bounding the flip number of the Renyi entropy H_α for α close to 1, we will be able to bound the flip number of H.

Remark 89. Note that any algorithm that gives an ϵ-additive approximation of the Shannon Entropy $H(x) : \mathbb{R}^n \to \mathbb{R}$ gives a $(1 \pm \epsilon)$ multiplicative approximation of $g(x) = 2^{H(x)}$, and vice-versa.

Proposition 7.6.1 (Theorem 3.1 of [HNO08a])**.** *Let $x \in \mathbb{R}^n$ be a probability distribution whose smallest non-zero value is at least $\frac{1}{m}$, where $m \geq n$. Let $0 < \epsilon < 1$ be arbitrary. Define $\mu = \epsilon/(4 \log m)$ and $\nu = \epsilon/(4 \log n \log m)$, $\alpha = 1 + \mu/(16 \log(1/\mu))$ and $\beta = 1 + \nu/(16 \log(1/\nu))$. Then*

$$1 \leq \frac{H_\alpha}{H} \leq 1 + \epsilon \text{ and } 0 \leq H - H_\beta \leq \epsilon$$

where $H(x) = H$ and $H_\alpha(x) = H_\alpha$.

Proposition 7.6.2. *Let $g : \mathbb{R}^N \to R$ be $g(x) = 2^{H(x)}$, i.e., the exponential of the Shannon entropy. Then the (ϵ, m)-flip number of g for the insertion only streaming model is $\lambda_{\epsilon,m}(g) = O(\frac{1}{\epsilon^3} \log^3 m)$.*

The proof of the above proposition is given later in this section. We now state the main result on adversarially robust entropy estimation. An improved result is stated for the *random oracle*

model in streaming, which recall implies that the algorithm is given random (read-only) access to an arbitrarily large string of random bits.

Theorem 90 (Robust Additive Entropy Estimation)**.** *There is an algorithm for ϵ-additive approximation of entropy in the insertion-only adversarial streaming model using $O(\frac{1}{\epsilon^5}\log^4 n)$-bits of space in the random oracle model, and $O(\frac{1}{\epsilon^5}\log^6 n)$-bits of space in the general insertion-only model.*

To obtain our entropy estimation algorithm of Theorem 90, we will first need to state the results for the state of the art non-adversarial streaming algorithms for additive entropy estimation. The first algorithm is a $O(\epsilon^{-2}\log^2 n)$-bit streaming algorithm for additive approximation of the entropy of a turnstile stream, which in particular holds for insertion only streams. The second result is a $\tilde{O}(1/\epsilon^2)$ upper bound for entropy estimation in the insertion only model when a random oracle is given.

Lemma 7.6.3 ([CC13])**.** *There is an algorithm in the turnstile model that gives an ϵ-additive approximation to the Shannon Entropy $H(x)$ of the stream. The failure probability is δ, and the space required is $O(\frac{1}{\epsilon^2}\log^2 n \log \delta^{-1})$ bits.*

Lemma 7.6.4 ([JW19])**.** *There is an algorithm in the insertion-only random oracle model that gives an ϵ-additive approximation to the Shannon Entropy $H(x)$ of the stream. The failure probability is δ, and the space required is $O(\frac{1}{\epsilon^2}(\log \delta^{-1} + \log\log n + \log \epsilon^{-1}))$*

We now give the proof of Proposition 7.6.2, and then the proof of Theorem 90.

Proof of Proposition 7.6.2. By Proposition 7.6.1, it suffices to get a bound on the flip number of H_β for the parameters $\beta = 1 + \nu/(16\log(1/\nu))$ and $\nu = \epsilon/(4\log n \log m)$. Recall $g(x) = 2^{H_\beta(x)} = (\|x\|_\beta^\beta/\|x\|_1^\beta)^{1/(1-\beta)}$. Now to increase $g(x)$ by a factor of $(1+\epsilon)$, one must increase $\|x\|_\beta^\beta/\|x\|_1^\beta$ by a factor of $1 + \Theta(\epsilon(1-\beta))$. For this to happen, $\|x\|_\beta$ must increase by a factor of $1 + \Theta(\epsilon(1-\beta))$. It holds that $\|x\|_\beta \le \|x\|_1 \le n^{1-1/\beta}\|x\|_1 = (1 + O(\epsilon/\log n))\|x\|_1$, and thus $\|x\|_1$ must also increase by a factor of $1 + \Theta(\epsilon)(1-\beta)$.

Similarly, for $g(x)$ to decrease by a factor of $1+\epsilon$, this requires $\|x\|_1$ to increase by a factor of $1 + \Theta(\epsilon(1-\beta))$. In summary, if for time steps $1 \le t_1 < t_2 \le m$ of the stream we have $g(x^{(t_2)}) > (1+\epsilon)g(x^{(t_1)})$ or $g(x^{(t_2)}) < (1-\epsilon)g(x^{(t_1)})$, then it must be the case that $\|x^{(t_2)}\|_1 > (1 + \Theta(\epsilon(1-\beta)))\|x^{(t_1)}\|_1 = (1+\tau)\|x^{(t_1)}\|_1$, where $\tau = \tilde{\Theta}(\epsilon^2/\log^2 n)$. Since $\|x^{(m)}\|_1 \le Mn$ and $\|\cdot\|_1$ is monotone for insertion only streams, it follows that this can occur at most $O(\frac{\log^3 n}{\epsilon^2})$ times, which completes the proof since $\log n = \Theta(\log m)$. □

Proof of Theorem 90. The proof follows directly from an application of Lemma 7.2.5, using the non-adversarial algorithms of Lemmas 7.6.3 and 7.6.4, as well as the flip number bound of Lemma 7.6.2. Note that to turn the algorithms of Lemmas 7.6.3 and 7.6.4 into tracking algorithms, one must set $\delta < 1/m$, which yields the stated complexity. □

7.7 Adversarial Robustness in the Bounded Deletion Model

In this section, we show how our results can be used to obtain adversarially robust streaming algorithms for the *bounded-deletion model*, introduced in our paper [JW18a], and the subject of Chapter 6. Recall that intuitively, a bounded deletion stream is one where the F_p moment of the stream is a $\frac{1}{\alpha}$ fraction of what the F_p moment would have been had all updates been replaced with their absolute values. We recall that formal definition here:

Definition 7.7.1. *Fix any $p \geq 1$ and $\alpha \geq 1$. A data stream $u_1, \ldots, u_m$, where $u_i = (a_i, \Delta_i) \in [n] \times \{1, -1\}$ are the updates to the frequency vector f, is said to be an F_p α-bounded deletion stream if at every time step $t \in [m]$ we have $\|x^{(t)}\|_p^p \geq \frac{1}{\alpha} \sum_{i=1}^{n} (\sum_{t' < t : a_{t'} = i} |\Delta_{t'}|)^p$.*

Specifically, the α-bounded deletion property says that the F_p moment $\|x^{(t)}\|_p$ of the stream is at least $\frac{1}{\alpha}\|y^{(t)}\|_p$, where y is the frequency vector of the stream with updates $u_i' = (a_i, \Delta_i')$ where $\Delta_i' = |\Delta_i|$ (i.e., the absolute value stream). Note here that the model assumes unit updates, i.e., we have $|\Delta_i| = 1$ for each $i \in [m]$, which can be accomplished without loss of generality with respect to the space complexity of algorithms, by simply duplicating integral updates into unit updates.

In Chapter 6, we saw that for α-bounded deletion streams, a factor of $\log n$ in the space complexity of turnstile algorithms can be replaced with a factor of $\log \alpha$ for many important streaming problems. In this section, we show another useful property of bounded-deletion streams: norms in such streams have bounded flip number. We use this fact to design adversarially robust streaming algorithms for data streams with bounded deletions.

Lemma 7.7.2. *Fix any $p \geq 1$. The $\lambda_{\epsilon,m}(\|\cdot\|_p)$ flip number of the L_p norm of a α-bounded deletion stream is at most $O(p\frac{\alpha}{\epsilon^p}\log n)$*

Proof. Let y be the frequency vector of the stream with updates $u_i' = (a_i, \Delta_i')$ where $\Delta_i' = |\Delta_i|$. Note that y is then the frequency vector of an insertion only stream. Now let $0 \leq t_1 < t_2 < \cdots < t_k \leq m$ be any set of time steps such that $\|x^{(t_i)}\|_p \notin (1 \pm \epsilon)\|x^{(t_{i+1})}\|_p$ for each $i \in [k-1]$.

Since by definition of the α-bounded deletion property, we have $\|x^{(t)}\|_p \geq \frac{1}{\alpha^{1/p}}\|y^{(t)}\|_p$ for each $t \geq T$, it follows that

$$\|x^{(t_{i+1})} - x^{(t_i)}\|_p \geq \left|\|x^{(t_{i+1})}\|_p - \|x^{(t_i)}\|_p\right| \geq \epsilon\|x^{(t_{i+1})}\|_p \geq \frac{\epsilon}{\alpha^{1/p}}\|y^{(t_{i+1})}\|_p \geq \frac{\epsilon^p}{\alpha^{1/p}}\|y^{(t_i)}\|_p \tag{7.2}$$

where in the last inequality we used the fact that y is an insertion only stream. Now since the updates to y are the absolute value of the updates to x, we also have that $\|y^{(t_{i+1})} - y^{(t_i)}\|_p^p \geq \|x^{(t_{i+1})} - x^{(t_i)}\|_p^p \geq \frac{\epsilon^p}{\alpha}\|y^{(t_i)}\|_p^p$. Thus

$$\|y^{(t_{i+1})}\|_p^p = \|y^{(t_i)} + \left(y^{(t_{i+1})} - y^{(t_i)}\right)\|_p^p \geq \|y^{(t_i)}\|_p^p + \|y^{(t_{i+1})} - y^{(t_i)}\|_p^p \geq (1 + \frac{\epsilon^p}{\alpha})\|y^{(t_i)}\|_p^p \tag{7.3}$$

where in the second inequality, we used the fact that $\|X+Y\|_p^p \geq \|X\|p^p + \|Y\|_p^p$ for non-negative integral vectors X, Y when $p \geq 1$. Thus $\|y^{(t_{i+1})}\|_p^p$ must increase by a factor of $(1 + \epsilon^p/\alpha)$ from $\|y^{(t_i)}\|_p^p$ whenever $\|x^{(t_i)}\|_p \notin (1 \pm \epsilon)\|x^{(t_{i+1})}\|_p$. Since $\|0\|_p^P = 0$, and $\|y^{(m)}\|_p^p \leq M^p n \leq n^{cP}$ for some constant $c > 0$, it follows that this can occur at most $O(p\frac{\alpha}{\epsilon^p}\log n)$ many times. Thus $k = O(p\frac{\alpha}{\epsilon^p}\log n)$, which completes the proof. □

We now use our computation paths technique of Lemma 7.2.6, along with the space optimal turnstile F_p estimation algorithm of [KNW10a], to obtain adversarially robust algorithms for α-bounded deletion streams. Specifically, we show that we can estimate the F_p moment of a bounded deletion stream robustly. We remark that once F_2 moment estimation can be done, one can similarly solve the heavy hitters problem in the robust model using a similar argument as in Section 4.3.2, except without the optimization used within the proof of Theorem 88 which restarts sketches on a suffix of the stream. The resulting space would be precisely an $(\frac{\alpha}{\epsilon}\log n)$-factor larger than the space stated in Theorem 88.

Theorem 91. *Fix $p \in [1, 2]$ and any constant $C > 1$. Then there is an adversarially robust F_p estimation algorithm which, with probability $1 - n^{-C}$, returns at each time step $t \in [m]$ an estimate R^t such that $R^t = (1\pm\epsilon)\|x^{(t)}\|_p^p$. The space used by the algorithm is $O(\alpha\epsilon^{-(2+p)}\log^3 n)$.*

of Theorem 91. We use the turnstile algorithm of [KNW10a], which gives an estimate $R^t = (1\pm\epsilon)\|x^{(t)}\|_p^p$ at a single point $t \in [m]$ with probability $1-\delta$, using $O(\epsilon^{-2}\log n\log\delta^{-1})$ bits of space. We can set $\delta = 1/\text{poly}(m)$, and union bound over all steps, to obtain that $R^t = (1 \pm \epsilon)\|x^{(t)}\|_p^p$ at all time steps $t \in [m]$ with probability $1 - n^{-C}$. Thus this gives a strong F_p tracking algorithm

using $O(\epsilon^{-2}\log n \log(n/\delta))$ bits of space. The theorem then follows from applying Lemma 7.2.6, along with the flip number bound of Lemma 7.7.2. □

7.8 Adversarial Attack Against the AMS Sketch

It was shown by [HW13] that linear sketches can in some cases be vulnerable to adaptive adversarial attacks (see Subsection 7.1.2). In this section, we show another instance of this phenomenon, demonstrating that the classic Alon-Matias-Szegedy (AMS) sketch [AMS96] for estimating the L_2 norm of a data stream is inherently non-robust. To this end, we describe an attack fooling the AMS sketch into outputting a value which is not a good approximation of the norm $\|x\|_2^2$ of the frequency vector. Our attack provides an even stronger guarantee: for any $r \geq 1$ and an AMS sketch with r/ϵ^2 rows, our adversary needs to only create $O(r)$ adaptive stream updates before it can fool the AMS sketch into outputting an incorrect result.

We first recall the AMS sketch for estimating the L_2 norm. The AMS sketch generates (implicitly) a random matrix $\mathbf{A} \in \mathbb{R}^{t\times n}$ such that the entries $\mathbf{A}_{i,j} \sim \{-1, 1\}$ are i.i.d. Rademacher.[9] The algorithm stores the sketch $\mathbf{A}x^{(j)}$ at each time step j, and since the sketch is linear it can be updated throughout the stream: $\mathbf{A}x^{(j+1)} = \mathbf{A}x^{(j)} + \mathbf{A}\cdot e_{i_{j+1}}\Delta_{j+1}$ where (i_{j+1}, Δ_{j+1}) is the $j+1$-st update. The estimate of the sketch at time j is $\frac{1}{t}\|\mathbf{A}x^{(j)}\|_2^2$, which is guaranteed to be with good probability a $(1\pm\epsilon)$ estimate of $\|x^{(j)}\|_2^2$ in non-adversarial streams if $t = \Theta(\epsilon^{-2})$.

We now describe our attack. Let $\mathbf{S}$ be a $t \times n$ Alon-Matias-Szegedy sketch. Equivalently, $\mathbf{S}_{i,j}$ is i.i.d. uniformly distributed in $\{-t^{-1/2}, t^{-1/2}\}$, and the estimate of AMS is $\|\mathbf{S}x^{(j)}\|_2^2$ at the j-th step. The protocol for the adversary is as follows. In the following, we let $e_i \in \mathbb{R}^n$ denote the standard basis vector which is zero everywhere except the i-th coordinate, where it has the value 1.

Note that the vector w in the above algorithm is always equal to the current frequency vector of the stream, namely $w = x^{(j)}$ after the j-th update. Note that the above algorithm can be implemented by an adversary who only is given the estimate $\|\mathbf{S}w\|_2^2 = \|\mathbf{S}x^{(j)}\|_2^2$ of the AMS sketch after every step j in the stream. To see this, note that the algorithm begins by inserting the first item $(i_1, \Delta_1) = (1, C\cdot\sqrt{t})$ for a sufficiently large constant C. Next, for $i = 2, \ldots, n$, it inserts the item $i \in [n]$ once if doing so increases the estimate of AMS by more than 1. If the estimate of AMS is increased by less than 1, it inserts the item i twice (i.e., it inserts an update

[9] In fact, the AMS sketch works even if the entries within a row of $\mathbf{A}$ are only 4-wise independent. Here, we show an attack against the AMS sketch if it is allowed to store a fully independent sketch $\mathbf{A}$.

Algorithm 6: Adversary for AMS sketch

1 $w \leftarrow C \cdot \sqrt{t} \cdot e_1$
2 **for** $i = 2, \ldots, m$ **do**
3 | old $\leftarrow \|\mathbf{S}w\|_2^2$
4 | $w \leftarrow w + e_i$
5 | new $\leftarrow \|\mathbf{S}w\|_2^2$
6 | **if** *new* $-$ *old* < 1 **then**
7 | | $w \leftarrow w + \mathbf{S}e_i$
8 | **else if** *new* $-$ *old* $= 1$ **then**
9 | | with probability $1/2$, set $w \leftarrow w + \mathbf{S}e_i$

$(i, 2) \in [n] \times \mathbb{Z}$). Lastly, if inserting the item $i \in [n]$ increases the estimate of AMS by *exactly* 1, the adversary chooses to insert $i \in [n]$ once with probability $1/2$, otherwise it inserts $i \in [n]$ twice.

We now claim that at the end of a stream of $m = O(t)$ updates, with good probability $\|w\|_2^2 = \|\mathbf{S}x^{(m)}\|_2^2 \notin (1 \pm \epsilon)\|x^{(m)}\|_2^2$. In fact, we show that regardless of the number of rows t in the AMS sketch, we force the AMS to give a solution that is not even a 2-approximation.

Theorem 92. *Let $\mathbf{S} \in \mathbb{R}^{t \times n}$ be an AMS sketch (i.i.d. Rademacher matrix scaled by $t^{-1/2}$), where $1 \le t < n/c$ for some constant c. Suppose further that the adversary performs the adaptive updates as described in Algorithm 6. Then with probability $9/10$, by the m-th stream update for some $m = O(t)$, the AMS estimate $\|\mathbf{S}x^{(m)}\|_2^2$ of the norm $\|x^{(m)}\|_2^2$ of the frequency vector $x \in \mathbb{R}^n$ defined by the stream fails to be a $(1 \pm 1/2)$ approximation of the true norm $\|x^{(m)}\|_2^2$. Specifically, we will have $\|\mathbf{S}x^{(m)}\|_2^2 < \frac{1}{2}\|x^{(m)}\|_2^2$.*

Proof. For $j = 2, 3 \ldots$ we say that the j-th *step* of Algorithm 6 is the step in the for loop where the parameter i is equal to j, and we define the first step to just be the state of the stream after line 1 of Algorithm 6. Let w^i be the state of the frequency vector at the end of the i-th step of the for loop in Algorithm 6, let $y^i = \mathbf{S}w^i$ be the AMS sketch at this step, and let $s_i = \|\mathbf{S}w^i\|_2^2$ be the estimate of AMS at the same point. Note that we have $w^1 = C \cdot \sqrt{t} \cdot e_1$ for a sufficiently large constant C, and thus $s_1 = C^2 t$. That is, already on the first step of the algorithm we have $\|w^1\|_2^2 = C^2 t$, and moreover since the stream is insertion only, we always have $\|w^i\|_2^2 \ge C^2 t$. Thus, it suffices to show that with good probability, at some time step $i \ge 2$ we will have $s_i < C^2 t/2$.

First, note that at any step $i = 2, 3, \ldots$, if we add $\mathbf{S}e_{i+1}$ once, we have $s_{i+1} = \|y^i + \mathbf{S}e_{i+1}\|_2^2 = \sum_{j=1}^t ((y_j^i)^2 + 2y_j^i \mathbf{S}_{j,i+1} + 1/t) = s_i + 1 + 2\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1}$. If we add $\mathbf{S}e_{i+1}$ twice, we have

$s_{i+1} = \|y^i + 2\underline{e}_{i+1}\|_2^2 = s_i + 4 + 4\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1}$. By definition of the algorithm, we choose to insert $\mathbf{S}e_{i+1}$ twice if $\|y^i + \mathbf{S}e_{i+1}\|_2^2 - s_i = 1 + 2\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1} < 1$, or more compactly whenever $\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1} < 0$. If $\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1} > 0$, we insert $\mathbf{S}e_{i+1}$ only once. Finally, if $\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1} = 0$, we flip an unbiased coin, and choose to insert $\mathbf{S}e_{i+1}$ either once or twice with equal probability $1/2$. Now observe that the random variable $\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1}$ is symmetric, since for any fixed y^i the $\mathbf{S}_{j,i+1}$'s are symmetric and independent. Thus, we have that

$$\begin{aligned} \mathbf{E}\left[\left|\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1}\right|\right] &= \mathbf{E}\left[\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1} \mid \mathbf{S}e_{i+1} \text{ inserted once}\right] \\ &= -\mathbf{E}\left[\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1} \mid \mathbf{S}e_{i+1} \text{ inserted twice}\right] \end{aligned} \tag{7.4}$$

Now recall that the vector $\mathbf{S}_{*,i+1}$ given by the $(i+1)$-st column of $\mathbf{S}$ is just an i.i.d. Rademacher vector scaled by $1/\sqrt{t}$. Thus by the classic Khintchine inequality [Haa81], we have that

$$\mathbf{E}\left[|\sum_{j=1}^t y_j^i \mathbf{S}_{j,i+1}|\right] = \frac{1}{\sqrt{t}} \cdot \alpha \cdot \|y^i\|_2 = \alpha\sqrt{s_i}/\sqrt{t}$$

for some absolute constant $\alpha > 0$ (in fact, $\alpha \geq 1/\sqrt{2}$ suffices by Theorem 1.1 of [Haa81]). Putting these pieces together, the expectation of the estimate of AMS is then as follows:

$$\begin{aligned} \mathbf{E}[s_{i+1}] &= \frac{1}{2}(s_i + 1 + 2\alpha\frac{\sqrt{s_i}}{\sqrt{t}}) + \frac{1}{2}(s_i + 4 - 4\alpha\frac{\sqrt{s_i}}{\sqrt{t}}) \\ &= s_i + 5/2 - \alpha\sqrt{s_i/t} \\ &\leq s_i + 5/2 - \sqrt{s_i/2t} \end{aligned} \tag{7.5}$$

Where again the last line holds using the fact that $\alpha \geq 1/\sqrt{2}$. Thus $\mathbf{E}[s_{i+1}] = \mathbf{E}[s_i] + 5/2 - \mathbf{E}\left[\sqrt{s_i/2t}\right]$. First, suppose there exists some $i \leq C^2t + 2$ such that $\mathbf{E}\left[\sqrt{s_i}\right] < C\sqrt{t/200}$. This implies by definition that $\sum_j \sqrt{j} \cdot \mathbf{Pr}[s_i = j] < C\sqrt{t/200}$, thus

$$\sqrt{C^2t/2} \cdot \mathbf{Pr}\left[s_i \geq C^2t/2\right] \leq \sum_{j \geq C^2t/2} \sqrt{j} \cdot \mathbf{Pr}[s_i = j] < \sqrt{C^2t/200} \tag{7.6}$$

Which implies that $\mathbf{Pr}[s_i \geq C^2t/2] \leq 1/10$. Thus, on that step i, we have $\mathbf{Pr}[s_i < C^2t/2] > 9/10$, and thus by time step i we have fooled the AMS sketch with probability at least $9/10$. Thus, we can assume that for all $i = 2, 3, \ldots, (C^2t + 2)$ we have $\mathbf{E}\left[\sqrt{s_i}\right] \geq C\sqrt{t/200}$. Setting $C > 200$, we have that $\mathbf{E}[s_{i+1}] < \mathbf{E}[s_i] - 1$ for all steps $i = 2, 3, \ldots, (C^2t + 2)$ However,

since $s_1 = C^2 t$, this implies that $\mathbf{E}[s_{C^2 t+2}] < -1$, which is impossible since s_j is always the value of a norm. This is a contradiction, which implies that such an i with $i \leq C^2 t + 2$ and $\mathbf{Pr}[s_i \geq C^2 t/2] \leq 1/10$ must exist, demonstrating that we fool the AMS sketch by this step with probability $9/10$, which completes the proof. □

7.9 Optimal Distinct Elements via Cryptographic Assumptions

Estimating the number of distinct elements (F_0-estimation) in a data stream is a fundamental problem in databases, network traffic monitoring, query optimization, data mining, and more. After a long line of work, [Woo04, KNW10b] settled space (and time) complexities of F_0-estimation by giving an algorithm using $O(\varepsilon^{-2} + \log n)$ bits of space (with constant worst-case update time). Recall that the tracking version of this algorithm (where it outputs a correct estimate at each time step) takes memory $O(\varepsilon^{-2}(\log \epsilon^{-1} + \log\log n) + \log n)$ bits and is also optimal [Bła18].

However, these results only hold in the (standard) static setting. We show that using cryptographic tools (pseudorandom functions), we can transform this algorithm, using the same amount of memory to be robust in the adversarial setting as well, where the adversary is assumed to be *computationally bounded* (as opposed to our other results which have no assumptions on the adversary whatsoever).

The transformation actually works for a large class of streaming algorithms. Namely, any algorithm such that when given an element that appeared before, does not change its state at all (with probability 1). Since the F_0 tracking algorithm of [Bła18] has this property, we can black-box apply our results to this algorithm.

First, we show how this transformation works assuming the existence of a truly random function, where the streaming algorithm has access to the function without needing to store it explicitly (the memory is free). This is known as the random oracle model. The model is appealing since we have different heuristic functions (e.g., SHA-256) that behave, as far as we can tell in practice, like random functions. Moreover, there is no memory cost when using them in an implementation, which is very appealing from a practical perspective. Nevertheless, we discuss how to implement such a function with cryptographic tools (e.g., pseudorandom functions) while storing only a small secret key in the memory.

Theorem 93 (Distinct Elements by Cryptographic Assumptions)**.** *In the random oracle model, there is an F_0-estimation (tracking) streaming algorithm in the adversarial setting, that for an approximation parameter ε uses $O(\varepsilon^{-2}(\log 1/\varepsilon + \log\log n) + \log n)$ bits of memory, and succeeds with probability $3/4$.*

Moreover, under a suitable cryptographic assumption, assuming the adversary has bounded running time of n^c, where m is the stream length and c is fixed, the random oracle can be replaced with a concrete function and the total memory is $O(\varepsilon^{-2}(\log 1/\varepsilon + \log\log n) + c\log n)$.

Proof. For simplicity, in the following proof, we assume that we have a random permutation. We note that the proof with a random function is exactly the same conditioned on not having any collisions. If the random function maps the universe to a large enough domain (say of size at least m^2) then there will be no collisions with high probability. Thus, it suffices to consider permutations.

The solution is inspired by the work of [NY15b] (which had a similar adaptive issue in the context of Bloom filters). Let Π be a truly random permutation, and let S be a tracking steaming algorithm with parameter ε. Let $L(\varepsilon, n)$ be the memory consumption of the algorithm. We construct an algorithm S' that works in the adversarial setting as follows. Upon receiving an element $e \in [n]$ (thought of as an insertion to the e-th coordinate of the underlying frequency vector $x \in \mathbb{R}^n$) the algorithm S' computes $e' = \Pi(e)$ and feeds it to S. The output of S' is exactly the output of S. Notice that applying Π to the stream does not change the number of distinct elements.

We sketch the proof. Assume towards a contradiction that there is adaptive adversary A' for S'. Consider the adversary A' at some point in time t, where the stream is currently $e_1, \ldots, e_t$. It has two options: (i) it can choose an element e_i, where $i \in [t]$ that appeared before, or (ii) it could choose a new element $e^* \notin \{e_1, \ldots, e_i\}$. Since the state of S' does not change when receiving duplicate items, and also does not change the number of distinct elements, option (i) has no effect on the success probability of A'. Thus, in order to gain a chance of winning, A' must submit a new query. Thus, we can assume without loss of generality that A' submits only distinct elements.

For such an adversary A' let D_t be the distribution over states of S' at time t. Let D'_t be the distribution over states of S' for the fixed sequence $1, 2, \ldots, t$. We claim that $D_t \equiv D'_t$ (identical distributions) for every $t \in [m]$. We show this by induction. The first query is non-adaptive, denote it by e_1. Then, since Π is a random permutation, we get that $\Pi(1) \equiv \Pi(e_1)$ which is what is fed to S. Thus, the two distribution are identical. Assume it holds for $t-1$. Consider the next

query of the adversary (recall that we assumed that this is a new query). Then, for any e_t (that has not been previously queried by Π) the distribution of $\Pi(e_t) \equiv \Pi(t)$, and therefore we get that $D_t \equiv D'_t$.

Given the claim above, we get that A' is equivalent to a static adversary A that outputs $1, 2, \ldots, k$ for some $k \in [m]$. However, the choice of k might be adaptive. We need to show that S' works for all k simultaneously. Here we use the fact that S was a tracking algorithm (and thus also S'), which means that S' succeeds on every time step. Thus, for the stream $1, 2, \ldots, m$, the algorithm S' succeeds at timestamp k, which consists of k distinct elements. Thus, if there exists an adaptive choice of k that would make S' fail, then there would exist a point in time, k, such that S' fails at $1, \ldots, k$. Since S is tracking, such a point does not exist (w.h.p.).

For the second part of the theorem, we note that we can implement the random function using an exponentially secure pseudorandom function (see [Gol05] for the precise definition and discussion). For a key K of size λ, the pesudorandom function $F_K(\cdot)$ looks random to an adversary that has oracle access to $F_K(\cdot)$ and runs in time at most $2^{\gamma\lambda}$ for some constant $\gamma > 0$. Let A be an adversary that runs in time at most n^c. Then, we set $O(\lambda = 1/\gamma \cdot c \cdot \log n)$ and get that A cannot distinguish between $F_K(\cdot)$ and the truly random function except when a negligible probability event occurs (i.e., the effect on δ is negligible and hidden in constants). Indeed, if A would be able to succeed against S' when using the oracle $F_K(\cdot)$, but, as we saw, it does not succeed when using a truly random function, then A' could be used to break the security of the pseudorandom function.

There are many different ways to implement such a pseudorandom function with exponential security and concrete efficiency. First, one could use heuristic (and extremely fast) functions such as AES or SHA256 (see also [NY15b] for a discussion on fast implementations of AES in the context of hash functions). Next, one can assume that the discrete logarithm problem (see [McC90] for the precise definition) over a group of size q is exponentially hard. Indeed, the best-known algorithm for the problem runs in time $O(\sqrt{q})$. Setting $q \geq 2^\lambda$ gets us the desired property for $\gamma = 1/2$.

To complete the proof, we note that the only property of A we needed was that when given an element in the stream that has appeared before, A does not change its state at all. This property holds for many F_0 estimation algorithms, such as the one-shot F_0 algorithm of [KNW10b], and the F_0 tracking algorithm of [Bła18]. Thus we can simply use the F_0 tracking algorithm of [Bła18], which results in the space complexity as stated in the theorem. □

Chapter 8

Streaming Algorithms for Earth Movers Distance

In this chapter, we study streaming algorithms for the *Earth Movers Distance* (EMD), which is a classic geometric distance function. The EMD between two multi-sets $A, B \subset \mathbb{R}^d$, each of size s, is the minimum cost of a perfect bipartite matching between the points in A and B, where cost is measured by distance in the metric – most commonly, the L_1 metric is used.[1] In a streaming context, the algorithm sees the points in $A = \{a_1, \ldots, a_s\}, B = \{b_1, \ldots, b_s\}$ one at a time, in an arbitrary order; we allow points to be both inserted *and* deleted. For simplicity, we now restrict our attention to points lying the in hypercube $\{0, 1\}^d$.[2]

Computational aspects of EMD, and more generally geometric minimum cost matchings, consistently arise in multiple areas of computer science [RTG00b, PC19]. Efficient computation of EMD, however, has been a well-known bottleneck in practice [Cut13, GCPB16, AWR17, LYFC19, BDI+20]. In particular, exact computation or even $(1 + \epsilon)$ approximations of the EMD between points, even lying in Euclidean space, is known require $\Omega(s^2)$ runtime under common fine-grained complexity assumptions [Roh19]. This has motivated many works from the theoretical perspective to study approximation and sketching algorithms for EMD [Cha02, IT03, AIK08b, ABIW09, AS14, BI14, ANOY14, She17, KNP19, BDI+20].

The primary and most successful analytical framework for developing fast offline approxi-

[1] More generally, one can define EMD between *weighted* multisets, lying in general metric spaces. For our purposes, we can assume the sets are equi-sized without loss of generality, and we will focus on the case of $\mathbb{R}^d$.

[2] This step will be without loss of generality for our purposes, due to isometric embeddings from $(\mathbb{R}^d, \ell_1)$ into $\{0, 1\}^{d'}$. Although the dimension can increase significantly here, our results demonstrate $\tilde{O}(\log s)$ approximations, which are therefore independent of the ambient dimension.

mation algorithms, as well as streaming algorithms, is the so called method of *randomized tree embeddings* [Bar96, Bar98, CCG+98, FRT04]. In this method, one recursively and randomly partitions the ambient space $\{0,1\}^d$, creating a recursion tree T. The points in $A \cup B$ are then mapped to leaves of the tree based on which pieces of the space partition they lie in. One then assigning positive weights to the edges of T to define a tree metric. Finally, one computes the EMD in the resulting tree metric. Crucially, the EMD between sets of points in a tree metric can be isometrically embedded into the L_1-norm, and therefore one obtains extremely efficient sketching and streaming algorithms (due to the existence efficient L_1 sketches). The resulting data structure is broadly referred to as a *quadtree* [Sam84].

It is known that, when run on s-point multisets $A, B \subset \{0,1\}^d$, the quadtree algorithm yields a $O(\min\{\log s, \log d\} \log s)$ approximation to the EMD; moreover, it is known that the cost of the quadtree can be maintained in a stream using at most $O(\log ds)$ bits of space [AIK08b, BDI+20]. Besides sketching, the quadtree algorithm itself is useful from the perspective of offline algorithms, since it is the best known *linear time* approximation algorithm for computing EMD. Prior to our results, this approximation was the state of the art known for computing EMD in a stream, or in linear time in an offline setting.

In this chapter, we present an *improved analysis* of the quadtree, showing that the approximation achieved is in fact $\tilde{\Theta}(\log s)$, instead of the prior known $O(\min\{\log s, \log d\} \log s)$. This immediately improves the state of the art for linear time approximations of high-dimensional EMD. On the other hand, it does not immediately result in streaming algorithms with the improved approximation; roughly, this is due to the fact that the edge weights utilized in the quadtree data structure we consider are *data-dependent* (precisely the same edge weights have been previously considered in the literature [BDI+20]). Nevertheless, by extending the L_p sampling techniques developed in Chapter 3, we design streaming algorithms for EMD with the new $\tilde{\Theta}(\log s)$ approximation.

Specifically, we give a *two-pass* streaming algorithm that gives a $\tilde{O}(\log s)$ approximation in $\mathrm{poly}(\log s, \log d)$-bits of space. With several additional technical insights, we demonstrate how this algorithm can be compressed to a single, one-pass streaming algorithm, at the cost of a small additive error of ϵs, in space $O(\frac{1}{\epsilon}) \cdot \mathrm{poly}(\log s, \log d)$. Since the distance between any two non-identical points is least 1, this algorithm gives a multiplicative $\tilde{O}(\log s)$ approximation in $\tilde{O}(d \cdot \mathrm{poly}(\log s))$ space, so long as $|A \cap B| \leq (1-\epsilon)s$. The main technical contribution for the sketch is the development of specialized algorithms for a generalization of L_p sampling, known as *sampling with meta-data*. Specifically, our algorithm builds on the precision sampling framework introduced in Chapter 3,

The materials in this chapter are based on a manuscript with Xi Chen, Amit Levi, and Erik Waingarten [CJLW20b].

Highlighted Contributions

The main contributions of this chapter are as follows:

- An improved analysis of the classic, offline *quadtree* Algorithm, demonstrating that it yields a $\tilde{\Theta}(\log s)$-approximation for EMD, instead of a $O(\min\{\log s, \log d\} \log s)$ approximation (Section 8.4).
- A two-pass streaming algorithm for approximating EMD to multiplicative error $\tilde{\Theta}(\log s)$, using $\mathrm{poly}(\log s, \log d)$-bits of space (Section 8.5).
- A one-pass streaming algorithm for approximating EMD to multiplicative error $\tilde{\Theta}(\log s)$ and additive error ϵs, using $d \cdot \mathrm{poly}(\log s, \log d, \frac{1}{\epsilon})$-bits of space (Section 8.6).

8.1 Background

Let (X, d_X) be a metric space. Given two (multi-)sets $A, B \subseteq X$ of size $|A| = |B| = s$, the Earth Mover distance (EMD) between A and B is

$$\mathrm{EMD}_X(A, B) = \min_{\substack{\text{matching} \\ M \subset A \times B}} \sum_{(a,b) \in M} d_X(a, b).$$

Computational aspects of EMD, and more generally, geometric minimum cost matchings consistently arise in multiple areas of computer science [RTG00b, PC19], and its non-geometric version is a classic algorithms topic [Kuh55, Mun57, EK72, AMO93]. Efficient computation of EMD, however, has been a well-known bottleneck in practice [Cut13, GCPB16, AWR17, LYFC19, BDI+20] which motivated many works from the theoretical perspective to study approximation and sketching algorithms for EMD [Cha02, IT03, AIK08b, ABIW09, AS14, BI14, ANOY14, She17, KNP19, BDI+20] in both low- and high-dimensional settings. A particularly relevant example of EMD in high dimensions comes from document retrieval and classification, a topic that received a remarkable amount of attention [KSKW15b]. Each document is represented as a collection of vectors given by embedding each of its words into a geometric space

[MSC+13, PSM14], and the distance between documents (as their similarity measure) is high-dimensional EMD, aptly named the *Word Mover's Distance*.

8.1.1 Contributions

We study a natural "divide-and-conquer" algorithm for estimating $\mathrm{EMD}_X(A, B)$, when the underlying metric space is the high-dimensional hypercube $\{0,1\}^d$ with ℓ_1 distance. The algorithm and analysis will be especially clean in this setting, since random coordinate sampling gives a simple partitioning scheme for the hypercube. Our analysis extends, via metric embeddings, to approximating $\mathrm{EMD}_X(A, B)$ when $X = (\mathbb{R}^d, \|\cdot\|_p)$ is an ℓ_p space with $p \in [1, 2]$ (see Section 8.11; we focus on $\{0,1\}^d$ with ℓ_1 distance in the rest of the chapter due to its simplicity).[3]

At a high level, the algorithm recursively applies randomized space partitions and builds a rooted (randomized) tree. Each point of A and B then goes down to a leaf of the tree, and the matching is formed with a bottom-up approach. The description of the algorithm, named ComputeEMD, is presented in Figure 8.1. More formally, ComputeEMD(A, B, h) takes three inputs, where h specifies the depth of the tree left and $A, B \subseteq \{0,1\}^d$ but their sizes do not necessarily match. It returns a triple (M, A', B') where M is a *maximal* partial matching between A and B, and $A' \subset A$ and $B' \subset B$ are the unmatched points. (Note that A' and B' will be empty whenever $|A| = |B|$ since M is maximal, and this will be the case in the initial call.) The algorithm samples a random coordinate $\boldsymbol{i} \sim [d]$ in order to partition $\{0,1\}^d$ into two sets (according to the value of coordinate $\boldsymbol{i}$), recursively solves the problem in each set, and matches unmatched points arbitrarily.

Our first result is an improved analysis of the approximation the algorithm achieves.

Theorem 94. *Let $A, B \subset \{0,1\}^d$ be any two multi-sets of size s. Then, with probability at least 0.9, ComputeEMD(A, B, d) outputs a tuple $(\mathbf{M}, \emptyset, \emptyset)$, where $\mathbf{M}$ is a matching of A and B satisfying*

$$\mathrm{EMD}(A, B) \leq \sum_{(a,b)\in\mathbf{M}} \|a - b\|_1 \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B).$$

Moreover, ComputeEMD *runs in time linear in the input size sd.*

[3] In particular, [KSKW15b] considers high-dimensional EMD over $\mathbb{R}^d$ with ℓ_2. Our results also extend to computing $\mathrm{EMD}_X(\mu_A, \mu_B)$, where μ_A and μ_B are arbitrary distributions supported on size-s sets A and B, respectively, and the goal is to approximate the minimum expectation of $\|\boldsymbol{a} - \boldsymbol{b}\|_1$, where $(\boldsymbol{a}, \boldsymbol{b}) \sim \mu_{A,B}$ for a coupling of μ_A and μ_B.

COMPUTEEMD (A, B, h)

Input: Two multi-sets $A, B \subseteq \{0,1\}^d$ and a nonnegative integer h.
Output: A tuple (M, A', B'), where M is a maximal matching between A and B, and $A' \subset A$ and $B' \subset B$ are points not participating in the matching M.

1. **Base Case**: If $|A| = 0$ or $|B| = 0$ or $h = 0$, or there is a single point $x \in \{0,1\}^d$ where $A = \{x, \ldots, x\}$ and $B = \{x, \ldots, x\}$, return (M, A', B'), where M is an arbitrary maximal partial matching between A and B, and let $A' \subset A$ and $B' \subset B$ be the left-over points. When $h = 0$, M should match as many identical pairs of points of A and B as possible.
2. **Divide**: Otherwise, sample $\boldsymbol{i} \sim [d]$ uniformly at random, and consider the four subsets

$$A_1 = \{a \in A : a_{\boldsymbol{i}} = 1\} \qquad A_0 = \{a \in A : a_{\boldsymbol{i}} = 0\}$$
$$B_1 = \{b \in B : b_{\boldsymbol{i}} = 1\} \qquad B_0 = \{b \in B : b_{\boldsymbol{i}} = 0\}.$$

   Make two recursively calls: run COMPUTEEMD$(A_1, B_1, h-1)$ to obtain (M_1, A_1', B_1') and COMPUTEEMD$(A_0, B_0, h-1)$ to obtain (M_0, A_0', B_0').
3. **Conquer**: Find an arbitrary maximal matching M_L between $A_1' \cup A_0'$ and $B_1' \cup B_0'$, and return $(M_1 \cup M_0 \cup M_L, A', B')$, where A', B' are unmatched points of A, B, respectively.

Figure 8.1: The COMPUTEEMD algorithm.

The above algorithm is not new, and falls within the framework of hierarchal methods for geometric spaces, building on what is broadly referred to as a *quadtree* [Sam84]. In fact, essentially the same tree structure is studied in [AIK08b, BDI+20]. In the rest of the chapter we will follow this convention and name the tree a quadtree, in which each node corresponds to a recursive call and is associated with a subcube of $\{0,1\}^d$. Leaves of this tree correspond to the base case in COMPUTEEMD.

Prior to our work, the analytical framework for studying tree-based algorithms for EMD is the method of randomized tree embeddings [Bar96, Bar98, CCG+98, FRT04]. For example, [AIK08b] consider sampling a random quadtree and assigning positive weights to its edges to define a tree metric. By studying the worst-case distortion of the original points in the randomized tree metric (given by the quadtree and weights), the analysis shows that the cost of the bottom-up matching achieves an $O(\min\{\log s, \log d\} \log s)$-approximation to EMD.[4] We note that this

[4]In particular, [AIK08b] proves a $O(\log d \log s)$-distortion bound for embedding into ℓ_1, which would naively result in a $O(\log d \log s)$-approximation to EMD. We note that even though the *expected* cost of the matching output by the bottom-up approach is an $O(\log d \log s)$-approximation to $\mathrm{EMD}(A, B)$, the matching output is an $O(\min\{\log d, \log s\} \log s)$-approximation with probability 0.9. See Section 8.7.

method of randomized tree embeddings is the de-facto technique for analyzing tree-based algorithms for EMD in both low- and high-dimensions [Cha02, IT03, Ind07, AIK08b, ABIW09, BI14, BDI+20]. It has the added benefit of producing an embedding of EMD into ℓ_1, which immediately implies sketching and nearest neighbor search algorithms.

Our work directly improves on [BDI+20] (their Theorem 3.5 is analogous to Theorem 94 with a bound of $O(\log^2 s)$), who (essentially) study ComputeEMD in the context of similarity search. Especially compelling are their experimental results, which show that the quality of the matching output by the quadtree *significantly* outperforms the cost of that same matching when measured with respect to distances in the tree embedding. This empirical observation is immensely useful in practice, since computing the cost of the matching incurs a relatively minor overhead. From a theoretical perspective, the observation suggests an avenue for improved analyzes, which in particular, should not heavily rely on the worst-case distortion of randomized tree embeddings. Theorem 94 gives a sound theoretical explanation to this empirical observation.

Theorem 94 is tight for ComputeEMD up to $\mathrm{poly}(\log \log s)$ factors (see Section 8.8). Conceptually, our analysis goes beyond worst-case distortion of randomized tree embeddings and implies the best linear time approximation for EMD over the hypercube. We note that the proof will not give an embedding into ℓ_1. Rather, Theorem 94 suggests an approach to improved approximations by linear sketches; implementing this approach, however, will require additional work, which we discuss next. Besides algorithms based on the quadtree, another class of algorithms proceed by sampling a geometric spanner [HPIS13] and utilizing min-cost flow solvers [She17]; these algorithms may achieve $O(1/\epsilon)$-approximations but require $ds^{1+\epsilon} \cdot \mathrm{polylog}(s, d)$ time. It is also not clear whether these algorithms and their analysis imply efficient sketching algorithms.

Streaming Algorithms for EMD

We first recall the notion of linear sketches, which recall from Section 1.1 imply the existence of turnstile streaming algorithms. Given an input vector $f \in \mathbb{R}^n$ and a function $\varphi(f) : \mathbb{R}^n \to \mathbb{R}$ that we would like to approximate, a (one-round) linear sketch generates a random matrix $\mathbf{S}_1 \in \mathbb{R}^{k \times n}$ and outputs an approximation $\tilde{\varphi} = \tilde{\varphi}(\mathbf{S}_1, \mathbf{S}_1 f)$ to $\varphi(f)$ based only on the matrix $\mathbf{S}_1$ and vector $\mathbf{S}_1 f$. A two-round linear sketch is allowed to generate a second random matrix $\mathbf{S}_2$, from a distribution depending on $(\mathbf{S}_1, \mathbf{S}_1 f)$, and output $\tilde{\varphi} = \tilde{\varphi}(\mathbf{S}_1, \mathbf{S}_1 f, \mathbf{S}_2, \mathbf{S}_2 f)$. The space of a linear sketch (resp. two-round linear sketch) is the number of bits needed to store $\mathbf{S}_1 f$ (resp. $\mathbf{S}_1 f$ and

$\mathbf{S}_2 f$).[5]

For sketching EMD over $\{0,1\}^d$, we encode two size-s multi-sets $A, B \subset \{0,1\}^d$ as an input vector $f = f_{A,B} \in \mathbb{R}^{2 \cdot 2^d}$, where for $x \in \{0,1\}^d$, f_x and f_{x+2^d} contains the number of occurrences of $x \in A$ and $x \in B$, respectively. We let $\varphi(f_{A,B}) = \mathrm{EMD}(A, B)$. One- and two-round linear sketches immediately result in one- and two-pass *streaming* algorithms in the *turnstile* model (where insertions and deletions of points are allowed), as well as one- and two-round two-party communication protocols, and one- and two-round MPC algorithms. (See Section 8.5 and Section 8.12 for more detail.)

Our first result for sketching EMD shows that it is possible to losslessly capture the approximation of Theorem 94 with a *two-round* linear sketch.

Theorem 95. *There is a two-round linear sketch using* $\mathrm{poly}(\log s, \log d)$ *bits of space which, given a pair of size-*s *multi-sets* $A, B \subset \{0,1\}^d$*, outputs* $\hat{\boldsymbol{\eta}} \in \mathbb{R}$ *satisfying*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\eta}} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B) \tag{8.1}$$

with probability at least $2/3$*.*

Theorem 95 implies a two-pass streaming algorithm as well as a two-round communication protocol for approximating EMD with the same space and approximation factors. Previously, the best sublinear space linear sketch (or streaming algorithm) for *any* number of rounds (or passes) utilizes the ℓ_1-embedding [AIK08b] and achieves approximation $O(\min\{\log s, \log d\} \log s)$.

Next we show that the two-round linear sketch of Theorem 95 can be further compressed into a single round, albeit at the cost of a small additive error.

Theorem 96. *Given* $\epsilon \in (0,1)$*, there is a (one-round) linear sketch using* $O(1/\epsilon) \cdot \mathrm{poly}(\log s, \log d)$ *bits of space which, given a pair of size-*s *multi-sets* $A, B \subset \{0,1\}^d$*, outputs* $\hat{\boldsymbol{\eta}} \in \mathbb{R}$ *satisfying*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\eta}} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B) + \epsilon s d$$

with probability at least $2/3$*.*

[5] For now, consider the random oracle model, where the algorithm is not charged for the space required to store the random matrices $\mathbf{S}_1, \mathbf{S}_2$. Using standard techniques and an application of Nisan's pseudo-random generator, we show that (in our application) this random oracle assumption can be dropped with a small additive $\tilde{O}(d)$ in the space complexity (see Corollaries 8.12.1 and 8.6.10). The $\tilde{O}(d)$ is a minor overhead given that each update of $A \cup B$ in the streaming model requires $d + 1$ bits to store.

Notice that $\mathrm{EMD}(A, B) \geq s$ when $A \cap B = \emptyset$, in which case Theorem 96 yields an $\tilde{O}(\log s)$ approximation in $\tilde{O}(d)$ space. More generally, if the *Jaccard Index* of A and B is bounded away from 1, we have the following corollary.

Corollary 8.1.1. *Given $\epsilon \in (0, 1)$ there is a (one-round) linear sketch using $O(d/\epsilon) \cdot \mathrm{poly}(\log s, \log d)$ space which, given size-s $A, B \subset \{0, 1\}^d$ such that $|A \cap B|/|A \cup B| \leq 1 - \epsilon$, outputs $\hat{\boldsymbol{\eta}} \in \mathbb{R}$ satisfying*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\eta}} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B)$$

with probability at least $2/3$.

Theorem 96 implies a one-pass streaming and a one-round communication protocol using the same space and achieving the same approximation. The proof of Theorem 96 involves the design of several new sketching primitives, which may be of broader interest. Specifically, our linear sketch needs to address the problem of *sampling with meta-data* (see Section 8.2), which can be used to approximate data-dependent weighted ℓ_1-distances. Its analysis extended and generalized error analysis of the precision sampling framework [AKO11, JST11**?**] to multivariate sampling, and provides new insights into the power of this framework.

8.2 Technical Overview

Analysis of Quadtrees

We start with an overview of the proof of Theorem 94. The running time analysis is straightforward so we focus on (8.1). The first inequality of (8.1) holds trivially. It suffices to show that the matching $\mathbf{M}$ returned by ComputeEMD has cost in the original metric upperbounded by $\tilde{O}(\log s) \cdot \mathrm{EMD}(A, B)$ with probability at least 0.9.

Compressed Quadtree and Worst-Case Analysis. The execution of ComputeEMD induces a complete binary tree T_0 of depth d which we refer to as a *Quadtree*[6]. Each internal node is labelled by a coordinate $\boldsymbol{i}$ sampled from $[d]$; its two children correspond to further subdividing $\{0, 1\}^d$ in half by fixing coordinate $\boldsymbol{i}$ to be either 0 or 1. We use $S_u \subset \{0, 1\}^d$ for each node u to denote the subcube associated with it. After sampling a Quadtree $\mathbf{T}_0$, each point of A and B is assigned to the leaf that contains it and the matching $\mathbf{M}$ is obtained in the bottom-up fashion as

[6]Note that the definition given here is slightly different from its formal definition at the beginning of Section 8.4, where a Quadtree will have depth $2d$. This difference is not important for the overview.

in Figure 8.1.

The first step is a simple compression of the Quadtree $\mathbf{T}_0$. To this end, we only keep the root of $\mathbf{T}_0$ (at depth 0) and its nodes at depths that are powers of 2; we also keep subcubes S_u associated with them. All other nodes are deleted and their adjacent edges are contracted (see Figure 8.2). The resulting "compressed Quadtree" $\mathbf{T}$ has depth $h = O(\log d)$, where each node u at depth i has $2^{2^{i-1}}$ children and is associated with a subcube $S_u \subseteq \{0,1\}^d$. An execution of COMPUTEEMD can now be viewed as first drawing a random compressed Quadtree $\mathbf{T}$, assigning points to its leaves, and then building $\mathbf{M}$ from bottom up.

Compressed Quadtrees are essentially the same as tree structures used in [AIK08b, BDI+20]. It will be helpful to begin with a recap of the analysis of tree embedding of [AIK08b] before presenting an overview of our new techniques.[7] [AIK08b] assigns a weight of $d/2^i$ to each edge from a node at depth i to a node at depth $i+1$ in a compressed Quadtree $\mathbf{T}$. This defines a metric embedding $\varphi : A \cup B \to \mathbf{T}$ by mapping each point to a leaf of $\mathbf{T}$. Their analysis bounds (from both sides) the cost of the bottom-up matching in the resulting tree metric. Their choice of edge weights is motivated by the observation that two points $x, y \in \{0,1\}^d$ with $\|x-y\|_1 = d/2^i$ are expected to have their paths diverge for the first time at depth i. If this is indeed the case then $d_{\mathbf{T}}(\varphi(x), \varphi(y))$ would be $\|x-y\|_1$ up to a constant.

To analyze the cost of the bottom-up matching in $\mathbf{T}$, one studies the distortion of this embedding. Firstly, for any $\lambda > 1$ and $x, y \in \{0,1\}^d$, it is easy to verify that

$$\Pr_{\mathbf{T}}\left[d_{\mathbf{T}}(\varphi(x),\varphi(y)) < \frac{1}{\lambda}\cdot\|x-y\|_1\right] \le \left(1-\frac{\|x-y\|_1}{d}\right)^{1+2+\cdots+2^{\left\lfloor \log_2\left(\frac{\lambda d}{\|x-y\|_1}\right)\right\rfloor}} \le 2^{-\Omega(\lambda)}.$$

Thus by a union bound, for all $x, y \in A \cup B$ we have

$$d_{\mathbf{T}}(\varphi(x),\varphi(y)) \ge \Omega\left(\frac{1}{\log s}\right)\cdot\|x-y\|_1 \tag{8.2}$$

with probability at least $1 - 1/\mathrm{poly}(s)$, which essentially means that we can assume (8.2) in the worst case. Thus, the cost of the bottom-up matching in the tree metric is at least $\Omega(1/\log s)\cdot \mathrm{EMD}(A,B)$. Furthermore, given points x, y with $\|x-y\|_1 = \Theta(d/2^j)$, the probability that paths of x, y diverge at level $j-k$ is at $\Theta(2^{-k})$ for each k, and when it does, $d_{\mathbf{T}}(\varphi(x),\varphi(y)) =$

[7] We refer the reader to a formal treatment of COMPUTEEMD via tree embeddings in Section 8.7.

$\|x-y\|_1 \cdot \Theta(2^k)$. Since $j \le h = O(\log d)$,

$$\mathbf{E}\Big[d_{\mathbf{T}}(\varphi(x),\varphi(y))\Big] \le \|x-y\|_1 + \sum_{k=0}^{j} \Theta(2^{-k}) \cdot \|x-y\|_1 \cdot \Theta(2^k) = O(\log d) \cdot \|x-y\|_1. \quad (8.3)$$

This, combined with the fact that the bottom-up matching is *optimal* for a tree metric, implies that the expected cost of the bottom-up matching in the tree metric is at most $O(\log d) \cdot \mathrm{EMD}(A,B)$.[8] Together they yield the aforementioned $O(\log s \log d)$ approximation [AIK08b]. In fact, the analysis is tight: see Section 8.9 for instances where the cost of the matching in the tree metric can be both a $(\log d)$-factor larger and a $(1/\log s)$-factor smaller than the EMD.

The Challenge of Worst-Case Distortion. Let $A_v = \{a \in A : a \in S_v\}$ be the set of points in A that pass through v, and let B_v be defined similarly. Then (8.2) upperbounds the diameter of $A_v \cup B_v$ in the original metric with high probability, i.e., every $x, y \in A_v \cup B_v$ has their distance $\|x-y\|_1$ bounded by $O(\log s) \cdot (d/2^i)$ with high probability if v is at depth i. Intuitively, upper bounds on diameters of $A_v \cup B_v$ can be very helpful for bounding the cost of the bottom-up matching in the original metric: If $a \in A$ and $b \in B$ are matched by COMPUTEEMD and their root-to-leaf paths diverge first at v, then we can use the diameter of v to bound their contribution $\|a-b\|_1$. However, as suggested by instances in Section 8.9, the loss of $O(\log s)$ in (8.2) is the best one can hope for in the worst-case.

Since the approximation we aim for is $\tilde{O}(\log s)$, it seems considering worst-case diameter may be sufficient; however, (8.3) naively results in an additional $O(\log d)$-factor loss in the approximation. Suppose that $a \in A$ and $b \in B$ are matched by the optimal matching and are at distance $\|a-b\|_1 = \Theta(d/2^j)$. As in (8.3), for every $k \le j$ the root-to-leaf paths of a and b may diverge at depth $j-k$ with probability $\Theta(2^{-k})$. If, once a and b diverge at depth $j-k$, a is matched to a point within distance $d/2^{j-k} \cdot O(\log s)$ (using the worst-case bound on the radii at depth $j-k$), then once we evaluate the expectation of $\|a - \mathbf{M}(a)\|_1$, where $\mathbf{M}(a) \in B$ is the matching produced by COMPUTEEMD, we would get $O(\log d \log s) \cdot d/2^j = O(\log d \log s) \cdot \|a-b\|_1$.

Inspector Payments. We setup the analytical framework to go beyond worst-case distortion, and give the main geometric insights next. Our analytical strategy is similar, in spirit, to the "accounting method" in amortized analysis. We consider a protocol carried out by an *Inspector*, who knows A,B, as well as the optimal matching M^*, and the *Algorithm*, who executes COMPUTEEMD(A,B,d), building the compressed Quadtree $\mathbf{T}$ and the bottom-up matching $\mathbf{M}$.

[8] The reason that this analysis can achieve approximation $O(\min\{\log s, \log d\} \log s)$, as opposed to $O(\log d \log s)$ is that with probability $1 - 1/s$, every $x, y \in A \cup B$ with $\|x-y\|_1 = \Theta(d/2^j)$ diverges at depth after $j - O(\log s)$.

As the algorithm executes, the inspector monitors the execution, and makes payments to the algorithm using knowledge of M^*. Specifically, the inspector will track each pair $(a, b) \in M^*$, and pays $\textbf{PAY}_{\mathbf{T}}(a)$ for a as well as a payment $\textbf{PAY}_{\mathbf{T}}(b)$ for b. The protocol effectively produces a coupling between the cost of the bottom-up matching $\mathbf{M}$ from $\mathbf{T}$, and the payments the inspector makes; we show that total payment can always cover the cost of $\mathbf{M}$, and is at most $\tilde{O}(\log s) \cdot \text{EMD}(A, B)$ with probability 0.9.

Formally we let $(v_0(x), v_1(x), \ldots, v_h(x))$ denote the root-to-leaf path of x in T. For each $(a, b) \in M^*$,

$$\textbf{PAY}_T(a) \stackrel{\text{def}}{=} \sum_{i \in [h]} \mathbf{1}\{v_i(a) \neq v_i(b)\} \Big(\|a - c_{v_i(a)}\|_1 + \|a - c_{v_{i-1}(a)}\|_1 \Big), \tag{8.4}$$

where c_v denotes the center-of-mass of $A_v \cup B_v$:

$$c_v \stackrel{\text{def}}{=} \frac{1}{|A_v \cup B_v|} \sum_{x \in A_v \cup B_v} x,$$

and $\textbf{PAY}_T(b)$ is defined similarly. Intuitively, this payment strategy corresponds to an inspector who tracks each pair $(a, b) \in M^*$, and whenever a and b first diverge in the tree at node v, pays twice the distance between a (as well as b) and to the center-of-mass along the v-to-leaf paths of a (as well as b).

In Lemmas 8.4.3 and 8.4.4, we show that for any T, the total inspector payments is sufficient to cover the cost of the matching $\mathbf{M}$ produced by the Quadtree:

$$\text{cost}(\mathbf{M}) = \sum_{(a,b) \in \mathbf{M}} \|a - b\|_1 \leq \sum_{(a,b) \in M^*} \textbf{PAY}_T(a) + \textbf{PAY}_T(b). \tag{8.5}$$

The inspector payments (8.4) depend on the data A, B, as well as a compressed tree T in two ways. The first is the depth when $(a, b) \in M^*$ first diverge, captured by the indicator $\mathbf{1}\{v_i(a) \neq v_i(b)\}$. The second is the distance between a to centers-of-mass, which not only depends on (a, b), but also on global properties of $A \cup B$. At a high level, incorporating this second aspect is the main novelty, since the distance between a and the center-of-mass of $A_v \cup B_v$ is an *average* notion of radii at v. In particular, the distance between a and the center-of-mass at v is at most the average distance between a and points in $A_v \cup B_v$.[9] Therefore, if the inspector pays a large amount, then an average point in $A_v \cup B_v$ is far from a (as opposed to the farthest point implied by worst-case radii).

[9]This used the fact that ℓ_1 is a normed space.

Bounding Inspector Payments. The technically most challenging part is upperbounding the expected total inspector payment (8.5), over a random compressed $\mathbf{T}$, by $\tilde{O}(\log s) \cdot \mathrm{EMD}(A, B)$. For the remainder of this subsection, consider a fixed $(a, b) \in M^*$ at distance $\|a-b\|_1 = \Theta(d/2^j)$, and we will upperbound the expectation of $\textsc{Pay}_{\mathbf{T}}(a) + \textsc{Pay}_{\mathbf{T}}(b)$ for random $\mathbf{T}$; furthermore, consider the payment in $\textsc{Pay}_{\mathbf{T}}(a)$ incurred from depth i after a and b have diverged. Namely, by linearity of expectation, we want to upper bound $\mathbf{E}\left[\|a - \mathbf{c}_{\mathbf{v}_i(a)}\|_1\right]$ for each depth i, where the expectation is over $\mathbf{T}$ conditioned on already having diverged from b (the conditioning will not heavily influence the geometric intuition, so we will forget about this for the rest of this overview). Let $\mathbf{v}_i = \mathbf{v}_i(a)$ be the vertex at depth i containing a, and let $\mathbf{c}_i = \mathbf{c}_{\mathbf{v}_i}$.

Similar to the worst-case bounds on radii, $\mathbf{E}\left[\|a - \mathbf{c}_i\|_1\right]$ may still be $d/2^i \cdot \Omega(\log s)$. For instance, let i_1 be a relatively large depth and for small $\epsilon \approx 10^{-6}$, consider a set P_1 of s^ϵ points at distance $\epsilon \log s \cdot d/2^{i_1}$ around a. Then, at depth i_1 of a random tree $\mathbf{T}$, a point in P_1 traverses down to node $\mathbf{v}_{i_1}$ with non-negligible probability, roughly $1/s^{-\epsilon}$. If no other points lie closer to a than P_1, then $\mathbf{E}\left[\|a - \mathbf{c}_{i_1}\|_1\right] = d/2^{i_1} \cdot \Omega(\epsilon \log s)$, since in expectation, some points from P_1 make it to $\mathbf{v}_{i_1}$ and move the center-of-mass $\mathbf{c}_{i_1}$ away from a. If this happened for every depth i, the inspector would be in trouble, as there are $O(\log d)$ levels and a similar argument to that of worst-case radii would mean they would pay $O(\log d \log s) \cdot \|a - b\|_1$ in expectation.

However, we claim that if the arrangement of P_1 resulted in $\mathbf{E}\left[\|a - \mathbf{c}_{i_2}\|_1\right] = d/2^{i_1} \cdot \Omega(\epsilon \log s)$, the same situation will be more difficult to orchestrate for depth $i_2 \le i_1 - O(\log \log s)$. In particular, at depth i_2, in order to have $\mathbf{E}\left[\|a - \mathbf{c}_{i_2}\|_1\right] = d/2^{i_2} \cdot \Omega(\epsilon \log s)$, there must be a set of points P_2 at distance $d/2^{i_2} \cdot \Omega(\epsilon \log s)$ which will cause $\mathbf{c}_{i_2}$ to be far from a. However, it is no longer enough to have $|P_2| = s^\epsilon$. The reason is that points of P_1 in $\mathbf{v}_{i_2}$ move the center-of-mass towards a. Since points in P_1 are at distance $\epsilon \log s \cdot d/2^{i_1} \ll d/2^{i_2}$ from a, there will oftentimes be $\Omega(s^\epsilon)$ points from P_1 in $\mathbf{v}_{i_2}$. In order to significantly affect the center-of-mass $\mathbf{c}_{i_2}$, $\mathbf{v}_{i_2}$ must oftentimes have at least $s^\epsilon/\mathrm{polylog}(s)$ points from P_2; otherwise, $\mathbf{c}_{i_2}$ will be mostly an average of points in P_1. Since any given point from P_2 traverses down to $\mathbf{v}_{i_2}$ with probability roughly $1/s^\epsilon$, we must have $|P_2| \ge s^{2\epsilon}/\mathrm{polylog}(s)$. This argument can only proceed for at most $O(1/\epsilon)$ depths before $|P_{O(1/\epsilon)}| > 2s$, in which case we obtain a contradiction, since the points $P_1, \ldots, P_{O(1/\epsilon)}$ must be in $A \cup B$.

Generally, in order to move the center-of-mass $\mathbf{c}_i$ of a vertex $\mathbf{v}_i$ away from a *multiple times* as the depth i goes down, the number of points around a at increasing distances must grow very rapidly. More specifically, we show that if a depth i is "bad," meaning that $\mathbf{E}\left[\|a - \mathbf{c}_i\|_1\right] \ge \alpha \cdot d/2^i$ for some $\alpha = \omega(\log \log s)$, then the number of points within a ball of radius $d/(2^i \log s)$ around a and within a larger ball of radius $O(\log s \cdot d/2^i)$ around a must have increased by a factor

of $\exp(\Omega(\alpha))$; this means the number of such depths i is at most $((\log s)/\alpha) \cdot \text{poly}(\log\log s)$. Combining this analysis and the fact that a and b must diverge in order to incur payment from the inspector, we obtain our upper bound $\mathbf{E}\left[\textbf{PAY}_{\mathbf{T}}(a) + \textbf{PAY}_{\mathbf{T}}(b)\right] = \tilde{O}(\log s) \cdot \|a - b\|_1$.

Sketching Algorithms

At a high level, our approach to sketching will involve sampling a compressed Quadtree $\mathbf{T}$ and sketching approximations to the inspector payments. As demonstrated by Theorem 94, these provide a good approximation to $\text{EMD}(A, B)$. The first step is to consider a modified inspector strategy $\boldsymbol{\mathcal{I}}$ which is *oblivious* to M^*, while still achieving

$$\text{EMD}(A, B) \le \boldsymbol{\mathcal{I}} \le \tilde{O}(\log s) \cdot \text{EMD}(A, B) \tag{8.6}$$

with probability at least 0.9 over $\mathbf{T}$, and approximate $\boldsymbol{\mathcal{I}}$ instead. (This $\boldsymbol{\mathcal{I}}$ will be what we call the cost of a compressed Quadtree in Section 8.4.) More specifically, let E_i denote the set of directed edges from depth $i-1$ to i in a compressed Quadtree $\mathbf{T}$, and let $\boldsymbol{\mathcal{I}} = \sum_{i \in [h]} \boldsymbol{\mathcal{I}}_i$ where

$$\boldsymbol{\mathcal{I}}_i \stackrel{\text{def}}{=} \sum_{(u,v) \in E_i} \Big| |A_v| - |B_v| \Big| \cdot \|c_v - c_u\|_1. \tag{8.7}$$

The upper bound in (8.6) follows from upper bounding $\boldsymbol{\mathcal{I}}$ by $\sum_{(a,b)\in M^*} (\textbf{PAY}_{\mathbf{T}}(a) + \textbf{PAY}_{\mathbf{T}}(b))$,[10] and the lower bound follows from Lemma 8.4.3. Intuitively $\boldsymbol{\mathcal{I}}$ measures the cost of moving all unmatched points in $A_v \cup B_v$ from the center-of-mass c_v up to the center-of-mass c_u of the parent of v. A consequence of the proof of Theorem 94 is that we can truncate $\|c_v - c_u\|_1$ and consider

$$w_{u,v} \stackrel{\text{def}}{=} \begin{cases} \frac{d \log s}{2^i} & \|c_v - c_u\|_1 \gg \frac{d \log s}{2^i} \\ \frac{d}{2^i} & \|c_v - c_u\|_1 \ll \frac{d}{2^i} \\ \|c_v - c_u\|_1 & \text{o.w.} \end{cases}, \tag{8.8}$$

since replacing $\|c_v - c_u\|_1$ with $w_{u,v}$ in (8.7) will still satisfies (8.6) with probability at least 0.89. Every edge (u, v) satisfies $\|c_v - c_u\| = O(\frac{d \log s}{2^i})$ with high probability over a random compressed Quadtree $\mathbf{T}$, and considering $d/2^i$ whenever $\|c_v - c_u\|_1 \ll d/2^i$ only adds an additive $O(\log s) \cdot \text{EMD}(A, B)$ to $\boldsymbol{\mathcal{I}}$ (both are consequences of a simple analysis using worst-case distortion).

Both our one and two round sketching algorithms proceed as follows. First, consider drawing

[10] This can be seen by two applications of the triangle inequality, one to upper bound $||A_v| - |B_v||$ by $\sum_{(a,b)\in M^*} \mathbf{1}\{v_i(a) \neq v_i(b) \text{ and } v_i(a) = v \text{ or } v_i(b) = v\}$, and the second by $\|c_{v_i(a)} - c_{v_{i-1}(a)}\|_1 \le \|a - c_{v_i(a)}\|_1 + \|a - c_{v_{i-1}(a)}\|_1$.

a random compressed Quadtree $\mathbf{T}$, and then for each $i \in \{0, \ldots, h-1\}$ define the (implicit) vector $\Delta^i \in \mathbb{R}^{E_i}$, indexed by directed edges from depth $i-1$ to depth i, given by

$$\Delta^i_{u,v} \stackrel{\text{def}}{=} |A_v| - |B_v|.$$

Now consider the distribution $\mathcal{D}_i$, supported on edges E_i, given by sampling $(u,v) \in E_i$ with probability $|\Delta^i_{u,v}|/\|\Delta^i\|_1$. Then for every $i \in \{0, \ldots, h-1\}$, we have

$$\boldsymbol{\mathcal{I}}_i = \|\Delta^i\|_1 \cdot \mathop{\mathbf{E}}_{(\boldsymbol{u},\mathbf{v})\sim\mathcal{D}_i}[w_{\boldsymbol{u},\mathbf{v}}]\,. \tag{8.9}$$

Note that we can estimate $\|\Delta^i\|_1$ with an L_1-sketch (Theorem 8), and produce a sample $(\boldsymbol{u},\mathbf{v}) \sim \mathcal{D}_i$ via the L_1-sampling sketches described in Chapter 3. Furthermore, once we fix a sample $(u,v) \in E_i$, we may estimate $w_{u,v}$ with L_1-sketches as well. Specifically, the sketch of $\|c_u - c_v\|_1$ proceeds by storing $|A_u|$, $|A_v|$, $|B_u|$, and $|B_v|$ using $O(\log s)$ bits, and storing L_1-linear sketches of

$$\chi_{A,u} \stackrel{\text{def}}{=} \sum_{a\in A_u} a, \quad \chi_{A,v} \stackrel{\text{def}}{=} \sum_{a\in A_v} a, \quad \chi_{B,u} \stackrel{\text{def}}{=} \sum_{b\in B_u} b, \quad \text{and} \quad \chi_{B,v} \stackrel{\text{def}}{=} \sum_{b\in B_v} b. \tag{8.10}$$

Since $c_u = (\chi_{A,u}+\chi_{B,u})/(|A_u|+|B_u|)$ and $c_v = (\chi_{A,v}+\chi_{B,v})/(|A_v|+|B_v|)$, L_1-linear sketches of the above quantities suffice for estimating $\|c_u - c_v\|_1$.

We notice that by definition (8.8), $w_{\boldsymbol{u},\mathbf{v}}$ is always bounded within $d/2^i$ and $d\log s/2^i$, so that we may estimate (8.9) by sampling $\mathrm{polylog}(s)$ times from $\mathcal{D}_i$ and computing the empirical average of $w_{\boldsymbol{u},\mathbf{v}}$. This plan is straight-forward to implement with two-rounds of linear sketching. In the first round, we produce the $t = \mathrm{polylog}(s)$ samples $(\boldsymbol{u}_1,\mathbf{v}_1), \ldots, (\boldsymbol{u}_t,\mathbf{v}_t) \sim \mathcal{D}_i$, as well as an estimate of $\|\Delta^i\|_1$. Then in the second round, we produce estimates of $w_{\boldsymbol{u}_1,\mathbf{v}_1}, \ldots, w_{\boldsymbol{u}_t,\mathbf{v}_t}$ given knowledge of $(\boldsymbol{u}_1,\mathbf{v}_1), \ldots, (\boldsymbol{u}_t,\mathbf{v}_t)$ by the linear sketches of the vectors in equation (8.10).

The remaining challenge, however, is to produce $(\boldsymbol{u},\mathbf{v}) \sim \mathcal{D}_i$ and an estimate of $w_{\boldsymbol{u},\mathbf{v}}$ *simultaneously*. We call this problem *sampling with meta-data*, since the L_1-linear sketches of (8.10) will be the meta-data of the sample $(\boldsymbol{u},\mathbf{v}) \sim \mathcal{D}_i$ needed to reconstruct $w_{\boldsymbol{u},\mathbf{v}}$.

Sampling with Meta-Data and One-Round Sketching. The key task of sampling with meta-data is the following: for $n,k \in \mathbb{N}$, we are given a vector $x \in \mathbb{R}^n$ and collection of *meta-data* vectors $\lambda_1, \lambda_2, \ldots, \lambda_n \in \mathbb{R}^k$, and the goal is to sample $i \in [n]$ with probability $|x_i|/\|x\|_1$ (or more generally, $|x_i|^p/\|x\|_p^p$), and output both i and an approximation $\hat{\lambda}_i \in \mathbb{R}^k$ of the vector λ_i. The challenge is to solve this problem with a small-space linear sketches of x and the meta-data

vectors $\lambda_1, \ldots, \lambda_n$. Notice that sampling with meta-data is a generalization of L_p sampling.

It is not hard to see that sampling with meta-data is exactly the problem we seek to solve for linear sketching of EMD.[11] We refer the reader to Section 8.10 for generic bounds on sampling from meta-data. However, our application to EMD for Theorem 96 will require significant additional techniques, which we detail next.

Our algorithm utilizes the *precision sampling* framework from Chapter 3 for L_p sampling. Recall that the idea is to produce, for each $i \in [n]$ an independent exponential random variable $\boldsymbol{t}_i \sim \mathrm{Exp}(1)$, and construct a "scaled vector" $\boldsymbol{z} \in \mathbb{R}^n$ with coordinates $\boldsymbol{z}_i = x_i/\boldsymbol{t}_i$. One then attempts to return the index $i_{\max} = \mathrm{argmax}_{i \in [n]} \boldsymbol{z}_i$, since

$$\mathop{\mathbf{Pr}}_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n \sim \mathrm{Exp}(1)} \left[\mathop{\mathrm{argmax}}_{i' \in [n]} \frac{|x_{i'}|}{\boldsymbol{t}_{i'}} = i \right] = \frac{|x_i|}{\|x\|_1}.$$

To find the the index $i_{\max}$ with a linear sketch, we can use a "heavy-hitters" algorithm, specifically the Count-Sketch from Section 2.3.2. Recall that Count-Sketch with error $\epsilon \in (0,1)$ allows us to recover an estimate $\tilde{\boldsymbol{z}}$ to $\boldsymbol{z}$ satisfying (roughly) $\|\tilde{\boldsymbol{z}} - \boldsymbol{z}\|_\infty \leq \epsilon \|\boldsymbol{z}\|_2$. Then one can show that $\mathrm{argmax}_{i' \in [n]} |\tilde{\boldsymbol{z}}_{i'}|$ is close to being distributed as $|x_i|/\|x\|_1$.[12]

In order to sample with meta-data, our sketch similarly samples independent exponential $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n \sim \mathrm{Exp}(1)$ and applies a Count-Sketch data structure on $\boldsymbol{z} \in \mathbb{R}^n$, where $\boldsymbol{z}_i = x_i/\boldsymbol{t}_i$, and obtains an estimate $\tilde{\boldsymbol{z}}$ of $\boldsymbol{z}$. In addition, for each $\ell \in [k]$, we apply a Count-Sketch data structure with error ϵ for the vector $\boldsymbol{w}^\ell \in \mathbb{R}^n$ given by the ℓ-th coordinates of the meta-data vectors $\lambda_i/\boldsymbol{t}_i$, namely $\boldsymbol{w}^\ell_i = (\lambda_i)_\ell/\boldsymbol{t}_i$.[13] From this we obtain an estimate $\tilde{\boldsymbol{w}}^\ell$ of $\boldsymbol{w}^\ell$. The insight is the following: suppose the sample produced is $i^* \in [n]$, which means it satisfies $\tilde{\boldsymbol{z}}_{i^*} \approx \max_{i \in [n]} |x_i|/\boldsymbol{t}_i$. Then the value $\boldsymbol{t}_{i^*}$ should be relatively small: in particular, we expect $\boldsymbol{t}_{i^*}$ to be $\Theta(|x_{i^*}|/\|x\|_1)$. When this occurs, for each $\ell \in [k]$, the guarantees of Count-Sketch imply that the estimate $\boldsymbol{t}_{i^*} \cdot \tilde{\boldsymbol{w}}^\ell_{i^*}$ satisfies

$$\left| \boldsymbol{t}_{i^*} \cdot \tilde{\boldsymbol{w}}^\ell_{i^*} - (\lambda_{i^*})_\ell \right| = \boldsymbol{t}_{i^*} \left| \tilde{\boldsymbol{w}}^\ell_{i^*} - \boldsymbol{w}^\ell_{i^*} \right| \leq \epsilon \boldsymbol{t}_{i^*} \|\boldsymbol{w}^\ell\|_2 \left(= O\left(\epsilon |x_{i^*}| \cdot \frac{\|(\lambda_\cdot)_\ell\|_1}{\|x\|_1} \right) \text{ in expectation} \right)$$

[11] Namely, x is the vector Δ^i (after hashing the universe E_i to size $\mathrm{poly}(s)$, since Δ^i is a $2s$-sparse vector), and the meta-data vectors are the L_1-linear sketches of (8.10) for each $(u,v) \in E_i$. In other words, $n = \mathrm{poly}(s)$, and $k = \mathrm{polylog}(s,d)$.

[12] We remark that prior works [AKO11, JST11, JW18b] using this framework for ℓ_p sampling require an additional statistical test, where the algorithm outputs **FAIL** when the test fails. The statistical test adds a layer of complexity to the presentation of an otherwise elegant algorithm, and we observe in Lemma 8.6.4 that such a statistical test is unnecessary, up to minor errors in sampling distribution.

[13] Since the Count-Sketch data structures will be of size $\mathrm{polylog}(n)$, the final size of the sketch is $k \cdot \mathrm{polylog}(n)$.

where $(\lambda_\cdot)_\ell \in \mathbb{R}^n$ is the vector of ℓ-th coordinates of the meta-data $\lambda_1, \ldots, \lambda_n$. In other words, if the size of $(\lambda_{i^*})_\ell$ is comparable to $|x_{i^*}|$, and if the ratio $\|(\lambda_\cdot)_\ell\|_1/\|x\|_1$ of the meta-data norms to the norm of x is bounded, then $\boldsymbol{t}_{i^*}\tilde{\boldsymbol{w}}^\ell_{i^*}$ is a relatively good approximation to $(\lambda_{i^*})_\ell$. Repeating the above argument for every $\ell \in [k]$ recovers an approximation for $\lambda_{i^*} \in \mathbb{R}^k$.

In our application, the vector $x \in \mathbb{R}^n$ is given by Δ^i, and the meta-data are the L_1-linear sketches of (8.10), as well as counts $|A_v|, |A_u|, |B_v|$, and $|B_u|$ needed to reconstruct $w_{u,v}$. At a high level, applying the above algorithm results in an L_1-sample $(\boldsymbol{u}, \mathbf{v}) \sim \mathcal{D}_i$, and an approximation to the L_1-linear sketches of (8.10) and approximations to the counts $|A_u|, |A_v|, |B_u|$, and $|B_v|$, where the error guarantee is additive and linearly related to the ratio between the sum of magnitudes of the coordinates we seek to recover and $\|\Delta^i\|_1$ (this plan will runs into a technical issue, which we will soon expand on).

In order to see how this may be implemented, consider the problem of recovering the L_1-linear sketch of $\chi_{A,v}$. Recall that Indyk's L_1-sketch [Ind06] (Theorem 8) has coordinates given by inner products of (8.10) with vectors of independent Cauchy random variables, which means that the ℓ-th coordinate in the L_1-linear sketch of $\chi_{A,v}$ is given by

$$\frac{1}{\text{median}(|\mathcal{C}|)} \sum_{j=1}^{d} \mathbf{C}_j \left(\sum_{a \in A_v} a_j \right),$$

where $\mathbf{C}_1, \ldots, \mathbf{C}_d$ are independent Cauchy random variables $\mathcal{C}$ and $\text{median}(|\mathcal{C}|)$ is the median of the distribution.[14] From the fact that $\mathbf{C}_1, \ldots, \mathbf{C}_d$ are Cauchy random variables, the above sum is expected to have magnitude $O(\log d) \cdot O(d) \cdot |A_v|$. Since the error guarantee depends on the sum of magnitudes across the ℓ-th coordinate of all meta-data vectors (one for each edge in E_i), if we sample $(\boldsymbol{u}, \mathbf{v}) \sim \mathcal{D}_i$, the additive error on the ℓ-th coordinate of the L_1-sketch of $\chi_{A,\mathbf{v}}$ we recover becomes

$$\epsilon \cdot \frac{|\Delta^i_{\boldsymbol{u},\mathbf{v}}|}{\|\Delta^i\|_1} \cdot O(\log d) \cdot O(d) \cdot \sum_v |A_v| = O\left(\epsilon \cdot sd \log d \cdot \frac{|\Delta^i_{\boldsymbol{u},\mathbf{v}}|}{\|\Delta^i\|_1} \right), \tag{8.11}$$

where we used the fact that $\{A_v\}_v$ partitions A in order to say $\sum_v |A_v| = s$. Furthermore, if $\|\Delta^i\|_1 \ll \epsilon_0 s 2^i/(\log d \log s)$, then from (8.9) and the fact $w_{u,v} \leq d \log s/2^i$, we can already conclude that $\boldsymbol{\mathcal{I}}_i \ll \epsilon_0 sd/\log d$, which is a negligible (since we allow $\epsilon_0 sd$ additive error, and there are at most $O(\log d)$ depths), so that we may assume $\|\Delta^i\|_1 = \tilde{\Omega}(\epsilon s 2^i)$. For these depths where $\|\Delta^i\|_1$ is sufficiently large, the additive error in (8.11) incurred will be smaller than a typical ℓ-th coordinate of the sketch of $\chi_{A,\mathbf{v}}$, giving us a coordinate-wise relative error of the

[14] Namely, $\text{median}(|\mathcal{C}|) = \sup\{t \in \mathbb{R} : \mathbf{Pr}_{\mathbf{C} \sim \mathcal{C}}[|\mathbf{C}| \leq t] \leq 1/2$.

sketch.

The above is sufficient for recovering a relative error approximation to the L_1-linear sketch of $\chi_{A,v}$ and $\chi_{B,v}$ for the vertices v at depth i, by setting ϵ to be a small enough $1/\mathrm{polylog}(s,d)$. However, the same argument loses additional $O(s)$-factors when recovering approximations to L_1-linear sketches of $\chi_{A,u}$ and $\chi_{B,u}$ for the vertices u at depth $i-1$. The reason is that the size of $\|(\lambda_\cdot)_\ell\|_1$ becomes $O(\log d)\cdot O(d)\cdot\sum_v |A_{\pi(v)}|$, where $\pi(v)$ is the parent of v; in some cases, this is $\Omega(s^2 d\log d)$. Intuitively, the problem is that while the the multi-sets $\{A_v\}_v$ and $\{B_v\}_v$ partition A and B, the multi-sets $\{A_{\pi(v)}\}_v$ and $\{B_{\pi(v)}\}_v$ may duplicate points s times, causing the magnitudes of coordinates in the L_1-linear sketches to much larger. This additional factor of $O(s)$ would require us to make $\epsilon = 1/(s\cdot\mathrm{polylog}(s,d))$, increasing the space complexity of the sketch to $\tilde{O}(s)$, effectively rendering it trivial.[15]

To get around this issue, we utilize a two-step precision sampling with meta-data algorithm. We first sample $\boldsymbol{u}^*$ with probability proportional to the L_1-norm of Δ^i restricted to coordinates corresponding to the children of $\boldsymbol{u}^*$; namely, we sample $\boldsymbol{u}^*$ with probability proportional to $\sum_{v:\pi(v)=\boldsymbol{u}^*}|\Delta^i_{\boldsymbol{u}^*,v}|$.[16] Since the multi-sets $\{A_u\}_u$ and $\{B_u\}_u$ partition A and B, and we can recover L_1-linear sketches for $\chi_{A,\boldsymbol{u}^*}$ and $\chi_{B,\boldsymbol{u}^*}$ up to relative error, as well as approximations to the counts for $|A_{\boldsymbol{u}^*}|$ and $|B_{\boldsymbol{u}^*}|$. Once we have $\boldsymbol{u}^*$, we apply precision sampling with meta-data once more, to sample a child $\mathbf{v}^*$ from $\boldsymbol{u}^*$ proportional to $|\Delta^i_{\boldsymbol{u}^*,\mathbf{v}^*}|$. Specifically, we generate a second set of exponentials $\{\boldsymbol{t}_v\}_v$, one for each child node v. In order to ensure that the sample produced by the second sketch actually returns a child $\mathbf{v}^*$ of $\boldsymbol{u}^*$, and not a child of some other node, we crucially must scale the vector Δ^i by *both* the child exponentials $\{\boldsymbol{t}_v\}_v$ as well as the parent exponentials $\{\boldsymbol{t}_u\}_u$ from the first sketch. Thus, we analyze the twice-scale vector $\boldsymbol{z}$ with coordinates $\boldsymbol{z}_v = \Delta^i_{u,v}/(\boldsymbol{t}_u\boldsymbol{t}_v)$, and attempt to find the largest coordinate of $\boldsymbol{z}$. Importantly, notice that this makes the scaling factors in $\boldsymbol{z}_v$ no longer independent: two children of the same parent share one of their scaling factors. Moreover, the Cauchy random variables used in the L_1-linear sketches must also be the same for children with the same parent. Executing this plan requires a careful analysis of the behavior of norms of vectors scaled by several non-independent variables, as well as a nested Count-Sketch to support the two-stage sampling procedure.

[15] In particular, a $\tilde{O}(s/\epsilon^2)$-size sketch may proceed by taking L_1-sketches of the s vectors in A and B such that all pairwise distances are preserved, giving a $(1+\epsilon)$-approximation to $\mathrm{EMD}(A,B)$.

[16] This value must be approximated via a Cauchy sketch, since the L_1 norm of the children is norm of $\boldsymbol{u}^*$ is not a linear function.

8.3 Quadtrees and Compressed Quadtrees

Fix $s, d \in \mathbb{N}$, and consider two multi-sets $A, B \subset \{0,1\}^d$ of size $|A| = |B| = s$. We write

$$\mathrm{cost}(M) = \sum_{(a,b)\in M} \|a - b\|_1$$

to denote the cost of a matching M between A and B (in the original ℓ_1 distance). For convenience, we will assume without loss of generality that d is a power of 2 and write $h = \log 2d = \log d + 1$. Given a vertex v in a rooted tree T, if v is not the root then we use the notation $\pi(v)$ to denote the parent of v in T.

We start with a formal definition of Quadtrees used in this paper:

Definition 8.3.1 (Quadtrees)**.** *A Quadtree is a complete binary tree T_0 of depth $2d$. We say a node v is at depth j if there are $j+1$ nodes on the root-to-v path in T_0 (so the root is at depth 0 and its leaves are at depth $2d$). Each internal node of T_0 is labelled a coordinate $i \in [d]$, and we refer to its two children as the "zero" child and the "one" child. A random Quadtree $\mathbf{T}_0$ is drawn by sampling a coordinate $\boldsymbol{i} \sim [d]$ uniformly and independently for each internal node of depth at most d as its label; every internal node of depth $d + j$, $j \in [d]$, is labelled j.*

Given a Quadtree T_0, each point $x \in \{0,1\}^d$ can be assigned to a leaf of T_0 by starting at the root and repeatedly going down to the "zero" child if $x_i = 0$ for the label i of the current node, or to the "one" child if $x_i = 1$. Alternatively one can define a subcube $S_v \subseteq \{0,1\}^d$ for each node v: The set for the root is $\{0,1\}^d$; If (u, v) is an edge, u is labelled i, and v is the "zero" (or "one") child of u, then $S_v = \{y \in S_u : x_i = 0\}$ (or $S_v = \{y \in S_u : x_i = 1\}$). Each point $x \in \{0,1\}^d$ is then assigned to the unique leaf v with $x \in S_v$. Given a Quadtree T_0, we write A_v to denote $A \cap S_v$ and B_v to denote $B \cap S_v$ for each node v of T_0 (thus A_v contains all points $a \in A$ such that v is on its root-to-leaf path in T_0, and the same holds for B_v).

Next we define the compressed version of a Quadtree. Roughly speaking, we compress a Quadtree T_0 by contracting all nodes of T_0 at depths that are not powers of 2, and keeping every other node as well as its subcube S_v (including the pair of multi-sets A_v and B_v).

Definition 8.3.2 (Compression)**.** *Given any Quadtree T_0 of depth $2d$, we write $T = \textsc{Compress}(T_0)$ to denote the following rooted tree of depth $h = \log 2d$. For each $i = 1, \ldots, h$, there is a one-to-one correspondence between nodes of T at depth i and nodes of T_0 at depth 2^i; the root of T corresponds to the root of T_0. Node v is a child of u in T if the corresponding node of v is*

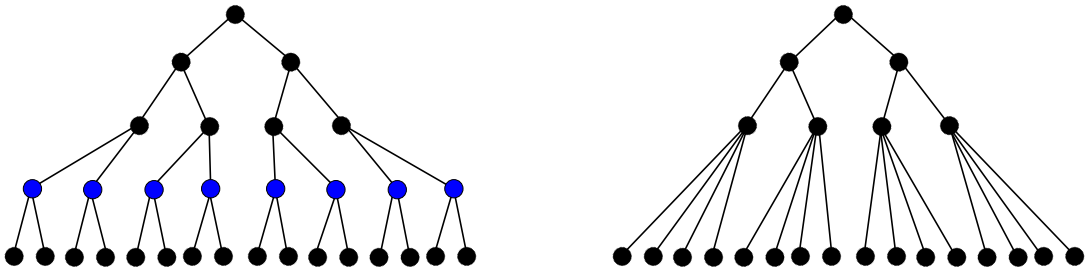


Figure 8.2: The Compression Operation. The Quadtree T_0 on the left-hand side is a complete binary tree, and its $\textsc{Compress}(T_0)$ is represented on the right-hand side. The compression contracts nodes which are not at a depth that is a power of 2, which in this case, corresponds to nodes at depth 3 (labeled in blue). Note that every node of T naturally corresponds with a node of T_0 (in particular, we have displayed nodes as remaining in the same relative position to illustrate this point).

a descendant of the corresponding node of u in T_0. Each node v in T is labelled a subcube S_v and a pair of multi-sets A_v and B_v, copied from its corresponding node in T_0. See Figure 8.2 for an example of $\textsc{Compress}(T_0)$. *A random compressed Quadtree $\mathbf{T}$ is drawn by first drawing a random Quadtree $\mathbf{T}_0$ and then setting $\mathbf{T} = \textsc{Compress}(\mathbf{T}_0)$.*

The key property we use about a random compressed Quadtree is summarized in the lemma below (which follows directly from its definition):

Lemma 8.3.3. *Fix any point $x \in \{0,1\}^d$. Let $\mathbf{T}$ be a random compressed Quadtree and $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_h$ be the root-to-leaf path of x in $\mathbf{T}$. Then $(S_{\mathbf{v}_1}, \ldots, S_{\mathbf{v}_{h-1}})$ can be drawn equivalently as follows*[17]*: First draw $\mathbf{I}_0, \mathbf{I}_1, \ldots, \mathbf{I}_{h-2}$ where $\mathbf{I}_0, \mathbf{I}_1 \sim [d]$ and $\mathbf{I}_i \sim [d]^{2^{i-1}}$ for each $i \geq 1$ independently and uniformly at random, and set $S_{\mathbf{v}_i}$ to be the subcube of $\{0,1\}^d$ that agrees with x on all coordinates in $\mathbf{I}_0, \ldots, \mathbf{I}_{i-1}$.*

Remark 97. Both works of *[AIK08b, BDI+20]* use a tree structure that is very similar to a compressed Quadtree as defined above. They consider a slightly different divide-and-conquer algorithm which at depth i, samples 2^i coordinates from $[d]$ and divides into 2^{2^i} branches according to settings of $\{0,1\}$ to these 2^i coordinates. The resulting tree structure is akin to a compressed Quadtree, in which each node at depth i has 2^{2^i} children. As we will see, our analysis apply to trees of *[AIK08b, BDI+20]* as well given that the lemma above also holds trivially for their trees.

[17] Note that $S_{\mathbf{v}_0}$ is trivially $\{0,1\}^d$ and $S_{\mathbf{v}_h}$ is trivially $\{x\}$.

8.4 Analysis of Compressed Quadtrees

Let A, B be multi-sets of $\{0,1\}^d$ of size s, and let $M^* \subset A \times B$ be an optimal matching so that

$$\mathrm{EMD}(A,B) = \sum_{(a,b)\in M^*} \|a-b\|_1.$$

We prove Theorem 94 in this section. For the time complexity of COMPUTEEMD(A,B,d), we note that the running time of each call with $h > 1$ (excluding running time from its recursive calls) is linear in $|A| + |B|$, the total size of its two input sets. On the other hand, the running time of each call with $h = 0$ can be bounded by $O((|A| + |B|)d)$. (Recall that we need to find a maximal partial matching M that matches as many identical pairs of points of A and B as possible. This can be done by first sorting $A \cup B$ and noting that given there are only 2^d points in the space, we only need to pay $O(d)$ for each insertion instead of $O(\log(|A| + |B|))$ which could be larger than d.) Equivalently, one can charge $O(1)$ running time to each point in the two input sets of an internal node and $O(d)$ to each point at each leaf of the recursion tree. Therefore, each point pays at most $O(d)$ given that COMPUTEEMD has depth at most d. It follows that its overall running time is $O(sd)$. Given the cost of any matching is trivially at least $\mathrm{EMD}(A,B)$, it suffices to upperbound the cost of the matching returned by COMPUTEEMD by $\tilde{O}(\log s) \cdot \mathrm{EMD}(A,B)$ with probability at least 0.9, which we focus on in the rest of the section.

To this end, we first note that COMPUTEEMD can be viewed as first drawing a random compressed Quadtree $\mathbf{T}$ and then returning a matching $\mathbf{M}$ built in the bottom-up fashion as described in Figure 8.1. Given a fixed compressed Quadtree T, the following definition captures the class of matchings that COMPUTEEMD may return; basically these are matchings that satisfy a "depth-greedy" property:

Definition 8.4.1. *Let T be a compressed Quadtree. For any $a \in A$ and $b \in B$, let*

$$\textit{depth}_T(a,b) \stackrel{\text{def}}{=} \textit{depth of the least-common ancestor of leaves of } a,b \textit{ in } T.$$

The class of depth-greedy matchings*, denoted by $\mathcal{M}_T(A,B)$, is the set of all matchings $M \subseteq A \times B$ which maximize the sum of $\textit{depth}_T(a,b)$ over all pairs $(a,b) \in M$.*

Fixing a compressed Quadtree T, every matching COMPUTEEMD may return when running with T belongs to $\mathcal{M}_T(A,B)$ (indeed $\mathcal{M}_T(A,B)$ can contain strictly more matchings in general). Therefore the goal is now to show that with probability at least 0.9 over $\mathbf{T}$, $\mathrm{cost}(M)$ of every

$M \in \mathcal{M}_{\mathbf{T}}(A, B)$ is upperbounded by $\tilde{O}(\log s) \cdot \mathrm{EMD}(A, B)$.

Our plan consists of three steps. First we introduce the *cost* of a compressed Quadtree T (Definition 8.4.2 below), denoted by $\mathrm{cost}(T)$, and show that $\mathrm{cost}(M)$ of any matching $M \in \mathcal{M}_T(A, B)$ can be bounded from above by $\mathrm{cost}(T)$ (Lemma 8.4.3). (Looking ahead, $\mathrm{cost}(T)$ will be the quantity our linear sketches estimate in Sections 8.5 and 8.6; see Remark 98.) Next we define payments $\textbf{PAY}_T(a)$ and $\textbf{PAY}_T(b)$ of the inspector as mentioned in the overview earlier, and show that $\mathrm{cost}(T)$ is bounded from above by the total inspector payment (Lemma 8.4.4). The final and most challenging step is to bound the expected total inspector payment. The key technical lemma, Lemma 8.4.5, will be proved in Section 8.4.1.

We start with the cost of a compressed Quadtree. We need a couple of definitions used throughout the rest of the paper. Given a compressed Quadtree T and a point $x \in \{0,1\}^d$, we let the sequence

$$v_{0,T}(x), v_{1,T}(x), \ldots, v_{h,T}(x)$$

denote the root-to-leaf path of x in the tree T. For any node v at depth $i \in \{0, \ldots, h\}$, we let

$$A_{v,T} \stackrel{\text{def}}{=} \{a \in A : v_{i,T} = v\} \qquad \text{and} \qquad B_{v,T} \stackrel{\text{def}}{=} \{b \in B : v_{i,T} = v\},$$

and we let

$$c_{v,T} \stackrel{\text{def}}{=} \frac{1}{|A_{v,T} \cup B_{v,T}|} \sum_{x \in A_{v,T} \cup B_{v,T}} x$$

be the *center-of-mass* of points in $A_{v,T} \cup B_{v,T}$ at v; we set $c_{v,T}$ to be the all-0 point by default when $A_{v,T} = B_{v,T} = \emptyset$. For notational simplicity, we will usually suppress T from the notation when it is clear from context (this will also be the case for notation introduced later in this section). We are now ready to define the cost of a compressed Quadtree:

Definition 8.4.2. *Let T be a compressed Quadtree. Then its cost is defined as*[18]

$$\mathrm{cost}(T) \stackrel{\text{def}}{=} \sum_{(u,v) \in E_T} \Big| |A_v| - |B_v| \Big| \cdot \|c_v - c_u\|_1. \tag{8.12}$$

Notice that the right-hand side of (8.12) is data-dependent in two respects: 1) each edge of the tree contributes the discrepancy between A and B proceeding down that edge, and 2) the amount each edge pays times the discrepancy between A and B is given by the distance between centers-of-mass of the point set of $A \cup B$ mapped to u and v.

[18] Whenever we refer to an edge (u, v) in T, u is always the parent node and v is the child node.

We prove the following lemma for the first step of the proof.

Lemma 8.4.3. *Let T be a compressed Quadtree. Then $\mathrm{cost}(M) \le \mathrm{cost}(T)$ for all $M \in \mathcal{M}_T(A, B)$.*

Proof. Given a matching $M \in \mathcal{M}_T(A, B)$ and a pair $(a, b) \in M$, we write v and w to denote the leaves of a and b and use u to denote their least-common ancestor. We also write $u, v_1, \ldots, v_k = v$ and $u, w_1, \ldots, w_k = w$ to denote the paths from u to v and w, respectively. By triangle inequality,

$$\begin{aligned}\|a - b\|_1 &\le \|a - c_{v_k}\|_1 + \|c_{v_k} - c_{v_{k-1}}\|_1 + \cdots + \|c_{v_1} - c_u\|_1 \\ &\quad + \|c_u - c_{w_1}\| + \cdots + \|c_{w_{k-1}} - c_{w_k}\|_1 + \|c_{w_k} - b\|_1 \\ &= \|c_{v_k} - c_{v_{k-1}}\|_1 + \cdots + \|c_{v_1} - c_u\|_1 + \|c_u - c_{w_1}\| + \cdots + \|c_{w_{k-1}} - c_{w_k}\|_1,\end{aligned}$$

where the equation follows from the fact that all points at a leaf of T must be identical and so is their center. Summing up these inequalities over $(a, b) \in M$ gives exactly $\mathrm{cost}(T)$. For this, note that every M in $\mathcal{M}_T(A, B)$ has the property that, for any edge (u, v) in T, the number of $(a, b) \in M$ such that the path between their leaves contains (u, v) is exactly $||A_v| - |B_v||$. □

Now it suffices to upperbound $\mathrm{cost}(\mathbf{T})$ by $\tilde{O}(\log s) \cdot \mathrm{EMD}(A, B)$ with probability at least 0.9 for a random compressed Quadtree $\mathbf{T}$. For this purpose, we define an inspector payment for each point in $A \cup B$ based on an optimal matching M^* between A and B and a compressed Quadtree T. Given a point $a \in A$, if $b \in B$ is the point matched with a in M^* (i.e., $(a, b) \in M^*$), we let

$$\mathbf{PAY}_T(a) \stackrel{\text{def}}{=} \sum_{i \in [h]} \mathbf{1}\{v_{i,T}(a) \neq v_{i,T}(b)\} \Big(\|a - c_{v_{i,T}(a)})\|_1 + \|a - c_{v_{i-1,T}(a)}\|_1 \Big). \tag{8.13}$$

Intuitively $\mathbf{PAY}_T(a)$ pays for the distance between a and centers-of-mass along its root-to-leaf path but the payment only starts *at* the least-common ancestor of leaves of a, b. The payment $\mathbf{PAY}_T(b)$ is defined similarly (with a and b switched). Note that $\mathbf{PAY}_T(a) = \mathbf{PAY}_T(b) = 0$ if $a = b$.

We show that the total inspector payment from points of $A \cup B$ is enough to cover $\mathrm{cost}(T)$:

Lemma 8.4.4. *Let T be any compressed Quadtree. Then we have*

$$\mathrm{cost}(T) \le \sum_{a \in A} \mathbf{PAY}_T(a) + \sum_{b \in B} \mathbf{PAY}_T(b). \tag{8.14}$$

Proof. Using the definition of $\mathrm{cost}(T)$, it suffices to show that

$$\sum_{(u,v)\in E_T} \Big||A_v| - |B_v|\Big| \cdot \|c_v - c_u\|_1 \le \sum_{a\in A} \mathbf{PAY}_T(a) + \sum_{b\in B} \mathbf{PAY}_T(b).$$

By triangle inequality,

$$\mathbf{PAY}_T(a) \ge \sum_{i\in[h]} \mathbf{1}\{v_{i,T}(a) \neq v_{i,T}(b)\}\Big(\|c_{v_{i-1,T}}(a) - c_{v_{i,T}(a)})\|_1\Big),$$

i.e., $\mathbf{PAY}_T(a)$ is enough to cover $\|c_u - c_v\|$ for every edge (u,v) along its leaf-to-root path until the least-common ancestor with b is met. The lemma then follows from the following claim: For every edge (u,v) in T, $||A_v| - |B_v||$ is at most the number of points $a \in A_v$ such that its matched point in M^* is not in B_v plus the number of points $b \in B_v$ such that its matched point in M^* is not in A_v. This follows from the simple fact that every $(a,b) \in M^*$ with $a \in A_v$ and $b \in B_v$ would get cancelled in $|A_v| - |B_v|$. This finishes the proof of the lemma. □

By Lemma 8.4.4 the goal now is to upperbound the total inspector payment by $\tilde{O}(\log s) \cdot \mathrm{EMD}(A,B)$ with probability at least 0.9 over a randomly picked compressed Quadtree $\mathbf{T}$. We consider a slight modification of the payment scheme given in (8.13) which we define next; the purpose is that the latter will be easier to bound in expectation, and most often exactly equal to (8.13).

Specifically, given a pair $(a,b) \in M^*$ and $i_0 \in [h]$, we let $\widetilde{\mathbf{PAY}}_{i_0,T}(a) = 0$ if $a = b$ and let

$$\widetilde{\mathbf{PAY}}_{i_0,T}(a) \stackrel{\text{def}}{=} \sum_{i=i_0}^{h} \mathbf{1}\{v_{i,T}(a) \neq v_{i,T}(b)\}\Big(\|a - \tilde{c}_T(i,a))\|_1 + \|a - \tilde{c}_T(i-1,a)\|_1\Big) \tag{8.15}$$

when $a \neq b$, where

$$\tilde{c}_T(i,a) \stackrel{\text{def}}{=} \frac{1}{|C_T(i,a)|} \sum_{x\in C_T(i,a)} x$$

is the center-of-mass of a subset $C_T(i,a)$ of $A_{v_{i,T}(a)} \cup B_{v_{i,T}(a)}$ that are not too far away from a:

$$C_T(i,a) \stackrel{\text{def}}{=} \left\{ x \in A_{v_{i,T}(a)} \cup B_{v_{i,T}(a)} : \|x - a\|_1 \le \frac{10d\log s}{2^i} \right\}.$$

Roughly speaking, two points in $A \cup B$ that share the same node $v_{i,T}$ are expected to have distance around $d/2^i$ (given that they agreed so far on roughly 2^i random coordinates sampled);

this is why we referred to points in $C_T(i,a)$ as those that are not too far away from a. Similar to $\mathbf{PAY}_T(a)$, we define $C_T(i,b)$ and $\tilde{c}_T(i,b)$ for each $b \in B$ and use them to define $\widetilde{\mathbf{PAY}}_T(b)$.

The following is the crucial lemma for bounding the total expected payment from points in $A \cup B$. We delay its proof to Section 8.4.5 and first use it to prove Theorem 94.

Lemma 8.4.5. *For any $(a,b) \in M^*$ with $a \neq b$ and $i_0 \in [h]$ that satisfies*

$$i_0 \leq \min\left\{1, \left\lfloor \log_2\left(\frac{d}{\|a-b\|_1}\right)\right\rfloor\right\}, \tag{8.16}$$

we have

$$\max\left\{\mathop{\mathbf{E}}_{\mathbf{T}}\left[\widetilde{\mathbf{PAY}}_{i_0,\mathbf{T}}(a)\right], \mathop{\mathbf{E}}_{\mathbf{T}}\left[\widetilde{\mathbf{PAY}}_{i_0,\mathbf{T}}(b)\right]\right\} \leq \left(\tilde{O}(\log s) + O(\log\log s)\left(\log\left(\frac{d}{\|a-b\|_1}\right) - i_0\right)\right)\cdot\|a-b\|_1,$$

where the randomness is over a random compressed Quadtree $\mathbf{T}$.

Proof of Theorem 94 assuming Lemma 8.4.5. Let $\mathbf{T}$ be a random compressed Quadtree. Then

$$\mathrm{cost}(\mathbf{T}) \leq \sum_{a\in A}\mathbf{PAY}_{\mathbf{T}}(a) + \sum_{b\in B}\mathbf{PAY}_{\mathbf{T}}(b) = \sum_{\substack{(a,b)\in M^*\\ a\neq b}}\mathbf{PAY}_{\mathbf{T}}(a) + \mathbf{PAY}_{\mathbf{T}}(b) \tag{8.17}$$

given that $\mathbf{PAY}_{\mathbf{T}}(a) = \mathbf{PAY}_{\mathbf{T}}(b) = 0$ for every pair $(a,b) \in M^*$ with $a = b$. We focus on the subset M' of M^* with $(a,b) \in M^*$ and $a \neq b$. For each pair $(a,b) \in M'$, let

$$\ell_{\min}(a,b) \stackrel{\text{def}}{=} \min\left\{1, \left\lfloor \log_2\left(\frac{d}{\|a-b\|_1}\right)\right\rfloor - 2\lceil\log_2 s\rceil\right\}.$$

We show that with probability at least $1 - o(1)$ over the draw of $\mathbf{T}$, every $(a,b) \in M'$ satisfies

$$\mathbf{PAY}_{\mathbf{T}}(a) = \widetilde{\mathbf{PAY}}_{\ell_{\min}(a,b),\mathbf{T}}(a) \quad \text{and} \quad \mathbf{PAY}_{\mathbf{T}}(b) = \widetilde{\mathbf{PAY}}_{\ell_{\min}(a,b),\mathbf{T}}(b). \tag{8.18}$$

Combining (8.17) and (8.18), we have that with probability at least $1 - o(1)$ over the draw of $\mathbf{T}$,

$$\mathrm{cost}(\mathbf{T}) \leq \sum_{(a,b)\in M'}\widetilde{\mathbf{PAY}}_{\ell_{\min}(a,b),\mathbf{T}}(a) + \widetilde{\mathbf{PAY}}_{\ell_{\min}(a,b),\mathbf{T}}(b). \tag{8.19}$$

By applying Lemma 8.4.5 to every $(a,b) \in M'$ with $i_0 = \ell_{\min}(a,b)$, as well as Markov's in-

equality, we have that with probability at least 0.99 over $\mathbf{T}$, the right hand side of (8.19) is at most

$$\tilde{O}(\log s) \sum_{(a,b)\in M'} \|a-b\|_1 = \tilde{O}(\log s) \cdot \mathrm{EMD}(A,B).$$

By a union bound, $\mathrm{cost}(\mathbf{T}) \le \tilde{O}(\log s) \cdot \mathrm{EMD}(A,B)$ with probability at least $.99 - o(1) \ge 0.9$.

It suffices to define an event that implies (8.18) and then bound its probability. The first part of the event requires that for every pair $(a,b) \in M'$, $v_{i,\mathbf{T}}(a) = v_{i,\mathbf{T}}(b)$ for every $i : 1 \le i < \ell_{\min}(a,b)$. The second part requires that for any two distinct points $x, y \in A \cup B$ (not necessarily as a pair in M^* and not even necessarily in the same set), we have $v_{i,\mathbf{T}}(x) \ne v_{i,\mathbf{T}}(y)$ for all i with

$$2^i \ge \frac{10d \log s}{\|x-y\|_1}. \tag{8.20}$$

By the definition of $\widetilde{\mathbf{PAY}}_{\ell_{\min}(a,b),\mathbf{T}}(a)$ in (8.15) the first part of the event makes sure that we don't miss any term in the sum (8.15); the second part of the event makes sure that every $C_{\mathbf{T}}(i,a)$ is exactly the same as $A_{v_{i,\mathbf{T}}}(a) \cup B_{v_{i,\mathbf{T}}}(a)$ (and the same holds for b). It follows that this event implies (8.18).

Finally we show that the event occurs with probability at least $1 - o(1)$. First, for every $(a,b) \in M'$, the probability of $v_{i,\mathbf{T}}(a) \ne v_{i,\mathbf{T}}(b)$ for some $i : 1 \le i < \ell_{\min}(a,b)$ is at most

$$1 - \left(1 - \frac{\|a-b\|_1}{d}\right)^{2^{\ell_{\min}(a,b)}} \le 2^{\ell_{\min}(a,b)} \cdot \frac{\|a-b\|_1}{d} \le \frac{1}{s^2}.$$

Hence, by a union bound over the at most s pairs $(a,b) \in M'$, the first part of the event holds with probability at least $1 - o(1)$. Furthermore, for any two distinct points $x, y \in A \cup B$, let

$$\ell_{\max}(x,y) = \left\lfloor \log_2 \left(\frac{10d \log s}{\|x-y\|_1} \right) \right\rfloor.$$

Then $v_{i,\mathbf{T}}(x) = v_{i,\mathbf{T}}(y)$ for some i that satisfies (8.20) would imply $v_{\ell_{\max}(x,y),\mathbf{T}}(x) = v_{\ell_{\max}(x,y),\mathbf{T}}(y)$ and $\ell_{\max}(x,y) \le \log d$ (as $v_{h,\mathbf{T}}(x) \ne v_{h,\mathbf{T}}(y)$ given $x \ne y$). The event above happens with probability

$$\left(1 - \frac{\|x-y\|_1}{d}\right)^{2^{\ell_{\max}(x,y)-1}} \le \exp(-5 \log s) = \frac{1}{s^5}.$$

Via a union bound over at most $(2s)^2$ many pairs of x, y, we have that the second part of the event also happens with probability at least $1 - o(1)$. This finishes the proof of the theorem. □

Remark 98 (Looking toward Section 8.5 and 8.6)**.** Before moving on to the proof of Lemma 8.4.1 we define a quantity that is slightly different from $\mathrm{cost}(T)$ which will be used in our linear sketches of Section 8.5 and 8.6 to estimate $\mathrm{EMD}(A, B)$.

An inspection of the proof of Theorem 94 reveals that with probability $1-o(1)$ over $\mathbf{T}$ (which we skip in subscripts below), every non-empty node v ($A_v \cup B_v \neq \emptyset$) at depth i with parent u satisfies

$$\|\mathbf{c}_v - \mathbf{c}_u\|_1 \leq \frac{30d \log s}{2^i}.$$

The reason is that we have argued, with probability at least $1 - o(1)$ over $\mathbf{T}$, for all $x \in A \cup B$ and all depth i, all points in $A_{v_i(x)} \cup B_{v_i(x)}$ are within distance $10d \log s/2^i$ of x. This implies that every non-empty node v (say $x \in A_v \cup B_v$) at depth i and its parent u satisfy

$$\|\mathbf{c}_v - \mathbf{c}_u\|_1 \leq \|\mathbf{c}_v - x\|_1 + \|\mathbf{c}_u - x\|_1 \leq \frac{10d \log s}{2^i} + \frac{10d \log s}{2^{i-1}} = \frac{30d \log s}{2^i}.$$

In particular, with probability at least 0.9 over $\mathbf{T}$, we have

$$\mathrm{EMD}(A, B) \leq \sum_{(u,v)\in E_{\mathbf{T}}} \Big||A_v| - |B_v|\Big| \cdot \min\left\{\|\mathbf{c}_v - \mathbf{c}_u\|_1, \frac{30d \log s}{2^i}\right\} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B). \tag{8.21}$$

Furthermore, recall from the embedding into ℓ_1, we have with probability at least 0.99 over $\mathbf{T}$,

$$\sum_{(u,v)\in E_{\mathbf{T}}} \Big||A_v| - |B_v|\Big| \cdot \frac{d}{2^i} \leq O(\log s) \cdot \mathrm{EMD}(A, B). \tag{8.22}$$

This follows from an analysis similar to (8.3) (although (8.3) only gives $O(\log d)$ on the right hand side instead of the $O(\log s)$ we need, one can improve it to $\min(\log s, \log d)$; see footnote 8 and an example of implementing this idea in Lemma 8.7.2).

Combining (8.21) and (8.22), we have that with probability at least 0.9 over the draw of $\mathbf{T}$,

$$\mathrm{EMD}(A, B) \leq \sum_{(u,v)\in E_{\mathbf{T}}} \Big||A_v| - |B_v|\Big| \cdot \boldsymbol{q}_{u,v} \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B), \tag{8.23}$$

where $\boldsymbol{q}_{u,v}$ is defined as the following truncation of $\|\mathbf{c}_u - \mathbf{c}_v\|_1$:

$$\boldsymbol{q}_{u,v} \stackrel{\text{def}}{=} \begin{cases} d/2^i & \text{if } \|\mathbf{c}_u - \mathbf{c}_v\|_1 \le d/2^i \\ 30d\log s/2^i & \text{if } \|\mathbf{c}_u - \mathbf{c}_v\|_1 \ge 30d\log s/2^i \\ \|\mathbf{c}_u - \mathbf{c}_v\|_1 & \text{otherwise.} \end{cases}$$

The sum in the center of (8.23) will be crucial in later sections; this will be the quantity we estimate in our linear sketches in Section 8.5 and 8.6.

8.4.1 Proof of Lemma 8.4.5

Recall $h = \log 2d$ is the depth of a compressed Quadtree. Let $(a,b) \in M^*$ with $a \neq b$ and $i_0 \in [h]$ that satisfy (8.16). We need to bound the expectation of $\widetilde{\mathbf{PAY}}_{i_0}(a)$ over the draw of a random compressed Quadtree $\mathbf{T}$; the upper bound for $\widetilde{\mathbf{PAY}}_{i_0}(b)$ is analogous. In what follows, all expectations are taken with respect to a random choice of $\mathbf{T}$ so we skip $\mathbf{T}$ in subscripts (just as in $\widetilde{\mathbf{PAY}}_{i_0}(a)$). In particular, we write $\mathbf{v}_i(a)$ to denote $v_{i,\mathbf{T}}(a)$ just to emphasize that it is a random variable that depends on $\mathbf{T}$.

Given that we always have $\|a - \tilde{\mathbf{c}}(i,a)\|_1 \le 10d\log s/2^i$ by definition, $\widetilde{\mathbf{PAY}}_{i_0}(a) = O(d\log s)$; hence, we assume that $\|a-b\|_1 \le d/2$, since otherwise the lemma is trivially true. To understand $\widetilde{\mathbf{PAY}}_{i_0}(a)$ for a random compressed Quadtree $\mathbf{T}$, we need to examine $A_{\mathbf{v}_i(a)}$ and $B_{\mathbf{v}_i(a)}$ where $\mathbf{v}_0(a), \ldots, \mathbf{v}_h(a)$ is the root-to-leaf path of a in $\mathbf{T}$. Let $\tau_0 = 1$ and $\tau_i = 2^{i-1}$ when $i \ge 1$. Recall from Lemma 8.3.3 that to draw $A_{\mathbf{v}_i(a)}$ and $B_{\mathbf{v}_i(a)}$, it suffices to consider independent draws

$$\mathbf{I}_0, \mathbf{I}_1, \ldots, \mathbf{I}_{h-2}$$

with $\mathbf{I}_i \sim [d]^{\tau_i}$, and then use them to define $A_{\mathbf{v}_i(a)}$ and $B_{\mathbf{v}_i(a)}$ as follows:

$$A_{\mathbf{v}_i(a)} = \{x \in A : x_j = a_j \text{ for all } j \text{ that appears in } \mathbf{I}_0, \ldots, \mathbf{I}_{i-1}\}$$
$$B_{\mathbf{v}_i(a)} = \{x \in B : x_j = a_j \text{ for all } j \text{ that appears in } \mathbf{I}_0, \ldots, \mathbf{I}_{i-1}\}$$

for each $i = 1, \ldots, h-1$; $A_{\mathbf{v}_0(a)} = A$ and $B_{\mathbf{v}_0(a)} = B$ since $\mathbf{v}_0(a)$ is always the root; $A_{\mathbf{v}_h(a)}$ contains all copies of a in the multi-set A and $B_{v_h(b)}$ contains all copies of a in B. We let

$$\mathbf{D} = \Big\{(i,\ell) : i \in \{0,\ldots,h-2\} \text{ and } \ell \in [\tau_i] \text{ such that } j = (\mathbf{I}_i)_\ell \text{ satisfies } a_j \neq b_j\Big\}$$

be the set of index pairs of sampled coordinates where a and b disagree, and let

$$(\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}) \stackrel{\text{def}}{=} \begin{cases} \min \mathbf{D} & \mathbf{D} \neq \emptyset \\ (*,*) & \mathbf{D} = \emptyset \end{cases},$$

where the ordering in $\min \mathbf{D}$ is lexicographic.[19] Note, in particular, that the node $\mathbf{v}_{\boldsymbol{i}^{(s)}}(a) = \mathbf{v}_{\boldsymbol{i}^{(s)}}(b)$ is the least common ancestor of a and b in $\mathbf{T}$.[20] The coordinate which is the first to satisfy $a_j \neq b_j$ is specified by the random variable

$$\boldsymbol{j}^{(s)} \stackrel{\text{def}}{=} \begin{cases} (\mathbf{I}_{\boldsymbol{i}^{(s)}})_{\boldsymbol{\ell}^{(s)}} & \text{if } (\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}) \neq (*,*) \\ * & \text{otherwise} \end{cases},$$

where, notice that, $\boldsymbol{j}^{(s)} = *$ only if a and b always satisfy $\mathbf{v}_k(a) = \mathbf{v}_k(b)$ for all $k \in \{0, \ldots, h-1\}$. A trivial consequence of the above definitions is that for any $k \in \{0, \ldots, h-1\}$,

$$\mathbf{1}\{\mathbf{v}_k(a) \neq \mathbf{v}_k(b)\} = \sum_{i=0}^{k-1} \sum_{\ell=1}^{\tau_i} \sum_{\substack{j \in [d] \\ a_j \neq b_j}} \mathbf{1}\left\{(\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i, \ell, j)\right\}. \tag{8.24}$$

From the definition of $\widetilde{\mathbf{PAY}}_{i_0}(a)$ in (8.15), letting

$$h_{a,b} \stackrel{\text{def}}{=} \left\lfloor \log_2 \left(\frac{d}{\|a-b\|_1} \right) \right\rfloor \leq \log_2 d = h - 1$$

and we have

$$\begin{aligned} &\mathbf{E}\left[\widetilde{\mathbf{PAY}}_{i_0}(a)\right] \\ &\leq \sum_{k=i_0}^{h_{a,b}} \mathbf{E}\left[\mathbf{1}\{\mathbf{v}_k(a) \neq \mathbf{v}_k(b)\}\left(\|a - \tilde{\mathbf{c}}(k,a)\|_1 + \|a - \tilde{\mathbf{c}}(k-1,a)\|_1\right)\right] + O(\log s) \cdot \|a-b\|_1, \end{aligned}$$

where we used the fact that $\|a - \tilde{\mathbf{c}}(k,a)\|_1 \leq 10d \log s / 2^k$ always holds. As a result $O(\log s) \cdot \|a-b\|_1$ as in the last term is enough to cover the sum over $k > h_{a,b}$ skipped in the above

[19] All (i', ℓ') with $i' < \boldsymbol{i}^{(s)}$ satisfy $a_j = b_j$ for $j = (\mathbf{I}_{i'})_{\ell'}$; all $(\boldsymbol{i}^{(s)}, \ell')$ with $\ell' < \boldsymbol{\ell}^{(s)}$ satisfy $a_j = b_j$ for $j = (\mathbf{I}_{\boldsymbol{i}^{(s)}})_{\ell'}$.

[20] Hence, the "s" in $\boldsymbol{i}^{(s)}$ and $\boldsymbol{\ell}^{(s)}$ stands for "split."

expression. Using (8.24), we may re-write the first summand above as

$$\sum_{k=i_0}^{h_{a,b}} \mathbf{E}\left[\mathbf{1}\{\mathbf{v}_k(a) \neq \mathbf{v}_k(b)\}\left(\|a - \widetilde{\mathbf{c}}(k,a)\|_1 + \|a - \widetilde{\mathbf{c}}(k-1,a)\|_1\right)\right] \tag{8.25}$$

$$= \sum_{k=i_0}^{h_{a,b}} \sum_{i=0}^{k-1} \sum_{\ell=1}^{\tau_i} \sum_{\substack{j\in[d]\\ a_j\neq b_j}} \mathbf{Pr}\left[(\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j)\right] \left(\mathbf{E}\left[\|a - \widetilde{\mathbf{c}}(k,a)\|_1 \,\middle|\, (\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j)\right] + \mathbf{E}\left[\|a - \widetilde{\mathbf{c}}(k-1,a)\|_1 \,\middle|\, (\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j)\right]\right).$$

Notice that for $i \in \{0,\dots,h-2\}$, $\ell \in [\tau_i]$, and $j \in [d]$ with $a_j \neq b_j$,

$$\mathbf{Pr}\left[(\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j)\right] = \frac{1}{d}\left(1 - \frac{\|a-b\|_1}{d}\right)^{\tau_0+\cdots+\tau_{i-1}+\ell-1}. \tag{8.26}$$

Consider, for each $k \in \{i_0 - 1, \dots, h_{a,b}\}$ and $j \in [d]$ with $a_j \neq b_j$, the value

$$Q_{k,j} \stackrel{\text{def}}{=} \sup_{\substack{i\in\{0,\dots,k\}\\ \ell\in[\tau_i]}} \mathbf{E}\left[\|a - \widetilde{\mathbf{c}}(k,a)\|_1 \,\middle|\, (\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j)\right],$$

so that invoking (8.26), we may upper bound (8.25) by

$$\sum_{\substack{j\in[d]\\ a_j\neq b_j}} \sum_{k=i_0}^{h_{a,b}} \left(Q_{k,j} + Q_{k-1,j}\right) \sum_{i=0}^{k-1}\sum_{\ell=1}^{\tau_i} \frac{1}{d}\left(1 - \frac{\|a-b\|_1}{d}\right)^{\tau_0+\cdots+\tau_{i-1}+\ell-1} \leq \frac{4}{d} \sum_{\substack{j\in[d]\\ a_j\neq b_j}} \sum_{k=i_0-1}^{h_{a,b}} Q_{k,j}\cdot 2^k.$$

Therefore, it remains to show that for all $j \in [d]$ with $a_j \neq b_j$,

$$\sum_{k=i_0-1}^{h_{a,b}} Q_{k,j}\cdot 2^k = O(d)\cdot\left(\tilde{O}(\log s) + (h_{a,b} - i_0)\cdot \log\log s\right). \tag{8.27}$$

To prove this, we start with a lemma that shows that if $Q_{k,j}$ is large, then it must be the case that there are many points of distance between (roughly) $d/(2^k \log s)$ and $d\log s/2^k$ from a in $A \cup B$.

Lemma 8.4.6. *Fix $j \in [d]$ which satisfies $a_j \neq b_j$ and $k \in \{2,\dots,h_{a,b}\}$. Let $L \subset A_j \cup B_j \subset$*

$A \cup B$ *be the muti-sets given by*

$$A_j = \{x \in A : x_j = a_j\}, \quad B_j = \{x \in B : x_j = a_j\} \quad \textit{and}$$
$$L = \left\{ x \in A_j \cup B_j : \|a - x\|_1 \leq \frac{d}{2^k} \cdot \frac{1}{\log s} \right\}.$$

Suppose that for some $i = \{0, \ldots, k\}$ *and some* $\ell \in [\tau_i]$*, as well as some* $\alpha \geq 300$*, we have*

$$\mathbf{E}\left[\|a - \widetilde{\mathbf{c}}(k,a)\|_1 \,\middle|\, (\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j) \right] \geq \frac{d}{2^k} \cdot \alpha. \tag{8.28}$$

Then, the set

$$H = \left\{ x \in A_j \cup B_j : \|a - x\|_1 \leq \frac{10d \log s}{2^k} \right\} \quad \textit{satisfies} \quad |H| \geq \frac{|L| \exp(\alpha/8)}{\log s}.$$

Proof. Let $\boldsymbol{\mathcal{E}}$ be the event that $(\boldsymbol{i}^{(s)}, \boldsymbol{\ell}^{(s)}, \boldsymbol{j}^{(s)}) = (i,\ell,j)$. For simplicity in the notation, let

$$\mathbf{v} = \mathbf{v}_k(a), \quad \mathbf{C} = \mathbf{C}(k,a) \quad \text{and} \quad \mathbf{c} = \widetilde{\mathbf{c}}(k,a).$$

Every $x \in L$ is of distance at most $d/(2^k \log s)$ from a. Since $x \in \mathbf{C}$ whenever $\mathbf{v}_k(x) = \mathbf{v}$, we have

$$\mathbf{E}\left[|L \setminus \mathbf{C}| \,\middle|\, \boldsymbol{\mathcal{E}} \right] = \sum_{x \in L} \mathbf{Pr}\left[\mathbf{v}_k(x) \neq \mathbf{v} \,\middle|\, \boldsymbol{\mathcal{E}} \right] \leq |L| \cdot 2^{k-1} \cdot \frac{d/(2^k \log s)}{d - \|a-b\|_1} \leq \frac{|L|}{\log s}, \tag{8.29}$$

because there are 2^{k-1} coordinates sampled up to (but not including) depth k. The last inequality above also used the assumption that $\|a - b\|_1 \leq d/2$. Then, we have

$$\mathbf{Pr}\left[|\mathbf{C}| \geq \frac{|L|}{10} \,\middle|\, \boldsymbol{\mathcal{E}} \right] \geq \mathbf{Pr}\left[|L \cap \mathbf{C}| \geq \frac{|L|}{10} \,\middle|\, \boldsymbol{\mathcal{E}} \right] = 1 - \mathbf{Pr}\left[|L \setminus \mathbf{C}| \geq \frac{9|L|}{10} \,\middle|\, \boldsymbol{\mathcal{E}} \right] \geq 1 - \frac{10}{9 \log s}, \tag{8.30}$$

where the last inequality follows by applying Markov's inequality to (8.29). Hence, by an application of triangle inequality, as well as the fact that $|\mathbf{C}| \geq 1$ (since it always contains a),

$$\|a - \mathbf{c}\|_1 \leq \frac{1}{|\mathbf{C}|} \sum_{x \in \mathbf{C}} \|a - x\|_1 \leq \frac{d\alpha}{2^{k+1}} + \frac{1}{|\mathbf{C}|} \sum_{x \in \mathbf{C}} \mathbf{1}\left\{ \|a - x\|_1 \geq \frac{d\alpha}{2^{k+1}} \right\} \cdot \|a - x\|_1.$$

Thus, we have (by splitting into two cases of $|\mathbf{C}| \geq |L|/10$ and $|\mathbf{C}| < |L|/10$ and applying

(8.30))

$$\mathbf{E}\left[\|a-\mathbf{c}\|_1 \,\middle|\, \boldsymbol{\mathcal{E}}\right] \le \frac{d\alpha}{2^{k+1}} + \mathbf{E}\left[\frac{1}{|\mathbf{C}|}\sum_{x\in\mathbf{C}} \mathbf{1}\left\{\|a-x\|_1 \ge \frac{d\alpha}{2^{k+1}}\right\}\|a-x\|_1 \,\middle|\, \boldsymbol{\mathcal{E}}\right]$$
$$\le \frac{d\alpha}{2^{k+1}} + \frac{10}{|L|}\cdot\mathbf{E}\left[\sum_{x\in\mathbf{C}} \mathbf{1}\left\{\|a-x\|_1 \ge \frac{d\alpha}{2^{k+1}}\right\}\|a-x\|_1 \,\middle|\, \boldsymbol{\mathcal{E}}\right] + \frac{10}{9\log s}\cdot\frac{10d\log s}{2^k},$$

where the final term used the fact that $\|a-\mathbf{c}\|_1$ is always at most $10d\log s/2^k$. Combining (8.28) and the inequality above, we have

$$\frac{d}{2^{k+1}}\left(\frac{\alpha}{10}-\frac{20}{9}\right)|L| \le \mathbf{E}\left[\sum_{x\in\mathbf{C}} \mathbf{1}\left\{\|a-x\|_1 \ge \frac{d\alpha}{2^{k+1}}\right\}\|a-x\|_1 \,\middle|\, \boldsymbol{\mathcal{E}}\right], \tag{8.31}$$

and we next upperbound the right-hand side above in terms of the size of H. In particular, let H' be the set of points $x \in H$ with $\|a-x\|_1 \ge d\alpha/2^{k+1}$. Then

$$\mathbf{E}\left[\sum_{x\in\mathbf{C}} \mathbf{1}\left\{\|a-x\|_1 \ge \frac{d\alpha}{2^{k+1}}\right\}\|a-x\|_1 \,\middle|\, \boldsymbol{\mathcal{E}}\right] \le \sum_{x\in H'} \mathbf{Pr}\left[x\in\mathbf{C} \,\middle|\, \boldsymbol{\mathcal{E}}\right]\cdot\left(\frac{10d\log s}{2^k}\right). \tag{8.32}$$

We consider two cases: $i=k$ and $i<k$. For the easier case when $i=k$, in order for $x\in\mathbf{C}$ to occur conditioned on $\boldsymbol{\mathcal{E}}$, we have that all 2^{k-1} coordinates sampled before depth k avoid separating x and a conditioning on not separating a and b. The conditional probability is at most

$$\frac{d-\|a-x\|_1}{d-\|a-b\|_1} = 1-\frac{\|a-x\|_1-\|a-b\|_1}{d-\|a-b\|_1} \le 1-\frac{\|a-x\|_1-\|a-b\|_1}{d}.$$

Therefore in this case we have

$$\Pr\left[x\in\mathbf{C} \,\middle|\, \boldsymbol{\mathcal{E}}\right] \le \left(1-\frac{\|a-x\|_1-\|a-b\|_1}{d}\right)^{2^{k-1}}.$$

Notice that by definition of $h_{a,b}$, and the fact that $k \le h_{a,b}$, we have

$$\|x-a\|_1 - \|a-b\|_1 \ge \frac{d}{2^{k+1}}(\alpha-2) \tag{8.33}$$

which implies that $\Pr[x\in\mathbf{C}\,|\,\boldsymbol{\mathcal{E}}]$ is at most $\exp(-\alpha/8)$ using $\alpha\ge 300$.

Next we deal with the case when $i<k$. In order for $x\in\mathbf{C}$ to occur conditioned on $\boldsymbol{\mathcal{E}}$, it needs to be the case that all coordinates sampled before (i,ℓ), of which there are $2^{i-1}+\ell-1$ many (or none if $(i,\ell)=(0,1)$), avoid separating x and a conditioning on not separating a and b;

the (i,ℓ)-th sample is j (which better not separate x and a; otherwise the probability is trivially 0); and the remaining $2^{k-1} - 2^{i-1} - \ell$ (or $2^{k-1} - 1$ if $(i,\ell) = (0,1)$) coordinates do not separate x and a (but there will be conditioning on not separating a and b). Hence,

$$\begin{aligned}\mathbf{Pr}\left[x \in \mathbf{C} \,\middle|\, \boldsymbol{\mathcal{E}}\right] &\leq \left(1 - \frac{\|x-a\|_1 - \|a-b\|_1}{d}\right)^{2^{i-1}+\ell-1} \left(1 - \frac{\|x-a\|_1}{d}\right)^{2^{k-1}-2^{i-1}-\ell} \\ &\leq \left(1 - \frac{\|x-a\|_1 - \|a-b\|_1}{d}\right)^{2^{k-1}-1} \\ &\leq \left(1 - \frac{\alpha - 2}{2^{k+1}}\right)^{2^{k-1}-1}, \end{aligned} \tag{8.34}$$

which is at most $\exp(-\alpha/8)$ using $k \geq 2$ and $\alpha \geq 300$. Hence, we can combine (8.31) and (8.32) to get

$$\frac{d}{2^{k+1}}\left(\frac{\alpha}{10} - \frac{20}{9}\right)|L| \leq |H| \cdot \exp\left(-\frac{\alpha}{8}\right) \cdot \frac{10d\log s}{2^k}.$$

Re-arranging the inequality, the lemma follows using $\alpha \geq 300$. □

The next lemma helps bound the number of large $Q_{k,j}$'s.

Lemma 8.4.7. *Fix any $j \in [d]$ with $a_j \neq b_j$. For any $\alpha \geq 20\log\log s$, the set*

$$G_j(\alpha) = \left\{k \in \{0,\ldots,h_{a,b}\} : Q_{k,j} \geq \frac{d}{2^k} \cdot \alpha\right\} \quad \textit{satisfies} \quad \left|G_j(\alpha)\right| \leq O\left(\left\lceil \frac{16\log(2s)}{\alpha}\right\rceil \cdot \log\log s\right).$$

Proof. Assume for a contradiction that

$$\left|G_j(\alpha)\right| \geq \beta \cdot \left\lceil \log_2(10\log^2 s)\right\rceil + 2, \quad \text{where} \quad \beta \stackrel{\text{def}}{=} \left\lceil \frac{16\log(2s)}{\alpha}\right\rceil.$$

Then there must be $k_1,\ldots,k_\beta \in \{2,\ldots,h_{a,b}\}$ with $k_1 > k_2 > \cdots > k_\beta$ and every $t \in [\beta - 1]$ satisfies

$$k_t - k_{t+1} \geq \left\lceil \log_2(10\log^2 s)\right\rceil.$$

This implies that every $t \in [\beta - 1]$ satisfies

$$\frac{10d\log s}{2^{k_t}} \leq \frac{d}{2^{k_{t+1}}\log s}.$$

If we consider for each $t \in [\beta]$, the following two multi-sets

$$L_t = \left\{ x \in A_j \cup B_j : \|a - x\|_1 \le \frac{d}{2^{k_t} \log s} \right\} \quad \text{and} \quad H_t = \left\{ x \in A_j \cup B_j : \|a - x\|_1 \le \frac{10d \log s}{2^{k_t}} \right\},$$

they satisfy

$$L_1 \subseteq H_1 \subseteq L_2 \subseteq H_2 \subseteq \cdots \subseteq H_{\beta-1} \subseteq L_\beta \subseteq H_\beta, \tag{8.35}$$

but then invoking Lemma 8.4.6 (and using $20 \log\log s \ge 300$), we have that every $t \in [\beta]$ satisfy

$$|H_t| \ge \frac{|L_t| \exp(\alpha/8)}{\log s}.$$

Using $|L_1| \ge 1$ (since it contains a) and (8.35), we have

$$|H_\beta| \ge \frac{\exp(\alpha\beta/8)}{(\log s)^\beta} > 2s,$$

using $\exp(\alpha\beta/8) \ge (2s)^2$, and $(\log s)^\beta \le 2s$, by our settings of β and α. This is a contradiction, as we have $H_\beta \subseteq A \cup B$ and thus, $|H_\beta| \le 2s$. □

Finally we finish the proof of (8.27). Let $\alpha_0 = 20 \log\log s$. Use Lemma 8.4.7 we have

$$\begin{aligned}\sum_{k=i_0-1}^{h_{a,b}} Q_{k,j} \cdot 2^k &\le (h_{a,b} - i_0 + 2) \cdot \alpha_0 d + \sum_{\kappa=0}^{\lceil \log_2(10 \log s) \rceil} \left| G_j(\alpha_0 2^\kappa) \right| \cdot \alpha_0 2^{\kappa+1} \cdot d \\ &\le O(d) \cdot \Big((h_{a,b} - i_0) \cdot \log\log s + \log s \cdot (\log\log s)^3 \Big),\end{aligned}$$

where the upper limit of $\kappa \le \lceil \log_2(10 \log s) \rceil$ comes from the fact that $G_j(10 \log s)$ is empty because $\|a - \widetilde{\mathbf{c}}(k, a)\|_1$ is always at most $10d \log s / 2^k$ by definition. This completes Lemma 8.4.5.

8.5 Two-Round Linear Sketch

In this section and the following, we leverage our improved analysis to design linear sketches for EMD over $\{0,1\}^d$; the extension to EMD over $(\mathbb{R}^d, \|\cdot\|_p)$ for $p \in [1,2]$ follows from a standard application of metric embeddings (see Section 8.11). Specifically, we first demonstrate how a $\tilde{O}(\log s)$ approximation can be obtained using $\mathrm{polylog}(s, d)$ bits of space in a *two-round* linear sketching model (we give a formal description shortly). In the subsequent section, we implement

the linear sketch in a single round, at the cost of a small additive error. In cases where sets A, B do not substantially overlap, the one-round protocol yields the same multiplicative guarantee. As is relatively standard in the sketching literature, the two-round and one-round linear sketch may be used for two-round and one-round communication settings, as well as two-pass and one-pass streaming algorithms (see Section 8.12 for a more thorough explanation of the model).

Linear Sketching for EMD over $\{0,1\}^d$. For some $m \in \mathbb{N}$, we encode the inputs of a computational problem as a vector f in $\mathbb{R}^m$. For that encoding and a parameter $k \in \mathbb{N}$, a *linear sketch* is a distribution over $k \times m$ matrices $\mathbf{S}$, where ideally $k \ll m$, accompanied by an algorithm $\mathrm{Alg}_{\mathbf{S}}$, which receives an input $\mathbf{S}f \in \mathbb{R}^k$ and outputs an answer to the computational problem. The algorithm maintains the vector $\mathbf{S}x$ in k words of space and utilizes $\mathrm{Alg}_{\mathbf{S}}$ to produce an output. In the public coin model, storing the random matrix $\mathbf{S}$ is not counted against the space complexity of the algorithm ($\mathbf{S}$ can often be stored in small space, such as for streaming, see Corollary 8.12.1). In order to define linear sketching for EMD over size-s subsets of $\{0,1\}^d$, we must specify the encoding in $\mathbb{R}^m$, as well as the space complexity k, and the distribution over $k \times m$ matrices $\mathbf{S}$ and decoding algorithms.

A pair of multi-sets $A, B \subset \{0,1\}^d$ is encoded as a vector $f_{A,B} \in \mathbb{R}^{2 \cdot 2^d}$, where the first 2^d coordinates represent the indicator vector of A in $\{0,1\}^d$, and the second 2^d coordinates represent the indicator vector of B in $\{0,1\}^d$. More precisely, for $i \in \{0,1\}^d$, the i-th coordinate (in standard binary encoding) of $f_{A,B}$ is the number of points in A at i; the $(2^d + i)$-th coordinate of $f_{A,B}$ is the number of points in B at i.

The two-round linear sketch is defined by a linear sketch which produces an intermediate output for the first round, and another linear sketch (parametrized by the output of the first round) which produces the final output. Specifically, we have distribution $\mathcal{D}^{(1)}$ supported on $k \times 2 \cdot 2^d$ matrices, as well a decoding algorithm for a vector $\boldsymbol{y}_1 = \mathbf{S}_1 f_{A,B} \in \mathbb{R}^k$. In one-round sketching, the output is produced by the decoding algorithm. For two-round round linear sketching, we consider another distribution $\mathcal{D}^{(2)}$, parametrized by the decoding of $\boldsymbol{y}_1$, as well as $\mathbf{S}_1$, supported on $k \times 2 \cdot 2^d$ matrices $\mathbf{S}_2$, as well as a decoding algorithm for a vector $\boldsymbol{y}_2 = \mathbf{S}_2 f_{A,B} \in \mathbb{R}^k$ which produces the output.

8.5.1 The Two-Round Linear Sketching Algorithm

In this section, we give a two-round linear sketching algorithm for approximating the Earth Movers distance. Recall that for a linear sketch, the multisets $A, B \subset \{0,1\}^d$ are *implicitly*

encoded as a vector $f_{A,B} \in \mathbb{R}^{2^{d+1}}$ which indicates the members and frequencies of the points in A, B. Specifically, we prove the following theorem.

Theorem 99. *For $d, s \in \mathbb{N}$, there exists a 2-round linear sketching algorithm such that given multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$, computes an approximation $\hat{\boldsymbol{\mathcal{I}}}$ to $\mathrm{EMD}(A, B)$ with*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\mathcal{I}}} \leq \tilde{O}(\log s)\,\mathrm{EMD}(A, B)$$

with probability at least $3/4$. Moreover, the space of the linear sketch is $\mathrm{polylog}(s, d)$ bits.

The protocol will proceed by sampling a compressed quadtree $\mathbf{T}$ of depth $h = \log_2(2d)$, and for each depth $i \in \{0, \ldots, h-1\}$, using L_1-sampling and L_1-sketching to estimate

$$\boldsymbol{\mathcal{I}}_i \stackrel{\text{def}}{=} \sum_{v \in \mathcal{V}_i} \Big||A_v| - |B_v|\Big| \cdot \boldsymbol{q}_{\pi(v),v},$$

up to a constant factor. We recall that for a node v in the tree, the multi-sets

$$A_v = \{a \in A : \mathbf{v}_i(a) = v\} \qquad \text{and} \qquad B_v = \{b \in B : \mathbf{v}_i(b) = v\}$$

are the points from A and B which are mapped to node v in the compressed quadtree $\mathbf{T}$. The point

$$\mathbf{c}_v = \frac{1}{|A_v \cup B_v|} \sum_{x \in A_v \cup B_v} x,$$

is the center of mass among all points which map to node v. We have that

$$\boldsymbol{q}_{u,v} = \begin{cases} \frac{d}{2^i} & \|\mathbf{c}_u - \mathbf{c}_v\|_1 \leq \frac{d}{2^i} \\ \frac{30d \log s}{2^i} & \|\mathbf{c}_u - \mathbf{c}_v\|_1 \geq \frac{30d \log s}{2^i} \\ \|\mathbf{c}_u - \mathbf{c}_v\|_1 & \text{o.w.} \end{cases}. \tag{8.36}$$

Recall that by Remark 98,

$$\mathrm{EMD}(A, B) \leq \sum_{i=0}^{h-1} \boldsymbol{\mathcal{I}}_i \leq \tilde{O}(\log s) \cdot \mathrm{EMD}(A, B),$$

with probability at least 0.89 over the draw of $\mathbf{T}$. Hence, the protocol produces an estimate $\hat{\boldsymbol{\mathcal{I}}}_i \in \mathbb{R}$ where $\mathcal{I}_i \leq \hat{\boldsymbol{\mathcal{I}}}_i \leq O(1) \cdot \mathcal{I}_i$ with probability at least $1 - 1/(100h)$. Via a union bound

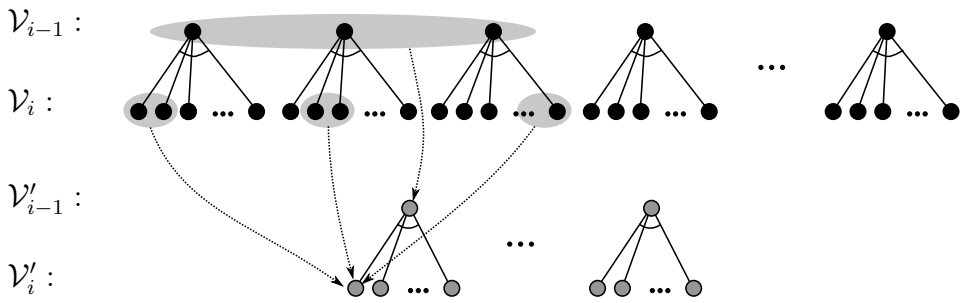


Figure 8.3: An example of the universe reduction from the bipartite graph defined on $(\mathcal{V}_{i-1}, \mathcal{V}_i)$ to the bipartite graph defined on $(\mathcal{V}'_{i-1}, \mathcal{V}'_i)$, as well as the corresponding mapping from $\boldsymbol{h}_{i-1}$ and $\boldsymbol{h}_i$. The shaded regions correspond to pre-images of the node mappings.

over all h levels of the tree as well as the good draw of $\mathbf{T}$, we have that with probability at least 0.88, $\sum_{i=0}^{h-1} \hat{\boldsymbol{\mathcal{I}}}_i$ satisfies the requirements of Theorem 99. See Figure 8.4 for a description of the protocol which estimates $\boldsymbol{\mathcal{I}}_i$ up to constant factor with high (constant) probability.

Universe Reduction

Observe that $\boldsymbol{\mathcal{I}}_i$ is a sum over $|\mathcal{V}_i| = 2^{2^i}$ vertices of $\mathbf{T}$. However, since $|A| = |B| = s$, at most $2s$ of these summands will be non-zero. Thus, we perform a standard universe reduction on the layer $(\mathcal{V}_{i-1}, \mathcal{V}_i)$ as follows. Select two fully independent hash functions $\boldsymbol{h}_{i-1} : [\mathcal{V}_{i-1}] \to [s']$, $\boldsymbol{h}_i : [\mathcal{V}_i] \to [s']$, where $s' = O(s^3)$. We then define

$$\mathcal{V}'_{i-1} = [s'], \qquad \text{and} \qquad \mathcal{V}'_i = [s'] \times [s'],$$

and construct the bipartite graph $(\mathcal{V}'_{i-1}, \mathcal{V}'_i)$ corresponding to layers $i-1$ and i of $\mathbf{T}$ after a universe reduction. The nodes of $\mathcal{V}'_{i-1}$ represent groups of nodes in $\mathcal{V}'_{i-1}$ under the mapping $\boldsymbol{h}_{i-1}$. The nodes of $\mathcal{V}'_i$ are indexed by pairs $(y, z) \in [s'] \times [s']$, where the index y specifies the group of the parent node (under $\boldsymbol{h}_{i-1}$), and z represents group of child nodes in $\mathcal{V}_i$. The edges are added to preserve parent-child relationships in $\mathcal{V}'_{i-1}$ and $\mathcal{V}'_i$: $(x, (y, z)) \in \mathcal{V}'_{i-1} \times \mathcal{V}'_i$ is added as an edge if $x = y$ for $x, y, z \in [s']$. (See Figure 8.3.)

For $x \in \mathcal{V}'_{i-1}$ and $(y, z) \in \mathcal{V}'_i$, we can naturally define

$$A_x = \{a \in A : \boldsymbol{h}_{i-1}(\mathbf{v}_{i-1}(a)) = x\} \quad A_{(y,z)} = \{a \in A : \boldsymbol{h}_i(\mathbf{v}_i(a)) = z, \boldsymbol{h}_{i-1}(\mathbf{v}_{i-1}(a)) = y\}$$

We similarly define B_u, B_v for $u \in \mathcal{V}'_{i-1}, v \in \mathcal{V}'_i$. We can also define centers $\mathbf{c}'_v, \mathbf{c}'_u$ of the sets $A_u \cup B_u, A_v \cup B_v$ where $u \in \mathcal{V}'_{i-1}, v \in \mathcal{V}'_i$, and define $\boldsymbol{q}'_{\pi(v),v}$ identically to $\boldsymbol{q}'_{\pi(v),v}$ but using the

centers $\mathbf{c}'_v$. Altogether, we can set

$$\boldsymbol{\mathcal{I}}'_i \stackrel{\text{def}}{=} \sum_{v \in \mathcal{V}'_i} \Big| |A_v| - |B_v| \Big| \cdot \boldsymbol{q}'_{\pi(v),v}$$

where $\pi(v) \in \mathcal{V}'_{i-1}$ is the parent of v in the new tree. Notice that since both $\mathcal{V}_i, \mathcal{V}_{i-1}$ have at most $2s$ non-empty nodes each, with probability $1 - 1/s$ we have that $\boldsymbol{h}_{i-1}$ and $\boldsymbol{h}_i$ are injective into $[s']$ and $[s']^2$ respectively on the non-empty nodes (i.e., the non-empty nodes perfectly hash). Since $\boldsymbol{\mathcal{I}}_i$ is just a sum over edges, if the non-empty edges perfectly hash, the construction of $\boldsymbol{\mathcal{I}}'_i$ just amounts to a renaming of the non-zero edges.

Proposition 8.5.1. *With probability $1 - 1/s$ over the choice of $\boldsymbol{h}_{i-1}, \boldsymbol{h}_i$, we have $\boldsymbol{\mathcal{I}}_i = \boldsymbol{\mathcal{I}}'_i$. Moreover, the hash functions $\boldsymbol{h}_{i-1}, \boldsymbol{h}_i$ need only be 2-wise independent.*

Proof. This simply follows from the fact that s' is at least $8s^3$, and there are at most $2s$ nodes $v \in \mathcal{V}_{i-1}$ where $A_v \cup B_v \neq \emptyset$. Hence, the probability that two non-empty nodes collide in $\boldsymbol{h}_{i-1}$ is at most $1/(8s^3)$ (where we use 2-wise independence), and we may union bound over at most $4s^2$ pairs of non-empty nodes. The same argument can be made that $\boldsymbol{h}_i$ perfectly hashes all non-empty nodes in $\mathcal{V}_i$ with probability $1 - 1/(2s)$. Conditioned on this, there is a bijection between non-zero terms in $\boldsymbol{\mathcal{I}}_i$ and $\boldsymbol{\mathcal{I}}'_i$, where terms mapped to each other are equal. □

Proposition 8.5.1 demonstrates that it will be sufficient to estimate $\boldsymbol{\mathcal{I}}'_i$. Thus, we condition on the success of this universe reduction step now. Since conditioned on this step the non-zero edges $(\mathcal{V}_{i-1}, \mathcal{V}_i)$ are in bijective correspondence with the non-zero edges of $(\mathcal{V}'_{i-1}, \mathcal{V}'_i)$, and moreover since under this equivelance the points contained in A_v, B_v and the centers $\mathbf{c}_v$ are equivalant, in the following we will abuse notation and drop the prime notation in $\boldsymbol{\mathcal{I}}'_i, \mathcal{V}'_i, \mathcal{V}'_{i-1}$, and simply write $\boldsymbol{\mathcal{I}}_i, \mathcal{V}_i, \mathcal{V}_{i-1}$, with the understanding that the universe reduction step has already been carried out.

Sketching Tools

We recall two standard sketching tools occurring in this thesis: the L_1 sketch of [Ind06] (Theorem 8), and the perfect L_1 sampler from Chapter 3. Further recall that both of these algorithms are linear sketches. In order to simplify the notation involved in the recovery procedures of these two algorithms, we black-box the recovery procedures via notation, and reformulate these two results with this simplified notation.

Theorem 100 (L_1-sketch, reformulation of Theorem 8)**.** *For $m \in \mathbb{N}$ and $\delta \in (0, 1/2)$, let*

$t = O(\log(1/\delta))$. *There exists a distribution* $\mathcal{S}_k^1(m,t)$ *supported on* $t \times m$ *matrices* $\mathbf{C}$*, as well as a recovery algorithm* Alg^1 *which receives a vector* $y \in \mathbb{R}^t$ *and outputs a real number. For any fixed* $x \in \mathbb{R}^m$*,*

$$\|x\|_1 \leq \mathrm{Alg}^1(\mathbf{C}x) \leq 2\|x\|_1,$$

with probability at least $1-\delta$ *over the draw of* $\mathbf{C} \sim \mathcal{S}_k^1(m,t)$*.*[21]

Theorem 101 (Perfect L_1-sampling, reformulation of Theorem 3)**.** *For* $m \in \mathbb{N}$*, let* $c > 1$ *be an arbitrarily large constant and* $t = O(\log^2 m)$*. There exists a distribution* $\mathcal{S}_a^1(m,t)$ *supported on pairs* $(\mathbf{S}, \mathrm{Alg}_{\mathbf{S}}^1)$ *where* $\mathbf{S}$ *is an* $t \times m$ *matrix and* $\mathrm{Alg}_{\mathbf{S}}^1$ *is an algorithm which receives as input a vector* $y \in \mathbb{R}^t$ *and outputs a failure symbol* $\perp$ *with probability at most* $1/3$*, otherwise it returns an index* $j \in [m]$*. For any* $x \in \mathbb{R}^m$ *and* $j \in [m]$*,*

$$\left| \mathop{\mathbf{Pr}}_{(\mathbf{S},\mathrm{Alg}_{\mathbf{S}}^1)\sim\mathcal{S}_a^1(m,t)} \left[\mathrm{Alg}_{\mathbf{S}}^1(\mathbf{S}x) = j \mid \mathrm{Alg}_{\mathbf{S}}^1(\mathbf{S}x) \neq \perp \right] - \frac{|x_j|}{\|x\|_1} \right| \leq \frac{1}{m^c}.$$

Description and Analysis of Two-Round Sketch

We give a description of the two-round protocol, and the precise formulation is given in Figure 8.4. As we explain the protocol we specify some claims, which are then combined to prove Theorem 99.

Round 0**.** We begin by mapping the vector $f_{A,B} \in \mathbb{R}^{2^{d+1}}$ to a vector $f^i \in \mathbb{R}^{|\mathcal{V}_i| \times \{A,B\}}$ indexed by vertices $v \in \mathcal{V}_i$ and $\{A,B\}$ via

$$f_{v,A}^i = \sum_{\substack{\vec{x}\in\{0,1\}^d \\ v_i(\vec{x})=v}} (f_{A,B})_x = |A_v| \quad \text{and} \quad f_{v,B}^i = \sum_{\substack{\vec{x}\in\{0,1\}^d \\ v_i(\vec{x})=v}} (f_{A,B})_{x+2^d} = |B_v|$$

where $\vec{x} \in \{0,1\}^d$ and $x \in [2^d]$ is the index in $[2^d]$ that $\vec{x}$ encodes. In other words, $f_{v,A}^i$ and $f_{v,B}^i$ hold the frequency counts of points in A and B, respectively, at vertex $v \in \mathcal{V}_i$, which is a linear mapping of the input $f_{A,B}$. Recall we assume the universe reduction has already been performed, so $|\mathcal{V}_i| = O(s^6)$. At this point, we initialize sketches to perform L_1-sampling and L_1-sketching.

[21] The notation for the distribution $\mathcal{S}_k^1$, is for L_1-sketching; the 1 in the superscript for the L_1 aspect, and k in the subscript is for "sketching". Later, we will see analogous $\mathcal{S}_k^\infty$ for "ℓ_∞-sketching" (i.e., Count-Sketch), and $\mathcal{S}_a^1$ for L_1-sampling.

Round 1. At the end of round 1, the estimate $\hat{\beta} \in \mathbb{R}$ is the result of an L_1-sketch on the vector F^i. In other words, we will have that $\hat{\beta}$ is a constant factor approximation to $\|F^i\|_1$. Furthermore, the indices $v_1, \ldots, v_k \in \mathcal{V}_i \cup \{\perp\}$ represent L_1-samples from F^i; while the L_1-sample does not fail (i.e., $v_t \neq \perp$), the sample $v_t \in \mathcal{V}_i$ is meant to be distributed proportional to $F^i_{v_t}$. Specifically, we have the following two claims:

Claim 8.5.2 (Round 1 Claim A)**.** *Let $\boldsymbol{\mathcal{E}}_1$ be the event, defined over the randomness of the sample $\mathbf{C} \sim \mathcal{S}^1_k((s')^3, O(1))$ in Line 3 of Round 0 that in Line 2 of Round 1,*

$$\sum_{v \in \mathcal{V}_i} \Big| |A_v| - |B_v| \Big| \leq \hat{\beta} \leq 2 \sum_{v \in \mathcal{V}_i} \Big| |A_v| - |B_v| \Big|. \tag{8.37}$$

Then, $\boldsymbol{\mathcal{E}}_1$ occurs with probability at least 0.99.

Proof. This is a consequence of Theorem 100 with $m = (s')^2$, $\delta = .01$, applied to the vector F^i which has L_1 norm $\|F^i\|_1 = \sum_{v \in \mathcal{V}_i} ||A_v| - |B_v||$ □

Claim 8.5.3 (Round 1 Claim B)**.** *The random variables $v_1, \ldots, v_k$ defined in Line 2 are independent and identically distributed. We have that $v_t = \perp$ with probability at most $1/3$ for each t. Otherwise, for any $v \in \mathcal{V}_i$ we have*

$$\left| \mathbf{Pr}\left[v_t = v\right] - \frac{\Big| |A_v| - |B_v| \Big|}{\sum_{v' \in \mathcal{V}_i} \Big| |A_{v'}| - |B_{v'}| \Big|} \right| \leq \frac{1}{s^{10}}.$$

Proof. We first note that independence follows from the fact that the random variables $v_1, \ldots, v_k$ depend on the independent samples $(\mathbf{S}_t, \mathrm{Alg}^1_{\mathbf{S}_t}) \sim \mathcal{S}^1_a((s')^2, O(\log^2 s))$ for $t \in [k]$ of Line 2 of Round 0. The distributional bound then follows from Theorem 101 with $c = 10$, applied to the input vector F^i.

□

Round 2. The second round is meant to communicate, for each $t \in [k]$, the information necessary to estimate $\boldsymbol{q}_{\pi(v_t), v_t}$. In order to do this, the players must jointly approximate $\|c_{\pi(v_t)} - c_{v_t}\|_1$. For each $t \in [k]$, n_t is the number of points in A and B which lie in $\pi(v_t)$, and notice that, once both players know v_t, n_t can be computed exactly by communicating $|A_{\pi(v_t)}|$ and $|B_{\pi(v_t)}|$ with

$O(\log s)$ bits. The centers of mass, c_{v_t} and $c_{\pi(v_t)}$ are given by the vectors

$$c_{v_t} = \frac{1}{f^i_{v_t,A} + f^i_{v_t,B}} \sum_{\substack{\vec{x}\in\{0,1\}^d \\ v_i(\vec{x})=v_t}} \vec{x}\left((f_{A,B})_x + (f_{A,B})_{x+2^d}\right)$$

$$c_{\pi(v_t)} = \frac{1}{n_t} \sum_{\substack{\vec{x}\in\{0,1\}^d \\ v_{i-1}(\vec{x})=\pi(v_t)}} \vec{x}\left((f_{A,B})_x + (f_{A,B})_{x+2^d}\right).$$

Therefore, $\boldsymbol{\chi}^i_t$ is an L_1-sketch of $(f^i_{v_t,A} + f^i_{v_T,B})c_{v_t}$ and $\boldsymbol{\chi}^{i-1}_t$ an L_1-sketch of $n_t c_{\pi(v_t)}$ such that $\|c_{\pi(v_t)} - c_{v_t}\|_1$ can be approximated, and $\hat{\boldsymbol{q}}_t$ becomes an approximation of $\boldsymbol{q}_{\pi(v_t),v_t}$. This last point is captured by the following claim.

Claim 8.5.4 (Round 2 Claim)**.** *Let $v_t \in \mathcal{V}_i$ be any sample returned by Line 2 in Round 1, and suppose $v_t \neq \perp$. Let $\boldsymbol{\mathcal{E}}_{2,t}$ be the event, defined over the randomness of the sample $\mathbf{C}_t \sim \mathcal{S}^1_k(d, O(\log\log s))$ in Line 3 of Round 0 that in Line 3 of Round 2,*

$$\|\mathbf{c}_{v^*} - \mathbf{c}_{u^*}\|_1 \leq \mathrm{Alg}^1\left(\frac{\boldsymbol{\chi}^{i-1}_t}{\boldsymbol{n}_t} - \frac{\boldsymbol{\chi}^i_t}{f^i_{v_t,a} + f^i_{v_t,b}}\right) \leq 2\|\mathbf{c}_{v^*} - \mathbf{c}_{u^*}\|_1.$$

Then, $\boldsymbol{\mathcal{E}}_{2,t}$ occurs with probability at least $1 - 1/(100k)$.

Proof. We apply Theorem 100 with $m = d$, $t = O(\log\log s)$, and notice by definition

$$\frac{\boldsymbol{\chi}^{i-1}_t}{n_t} = \frac{\sum_{\substack{\vec{x}\in\{0,1\}^d \\ v_{i-1}(x)=\pi(v_t)}} (\mathbf{C}_t\vec{x}) \cdot \left((f_{A,B})_x + (f_{A,B})_{x+2^d}\right)}{|A_{\pi(v_t)}| + |B_{\pi(v_t)}|} = \frac{\sum_{x\in A_{\pi(v_t)}\cup B_{\pi(v_t)}} \mathbf{C}_t x}{|A_{\pi(v_t)}| + |B_{\pi(v_t)}|} = \mathbf{C}_t \cdot c_{\pi(v_t)}$$

similarly

$$\frac{\boldsymbol{\chi}^i_t}{f^i_{v_t,a} + f^i_{v_t,b}} = \frac{\sum_{\substack{\vec{x}\in\{0,1\}^d \\ v_i(x)=v_t}} (\mathbf{C}_t\vec{x}) \cdot \left((f_{A,B})_x + (f_{A,B})_{x+2^d}\right)}{|A_{v_t}| + |B_{v_t}|} = \frac{\sum_{x\in A_{v_t}\cup B_{v_t}} \mathbf{C}_t x}{|A_{v_t}| + |B_{v_t}|} = \mathbf{C}_t \cdot c_{v_t}$$

thus, by linearity

$$\frac{\boldsymbol{\chi}^{i-1}_t}{n_t} - \frac{\boldsymbol{\chi}^i_t}{f^i_{v_t,a} + f^i_{v_t,b}} = \mathbf{C}_t(c_{v_t} - c_{\pi(v_t)})$$

where $\mathbf{C}_t \sim \mathcal{S}^1_k(d, O(\log\log s))$, and noting that $k = O(\log s)$ so the failure probability of Theorem 100 is $1/\mathrm{poly}(k)$. □

Proof of Theorem 99. As mentioned in the paragraph subsequent to stating Theorem 99, we

Two-round Sketch $(f_{A,B}, T, i)$

Input: Vector $f_{A,B} \in \mathbb{R}^{2^{d+1}}$ encoding two multi-sets $A, B \subset \{0,1\}^d$, a compressed quadtree T, and index $i \in \{0, \ldots, h-1\}$.
Output: A real number $\hat{\boldsymbol{\mathcal{I}}}_i$.

- **Round 0:** Perform the universe reduction step (Proposition 8.5.1) so that $|\mathcal{V}_{i-1}| = s', |\mathcal{V}_i| = (s')^2$, where $s' = O(s^3)$. Set $k = O(\log^2 s)$.
  1. Define the linear functions $f^i \in \mathbb{R}^{|\mathcal{V}_i| \times \{A,B\}}$ and $F^i \in \mathbb{R}^{|\mathcal{V}_i|}$ of $f_{A,B}$ via

$$f^i_{v,A} = \sum_{\substack{\vec{x} \in \{0,1\}^d \\ v_i(\vec{x}) = v}} (f_{A,B})_x = |A_v| \quad \text{and} \quad f^i_{v,B} = \sum_{\substack{\vec{x} \in \{0,1\}^d \\ v_i(\vec{x}) = v}} (f_{A,B})_{x+2^d} = |B_v|$$

$$F^i_v = f^i_{v,A} - f^i_{v,B} = |A_v| - |B_v|$$

  2. For each $t \in [k]$, initialize linear sketch $(\mathbf{S}_t, \mathrm{Alg}^1_{\mathbf{S}_t}) \sim \mathcal{S}^1_a((s')^2, O(\log^2 s))$.
  3. For each $t \in [k]$, initialize linear sketch $\mathbf{C}_t \sim \mathcal{S}^1_k(d, O(\log\log s))$, and initialize $\mathbf{C} \sim \mathcal{S}^1_k((s')^2, O(1))$.
- **Round 1:**
  1. Construct the linear sketch $\boldsymbol{\beta} \stackrel{\text{def}}{=} \mathbf{C}F^i$, and for each $t \in [k]$, construct the linear sketches $\boldsymbol{\alpha}_t \stackrel{\text{def}}{=} \mathbf{S}_t F^i$.
  2. Generate k samples $v_1, \ldots, v_k \in \mathcal{V}_i \cup \{\perp\}$ obtained from

$$v_t \stackrel{\text{def}}{=} \mathrm{Alg}^1_{\mathbf{S}_t}(\boldsymbol{\alpha}_t), \qquad \text{and let} \qquad \hat{\boldsymbol{\beta}} \stackrel{\text{def}}{=} \mathrm{Alg}^1(\boldsymbol{\beta}).$$

- **Round 2:** Let $L \subset [k]$ be the set of indices such that $v_t \neq \perp$. For each $t \in L$:
  1. Compute the value $n_t \stackrel{\text{def}}{=} \sum_{\substack{v \in \mathcal{V}_i \\ \pi(v) = \pi(v_t)}} f^i_{v,A} + f^i_{v,B} = |A_{\pi(v_t)}| + |B_{\pi(v_t)}|$.
  2. Compute the vectors $\boldsymbol{\chi}^i_t \stackrel{\text{def}}{=} \sum_{\substack{\vec{x} \in \{0,1\}^d \\ v_i(\vec{x}) = v_t}} (\mathbf{C}_t \vec{x}) \cdot ((f_{A,B})_x + (f_{A,B})_{x+2^d})$ and $\boldsymbol{\chi}^{i-1}_t \stackrel{\text{def}}{=} \sum_{\substack{\vec{x} \in \{0,1\}^d \\ v_{i-1}(\vec{x}) = \pi(v_t)}} (\mathbf{C}_t \vec{x}) \cdot ((f_{A,B})_x + (f_{A,B})_{x+2^d})$.
  3. Let

$$\hat{\boldsymbol{q}}_t \stackrel{\text{def}}{=} \max\left\{\min\left\{\mathrm{Alg}^1\left(\frac{\boldsymbol{\chi}^{i-1}_t}{n_t} - \frac{\boldsymbol{\chi}^i_t}{f^i_{v_t,A} + f^i_{v_t,B}}\right), \frac{30d\log s}{2^i}\right\}, \frac{d}{2^i}\right\},$$

  if all $\boldsymbol{n}_t$ and $f^i_{v_t,A} + f^i_{v_t,B}$ are non-zero. Output

$$\hat{\boldsymbol{\mathcal{I}}}_i \stackrel{\text{def}}{=} \hat{\boldsymbol{\beta}} \cdot \frac{1}{|L|} \sum_{t \in L} \hat{\boldsymbol{q}}_t.$$

Figure 8.4: The Two-round Protocol.

show that for any compressed quadtree T and any $i \in \{0, \ldots, h-1\}$, with probability at least $3/4$ over the execution of the sketching algorithm, the output $\hat{\boldsymbol{\mathcal{I}}}_i$ satisfies $\boldsymbol{\mathcal{I}}_i/8 \le \hat{\boldsymbol{\mathcal{I}}}_i \le 8\boldsymbol{\mathcal{I}}_i$. Repeating the protocol for $O(\log \log d)$ iterations and outputting the median (in order to decrease the error probability), and repeating it for all $i \in \{0, \ldots, h-1\}$ obtains the desired theorem.

For the rest of the proof, consider a fixed $i \in \{0, \ldots, h-1\}$ and T. We consider the distribution ν supported on $\mathcal{V}_i$ where

$$\mathop{\mathbf{Pr}}_{\mathbf{v}\sim\nu}[\mathbf{v} = v^*] \propto \Big||A_{v^*}| - |B_{v^*}|\Big|, \qquad \text{and} \qquad Z \stackrel{\text{def}}{=} \sum_{v\in\mathcal{V}_i} \Big||A_v| - |B_v|\Big|$$

is the normalizing constant. We have

$$\boldsymbol{\mathcal{I}}_i = Z \cdot \mathop{\mathbf{E}}_{\mathbf{v}\sim\nu}\left[\boldsymbol{q}_{\mathbf{v},\pi(\mathbf{v})}\right] \ge Z \cdot \frac{d}{2^i}.$$

To complete the proof, we show the following:

1. The random variable $\hat{\boldsymbol{\beta}}$ satisfies $Z \le \hat{\boldsymbol{\beta}} \le 2Z$, with probability at least 0.99 over the randomness in $\mathbf{C} \sim \mathcal{S}_k^1((s')^3, O(1))$. This follows immediately from Claim 8.5.2.

2. For every $t \in L$, consider the sample $v_t \in \mathcal{V}_i$ in Line 2 of Round 1. Then, with probability at least $1 - 1/(100k)$ over the draw of $\mathbf{C}_t \sim \mathcal{S}_k^1(d, O(\log \log s))$,

$$\frac{d}{2^i} \le \boldsymbol{q}_{v_j,u_j} \le \hat{\boldsymbol{q}}_t \le 2\boldsymbol{q}_{v_j,u_j} \le \frac{60d \log s}{2^i},$$

   which follows from Claim 8.5.4.

3. We have $|L| > k/2$ with probability $1 - 1/s$. This follows from a Chernoff bound on the number of v_t such that $v_t = \perp$.

4. The random variables v_i for $i \in [L]$ in Line 2 are independent and identically distributed, depending on the draw of $(\mathbf{S}_t, \mathrm{Alg}_{\mathbf{S}_t}^{(L_1\text{-sa})}) \sim \mathcal{S}_a^1((s')^2, O(\log^2 s))$. Specifically, they are drawn from a distribution $\mathcal{D}'$ over $\mathcal{V}_i$, such that we have $d_{\mathrm{TV}}(\mathcal{D}', \nu) \le 1/s$. This follows from Claim 8.5.3.

From 1 and 4 above, we have that with probability at least 0.99,

$$\boldsymbol{\mathcal{I}}_i/2 \le \boldsymbol{\mathcal{I}}_i - O\left(\frac{Z \cdot d \log s}{s2^i}\right) \le \mathop{\mathbf{E}}_{\mathbf{v}\sim\mathcal{D}'}\left[\hat{\boldsymbol{\beta}} \cdot \boldsymbol{q}_{\mathbf{v},\pi(\mathbf{v})}\right] \le \boldsymbol{\mathcal{I}}_i + O\left(\frac{Z \cdot d \log s}{s2^i}\right) \le 2\boldsymbol{\mathcal{I}}_i$$

and by 2, with probability $0.99 \cdot 0.99$ over the draws of $\mathbf{C} \sim \mathcal{S}_k^1((s')^2, O(1))$ and all $\mathbf{C}_1, \ldots, \mathbf{C}_k \sim \mathcal{S}_k^1(d, O(\log\log s))$,

$$\frac{1}{4} \cdot \boldsymbol{\mathcal{I}}_i \leq \mathop{\mathbf{E}}_{\mathbf{v} \sim \mathcal{D}'} \left[\hat{\boldsymbol{\beta}} \cdot \hat{\boldsymbol{q}}_t\right] \leq 4\boldsymbol{\mathcal{I}}_i$$

Thus, the output $\hat{\boldsymbol{q}}$ of the sketching algorithm has expectation between $\boldsymbol{\mathcal{I}}_i/4$ and $4\boldsymbol{\mathcal{I}}_i$, and by the boundedness of $\hat{\boldsymbol{q}}_t$ and independence across $t \in [k]$,

$$\mathbf{Var}\left[\frac{\hat{\boldsymbol{\beta}}}{k}\sum_{t=1}^{k} \hat{\boldsymbol{q}}_t\right] = O\left(\frac{Z^2 d^2 \log^2 s}{k \cdot 2^{2i}}\right)$$

which is less than $c\boldsymbol{\mathcal{I}}_i^2$ for arbitrarily small constant $c > 0$, given large enough $k = O(\log^2 s)$. As a result, we apply Chebyshev's inequality to conclude that the output $\hat{\boldsymbol{q}}$ is between $\boldsymbol{\mathcal{I}}_i/8$ and $8\boldsymbol{\mathcal{I}}_i$ with probability at least $0.99 \cdot 0.99 \cdot 0.99 > 3/4$ as needed, and we conclude the theorem. □

8.6 One-Round Linear Sketch

In this section, we demonstrate how the two-round sketch of Section 8.5 can be compressed into a *single* round, albeit with the addition of a small additive error. Recall that a one-round linear sketch first implicitly generates a random matrix $\mathbf{S} \in \mathbb{R}^{k \times 2^{d+1}}$, and stores only the matrix vector product $\mathbf{S}f$, where $f \in \mathbb{R}^{2^{d+1}}$ is the vectorized representation of the two multi-sets $A, B \subset \{0,1\}^d$. The space used by a linear sketch is the number of bits required to store $\mathbf{S}f$. In our setting, since $\mathbf{S}$ will have small bit complexity and f is $2s$ sparse, the space complexity will be a $\log s$ factor larger than the number of rows of $\mathbf{S}$. Specfically, the goal of this section is to prove the following Theorem.

Theorem 102. *For $d, s \in \mathbb{N}$, there exists a 1-round linear sketching algorithm such that given multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$, computes an approximate $\hat{\boldsymbol{\mathcal{I}}}$ to $\mathrm{EMD}(A, B)$ with*

$$\mathrm{EMD}(A, B) \leq \hat{\boldsymbol{\mathcal{I}}} \leq \tilde{O}(\log s)\,\mathrm{EMD}(A, B) + \epsilon d s$$

with probability at least $3/4$. Moreover, the space used by the linear sketch is $O(1/\epsilon) \cdot \mathrm{polylog}(s, d)$.

Rescaling ϵ by a factor of d yields a additive ϵs approximation in $O(d/\epsilon) \cdot \mathrm{polylog}(s, d)$ space. Whenever, the size-s multi-sets $A, B \subset \{0,1\}^d$ do not intersect drastically, i.e., $|A \cap B|/|A \cup B| < 1 - \epsilon$ where $\cup, \cap$ are multi-set unions and intersections, then $\mathrm{EMD}(A, B) > \epsilon s$. This instances, also known as those having Jaccard index bounded away from 1, have been studied for EMD in

low-dimensions [YO14], and in this case, we obtain the following corollary.

Corollary 8.6.1. *For $d, s \in \mathbb{N}$ and $\epsilon \in (0,1)$, there exists a 1-round linear sketching algorithm such that given multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$ satisfying $|A \cap B|/|A \cup B| \leq 1-\epsilon$, computes an approximate $\hat{\boldsymbol{\mathcal{I}}}$ to $\mathrm{EMD}(A,B)$ with*

$$\mathrm{EMD}(A,B) \leq \hat{\boldsymbol{\mathcal{I}}} \leq \tilde{O}(\log s)\,\mathrm{EMD}(A,B)$$

with probability at least $3/4$. Moreover, the space used by the linear sketch is $O(d/\epsilon)\cdot\mathrm{polylog}(s,d)$.

The goal is similar to the two-round protocol. We begin by sampling a compressed quadtree $\mathbf{T}$ if depth $h = \log_2(2d)$, as well as a universe reduction step of Proposition 8.5.1 in Section 8.5, using public randomness. For each depth $i \in \{0,\ldots,h-1\}$, we estimate the quantity

$$\boldsymbol{\mathcal{I}}_i \stackrel{\text{def}}{=} \sum_{v \in \mathcal{V}_i} \Big||A_v| - |B_v|\Big| \cdot \boldsymbol{q}_{\pi(v),v}.$$

To estimate $\boldsymbol{\mathcal{I}}_i$, we first sample a vertex $v \sim \mathcal{V}_i$ with probability proportional to $\Big||A_v| - |B_v|\Big|$. In addition, we obtain an estimate $\hat{\Delta}_i$ of $\Delta_i = \sum_{v \in \mathcal{V}_i} \Big||A_v| - |B_v|\Big|$. Once we have such a sample v, we approximate compute the cost $\boldsymbol{q}_{\pi(v),v}$ via a value $\hat{\boldsymbol{q}}_{\pi(v),v}$, and output $\hat{\Delta}_i \cdot \hat{\boldsymbol{q}}_{\pi(v),v}$ as an approximation of $\boldsymbol{\mathcal{I}}_i$. Repeating the process $\mathrm{poly}(\log s)$ times, we will obtain our desired estimate. While in Section 8.5, we could use one round to produce the sample v, and another round to estimate $\hat{\boldsymbol{q}}_{\pi(v),v}$, the main challenge here is to procedure the sample and estimate *simultaneously*. The approach is based on a new technique we call *precision sampling with meta-data*, building on the precision sampling framework from Chapter 3 and prior works [AKO11, JST11].

8.6.1 Precision Sampling Lemma

The goal of this section is to prove a useful Precision Sampling Lemma, which results in a significantly simplified sampling algorithm than the one described in Chapter 3 and prior works [AKO11, JST11]. Specifically, the simplified reporting algorithm always returns the maximum coordinate from the count-sketch estimate of the scaled stream vector; in other words, it has no failure procedure (it never outputs $\perp$, in the language of Definition 3.1.2). This Lemma does not supercede prior work, since from the perspective of typical streaming algorithms, the complexity of the sampler will be a $\log n$ factor worse than the approximate L_p samplers of [AKO11, JST11], and will not be a perfect sampler like the one from Chapter 3.

We begin by recalling several useful properties of the order statistics of n independent non-identically distributed exponential random variables (see Section 3.2.2 for a further discussion). Let $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$ be independent exponential random variables where $\boldsymbol{t}_i$ has mean $1/\lambda_i$ (equivalently, $\boldsymbol{t}_i$ has rate λ_i), abbreviated as $Exp(\lambda)$. Recall that $\boldsymbol{t}_i$ is given by the cumulative distribution function $\mathbf{Pr}[\boldsymbol{t}_i < x] = 1 - e^{-\lambda_i x}$. Our algorithm will require an analysis of the distribution of values $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$, which we will now describe. Recall earlier from Fact 3.2.3 that constant factor scalings of an exponential variable result in another exponential variable.

Fact 3.2.3 (Scaling of exponentials). *Let $\boldsymbol{t}$ be exponentially distributed with rate λ, and let $\alpha > 0$. Then $\alpha\boldsymbol{t}$ is exponentially distributed with rate λ/α.*

Definition 8.6.2. *Let $\boldsymbol{t} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$ be independent exponentials. For $k = 1, 2, \ldots, n$, we define the k-th anti-rank $D_{\boldsymbol{t}}(k) \in [n]$ of $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$ to be the values $D_{\boldsymbol{t}}(k)$ such that $\boldsymbol{t}_{D_{\boldsymbol{t}}(1)} \leq \boldsymbol{t}_{D_{\boldsymbol{t}}(2)} \leq \cdots \leq \boldsymbol{t}_{D_{\boldsymbol{t}}(n)}$.*

Recall that using the structure of the anti-rank vector, one can derive a simple formula fo the distribution of $\boldsymbol{t}_{D_{\boldsymbol{t}}(k)}$ as a function of $(\lambda_1, \ldots, \lambda_n)$ and the anti-rank vector. This formula is presented in Fact 3.2.5, and crucially utilized in Chapter 3 to bound the dependency of failure event (outputting $\perp$) of the sampler on the identity of $D(1)$. For this chapter, we will only need to apply the formula to the first two order statistics $D(1), D(2)$, thus we present, with proof, a reformulation of Fact 3.2.5 here, which will be more directly applicable for our purposes.

Fact 8.6.3 (Reformulation of Fact 3.2.5, derived by [Nag06])**.** *Let $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$ be independently distributed exponentials, where each $\boldsymbol{t}_i$ has rate $\lambda_i > 0$. Then, $D_{\boldsymbol{t}}(1)$ is i with probability $\lambda_i / \sum_{j \in [n]} \lambda_j$. Furthermore, the following two sampling procedures produce the same distribution over pairs in $\mathbb{R}^2$:*

1. *We sample $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$, where $\boldsymbol{t}_i \sim Exp(\lambda_i)$, and output $(\boldsymbol{t}_{D_{\boldsymbol{t}}(1)}, \boldsymbol{t}_{D_{\boldsymbol{t}}(2)} - \boldsymbol{t}_{D_{\boldsymbol{t}}(1)})$.*
2. *We sample $\boldsymbol{i}_1 \in [n]$ where $\Pr[\boldsymbol{i}_1 = i] = \lambda_i / \sum_{j=1}^{n} \lambda_i$, and independently sample $\boldsymbol{E}_1, \boldsymbol{E}_2 \sim Exp(1)$. We output*
$$\left(\frac{\boldsymbol{E}_1}{\sum_{j \in [n]} \lambda_j}, \frac{\boldsymbol{E}_2}{\sum_{j \in [n] \setminus \{i_1\}} \lambda_j} \right).$$

Proof. This is a simple computation. We have that for any $r, r' \in \mathbb{R}_{\geq 0}$ and $i \in [n]$,

$$\begin{aligned}
\Pr_{\boldsymbol{t}}\left[\begin{array}{c} D_{\boldsymbol{t}}(1) = i, \\ \boldsymbol{t}_{D_{\boldsymbol{t}}(1)} \geq r, \\ \boldsymbol{t}_{D_{\boldsymbol{t}}(2)} - \boldsymbol{t}_{D_{\boldsymbol{t}}(1)} \geq r' \end{array}\right] &= \int_{y:r}^{\infty} \lambda_i \exp\left(-\lambda_i y\right) \prod_{j \in [n]\setminus\{i\}} \Pr_{\boldsymbol{t}_j \sim Exp(\lambda_j)}\left[\boldsymbol{t}_j - y \geq r'\right] dy \\
&= \lambda_i \exp\left(-r' \sum_{j\in[n]\setminus\{i\}} \lambda_j\right) \int_{y:r}^{\infty} \exp\left(-y \sum_{j=1}^{n} \lambda_j\right) dy \\
&= \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_i} \cdot \exp\left(-r' \sum_{j\in[n]\setminus\{i\}} \lambda_j\right) \cdot \exp\left(-r \sum_{j=1}^{n} \lambda_i\right) \\
&= \mathbf{Pr}\left[\boldsymbol{i}_1 = i \wedge \frac{\boldsymbol{E}_1}{\sum_{j\in[n]} \lambda_j} \geq r \wedge \frac{\boldsymbol{E}_2}{\sum_{j\in[n]\setminus\{\boldsymbol{i}_1\}} \lambda_j} \geq r'\right].
\end{aligned}$$

□

We are now ready to present the simplified precision sampling lemma which we will centrally apply in this section.

Lemma 8.6.4. *Let $x \in \mathbb{R}^n$ be any vector, and let $\boldsymbol{z} \in \mathbb{R}^n$ be the random vector given by letting*

$$\boldsymbol{z}_i = \frac{x_i}{\boldsymbol{t}_i}, \qquad \textit{where} \qquad \boldsymbol{t}_i \sim Exp(1).$$

Let $\epsilon \in (0,1)$, and suppose $\tilde{\boldsymbol{z}}$ is an adversarially corrupted vector satisfying $\|\tilde{\boldsymbol{z}} - \boldsymbol{z}\|_\infty \leq \epsilon \|x\|_1$, and define the random variable $\boldsymbol{i}^* = \arg\max_i |\tilde{\boldsymbol{z}}_i|$. Then we have for all $i \in [n]$:

$$\Pr_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n \sim Exp(1)}\left[\boldsymbol{i}^* = i\right] = (1 \pm O(\epsilon)) \frac{|x_i|}{\|x\|_1} \pm O(e^{-\frac{1}{4\epsilon}}) \tag{8.38}$$

Proof. Consider the random variable $\boldsymbol{t}' = (\boldsymbol{t}'_1, \ldots, \boldsymbol{t}'_n)$ where $\boldsymbol{t}'_i \sim Exp(|x_i|)$, and notice, by Fact 3.2.3, that $\boldsymbol{t}'_i$ is distributed as $1/|\boldsymbol{z}_i|$. Hence, $\boldsymbol{i}^*$ is set to i whenever $D_{\boldsymbol{t}'}(1) = i$ and $1/\boldsymbol{t}'_{D_{\boldsymbol{t}'}(1)} - 1/\boldsymbol{t}'_{D_{\boldsymbol{t}'}(2)} \geq 2\epsilon\|x\|_1$. Thus, by Fact 8.6.3,

$$\begin{aligned}
\Pr_{\substack{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n \\ \sim Exp(1)}}\left[\boldsymbol{i}^* = i\right] &\geq \Pr_{\substack{\boldsymbol{t}'=(\boldsymbol{t}'_1,\ldots,\boldsymbol{t}'_n) \\ \boldsymbol{t}'_i \sim Exp(|x_i|)}}\left[D_{\boldsymbol{t}'}(1) = i \wedge \frac{1}{\boldsymbol{t}'_{D_{\boldsymbol{t}'}(1)}} - \frac{1}{\boldsymbol{t}'_{D_{\boldsymbol{t}'}(2)}} \geq 2\epsilon\|x\|_1\right] \\
&\geq \Pr_{\boldsymbol{i}_1, \boldsymbol{E}_1, \boldsymbol{E}_2}\left[\boldsymbol{i}_1 = i \wedge \frac{1}{\boldsymbol{E}_1} - \frac{1}{\boldsymbol{E}_1 + \boldsymbol{E}_2} \geq 2\epsilon\right] \\
&= \frac{|x_i|}{\|x\|_1}\left(1 - \int_{r:0}^{1/(2\epsilon)} \exp\left(-r\right) \cdot \Pr_{\boldsymbol{E}_2}\left[\boldsymbol{E}_2 \leq \frac{2\epsilon r^2}{1 - 2\epsilon r}\right] dr - \exp\left(-\frac{1}{2\epsilon}\right)\right),
\end{aligned}$$

and we have

$$\int_{r:0}^{1/(2\epsilon)} \exp(-r) \mathop{\mathbf{Pr}}_{\boldsymbol{E}_2}\left[\boldsymbol{E}_2 \le \frac{2\epsilon r^2}{1-2\epsilon r}\right] dr \le \int_{r:0}^{1/(4\epsilon)} 4\epsilon r^2 \exp(-r) dr + \frac{1}{4\epsilon}\exp(-1/(4\epsilon)) \lesssim \epsilon,$$

which results in the lower bound in (8.38). In order to upper bound the probability we pick $\boldsymbol{i}^* = i$, we notice that

$$\mathop{\mathbf{Pr}}_{\substack{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n \\ \sim Exp(1)}}[\boldsymbol{i}^* = i] \le \frac{|x_i|}{\|x\|_1} + \mathop{\mathbf{Pr}}_{\substack{\boldsymbol{t}'=(\boldsymbol{t}'_1,\ldots,\boldsymbol{t}'_n) \\ \boldsymbol{t}'_i \sim Exp(|x_i|)}}\left[D_{\boldsymbol{t}'}(1) \neq i \wedge \frac{1}{\boldsymbol{t}'_{D_{\boldsymbol{t}'}(1)}} - \frac{1}{\boldsymbol{t}'_i} \le 2\epsilon\|x\|_1\right],$$

and

$$\begin{aligned}
&\mathop{\mathbf{Pr}}_{\substack{\boldsymbol{t}'=(\boldsymbol{t}'_1,\ldots,\boldsymbol{t}'_n) \\ \boldsymbol{t}'_i \sim Exp(|x_i|)}}\left[D_{\boldsymbol{t}'}(1) \neq i \wedge \frac{1}{\boldsymbol{t}'_{D_{\boldsymbol{t}'}(1)}} - \frac{1}{\boldsymbol{t}'_i} \le 2\epsilon\|x\|_1\right] \\
&\quad \le \sum_{j\in[n]\setminus\{i\}} \int_{r:0}^{\frac{1}{2\epsilon\|x\|_1}} |x_j| \exp\left(-r\sum_{j'\in[n]\setminus\{i\}}|x_{j'}|\right) \mathop{\mathbf{Pr}}_{\boldsymbol{t}'_i\sim Exp(|x_i|)}\left[\frac{r}{1-2\epsilon\|x\|_1 r} \ge \boldsymbol{t}'_i \ge r\right] dr + \exp\left(-\frac{1}{2\epsilon}\right),
\end{aligned}$$

where the first term bounds the probability that $\frac{1}{\boldsymbol{t}'_i}$ is not the maximum value, but is large enough to be corrupted in $\tilde{\boldsymbol{z}}$ to appear as the maximum, and the second term bounds the probability that the maximum value is less than $2\epsilon\|x\|_1$ (in which case a corruption index i may make it the maximum). Then, we have

$$\begin{aligned}
&\sum_{j\in[n]\setminus\{i\}} \int_{r:0}^{\frac{1}{2\epsilon\|x\|_1}} |x_j| \exp\left(-r\sum_{j'\in[n]\setminus\{i\}}|x_{j'}|\right) \mathop{\mathbf{Pr}}_{\boldsymbol{t}'_i\sim Exp(|x_i|)}\left[\frac{r}{1-2\epsilon\|x\|_1 r} \ge \boldsymbol{t}'_i \ge r\right] dr \\
&\quad \le \sum_{j\in[n]\setminus\{i\}} \int_{r:0}^{\frac{1}{4\epsilon\|x\|_1}} |x_j| \exp\left(-r\|x\|_1\right)\left(1 - \exp\left(-\frac{2\epsilon\|x\|_1|x_i|r^2}{1-2\epsilon\|x\|_1 r}\right)\right) dr + \exp\left(-\frac{1}{4\epsilon}\right) \\
&\quad \lesssim \epsilon\|x\|_1 \cdot |x_i| \sum_{j\in[n]\setminus\{i\}} |x_j| \int_{r:0}^{\frac{1}{4\epsilon\|x\|_1}} r^2 \exp\left(-r\|x\|_1\right) dr + \exp\left(-\frac{1}{4\epsilon}\right) \lesssim \frac{\epsilon|x_i|}{\|x\|_1} + \exp\left(-\frac{1}{4\epsilon}\right),
\end{aligned}$$

where in the second inequality, we have $1 - \exp\left(-\frac{2\epsilon\|x\|_1|x_i|r^2}{1-2\epsilon\|x\|_1 r}\right) \lesssim \epsilon\|x\|_1|x_i|r^2$ when $r \le 1/(4\epsilon\|x\|_1)$. □

We will also need the following Lemma, which generalizes Proposition 3.3.5 from Chapter 3 to scalings beyond exponential random variables.

Lemma 8.6.5 (Generalization of Proposition 3.3.5, Chapter 3)**.** *Fix $n \in \mathbb{N}$ and $\alpha \ge 0$, and let*

$\{\mathcal{D}_i\}_{i\in[n]}$ be a collection of distributions over $\mathbb{R}$ satisfying

$$\Pr_{\boldsymbol{y}\sim\mathcal{D}_i}[|\boldsymbol{y}| \geq t] \leq \frac{\alpha}{t} \qquad \text{for all } i \in [n] \text{ and all } t \geq 1.$$

For any fixed vector $x \in \mathbb{R}^n$ and integer $\beta \geq 1$, consider the random vector $\boldsymbol{z} \in \mathbb{R}^n$ given by letting

$$\boldsymbol{z}_i \stackrel{\text{def}}{=} \boldsymbol{t}_i \cdot x_i, \qquad \text{where } \boldsymbol{t}_i \sim \mathcal{D}_i \text{ independently for all } i \in [n].$$

Then, $\|\boldsymbol{z}_{-2\beta}\|_2 \leq 12\alpha\|x_{-\beta}\|_1/\sqrt{\beta}$ and $\|\boldsymbol{z}_{-\beta}\|_1 \leq 9\alpha\|x\|_1\lceil\log_2 n\rceil$ with probability at least $1 - 3e^{-\beta/4}$ over the draws of $\boldsymbol{t}_i \sim \mathcal{D}_i$.[22]

Proof. Define the random sets

$$\mathbf{I}_k = \left\{ i \in [n] : \frac{\alpha\|x\|_1}{2^{k+1}} \leq |\boldsymbol{z}_i| \leq \frac{\alpha\|x\|_1}{2^k} \right\} \qquad \text{for } k = 0, 1, 2, \ldots, \lceil\log_2 n\rceil,$$

and notice that, for any $i \in [n]$,

$$\Pr_{\boldsymbol{t}_i\sim\mathcal{D}_i}[i \in \mathbf{I}_k] \leq \Pr_{\boldsymbol{t}_i\sim\mathcal{D}_i}\left[|\boldsymbol{t}_i| \geq \frac{\alpha\|x\|_1}{2^{k+1}|x_i|}\right] \leq \frac{2^{k+1}|x_i|}{\|x\|_1} \qquad \text{implying} \qquad \mathop{\mathbf{E}}_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n}[|\mathbf{I}_k|] \stackrel{\text{def}}{=} \mu_k \leq 2^{k+1}.$$

Let $\boldsymbol{\mathcal{E}}_1$ denote the event that there exists $k \geq \lceil\log_2(\beta/4)\rceil$ such that $|\mathbf{I}_k| > 4 \cdot 2^{k+1}$. Since all draws $\boldsymbol{t}_i \sim \mathcal{D}_i$ are independent, we let $\delta_k = \frac{4\cdot 2^{k+1}}{\mu} - 1 \geq 3$,

$$\begin{aligned}\Pr_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n}[\boldsymbol{\mathcal{E}}_1] \leq \sum_{k=\lceil\log_2(\beta/4)\rceil}^{\lceil\log_2 n\rceil} \Pr_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n}\left[|\mathbf{I}_k| > 4\cdot 2^{k+1}\right] &= \sum_{k=\lceil\log_2(\beta/4)\rceil}^{\lceil\log_2 n\rceil} \Pr_{\boldsymbol{t}_1,\ldots,\boldsymbol{t}_n}[|\mathbf{I}_k| > (1+\delta_k)\mu_k] \\ &\leq \sum_{k=\lceil\log_2(\beta/4)\rceil}^{\lceil\log_2 n\rceil} \exp\left(-2^k\right) \leq 2\exp(-\beta/4),\end{aligned}$$

by a Chernoff bound. Similarly, every $i \in [n]$ satisfies $|\boldsymbol{z}_i| \geq 4\alpha\|x\|_1/\beta$ with probability at most $\beta|x_i|/(4\|x\|_1)$ over the draw of $\boldsymbol{t}_i \sim \mathcal{D}_i$. The event $\boldsymbol{\mathcal{E}}_2$ that more than β indices $i \in [n]$ satisfy $|\boldsymbol{z}_i| \geq 4\alpha\|x\|_1/\beta$ occurs with probability at most $\exp(-\beta/4)$. Thus, whenever $\boldsymbol{\mathcal{E}}_1$ and $\boldsymbol{\mathcal{E}}_2$ do not

[22] For any vector $x \in \mathbb{R}^n$ and any integer $\beta \geq 1$, recall that we define $x_{-\beta} \in \mathbb{R}^n$ be the vector given by x where the β highest magnitude coordinates are set to 0. When $\beta \geq 1$ is not an integer, $x_{-\beta}$ is interpreted as $x_{-\lfloor\beta\rfloor}$.

occur, (which happens with probability at least $1 - 3e^{-\beta/4}$), we have

$$\|\boldsymbol{z}_{-\beta}\|_2^2 \leq \sum_{k=\lceil \log_2(\beta/4)\rceil}^{\lceil \log_2 n\rceil} |\mathbf{I}_k| \cdot \frac{\alpha^2\|x\|_1^2}{2^{2k}} + n \cdot \frac{\alpha^2\|x\|_1^2}{n^2} \leq \frac{128\alpha^2\|x\|_1^2}{\beta},$$

$$\|\boldsymbol{z}_{-\beta}\|_1 \leq \sum_{k=\lceil \log_2(\beta/4)\rceil}^{\lceil \log_2 n\rceil} |\mathbf{I}_k| \cdot \frac{\alpha\|x\|_1}{2^{k}} + n \cdot \frac{\alpha\|x\|_1}{n} \leq \alpha\|x\|_1\left(8\lceil \log_2 n\rceil + 1\right)$$

since once $\beta \geq n$, the bound becomes 0. Hence, it follows that $\|\boldsymbol{z}_{-\beta}\|_2 \leq 12\alpha\|x\|_1/\sqrt{\beta}$ with probability at least $1 - 3e^{-\beta/4}$. Consider applying the above argument with the vector $x_{-\beta}$ with $\boldsymbol{z}'_i = (x_{-\beta})_i \cdot \boldsymbol{t}_i$ to bound $\|\boldsymbol{z}'_{-\beta}\|_2 \leq 12\alpha\|x_{-\beta}\|_1/\sqrt{\beta}$, and then note that $\|\boldsymbol{z}_{-2\beta}\|_2 \leq \|\boldsymbol{z}'_{-\beta}\|_2$. □

Remark 103. Using Lemmas 8.6.4 and 8.6.5, along with the standard Count-Sketch from Section 2.3.2, one obtains a $O(\log^3(n))$ bits of space algorithm for ℓ_1 sampling with relative error $(1/\log n)$ and additive error $1/\mathrm{poly}(n)$ which never outputs **FAIL.** This can be naturally extended to $p \in (0, 2]$, where the space increases by a $\log(n)$ factor for $p = 2$. While this is still much weaker than the perfect sampler from Chapter 3, it matches the complexity of the best known approximate sampler of [JST11] prior to the results of Chapter 3, which yields the same space and error guarantee. The advantage of the algorithm resulting from Lemmas 8.6.4 is that it is perhaps simpler to state and present, and may be useful for pedagogic purposes.

8.6.2 Additional Sketching Tools

In this section, we recall some classical tools from the sketching literature, and also develop some augmentations to them. We begin by restating the count-sketch guarantees from Section 2.3.2 in a notation which avoids defining the recovery procedure precisely.

Theorem 104 (Theorem 9)**.** *Fix any $\epsilon > 0$ and $n \in \mathbb{N}$, and let $k = O(\log n/\epsilon^2)$. There is a distribution $\mathcal{S}_k^\infty(n, \epsilon)$ supported on pairs $(\mathbf{S}, \mathrm{Alg}_{\mathbf{S}}^\infty)$, where $\mathbf{S} \in \mathbb{R}^{k\times n}$ is a matrix encoded with $O(k \log n)$ bits of space, and $\mathrm{Alg}_{\mathbf{S}}^\infty$ is an algorithm which receives a vector in $\mathbb{R}^k$ and outputs a vector in $\mathbb{R}^n$. For any $x \in \mathbb{R}^n$,*

$$\|x - \mathrm{Alg}_{\mathbf{S}}^\infty(\mathbf{S}x)\|_\infty \leq \epsilon\|x_{-1/\epsilon^2}\|_2$$

with probability at least $1 - 1/\mathrm{poly}(n)$ over the draw of $(\mathbf{S}, \mathrm{Alg}_{\mathbf{S}}^\infty) \sim \mathcal{S}_k^\infty(n, \epsilon)$.

For notational simplicity, notice that count-sketch may be applied to an $n \times m$ matrix $\mathbf{X}$, by

multiplying $\mathbf{SX}$ and applying $\mathrm{Alg}_{\mathbf{S}}$ to each column of $\mathbf{SX}$. If $m \leq \mathrm{poly}(n)$, we may apply a union bound over all m columns and obtain analogous point-wise estimates of all entries of $\mathbf{X}$. The error obtained for an entry $\mathbf{X}_{i,j}$ naturally depends on the (tail)-ℓ_2 norm of the j-th column of $\mathbf{X}$ as per the above theorem. Our algorithm will use the count-sketch from Section 2.3.2, but also a more general version will be needed.

Theorem 105 (Nested Count-Sketch)**.** *Let $n, m, k \in \mathbb{N}$, $\eta \in (0,1)$, and consider a partition $U = \{U_\ell\}_{\ell \in [m]}$ of $[n]$, $k \leq m$, as well as $s = O(k(\log n)^2/\eta^2)$. There exists a distribution $\mathcal{S}_k^{\infty,U}(n,\eta)$ supported on pairs $(\mathbf{S}, \mathrm{Alg}_{\mathbf{S}}^{\infty,U})$, where $\mathbf{S} \in \mathbb{R}^{s \times n}$ is a matrix encoded with $O(s \log n)$ bits of space, and $\mathrm{Alg}_{\mathbf{S}}^{\infty,U}$ is an algorithm which receives a vector in $\mathbb{R}^s$ and outputs a vector in $\mathbb{R}^n$. Fix any vector $x \in \mathbb{R}^n$, any subset $J \subset [m]$ of size at most k, and any $j \in J$, and let $y \in \mathbb{R}^n$ be the vector given by letting for $i \in [n]$,*

$$y_i = \begin{cases} 0 & i \in \bigcup_{\ell \in J \setminus \{j\}} U_\ell \\ x_i & \text{o.w.} \end{cases}.$$

Then, with probability at least $1 - 1/\mathrm{poly}(n)$ over the draw of $(\mathbf{S}, \mathrm{Alg}_{\mathbf{S}}^{\infty,U}) \sim \mathcal{S}_k^{\infty,U}(n,k,\eta)$, every $i \in U_j$ satisfies

$$\left| x_i - \mathrm{Alg}_{\mathbf{S}}^{\infty,U}(\mathbf{S}x)_i \right| \leq \eta \|y_{-1/\eta^2}\|_2.$$

Proof. Let $\boldsymbol{h} \colon [m] \to [10k]$ be a 4-wise independent hash function. For each $t \in [10k]$, we let $\boldsymbol{z}^{(t)} \in \mathbb{R}^n$ be the vector given by setting, for each $i \in [n]$

$$\boldsymbol{z}_i^{(t)} = \begin{cases} x_i & i \in U_j \text{ where } \boldsymbol{h}(j) = t \\ 0 & \text{o.w.} \end{cases}.$$

Instantiate a Count-Sketch data structure by sampling from $(\mathbf{S}_t, \mathrm{Alg}_{\mathbf{S}_t}^{\infty}) \sim \mathcal{S}_k^{\infty}(n,\eta)$, and storing $\mathbf{S}_t \boldsymbol{z}^{(t)}$. Letting $\boldsymbol{t} = \boldsymbol{h}(j)$, notice that whenever $\boldsymbol{h}(j') \neq \boldsymbol{t}$ for all $j' \in J \setminus \{j\}$ (which occurs with probability at least $(1 - 1/(10k))^{k-1} \geq 0.9$), then $\boldsymbol{z}^{(\boldsymbol{t})} = y$. Hence, the algorithm $\mathrm{Alg}_{\mathbf{S}}^{\infty}(\mathbf{S}_{\boldsymbol{t}} \boldsymbol{z}^{(\boldsymbol{t})})$ recovers y up to ℓ_∞ error at most $\eta \|y_{-1/\eta^2}\|_2$. We note that the nested application of $\boldsymbol{h}$ and $\mathbf{S}_1, \ldots, \mathbf{S}_{10k}$ may be represented as a matrix multiplication of x, and we may boost error probability by repeating $O(\log n)$ times and taking the median.

□

We will sometimes use the following shorthand notation to represent the error yielded from Theorem 105.

Definition 8.6.6. *Given a dimension n, $t \geq 0$, and a partition $U = \{U_\ell\}_{\ell \in [m]}$ of $[n]$, let $x \in \mathbb{R}^n$. Let $J \subset [m]$ be a subset of the partition indices. Then we write $x_{-(J,t)} = (y^J)_{-t}$, where y^J is defined via*

$$y_i^J = \begin{cases} x_i & \text{if } i \notin \cup_{\ell \in J} U_\ell \\ 0 & \text{otherwise} \end{cases}$$

We now restate Theorem 106 once more, itself a simplification of Indyk's p-stable sketch, with the addition of an error parameter ϵ.

Theorem 106 (L_1-sketch, reformulation of Theorem 8)**.** *Fix $n \in \mathbb{N}$, $\epsilon, \delta \in (0,1)$, and $k = O(\log(1/\delta)/\epsilon^2)$. Let $\mathcal{S}_k^1(n)$ be the distribution on $k \times n$ matrices $\mathbf{\Omega}$ with independent Cauchy random variables $\mathcal{C}$, and let Alg^1 be the algorithm which takes a vector $y \in \mathbb{R}^k$, and outputs*

$$\mathrm{Alg}^1(y) \stackrel{\text{def}}{=} \text{median}\left\{ \frac{|y_i|}{\text{median}(|\mathcal{C}|)} : i \in [k] \right\}, \quad \textit{where} \quad \text{median}(|\mathcal{C}|) \stackrel{\text{def}}{=} \sup\left\{ t : \mathop{\mathbf{Pr}}_{\boldsymbol{\omega} \sim \mathcal{C}} [|\boldsymbol{\omega}| \leq t] \leq \frac{1}{2} \right\}.$$

Then, for any vector $x \in \mathbb{R}^n$,

$$\mathop{\mathbf{Pr}}_{\mathbf{\Omega} \sim \mathcal{S}_k^1(n)} \left[\left| \mathrm{Alg}^1(\mathbf{\Omega} x) - \|x\|_1 \right| \leq \epsilon \|x\|_1 \right] \geq 1 - \delta.$$

8.6.3 Construction of the Sketch

Fix any $i \in [h]$, we now describe the sketching algorithm to estimate $\mathcal{I}_i$. We include a table of notation used in this section and the following in Figure 8.1. As we will see, we will not always be able to obtain a good approximation to $\mathcal{I}_i$ for all levels i. Specifically, for the levels i for which $\mathcal{I}_i$ is not sufficiently large, we will not obtain an estimate of $\mathcal{I}_i$, however we will be able to safely ignore such "small" levels. Moreover, we will be able to easily determine when a level is small, and avoid obtaining an estimate for it. Thus, we now describe the construction for the sketch for level i. Recall that the input to the $\mathrm{EMD}(A, B)$ sketching problem can be represented as a vector $f_{A,B} \in \mathbb{R}^{2 \cdot 2^d}$, where the first 2^d coordinates describe the multi-set $A \subset \{0,1\}^d$ and the last 2^d coordinates describe $B \subset \{0,1\}^d$. We design a linear sketch which can be represented as $\mathbf{S} \cdot f_{A,B}$ for some matrix $\mathbf{S}$.

As discussed earlier, our approach will be to first sample a vertex v^* from $\mathcal{V}_i$, such that

$$\mathbf{Pr}\left[v^* = v\right] \tilde{\propto} \left| |A_v| - |B_v| \right|,$$

Table 8.1: Table of Notation

η	$\triangleq$	precision for Count-Sketch
ρ	$\triangleq$	$\Theta(\frac{1}{\epsilon_0^2}\log s)$ number of Cauchy Sketches
$\mathbf{X}$	$\triangleq$	matrix Encoding parents in $\mathcal{V}_{i-1}$
$\mathbf{Y}$	$\triangleq$	matrix Encoding children in $\mathcal{V}_i$
Δ_i	$\triangleq$	$\sum_{v\in\mathcal{V}_i} \|\|A_v\| - \|B_v\|\|$
$\Delta_i(u)$	$\triangleq$	$\sum_{v\,:\,\pi(v)=u} \|\|A_v\| - \|B_v\|\|$
λ_u	$\triangleq$	vector with coordinates $\lambda_{u,v} = \|A_v\| - \|B_v\|$
$\omega_{u,j}$	$\triangleq$	vector of i.i.d. Cauchy random variables
$\mathbf{\Omega}_u$	$\triangleq$	$\rho \times d$ matrix of i.i.d. Cauchy random variables
$\mathbf{D}^1, \mathbf{D}^2$	$\triangleq$	Diagonal exponential scaling matrices
$\mathbf{Z}^1$	$\triangleq$	$\mathbf{D}^1\mathbf{X}$
$\mathbf{Z}^2$	$\triangleq$	$\mathbf{D}^2\mathbf{Y}$
$\mathbf{S}^1$	$\triangleq$	Count-Sketch Matrix from Theorem 104
$\mathbf{S}^2$	$\triangleq$	*Nested* Count-Sketch Matrix from Theorem 105
$\tilde{\mathbf{Z}}^j$	$\triangleq$	estimate of $\mathbf{Z}^j$ from Count-Sketch and Nested Count-Sketch

where $\tilde{\propto}$ is meant to indicate that the relationship is approximately proportional (since we encounter some errors). Letting u^* be the parent of v^*, the sketch will estimate q_{u^*,v^*}. If we can produce $\mathrm{polylog}(s,d)$ samples from such a distribution and obtain estimates of q_{u^*,v^*}, the empirical mean will produce an estimate to $\mathcal{I}_i$. We design a linear data-structured based on precision sampling that will allow us to both sample v^* and simultaneously recover an estimate of q_{u^*,v^*}. To do this, we define two matrices $\mathbf{X}, \mathbf{Y}$ whose entries will be linear combinations of the entries of f. The first matrix $\mathbf{X}$ encodes information about the nodes in the level $\mathcal{V}_{i-1}$, and $\mathbf{Y}$ will encode information about the children nodes in $\mathcal{V}_i$.

The rows of $\mathbf{X}$ will be indeed by nodes $u \in \mathcal{V}_{i-1}$, and for each row $\mathbf{X}_u$ we will store several pieces of information: **(1)** an approximation of the total discrepancy of its children, namely $\sum_{v:\pi(v)=u} \|\|A_v\| - \|B_v\|\|$, and **(2)** sufficient meta-data to compute the center c_u. Similarly, in $\mathbf{Y}$, each row is indexed by a vertex $v \in T_i$, and will store the discrepancy $\|\|A_v\| - \|B_v\|\|$ at that node, as well as sufficient information to compute the center c_v. The complete details now follow.

Let η be the precision parameter for count-sketch which we will soon fix, and let ϵ_0 be another precision parameter for Cauchy-sketches that set to $\epsilon_0 = \Theta(1/\log s)$. In what follows, let Δ_i be the total discrepancy at level i, namely $\Delta_i = \sum_{v\in\mathcal{V}_i} \|\|A_v\| - \|B_v\|\|$. For each parent node

$u \in \mathcal{V}_{i-1}$, define the vector $\lambda_u \in \mathbb{R}^{2^{2^i}}$ indexed by the children of u. Specifically, for each $v \in \mathcal{V}_i$ with $\pi(v) = u$, we have an entry $\lambda_{u,v}$, with the value $\lambda_{u,v} = |A_v| - |B_v|$. Thus, λ_u is the vector of discrepancies at u, and we have $\sum_{u \in \mathcal{V}_{i-1}} \|\lambda_u\|_1 = \Delta_i$.

Construction of the Matrices. We first define the matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}_i| \times D_1}$, where $D = 2\rho + 1$, and $\rho = \Theta(\epsilon_0^{-2} \log s)$. Each row, denoted $\mathbf{X}_u$, of $\mathbf{X}$ corresponds to a unique vertex $u \in \mathcal{V}_{i-1}$ (i.e., the rows of $\mathbf{X}$ are indexed by vertices in $\mathcal{V}_{i-1}$). Let $\mathbf{X}_{u,i}$ denote the i-th entry in the v-th row of A. Ideally, we would like to set $\mathbf{X}_{u,1} = \sum_{v:\pi(v)=u} ||A_v| - |B_v|| = \|\lambda_u\|_1$. However, this will not be possible to do in the linear sketching setting, because $\|\lambda_u\|_1$ cannot be expressed as a linear function of the input vector f. Instead, we will try to approximate this value ℓ_1 norm of $\|\lambda_u\|_1$ with a Cauchy sketch. Specifically, for $i = 1, 2, \ldots, \rho$, we generate a vector $\omega_{u,i} \in \mathbb{R}^{2^{2^i}}$ of i.i.d. Cauchy random variables $\mathcal{C}$. Note that the algorithm will only need to actually generate at most $2s$ random variables in the vector $\omega_{u,i}$, and moreover $\omega_{u,i}$ can be stored in small space using limited independence [KNW10a]. Next, for $i = 1, 2, \ldots, \rho$, we set $\mathbf{X}_{u,i} = \langle \lambda_u, \omega_{u,i} \rangle$. Note that by Theorem 106, we have $\frac{1}{\text{median}(|\mathcal{C}|)} \text{median}_{i \in [\rho]} |\mathbf{X}_{u,i}| = (1 \pm \epsilon_0)\|\lambda_u\|_1$.

Next, for the remaining $\rho+1$ coordinates in each row $\mathbf{X}_u$, we set $\mathbf{X}_{u,\rho+1} = |A_u| + |B_u|$, which can be accomplished exactly via an linear combination of the entries in f. Lastly, we generate an i.i.d. Cauchy matrix $\Omega_u \in \mathbb{R}^{\rho \times d}$, and e set the final ρ entries to be $\Omega_u \cdot \left(\sum_{p \in A_u \cup B_u} p \right)$, where $p \in \{0,1\}^d$ is thought of as a column vector. Thus, the information in the row $\mathbf{X}_u$ contains a $(1 \pm \epsilon_0)$ approximation of the total discrepancy at the vertex u, the number of points in $|A_u \cup B_u|$, and the sum of the points in u after being multiplied by a Cauchy sketch.

We now describe the construction of the matrix $\mathbf{Y}$, which will be similar but simpler since we will not need the Cauchy sketch for the discrepancies. For each $v \in \mathcal{V}_i$, we similarly have a assigned row $\mathbf{Y}_v$. We set $\mathbf{Y}_{v,1} = ||A_v| - |B_v||$, and we set $\mathbf{Y}_{v,2} = |A_v| + |B_v|$, and the last ρ coordinates of $\mathbf{Y}_v$ will be set to $\Omega_{\pi(v)} \cdot \left(\sum_{p \in A_v \cup B_v} p \right)$. In summary, we have

$$\mathbf{X}_u = \left[\langle \lambda_u, \omega_{u,1} \rangle, \quad \ldots \ , \quad \langle \lambda_u, \omega_{u,\rho} \rangle, \quad |A_u| + |B_u|, \quad \left(\Omega_u \cdot \sum_{p \in A_u \cup B_u} p \right)_1, \quad \ldots, \quad \left(\Omega_u \cdot \sum_{p \in A_u \cup B_u} p \right)_\rho \right]$$

$$\mathbf{Y}_v = \left[||A_v| - |B_v||, \quad |A_v| + |B_v|, \quad \left(\Omega_{\pi(v)} \cdot \sum_{p \in A_v \cup B_v} p \right)_1, \quad \ldots \ , \quad \left(\Omega_{\pi(v)} \cdot \sum_{p \in A_v \cup B_v} p \right)_\rho \right]$$

Construction of the sketch. The sketch is now as follows. We generate two count-sketch matrices $\mathbf{S}^1 \in \mathbb{R}^{k \times n_1}, \mathbf{S}^2 \in \mathbb{R}^{k' \times n_2}$ with precision parameter η, where $\mathbf{S}^1 \sim \mathcal{S}_k^{\infty}(n_1, \eta)$ is a standard count-sketch matrix from Theorem 104, and $\mathbf{S}^2 \sim \mathcal{S}_k^{\infty,U}(n_2, \eta)$ is a *nested-count sketch* from Theorem 105 (applied with $\epsilon = \epsilon_0 = \Theta(1/\log s)$), so that $k = O(1/\eta^2 \log(n_1 + n_2))$ and $k' = O(\eta^{-2} \log(n_1 + n_2) \log s)$, and where $n_1 = |\mathcal{U}_1| = \tilde{O}(s^2)$ and $n_1 = |\mathcal{U}_1 \times \mathcal{U}_2| = \tilde{O}(s^4)$. We also generate two diagonal precision scaling matrices $\mathbf{D}^1, \mathbf{D}^2$, where $\mathbf{D}^1_{u,u} = 1/t^1_u$, and $D^2_{v,v} = \frac{1}{t^1_{\pi(v)} t^2_v}$, where $t^j_i \sim Exp(1)$ are independent exponential random variables. In the sequel, we will drop the superscript and write t_v for some vertex v, since v will be in exactly one of $\mathcal{V}_i$ or $\mathcal{V}_{i-1}$ it will be clear whether t_v was from the first or second set of exponential variables. Next, we store a Cauchy sketch $\mathbf{\Omega}\mathbf{Y}_{*,1}$ with $O(\log(s)/\epsilon_0^2)$ rows, which by Theorem 106 gives a estimate $\tilde{s} = (1 \pm \epsilon_0)\Delta_i$ with probability $1 - 1/\text{poly}(s)$. Altogether, our algorithm stores the three linear sketches $\mathbf{S}^1\mathbf{D}^1\mathbf{X}$, $\mathbf{S}^2\mathbf{D}^2\mathbf{Y}$, and $\mathbf{\Omega}\mathbf{Y}_{*,1}$. To define how the error for the nested-count sketch $\mathbf{S}^2\mathbf{D}^2\mathbf{Y}$ is applied, we need to define a partition of the rows of $\mathbf{Y}$, which we do naturally by partitioning $\mathcal{V}_i$ into the subsets of children v which share the same parent in $u \in \mathcal{V}_{i-1}$. Notice that if $\tilde{\mathbf{Z}}^2$ is the estimate of $\mathbf{D}^2\mathbf{Y}$ produced by the nested count-sketch, we have for any fixed subset $U \subset \mathcal{V}_{i-1}$ of size $|U| = 1/\epsilon_0$, for any $v \in \mathcal{V}_i$ we have

$$\|\tilde{\mathbf{Z}}^2_{v,j} - (\mathbf{D}^2\mathbf{Y})_{v,j}| \leq \eta \|((\mathbf{D}^2\mathbf{Y})_{*,j})_{-(U \setminus \pi(v), 1/\eta^2)}\|_2$$

where for a vector x, $(x)_{-(U,c)}$ is defined as in Theorem 105. Notice that $\|((\mathbf{D}^2\mathbf{Y})_{*,j})_{-(U,1/\eta^2)}\|_2 \leq \|((\mathbf{D}^2\mathbf{Y})_{*,j})_{-1/\eta^2}\|_2$, so we will sometimes use this weaker bound when the tighter one is not required.

8.6.4 Analysis of the Algorithm.

We begin this section with a table of notation for reference, followed by the full algorithm. The algorithm proceeds in two main steps. First, it samples a parent $u^* \sim \mathcal{V}_{i-1}$ with probability proportional to $\Delta_i(u^*)$. It does this by first scaling a vector whose coordinates are approximations of $\Delta_i(u)$, for $u \in \mathcal{V}_{i-1}$, by independent inverse exponentials. By returning an approximation of the maximum coordinate after scaling, we can apply Lemma 8.6.4 to demonstrate that the parent u^* obtained is indeed from the correct distribution. We then iterate the process, searching over all children $v \in \mathcal{V}_i$ of u^*, and finally sampling v^* with $\pi(v^*) = u^*$ with probability proportional to $||A_{v^*}| - |B_{v^*}||$, again by the same procedure using Lemma 8.6.4. Once we have obtained our desired samples (u^*, v^*), we would like to output the quantity $\Delta_i \cdot q_{u^*,v^*}$, where q_{u^*,v^*} is as defined as in Equation 8.36. We can easily obtain an approximation $\tilde{\Delta}_i$ of Δ_i using a Cauchy

Sketch of Lemma 106. To obtain an approximation of q_{u^*,v^*}, we utilize the metadata contained within the rows $\mathbf{X}_{u^*,*}, \mathbf{Y}_{v^*,*}$, of which we have approximations of due to count-sketch.

We prove two main Lemmas which will yield the correctness of our algorithm. First Lemma 8.6.7 demonstrates that the pair (u^*, v^*) is sampled from the correct distribution. Second, the more involved Lemma 8.6.8 demonstrates that our approximation of q_{u^*,v^*} is sufficiently accurate. The first demonstrates that the sample $v^* \sim \mathcal{V}_i$ is drawn from the correct distribution. The second demonstrates that conditioned on sampling a $(u^*, v^*) \sim \mathcal{V}_{i-1} \times \mathcal{V}_i$, we can recover good approximations to the centers c_{u^*}, c_{v^*} *if* the parent u^* is not among a small set of "bad parents".

Procedure to Estimate $\mathcal{I}_i$

Input: $(\mathbf{S}^1\mathbf{D}^1\mathbf{X}, \mathbf{S}^2\mathbf{D}^2\mathbf{Y})$, an estimate $\tilde{\Delta}_i = (1 \pm \epsilon_0)\Delta_i$.
Parameters: $\epsilon > 0$, count-sketch precision $\eta = \Omega\left(\sqrt{\frac{\epsilon}{\log d \log^\ell s}}\right)$ for $\ell = O(1)$, and $\epsilon_0 = \Theta(1/\log s)$.

1. Via $\mathbf{S}^1\mathbf{D}^1\mathbf{X}$ and $\mathbf{S}^2\mathbf{D}^2\mathbf{Y}$ recover approximate matrices $\tilde{\mathbf{Z}}^1, \tilde{\mathbf{Z}}^2$ such that

$$\|\tilde{\mathbf{Z}}^1_{*,j} - (\mathbf{D}^1\mathbf{X})_{*,j}\|_\infty \le \eta \|(\mathbf{D}^1\mathbf{X}_{*,j})_{-1/\eta^2}\|_2$$
$$\|\tilde{\mathbf{Z}}^2_{*,j} - (\mathbf{D}^2\mathbf{Y})_{*,j}\|_\infty \le \eta \|(\mathbf{D}^2\mathbf{Y}_{*,j})_{-1/\eta^2}\|_2$$

   for all columns j.
2. For each row u of $\tilde{\mathbf{Z}}^1$, compute

$$\tilde{\Delta}_i(u) = \text{median}_{j\in[\rho]}|\tilde{\mathbf{Z}}^1_{u,j}, \quad \text{and} \quad u^* = \arg\max_u \tilde{\Delta}_i(u)$$

3. Compute $v^* = \arg\max_{v:\pi(v)=u^*} |\tilde{\mathbf{Z}}^2_{v,1}|$, and output

$$Q = \frac{\tilde{\Delta}_i}{\text{median}(|\mathcal{C}|)} \cdot \min\left\{\max\left\{\text{median}_j \left|\frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}} - \frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}}\right|, \frac{d}{2^i}\right\}, \frac{30d\log s}{2^i}\right\}$$

Figure 8.5: Main Sketching Primitive to Estimate $\mathcal{I}_i$

We state Lemmas 8.6.7 and 8.6.8 below, and proceed to prove our main Theorem 102 given these Lemmas. The proofs of Lemmas 8.6.7 and 8.6.8 will then be given in Section 8.6.4.

Lemma 8.6.7. *Let $v^* \in \mathcal{V}_i$ be the vertex which is sampled in Figure 8.5. Then for any $v \in \mathcal{V}_i$, we have $\mathbf{Pr}[\mathbf{v}^* = v] = (1 \pm \epsilon_0)\frac{||A_v|-|B_v||}{\Delta_i} \pm s^{-c}$, where $c > 1$ is an arbitrarily large constant.*

Lemma 8.6.8. *Fix η, ϵ_0 as in Figure 8.5, and let (u^*, v^*) be the samples accepted by Figure 8.5. Then assuming that $\frac{2s}{\Delta_i} < \frac{\log s \log d}{\epsilon 2^i}$, and moreover that $\frac{|A_{u^*}|+|B_{u^*}|}{\Delta_i(u^*)} \leq \frac{\log s \log d}{\epsilon \nu 2^i}$, where $\nu = \Theta(1/\log s)$, then with probability $1 - 1/\mathrm{poly}(s)$, we have*

$$\frac{1}{\mathrm{median}(|\mathcal{C}|)} \cdot \mathrm{median}_{j\in[\rho]} \left| \frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}} - \frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}} \right| = (1 \pm \epsilon_0)\|c_{u^*} - c_{v^*}\|_1 \pm \epsilon_1 \frac{d}{2^i}$$

where ϵ_1 is an arbitrarily small constant.

Finally, we will need a brief fact that allows us to disregard a small fraction $u \in \mathcal{V}_{i-1}$ which are in a subset of "bad parents".

Fact 8.6.9. *Fix $\gamma > 1$ and $\nu \in (0, 1)$, and suppose that we have $\frac{2s}{\Delta_i} < \gamma$. Let $W \subset \mathcal{V}_{i-1}$ be the set of $u \in \mathcal{V}_{i-1}$ such that $\frac{|A_{u^*}|+|B_{u^*}|}{\Delta_i(u)} \leq \frac{\gamma}{\nu}$. Then $\sum_{u\in W} \Delta_i(u) \geq (1-\nu)\Delta_i$*

Proof. Suppose otherwise. Then for each $u \notin W$, we have $\frac{\Delta_i(u)}{||A_{u^*}|+|B_{u^*}||} < \frac{\nu}{\gamma}$, so

$$\frac{\Delta_i}{2s} < \frac{1}{2s\nu} \sum_{u\notin W} \Delta_i(u) \leq \frac{1}{2\gamma s} \sum_{u\notin W} ||A_{u^*}| + |B_{u^*}|| \leq \frac{1}{\gamma} \tag{8.39}$$

Thus $\Delta_i/(2s) < \gamma$, a contradiction. □

We now restate Theorem 102, and prove it given Lemmas 8.6.7 and 8.6.8.

Theorem 102 *For $d, s \in \mathbb{N}$, there exists a 1-round linear sketching algorithm such that given multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$, computes an approximate $\hat{\mathcal{I}}$ to $\mathrm{EMD}(A, B)$ with*

$$\mathrm{EMD}(A, B) \leq \hat{\mathcal{I}} \leq \tilde{O}(\log s)\,\mathrm{EMD}(A, B) + \epsilon ds$$

with probability at least $3/4$. Moreover, the space used by the linear sketch is $\tilde{O}(1/\epsilon)$.

Proof. Fix any level $i \in [h]$ (where $h = O(\log d)$). First note that setting the Cauchy Sketch $\mathbf{\Omega Y}_{*,1}$ to be derandomized by [KNW10a], the matrix $\mathbf{\Omega Y}_{*,1}$ and $\mathbf{\Omega}$ can be stored using $O(1/\epsilon_0^2 \log^2(s))$ bits of space, and yields the approximation $\tilde{\Delta}_i = (1 \pm \epsilon_0)\Delta_i = (1 \pm \epsilon_0)\|\mathbf{Y}_{*,1}\|_1$ with probability $1 - 1/\mathrm{poly}(s)$. We condition on this correctness now, which is independent of the randomness of the rest of the algorithm, and use the same estimate $\tilde{\Delta}_i$ for all repetitions of the sampling

algorithm in Figure 8.5. Now note that if $\frac{2s}{\Delta_i} > \frac{\log s \log d}{4\epsilon 2^i}$, then

$$\mathcal{I}_i \leq \Delta_i \frac{30d \log s}{2^i} \leq O\left(\epsilon ds \frac{1}{\log d}\right)$$

so the total contribution of all such $i \in [\log d]$ is at most $O(\epsilon ds)$, which we absorb into the additive error (after rescaling ϵ by a constant). We can then use our estimate $\tilde{\Delta}_i = (1 \pm \epsilon_0)\Delta_i$ to test if this is the case. Specifically, if we have $\frac{2s}{\tilde{\Delta}_i} > \frac{\log s \log d}{2\epsilon 2^i}$, then we can be sure that $\frac{\log s \log d}{\epsilon 2^i} > \frac{2s}{\Delta_i} > \frac{\log s \log d}{4\epsilon 2^i}$, and in this case we set $\hat{\mathcal{I}}_i = 0$. Otherwise, we will attempt to estimate $\mathcal{I}_i$ via the algorithm in Figure 8.5. Notice that if we happened to choose $\epsilon > \Omega(\frac{\log s \log d}{d})$, then necessarily there will be some levels i (lower in the tree) such that $\frac{2s}{\Delta_i} > \frac{\log s \log d}{4\epsilon 2^i}$ no matter what, since for such a setting of ϵ the right hand side is less than 1. For smaller ϵ, however, every level $i \in [h]$ may have a chance to contribute.

We apply Lemma 8.6.8 with $\nu = \Theta(1/\log s)$ as small enough constant, and note by Fact 8.6.9, that the set W of $u \in \mathcal{V}_{i-1}$ that satisfy the assumption of Lemma 8.6.13 satisfies $\sum_{u \in W} \Delta_i(u) > (1-\nu)\Delta_i$. By the setting of ν and the fact that $q_{u,v}/q_{u',v'} < 30 \log s$ for any non-empty $v, v' \in \mathcal{V}_i$, it follows that $\sum_{u \in W} \sum_{v:\pi(v)=u} q_{u,v} = (1 \pm 1/100)\mathcal{I}_i$. By Lemma 8.6.8, conditioned on sampling $u^* \in W$, our estimate satisfies

$$\frac{1}{\text{median}(|\mathcal{C}|)} \cdot \text{median}_{j \in [\rho]} \left| \frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}} - \frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}} \right| = (1 \pm \epsilon_0)\|c_{u^*} - c_{v^*}\|_1 \pm \epsilon_1 \frac{d}{2^i}$$

with probability $1 - s^{-c}$. Conditioned on this, and using that $q_{\pi(v),v} \geq d/2^i$ for all $v \in \mathcal{V}_i$ by definition, we then have $Z = (1 \pm 8\epsilon_1)\Delta_i \cdot q_{u^*,v^*}$. Thus the expectation of Z conditioned on sampling (u^*, v^*) with $u^* \in W$ is $(1 \pm 8\epsilon_1)\Delta_i(q_{u^*,v^*} \pm s^{-c}(30d \log s/2^i)) = (1 \pm 10\epsilon_1)\Delta_i q_{u^*,v^*}$. Moreover, by Lemma 8.6.7, we know that $v^* = v$ with probability $(1 \pm \epsilon_0)||A_v| - |B_v||/\Delta_i \pm s^{-c}$. It follows that the probability that we sample $v^* = v \in \mathcal{V}_i$ with $\pi(v) \notin W$ is at most $(3/2)\nu$.

Putting these facts together, we have that

$$\begin{aligned}
\mathbf{E}\left[Q\right] &= \left(\sum_{u\in W}\sum_{v:\pi(v)=u}(1\pm\epsilon_0)\left(\frac{||A_v|-|B_v||}{\Delta_i}\pm s^{-c}\right)(1\pm 10\epsilon_1)\Delta_i\cdot q_{u,v}\right)\pm 60\nu\Delta_i\frac{d\log s}{2^i}\\
\mathbf{E}\left[Q\right] &= (1\pm 12\epsilon_1)\left(\sum_{u\in W}\sum_{v:\pi(v)=u}||A_v|-|B_v||\boldsymbol{q}_{u,v}\right)\pm\left(60\nu\Delta_i\frac{d\log s}{2^i}+s^{-c+3}\frac{d}{2^i}\right)\\
&\leq (1\pm 1/50))\mathcal{I}_i\pm\Delta_i\frac{d}{100\cdot 2^i}\\
&\leq (1\pm\frac{1}{20})\mathcal{I}_i
\end{aligned}\tag{8.40}$$

where we used that $\mathcal{I}_i\geq\Delta_i\frac{d}{2^i}$ by definition. Similarly, since $Q\leq\Delta_i\frac{20d\log s}{2^i}$ (conditioned on the high probability success of our estimate for Δ_i), it follows that $\mathbf{Var}[Q]<(\Delta_i\frac{20d\log s}{2^i})^2$. Thus, repeating the procedure $O(\log^2 s)$ times and obtaining $Q_1,Q_2,\ldots,Q_{O(\log^2 s)}$, we have that $\sqrt{\mathbf{Var}\left[\sum_{j=1}^{O(\log^2 s)}Q_j\right]}<\Delta_i\frac{d}{100\cdot 2^i}\leq\mathcal{I}_i/100$. By Chebyshev's inequality, we have $\sum_{j=1}^{O(\log^2 s)}Q_j=(1\pm 1/10)\mathcal{I}_i$ with probability at least $99/100$. Thus, after scaling $\hat{\mathcal{I}}_i$ up by a factor of $(1+1/3)$, we can set $\hat{\mathcal{I}}=\sum_i\hat{\mathcal{I}}_i+O(\epsilon s)$, and after scaling ϵ down by a constant we have $\mathcal{I}\leq\mathcal{I}_i\leq 2\mathcal{I}+\epsilon s$. By Remark 98, we know that $\mathrm{EMD}(A,B)\leq\mathcal{I}\leq\tilde{O}(\log s)\,\mathrm{EMD}(A,B)$ with probability at least .89 over the draw of the tree, which completes the proof of the claimed approximation after a union bound over all aforementioned events. To see the space complexity, note for each of the $h=O(\log d)$ levels, and for each of the $\tilde{O}(1)$ samples used in that level, we the size of the count-sketch and nested count-sketch is $\tilde{O}(\eta^{-2})$ by Theorems 104 and 105. Notice that to apply Theorems 104 and 105, we used the fact that, after the universe reduction step from Proposition 8.5.1, both $\mathbf{X},\mathbf{Y}$ have at most $\mathrm{poly}(s)$ rows. This completes the proof after noting that $\eta=\tilde{O}(\sqrt{\epsilon/d})$. □

A Streaming Algorithm. Next, we demonstrate how this linear sketching algorithm results in a one-pass streaming algorithm for the turnstile model. Recall in this model, a sequence of at most $\mathrm{poly}(s)$ updates arrives in the stream, where each update either inserts or deletes a point $p\in\{0,1\}^d$ from A, or inserts or deletes a point from B. Noticed that the t-th update can be modeled by coordinate-wise updates to $f_{A,B}$ of the form $(i_t,\Delta_t)\in[2\cdot 2^d]\times\{-1,1\}$, causing the change $(f_{A,B})_{i_t}\leftarrow(f_{A,B})_{i_t}+\Delta_t$. At the end of the stream, we are promised that $f_{A,B}$ is a valid encoding of two multi-sets $A,B\subset\{0,1\}^d$ with $|A|=|B|=s$.

Corollary 8.6.10. *For $d,s\in\mathbb{N}$, there exists a one-pass turnstile streaming algorithm which, on*

a stream vector $f_{A,B}$ encoding multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$, the algorithm then computes an approximate $\hat{\mathcal{I}}$ to $\mathrm{EMD}(A, B)$ *with*

$$\mathrm{EMD}(A, B) \leq \hat{\mathcal{I}} \leq \tilde{O}(\log s)\,\mathrm{EMD}(A, B)$$

with probability at least $3/4$*, and uses* $O(d + 1/\epsilon) \cdot \mathrm{polylog}(s, d)$ *bits of space.*

Proof. As in Corollary 8.12.1, we explicitly store the $O(d \log d)$ bits of randomness required to specify the entire Quadtree T, which we now fix. Also, as in the proof of Corollary 8.12.1, we use $O(d)$ bits of space to generate the 2-wise independent hash functions h_i, h_{i-1} needed for the universe reduction step of Proposition 8.5.1. Moreover, the randomness used in the Cauchy sketch $\mathbf{\Omega Y}_{*,1}$ can be derandomized by the results of [KNW10a] to use only $O(1/\epsilon_0^2 \log^2 s)$ bits of space and succeed with probability $1 - 1/\mathrm{poly}(s)$.

We now note that there are several other sources of randomness that must be deranndomzied: namely the parent and child exponentials $\{t_u\}_{u \in \mathcal{V}_{i-1}}, \{t_v\}_{v \in \mathcal{V}_i}$, the Cauchy sketches $\{\Omega_u\}_{u \in \mathcal{V}_{i-1}}$, and $\{\omega_{u,j}\}_{u \in \mathcal{V}_{i-1}, j \in [\rho]}$. We remark that the randomness needed for the count-sketch and nested count sketch matrices $\mathbf{S}^1, \mathbf{S}^2$ of Theorems 104 and 105 use 4-wise independent hash functions, and therefore do not need to be derandomized. We first remark that we can apply a standard truncation of these continuous distributions to be stored in $O(\log sd)$ bits of space (by simply only storing generating them to $O(\log sd)$ bits of accuracy). This introduced a $1/\mathrm{poly}(sd)$ additive error into each coordinate of the linear sketch. First note that this additional error can be absorbed into the error of the count sketch estimates $\tilde{\mathbf{Z}}^1, \tilde{\mathbf{Z}}^2$, increasing the error bounds by at most a $(1 + 1/\mathrm{poly}(sd)))$ factor. The analysis from Lemmas 8.6.7 and 8.6.8 apply with this (slightly) larger error after scaling the precision parameter η by the same factor.

We demonstrate how to derandomize these (now discrete) distributions. We adapt a standard argument due to Indyk [Ind06] based on Nisan's PRG [Nis92]. Since the algorithm is a linear sketch, we can reorder the stream so that for every $u \in \mathcal{V}_{i-1}$ all updates to points $x \in \{0,1\}^d$ with $v_{i-1}(x) = u$ occur in a consecutive block B_u of updates, and B_u is broken up further into blocks B_v for all children v of u, such that all updates to points $x \in \{0,1\}^d$ with $v_i(x) = v$ occur in the consecutive block B_v. We now describe a small space tester which computes the output of our algorithm while reading the randomness for the exponentials $\{t_u\}_{u \in \mathcal{V}_{i-1}}, \{t_v\}_{v \in \mathcal{V}_i}$, the Cauchy sketches $\{\mathbf{\Omega}_u\}_{u \in \mathcal{V}_{i-1}}$, and $\{\omega_{u,j}\}_{u \in \mathcal{V}_{i-1}, j \in [\rho]}$, in a stream. When the block B_u begins for any $u \in \mathcal{V}_{i-1}$, the algorithm reads and stores the randomness needed for $t_u, \mathbf{\Omega}_u$ and $\{\omega_{u,j}\}_{j \in [\rho]}$, which is a total of $\tilde{O}(d\rho) = \tilde{O}(d)$ bits of space. Within the block B_u, when the updates for a block B_v begin for any child v of u, we also store the random variable t_v. We then can fully

process all updates to the linear sketches $\mathbf{S}^1\mathbf{D}^1\mathbf{X}, \mathbf{S}^2\mathbf{D}^2\mathbf{Y}$ due to the updates in B_v in space $\tilde{O}(d)$ using only the randomness stored within the blocks B_u, B_v.

After the block B_v is complete, we can discard t_v, since it will never be needed for to process any other update. Similarly, once B_u is complete, we can discard the stored randomness $t_u, \mathbf{\Omega}_u$ and $\{\omega_{u,j}\}_{j\in[\rho]}$, since it will not be needed to process updates to any other point in the stream. The total space required to compute the entire sketches $\mathbf{S}^1\mathbf{D}^1\mathbf{X}, \mathbf{S}^2\mathbf{D}^2\mathbf{Y}$ is $\tilde{O}(d + 1/\epsilon)$, where the $\tilde{O}(d)$ comes from storing the randomness in the blocks B_u, the Quadtree randomness, and the hash functions h_i, h_{i-1}, and $\tilde{O}(1/\epsilon)$ is the space required to store the sketches $\mathbf{S}^1\mathbf{D}^1\mathbf{X}, \mathbf{S}^2\mathbf{D}^2\mathbf{Y}$. Since after the universe reduction step we have $|\mathcal{V}_{i-1}|, |\mathcal{V}_i| = \text{poly}(s)$, our algorithm requires a total of $\text{poly}(s, d)$ bits of randomness, and can be tested in by a space $\tilde{O}(d + 1/\epsilon)$ algorithm that reads the random bits to be deradnomized in a stream. Thus, applying Nisans PRG [Nis92], it follows that the algorithm can be derandomized with a multiplicative increase of $O(\log sd)$ bits of space over the space of the tester, which completes the proof.

□

Proofs of Lemmas 8.6.7 and 8.6.8

We now provide the proofs of Lemmas 8.6.7 and 8.6.8. We restate the Lemmas again here for convenience.

Lemma 8.6.7 *Let $v^* \in \mathcal{V}_i$ be the vertex which is sampled in Figure 8.5. Then for any $v \in \mathcal{V}_i$, we have $\mathbf{Pr}[v^* = v] = (1 \pm \epsilon_0)\frac{||A_v|-|B_v||}{\Delta_i} \pm s^{-c}$, where $c > 1$ is an arbitrarily large constant.*

Proof. We first condition on the success of the count-sketch matrices $\mathbf{S}^1, \mathbf{S}^2$ on all $O(\rho)$ columns of $\mathbf{X}, \mathbf{Y}$, which by a union bound occurs with probability $1 - s^{-c}$ for any constant c by Theorem 104 and Theorem 105. The possibility of the failure of this event will only introduce an additive s^{-c} into the variation distance of the distribution of the sampler, which is safely within the guarantees of the theorem. In the following, C, C', C'' will be sufficiently large constants.

It will now suffice to bound our error in estimating the target vectors, since we can then apply Lemma 8.6.4. First, we show that $|\tilde{\mathbf{Z}}^1_{v,j} - (\mathbf{D}^1\mathbf{X})_{v,j}| \leq \eta\Delta_i$ with probability $1 - 1/\text{poly}(s)$ for each of the first $j \in [\rho]$ columns. By the Count-Sketch guarantee, we know that $|\tilde{\mathbf{Z}}^1_{v,j} - (\mathbf{D}^1\mathbf{X})_{v,j}| \leq \eta\|(\mathbf{D}^1\mathbf{X}_{*,j})_{-1/\eta^2}\|_2$. First, since inverse exponentials satisfies $\mathbf{Pr}[1/x > t] < 1/t$ for all $t > 1$, by Lemma 8.6.5 we have $\|(\mathbf{D}^1\mathbf{X}_{*,j})_{-1/\eta^2}\|_2 \leq C\eta\|(\mathbf{X}_{*,j})_{-1/(2\eta^2)}\|_1$ with probability $1 - 1/\text{poly}(s)$ for some constant C. Next, recall that $\mathbf{X}_{u,j} = \langle\lambda_u, \omega_{u,j}\rangle$, where $\omega_{u,j}$ is a vector of independent Cauchy random variables, which by 1-stability of the Cauchy

distribution is itself distributed as $\omega\|\lambda_u\|_1 = \omega\Delta_i(u)$ where ω is a Cauchy random variable. Thus the column vector $\mathbf{X}_{*,j}$ is the result of scaling the column vector with coordinates equal to $\Delta_i(u)$ for each $u \in \mathcal{V}_{i-1}$ by independent Cauchy random variables, which also satisfy the tail bound $\mathbf{Pr}[1/x > t] < 1/t$ for all $t > 1$. So applying Lemma 8.6.5 on the non-zero rows of $\mathbf{X}$ (of which there are at most $2s$), we have

$$\|(\mathbf{X}_{*,j})_{-1/(2\eta^2)}\|_1 \leq C\log s \sum_{u\in\mathcal{V}_{i-1}} \Delta_i(u) = (C\log s)\cdot\Delta_i$$

with probability $1-s^{-c}$. Butting these bounds together, we have $|\tilde{\mathbf{Z}}^1_{u,j} - (\mathbf{D}^1\mathbf{X})_{u,j}| \leq (C'\eta^2\log s)\Delta_i$ for all $u \in \mathcal{V}_{i-1}$ and $j \in [\rho]$. It follows that setting $\tilde{\Delta}_i(u) = \text{median}_{j\in[\rho]}|\tilde{\mathbf{Z}}^1_{u,j}|$, the median can change by at most the error $(2C'\eta^2\log s)\Delta_i$, so

$$\left|\tilde{\Delta}_i(u) - \frac{1}{t_u}\text{median}_{j\in[\rho]}|\mathbf{X}_{u,j}|\right| \leq (2C'\eta^2\log s)\Delta_i$$

Thus, if we define a vector $a \in \mathbb{R}^{|\mathcal{V}_{i-1}|}$ via the coordinates $a_u = \text{median}_{j\in[\rho]}|\mathbf{X}_{u,j}|$, we obtain an entry-wise approximation $\tilde{\Delta}_i(u)$ to the scaling $\mathbf{D}^1 a$ with error at most $(2C'\eta^2\log s)\Delta_i$. By standard arguments for p-stable variables (Indyk's p-stable sketch [Ind06], see Theorem 8), we have that $\text{median}_{j\in[\rho]}|\mathbf{X}_{u,j}| = (1\pm\epsilon_0)\Delta_i(u)$ with probability $1-s^{-c}$, thus $\|a\|_1 = (1\pm\epsilon_0)\Delta_i$ after a union bound over all $2s$ non-zero coordinates. So by Lemma 8.6.4 applied to the starting vector a, we have that if $u^* = \arg\max_u .\tilde{\Delta}_i(u)$ then we have $\mathbf{Pr}[u^* = u] = (1\pm O(\log s\eta^2))\frac{|a_u|}{\|a\|_1} \pm s^{-c}$. Moreover, since $|a_u| = (1\pm\epsilon_0)|\Delta_i(u)|$ for all u, and $\eta < \epsilon_0$, it follows that $\mathbf{Pr}[u^* = u] = (1\pm 2\epsilon_0)\frac{\Delta_i(u)}{\Delta_i} \pm s^{-c}$ as needed.

Next, we move on to the sampling of v^* given u^*. We begin by bounding the tail $\|(\mathbf{D}^2\mathbf{Y}_{*,1})_{-1/\eta^2}\|_2$. Similarly as before, we can bound this by $C\eta\sum_{v\in\mathcal{V}_i}\left(|\mathbf{Y}_{v,1}|/t_{\pi(v)}\right)$ with probability $1-s^{-c}$ using Lemma 8.6.5 applied to the exponential t_v for the children. Note that $\sum_{v\in\mathcal{V}_i}\left(|\mathbf{Y}_{v,1}|/t_{\pi(v)}\right) = \|b\|_1$, where b is the vector with coordinates $b_u = \Delta_i(u)/t_u$. Next, we will prove that $\|b\|_1 \leq C''\log s(\Delta_i(u^*)/t_{u^*} + \Delta_i)$ with high probability. To see this, notice $\|b\|_1 \leq \log s\|b\|_\infty + \|b_{-\log s}\|_1$, so we can apply Lemma 8.6.5 to bound $\|b_{-\log s}\|_1 \leq (C\log s)\Delta_i$ with probability $1 - s^{-c}$. Since our algorithm choose u^* as the maximizer of b, we have

$$\left|\frac{\Delta_i(u^*)}{t_{u^*}} - \|b\|_\infty\right| \leq (1+O(\epsilon_0))2C'\eta^2\log s\Delta_i < \frac{\Delta_i}{100c\log s}$$

where c is a constant, which follows due to our count-sketch error in estimating $\mathbf{Z}$ as argued above, after setting $\eta = \Omega\left(\sqrt{\frac{\epsilon}{\log d\log^\ell s}}\right)$ with a small enough constant, where $\ell \geq 2$. Given this, it

follows that $\|b\|_\infty < \Delta_i(u^*)/t_{u^*} + \Delta_i/(\log s)$, from which the bound $\|b\|_1 \leq C'' \log s(\Delta_i(u^*)/t_{u^*} + \Delta_i)$ follows. Now notice that since $\|b\|_\infty$ is the max order statistic of a set of independent exponential, by the results of Section 8.6.1 we have the distributional equality $\|b\|_\infty = \Delta_i/E_1$ where E_1 is an exponential random variable, so with probability $1 - s^{-c}$ we have $\|b\|_\infty > \Delta_i/(c \log s)$, thus we also have

$$\frac{\Delta_i(u^*)}{t_{u^*}} > \frac{1}{2}\|b\|_\infty > \frac{\Delta_i}{2c \log s} \tag{8.41}$$

Thus, plugging everything into the Count-Sketch guarantee, we obtain

$$\begin{aligned} \left|\tilde{\mathbf{Z}}^2_{v,1} - \mathbf{D}^2\mathbf{Y}_{v,1}\right| &\leq C''\eta^2 \log s \left(\frac{\Delta_i(u^*)}{t_{u^*}} + \Delta_i\right) \\ &\leq O\left(\eta^2 \log^2 s \left(\frac{\Delta_i(u^*)}{t_{u^*}}\right)\right) \end{aligned} \tag{8.42}$$

for all $v \in \mathcal{V}_i$. Recall that we sample v^* by choosing $v^* = \arg\max_{v:\pi(v)=u^*} |\tilde{\mathbf{Z}}^2_{v,1}|$. We already have a bound on the error of the estimation given by $\tilde{\mathbf{Z}}^2_{*,1}$. Thus, to apply Lemma 8.6.4, the only remaining piece is to notice that the original ℓ_1 norm of the vector we are sampling from is $\sum_{v:\pi(v)=u^*} \mathbf{Y}_{v,1} = \frac{1}{t_{u^*}} \sum_{v:\pi(v)=u^*} ||A_v| - |B_v|| = \frac{\Delta_i(u)}{t_u}$ after first fixing the parent exponentials t_u. Thus we can apply Lemma 8.6.4 with the error parameter $O(\eta^2 \log^2 s)$, which is at most ϵ_0 after setting η with a small enough constant, to obtain that $\mathbf{Pr}\left[v^* = v \mid u^*\right] = (1 \pm \epsilon_0)\frac{||A_v| - |B_v||}{\Delta_i(u^*)} \pm s^{-c}$. Noting that the randomness for which this event is determined only depends on the randomness in the second set of exponentials t_v^2, we have that

$$\begin{aligned} \mathbf{Pr}\left[v^* = v\right] &= \left((1 \pm \epsilon_0)\frac{||A_v| - |B_v||}{\Delta_i(u^*)} \pm s^{-c}\right)\left((1 \pm 2\epsilon_0)\frac{\Delta_i(u^*)}{\Delta_i} \pm s^{-c}\right) \\ &= (1 \pm 4\epsilon_0)\frac{||A_v| - |B_v||}{\Delta_i} \pm 3s^{-c} \end{aligned} \tag{8.43}$$

which yields the desired theorem after rescaling of c and ϵ_0 by a constant. □

To prove the next main Lemma, Lemma 8.6.8, we will need a few minor technical propositions.

Proposition 8.6.11. *Let X be an inverse exponential random variable, and let Y be a random*

variable supported on $[1,\infty)$ *with the property that for all* $t \geq 1$*:*

$$\mathbf{Pr}\left[Y > t\right] < \frac{f(t)}{t}$$

for some non-decreasing function $f : [1,\infty) \to [1,\infty)$ *with* $f(t) = o(t)$*. Then there is a universal constant* C *such that* $\mathbf{Pr}\left[XY > t\right] \leq C\frac{f(t)\log(t+1)}{t}$ *for all* $t \geq 1$*.*

Proof. Note that X has the probability density function $p(x) = \frac{1}{x^2}e^{-\frac{1}{x}}$. Thus, we have

$$\begin{aligned}
\mathbf{Pr}\left[XY > t\right] &\leq \int_{x=0}^{t} \mathbf{Pr}\left[Y > \frac{t}{x}\right]\frac{1}{x^2}e^{-\frac{1}{x}}dx + \int_{x=t}^{\infty}\frac{1}{x^2}e^{-\frac{1}{x}}dx \\
&\leq \int_{x=0}^{1}\mathbf{Pr}\left[Y > t\right]dx + \int_{x=1}^{t}\mathbf{Pr}\left[Y > \frac{t}{x}\right]\frac{1}{x^2}e^{-\frac{1}{x}}dx + O(\frac{1}{t}) \\
&\leq O\left(\frac{f(t)}{t}\right) + \frac{f(t)}{t}\int_{x=1}^{t}\frac{1}{x}e^{-\frac{1}{x}}dx \\
&\leq O\left(\frac{f(t)\log(t+1)}{t}\right)
\end{aligned} \tag{8.44}$$

where in the second line, we used that $e^{-1/x}/x^2 < 1$ for all $x > 0$. □

Proposition 8.6.12. *Let* $x \in \mathbb{R}^n$ *be any vector with* $\|x\|_1 = s$*. Let* X *be an inverse exponential random variable, and let* $Y_1, Y_2, \ldots, Y_n$ *be independent random variables supported on* $[1,\infty)$*,with the property that* $\mathbf{Pr}\left[Y_i > t\right] < f(t)/t$ *for each* i *and all* $t > 1$*, where* $f(t) = \log^{O(1)}((t+1)\mathrm{poly}(s))$*. Let* $\lambda_1, \lambda_2, \ldots, \lambda_n$ *be independent variables, where* $\lambda_i \sim X_iY_i$*, and* X_i *is an independent copy of* X*. Let* $z \in \mathbb{R}^n$ *be defined via* $z_i = x_i\lambda_i$*. Then we have* $\|z_{-\beta}\|_1 \leq C(\log s)(\log n)f(s)\|x\|_1$ *for a fixed constant* C*, with probabiltiy* $1 - e^{-\beta/8}$*.*

Proof. We have $\mathbf{Pr}\left[z_i > C\log s f(s)\|x\|_1\right] \leq \frac{x_i}{10\|x\|_1}$ by Proposition 8.6.11, for some sufficiently large constant C. By Chernoff bounds, with probability $1 - s^{-\beta}$ there are at most $\beta/2$ coordinates $i \in [n]$ with $z_i > C\log s f(s)\|x\|_1$. Thus if we truncate the variables λ_i into new variables λ_i' by enforcing that

$$\lambda_i' < 100\log s f(s)\frac{\|x\|_1}{x_i}$$

for all $i \in [n]$, and set $z' = \lambda_i' x_i$, the we have $\|z_{-\beta}\|_1 \leq \|z'_{-\beta/2}\|_1$. We can then apply Lemma 8.6.5 on the truncated variables to bound $\|z'_{-\beta/2}\|_1$, noting that for all $t \geq 1$ we have $\mathbf{Pr}\left[\lambda_i' > t\right] < \frac{\alpha}{t}$ where $\alpha = O(f(s)\log s)$, to obtain the proposition. □

We will also need the following technical lemma. It appears as a special case of Lemma 4.3.3

from Chapter 4, but with the assumption that X_i's are non-negative vectors, and a tighter bound of $\|\sum_i X_i\|_1$ instead of $\sum_i \|X_i\|_1$ (note that this is only tighter if the X_i's are not non-negative). Our vectors will in fact be non-negative, but we provide a simple proof for version of the lemma where the vectors may have negative coordinates.

Proposition 8.6.13 (Special case of Lemma 4.3.3 from Chapter 4)**.** *Let $Z \in \mathbb{R}^d$ be a vector of i.i.d. Cauchy random variables. Let $X_1, \ldots, X_k \in \mathbb{R}^d$ be fixed vectors. Then there is a fixed constant C, such that for any $t \geq 1$, we have*

$$\mathbf{Pr}\left[\sum_{i=1}^{k} |\langle Z, X_i\rangle| \geq C\log(tk)t \cdot \sum_{i=1}^{k} \|X_i\|_1\right] \leq \frac{1}{t}$$

Proof. The quantity $\langle \omega, p_i\rangle$ is distributed as $\alpha_i\|p_i\|_1$ where the α_i's are *non-independent* Cauchy random variables. Let $\mathcal{E}_i$ be the event that $|\alpha_i| \leq 1/\delta$, which occurs with probability $1-\delta$ by the tails of Cauchy variables. Let $\mathcal{E} = \cap_i \mathcal{E}_i$, and note $\mathbf{Pr}[\mathcal{E}] > 1 - O(k\delta)$ by a union bound. We have

$$\mathbf{E}[|a_i| \mid \mathcal{E}_i] \leq 3\int_{x=0}^{1/\delta} \frac{x}{\pi(1+x^2)} \leq \log(1/\delta)/2$$

It follows that $\mathbf{E}[|a_i| \mid \mathcal{E}] \leq 2\,\mathbf{E}[|a_i| \mid \mathcal{E}_i]$ since $\mathbf{Pr}[\mathcal{E}|\mathcal{E}_i] > 1 - 2$, and so:

$$\mathbf{E}\left[\sum_{i=1}^{k} |\alpha_i|\|p_i\|_1 \mid \mathcal{E}\right] \leq C\log(1/\delta)\sum_{i=1}^{k}\|p_i\|_1$$

Setting $\delta < \frac{1}{2tk}$, by Markov's inequality.

$$\mathbf{Pr}\left[\sum_{i=1}^{k} |\langle Z, X_i\rangle| \geq 2C\log(2tk)t \cdot \sum_{i=1}^{k} \|X_i\|_1\right] \leq \frac{1}{2t} + \mathbf{Pr}[\mathcal{E}] < \frac{1}{t}$$

Which yields the proposition after rescaling C by constant. □

Lemma 8.6.8 *Fix $\eta = \Theta\left(\sqrt{\frac{\epsilon}{\log d \log^{\ell} s}}\right), \epsilon_0 = \Theta(1/\log s)$, and let (u^*, v^*) be the samples accepted by Figure 8.5. Then assuming that $\frac{2s}{\Delta_i} < \frac{\log s \log d}{\epsilon 2^i}$, and moreover that $\frac{|A_{u^*}|+|B_{u^*}|}{\Delta_i(u^*)} \leq \frac{\log s \log d}{\epsilon\nu 2^i}$, where $\nu = \Theta(1/\log s)$, then with probability $1 - 1/\mathrm{poly}(s)$, we have*

$$\frac{1}{\mathrm{median}(|\mathcal{C}|)} \cdot \mathrm{median}_{j\in[\rho]}\left|\frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}} - \frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}}\right| = (1\pm\epsilon_0)\|c_{u^*} - c_{v^*}\|_1 \pm \epsilon_1\frac{d}{2^i}$$

where ϵ_1 is an arbitrarily small constant.

Proof. First suppose that our output was actually $\text{median}_{j\in[\rho]}\left|\frac{\mathbf{Z}^1_{u^*,\rho+j+1}}{\mathbf{Z}^1_{u^*,\rho+1}}-\frac{\mathbf{Z}^2_{v^*,j+2}}{\mathbf{Z}^2_{v^*,2}}\right|$, and let us see that this would be a good approximation. Noticed that since each row u of $\mathbf{Z}^1$ is scaled by the same value $1/t_u$, and similarly with $\mathbf{Z}^2$, this is the same as the quantity $\text{median}_{j\in[\rho]}\left|\frac{\mathbf{X}_{u^*,\rho+j+1}}{\mathbf{X}_{u^*,\rho+1}}-\frac{\mathbf{Y}_{v^*,j+2}}{\mathbf{Y}_{v^*,2}}\right|$. Plugging in definitions, this is just

$$\begin{aligned}&=\text{median}_{j\in[\rho]}\left|\frac{\left(\Omega_u\left(\sum_{p\in A_u\cup B_u}p\right)\right)_j}{||A_u|+|B_u||}-\frac{\left(\Omega_u\left(\sum_{p\in A_u\cup B_u}p\right)\right)_j}{||A_v|+|B_v||}\right| \\ &=\text{median}_{j\in[\rho]}\left|\left(\Omega_u(c_u-c_v)\right)_j\right|\end{aligned} \tag{8.45}$$

By standard concentration for medians of p-stables ([Ind06], see Theorem 8), it follows that

$$\frac{\text{median}_{j\in[\rho]}\left|\left(\Omega_u(c_u-c_v)\right)_j\right|}{\text{median}(|\mathcal{D}_1|)}=(1\pm\epsilon_0)\|c_u-c_v\|_1 \tag{8.46}$$

with probability $1-s^{-c}$.

Claim 8.6.14. *To prove the Lemma, it suffices to prove that both*

$$\left|\frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}}-\frac{\mathbf{Z}^1_{u^*,\rho+j+1}}{\mathbf{Z}^1_{u^*,\rho+1}}\right|\le\epsilon_1\frac{d}{2^{i+2}},\qquad\left|\frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}}-\frac{\mathbf{Z}^2_{v^*,j+2}}{\mathbf{Z}^2_{v^*,2}}\right|\le\epsilon_1\frac{d}{2^{i+2}} \tag{8.47}$$

hold with probability at least $1-1/\zeta$ independently for each $j\in[\rho]$, where $\zeta=\Theta(1/\epsilon_0)$ with a large enough constant.

Proof. Let

$$\theta_j=\frac{\left|\left(\Omega_u(c_u-c_v)\right)_j\right|}{\text{median}(|\mathcal{C}|)},\quad\hat{\theta}_j=\frac{\left|\frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}}-\frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}}\right|}{\text{median}(|\mathcal{C}|)}.$$

By the assumption and triangle inequality, we have that $|\hat{\theta}_j-\theta_j|<\epsilon_2 d/2^{i+1}$, where $\epsilon_2=\frac{\epsilon_1}{\text{median}(|\mathcal{D}_1|)}$ is within a fixed constant of ϵ_1, with probability at least $1-\zeta^{-1}$. Since there are ρ repetitions, it follows by Chernoff bounds that there will be at most $2\rho/\zeta$ values of $j\in[\rho]$ such that this approximation does not hold, with probability at least $1-1/\text{poly}(s)$. Let $W\subset[\rho]$ be the subset of indices where this guarantee fails, where $|W|\le 2\rho/\zeta$. Let $\theta'_j=\theta_j$ for $j\notin W$, and let $\theta'_j=\hat{\theta}_j$ for $j\in W$. Then we have $|\hat{\theta}_j-\theta'_j|<\epsilon_2 d/2^{i+1}$ for all $j\in[\rho]$. It follows that $|\text{median}_{j\in[\rho]}\hat{\theta}_j-\text{median}_{j\in[\rho]}\theta'_j|\le\epsilon_2 d/2^i$. Thus, it will suffice to show that $\text{median}_{j\in[\rho]}\theta'_j$ is a good approximation of $\|c_u-c_v\|_1$.

To see this, first note by Lemma 2 of [Ind06], for any $\phi>0$ smaller than a constant,

if C is Cauchy we have that $\mathbf{Pr}\left[|C| < (1-\phi)\cdot \text{median}(|\mathcal{C}|)\right] < 1/2 - \phi/4$, and similarly $\mathbf{Pr}\left[|C| > (1+\phi)\cdot \text{median}(|\mathcal{D}_1|)\right] < 1/2 + \phi/4$. Then by Chernoff bounds, the number j for which θ_j is less than $(1-\epsilon_0)\|c_u - c_v\|_1$ is at most $(1-\epsilon_0/4)\rho/2$, and similarly the number of j for which this value is at least $(1+\epsilon_0)\|c_u - c_v\|_1$ is at most $(1-\epsilon_0/4)\rho/2$, both with probability $1 - s^{-c}$. Now $\theta'_1, \ldots, \theta'_\rho$ is the result of arbitrarily corrupting the value of an arbitrary subset of $2\rho/\zeta < \epsilon\rho/10$ of the values in $\theta_1, \ldots, \theta_\rho$. After any such corruption, it follows that there must still be at most $\rho/2 - \epsilon_0\rho/8 + \epsilon_0\rho/10 < \rho/2 - \epsilon_0\rho/40$ indices j with $\theta'_j < (1-\epsilon_0)\|c_u - c_v\|_1$, and at most $\rho/2 - \epsilon_0\rho/40$ indices j with $\theta'_j > (1+\epsilon_0)\|c_u - c_v\|_1$. It follows that $\text{median}_{j\in[\rho]}\theta'_j = (1\pm\epsilon_0)\|c_u - c_v\|_1$, thus $\text{median}_{j\in[\rho]}\hat{\theta}_j = (1\pm\epsilon_0)\|c_u - c_v\|_1 + \epsilon_2 d/2^i$, which is the desired result after scaling ϵ_2 down by a constant $1/\text{median}(|\mathcal{D}_1|)$.

□

We now prove that Equation 8.47 holds with probability at least $1 - 1/\zeta$.We show that it holds for each of the two terms with probability at least $1 - 1/(2/\zeta)$, and the result will then follow by a union bound. In what follows, we let C, C', C'' be sufficiently large constants. We begin with the first term. First, note that by the count-sketch guarantee, we have $|\tilde{\mathbf{Z}}^1_{u^*,\rho+1} - \mathbf{Z}^1_{u^*,\rho+1}| \le \eta\|(\mathbf{Z}_{*,\rho+1})^1_{-1/\eta^2}\|_2$. Applying Lemma 8.6.5, we have $\|(\mathbf{Z}_{*,\rho+1})_{-1/\eta^2}\|_2 \le C\eta\|(\mathbf{X}_{*,\rho+1})_{-1/(2\eta^2)}\|_1 \le 2C\eta s$, where the last inequality uses the fact that each $\mathbf{X}_{u,\rho+1} = |A_u| + |B_u|$ so $\|\mathbf{X}_{*,\rho+1}\|_1 = 2s$. Thus $|\tilde{\mathbf{Z}}^1_{u,\rho+1} - \mathbf{Z}^1_{u,\rho+1}| \le 2C\eta^2 s$. Moreover, by Equation 8.41 in the proof of Lemma 8.6.7, we have that $\Delta_i < \frac{10c\log s\Delta_i(u^*)}{t_{u^*}}$ with probability at least $1 - s^{-c}$, which recall followed from the observation that our error from count-sketch on the first column of $\mathbf{Z}^1$ was at most $\tilde{O}(\eta^2\Delta_i)$ and that the maximum value of $\Delta_i(u)/t_u$ for $u \in \mathcal{V}_{i-1}$ is distributed like Δ_i/E_1 where E_1 is an exponential random variable, and then we conditioned on the event $E_1 < 5c\log s$. Since $|A_{u^*}| + |B_{u^*}| \ge \Delta_i(u^*)$, it follows that $\mathbf{Z}^1_{u^*,\rho+1} > \Delta_i/(2c\log s)$. Using that $\frac{2s}{\Delta_i} < \frac{\log s\log d}{\epsilon 2^i}$, we have $2C\eta^2 s < \frac{\epsilon s}{\log d\log^\ell s} < \frac{\Delta_i}{2^{i+1}\log^{\ell-1} s}$, so

$$\tilde{\mathbf{Z}}^1_{u^*,\rho+1} = \frac{1}{t_{u^*}}(1 \pm \epsilon_0^2 2^{-i})\left(|A_{u^*}| + |B_{u^*}|\right)$$

Next, we consider $\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}$ for $j = 1, 2, \ldots, \rho$. Applying the same argument as above, we have $|\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1} - \mathbf{Z}^1_{u^*,\rho+j+1}| \le C\eta^2\|(\mathbf{X}_{*,\rho+1})_{-1/(2\eta^2)}\|_1$. Note by 1-stability that $\|(\mathbf{X}_{*,\rho+1})_{-1/(2\eta^2)}\|_1$ is distributed as $\sum_{u\in\mathcal{V}_{i-1}}\alpha_{u,j}\|\sum_{p\in A_u\cup B_u} p\|_1$ where $\alpha_{u,j}$ are i.i.d. Cauchy random variables. Thus we can apply Lemma 8.6.5 to obtain $\|(\mathbf{X}_{*,\rho+1})_{-1/(2\eta^2)}\|_1 \le C\log s(2sd)$ with probability $1 - 1/\text{poly}(s)$, where we used that that $\sum_{u\in\mathcal{V}_{i-1}}\|\sum_{p\in A_u\cup B_u} p\|_1 \le 2sd$. Taken together, we have

$$|\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1} - \mathbf{Z}^1_{u^*,\rho+j+1}| \le C'\eta^2 \log s \cdot (sd)$$

for all $j \in [\rho]$ with probability $1 - 1/\text{poly}(s)$ after a union bound. Now let $\mathcal{Q}^1_j$ denote the event that $|(\Omega_u c_u)_j| \le Cd\zeta$, where $\zeta = \Theta(1/\epsilon_0)$, (where C is taken as a large enough constant) Because $(\Omega_u c_u)_j$ is distributed as $\omega\|c_u\|_1$, where ω is a Cauchy random variable, and because $\|c_u\|_1 \le d$, we have $\mathbf{Pr}\left[\mathcal{Q}^1_j\right] > 1 - 1/(2\zeta)$ independently for separate $j \in [\rho]$. Conditioned on this, we have

$$\begin{aligned}
\frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{\tilde{\mathbf{Z}}^1_{u^*,\rho+1}} &= (1 \pm 2\epsilon_1 2^{-i}) t_{u^*} \cdot \frac{\tilde{\mathbf{Z}}^1_{u^*,\rho+j+1}}{|A_{u^*}| + |B_{u^*}|} \\
&= (1 \pm 2\epsilon_0^2 2^{-i}) t_{u^*} \cdot \frac{\mathbf{Z}^1_{u^*,\rho+j+1} \pm C'\eta^2 \log s \cdot (sd)}{|A_{u^*}| + |B_{u^*}|} \\
&= (1 \pm 2\epsilon_0^2 2^{-i})(\Omega_u c_u)_j \pm 2t_{u^*} \frac{C'\eta^2 \log s \cdot (sd)}{|A_{u^*}| + |B_{u^*}|} \\
&= (\Omega_u c_u)_j \pm C'\left(\epsilon_0^2 2^{-i} |(\Omega_u c_u)_j| + \eta^2 \log^2 s \cdot \frac{sd}{\Delta_i}\right) \\
&= (\Omega_u c_u)_j \pm \left(\epsilon_1 \frac{d}{2^{i+2}} + C'\eta^2 \log^3 s \cdot \frac{d \log d}{\epsilon 2^i}\right) \\
&= (\Omega_u c_u)_j \pm \epsilon_1 \frac{d}{2^{i+1}}
\end{aligned} \tag{8.48}$$

Where we used the the fact that $\frac{|A_{u^*}|+|B_{u^*}|}{t_{u^*}} \ge \frac{\Delta_i(u^*)}{t_{u^*}} \ge \Delta_i/(2c \log s)$, and the assumptions $\frac{2s}{\Delta_i} < \frac{\log s \log d}{\epsilon 2^i}$ of the Lemma. Noting that $(\Omega_{u^*} c_{u^*})_j = \frac{\mathbf{Z}^1_{u^*,\rho+j+1}}{\mathbf{Z}^1_{u^*,\rho+1}}$, we have proven the first inequality with the desired probability.

To prove the second inequality from Claim 8.6.14, we first analyze $|\tilde{\mathbf{Z}}^2_{v^*,2} - \mathbf{Z}^2_{v^*,2}|$. Let $U_0 \subset \mathcal{V}_{i-1}$ be the top $1/\epsilon_0$ coordinates in magnitude of the vector $\mathbf{Z}^1_{*,\rho+1}$, which recall has coordinates $\mathbf{Z}^1_{u,\rho+1} = \sum_{v \in \mathcal{V}_i} \frac{1}{t_{\pi(v)}}(|A_v| + |B_v|)$. Applying the stronger nested count-sketch guarantee of Theorem 105, we have:

$$|\tilde{\mathbf{Z}}^2_{v^*,2} - \mathbf{Z}^2_{v^*,2}| \le \|(\mathbf{Z}^2_{*,2})_{-(U_0 \setminus u^*, \eta^{-2})}\|_2$$

with probability $1 - s^{-c}$, where recall $(\mathbf{Z}^2_{*,2})_{-(U_0 \setminus u^*, \eta^{-2})}$ is the notation from Definition 8.6.6, and is defined as the result of first zero-ing out all rows v of $\mathbf{Z}^2_{*,2}$ with parents $\pi(v) \in U_0 \setminus u^*$, and then removing the top η^{-2} largest of the remaining coordinates. We can now apply Lemma 8.6.5

on the exponential scalings t_v for the children to obtain

$$\begin{aligned}\|(\mathbf{Z}^2_{*,2})_{-(U_0\setminus u^*,\eta^{-2})}\|_2 &\leq C\eta\left(\sum_{u\in(\mathcal{V}_{i-1}\setminus U_0)\cup u^*}\frac{1}{t_u}\sum_{v:\pi(v)=u}|A_v|+|B_v|\right)\\ &= C\eta\left(\|\left(\mathbf{Z}^1_{*,\rho+1}\right)_{-1/\epsilon_0}\|_1+\frac{1}{t_{u^*}}(|A_{u^*}|+|B_{u^*}|)\right)\end{aligned} \tag{8.49}$$

We can apply then Lemma 8.6.5 to obtain $\|\left(\mathbf{Z}^1_{*,\rho+1}\right)_{-1/\epsilon_0}\|_1 \leq C\log s\|\mathbf{X}^1_{*,\rho+1}\|_1 = (2C\log s)\cdot s$ with probability $1-s^{-c}$, where we used that $\|\mathbf{X}^1_{*,\rho+1}\|_1$ is simply the number of points in $A\cup B$. This yields

$$\begin{aligned}\left|\tilde{\mathbf{Z}}^2_{v^*,2}-\mathbf{Z}^2_{v^*,2}\right| &\leq C\eta^2\left(\log s\cdot s+\frac{1}{t_{u^*}}(|A_{u^*}|+|B_{u^*}|)\right)\\ &\leq C\eta^2\left(\cdot\frac{\Delta_i\log^2 s\log d}{\epsilon 2^i}+\frac{1}{t_{u^*}}\cdot\frac{\Delta_i(u^*)\log s\log d}{\epsilon\nu 2^i}\right)\\ &\leq C'\eta^2\left(\frac{c}{t_{u^*}}\cdot\frac{\Delta_i(u^*)\log^3 s\log d}{\epsilon\nu 2^i}\right)\\ &\leq \epsilon_0^2\left(\frac{\Delta_i(u^*)}{t_{u^*}2^i}\cdot\right)\end{aligned} \tag{8.50}$$

Where in the second inequality we used the two assumptions $\frac{2s}{\Delta_i} < \frac{\log s\log d}{\epsilon 2^i}$ and $\frac{|A_{u^*}|+|B_{u^*}|}{\Delta_i(u^*)} \leq \frac{\log s\log d}{\epsilon\nu 2^i}$ of the Lemma, and in the third inequality we again used the earlier fact from Equation 8.41 in the proof of Lemma 8.6.7 that $\Delta_i < \frac{10c\log s\Delta_i(u^*)}{t_{u^*}}$ with probability at least $1-s^{-c}$. Similarly, we know that $\frac{||A_{v^*}|-|B_{v^*}||}{t_{u^*}t_{v^*}} \geq \frac{\Delta_i(u^*)}{2ct_{u^*}\log s}$ with probability $1-s^{-c}$, applying the lower bound on the top scaled coordinate as in the proof of the last Lemma. Thus $\mathbf{Z}^2_{v^*,2} = \frac{||A_{v^*}|+|B_{v^*}||}{t_{u^*}t_{v^*}} \geq \frac{||A_{v^*}|-|B_{v^*}||}{t_{u^*}t_{v^*}} \geq \frac{\Delta_i(u^*)}{2ct_{u^*}\log s}$, so using our bound on the error and $\epsilon_0 = \Theta(1/\log s)$, we have

$$\tilde{\mathbf{Z}}^2_{v^*,2} = \frac{1}{t_{u^*}t_{v*}}(1\pm\epsilon_0 2^{-i})\left(|A_{v^*}|+|B_{v^*}|\right)$$

Next, we must consider the estimates $\tilde{\mathbf{Z}}^2_{v^*,j+2}$. Let $\mathbf{M}^j \in \mathbb{R}^{|\mathcal{V}_{i-1}|}$ be the vector given by $\mathbf{M}^j_u = \frac{1}{t_u}\sum_{v\,:\,\pi(v)=u}|\mathbf{Y}_{v,j+2}|$ for $u\in\mathcal{V}_{i-1}$ and $j\in[\rho]$. By Lemma 8.6.13, we can write

$$\mathbf{M}^j_u = \alpha_u\sum_{v\,:\,\pi(v)=u}C\left\|\sum_{p\in A_v\cup B_v}p\right\|_1$$

where C is a constant, $\alpha_u = \frac{1}{t_u}\beta_u$, and $\{\beta_u\}$ are positive independent (non-identical) variables

with tails $\mathbf{Pr}[\beta_i > t] \leq \frac{\log(ts)}{t}$. Now define $U \subset \mathcal{V}_{i-1}$ to be the set of $1/\epsilon_0$ largest coordinates u of $\mathbf{M}$. We can then apply Theorem 105 on U. This yields

$$|\tilde{\mathbf{Z}}^2_{v^*,j+2} - \mathbf{Z}^2_{v^*,j+2}| \leq \eta \|(\mathbf{D}^2\mathbf{Y}_{*,j+2})_{-(U\setminus u^*,\eta^{-2})}\|_2$$

for any $j \in [\rho]$ with high probability. Recall that $(\mathbf{D}^2\mathbf{Y}_{*,j+2})_{-(U\setminus u^*,\eta^{-2})}$ first zeros out all rows corresponding to children of $u \in U \setminus \{u^*\}$, and then removes the top $1/\eta^2$ remaining coordinates. Using this fact, we can apply Lemma 8.6.5, to obtain

$$|\tilde{\mathbf{Z}}^2_{v^*,j+2} - \mathbf{Z}^2_{v^*,j+2}| \leq C\eta^2 \|(\mathbf{D}^2\mathbf{Y}_{*,j+2})_{-(U\setminus u^*,0)}\|_1 = C\eta^2 \left(\|\mathbf{M}^j_{-1/\epsilon_0}\|_1 + |\mathbf{M}^j_{u^*}|\right)$$

Now notice that $\mathbf{M}^j$ has coordinates $\mathbf{M}^j_u = \alpha_u \sum_{v\,:\,\pi(v)=u} C\|\sum_{p\in A_v\cup B_v} p\|_1$, where α_u are independent, so we can apply Proposition 8.6.12 to the at most $2s$ non-zero rows of $\mathbf{M}^j$ using the function $f(t) \leq \log(ts)$, to obtain

$$\|\mathbf{M}^j_{-1/\epsilon_0}\|_1 \leq C\log^3(s)\left(\sum_{v\in\mathcal{V}_i} \|\sum_{p\in A_v\cup B_v} p\|_1\right) \leq C_0\log^3(s)sd$$

for some constant C_0 with probability $1 - s^{-c}$. Now let $\mathcal{Q}^2_j$ be the event that, for the random variables β defined by the vector $\mathbf{M}^j$, we have $\beta_{u^*} < C\zeta\log(s\zeta)$ for some large enough constant C, where recall $\zeta = \Theta(1/\epsilon_0)$ is chosen with a large enough constant as earlier. By Lemma 8.6.13, we have $\mathbf{Pr}\left[\mathcal{Q}^2_j\right] < 1/(4\zeta)$, and note that $\mathcal{Q}^2_j$ is only a function in the randomness of the j-th rows of the sketches Ω_u for $u \in \mathcal{V}_{i-1}$. In particular, the events $\mathcal{Q}^2_j$ are independent for separate $j \in [\rho]$. Now let $\mathcal{Q}^3_j$ be the event that $|(\Omega_u c_v)_j| \leq Cd\zeta$, which, as in the earlier case of $\mathcal{Q}^1_j$, holds with probability at least $1 - 1/(4\zeta)$. Letting $\mathcal{Q}^4_j = \mathcal{Q}^2_j \cup \mathcal{Q}^3_j$, we have $\mathbf{Pr}\left[\mathcal{Q}^4_j\right] > 1 - 1/(2\zeta)$ by a union bound. The event $\mathcal{Q}^4_j$ will then be the desired event, depending only on the j-th row of the sketch Ω_u, which holds with probability at least $1 - 1/(2\zeta)$. Conditioned on $\mathcal{Q}^4_j$., we have

$|\mathbf{M}^j_{u^*}| \leq \frac{1}{t_{u^*}}\zeta 2d||A_{u^*}| + |B_{u^*}||$, and:

$$\begin{aligned}
\frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{\tilde{\mathbf{Z}}^2_{v^*,2}} &= (1 \pm 2\epsilon_0 2^{-i}) t_{u^*} t_{v^*} \cdot \frac{\tilde{\mathbf{Z}}^2_{v^*,j+2}}{|A_{v^*}| + |B_{v^*}|} \\
&= (1 \pm 2\epsilon_0 2^{-i}) \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm t_{u^*} t_{v^*} \eta^2 C' \frac{(\log^3 s)s + \frac{1}{t_{u^*}}\zeta 2d||A_{u^*}| + |B_{u^*}||}{|A_{v^*}| + |B_{v^*}|} \\
&= (1 \pm 2\epsilon_0 2^{-i}) \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm \eta^2 C' \left(t_{u^*} t_{v^*} \frac{(\log^3 s)s}{|A_{v^*}| + |B_{v^*}|} + \frac{t_{v^*}\zeta 2d||A_{u^*}| + |B_{u^*}||}{|A_{v^*}| + |B_{v^*}|} \right) \\
&= (1 \pm 2\epsilon_0 2^{-i}) \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm \eta^2 C'' \left(\frac{(\log^5 s)s}{\Delta_i} + \frac{(\log s)\zeta d||A_{u^*}| + |B_{u^*}||}{\Delta_i(u^*)} \right) \\
&= (1 \pm 2\epsilon_0 2^{-i}) \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm \eta^2 C'' \left(\frac{\log^6 s \cdot (\log d)d}{\epsilon 2^i} + \frac{\zeta(\log s)^2(\log d) \cdot d}{\epsilon\nu 2^i} \right) \\
&= \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm \left(2\epsilon_0 2^{-i} |(\Omega_u c_v)_j| + \epsilon_1 \frac{d}{2^{i+2}} \right) \\
&= \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm \left(\epsilon_1 \frac{d}{2^{i+2}} + \epsilon_1 \frac{d}{2^{i+2}} \right) \\
&= \frac{\mathbf{Z}^2_{v*,\rho+j+1}}{\mathbf{Z}^2_{v^*,2}} \pm \epsilon_1 \frac{d}{2^{i+1}}
\end{aligned} \tag{8.51}$$

with probability $1 - s^{-c}$, where C', C'' are constants, and where we used several facts including the bound on η. Chiefly, we used that $\frac{t_{v^*}}{|A_{v^*}|+|B_{v^*}|} \leq \frac{t_{v^*}}{||A_{v^*}|-|B_{v^*}||} \leq O(\frac{\log s}{\Delta_i(u^*)})$ with probability $1 - s^{-c}$, using that the maximum value of $\frac{t_v}{||A_v|-|B_v||}$ over children v of u^* will be at least this large, and the error from count-sketch in finding v^* is more than a constant factor smaller than this value (as in the proof of the last lemma, and used earlier in this lemma). The same fact is be applied again to show $\frac{t_{u^*}}{\Delta_i(u^*)} < O(\frac{\log s}{\Delta_i})$ with the same probability. We also used the assumptions that $\frac{2s}{\Delta_i} < \frac{\log s \log d}{\epsilon 2^i}$, and moreover that $\frac{|A_{u^*}|+|B_{u^*}|}{\Delta_i(u^*)} \leq \frac{\log s \log d}{\epsilon\nu 2^i}$. Finally, we used the bound on η^2 by definition, setting $\ell = 6$ as the exponential of $\log(s)$ in η^2. Thus, we have demonstrates Equation 8.47 holds with probability at least $1 - 1/\zeta$ independently for each $j \in [\rho]$, which completes the proof of the lemma. □

8.7 Analysis of COMPUTEEMD via Tree Embeddings

We sketch how the natural analysis of COMPUTEEMD(A, B, d) yields a $O(\min\{\log s, \log d\} \log s)$-approximation. The analysis proceeds via the method of randomized tree embeddings, and im-

mediately gives a (randomized) embedding $\boldsymbol{f}\colon (\{0,1\}^d)^s \to \ell_1$ satisfying

$$\mathrm{EMD}(A,B) \le \|\boldsymbol{f}(A) - \boldsymbol{f}(B)\|_1 \le O(\min\{\log s, \log d\} \log s)\,\mathrm{EMD}(A,B),$$

with probability 0.9 over the draw of $\boldsymbol{f}$. Specifically, let $A, B \subset \{0,1\}^d$ be two multi-sets of size s, and let $M^* \subset A \times B$ be the matching such that

$$\mathrm{EMD}(A,B) = \sum_{(a,b)\in M^*} \|a-b\|_1.$$

Consider the execution tree $\mathbf{T}_0$ in described the beginning of Section 8.4, where we execute the algorithm for at most $O(d)$ rounds of recursion. We assign weights to the edges, where an edge connecting a node at depth i and $i+1$ is given weight $d/(i+1)^2$, which defines a tree metric $(A \cup B, d_{\mathbf{T}_0})$ given by the sum of weights over the paths connecting two points in the tree.

The following claim is a simple observation, which follows from a greedy construction of the matching over a tree. See Figure 8.6.

Claim 8.7.1 (Greedy Bottom-Up Approach is Optimal for a Tree)**.** *Let* $\mathbf{M} \subset A \times B$ *be the matching that the execution tree* $\mathbf{T}_0$ *outputs, then*

$$\sum_{(a,b)\in\mathbf{M}} d_{\mathbf{T}_0}(a,b) \le \sum_{(a,b)\in M^*} d_{\mathbf{T}_0}(a,b).$$

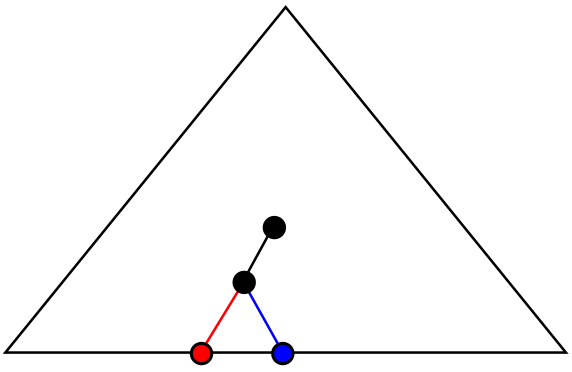


Figure 8.6: Proof sketch of Claim 8.7.1. The matching $\mathbf{M}$ satisfies that for any node v in the tree, the pairs $(a,b) \in \mathbf{M}$ within the subtree rooted at v forms a maximal matching of nodes in A and B within the subtree rooted at v. In order to see why this is optimal for a tree with positive edge weights, suppose the red point is in A and the blue point in B. These meet at the (lower) black node v, but if they both remain unmatched at the upper-black node u, then both must pay the black edge.

Lemma 8.7.2. *With probability* 0.9 *over the draw of* $\mathbf{T}_0$*,*

$$\sum_{(a,b)\in M^*} d_{\mathbf{T}_0}(a,b) \le O(\min\{\log s, \log d\}) \cdot \mathrm{EMD}(A,B),$$

and every $a \in A$ *and* $b \in B$ *satisfies* $d_{\mathbf{T}_0}(a,b) \ge \Omega\left(\|a-b\|_1 / \log s\right)$*.*

Proof. For $(a,b) \in A \times B$ with $a \neq b$, let

$$i_{\min}(a,b) = \left\lfloor \frac{d}{\|a-b\|_1 \cdot s^3} \right\rfloor \qquad i_{\max}(a,b) = \max\left\{ \left\lceil \frac{10 d \log s}{\|a-b\|_1} \right\rceil, d \right\}.$$

Then, we consider the random variable

$$\mathbf{D}(a,b) \stackrel{\text{def}}{=} 2 \sum_{i=i_{\min}(a,b)}^{i_{\max}(a,b)} \mathbf{1}\left\{(a,b) \text{ first split in } \mathbf{T}_0 \text{ at depth } i\right\} \sum_{j \ge i} \frac{d}{(i+1)^2} \cdot,$$

and notice that this is equal to $d_{\mathbf{T}_0}(a,b)$ whenever (a,b) are first split between depth $i_{\min}(a,b)$ and $i_{\max}(a,b)$. Then, we have

$$\begin{aligned}
\mathop{\mathbf{E}}_{\mathbf{T}_0}\left[\mathbf{D}(a,b)\right] &\le 2 \sum_{i=i_{\min}(a,b)}^{i_{\max}(a,b)} \Pr_{\mathbf{T}_0}\left[(a,b) \text{ first split at depth } i\right] \sum_{j \ge i} \frac{d}{(j+1)^2} \lesssim \sum_{i=i_{\min}(a,b)}^{i_{\max}(a,b)} \frac{\|a-b\|_1}{d} \cdot \frac{d}{i+1} \\
&= \|a-b\|_1 \cdot O\left(\log\left(\frac{i_{\max}(a,b)}{i_{\min}(a,b)+1}\right)\right) = \|a-b\|_1 \cdot O\left(\min\{\log s, \log d\}\right).
\end{aligned}$$

Furthermore, the probability that there exists some $(a,b) \in A \times B$ such that (a,b) are not split between levels $i_{\min}(a,b)$ and $i_{\max}(a,b)$ is at most

$$\begin{aligned}
&\sum_{a \in A} \sum_{\substack{b \in B \\ a \neq b}} \Pr_{\mathbf{T}_0}\left[(a,b) \text{ first split outside depths } i_{\min}(a,b) \text{ and } i_{\max}(a,b)\right] \\
&\qquad \le \sum_{a \in A} \sum_{\substack{b \in B \\ a \neq b}} \left(\frac{i_{\min}(a,b) \cdot \|a-b\|_1}{d s^3} + \left(1 - \frac{\|a-b\|_1}{d}\right)^{i_{\max}(a,b)} \right) \le \frac{2}{s}.
\end{aligned}$$

Hence, with probability $1 - 2/s$, every $a \in A$ and $b \in B$, with $a \neq b$, satisfies $d_{\mathbf{T}_0}(a,b) \gtrsim d / i_{\max}(a,b) = \Omega(\|a-b\|_1 / \log s)$, and

$$\sum_{(a,b)\in M^*} d_{\mathbf{T}_0}(a,b) = \sum_{(a,b)\in M^*} \mathbf{D}(a,b).$$

By Markov's inequality,

$$\sum_{(a,b)\in M^*} \mathbf{D}(a,b) \le 100 \cdot O(\min\{\log s, \log d\}) \sum_{(a,b)\in M^*} \|a-b\|_1 = O(\min\{\log s, \log d\})\,\mathrm{EMD}(A,B),$$

with probability $99/100$, so that with probability $99/100 - 2/s$, we obtain the desired lemma. □

In order to see that Claim 8.7.1 and Lemma 8.7.2, notice that with probability 0.9, we have the following string of inequalities:

$$\begin{aligned}\sum_{(a,b)\in \mathbf{M}} \|a-b\|_1 &\lesssim \log s \sum_{(a,b)\in \mathbf{M}} d_{\mathbf{T}_0}(a,b) \le \log s \sum_{(a,b)\in M^*} d_{\mathbf{T}_0}(a,b)\\ &\lesssim O(\min\{\log s, \log d\}\log s)\,\mathrm{EMD}(A,B),\end{aligned}$$

where the first and last inequality follow from Lemma 8.7.2, and the middle inequality is Claim 8.7.1.

8.8 Tightness of COMPUTEEMD

We show that we cannot improve the approximation factor of COMPUTEEMD beyond $O(\log s)$.

Lemma 8.8.1. *Fix $s, d \in \mathbb{N}$. There exists a distribution over inputs $\mathbf{A}, \mathbf{B} \subset \{0,1\}^d$ of size s such that with probability at least 0.9 over the draw of $\mathbf{A}, \mathbf{B}$, and an execution of $\textsc{ComputeEMD}(\mathbf{A}, \mathbf{B})$ which outputs the matching $\mathbf{M}$,*

$$\sum_{(a,b)\in \mathbf{M}} \|a-b\|_1 \ge \Omega(\log s) \cdot \mathrm{EMD}(\mathbf{A}, \mathbf{B}).$$

For $s, d \in \mathbb{N}$ and $\alpha \in (0, 1/2)$, we let $\mathcal{D}_{s,d}(\alpha)$ be the distribution over pairs of subsets $(\mathbf{A}, \mathbf{B})$ with $\mathbf{A}, \mathbf{B} \subset \{0,1\}^d$ of $|\mathbf{A}| = |\mathbf{B}| = s$ given by the following procedure (think of α as being set to $1/\log s$):

- For $t = 1, \ldots, s$, we generate the pair $(\boldsymbol{a}^{(t)}, \boldsymbol{b}^{(t)})$ where $\boldsymbol{a}^{(t)}, \boldsymbol{b}^{(t)} \in \{0,1\}^d$ are sampled by letting, for each $i \in [d]$,

$$\boldsymbol{a}_i^{(t)} \sim \{0,1\} \qquad \text{and} \qquad \boldsymbol{b}_i^{(t)} \sim \begin{cases} \boldsymbol{a}_i^{(t)} & \text{w.p } 1-\alpha \\ 1-\boldsymbol{a}_i^{(t)} & \text{w.p } \alpha \end{cases}. \tag{8.52}$$

- We let $\mathbf{A} = \left\{\boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(s)}\right\}$ and $\mathbf{B} = \left\{\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(s)}\right\}$.

Notice that for $\alpha \in (0, 1/2)$, we have

$$\mathop{\mathbf{E}}_{(\mathbf{A},\mathbf{B})\sim\mathcal{D}_{s,d}(\alpha)}\left[\mathrm{EMD}(\mathbf{A},\mathbf{B})\right] \leq \sum_{t=1}^{s} \mathop{\mathbf{E}}_{(\boldsymbol{a}^{(t)},\boldsymbol{b}^{(t)})}\left[\left\|\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}\right\|_1\right] = sd\alpha,$$

and by Markov's inequality, $\mathrm{EMD}(\mathbf{A},\mathbf{B}) \leq 100sd\alpha$ with probability at least 0.99.

On the other hand, let $\mathbf{T}$ be the (random) binary tree of depth h naturally produced by the execution of $\textsc{ComputeEMD}(\mathbf{A},\mathbf{B})$, and let $\mathbf{M}$ be the matching between $\mathbf{A}$ and $\mathbf{B}$ that $\textsc{ComputeEMD}(\mathbf{A},\mathbf{B})$ outputs. Fix $t \in [s]$, and consider the probability, over the randomness in $\mathbf{A},\mathbf{B}$ and the execution of $\textsc{ComputeEMD}(\mathbf{A},\mathbf{B})$ that $\boldsymbol{a}^{(t)}$ is *not* matched to $\boldsymbol{b}^{(t)}$. Notice that this occurs whenever the following event $\boldsymbol{\mathcal{E}}_t$ occurs: there exists a depth $j \in \{0, \ldots, h-1\}$ such that

- At depth j, the two points $\boldsymbol{a}^{(t)}$ and $\boldsymbol{b}^{(t)}$ are split in the recursion, which occurs whenever a node $\mathbf{v}$ of $\mathbf{T}$ at depth j, corresponding to an execution of $\textsc{ComputeEMD}(\mathbf{A}^{(\mathbf{v})},\mathbf{B}^{(\mathbf{v})})$ with $\boldsymbol{a}^{(t)} \in \mathbf{A}^{(\mathbf{v})}$ and $\boldsymbol{b}^{(t)} \in \mathbf{B}^{(\mathbf{v})}$ samples a coordinate $\boldsymbol{i} \sim [d]$ where $\boldsymbol{a}_{\boldsymbol{i}}^{(t)} = 1$ and $\boldsymbol{b}_{\boldsymbol{i}}^{(t)} = 0$.
- Furthermore, considering the subsets which are split

$$\mathbf{A}_0^{(\mathbf{v})} = \left\{a \in \mathbf{A}^{(\mathbf{v})} : a_{\boldsymbol{i}} = 0\right\} \qquad \mathbf{A}_1^{(\mathbf{v})} = \left\{a \in \mathbf{A}^{(\mathbf{v})} : a_{\boldsymbol{i}} = 1\right\}$$
$$\mathbf{B}_0^{(\mathbf{v})} = \left\{b \in \mathbf{B}^{(\mathbf{v})} : b_{\boldsymbol{i}} = 0\right\} \qquad \mathbf{B}_1^{(\mathbf{v})} = \left\{b \in \mathbf{B}^{(\mathbf{v})} : b_{\boldsymbol{i}} = 1\right\},$$

  we happen to have $|\mathbf{B}_1^{(\mathbf{v})}| \geq |\mathbf{A}_1^{(\mathbf{v})}|$.

In order to see why this forces $\mathbf{M}(\boldsymbol{a}^{(t)}) \neq \boldsymbol{b}^{(t)}$, notice that $\boldsymbol{a}^{(t)} \in \mathbf{A}_1^{(\mathbf{v})}$ by definition that $\boldsymbol{i}$ satisfies $\boldsymbol{a}_{\boldsymbol{i}}^{(t)} = 1$, yet $\boldsymbol{b}_{\boldsymbol{i}}^{(t)} = 0$, so that $\boldsymbol{b}^{(t)} \notin \mathbf{B}_1^{(\mathbf{v})}$. Notice that since $\textsc{ComputeEMD}(\mathbf{A}_1^{(\mathbf{v})},\mathbf{B}_1^{(\mathbf{v})})$ returns a maximal matching $\mathbf{M}^{(\mathbf{v},1)}$ between $\mathbf{A}_1^{(\mathbf{v})}$ and $\mathbf{B}_1^{(\mathbf{v})}$. Since $|\mathbf{B}_1^{(\mathbf{v})}| \geq |\mathbf{A}_1^{(\mathbf{v})}|$, $\boldsymbol{a}^{(t)}$ participates in the matching $\mathbf{M}^{(\mathbf{v},1)}$, and hence is not matched with $\boldsymbol{b}^{(t)}$. In order to lower bound this probability, consider the following sampling process:

1. We first sample the pair of points $(\boldsymbol{a}^{(t)}, \boldsymbol{b}^{(t)})$ according to (8.52).
2. We then sample the tree $\mathbf{T}$.
3. We sample $\ell \in [s-1]$ pairs of points $(\boldsymbol{a}^{(\ell)}, \boldsymbol{b}^{(\ell)})$ similarly to (8.52).

Consider a fixed $(\boldsymbol{a}^{(t)}, \boldsymbol{b}^{(t)})$, as well as a fixed sequence of coordinates which are sampled

$\boldsymbol{i}_1, \ldots, \boldsymbol{i}_j \in [d]$ such that

$$\boldsymbol{a}^{(t)}_{\boldsymbol{i}_k} = \boldsymbol{b}^{(t)}_{\boldsymbol{i}_k} \qquad \text{for all } k < j, \text{ and} \qquad \boldsymbol{a}^{(t)}_{\boldsymbol{i}_j} = 1, \boldsymbol{b}^{(t)}_{\boldsymbol{i}_j} = 0.$$

We have then

$$\mathop{\mathbf{Pr}}_{(\boldsymbol{a}^{(\ell)}\boldsymbol{b}^{(\ell)})}\left[\boldsymbol{b}^{(\ell)} \in \mathbf{B}_1^{(\mathbf{v})} \wedge \boldsymbol{a}^{(\ell)} \notin \mathbf{A}_1^{(\mathbf{v})}\right] = \frac{1}{2^j}\left(1 - (1-\alpha)^j\right),$$

and since $\boldsymbol{b}^{(\ell)}$ and $\boldsymbol{a}^{(\ell)}$ are symmetric,

$$\mathop{\mathbf{Pr}}_{(\boldsymbol{a}^{(\ell)},\boldsymbol{b}^{(\ell)})}\left[\boldsymbol{a}^{(\ell)} \in \mathbf{A}_1^{(\mathbf{v})} \wedge \boldsymbol{b}^{(\ell)} \notin \mathbf{B}_1^{(\mathbf{v})}\right] = \frac{1}{2^j}\left(1 - (1-\alpha)^j\right).$$

We note that event $\boldsymbol{\mathcal{E}}_t$ occurs if $(\boldsymbol{a}^{(t)}, \boldsymbol{b}^{(t)})$ are split at depth j at a coordinate $\boldsymbol{i}$ with $\boldsymbol{a}^{(t)}_{\boldsymbol{i}} = 1$, and when there exists a unique pair $(\boldsymbol{a}^{(\ell)}, \boldsymbol{b}^{(\ell)})$ which satisfy $\boldsymbol{b}^{(\ell)} \in \mathbf{B}_1^{(\mathbf{v})}$ and $\boldsymbol{a}^{(\ell)} \notin \mathbf{A}_1^{(\mathbf{v})}$. Hence,

$$\mathbf{Pr}\left[\boldsymbol{\mathcal{E}}_t\right] \geq \tag{8.53}$$
$$\mathop{\mathbf{E}}_{(\boldsymbol{a}^{(t)},\boldsymbol{b}^{(t)})}\left[\left(1 - \frac{\|\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}\|_1}{d}\right)^{j-1} \frac{\|\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}\|_1}{d}\left(\frac{(s-1)(1-(1-\alpha)^j)}{2^j}\right)\left(1 - \frac{(1-(1-\alpha)^j)}{2^{j-1}}\right)^{s-2}\right]$$

If we consider $j = \lfloor \log_2 s \rfloor$ and $\alpha = \frac{1}{\log_2 s}$, we have

$$\frac{s-1}{2^j}\left(1-(1-\alpha)^j\right)\left(1 - \frac{1}{2^{j-1}}\left(1-(1-\alpha)^j\right)\right)^{s-2} = \Omega(1),$$

so the probability in (8.53) is at least

$$\Omega(1) \cdot \mathop{\mathbf{E}}_{(\boldsymbol{a}^{(t)},\boldsymbol{b}^{(t)})}\left[\left(1 - \frac{\|\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}\|_1}{d}\right)^{j-1} \frac{\|\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}\|_1}{d}\right] = \Omega(1),$$

since $\|\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}\|_1$ is distributed as $\mathrm{Bin}(d, \alpha)$, and $\alpha = \frac{1}{\log_2 s}$, and $j = \lfloor \log_2 s \rfloor$. If we let

$$\boldsymbol{m} = \min_{t_1 \neq t_2} \|\boldsymbol{a}^{(t_1)} - \boldsymbol{b}^{(t_2)}\|_1,$$

then, we have that $\mathbf{M}$ is a matching of cost at least $\boldsymbol{m}$ times the number of pairs $t \in [s]$ where $\boldsymbol{\mathcal{E}}_t$ occurs, and each $\boldsymbol{\mathcal{E}}_t$ occurs with constant probability. By Markov's inequality, with probability 0.99 over the draw of $\mathbf{A}, \mathbf{B}$ and $\textsc{ComputeEMD}(\mathbf{A}, \mathbf{B})$, there are $\Omega(s)$ indices $t \in [s]$ where $\boldsymbol{\mathcal{E}}_t$ occurs. Furthermore, since for any $t_1 \neq t_2$, $\boldsymbol{a}^{(t_1)}$ and $\boldsymbol{b}^{(t_2)}$ are distributed as uniformly random

points, we have that with a Chernoff bound $\boldsymbol{m} \geq \Omega(d)$ with probability at least 0.99 whenever $d \geq c_0 \log s$ (for a large fixed constant c_0).

Putting everything together with a union bound, we have that with probability at least 0.97 over the draw of $\mathbf{A}, \mathbf{B}$ and $\text{COMPUTEEMD}(\mathbf{A}, \mathbf{B})$, $\text{EMD}(\mathbf{A}, \mathbf{B}) \leq 100sd/\log_2 s$, yet the matching output has cost at least $\Omega(sd)$.

8.9 Lower Bound for Quadtree Approximation via Tree Embeddings

In this section, we show that the analysis in Section 8.7 is tight. This, along with Theorem 94, demonstrates that the approximation given by evaluating the matching produced by Quadtree is strictly better in the original metric, rather than the tree metric, demonstrating the necessity of considering the former. Specifically, let $\mathbf{T}_0$ be the execution tree described the beginning of Section 8.4, where we execute the algorithm for at most $2d$ rounds of recursion. We assign weights to the edges, where an edge connecting a node at depth i and $i+1$ is given weight $d/(i+1)^2$. This defines a tree metric $(A \cup B, d_{\mathbf{T}_0})$ given by the sum of weights over the paths connecting two points in the tree. We remark that the following analysis applies in a straightforward way to the depth $O(\log d)$ *compressed* quadtree also described in Section 8.7, which is the same tree considered in [AIK08b].

Let $\mathbf{M}_T(A, B) \subset A \times B$ be any the greedy bottom-up matching. By Claim 8.7.1, we know that $\mathbf{M}_T$ is an optimal cost matching in the tree metric on T_0. Fix s, d larger than some constant, so that $d = s^{\Theta(1)}$ are polynomial related, and let $d^{.01} < \alpha < d^{.99}$ be a parameter which decides the cost of $\text{EMD}(A, B)$. We will define two distributions, $\mathcal{D}_1$, and $\mathcal{D}_2$, over pairs of multisets $A, B \subset \{0,1\}^d$ of size s, such that

$$\mathop{\mathbf{Pr}}_{(A,B)\sim\mathcal{D}_1,\mathbf{T}_0}[\text{cost}(\mathbf{M}_{\mathbf{T}_0}(A,B)) < \frac{c}{\log s} \cdot \text{EMD}(A,B)] > 99/100$$

and

$$\mathop{\mathbf{Pr}}_{(A,B)\sim\mathcal{D}_2,\mathbf{T}_0}[\text{cost}(\mathbf{M}_{\mathbf{T}_0}(A,B)) > c \log s \cdot \text{EMD}(A,B)] > 99/100$$

and finally

$$\Pr_{(A,B)\sim\mathcal{D}_1}[\mathrm{EMD}(A,B) = (1 \pm 1/3)\alpha s] > 1 - 1/s$$
$$\Pr_{(A,B)\sim\mathcal{D}_2}[\mathrm{EMD}(A,B) = \alpha s] > 1 - 1/s$$

for some fixed constants c, C'. This will demonstrate that the cost of the matching given by the embedding into the tree metric is a $\Omega(\log^2 s)$ approximation.

The First Distribution. We first describe $\mathcal{D}_1$. To draw $(A, B) \sim \mathcal{D}_1$, we set $d' = \alpha$, and pick $2s$ uniformly random points $a'_1, a'_2, \ldots, a'_s, b'_1, b'_2, \ldots, b'_s \sim \{0,1\}^{d'}$, and set $A = \{a_1, \ldots, a_s\}, B = \{b_1, \ldots, b_s\}$, where a_i is a'_i padded with 0's, and similarly for b_i.

The Second Distribution. We now describe the second distribution $\mathcal{D}_2$. To draw $(A, B) \sim \mathcal{D}_2$, we set $a_1, \ldots, a_s \sim \{0,1\}^d$ uniformly at random. Then, for each $i \in [s]$, we set b_i to be a uniformly random point at distance α from a_i. In other words, given a_i, the point b_i is obtained by selecting a random subset of α coordinates in $[d]$, and having b_i disagree with a_i on these coordinates (and agree elsewhere).

Proposition 8.9.1. *There exists a fixed constant $c > 0$ such that*

$$\Pr_{(A,B)\sim\mathcal{D}_1,\mathbf{T}_0}\left[\mathrm{cost}(\mathbf{M}_{\mathbf{T}_0}(A,B)) < \frac{c}{\log s}\,\mathrm{EMD}(A,B)\right] > 99/100$$

And moreover,

$$\frac{\alpha s}{3} < \mathrm{EMD}(A,B) \leq \frac{2\alpha s}{3}$$

with probability at least $1 - 1/s$ over the draw of $(A, B) \sim \mathcal{D}_1$.

Proof. Notice that for all a, b, we have $d'/3 \leq d_{\mathbf{T}_0}(a,b) \leq 2d'/3$ with probability $1 - 2^{-\Omega(d)}$, thus by a union bound this holds for all $s^2 = \mathrm{poly}(d)$ pairs with probability at least $1 - 1/s^2$. In this case, $\alpha s/3 < \mathrm{EMD}(A,B) \leq (2/3)\alpha s$ with probability $1 - 1/s$.

Now consider any fixed point $a \in A$. We compute the expected cost of matching a in $\mathbf{M}_{\mathbf{T}_0}(A,B)$. Specifically, let $\mathrm{cost}(a) = d_{\mathbf{T}_0}(a, b_a)$, where $(a, b_a) \in \mathbf{M}_{\mathbf{T}_0}$. To do this, we must first define how the matching $\mathbf{M}_{\mathbf{T}_0}(A,B)$ breaks ties. To do this, in the algorithm of Figure 8.1, we randomly choose the maximal partial matching between the unmatched points in a recursive call to COMPUTEEMD.

Now let $X = A \cup B$, and begin generating the randomness needed for the branch which handles the entire set of points in X. Namely, consider following down the entire set of points X down $\mathbf{T}_0$, until the first vertex r which splits X up into two non-empty subsets in its children. Namely, $A_r \cup B_r = X$, but at r an index $\boldsymbol{i} \in [d']$ is sampled which splits X. First note that we expect the depth of r to be $\ell = d/\alpha$, since $d'/d = 1/\sqrt{d}$. Moreover, with probability $199/200$ we have $d/(2000\alpha) < \ell < 2000d/\alpha$. Call this event $\mathcal{E}_0$, and condition on this now, which does not fix the affect randomness used in any of the subtree of r. Conditioned on $\mathcal{E}_0$, we have that $\mathrm{cost}(a)$ conditioned on $\mathcal{E}_0$ is at most $\sum_{i=\ell}^{2d} \frac{d}{i^2} = \Theta(d/\ell) = \Theta(\alpha)$, since all points are matched below depth ℓ.

Now consider the level $\ell_1 = \ell + (1/2)\log s(d/\alpha)$. Notice the the vertex v which contains the point a at depth ℓ_1 as a descendant of r in expectation samples between $(1/2)(1 - \frac{1}{1000})\log s$ and $(1/2)(1 + \frac{1}{1000})\log s$ **unique** coordinates $\boldsymbol{i} \in [\alpha]$ on the $(1/2)\log s(d/\alpha)$ steps between r and v (we have not fixed the randomness used to draw the coordinate a yet). By independence across samples, by Chernoff bounds we have that v sampled between $(1/3)\log s$ and $(2/3)\log s$ **unique** coordinates $\boldsymbol{i} \in [\alpha]$ with probability $1 - s^{-c}$ for some fixed constant $c = \Omega(1)$. Let $\mathcal{E}_a$ be the event that this occurs. Say that it samples exactly γ unique coordinates. Now $\mathbf{E}\left[|A_v \cup B_v|\right] = (2s)/2^{\gamma} \geq s^{1/3}$, where the randomness is taken over the choice of random points in A, B. By Chernoff bounds applied to both the size of $|A_v|, |B_v|$, we have

$$\mathbf{Pr}\left[||A_v| - |B_v|| > c_1 \log(s)\sqrt{s/2^{\gamma}} \geq\right] < 1/s^c$$

for a sufficiently large constant c_1. Call the event that the above does not occur $\mathcal{E}_1$, and condition on it now. Note that conditioned on $\mathcal{E}_1$, only a $c_1 \log s\sqrt{2^{\gamma}/s} < s^{-1/7}$ fraction of the points in v remain unmatched. Since the distribution of the path a point $x \in A_v \cup B_v$ takes in the subtree of v is identical to every other $x' \in A_v \cup B_v$ even conditioned on $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_a$, it follows that

$$\begin{aligned} \mathbf{E}\left[\mathrm{cost}(a) \mid \mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_a\right] &\leq O(\frac{d}{\ell_1}) + s^{-1/7} O(\frac{d}{\ell}) \\ &\leq O(\frac{\alpha}{\log s}) \end{aligned} \tag{8.54}$$

Thus

$$\begin{aligned}\mathbf{E}\left[\text{cost}(a) \mid \mathcal{E}_0\right] &\leq O(\frac{\alpha}{\log s}) + s^{-c}O(\alpha) \\ &\leq O(\frac{\alpha}{\log s})\end{aligned} \tag{8.55}$$

Then by Markov's inequality, using that $\alpha s/3 < \text{EMD}(A, B) \leq (2/3)\alpha s$ with probability $1 - 1/s$, we have

$$\mathbf{Pr}\left[\sum_{a \in A} \text{cost}(a) > \frac{c}{\log s}\,\text{EMD}(A, B) \mid \mathcal{E}_0\right] \leq 10^{-4}$$

By a union bound:

$$\begin{aligned}\mathop{\mathbf{Pr}}_{(A,B)\sim\mathcal{D}_1,\mathbf{T}_0}\left[\text{cost}(\mathbf{M}_{\mathbf{T}_0}(A, B)) < \frac{c}{\log s}\,\text{EMD}(A, B)\right] &> 1 - 10^{-4} - 1/200 - O(1/s) \\ &> 99/100\end{aligned} \tag{8.56}$$

which completes the proof.

□

Proposition 8.9.2. *There exists a fixed constant $c > 0$ such that*

$$\mathop{\mathbf{Pr}}_{(A,B)\sim\mathcal{D}_2,\mathbf{T}_0}\left[\text{cost}(\mathbf{M}_{\mathbf{T}_0}(A, B)) < c \log s \cdot \text{EMD}(A, B)\right] > 99/100$$

And moreover,

$$\text{EMD}(A, B) = \alpha \cdot s$$

with probability at least $1 - 1/s$ over the draw of $(A, B) \sim \mathcal{D}_2$.

Proof. Set $\beta = 20 \log s$. We first claim that the event $\mathcal{E}_0$, which states that every non-empty vertex $v \in \mathbf{T}_0$ at depth β satisfies either $A_v \cup B_v = \{a_i, b_i\}$, $A_v \cup B_v = \{a_i\}$, or $A_v \cup B_v = \{b_i\}$ for some $i \in [s]$, occurs with probability $1 - 2/s^2$. To see this, note that first for each $i \neq j$, $d(a_i, b_j), d(a_i, a_j), d(b_i, b_j)$ are each at least $d/3$ with probability $1 - 2^{-\Omega(d)}$, and we can the union bound over all s^2 pairs so that this holds for all $i \neq j$ with probability $1 - 1/s^2$. Given this, for any a_i to collide at depth β with either a_j or b_j when $i \neq j$, we would need to avoid the set of $d/3$ points where they do not agree. The probability that this occurs is $(2/3)^\beta < 1/s^4$, and we can union bound over all s^2 pairs again for this to hold with probability $1 - 1/s^2$. We condition on $\mathcal{E}_0$ now. First note that the probability that a_i is split from b_i at or before level β is at most

$\frac{3\beta}{d}$. Thus the expected number of $i \in [s]$ for which this occurs is $\frac{3s\beta}{d}$, and is at most $\frac{30^4 s\beta}{d}$ with probability $1 - 10^{-4}$. Call this latter event $\mathcal{E}_1$, and let $S \subset [s]$ be the set of points i for which a_i, b_i are together, with no other points, in their node at depth β. Conditioned on $\mathcal{E}_0$ and $\mathcal{E}_1$, we have $|S| > s - \frac{30^4 s\beta}{d}$

Now for $j = \log(\beta), \ldots, \log(d/\alpha)$, let $S_j \subset S$ be the set of $i \in [s]$ for which a_i and b_i split before level 2^j. We have that $\mathbf{E}[|S_j|] > \frac{s2^j\alpha}{10d}$. Notice that since each branch of the quadtree is independent, and all (a_i, b_i) are together in their own node at depth β when $i \in S$, we can apply Chernoff bounds to obtain $|S_j| > \frac{s2^j\alpha}{20d}$ with probability at least $1 - 1/s^2$ for every $j > \log(d/\alpha) - (1/10)\log s$, and we can then union bound over the set of such j. Note that for each $i \in S_j$, the point a_i pays a cost of at least $\Theta(\frac{d}{2^j})$, thus the total cost of all points in S_j is at least $\Omega(s\alpha)$, and summing over all $\log(d/\alpha) - (1/10)\log s < j < \log(d/\alpha)$, we obtain $\mathrm{cost}(\mathbf{M}_{\mathbf{T}_0}(A, B)) = \Omega(\log s\alpha)$.

We finally show that $\mathrm{EMD}(A, B) = \alpha \cdot s$ with probability at least $1 - 1/(2s)$. To see this, note that conditioned on $\mathcal{E}_0$, the optimal matching is (a_i, b_i). Since $d(a_i, b_i) = \alpha$, this completes the proof. □

8.10 Sampling with Meta-data

In this section, We demonstrate how the tools developed in Section 8.6 can be easily applied to obtain a linear sketching algorithm for the problem of *Sampling with Meta-Data*, which we now define.

Definition 8.10.1. *Fix any constant $c > 1$, and fix k, n with $k \leq n^c$, and let $\epsilon, \delta > 0$. Fix any $x \in \mathbb{R}^n$, and meta-data vectors $\lambda_1, \ldots, \lambda_n \in \mathbb{R}^k$, such that the coordinates of x and all λ_i are can be stored in $O(\log n)$ bits. In the sampling with meta-data problem, we must sample $i^* \sim [n]$ from the distribution*

$$\mathbf{Pr}[i^* = i] = (1 \pm \epsilon)|x_i|/\|x\|_1 \pm n^{-c}$$

for all $i \in [n]$. Conditioned on returning $i^* = i \in [n]$, the algorithm must then return an approximation $\hat{\lambda}_i$ to λ_i with probability $1 - \delta$.

Remark 107. A natural generalization of Definition 8.10.1 is to ℓ_p sampler for $p \in (0, 2]$, where we want to sample $i \in [n]$ proportional to $|x_i|^p$. We remark that the below precision sampling framework easily generalizes to solving ℓ_p sampling with meta-data, by replacing the scalings $1/t_i$ by $1/t_i^{1/p}$, and appropriately modifying Lemma 8.6.4 to have error $\epsilon\|x\|_p$ instead of $\epsilon\|x\|_1$.

One can then apply the ℓ_p variant of Lemma 8.6.5 from Proposition 3.3.5 from Chapter 3, and the remainder of the proof follows similarly as below.

We given a linear sketching algorithm for the problem in Definition 8.10.1. The algorithm is similar, and simpler, to the algorithm from Section 8.6. Specifically, given $x \in \mathbb{R}^n, \lambda_1, \ldots, \lambda_n \in \mathbb{R}^k$, we construct a matrix $\mathbf{X} \in \mathbb{R}^{n \times (k+1)}$, where for each $i \in [n]$ the row $\mathbf{X}_{i,*} = [x_i, \lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,k}]$. We then draw a random matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, where $\mathbf{D}_{i,i} = 1/t_i$ and $\{t_i\}_{i \in [n]}$ are i.i.d. exponential random variables. We then draw a random count-sketch matrix (Theorem 2.3.2) $\mathbf{S}$ and compute the sketch $\mathbf{SDX}$, and recover an estimate $\tilde{\mathbf{Z}}$ of $\mathbf{DX}$ from Count-sketch. To sample a coordinate, we output $i^* = \arg\max_i \tilde{\mathbf{Z}}_{i,1}$, and set our estimate $(\hat{\lambda}_{i^*})_j = t_{i^*} \tilde{\mathbf{Z}}_{i,j+1}$. This simple algorithm obtains the following guarantee.

Theorem 108. *Fix any $\epsilon > 0$ and constant $c > 1$. Then given $x \in \mathbb{R}^n$ and $\lambda_1, \ldots, \lambda_n \in \mathbb{R}^k$, as in Definition 8.10.1, the above linear sketching algorithm samples $i \in [n]$ with probability $(1 \pm \epsilon)\frac{|x_i|}{\|x\|_1} \pm n^{-c}$. Once i is sampled, it returns a value $\hat{\lambda}_i$ such that $\|\hat{\lambda}_i - \lambda_i\|_1 \leq \epsilon |x_i| \frac{\sum_{j \in [n]} \|\lambda_j\|_1}{\|x\|_1}$ with probability $1 - n^{-c}$. The total space required to store the sketch is $O(\frac{k}{\epsilon} \log^3(n))$ bits.*

Proof. Let $\mathbf{Z} = \mathbf{DX}$. We instantiate count-sketch with parameter $\eta = \Theta(\sqrt{\epsilon / \log(n)})$ with a small enough constant. By Theorem 2.3.2, for every $j \in [n]$ we have $|\tilde{\mathbf{Z}}_{i,1} - \mathbf{Z}_{i,1}| < \eta \|(\mathbf{Z}_{*,1})_{-1/\eta^2}\|_2$ with probability $1 - n^{-2c}$. By Lemma 8.6.5, we have $\|(\mathbf{Z}_{*,1})_{-1/\eta^2}\|_2 \leq \eta \|\mathbf{X}_{*,1}\|_1$ with probability $1 - n^{-2c}$, and thus $|\tilde{\mathbf{Z}}_{i,1} - \mathbf{Z}_{i,1}| < \eta^2 \|\mathbf{X}_{*,1}\|_1$, so by Lemma 8.6.4, the coordinate i^* satisfies

$$\mathbf{Pr}\left[i^* = i\right] = (1 \pm \epsilon)\frac{|x_i|}{\|x\|_1} \pm n^{-c}$$

for all $i \in [n]$, where we used the fact that the n^{-2c} failure probabilities of the above events can be absorbed into the additive n^{-c} error of the sampler. Now recall by Fact 8.6.3 that $\max_i \mathbf{Z}_{i,1} = \|\mathbf{X}_{*,1}\|_1 / E$, where E is an independent exponential random variable. With probability $1 - n^{-c}$, using the tails of exponential variables we have $\max_i \mathbf{Z}_{i,1} > \|\mathbf{X}_{*,1}\|_1 / (10c \log s)$, which we condition on now. Notice that given this, since our error satisfies $|\tilde{\mathbf{Z}}_{i,1} - \mathbf{Z}_{i,1}| < \eta^2 \|\mathbf{X}_{*,1}\|_1$, we have $\mathbf{Z}_{i^*,1} > \|\mathbf{X}_{*,1}\|_1 / (20c \log s)$, from which it follows by definition that $t_i < \frac{20c \log s |x_i|}{\|x\|_1}$.

Now consider any coordinate $j \in [k]$. By the count-sketch error, we have $|\tilde{\mathbf{Z}}_{i,j+1} - \mathbf{Z}_{i,j+1}| < \eta \|(\mathbf{Z}_{*,j+1})_{-1/\eta^2}\|_2$, and moreover by Lemma 8.6.5, we have $\|(\mathbf{Z}_{*,1})_{-1/\eta^2}\|_2 \leq \eta \|\mathbf{X}_{*,j+1}\|_1 = \eta \sum_{i \in [n]} |\lambda_{i,j}|$ with probability $1 - n^{-2c}$. Thus

$$
\begin{aligned}
|t_i\tilde{\mathbf{Z}}_{i,j+1} - t_i\mathbf{Z}_{i,j+1}| &< t_i\eta^2 \sum_{i\in[n]} |\lambda_{i,j}| \\
&\leq \eta^2 \frac{20c\log s|x_i|\sum_{i\in[n]}|\lambda_{i,j}|}{\|x\|_1} \\
&\leq \epsilon\frac{|x_i|\sum_{i\in[n]}|\lambda_{i,j}|}{\|x\|_1}
\end{aligned} \tag{8.57}
$$

Summing over all $j \in [k]$ (after a union bound over each $j \in [k]$) completes the proof of the error. For the space, note that $\mathbf{SDX}$ had $O(1/\eta^2 \log n) = O(\log^2 n/\epsilon)$ rows, each of which has $k+1$ columns. Note that we can truncating the exponential to be $O(\log n)$ bit discrete values, which introduces an additive n^{-10c} to each coordinate of $\mathbf{SDX}$ (using that the values of $\mathbf{X}$ are bounded by $n^c = \text{poly}(n)$), which can be absorbed into the additive error from count-sketch without increasing the total error by more than a constant. Moreover, assuming that the coordinates of x and all λ_i are integers bounded by $\text{poly}(n)$ in magnitude, since the entries of $\mathbf{S}$ are contained in $\{0, 1, -1\}$, each entry of $\mathbf{SDX}$ can be stored in $O(\log n)$ bits of space, which completes the proof. □

8.11 Embedding ℓ_p^d into $\{0,1\}^{d'}$

Lemma 8.11.1. *Let $p \in (1, 2]$. There exists a distribution $\mathcal{D}_p$ supported on $\mathbb{R}$ which exhibits p-stability: for any $d \in \mathbb{N}$ and any vector $x \in \mathbb{R}^d$, the random variables $\mathbf{Y}_1$ and $\mathbf{Y}_2$ given by*

$$
\begin{aligned}
\mathbf{Y}_1 &= \sum_{i=1}^{d} x_i \boldsymbol{z}_i \qquad \textit{for } \boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \sim \mathcal{D}_p \textit{ independently,} \\
\mathbf{Y}_2 &= \boldsymbol{z} \cdot \|x\|_p \qquad \textit{for } \boldsymbol{z} \sim \mathcal{D}_p,
\end{aligned}
$$

are equal in distribution, and $\mathbf{E}_{z\sim\mathcal{D}_p}[|\boldsymbol{z}|]$ is at most a constant C_p.

Consider fixed $R \in \mathbb{R}^{\geq 0}$ which is a power of 2, and for $p \in [1, 2]$, let $\boldsymbol{f} \colon \mathbb{R}^d \to \{0, 1\}$ be the randomized function given by sampling the following random variables

- $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d)$ where $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \sim \mathcal{D}_p$. In the case $p = 1$, we let $\mathcal{D}_1$ be the distribution where $\boldsymbol{z}_i \sim \mathcal{D}_1$ is set to 1 with probability $1/d$ and 0 otherwise.
- $\boldsymbol{h} \sim [0, R]$, and

- a uniformly random function $\boldsymbol{g}\colon \mathbb{Z} \to \{0,1\}$.

and we evaluate

$$\boldsymbol{f}(x) = \boldsymbol{g}\left(\left\lfloor \frac{\sum_{i=1}^{d} x_i \boldsymbol{z}_i - \boldsymbol{h}}{R} \right\rfloor\right) \in \{0,1\}.$$

If $x, y \in \mathbb{R}^d$ with $\|x-y\|_p \le R/t_p$ (for a parameter t_p which we specify later). When $p=1$, we consider $\|x-y\|_\infty \le R/t_1$. Then

$$\Pr_{\boldsymbol{z},\boldsymbol{h},\boldsymbol{g}}\left[\boldsymbol{f}(x) \ne \boldsymbol{f}(y)\right] = \frac{1}{2} \cdot \Pr_{\boldsymbol{z},\boldsymbol{h}}\left[\left\lfloor \frac{\sum_{i=1}^{d} x_i \boldsymbol{z}_i - \boldsymbol{h}}{R} \right\rfloor \ne \left\lfloor \frac{\sum_{i=1}^{d} y_i \boldsymbol{z}_i - \boldsymbol{h}}{R} \right\rfloor\right].$$

Consider a fixed $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d)$ and let

$$\boldsymbol{w} = \sum_{i=1}^{d} (x_i - y_i)\boldsymbol{z}_i.$$

Then, we have

$$\Pr_{\boldsymbol{h}\sim[0,R]}\left[\left\lfloor \frac{\sum_{i=1}^{d} x_i \boldsymbol{z}_i - \boldsymbol{h}}{R} \right\rfloor \ne \left\lfloor \frac{\sum_{i=1}^{d} y_i \boldsymbol{z}_i - \boldsymbol{h}}{R} \right\rfloor\right] = \max\left\{\frac{|\boldsymbol{w}|}{R}, 1\right\},$$

so that

$$\Pr_{\boldsymbol{z},\boldsymbol{h},\boldsymbol{g}}\left[\boldsymbol{f}(x) \ne \boldsymbol{f}(y)\right] = \frac{1}{2}\mathop{\mathbf{E}}_{\boldsymbol{z}}\left[\max\left\{\frac{|\boldsymbol{w}|}{R}, 1\right\}\right].$$

When $p \in (1,2]$, we notice that by Jensen's inequality and Lemma 8.11.1,

$$\mathop{\mathbf{E}}_{\boldsymbol{z}}\left[\max\left\{\frac{|\boldsymbol{w}|}{R}, 1\right\}\right] \le \max\left\{\frac{C_p\|x-y\|_p}{R}, 1\right\} = \frac{C_p\|x-y\|_p}{R},$$

and similarly, $\mathbf{E}_{\boldsymbol{z}}\left[\max\{|\boldsymbol{w}|/R, 1\}\right] \le \|x-y\|_1/(dR)$ when $p=1$. Furthermore, for $p \in (1,2]$, we lower bound the above quantity by

$$\mathop{\mathbf{E}}_{\boldsymbol{z}}\left[\max\left\{\frac{|\boldsymbol{w}|}{R}, 1\right\}\right] \ge \frac{\|x-y\|_p}{R} \cdot \mathop{\mathbf{E}}_{\boldsymbol{z}\sim\mathcal{D}_p}\left[\max\left\{|\boldsymbol{z}|, t_p\right\}\right] \ge \frac{\|x-y\|_p}{R} \cdot \frac{C_p}{2},$$

since $\mathbf{E}_{\boldsymbol{z}\sim\mathcal{D}_p}\left[\max\{|\boldsymbol{z}|, t_p\}\right] \to C_p$ as $t_p \to \infty$, which means that for some constant setting of high enough t_p, we obtain the above inequality. In particular, for $p \in (1,2]$, we may choose t_p to

be a large enough constant depending only on p. When $p = 1$,

$$\mathop{\mathbf{E}}_{\boldsymbol{z}}\left[\max\left\{\frac{|\boldsymbol{w}|}{R}, 1\right\}\right] \geq \sum_{i=1}^{d} \frac{1}{d}\left(1 - \frac{1}{d}\right)^{d-1} \max\left\{\frac{|x_i - y_i|}{R}, 1\right\} = \frac{1}{d}\left(1 - \frac{1}{d}\right)^{d-1} \frac{\|x - y\|_1}{R},$$

by setting $t_1 = 1$.

For $p \in (1, 2]$, we consider concatenating m independent executions of the above construction and consider that as an embedding of $\boldsymbol{\varphi}\colon \mathbb{R}^d \to \{0,1\}^m$, we have that for any $x, y \in \mathbb{R}^d$ with $\|x - y\|_p \leq R/t_p$,

$$\begin{aligned}\mathop{\mathbf{E}}_{\boldsymbol{\varphi}}\left[\|\boldsymbol{\varphi}(x) - \boldsymbol{\varphi}(y)\|_1\right] &= m \cdot \mathop{\mathbf{Pr}}_{\boldsymbol{f}}\left[\boldsymbol{f}(x) \neq \boldsymbol{f}(y)\right] \\ &\asymp m \cdot \frac{\|x - y\|_p}{R},\end{aligned}$$

where the notation $\asymp$ suppresses constant factors. Suppose $A \cup B \subset \mathbb{R}^d$ is a subset of at most $2s$ points with aspect ratio

$$\Phi \stackrel{\text{def}}{=} \frac{\max_{x,y\in A\cup B} \|x - y\|_p}{\min_{\substack{x,y\in A\cup B \\ x\neq y}} \|x - y\|_p}.$$

Then, we let $R = t_p \cdot \max_{x,y\in A\cup B} \|x - y\|_p$, and

$$m = O\left(t_p \Phi \log s\right).$$

For any fixed setting of $x, y \in A \cup B$, the distance between $\boldsymbol{\varphi}(x)$ and $\boldsymbol{\varphi}(y)$ is the number of independent of executions of $\boldsymbol{f}$ where $\boldsymbol{f}(x) \neq \boldsymbol{f}(y)$. Since executions are independent, we apply a Chernoff bound to say that with probability $1 - 1/s^{100}$ every $x, y \in A \cup B$ has $\|\boldsymbol{\varphi}(x) - \boldsymbol{\varphi}(y)\|_1$ concentrating up to a constant factor around its expectation, and therefore, we apply a union bound over $4s^2$ many pairs of points in $A \cup B$ to conclude $\boldsymbol{\varphi}$ is a constant distortion embedding. The case $p = 1$ follows similarly, except that

$$\mathop{\mathbf{E}}_{\boldsymbol{\varphi}}\left[\|\boldsymbol{\varphi}(x) - \boldsymbol{\varphi}(y)\|_1\right] \asymp m \cdot \frac{\|x - y\|_1}{dR},$$

and $R = 2 \max_{x,y\in A\cup B} \|x - y\|_\infty$. In summary, we have the following lemmas.

Lemma 8.11.2. *Fix any $p \in (1, 2]$, $n, d \in \mathbb{N}$, and a parameter $\Phi \in \mathbb{R}^{\geq 0}$. There exists a distribution $\mathcal{E}_p$ over embeddings $\boldsymbol{\varphi}\colon \mathbb{R}^d \to \{0,1\}^{d'}$, where*

$$d' = O(\Phi \log n),$$

such that for any fixed set $X \subset \mathbb{R}^d$ of size at most n and aspect ratio in ℓ_p at most Φ, a draw $\varphi \sim \mathcal{E}_p$ is a constant distortion embedding of X into the hypercube with Hamming distance with probability at least $1 - 1/n^{10}$.

Lemma 8.11.3. *Fix any $n, d \in \mathbb{N}$, and a parameter $r, R \in \mathbb{R}^{\geq 0}$. There exists a distribution $\mathcal{E}_1$ over embeddings $\varphi \colon \mathbb{R}^d \to \{0,1\}^{d'}$, where*

$$d' = O\left(\frac{dR \log n}{r}\right),$$

such that for any fixed set $X \subset \mathbb{R}^d$ of size at most n and ℓ_∞ distance at most R and ℓ_1 distance at least r, a draw $\varphi \sim \mathcal{E}_1$ is a constant distortion embedding of X into the hypercube with Hamming distance with probability at least $1 - 1/n^{10}$.

8.12 Relationship between Linear Sketching, Distributed Communication Protocols, and Streaming

In this chapter, we developed linear sketching algorithms for the problem of computing EMD. We now describe how these linear sketching algorithms result in streaming and distributed communication protocols. To do so, we briefly re-introduce these models, with further details which allow us to relate several nuances between the models.

Two-Party Communication. In the two-party communication problem, there are two parties, Alice and Bob. Alice is given as input a multi-set $A \subset \{0,1\}^d$, and Bob is also given a multi-set $B \subset \{0,1\}^d$, where $|A| = |B| = s$. Their goal is to jointly approximate the value of $\mathrm{EMD}(A, B)$. To do this, Alice and Bob exchange messages in *rounds*, where in each round one player sends exactly one message to the other player. Without loss of generality, we assume that Alice sends a message first. Thus, in a one-round protocol, Alice sends exactly one message M_1 to Bob, and then given M_1 and his input B, Bob must output an approximation $\tilde{R}$ to $\mathrm{EMD}(A, B)$. In a two-round protocol, Alice sends exactly one message M_1 to Bob. Given M_1 and B, Bob decides on a message M_2, and sends M_2 to Alice. After receiving M_2, Alice must then output an approximation $\tilde{R}$ to $\mathrm{EMD}(A, B)$. We work in the *public-coin* model of communication, where Alice and Bob have access to a shared infinite random string.

A protocol $\mathcal{P}$ for the two-party Earth-Mover Distance approximation problem is the procedure by which Alice and Bob compute the messages and their output. A protocol $\mathcal{P}$ is said to

be correct if it achieves a desired approximation with probability at least $2/3$ over the coin flips of $\mathcal{P}$ and the shared random string. For a protocol $\mathcal{P}$, the *communication complexity* of $\mathcal{P}$ is the maximum total length of all exchanged messages.

The Streaming Model. In the steaming model, points arrive from $A \cup B$ in a stream. Specifically, at each time step in the stream, a point $p \in \{0,1\}^d$ is inserted into the stream, along with an identifier of whether $p \in A$ or $p \in B$. At the end of the stream, the algorithm must output an approximation to $\mathrm{EMD}(A, B)$. Recall that in the *turnstile* model of streaming, points p can also be *deleted* from the stream (not necessarily having been inserted before), so long as at the end of the stream the sets A, B defined by the stream satisfy $|A| = |B| = s$. This geometric stream can in fact be modeled as a typical data stream, where the updates are coordinate-wise updates to a "frequency vector" f. Here, we let $f \in \mathbb{R}^n$, where $n = 2 \cdot 2^d$ be a vector initalized to 0. At each time step t, the vector f received a coordinate-wise update (i_t, Δ_t), where $i_t \in [n]$ and Δ_t is a integer, which causes the change $f_{i_t} \leftarrow f_{i_t} + \Delta_t$. To see the equivalence, if we want to insert a point p into A a total of z times, we can make the update (p, z), where p indexes into $[2^d]$ in the natural way. Similarly, to add a point p into B a total of z times, we make the update $(p + 2^d, z)$, and deletions are handled similarly. We make the common assumption that $|\Delta_t| = \mathrm{poly}(s)$ for all t, and that the length of the stream is at most $\mathrm{poly}(s)$, so that the coordinates of the vector f can be represented with $\log(s)$ bits of space at any intermediate point in the stream.

The goal of the streaming model is to maintain a small space sketch of the vector f, so at the end of the stream an algorithm can produce a good approximation to the earth-mover distance $\mathrm{EMD}(A, B)$. When discussing the space complexity of streaming algorithms, there are two separate notions: *working space* and *intrinsic space*. We remark that generally these two notations of space are the same (up to constant factors) for streaming algorithms, however for our purposes it will be useful to distinguish them. The *working space* of a streaming algorithm $\mathcal{A}$ is the space required to store an update (i_t, Δ_t) in the stream and process it. The *intrinsic space* is the space which the algorithm must store *between* updates. The intrinsic space coincides with the size of a message which must be passed from one party two another, if each party, for instance, holds some fraction of the data stream. Thus, streaming computation is generally focused on the intrinsic space, which we will hereafter just refer to as the *space* of the algorithm. Notice that the working space must necessarily be sufficient to read the index i_t. In the case of EMD, i_t must be represented with d bits of space, meaning that $\Omega(d)$ is a lower bound on the working memory of a streaming algorithm in this model. However, the space complexity of streaming algorithms for EMD may be smaller than this required working space.

In the streaming model, the corresponding notation for the *public coin* model is the *random*

oracle model of computation. This simply establishes that the steaming algorithm is given random access to an infinitely long string of random bits, whose size does not count against the space complexity of the algorithm. As show below, linear sketching immedately implies a streaming algorithm with the same space in the random oracle model. To remove this assumption, the usage of pseudo-random generators or limited independence is generally required. In the *one-pass* streaming model, the algorithm only sees the sequence of updates a single time, whereas in the *two-pass* model the algorithm sees the sequence of updates exactly twice, one after the other.

Linear Sketching We now recall the concept of a *linear-sketch*. Linear sketching results in algorithms both for the streaming and two-party communication models. In this model, the multiset inputs $A, B \subset \{0,1\}^d$ are implicitly encoded by a vector $f_{A,B} \in \mathbb{R}^{2 \cdot 2^d}$. A linear sketch stores only the value $\mathbf{S} \cdot f_{A,B}$, where $\mathbf{S}$ is a (possibly random) matrix with $k \ll 2^d$ rows. The algorithm then outputs an estimate of $\mathrm{EMD}(A, B)$ given only knowledge of $\mathbf{S}f_{A,B}$ and $\mathbf{S}$. The space of a linear sketch is the space required to store $\mathbf{S}f_{A,B}$, since $\mathbf{S}$ is generated with public randomness which is not charged against that algorithm. This coincides with the space of a public coin communication protocol, or the space of a streaming algorithm in the random oracle model.

Given an update (i_t, Δ_t), one can update the sketch $\mathbf{S}f_{A,B} \leftarrow \mathbf{S}f_{A,B} + \mathbf{S}_{*,i_t}\Delta_t$. This allows a linear sketch to be maintained in a stream. Moreover, since the sketch Sf is linear, the order or sequence of updates in a stream do not matter. Given a linear sketching algorithm for earth-mover distance with sketch size $\mathbf{S}f_{A,B}$ with k rows, this yields a $O(k \log s)$ communication protocol for two-party one-round communication. This follows from the fact that the matrix $\mathbf{S}$ can be generated with shared randomness. Alice can then compute the vector f_A which is induced by her set A, and Bob can similarly compute f_B, such that $f_{A,B} = f_A + f_B$. Alice then sends $\mathbf{S}f_A$ to Bob, who can compute $\mathbf{S}f_{A,B} = \mathbf{S}f_A + \mathbf{S}f_B$, and therefore solve the communication problem.

There is a corresponding notion of linear sketching for the two-round communication and two-pass streaming models, which we call a two-round sketching algorithm. A two round sketching algorithm first (with no knowledge of $f_{A,B}$) generates a matrix $\mathbf{S}_1$ from some distribution $\mathcal{D}^1$, and computes $\mathbf{S}_1 f_{A,B}$. Then, given knowledge only of $\mathbf{S}_1$ and $\mathbf{S}_1 f_{A,B}$, it generates a *second* matrix $\mathbf{S}_2$ from a distribution $\mathcal{D}^2(\mathbf{S}_1, \mathbf{S}_1 f_{A,B})$, and computes $\mathbf{S}_2 f_{A,B}$. Finally, given as input only the values $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_1 f_{A,B}, \mathbf{S}_2 f_{A,B})$, it outputs an approximation to $\mathrm{EMD}(A, B)$. The space of the algorithm is the number of bits required to store $\mathbf{S}_1 f_{A,B}$ and $\mathbf{S}_2 f_{A,B}$.

It is easy to see that a two-round linear sketching algorithm results in both a two-pass streaming algorithm and a two-round communication protocol. For the former, on the first pass the

algorithm maintains $\mathbf{S}_1 f$ and $\mathbf{S}_1$ using shared randomness. At the end of the first pass, it can generate $\mathbf{S}_2$ based on $\mathbf{S}_1 f_{A,B}$ and $\mathbf{S}_1$, and compute $\mathbf{S}_2 f_{A,B}$ as needed. For a two-round communication protocol, Alice and Bob jointly generate $\mathbf{S}_1$, then Alice sends $\mathbf{S}_1 f_A$ to Bob, who computes himself $\mathbf{S}_1 f_{A,B}$, and then using shared randomness and his knowledge of $\mathbf{S}_1, \mathbf{S}_1 f_{A,B}$ can compute $\mathbf{S}_2$. Bob then sends $\mathbf{S}_1 f_{A,B}, \mathbf{S}_2 f_B$ back to Alice, who can now fully determine $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_1 f_{A,B}, \mathbf{S}_2 f_{A,B}$ and output the approximation.

Two-Pass Streaming. We now demonstrate that this two-round linear sketch of Section 8.5 can be applied to obtain a two-pass streaming algorithm in the turnstile (insertion and deletion) model. Here, a stream of at most $\mathrm{poly}(s)$ updates arrives in the stream, where each update either inserts or deletes a point from A, or inserts or deletes a point from B. Noticed that the t-th update can be modelled by coordinate-wise updates to $f_{A,B}$ of the form $(i_t, \Delta_t) \in [2 \cdot 2^d] \times \{-M, -M+1, \ldots, M\}$, where $M = \mathrm{poly}(s)$, causing the change $(f_{A,B})_{i_t} \leftarrow (f_{A,B})_{i_t} + \Delta_t$. At the end of the stream, we are promised that $f_{A,B}$ is a valid encoding of two multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$. A two-pass streaming algorithm is allowed to make two passes over the stream before outputting an estimate.

Corollary 8.12.1. *For $d, s \in \mathbb{N}$, there exists a 2-pass turnstile streaming algorithm which, on a stream vector $f_{A,B}$ encoding multi-sets $A, B \subset \{0,1\}^d$ with $|A| = |B| = s$, the algorithm then computes an approximate $\hat{\mathcal{I}}$ to $\mathrm{EMD}(A, B)$ with*

$$\mathrm{EMD}(A, B) \leq \hat{\mathcal{I}} \leq \tilde{O}(\log s)\, \mathrm{EMD}(A, B)$$

with probability at least $3/4$, and uses $O(d \log d) + \mathrm{polylog}(s, d)$ bits of space. Moreover, the algorithm stores its own randomness (i.e., does not required the random oracle model).

Proof. The only step remaining is to derandomize the algorithm (i.e., modify the algorithm so that it can store its randomness in small space). First note that the universe reduction step requires the generation of two hash functions h_i, h_{i-1} mapping a universe of size at most 2^d to a universe of size s. Moreover, for the proof of Proposition 8.5.1, all that is needed is 2-wise independence, since the proof only argues about the probability of a collision of a fixed pair and applies a union bound. Since a 2-wise independent hash function $h : U_1 \to U_2$ can be stored in $O(\log(|U_1| + |U_2|))$ bits of space, this only adds an additive $O(d)$ bits to the space complexity of the algorithm.

Next, note that the linear sketching algorithms of [Ind06] and Chapter 3 used in the first and second passes are both are already derandomized in $\mathrm{poly}(\log s)$-bits of space. Thus, it will suffice to consider the randomness needed to store the Quadtree T. By Remark 97, in each depth

$t \in [h]$ of the Quadtree we can sample the same set $(\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_{2^t}) \sim [d]$ of coordinates for each vertex at that depth (instead of independently sampling coordinates in every vertex at the same depth). Since $h = \log 2d$, the total number of bits we must sample to define an entire quadtree is $2d \log d$. Thus, after sampling such a quadtree T with $O(d \log d)$ bits, and storing these bits, the remainder of the algorithm is already stores its randomness. Moreover, because there are at most $\mathrm{poly}(s)$ updates to $f_{A,B}$, the coordinates of each linear sketch can be stored in $O(\log s)$ bits of space, which completes the proof.

□

Part II

Numerical Linear Algebra

Chapter 9

Testing Positive Semi-Definiteness

The second part of this thesis is concerned with the application of sketching techniques to randomized *numerical linear algebra* (NLA). Sketching as a tool for randomized NLA has a distinguished history, leading to breakthrough results in the past decade such as the fastest regression and low-rank approximation algorithms [CW17, W$^+$14]. A thorough treatment of the fundamental results in sketching for NLA can be found in the surveys [W$^+$14, KV17, Mah11]. The material which follows is based on our work [BCJ20].

In this chapter, we study the problem of testing whether a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD). Recall that $\mathbf{A}$ is said to be PSD if it defines a non-negative quadtratic form: namely if $x^\top \mathbf{A} x \geq 0$ for all vectors $x \in \mathbb{R}^n$. If $\mathbf{A}$ is symmetric, this is equivalent to all its eigenvalues $\lambda_i(\mathbf{A})$ being non-negative. PSD matrices are central objects of interest in algorithm design, and are studied extensively in optimization, spectral graph theory, numerical linear algebra, statistics, and dynamical systems, among many others [VB96, WSV12, GW95, ARV09, AHK05, Ste10, ST04, DL09, Wai19, DK19, SL$^+$91].

Matrices which are positive semi-definite have particularly interesting and intricate structure, and admit a plethora of nice algebraic and algorithmic properties. Given a matrix $\mathbf{A}$, the problem of testing whether $\mathbf{A}$ is PSD is, therefore, a fundamental task in numerical linear algebra. For instance, certifying whether a matrix is PSD often provides crucial insights into the structure of metric spaces [Sch35], arises as a separation oracles in Semi-Definite Programming (SDP) [VB96], and leads to faster algorithms for solving linear systems and linear algebra problems [ST04, KOSZ13, MW17, BCW20].

In general, efficient, numerically stable algorithms for computing the spectrum of a matrix $\mathbf{A}$ have been known since Turing [Tur48]. However, such algorithms require reading the entire

matrix and require cubic $O(n^3)$ running time in practice or $O(n^\omega)$ in theory, where $\omega < 2.373$ is the exponent of matrix multiplication. This spectrum computation is often the bottleneck in applications, especially when just determining the existence of negative eigenvalues suffices. For instance, checking embeddability of a finite metric into Euclidean space, feasibility of a SDP, convexity of a function, and if specialized solvers are applicable for linear algebraic problems, all only require knowledge of whether a given matrix is PSD.

The focus of this Chapter is to study when the property of being PSD can be tested *sublinear* time and queries, without reading the entire matrix. This algorithmic perspective is known as the *property testing framework* [GGR98, Gol17]. Property testing is a natural model for sketching, as a small subset of queries which allow one to determine properties of a high-dimensional object is itself a low-dimensional sketch of that object.

More specifically, in the property testing model we are given as input a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, which is promised to either satisfy the desired property (i.e., $\mathbf{A}$ is PSD), or it is promised that $\mathbf{A}$ is "far", under an appropriate notion of distance (discussed in the following), from any PSD matrix. The goal is to distinguish between these two cases making as few queries to the entries of $\mathbf{A}$ as possible. In order to avoid trivialties, such as having arbitrarily large entries hidden in $\mathbf{A}$ which would force any algorithm to queries all the entries, some condition of boundedness is necessary. Consequentially, we choose to work in the natural *bounded entry model* proposed by Balcan, Li, Woodruff, and Zhang [BLWZ19], where the input matrix has bounded entries: $\|\mathbf{A}\|_\infty \leq 1$.

In this chapter, we initiate the study of testing postive semi-definiteness in the property testing model. We give algorithms for two related variants of the task, depending on how one defines the "far" case (discussed in further detail below in Section 9.1). The first, known as the ℓ_2^2 gap, promises that in the far case, we have $\min_{\mathbf{B}\text{-PSD}} \|\mathbf{A} - \mathbf{B}\|_F^2 \geq \epsilon n^2$ for some parameter $\epsilon \in (0, 2)$. The second is known as the ℓ_∞-gap, and promises that $\min_{\mathbf{B}\text{-PSD}} \|\mathbf{A} - \mathbf{B}\|_2 \geq \epsilon n$. Our main results are the design of a query-optimal algorithm for the latter task, and nearly matching upper and lower bounds for the former. Both our testers randomly sample a collection of principal submatrices and check whether these submatrices are PSD. Thus, our results can be seen as a intensive investigation into the eigenvalues of random submatrices. Specifically, the main technical challenge which follows is to demonstrate that random submatrices of a matrix which far from PSD is very likely itself not PSD.

Highlighted Contributions

The main contributions of this chapter as as follows:

- A non-adaptive tester which solves the ℓ_∞-gap problem using only $\tilde{O}(1/\epsilon^2)$ queries to the entries of $\mathbf{A}$, and a matching lower bound of $\Omega(1/\epsilon^2)$ even for any algorithm (Section 9.3).
- A non-adaptive tester which solves the ℓ_2^2-gap problem using only $\tilde{O}(1/\epsilon^4)$ queries to the entries of $\mathbf{A}$ (Section 9.4).
- A lower bound of $\tilde{O}(1/\epsilon^2)$ for any non-adaptive algorithm (Section 9.5).

9.1 Background

The goal of this chapter is to provide property testing algorithms for positive semi-definiteness. As discussed earlier, we work in the *bounded-entry model*, proposed by Balcan, Li, Woodruff, and Zhang [BLWZ19], where the input matrix has bounded entries: $\|\mathbf{A}\|_\infty \leq 1$. Boundedness is often a natural assumption in practice, and has numerous real world applications, such as recommender systems as in the Netflix Challenge [KBV09], unweighted or bounded weight graphs [Gol10, GGR98], correlation matrices, distance matrices with bounded radius, and others [LWW14, KIDP16, BLWZ19]. Further, as noted, the boundedness of entries avoids degenerate instances where an arbitrarily large entry is hidden in $\mathbf{A}$, thereby drastically changing the spectrum of $\mathbf{A}$, while being impossible to test without reading the entire matrix.

The work [BLWZ19] introduced the bounded entry model specifically to study the query complexity of testing spectral properties of matrices, such as stable rank (the value $\|\mathbf{A}\|_F^2/\|\mathbf{A}\|_2^2$) and Schatten p norms. A previously studied model, known as the *bounded row model*, which requires that rows instead of entries be bounded, was studied by Li, Wang, and Woodruff [LWW14], who gave tight bounds for testing stable rank in this model. The problem of testing the *rank* of a matrix has also been very well studied [PR03, KS03, LWW14, BBG18], and admits tight bounds without the requirement of boundedness. This is a consequence of the fact that matrix rank is not a smooth spectral property, so hiding an unbounded value in a single entry of $\mathbf{A}$ cannot drastically alter the rank. Thus, for testing rank, the condition of boundedness is not required.

The starting point for this chapter is a simple fact: a matrix $\mathbf{A}$ is PSD if and only if all

principal[1] submatrices of $\mathbf{A}$ are PSD. However, a much more interesting direction is: if $\mathbf{A}$ is not PSD, what can be said about the eigenvalues of the submatrices of $\mathbf{A}$? Specifically, if $\mathbf{A}$ is far from PSD, how large of a submatrix must one sample in order to find a negative eigenvalue? Note that given a principal submatrix $\mathbf{A}_{T\times T}$ with $x^\top \mathbf{A}_{T\times T} x < 0$ for some $x \in \mathbb{R}^{|T|}$, this direction x can be used as a certificate that the input matrix is not PSD, since $y^\top \mathbf{A} y = x^\top \mathbf{A}_{T\times T} x < 0$, where y is the result of padding x with 0's. Further, it leads us to a natural algorithm to test definiteness: sample multiple principal submatrices and compute their eigenvalues. If any are negative, then $\mathbf{A}$ must not be PSD. Determining the query complexity of this task is the principal focus of this chaoter. Specifically, we ask:

Can the positive semi-definiteness of a bounded matrix be tested via the semi-definiteness of a small random submatrix?

9.1.1 Property Testing PSD Matrices in the Bounded Entry Model

The distance from $\mathbf{A}$ to the PSD cone is given by $\min_{\mathbf{B}\succeq 0} \|\mathbf{A}-\mathbf{B}\|$, where $\|\cdot\|$ is a norm, and $\mathbf{B} \succeq 0$ denotes that $\mathbf{B}$ is PSD. To instantiate $\|\cdot\|$, we consider two natural norms over $n \times n$ matrices: the spectral norm ($\|\cdot\|_2$) and the Euclidean norm ($\|\cdot\|_F$). Perhaps surprisingly, the distance of a symmetric matrix $\mathbf{A}$ to the PSD cone under these norms can be characterized in terms of the eigenvalues of $\mathbf{A}$. In particular, let $\lambda \in \mathbb{R}^n$ be the vector of eigenvalues of $\mathbf{A}$. Then, the spectral norm distance corresponds to the ℓ_∞ distance between λ and the positive orthant. Similarly, the squared Frobenius distance corresponds to the ℓ_2^2 distance between λ and the positive orthant.

Therefore, we will refer to the two resulting gap problems as the ℓ_∞-gap and the ℓ_2^2-gap, respectively. This connection between matrix norms of $\mathbf{A}$ and vector norms of eigenvalues λ will be highly useful for the analysis of random submatrices. Next, we formally define the testing problems:

Problem 1 (PSD Testing with Spectral norm/ℓ_∞-gap)**.** *Given $\epsilon \in (0,1]$ and a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ such that $\|\mathbf{A}\|_\infty \le 1$, distinguish whether $\mathbf{A}$ satisfies:*

(1) $\mathbf{A}$ is PSD.

(2) $\mathbf{A}$ is ϵ-far from the PSD cone in Spectral norm: $\min_{\mathbf{B}\succeq 0} \|\mathbf{A}-\mathbf{B}\|_2 = \max_{i:\lambda_i<0} |\lambda_i(\mathbf{A})| \ge \epsilon n$.

[1] Recall that a principal submatrix $\mathbf{A}_{T\times T}$ for $T \subseteq [n]$ is the restriction of $\mathbf{A}$ to the rows and columns indexed by T.

The fact that the spectral norm distance from $\mathbf{A}$ to the PSD cone ($\min_{\mathbf{B} \succeq 0} \|\mathbf{A} - \mathbf{B}\|_2$) is equivalent to the magnitude of the smallest negative eigenvalue of $\mathbf{A}$ is a consequence of the variational principle for eigenvalues. For general non-symmetric matrices $\mathbf{A}$, one can replace (2) above with the condition $x^\top \mathbf{A} x < -\epsilon n$ for some unit vector $x \in \mathbb{R}^n$, which is equivalent to (2) if $\mathbf{A}$ is symmetric (again by the variational principle). We note that our results for the ℓ_∞-gap hold in this more general setting.[2]

Next, if we instantiate $\|\cdot\|$ with the (squared) Euclidean norm, we obtain the ℓ_2^2 gap problem.

Problem 2 (PSD Testing with ℓ_2^2-gap)**.** *Given $\epsilon \in (0, 1]$ and a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ such that $\|\mathbf{A}\|_\infty \leq 1$, distinguish whether $\mathbf{A}$ satisfies:*

(1) $\mathbf{A}$ is PSD.

(2) $\mathbf{A}$ is ϵ-far from the PSD cone in squared Euclidean norm:

$$\min_{\mathbf{B} \succeq 0} \|\mathbf{A} - \mathbf{B}\|_F^2 = \sum_{i : \lambda_i(\mathbf{A}) < 0} \lambda_i^2(\mathbf{A}) \geq \epsilon n^2 \tag{9.1}$$

Note that the identity $\min_{\mathbf{B} \succeq 0} \|\mathbf{A} - \mathbf{B}\|_F^2 = \sum_{i : \lambda_i(\mathbf{A}) < 0} \lambda_i^2(\mathbf{A})$ in equation 9.1 also follows from a simple application of the variational principle for eigenvalues (see Section 9.6). Similarly to the ℓ_∞-gap, if $\mathbf{A}$ is not symmetric one can always run a tester on the symmetrization $(\mathbf{A} + \mathbf{A}^\top)/2$. Also observe that $\|\mathbf{A}\|_F^2 \leq n^2$ and $\|\mathbf{A}\|_2 \leq n$ for bounded entries matrices, hence the respective scales of n, n^2 in the two gap instances above. Notice by definition, if a symmetric matrix $\mathbf{A}$ is ϵ-far from PSD in ℓ_∞ then $\mathbf{A}$ is ϵ^2-far from PSD in ℓ_2^2. However, the converse is clearly not true, and as we will see the complexity of PSD testing with ϵ^2-ℓ_2^2 gap is strictly harder than testing with ϵ-ℓ_∞ gap.[3]

We note that there are several important examples of matrices which are far from the PSD cone in ℓ_2^2, but which are not far in ℓ_∞.

Example 109. Let $\mathbf{A} \sim \{-1, 1\}^{n \times n}$ be a random sign matrix with independent entries. Then as a consequence of Wigner's Semicircle Law $\mathbf{A}$ will be $\Omega(1)$-far in ℓ_2^2 distance. However, $\|\mathbf{A}\|_2 = O(\sqrt{n})$ with high probability, so $\mathbf{A}$ will only be $O(1/\sqrt{n})$-far in ℓ_∞ distance. The same holds true when the entries of $\mathbf{A}$ are drawn independently from the Gaussian distribution, or any

[2]Also note that given query access to any $\mathbf{A} \in \mathbb{R}^{n \times n}$, one can always run a tester on the symmetrization $\mathbf{B} = (\mathbf{A} + \mathbf{A}^\top)/2$, which satisfies $x^\top \mathbf{A} x = x^\top \mathbf{B} x$ for all x, with at most a factor of 2 increase in query complexity.

[3]The difference in scaling of ϵ between the ℓ_∞ and ℓ_2^2 gap definitions (ϵ is squared in the latter) is chosen for the sake of convenience, as it will become clear the two problems are naturally studied in these respective paramaterizations.

distribution with bounded moments.

Intuitively, random examples like the above should be very "far" from being PSD, and the ℓ_2^2 distance captures this fact.

9.1.2 Contributions in the Bounded Entry Model

We now introduce our main contributions. Our algorithms for PSD testing randomly sample principal submatrices and check if they are PSD. Thus, all our algorithms have one-sided error; when $\mathbf{A}$ is PSD, they always return PSD, and whenever our algorithms return Not PSD, they output a certificate in the form of a principal submatrix which is not PSD. In what follows, $\omega < 2.373$ is the exponent of matrix multiplication. We first state our result for the ℓ_∞ gap problem in its most general form, which is equivalent to Problem 1 in the special case when $\mathbf{A}$ is symmetric.

Theorem 116 (ℓ_∞-gap Upper Bound) *There is a non-adaptive sampling algorithm which, given $\mathbf{A} \in \mathbb{R}^{n\times n}$ with $\|\mathbf{A}\|_\infty \le 1$ and $\epsilon \in (0,1)$, returns PSD if $x^\top \mathbf{A} x \ge 0$ for all $x \in \mathbb{R}^n$, and with probability $2/3$ returns Not PSD if $x^\top \mathbf{A} x \le -\epsilon n$ for some unit vector $x \in \mathbb{R}^n$. The algorithm make $\tilde{O}(1/\epsilon^2)$ queries to the entries of $\mathbf{A}$, and runs in time $\tilde{O}(1/\epsilon^\omega)$.*

We demonstrate that the algorithm of Theorem 116 is optimal up to $\log(1/\epsilon)$ factors, even for adaptive algorithms with two-sided error. Formally, we show:

Theorem 120 (ℓ_∞-gap Lower Bound) *Any adaptive or non-adaptive algorithm which solves the PSD testing problem with ϵ-ℓ_∞ gap with probability at least $2/3$, even with two-sided error and if $\mathbf{A}$ is promised to be symmetric, must query $\widetilde{\Omega}(1/\epsilon^2)$ entries of $\mathbf{A}$.*

Next, we present our algorithm for the ℓ_2^2-gap problem. Our algorithm crucially relies on first running our tester for the ℓ_∞-gap problem, which allows us to demonstrate that if $\mathbf{A}$ is far from PSD in ℓ_2^2 but close in ℓ_∞, then it must be far, under other notions of distance such as Schatten norms or residual tail error, from any PSD matrix.

Theorem 119 (ℓ_2^2-gap Upper Bound) *There is a non-adaptive sampling algorithm which, given a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with $\|\mathbf{A}\|_\infty \le 1$ and $\epsilon \in (0,1)$, returns PSD if $\mathbf{A}$ is PSD, and with probability $2/3$ returns Not PSD if $\min_{\mathbf{B}\succeq 0} \|\mathbf{A}-\mathbf{B}\|_F^2 \ge \epsilon n^2$. The algorithm make $\tilde{O}(1/\epsilon^4)$ queries to $\mathbf{A}$, and runs in time $\tilde{O}(1/\epsilon^{2\omega})$.*

We complement our upper bound by a $\widetilde{\Omega}(\frac{1}{\epsilon^2})$ lower bound for PSD-testing with ϵ-ℓ_2^2 gap, which holds even for algorithms with two sided error. Our lower bound demonstrates a separation between the complexity of PSD testing with $\sqrt{\epsilon}$-ℓ_∞ gap and PSD testing with ϵ-ℓ_2^2-gap, showing that the concentration of negative mass in large eigenvalues makes PSD testing a strictly easier problem.

Theorem 123 (ℓ_2^2-gap Lower Bound) *Any non-adaptive algorithm which solves the PSD testing problem with ϵ-ℓ_2^2 gap with probability at least $2/3$, even with two-sided error, must query $\widetilde{\Omega}(1/\epsilon^2)$ entries of $\mathbf{A}$.*

Our lower bound is built on discrete hard instances which are "locally indistinguishable", in the sense that the distribution of any small set of samples is completely identical between the PSD and ϵ-far cases. At the heart of the lower bound is a key combinatorial Lemma about arrangements of paths on cycle graphs (see discussion in Section 9.2). Our construction is highly general, and we believe will likely be useful for proving other lower bounds for matrix and graph property testing problems. Exemplifying the applicability of our construction, we obtain as an immediate corollary a new lower bound for testing the Schatten-1 norm of $\mathbf{A}$. Recall, that the Schatten-1 norm is defined via $\|\mathbf{A}\|_{\mathcal{S}_1} = \sum_i \sigma_i(\mathbf{A})$, where $\sigma_1(\mathbf{A}) \geq \cdots \geq \sigma_n(\mathbf{A})$ are the singular values of $\mathbf{A}$.

Theorem 124 (Schatten-1 Lower Bound) *Fix any $1/\sqrt{n} \leq \epsilon \leq 1$. Then any non-adaptive algorithm in the bounded entry model that distinguishes between*

1. $\|\mathbf{A}\|_{\mathcal{S}_1} \geq \epsilon n^{1.5}$,
2. $\|\mathbf{A}\|_{\mathcal{S}_1} \leq (1-\epsilon_0)\epsilon n^{1.5}$

with probability $2/3$, where $\epsilon_0 = 1/\log^{O(1)}(1/\epsilon)$, must make at least $\tilde{\Omega}(1/\epsilon^4)$ queries to $\mathbf{A}$.

Note that one always has $\|\mathbf{A}\|_{\mathcal{S}_1} \leq n^{1.5}$ in the bounded entry model ($\|\mathbf{A}\|_\infty \leq 1$), which accounts for the above scaling. Theorem 124 extends a lower bound of Balcan et. al. [BLWZ19], which is $\Omega(n)$ for the special case of $\epsilon, \epsilon_0 = \Theta(1)$. Thus, for the range $\epsilon = \tilde{O}(n^{-1/4})$, our lower bound is an improvement. To the best of our knowledge, Theorem 124 gives the first $\tilde{\Omega}(n^2)$ sampling lower bound for testing Schatten-1 in a non-degenerate range (i.e., for $\|\mathbf{A}\|_{\mathcal{S}_1} > n$).

Remark 110. We note that the lower bound of [BLWZ19] is stated for a slightly different version of gap (a "ϵ-ℓ_0"-gap), where either $\|\mathbf{A}\|_{\mathcal{S}_1} \geq c_1 n^{1.5}$ for a constant c_1, or at least ϵn^2 of the entries of $\mathbf{A}$ must be changed (while respecting $\|\mathbf{A}\|_\infty \leq 1$) so that the Schatten-1 is larger

than $c_1 n^{1.5}$. However, their lower bound construction itself satisfies the "Schatten-gap" version as stated in Theorem 124, where either $\|\mathbf{A}\|_{\mathcal{S}_1} \geq c_1 n^{1.5}$, or $\|\mathbf{A}\|_{\mathcal{S}_1} \leq c_2 n^{1.5}$ and $c_1 > c_2$ are constants. From here, it is easy to see that this gap actually implies the ℓ_0-gap (and this is used to obtain the ℓ_0-gap lower bound in [BLWZ19]), since if $\|\mathbf{A}\|_{\mathcal{S}_1} \leq c_2 n^{1.5}$ then for any $\mathbf{E}$ with $\|\mathbf{E}\|_\infty \leq 2$ and $\|\mathbf{E}\|_0 \leq \epsilon n^2$ for a small enough constant $\epsilon < c_2^2$, we have $\|\mathbf{A} + \mathbf{E}\|_{\mathcal{S}_1} \leq \|\mathbf{A}\|_{\mathcal{S}_1} + \|\mathbf{E}\|_{\mathcal{S}_1} \leq n^{1.5}(c_2 + 2\sqrt{\epsilon}) < c_1 n^{1.5}$. So Theorem 124 implies a lower bound of $\tilde{\Omega}(1/\epsilon^2)$ for distinguishing $\|\mathbf{A}\|_{\mathcal{S}_1} \geq \sqrt{\epsilon} n^{1.5}$ from the case of needing to change at least $\tilde{\Omega}(\epsilon n^2)$ entries of $\mathbf{A}$ so that $\|\mathbf{A}\|_{\mathcal{S}_1} \geq \sqrt{\epsilon} n^{1.5}$. Thus, our lower bound *also* extends the ℓ_0-gap version of the results of [BLWZ19] for the range $\epsilon = \tilde{O}(1/\sqrt{n})$.

In addition to Schatten-1 testing, the same lower bound construction and techniques from Theorem 123 also result in new lower bounds for testing the Ky-Fan s norm $\|\mathbf{A}\|_{\mathsf{KF}(s)} = \sum_{i=1}^{s} \sigma_i(\mathbf{A})$, as well as the cost of the best rank-s approximation $\|\mathbf{A} - \mathbf{A}_s\|_F^2 = \sum_{i>s} \sigma_i^2(\mathbf{A})$, stated below. In the following, s is any value $1 \leq s \leq n/(\operatorname{poly}\log n)$, and c is a fixed constant.

Theorem 125 (Ky-Fan Lower Bound) *Any non-adaptive algorithm in the bounded entry model which distinguishes between*

1. $\|\mathbf{A}\|_{KF(s)} > \frac{c}{\log s} n$
2. $\|\mathbf{A}\|_{KF(s)} < (1 - \epsilon_0) \cdot \frac{c}{\log s} n$

with probability $2/3$*, where* $\epsilon_0 = \Theta(1/\log^2(s))$*, must query at least* $\tilde{\Omega}(s^2)$ *entries of* $\mathbf{A}$.

Theorem 126 (Residual Error Lower Bound) *Any non-adaptive algorithm in the bounded entry model which distinguishes between*

1. $\|\mathbf{A} - \mathbf{A}_s\|_F^2 > \frac{c}{s \log s} n$
2. $\|\mathbf{A} - \mathbf{A}_s\|_F^2 < (1 - \epsilon_0) \cdot \frac{c}{s \log s} n$

with probability $2/3$*, where* $\epsilon_0 = 1/\log^{O(1)}(s)$*, must query at least* $\tilde{\Omega}(s^2)$ *entries of* $\mathbf{A}$.

Our lower bound for the Ky-Fan norm complements a Ky-Fan testing lower bound of [LW16], which is $\Omega(n^2/s^2)$ for distinguishing **1)** $\|\mathbf{A}\|_{KF(s)} < 2.1 s\sqrt{n}$ from **1)** $\|\mathbf{A}\|_{KF(s)} > 2.4 s\sqrt{n}$ when $s = O(\sqrt{n})$. Note their bound decreases with s, whereas ours increases, thus the two bounds are incomparable (although they match up to $\log(s)$ factors at $s = \Theta(\sqrt{n})$).[4] We also point out that there are (not quite matching) upper bounds for both the problems of Ky-Fan norm and

[4]The bound from [LW16] is stated in the sketching model, however the entries of the instance are bounded, thus it also applies to the sampling model considered here.

s-residual error testing in the bounded entry model, just based on a standard application of the Matrix Bernstein Inequality.[5] We leave the exact query complexity of these and related testing problems for functions of singular values in the bounded entry model as subject for future work.

9.1.3 Connections to Optimization, Euclidean Metrics and Linear Algebra

We now describe some explicit instances where our algorithms may be useful for testing positive semi-definiteness. We emphasize that in general, the distance between $\mathbf{A}$ and the PSD cone may be too small to verify via our testers. However, when the input matrices satisfy a non-trivial gap from the PSD cone, we can speed up some basic algorithmic primitives. The first is testing feasibility of the PSD constraint in a Semi-Definite Program (SDP) with sublinear queries and time, so long as the variable matrix has bounded entries. Importantly, our algorithms also output a separating hyperplane to the PSD cone.

Corollary 9.1.1 (Feasibility and Separating Hyperplanes for SDPs)**.** *Given a SDP $\mathcal{S}$, let $\mathbf{X} \in \mathbb{R}^{n \times n}$ be a symmetric matrix that violates the PSD constraint for $\mathcal{S}$. Further, suppose $\|\mathbf{X}\|_\infty \leq 1$ and $\mathbf{X}$ is ϵn^2-far in entry-wise ℓ_2^2 distance to the PSD cone. Then, there exists an algorithm that queries $\widetilde{O}(1/\epsilon^4)$ entries in $\mathbf{X}$ and runs in $\widetilde{O}(1/\epsilon^{2\omega})$ time, and with probability $9/10$, outputs a vector $\tilde{v}$ such that $\tilde{v}^T \mathbf{X} \tilde{v} < 0$. Moreover, if $\lambda_{\min}(\mathbf{X}) < -\epsilon n$, then there is an algorithm yielding the same guarantee, that queries $\widetilde{O}(1/\epsilon^2)$ entries in $\mathbf{X}$ and runs in $\widetilde{O}(1/\epsilon^{\omega})$ time.*

While in the worst case, our algorithm may need to read the whole matrix to exactly test if $\mathbf{X}$ is PSD, there may be applications where relaxing the PSD constraint to the convex set of matrices which are close to the PSD cone in Euclidean distance is acceptable. Moreover, our algorithm may be run as a preliminary step at each iteration of an SDP solver to check if the PSD constraint is badly violated, resulting in speed-ups by avoiding an expensive eigendecomposition of $\mathbf{X}$ whenever our algorithm outputs a separating hyperplane [VB96].

Next, we consider the problem of testing whether an arbitrary finite metric d over n points, $x_1, \ldots x_n \in \mathbb{R}^d$ is embeddable into Euclidean Space. Testing if a metric is Euclidean has a myriad of applications, such as determining whether dimensionality reduction techniques such as Johnson-Lindenstrauss can be used [PR03], checking if efficient Euclidean TSP solvers can be applied [Aro98], and more recently, computing a low-rank approximation in sublinear time

[5] See Theorem 6.1.1 of [Tro15], applied to $S_k = a_{(k)}(a_{(k)})^\top$, where $a_{(k)}$ is the k-th row sampled in $\mathbf{A}$; for the case of residual error, one equivalently applies matrix Bernstein inequality to estimate the head $\sum_{i \leq k} \sigma_i^2(\mathbf{A})$. These bounds can be tightened via the usage of interior Chernoff bounds [GT11].

[BW18, IVWW19]. It is well known (Schoenberg's criterion [Dat10]) that given a distance matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ such that $\mathbf{D}_{i,j} = d(x_i, x_j)$, the points are isometrically embeddable into Euclidean space if and only If $\mathbf{G} = \mathbf{1} \cdot \mathbf{D}_{1,*} + \mathbf{D}_{1,*}^\top \cdot \mathbf{1}^\top - \mathbf{D} \succeq 0$, where $\mathbf{D}_{1,*}$ is the first row of $\mathbf{D}$. Notice that embeddability is scale invariant, allowing one to scale distances to ensure boundedness. Furthermore, since our algorithms sample submatrices and check for non-positive semi-definiteness, the tester need not know this scaling in advance, and gives guarantees for distinguishing definiteness if the necessary gap is satisfied after hypothetically scaling the entries.

Corollary 9.1.2 (Euclidean Embeddability of Finite Metrics)**.** *Given a finite metric d on n points $\{x_1, \ldots, x_n\}$, let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the corresponding distance matrix, scaled so that $\|\mathbf{D}\|_\infty \leq 1/3$, and let $\mathbf{G} = \mathbf{1}\mathbf{D}_{1,*} + \mathbf{D}_{1,*}^\top \mathbf{1}^\top - \mathbf{D}$. Then if $\min_{\mathbf{B} \succeq 0} \|\mathbf{G} - \mathbf{B}\|_F^2 \geq \epsilon n^2$, there exists an algorithm that queries $\widetilde{O}(1/\epsilon^4)$ entries in $\mathbf{A}$ and with probability $9/10$, determines the non-embeddability of $\{x_1, \ldots, x_n\}$ into Euclidean space. Further, the algorithm runs in time $\widetilde{O}(1/\epsilon^{2\omega})$.*

Remark 111. An intriguing question is to characterized geometric properties of finite metrics based on the ℓ_2^2-distance of the Schoenberg matrix $\mathbf{G}$ from the PSD cone. For instance, given a finite metric with Schoenberg matrix $\mathbf{G}$ that is close to being PSD in ℓ_2^2-distance, can we conclude that the metric has a low worst or average case distortion embedding into Euclidean space?

Remark 112. Since rescaling entries to be bounded only affects the gap parameter ϵ, in both of the above cases, so long as the magnitude of the entries in $\mathbf{X}, \mathbf{D}$ do not scale with n, the running time of our algorithms is still sublinear in the input.

Finally, several recent works have focused on obtaining sublinear time algorithms for low-rank approximation when the input matrix is PSD [MW17, BCW20]. However, such algorithms only succeed when the input is PSD or close to PSD (in ℓ_2^2), and it is unknown how to verify whether these algorithm succeeded in sublinear time. Therefore, our tester can be used as a pre-processing step to determine input instances where the aforementioned algorithms provably will (or will not) succeed.

9.1.4 Open Problems and a Remark on the ℓ_2^2-Gap.

In this chapter, we present an optimal (up to $\log(1/\epsilon)$ factors) algorithm for testing if a matrix is PSD, or far in spectral norm distance from the PSD cone. In addition, we present query efficient algorithm for the more general ℓ_2^2 gap problem. However, there is still a gap of $1/\epsilon^2$ between the upper and lower bounds for PSD testing with ℓ_2^2 gap.

We note here that there appear to be several key barriers to improving the query complexity of PSD testing with ℓ_2^2-gap beyond $O(1/\epsilon^4)$, which we briefly discuss here. First, in general, to preserve functions of the squared singular values of $\mathbf{A}$ up to error ϵn^2, such as $\|\mathbf{A}\|_F^2 = \sum_i \sigma_i^2(\mathbf{A})$ or $\|\mathbf{A}\|_2^2 = \sigma_1^2(\mathbf{A})$, any algorithm which samples a submatrix must make $\Omega(1/\epsilon^4)$ queries (see Lemma 9.5.15 for estimating $\sum_{i \le k} \sigma_i^2$ for any k). In other words, detecting ϵn^2-sized perturbations in the spectrum of a matrix in general requires $\Omega(1/\epsilon^4)$ sized submatrices. This rules out improving the query complexity by detecting the ϵn^2 negative mass in $\mathbf{A}$ via, for instance, testing if the sum of squares of top $k = 1/\epsilon$ singular values has $\Theta(\epsilon n^2)$ less mass than it should if $\mathbf{A}$ were PSD (even this may require $\Omega(k^2/\epsilon^4)$ queries, see the discussion in Section 9.2).

The key issue at play in the above barrier appears to be the requirement of sampling submatrices. Indeed, notice for the simplest case of $\|\mathbf{A}\|_F^2$, we can easily estimate $\|\mathbf{A}\|_F^2$ to additive ϵn^2 via $O(1/\epsilon^2)$ queries to random entries of $\mathbf{A}$. On the other hand, if these queries must form a submatrix, then it is easy to see that $\Omega(1/\epsilon^4)$ queries are necessary, simply from the problem of estimating $\|\mathbf{A}\|_F^2$ whose rows (or columns) have values determined by a coin flip with bias either equal to $1/2$ or $1/2 + \epsilon$. On the other hand, for testing positive semi-definiteness, especially with one-sided error, the requirement of sampling a principal submatrix seems unavoidable.

In addition, a typical approach when studying spectral properties of submatrices is to first pass to a random row submatrix $\mathbf{A}_{S \times [n]}$, argue that it preserves the desired property (up to scaling), and then iterate the process on a column submatrix $\mathbf{A}_{S \times T}$. Unfortunately, these types of arguments are not appropriate when dealing with eigenvalues of $\mathbf{A}$, since after passing to the rectangular matrix $\mathbf{A}_{S \times [n]}$, any notion of negativity of the eigenvalues has now been lost. This forces one to argue indirectly about functions of the singular values of $\mathbf{A}_{S \times [n]}$, returning to the original difficulty described above. We leave it as an open problem to determine the exact non-adaptive query complexity of PSD testing with ℓ_2^2-gap. Indeed, it seems that perhaps the main tool that is lacking is a concentration inequality for the eigenvalues of random principal submatrices. Since most such decay results apply only to norms [Tro08, RV07], progress in this direction would likely result in important insights into eigenvalues of random matrices.

In addition to our techniques for analyzing the eigenvalues of random submatrices, an important contribution of this chapter is a new technique for proving lower bounds based on designing "subgraph-equivelant" matrices (Section 9.5). This technique is quite general, as shown by its immediate application to lower bounds for the Schatten-1 norm, Ky-Fan norm, and tail error testing. Our construction could also likely be useful for proving lower bounds against testing of *graph properties*, which is a well studied area [Gol10]. We pose the open problem to design

(or demonstrate the non-existence of) additional subgraph-equivalent matrices beyond the cycle graph construction utilized in this work, which have gaps in their spectral or graph-theoretic properties.

Finally, we note that the complexity of the testing problems for several matrix norms, specifically the Schatten p and Ky-Fan norms, are still open in the bounded entry model. In particular, for the Schatten 1 norm, to the best of our knowledge no non-trivial algorithms exist even for estimation with additive error $\Theta(n^{1.5})$, thus any improvements would be quite interesting.

9.2 Overview of the Testing Algorithms

In this section, we describe the techniques used in our non-adaptive testing algorithms for the ℓ_∞ and more general ℓ_2^2 gap problem, as well as the techniques involved in our lower bound construction for the ℓ_2^2-gap.

PSD Testing with ℓ_∞ Gap

Recall in the general statement of the ℓ_∞-gap problem, our task is to distinguish between $\mathbf{A} \in \mathbb{R}^{n\times n}$ satisfying $x^\top \mathbf{A} x \geq 0$ for all $x \in \mathbb{R}^n$, or $x^\top \mathbf{A} x \leq -\epsilon n$ for some unit vector $x \in \mathbb{R}^n$. Since if $x^\top \mathbf{A} x \geq 0$ for all $x \in \mathbb{R}^n$ the same holds true for all principal submatrices of $\mathbf{A}$, it suffices to show that in the ϵ-far case we can find a $k \times k$ principal submatrix $\mathbf{A}_{T\times T}$ such that $y^\top \mathbf{A}_{T\times T} y < 0$ for some $y \in \mathbb{R}^k$.[6]

Warmup: A $O(1/\epsilon^3)$ query algorithm. Since we know $x^\top \mathbf{A} x \leq -\epsilon n$ for some fixed x, one natural approach would be to show that the quadratic form with the *same* vector x, projected onto to a random subset $T \subset [n]$ of its coordinates, is still negative. Specifically, we would like to show that the quadratic form $\mathcal{Q}_T(x) = x_T^\top \mathbf{A}_{T\times T} x_T$, of x with a random principal submatrix $\mathbf{A}_{T\times T}$ for $T \subset [n]$ will continue to be negative. If $\mathcal{Q}_T(x) < 0$, then clearly $\mathbf{A}_{T\times T}$ is not PSD. Now while our algorithm does not know the target vector x, we can still analyze the concentration of the scalar random variable $\mathcal{Q}_T(x)$ over the choice of T, and show that it is negative with good probability.

Proposition 9.3.6 and Lemma 9.3.7 (informal) *Suppose $\mathbf{A} \in \mathbb{R}^{n\times n}$ satisfies $\|\mathbf{A}\|_\infty \leq 1$ and $x^\top \mathbf{A} x \leq -\epsilon n$ where $\|x\|_2 \leq 1$. Then if $k \geq 6/\epsilon$, and if $T \subset [n]$ is a random sample of expected*

[6] This can be efficiently checked by computing the eigenvalues of $\mathbf{A}_{T\times T} + \mathbf{A}_{T\times T}^\top$.

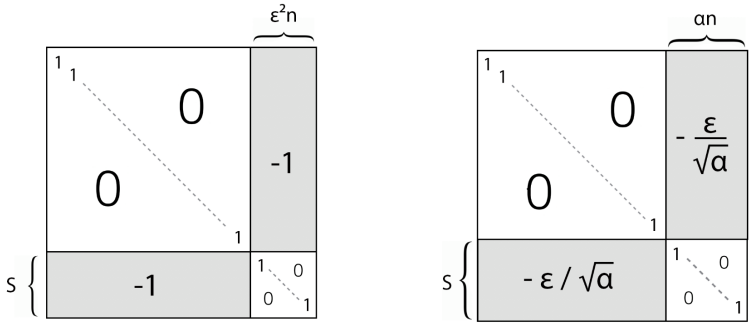


Figure 9.1: Hard instances for ℓ_∞ testing. On the left, the negative mass is highly concentrated in $|S| = \epsilon^2 n$ rows and columns, and on the right it more spread out over $|S| = \alpha n$, where $\epsilon^2 \leq \alpha \leq \epsilon$.

size k, we have $\mathbf{E}\left[\mathcal{Q}_T(x)\right] \leq -\frac{\epsilon k^2}{4n}$ *and Var*$(\mathcal{Q}_T(x)) \leq O(\frac{k^3}{n^2})$.

By the above Proposition, after setting $k = \Theta(1/\epsilon^2)$, we have that $|\,\mathbf{E}\left[\mathcal{Q}_T(x)\right]\,|^2 = \Omega(\text{Var}(\mathcal{Q}_T(x))$, and so by Chebyshev's inequality, with constant probability we will have $\mathcal{Q}_T(x) < 0$. This results in a $k^2 = O(1/\epsilon^4)$ query tester. To improve the complexity, we could instead set $k = \Theta(1/\epsilon)$ and re-sample T for k times independently to reduce the variance. Namely, one can sample submatrices $T_1, T_2, \ldots, T_k$, and analyze $\frac{1}{k}\sum_{i=1}^{k} \mathcal{Q}_{T_i}(x)$. The variance of this sum goes down to $O(\frac{k^2}{n^2})$, so, again by Chebyshev's inequality, the average of these quadratic forms will be negative with constant probability. If this occurs, then at least one of the quadratic forms must be negative, from which we can conclude that at least one of $\mathbf{A}_{T_i \times T_i}$ will fail to be PSD, now using only $O(1/\epsilon^3)$ queries.

A Family of Hard Instances One could now hope for an even tighter analysis of the concentration of $\mathcal{Q}_T(x)$, so that $O(1/\epsilon^2)$ total queries would be sufficient. Unfortunately, the situation is not so simple, and in fact the two aforementioned testers are tight in the query complexity for the matrix dimensions they sample. Consider the hard instance $\mathbf{A}$ in the left of Figure 9.1, which is equal to the identity on the diagonal, and is zero elsewhere except for a small subset $S \subset [n]$ of $|S| = \epsilon^2 n$ rows and columns, where we have $\mathbf{A}_{S \times \overline{S}} = \mathbf{A}_{\overline{S} \times S} = -\mathbf{1}$, where $\overline{S}$ is the complement of S. Notice that if we set $x_i^2 = 1/(2n)$ for $i \notin S$ and $x_i^2 = 1/(2\epsilon^2 n)$ for $i \in S$, then $x^\top \mathbf{A} x \leq -\epsilon n/4$. However, in order to even see a single entry from S, one must sample from at least $\Omega(1/\epsilon^2)$ rows or columns. In fact, this instance itself gives rise to a $\Omega(1/\epsilon^2)$ lower bound for any testing algorithm, even for adaptive algorithms (Theorem 120).

The difficulty of the above instance is that the negative mass of $x^\top \mathbf{A} x$ is hidden in only a ϵ^2-fraction of $\mathbf{A}$. On the other hand, since the negative entries are so large and concentrated, one need only sample $O(1)$ entries from a single row $i \in S$ in order for $\mathbf{A}_{T\times T}$ to be non-PSD in the prior example. Thus, an algorithm for such instances would be to sample $O(1/\epsilon^2)$ principal submatrices, each of *constant* size. On the other hand, the set S could also be more spread out; namely, we could have $|S| = \alpha n$ for any $\epsilon^2 \leq \alpha \leq \epsilon$, but where each entry in $\mathbf{A}_{S\times\overline{S}}$ is set to $-\epsilon/\sqrt{\alpha}$ (see the matrix in the right side of Figure 9.1). If instead, we define $x_i^2 = 1/(2\alpha n)$ for $i \in S$, we still have $x^\top \mathbf{A} x < -\epsilon n/4$. However, now any submatrix $\mathbf{A}_{T\times T}$ with $|T \cap S| = 1$ must have at least $|T| \geq \alpha/\epsilon^2$ rows and columns, otherwise $\mathbf{A}_{T\times T}$ would be PSD due to the identity on the diagonal.

The aforementioned instances suggest the following approach: query matrices at $O(\log \frac{1}{\epsilon})$ different scales of subsampling. Specifically, for each $\epsilon^2 \leq \alpha = 2^i \leq \epsilon$, we sample $\tilde{O}(\frac{\epsilon^2}{\alpha^2})$ independent $k \times k$ submatrices, each of size $k = \tilde{O}(\alpha/\epsilon^2)$, giving a total complexity of $\tilde{O}(\frac{1}{\epsilon^2})$. The analysis now proceeds by a complete characterization of the ways in which $x^\top \mathbf{A} x$ can be negative. Specifically, we prove the following: either a substantial fraction of the negative mass is hidden inside of a small set of rows and columns S with $|S| < \epsilon n$, or it is the case that $\mathrm{Var}(\mathcal{Q}_T(x))$ is small enough so that a single $k \times k$ submatrix will already be non-PSD with good probability when $k \gtrsim 1/\epsilon$. Given this classification, it suffices to demonstrate a level of subsampling which will find a non-PSD submatrix when the negative mass is concentrated inside inside a small set S.

Eigenvector Switching. To analyze this case, ideally, one would like to demonstrate that conditioned on T intersecting S at some level of subsampling, we will have $\mathcal{Q}_T(x) < 0$ with good probability. Unfortunately, the approach of analyzing the quadratic form with respects to x will no longer be possible; in fact, $\mathcal{Q}_T(x)$ may never be negative conditioned on $|T \cap S| = 1$ (unless $|T| > 1/\epsilon$, which we cannot afford in this case). The complication arises from the fact that the coordinates of x_i in the small set S can be extremely large, and thus the diagonal contribution of $x_i^2 \mathbf{A}_{i,i}$ will dominate the quadratic form of a small submatrix. For instance, if $\mathbf{A}_{T\times T}$ is a sample with $k = |T| = O(1)$ which intersects the set S in the leftmost matrix in Figure 9.1, where $x_i = 1/(\epsilon\sqrt{n})$ for $i \in S$ and $x_i = 1/\sqrt{n}$ otherwise, then $\mathcal{Q}_T(x) \approx k/n - (k/\sqrt{n})x_i + \mathbf{A}_{i,i}x_i^2$, which is dominated by the diagonal term $\mathbf{A}_{i,i}x_i^2 = 1/(\epsilon^2 n)$. Thus, while $\mathbf{A}_{T\times T}$ itself is not PSD, we have that $\mathcal{Q}_T(x) > 0$.

To handle this, we must and analyze the quadratic form $\mathcal{Q}_T(\cdot)$ with respect to *another* direction y. The vector y may not even satisfy $y^\top \mathbf{A} y < 0$, however conditioned on $|T \cap S| \geq 1$, we will have $\mathcal{Q}_T(y) < 0$ with good probability. Clearly, we must scale down the large coordinates

x_i for $i \in S$. However, one cannot scale too low, otherwise the negative contribution of the rows $i \in S$ would become too small. The correct scaling is then a careful balancing act between the contributions of the different portions of $\mathbf{A}_{T\times T}$. Informally, since the x_i's for $i \in S$ make up a $|S|/n$ fraction of all coordinates, they can be as large as $x_i^2 \geq (n/|S|) \cdot (1/n)$. However, inside of the smaller submatrix $\mathbf{A}_{T\times T}$, then conditioned on $i \in T$, since $|T|$ is small x_i now makes up a larger $1/|T|$ fraction of the submatrix, thus we should scale down x_i to only be $x_i^2 \approx |T|/n$. With this scaling in mind, we (very roughly) set $y_i^2 = (|S|/n) \cdot (|T|/n)$ if $i \in S$, and set $y_i = x_i$ otherwise. The remaining argument then requires a careful analysis of the contribution of entries of $\mathbf{A}$ outside of S to show that the target vector y indeed satisfies $\mathcal{Q}_T(y) < 0$ with good probability conditioned on T intersecting S.

PSD Testing with ℓ_2 Gap

Recall in the ℓ_2 gap problem, our task is to distinguish between $\mathbf{A}$ being PSD, and $\mathbf{A}$ being ϵ-far in ℓ_2^2 distance from any PSD matrix, namely that $\sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{A}) > \epsilon n^2$. In what follows, we refer to the quantity $\sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{A})$ as the *negative mass* of $\mathbf{A}$. First observe that in the special case that we had a "large" negative eigenvalue, say $\lambda_{\min}(\mathbf{A}) = -\sqrt{\epsilon}n$, then by applying our testing algorithm for ℓ_∞-gap, we could find a non-PSD submatrix with only $\tilde{O}(1/\epsilon)$ queries. However, in general the negative mass of $\mathbf{A}$ may be spread out over many smaller eigenvalues. Thus, we cannot hope to apply our earlier approach for the ℓ_∞-gap, which preserved the quadratic form $\mathcal{Q}_T(x) = x_T^\top \mathbf{A}_{T\times T} x_T$ with respects to a fixed direction x. Instead, our approach will be to show that if $\mathbf{A}$ is ϵ-far from PSD in ℓ_2^2, then the singular values of $\mathbf{A}$ must be "far" from PSD, in some other notion of distance, allowing us to indirectly infer the existence of negative eigenvalues in submatrices.

PSD matrices are top-heavy, and a reduction to estimating the tail. Our first step is to show that if $\mathbf{A} \in \mathbb{R}^{n\times n}$ is PSD, then the t-"tail" of $\mathbf{A}$, defined as $\sum_{i>t} \sigma_i^2(\mathbf{A})$, cannot be too large. This can be derived from the following fact: if $\mathbf{A}$ is PSD then we can bound the Schatten-1 norm of $\mathbf{A}$ by $\|\mathbf{A}\|_{\mathcal{S}_1} = \sum_i \sigma_i(\mathbf{A}) = \mathrm{Tr}(\mathbf{A})$, which is at most n if $\|\mathbf{A}\|_\infty \leq 1$. This simple fact will prove highly useful, since whenever we can demonstrate that the Schatten-1 norm of a submatrix $\mathbf{A}_{T\times T}$ is larger than $|T|$, we may immediately conclude the that $\mathbf{A}_{T\times T}$ is not PSD. In addition, it implies:

Proposition 9.4.2 (PSD matrices are top-heavy) *Fix any $n \in \mathbb{N}$, $1 \leq t \leq n$, and $\mathbf{D} \in \mathbb{R}^{n\times n}$.*

Then if $\mathbf{D}$ *is PSD, we have*

$$\sum_{i>t} \sigma_i(\mathbf{D})^2 \leq \frac{1}{t} \left(Tr(\mathbf{D})\right)^2$$

In particular, if $\mathbf{D}$ *has bounded entries* $\|\mathbf{D}\|_\infty \leq 1$*, we have* $\sum_{i>t} \sigma_i(\mathbf{D})^2 \leq \frac{1}{t} n^2$.

On the other hand, suppose that $\mathbf{A}$ is ϵ-far from PSD, and let $t > 10/\epsilon$. Then if no eigenvalue is smaller than $-\epsilon n/100$, a condition which can be checked with $\tilde{O}(1/\epsilon^2)$ queries by first running our ℓ_∞-gap tester, then the negative mass must be spread out, and it must be the case that a substantial fraction of the negative mass of $\mathbf{A}$ is contained in the bottom $n - t$ singular values. Specifically, we must have $\sum_{i>t} \sigma_i(\mathbf{A})^2 > (\epsilon/2)n^2$, whereas any PSD matrix $\mathbf{D}$ would have to satisfy $\sum_{i>t} \sigma_i^2(\mathbf{D}) \leq (\epsilon/10)n^2$ by the above Proposition. Thus, after first running our ℓ_∞ tester, it will suffices to estimate the tail $\sum_{i>t} \sigma_i^2(\mathbf{A})$. Equivelantly, since $\|\mathbf{A}\|_F^2 = \sum_i \sigma_i^2(\mathbf{A})$ can be efficiently estimated, it also suffices to estimate the "head" $\sum_{i\leq t} \sigma_i^2(\mathbf{A})$ to additive $O(\epsilon n^2)$.

In order to accomplish this, one could utilize the tools from random matrix concentration, such as Matrix Bernstein's inequality [Tro15], which allows one to estimate each σ_i^2 to error ηn^2 by taking a random rectangular $O(1/\eta^2) \times O(1/\eta^2)$ sized submatrix. The error in estimating $\sum_{i\leq t} \sigma_i^2(\mathbf{A})$ is then $t\eta n^2$, thus one needs to set $\eta = O(\epsilon/t)$, giving a $O(1/\epsilon^8)$ tester with two-sided error. Using a careful bucketing analysis on the error, along with the more powerful Interior Matrix Chernoff bounds of Gittens and Tropp [GT11], one can improve this to $O(t^2/\epsilon^4) = O(1/\epsilon^6)$. However, substantial improvements on unconditional estimation of $\sum_{i\leq t} \sigma_i^2(\mathbf{A})$ seem unlikely. In fact, we demonstrate that event for $t = 1$ (spectral norm estimation), tools such as matrix concentration inequalities which sample submatrices of $\mathbf{A}$, must make $\Omega(1/\epsilon^4)$ queries (Lemma 9.5.15), which rules out, for instance, a $o(t^2/\epsilon^4)$ upper bound for general t. Thus, instead of unconditional estimation, our main insight is to demonstrate conditions under which $\sum_{i\leq t} \sigma_i^2(\mathbf{A})$ can be efficiently estimated. When these conditions do not hold, we show that it is because the Schatten-1 norm of our sampled submatrix must be too large, from which we can deduce the existence of negative eigenvalues in our query.

In the first case, if the t-th singular value is not too large, say $\sigma_{t+1}(\mathbf{A}) \leq 10n/t$, we show that the (re-scaled) tail $\frac{n^2}{k^2} \sum_{i>t} \sigma_i^2(\mathbf{A}_{S\times T})$ of a random rectangular matrix, where $|S| = |T| = k = O(1/\epsilon^2)$, approximates the tail of $\mathbf{A}$ to error $O(\epsilon n^2)$. Our argument relies on splitting $\mathbf{A}$ into head and tail pieces $\mathbf{A} = \mathbf{A}_t + \mathbf{A}_{-t}$, where $\mathbf{A}_t$ is $\mathbf{A}$ projected onto the top-t eigenvectors of $\mathbf{A}$. We demonstrate that the spectral mass of each is preserved after passing to a random row submatrix, and additionally demonstrate that $\sigma_{\max}(\mathbf{A}_{-t}) = \sigma_{t+1}(\mathbf{A})$ does not grow too much using spectral decay inequalities for random submatrices [RV07]. This forces the spectrum of $(\mathbf{A}_{-t})_{S\times[n]}$ to be well spread out, allowing us to apply interlacing inequalities to demonstrate

that after adding $(\mathbf{A}_t)_{S\times[n]}$ back in, the resulting tail is still sufficiently large, and then iterate the argument when sampling columns to obtain $\mathbf{A}_{S\times T}$.

On the other hand, if $\sigma_{t+1}(\mathbf{A})$ is too large, then after moving to a random row submatrix the spectral norm of $\mathbf{A}_{-t}$ can concentrate highly in its top eigenvalues, which can then be absorbed by the top t eigenvalues of $\mathbf{A}_t$, stealing too much mass from the tail. Instead, note that if $\sigma_{t+1}(\mathbf{A}) \geq 10n/t$, then the Schatten norm of $\mathbf{A}$ must be large, namely $\sum_i \sigma_i(\mathbf{A}) > 10n$, which cannot occur if $\mathbf{A}$ is PSD. We show that by applying Interior Eigenvalue Matrix Chernoff bounds (mentioned above), we can preserve this fact, obtaining $\frac{n}{k}\sigma_{t+1}(\mathbf{A}_{S\times T}) > 10n/t$ with good probability when $k = \Omega(1/\epsilon^2)$. If this is the case, then the Schatten norm of the submatrix will be too large: $\|\mathbf{A}_{S\times T}\|_{\mathcal{S}_1} \geq t(10k/t) > 10k$. To obtain a certificate from this fact, we move to the larger principal submatrix $\mathbf{A}_{(S\cup T)\times(S\cup T)}$, which we show must still have large Schatten norm, from which we can infer the existence of negative eigenvalues. Similarly, in the earlier case, we show that the large tail of $\mathbf{A}_{S\times T}$ implies that the principal submatrix $\mathbf{A}_{(S\cup T)\times(S\cup T)}$ also has too large of a tail, meaning it must not be PSD.

Lower Bounds

As seen above, the distribution of negative mass in the matrix $\mathbf{A}$ plays an important role in the complexity of testing if $\mathbf{A}$ is PSD. Specifically, the problem becomes easier the more concentrated the negative mass is within a few eigenvalues. So in order to avoid a $o(1/\epsilon^2)$ upper bound from the ℓ_∞-testing algorithm, our hard instance must have $|\lambda_{\min}(\mathbf{A})| = O(\epsilon n)$ in the ϵ-far case. On the other hand, we cannot allow the negative mass to be extremely spread out, otherwise we would have to add many more *positive* eigenvalues to avoid violating the trace constraint $|\mathrm{Tr}(\mathbf{A})| = |\sum_i \lambda_i(\mathbf{A})| \leq n$ implied by the boundedness, creating further spectral differences between the instances. With this in mind, our hard distribution will have $1/\epsilon$ negative eigenvalues, each roughly equal to $\lambda_i(\mathbf{A}) = -\epsilon n$.

The Hard Instance. Our first insight is to construct a discrete instance, with the property that the distribution induced by observing a small sample of the "meaningful" entries of $\mathbf{A}$ is *identical* in both cases. Specifically, we construct two distribtuions: $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$ over $n \times n$ matrices. In both cases, $\mathbf{A}$ will be block diagonal, with k disjoint blocks $B_1, B_2, \ldots, B_k \subset [n]$, each of size $|B_i| = n/k$, for some parameter k; we will later set $k = \Theta(1/\epsilon)$, so our target lower bound is $\Omega(k^2)$. In $\mathcal{D}_{\text{YES}}$, each block $\mathbf{A}_{B_i\times B_i}$ will be PSD, whereas in $\mathcal{D}_{\text{NO}}$ we will have $\lambda_{\min}(\mathbf{A}_{B_i\times B_i}) = -\tilde{\Theta}(n/k) \approx -\epsilon n$. The partition $B_1 \cup B_2 \cup \cdots \cup B_k = [n]$ is chosen randomly, so that for any fixed set of samples, only a small fraction them will be contained inside any block

$\mathbf{A}_{B_i \times B_i}$. The diagonal entries will always be fixed to 1, and all off-diagonal entries are either $\{0, 1, -1\}$. The samples $a_1, a_2, \ldots, a_s \in [n] \times [n]$ of any algorithm can then be interpreted as a graph H (possibly with self-loops), where for each edge $a_r = (i, j) \in E(H)$, the algorithm learns the value $\mathbf{A}_{i,j} \in \{0, 1, -1\}$.

Now consider the algorithm which just samples a $t \times t$ principal submatrix $T \subset [n]$, so that H is a t-clique. Now in expectation $\mathbf{E}[|T \cap B_i|] = \frac{t}{k}$ for each i, however, by a balls and bins argument, as t approaches k we will obtain some blocks i with $|T \cap B_i| = \Omega(\log k / \log \log k)$. Thus, to fool this query, we must be able to "fool" cliques of size roughly $\log k$ within a block B_i. On the other hand, an algorithm could find many more entries in a block by lop-sided sampling: for instance, it could sample k^2 entries in a single column of $\mathbf{A}$ (H is a k^2-star), getting k entries inside a column of a block $\mathbf{A}_{B_i \times B_i}$. Thus we must also fool large star queries. It turns out that the right property to consider is the *matching number* $\nu(H)$ of the query graph H, i.e. the size of a maximum matching. Notice for a star H, we have $\nu(H) = 1$. We prove (roughly) that if within each block B_i, one can "fool" every query graph H inside $\mathbf{A}_{B_i \times B_i}$ with matching number $\nu(H) < \ell$, one would obtain a lower bound of $\Omega(k^{\frac{2(\ell-1)}{\ell}})$. Thus, it will suffice to fool all query graphs H within a block B_i with $\nu(H) \leq \log k$.

For a first step towards this, suppose that in $\mathcal{D}_{\text{YES}}$, we set each block independently to $\mathbf{A}_{B_i \times B_i} = vv^\top$, where $v \in \{1, -1\}^{|B_i|}$ is a random sign vector, and in $\mathcal{D}_{\text{NO}}$, we set $\mathbf{A}_{B_i \times B_i} = -vv^\top$ (except we fix the diagonal to be 1 in both cases). Now notice that the distribution of any individual entry $(\mathbf{A}_{B_i \times B_i})_{a,b}$ is symmetric, and identical in both $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$. Furthermore, it is not difficult to check that the distribution of a path or star query H inside of $\mathbf{A}_{B_i \times B_i}$ is also identical in both cases. On the other hand, if H contained a *triangle*, then this would not be the case, since in $\mathcal{D}_{\text{YES}}$ one could never have a negative cycle (x, y, z) where $v_x v_y = v_y v_z = v_z v_x = -1$, whereas this could occur in $\mathcal{D}_{\text{NO}}$, since we could have that $-v_x v_y = -v_y v_z = -v_z v_x = -1$. Thus, roughly, to distinguish between these distributions $\mathcal{D}_{\text{YES}}$ from $\mathcal{D}_{\text{NO}}$, an algorithm must sample a triangle within one of the blocks $\mathbf{A}_{B_i \times B_i}$, which one can show requires $\Omega(k^{4/3})$ queries, yielding a first lower bound.[7]

Boosting to $\Omega(k^2)$. Given the above example, we would now like to construct instances which fool H with larger and larger $\nu(H)$. In fact, our next insight is to have an even simpler structure on $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$: each of them will be a random permutation of one of two *fixed* matrices $\mathbf{D}_1, \mathbf{D}_2$ respectively. We now formalize the "fooling" condition we need. For a matrix $\mathbf{B}$ and a query graph H, let $(\mathbf{B})_H$ denote the result of setting all entries of $\mathbf{B}$ not in H equal to zero. Then the matrices $\mathbf{D}_1, \mathbf{D}_2$ must have the property that for any graph H with $\nu(H) \leq \log k$, if $\sigma : [m] \to$

[7]Note that $\nu(H) = 1$ for a triangle H, so the $\Omega(k^{2(\ell-1)/\ell})$ lower bound when $\nu(H) < \ell$ is actually loose here.

$[m]$ is a random permutation and $\mathbf{P}_\sigma \in \mathbb{R}^{m\times m}$ is the row permutation matrix corresponding to σ, then the distribution of $(\mathbf{P}_\sigma \mathbf{D}_1 \mathbf{P}_\sigma^\top)_H$ is identical to the distribution $(\mathbf{P}_\sigma \mathbf{D}_2 \mathbf{P}_\sigma^\top)_H$. We call this property *H-subgraph equivalence*. This implies that any algorithm which queries the edges in H inside of $\mathbf{P}_\sigma \mathbf{D}_1 \mathbf{P}_\sigma^\top$ or $\mathbf{P}_\sigma \mathbf{D}_2 \mathbf{P}_\sigma^\top$ will be unable to distinguish between them with any advantage. To obtain a lower bound, we must also have a gap between $\lambda_{\min}(\mathbf{D}_1)$ and $\lambda_{\min}(\mathbf{D}_2)$, so that their spectrum can be shifted to make one PSD and the other far. Furthermore, neither $\lambda_{\min}(\mathbf{D}_1)$ or $\lambda_{\min}(\mathbf{D}_2)$ can be too negative, otherwise by shifting we would lose boundedness of the entries.

A priori, it is not even clear that such matrices $\mathbf{D}_1, \mathbf{D}_2$ exist, even for fixed values of $\nu(H)$, such as $\nu(H) = 5$. Our main contribution now is to demonstrate their existence for every $\nu(H)$. Our construction is simple, but perhaps surprisingly so. Both $\mathbf{D}_1, \mathbf{D}_2$ will be adjacency matrices; in the PSD case, we set $\mathbf{D}_1$ to be the cycle graph C_{2m+1} on $2m+1 = \Theta(\log k)$ vertices, and in the ϵ-far case we set $\mathbf{D}_2$ to be the disjoint union of two cycles $C_{m+1} \oplus C_m$. Since one of m and $m+1$ is even, while $2m+1$ is odd, we will have that $\lambda_{\min}(C_{m+1} \oplus C_m) = -2$, but $\lambda_{\min}(C_{2m+1}) > -2$.[8] To show subgraph equivalence, it suffices to show a slightly more general version of the following: for any graph H with $\nu(H) < m/4$, the number of subgraphs of C_{2m+1} isomorphic to H is the same as the number of subgraphs of $C_{m+1} \oplus C_m$ isomorphic to H.[9] Note that if $\nu(H) < m/4$, then H is just a disjoint collection of paths.

Our proof of this fact is by a construction of a bijection from arrangements of H in C_{2m+1} to H in $C_{m+1} \oplus C_m$. While a seemingly simple property, some care must be taken when designing a bijection. Our mapping involves first "swapping" two paths (whose length depends on H) in C_{2m+1}, before "splitting" C_{2m+1} into two cycles of length m and $m+1$. We direct the reader to Section 9.5.2 for further details.

Amplifying the Gap. The subgraph equivalence between C_{2m+1} and $C_{m+1} \oplus C_m$ prevents any algorithm from distinguishing between them with a small number of samples, however the gap in the minimum eigenvalue shrinks at the rate of $\Theta(1/m^2)$. Meaning, if we set $\gamma = \lambda_{\min}(C_{2m+1}) = 2 - \Theta(1/m^2)$, while the matrix $\gamma\mathbf{I} + C_{2m+1}$ is PSD and has constant sized entries, we only have $\lambda_{\min}(\gamma\mathbf{I} + C_{m+1} \oplus C_m) = -\Theta(1/m^2)$, which is not far enough from PSD. Instead, recall that we only need $m = \Omega(\log k)$ to fool all H with $\nu(H) \le \log k$, but the block size which we must fill is much larger: $\mathbf{A}_{B_i \times B_i}$ has size $|B_i| = n/k$. Thus, instead of setting $m = \Theta(n/k)$ and filling all of $\mathbf{A}_{B_i \times B_i}$ with the cycles, we set $m = \Theta(\log k)$, and we amplify the spectral gap by taking the tensor product of the small graphs C_{2m+1} and $C_{m+1} \oplus C_m$ with a large, fixed matrix $\mathbf{M}$, so

[8] To intuitively see why this is true, note that if m is even and $v \in \{-1, 1\}^m$ is the vector that assigns opposite signs to adjacent vertices of C_m, then we have $C_m v = -2v$. However, if m is odd, this assignment v is no longer possible.

[9] A more general statement is needed since H can also query for edges which do not exist in C_{2m+1}.

that $(\gamma \mathbf{I} + C_{2m+1}) \otimes \mathbf{M}$ has $|B_i|$ rows and columns. We prove that taking the tensor product with any fixed $\mathbf{M}$ preserves the subgraph equivalence properties of the original matrices. From here, our lower bounds for testing PSD with ℓ_2 gap, Schatten norms, Ky fan, and the cost of the best rank-k approximation, all follow by a proper choice of $\mathbf{M}$. For PSD testing, we can choose $\mathbf{M} = \mathbf{1}$ to be the all 1's matrix, and to amplify the gap in Schatten 1 norm, we can choose $\mathbf{M}$ to be a random Rademacher matrix. Since $\mathbf{M} = \mathbf{1}$ is PSD and $\|\mathbf{M}\|_2 = \widetilde{\Omega}(n/k)$, the gap is amplified to the desired $-\widetilde{\Omega}(n/k)$. Finally, we remark that to obtain a lower bound for another norm, any matrix $\mathbf{M}$ which is large in that norm may be suitable, so long as the original sub-graph equivalent matrices also have a gap in that norm. We pose it as an interesting open problem to design other pairs of matrices $\mathbf{D}_1, \mathbf{D}_2$ with different spectral gaps which have good sub-graph equivalence properties.

9.3 PSD Testing with ℓ_∞ Gap

In this section, we introduce our algorithm for the PSD testing problem with ℓ_∞-gap. As discussed earlier, we consider a more general version of the ℓ_∞ gap than the definition presented in Problem 1, which allows one to test a notion of positive semi-definitness which applies to non-symmetric matrices as well. Specifically, we define the *PSD* case as when $x^\top \mathbf{A} x \geq 0$ for all $x \in \mathbb{R}^n$, and the *far* case as when $x^\top \mathbf{A} x < -\epsilon n$ for a unit vector x. We note that if $\mathbf{A}$ is symmetric, this definition is equivalent to Problem 1. In fact, as we will see shortly, one can always reduce the non-symmetric case to the symmetric case, so this distinction will not matter algorithmically. Formally, we solve the following problem:

Definition 9.3.1 (General PSD Testing with ℓ_∞-Gap.)**.** *Fix, $\epsilon \in (0, 1]$ and let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be any matrix satisfying $\|\mathbf{A}\|_\infty \leq 1$. The goal is to distinguish between the following two cases:*

- ***YES Instance**: $\mathbf{A}$ satisfies $x^\top \mathbf{A} x \geq 0$, for all $x \in \mathbb{R}^n$.*
- ***NO Instance**: There exists a unit vector $x \in \mathbb{R}^n$ such that $x^\top \mathbf{A} x < -\epsilon n$.*

with probability at least $2/3$.

Reducing to the symmetric case In the case where $\mathbf{A}$ is symmetric, as in Problem 1, the above gap instance can be restated in terms of the minimum eigenvalue of $\mathbf{A}$. Specifically, we are promised that either $\lambda_{\min}(\mathbf{A}) \geq 0$ or $\lambda_{\min}(\mathbf{A}) \leq -\epsilon n$. However, we now observe that one can reduce to the symmetric case with only a factor of 2 loss in the query complexity, by simply

querying the symmetrization $(\mathbf{A} + \mathbf{A}^\top)/2$. First note, that for any $x \in \mathbb{R}^n$ and any matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, we have $x^\top \mathbf{A} x = x^\top \mathbf{A}^\top x$, thus for any x we have $x^\top \mathbf{A} x = x^\top \frac{\mathbf{A}+\mathbf{A}^\top}{2} x$. Thus $x^\top \mathbf{A} x \geq 0$ for all x if and only if $x^\top \frac{\mathbf{A}+\mathbf{A}^\top}{2} x \geq 0$ for all x, which occurs if and only if the matrix $\frac{\mathbf{A}+\mathbf{A}^\top}{2}$ is PSD. Similarlly, we have that $x^\top \mathbf{A} x \leq -\epsilon n$ for some unit vector x if and only if $x^\top \frac{\mathbf{A}+\mathbf{A}^\top}{2} x \leq -\epsilon n$ for some unit vector x, which occurs if and only if $\lambda_{\min}(\frac{\mathbf{A}+\mathbf{A}^\top}{2}) \leq -\epsilon n$. Note also that the matrix $\frac{\mathbf{A}+\mathbf{A}^\top}{2}$ has bounded entries $\|\frac{\mathbf{A}+\mathbf{A}^\top}{2}\|_\infty \leq 1$ if $\|\mathbf{A}\|_\infty \leq 1$. Moreover, query access to $\frac{\mathbf{A}+\mathbf{A}^\top}{2}$ can be simulated via query access to $\frac{\mathbf{A}+\mathbf{A}^\top}{2}$ with a loss of at most a factor of 2 in the query complexity, by symmetrizing the queries. In fact, our algorithms will not even incur this factor of 2 loss, since all queries our algorithms make will belong to principal submatrices of $\mathbf{A}$. Thus, in what follows, we can restrict ourselves to the original formulation as specified in Problem 1, and assume our input $\mathbf{A}$ is symmetric.

The goal of this section is now to prove the following theorem, which demonstrate the existence of a $\tilde{O}(1/\epsilon^2)$ query one-sided error tester for the above problem. In Section 9.5, we demonstrate that this complexity is optimal (up to $\log(1/\epsilon)$ factors), even for testers with two sided error (Theorem 120).

Theorem 113 (Query Optimal One-Sided Tester for ℓ_∞ Gap (see Theorem 116))**.** *There is an algorithm which, given $\mathbf{A}$ with $\|\mathbf{A}\|_\infty \leq 1$ such that either $x^\top \mathbf{A} x \geq 0$ for all x (`YES` case), or such that $x^\top \mathbf{A} x \leq -\epsilon n$ for some $x \in \mathbb{R}^n$ with $\|x\|_2 \leq 1$ (`NO` case), distinguishes the two cases with probability at least $3/4$, while making at most $\tilde{O}(\frac{1}{\epsilon^2})$ queries to the entries of $\mathbf{A}$, and runs in time $\tilde{O}(1/\epsilon^\omega)$, where $\omega < 2.373$ is the exponent of matrix multiplication. Moreover, in the first case when $x^\top \mathbf{A} x \geq 0$ for all x, is PSD, the algorithm always correctly outputs `YES`.*

Algorithmic Setup First recall that if $\mathbf{A}$ is PSD, then then every *principal* submatrix $\mathbf{A}_{T \times T}$ of $\mathbf{A}$ for $T \subseteq [n]$ is also PSD. Thus, it will suffice to query a collection of principal submatrices $\mathbf{A}_{T_1 \times T_1}, \mathbf{A}_{T_2 \times T_2}, \ldots, \mathbf{A}_{T_t \times T_t}$ of $\mathbf{A}$, and return `Not PSD` if any one of them fails to be PSD. Such a algorithm then naturally has one-sided error, since if $\mathbf{A}$ is PSD it will always return PSD. Thus, in the remainder of the section, it suffices to consider only the `NO` case, and demonstrate that, if $\mathbf{A}$ satisfies $x^\top \mathbf{A} x \leq -\epsilon n$ for some unit vector $x \in \mathbb{R}^n$, then with good probability at least one of the sampled principal submatrices will fail to be PSD.

Moreover, as shown above, it suffices to consider the case where $\mathbf{A}$ is symmetric. In this case, we will fix x to be the eigenvector associated with the smallest eigenvalue of $\mathbf{A}$. Thus, in what follows, we can fix x so that $\min_{z \in \mathbb{R}^n : \|z\|_2 = 1} z^\top \mathbf{A} z = x^\top \mathbf{A} x = \lambda_{\min}(\mathbf{A}) = -\epsilon n$. Notice here we define ϵ to satisfy the *equality* $\lambda_{\min}(\mathbf{A}) = -\epsilon n$, however our algorithm need only know a lower bound $\epsilon_0 < \epsilon$ on ϵ. The reason for this is that the input parameter ϵ_0 will only effect the sizes

of the random submatrices being sampled (smaller ϵ_0 increases the size). Thus, an algorithm run with parameter ϵ_0 can be simulated by first running the algorithm with parameter $\epsilon > \epsilon_0$, and then randomly adding rows and columns to the sampled submatrices from the correct marginal distribution. Thus, there is a coupling such that the submatrices chosen by an algorithm with any input $\epsilon_0 < \epsilon$ will always contain the submatrices sampled by an algorithm given the input exactly ϵ, so if the latter algorithm sampled a non-PSD submatrix, so would the former. Thus, for the sake of analysis, we can assume that the value ϵ is known.

Throughout the following section, we will assume $1/\epsilon < c \cdot n$ for some sufficiently small constant c. Notice that if this was not the case, we would have $1/\epsilon^2 = \Omega(n^2)$, and we would be permitted to read the entire matrix $\mathbf{A}$, as this is within our target budget of $\tilde{O}(1/\epsilon^2)$.

9.3.1 Warm-up: a $O(1/\epsilon^3)$ algorithm

We first describe a $O(1/\epsilon^3)$ query algorithm for the problem of PSD testing with ℓ_∞-gap. The general approach and results of this algorithm will be needed for the more involved $\tilde{O}(1/\epsilon^2)$ query algorithm which we shall develop in the sequel. As noted above, it suffices to analyze the NO case, where we have $x^\top \mathbf{A} x = \lambda_{\min}(\mathbf{A}) = -\epsilon n$ for a unit vector $x \in \mathbb{R}^n$. Our goal will be to analyze the random variable $Z = x_T^\top \mathbf{A}_{T\times T} x_T$, where $T \subset [n]$ is a random subset, where each $i \in [n]$ is selected independently with some probability δ. Notice that if $\delta_i \in \{0,1\}$ is an indicator variable indicating that we sample $i \in T$, then we have $Z = x_T^\top \mathbf{A}_{T\times T} x_T = \sum_{i,j} x_i \mathbf{A}_{i,j} x_j \delta_i \delta_j$.

Now, algorithmically, we do not know the vector x. However, if we can demonstrate concentration of Z, and show that $Z < 0$ for our sampled set S, then we can immediately conclude that $\mathbf{A}_{T\times T}$ is not PSD, a fact which *can* be tested. Thus, the analysis will proceed by pretending that we did know x, and analyzing the concentration of $x_T^\top \mathbf{A}_{T\times T} x_T$. In the following section, however, we will ultimately analyze the concentration of this random variable with respects a slightly different vector than x.

We first remark that we can assume, up to a loss in a constant factor in the value of ϵ, that the diagonal of $\mathbf{A}$ is equal to the identity.

Proposition 9.3.2. *We can assume $\mathbf{A}_{i,i} = 1$, for all $i \in [n]$. Specifically, by modifying $\mathbf{A}$ so that $\mathbf{A}_{i,i} = 1$, for all $i \in [n]$, the completeness (PSD) case is preserved and the soundness (not PSD) case is preserved up to a factor of $1/2$ in the parameter ϵ.*

Proof. Every time we observe an entry $\mathbf{A}_{i,i}$ we set it equal to 1. In this new matrix, if $\mathbf{A}$ was

PSD to begin with, then $\mathbf{A}$ will still be PSD, since this modification corresponds to adding a non-negative diagonal matrix to $\mathbf{A}$. If $x\mathbf{A}x \leq -\epsilon n$ originally for some $x \in \mathbb{R}^n$, then $x^\top \mathbf{A}x \leq -\epsilon n + 1$ after this change, since the diagonal contributes at most $\sum_i \mathbf{A}_{i,i}x_i^2 \leq \|x\|_2^2 \leq 1$ to the overall quadratic form. Note this additive term of 1 is at most $(\epsilon n)/2$ since we can assume $\epsilon = \Omega(1/n)$. □

We now notice that if x is the eigenvector associated with a a large enough eigenvalue, the ℓ_2 mass of x cannot be too concentrated.

Proposition 9.3.3. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix with $\lambda_{\min}(\mathbf{A}) = -\epsilon n$, and let $x \in \mathbb{R}^n$ be the (unit) eigenvector associated with $\lambda_{\min}(\mathbf{A})$. Then we have that $\|x\|_\infty \leq \frac{1}{\epsilon\sqrt{n}}$.*

Proof. By Cauchy-Schwartz, for any $i \in [n]$:

$$|\lambda_{\min}| \cdot |x_i| = |\langle \mathbf{A}_{i,*}, x\rangle| \leq \|\mathbf{A}_{i,*}\|_2 \leq \sqrt{n}$$

from which the proposition follows using $\lambda_{\min}(\mathbf{A}) = -\epsilon n$. □

Recall that our goal is to analyze the random variable $Z = x_T^\top \mathbf{A}_{T\times T}x_T = \sum_{i,j} x_i\mathbf{A}_{i,j}x_j\delta_i\delta_j$. To proceed, we bound the moments of Z. Our bound on these moments can be tightened as a function of the *row and column contributions* of the target vector x, which we now define.

Definition 9.3.4. *Fix any $y \in \mathbb{R}^n$. Then for any $i \in [n]$, define the total row and column contributions of i as $\mathcal{R}_i(y) = \sum_{j\in[n]\setminus i} y_i\mathbf{A}_{i,j}y_j$ and $\mathcal{C}_i(y) = \sum_{j\in[n]\setminus i} y_j\mathbf{A}_{j,i}y_i$ respectively.*

Notice from the above definition, we have $\sum_i \mathcal{R}_i(y) + \mathcal{C}_i(y) = 2\left(y^\top \mathbf{A}y - \sum_i \mathbf{A}_{i,i}y_i^2\right)$.

Fact 9.3.5. *Let $x \in \mathbb{R}^n$ be the eigenvector associated with $\lambda_{\min}(\mathbf{A})$. Then we have $\mathcal{R}_i(x) + \mathcal{C}_i(x) \leq 0$ for all $i \in [n]$.*

Proof. Suppose there was an i with $\mathcal{R}_i(x) + \mathcal{C}_i(x) > 0$. Then setting $z = x_{[n]\setminus i}$ we have $z^\top \mathbf{A}z = \langle x, \mathbf{A}x\rangle - (\mathcal{R}_i(x) + \mathcal{C}_i(x)) - \mathbf{A}_{i,i}(x_i)^2$. Recall from Proposition 9.3.2 that we can assume $\mathbf{A}_{i,i} = 1$ for all i, thus it follows that $z^\top \mathbf{A}z < \langle x, \mathbf{A}x\rangle$, which contradicts the optimality of x. □

We now bound the expectation of the random quadratic form.

Proposition 9.3.6 (Expectation Bound)**.** *Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a matrix with $\|\mathbf{A}\|_\infty \leq 1$, and let $y \in \mathbb{R}^n$ be any vector with $\|y\|_2 \leq 1$ and $y^\top \mathbf{A} y < -\epsilon n$. Let $Z = \sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j$, where $\delta_1, \ldots, \delta_n \sim$ Bernoulli$(\frac{k}{n})$. Then if $k \geq 8/\epsilon$, we have $\mathbf{E}[Z] \leq -\frac{\epsilon k^2}{4n}$.*

Proof. Let $c_{i,j} = \mathbf{A}_{i,j} y_i y_j$. First note, for any $i \in [n]$, the term $c_{i,j}$ is included in T with probability k/n if $i = j$, and with probability k^2/n^2 if $i \neq j$. So

$$\begin{aligned}
\mathbf{E}[Z] &= \sum_{i \neq j} \frac{k^2}{n^2} c_{i,j} + \sum_{i \in [n]} \frac{k}{n} c_{i,i} \\
&= \frac{k^2}{n^2} \left(\langle y, \mathbf{A} y \rangle - \sum_{i \in [n]} \mathbf{A}_{i,i} y_i^2 \right) + \frac{k}{n} \sum_{i \in [n]} \mathbf{A}_{i,i} y_i^2 \\
&\leq -\frac{\epsilon k^2}{2n} + \left(\frac{k}{n} + \frac{k^2}{n^2} \right) \sum_{i \in [n]} y_i^2 \\
&\leq -\frac{\epsilon k^2}{2n} + \frac{2k}{n} \leq -\frac{\epsilon k^2}{4n}
\end{aligned} \tag{9.2}$$

Where in the last inequality, we assume $k \geq 8/\epsilon$. □

Next, we bound the variance of Z. We defer the proof of the following Lemma to Section 9.3.2.

Lemma 9.3.7 (Variance Bound)**.** *Let $\delta_1, \ldots, \delta_n \sim$ Bernoulli$(\frac{k}{n})$. Let $y \in \mathbb{R}^n$ be any vector such that $\|y\|_2 \leq 1, \|y\|_\infty \leq \frac{1}{\epsilon\sqrt{n}}$, and $y^\top \mathbf{A} y = -\epsilon n$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ satisfies $\|\mathbf{A}\|_\infty \leq 1$. Further suppose that $\mathcal{R}_i(y) + \mathcal{C}_i(y) \leq 0$ for each $i \in [n]$. Then, assuming $k \geq 6/\epsilon$, we have*

$$\mathbf{Var}\left[\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j \right] \leq O\left(\frac{k^3}{n^2} \right)$$

Moreover, if the tighter bound $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$ holds for some $\alpha \leq 1$, we have

$$\mathbf{Var}\left[\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j \right] \leq O\left(\frac{k^2}{n^2} + \frac{\alpha k^3}{n^2} \right)$$

We note that the variance of the random quadratic form can be improved if we have tighter bounds on certain properties of the target vector y. We demonstrate this fact in the following Corollary, which we will use in Section 9.3.3. Note that the assumptions of Corollary 9.3.8 differ in several minor ways from those of Lemma 9.3.7. For instance, we do not require $k \geq 6/\epsilon$ (we note that this assumption was required only to simply the expression in Lemma 9.3.7). Also

notice that we do not bound the diagonal terms in Corollary 9.3.8. We defer the proof of Corollary 9.3.8 to Section 9.3.2.

Corollary 9.3.8 (Tighter Variance Bound)**.** *Let $\delta_1, \ldots, \delta_n \sim$ Bernoulli$(\frac{k}{n})$. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $\|\mathbf{A}\|_\infty \leq 1$ be any matrix and y a vector such that $|y^\top \mathbf{A} y| \leq c_1 \epsilon n$ for some value $c_1 > 0$, and such that $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$ for some $\alpha > 0$. Let $\mathbf{Z} \in \mathbb{R}^n$ be defined by $\mathbf{Z}_i = \mathcal{R}_i(y) + \mathcal{C}_i(y)$ for $i \in [n]$, and suppose we have $\|\mathbf{Z}\|_2^2 \leq c_2 \epsilon n$. Then we have*

$$\mathbf{Var}\left[\sum_{i \neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] \leq O\left(\frac{k^2}{n^2} + \frac{c_1^2 k^4 \epsilon^2}{n^2} + \frac{(c_1 + c_2)\epsilon k^3}{n^2} + \frac{\alpha^2 k^3}{n^2}\right)$$

We now observe that the variance computations from Lemma 9.3.7 immediately gives rise to a $O(1/\epsilon^3)$ algorithm.

Theorem 114. *There is a non-adaptive sampling algorithm which queries $O(\epsilon^{-3})$ entries of $\mathbf{A}$, and distinguishes the case that $\mathbf{A}$ is PSD from the case that $\lambda_{\min}(\mathbf{A}) < -\epsilon n$ with probability $2/3$.*

Proof. Let $x \in \mathbb{R}^n$ be the eigenvector associated with $\lambda_{\min}(\mathbf{A}) = -\epsilon n$ (recall that we can assume equality), and let $Z_1, \ldots, Z_d$ be independent repetitions of the above process, with $k = 10/\epsilon$ and $d = 3840/\epsilon$. Let $Z = \frac{1}{d}\sum_{i=1}^d Z_i$. Then $\mathbf{Var}(Z) \leq \frac{6}{d}\frac{k^3}{n^2}$ by Lemma 9.3.7, where we used the bounds on $\|x\|_\infty$ from Proposition 9.3.3 and the property that $\mathcal{R}_i(x) + \mathcal{C}_i(x) \leq 0$ for all i from Fact 9.3.5 to satisfies the assumptions of Lemma 9.3.7. By Chebysev's inequality:

$$\begin{aligned}\mathbf{Pr}\left[Z \geq -\frac{\epsilon k^2}{4n} + \frac{\epsilon k^2}{8n}\right] &\leq \left(\frac{64n^2}{\epsilon^2 k^4}\right)\left(\frac{6k^3}{dn^2}\right) \\ &\leq \frac{1}{10\epsilon k} \\ &\leq \frac{1}{100}\end{aligned} \tag{9.3}$$

It follows that with probability $99/100$, the average of the Z_i's will be negative. Thus at least one of the Z_i's must be negative, thus the submatrix corresponding to this Z_i will not be PSD. The total query complexity is $O(k^2 d) = O(\epsilon^{-3})$.

□

9.3.2 Variance Bounds

In this section, we provide the proofs of the variance bounds in Lemma 9.3.7 and Corollary 9.3.8. For convenience, we restate the Lemma and Corollary here before the proofs.

Lemma 9.3.7 *Let $\delta_1, \dots, \delta_n \sim$ Bernoulli$(\frac{k}{n})$. Let $y \in \mathbb{R}^n$ be any vector such that $\|y\|_2 \leq 1, \|y\|_\infty \leq \frac{1}{\epsilon\sqrt{n}}$, and $y^\top \mathbf{A} y = -\epsilon n$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ satisfies $\|\mathbf{A}\|_\infty \leq 1$. Further suppose that $\mathcal{R}_i(y) + \mathcal{C}_i(y) \leq 0$ for each $i \in [n]$. Then, assuming $k \geq 6/\epsilon$, we have*

$$\mathbf{Var}\left[\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] \leq O\left(\frac{k^3}{n^2}\right)$$

Moreover, if the tighter bound $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$ holds for some $\alpha \leq 1$, we have

$$\mathbf{Var}\left[\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] \leq O\left(\frac{k^2}{n^2} + \frac{\alpha k^3}{n^2}\right)$$

Proof. Let $c_{i,j} = \mathbf{A}_{i,j} y_i y_j$. We have

$$\begin{aligned}
\mathbf{Var}\left[\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] &\leq \frac{k}{n}\sum_i c_{i,i}^2 + \frac{k^2}{n^2}\sum_{i\neq j} c_{i,j}^2 + \frac{k^2}{n^2}\sum_{i\neq j} c_{i,j}c_{j,i} + \frac{k^2}{n^2}\sum_{i\neq j} c_{i,i}c_{j,j} + \frac{k^2}{n^2}\sum_{i\neq j} c_{i,i}c_{i,j} \\
&+ \frac{k^2}{n^2}\sum_{i\neq j} c_{i,i}c_{j,i} + \frac{k^3}{n^3}\sum_{i\neq j\neq u} c_{i,j}c_{u,j} + \frac{k^3}{n^3}\sum_{j\neq i\neq u} c_{i,j}c_{i,u} + \frac{k^3}{n^3}\sum_{i\neq j\neq u} c_{i,j}c_{j,u} + \frac{k^3}{n^3}\sum_{j\neq i\neq u} c_{i,j}c_{u,i} \\
&+ \frac{2k^3}{n^3}\sum_{i\neq j\neq u} c_{i,i}c_{j,u} + \frac{k^4}{n^4}\sum_{i\neq j\neq v\neq u} c_{i,j}c_{u,v} - \left(\frac{k^2}{n^2}\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j - \frac{k}{n}\sum_i \mathbf{A}_{i,i} y_i^2\right)^2
\end{aligned} \tag{9.4}$$

We first consider the last term $\frac{k^4}{n^4}\sum_{i\neq j\neq v\neq u} c_{i,j}c_{u,v} = \sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq v\neq i\neq j} y_u \mathbf{A}_{u,v} y_v$. Here $i \neq j \neq v \neq u$ means all 4 indices are distinct. Note that this term is canceled by a subset of the terms within $\left(\frac{k^2}{n^2}\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j - \frac{k}{n}\sum_i \mathbf{A}_{i,i} y_i^2\right)^2$. Similarly, the term $\frac{k^2}{n^2}\sum_{i\neq j} c_{i,i}c_{j,j}$ cancels. Moreover, after expanding $\left(\frac{k^2}{n^2}\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j - \frac{k}{n}\sum_i \mathbf{A}_{i,i} y_i^2\right)^2$, every remaining term which does not cancel with another term exactly is equal to another term in the variance above, but with an additional one (or two) factors of $\frac{k}{n}$ attached. Thus, if we can bound the remaining terms in Equation 9.4 by some value B, then an overall variance bound of $2 \cdot B$ will follow.

We now consider $\mathcal{T} = \Big(\sum_{j\neq i\neq u} c_{i,j}c_{i,u} + \sum_{i\neq j\neq u} c_{i,j}c_{u,j} + \sum_{j\neq i\neq u} c_{i,j}c_{u,i} + \sum_{i\neq j\neq u} c_{i,j}c_{j,u}\Big)$. We have

$$\sum_{i\neq j\neq u} c_{i,j}c_{i,u} = \sum_i \sum_{j\neq i} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_i \mathbf{A}_{i,u} y_u$$

$$\sum_{i\neq j\neq u} c_{i,j}c_{u,j} = \sum_j \sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_u \mathbf{A}_{u,j} y_j$$

$$\sum_{j\neq i\neq u} c_{i,j}c_{u,i} = \sum_i \sum_{j\neq i} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_u \mathbf{A}_{u,i} y_i$$

$$\sum_{i\neq j\neq u} c_{i,j}c_{j,u} = \sum_j \sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_j \mathbf{A}_{j,u} y_u$$

Now for simplicity, we write $\mathcal{R}_i = \mathcal{R}_i(y)$ and $\mathcal{C}_i = \mathcal{C}_i(y)$ for $i \in [n]$. Then by assumption, we have $\mathcal{R}_i + \mathcal{C}_i \leq 0$ for each i, thus $|\sum_i (\mathcal{R}_i + \mathcal{C}_i)| = \sum_i |(\mathcal{R}_i + \mathcal{C}_i)|$. Also note that we have $|\sum_i (\mathcal{R}_i + \mathcal{C}_i)| = |2y^\top \mathbf{A} y - 2\sum_i \mathbf{A}_{i,i} y_i^2| \leq 4\epsilon n$. Now observe

$$\left| \left(\sum_i \sum_{j\neq i} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_i \mathbf{A}_{i,u} y_u \right) - \sum_i \mathcal{R}_i^2 \right| = \sum_i \sum_{u\in[n]\setminus i} y_i^2 \mathbf{A}_{i,u}^2 y_u^2 \leq \sum_i y_i^2 \leq 1$$

And similarly

$$\left| \left(\sum_j \sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_u \mathbf{A}_{u,j} y_j \right) - \sum_j \mathcal{C}_i^2 \right| = \sum_j \sum_{u\in[n]\setminus j} y_u^2 \mathbf{A}_{u,j}^2 y_j^2 \leq \sum_j y_j^2 \leq 1$$

$$\left| \left(\sum_i \sum_{j\neq i} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_u \mathbf{A}_{u,i} y_i \right) - \sum_i \mathcal{R}_i \mathcal{C}_i \right| = \sum_i \sum_{u\in[n]\setminus i} y_i^2 \mathbf{A}_{i,u} \mathbf{A}_{u,i} y_u^2 \leq \sum_i y_i^2 \leq 1$$

$$\left| \left(\sum_j \sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \sum_{u\neq i\neq j} y_j \mathbf{A}_{j,u} y_u \right) - \sum_j \mathcal{R}_j \mathcal{C}_j \right| = \sum_j \sum_{u\in[n]\setminus j} y_u^2 \mathbf{A}_{u,j} \mathbf{A}_{j,u} y_j^2 \leq \sum_j y_j^2 \leq 1$$

Taking these four equations together, we obtain $|\mathcal{T} - \sum_i (\mathcal{R}_i + \mathcal{C}_i)^2| \leq 4$, so it will suffice to upper bound the value $\sum_i (\mathcal{R}_i + \mathcal{C}_i)^2$ instead. First note that since $|y_i| \leq \frac{1}{\epsilon\sqrt{n}}$ for all i, so for any $i \in [n]$ we have

$$|(\mathcal{R}_i + \mathcal{C}_i)| \leq |\sum_{j\neq i} y_i \mathbf{A}_{i,j} y_j| + |\sum_{j\neq i} y_j \mathbf{A}_{j,i} y_i| \leq \frac{1}{\epsilon\sqrt{n}} (\sum_j 2y_j) \leq \frac{2}{\epsilon\sqrt{n}} \|y\|_1 \leq \frac{2}{\epsilon}$$

Combining this bound with the fact that $\sum_i |(\mathcal{R}_i + \mathcal{C}_i)| \leq 4\epsilon n$ from earlier, it follows that the sum $\sum_i (\mathcal{R}_i + \mathcal{C}_i)^2$ is maximized by setting $2\epsilon^2 n$ of the terms $(\mathcal{R}_i + \mathcal{C}_i)$ equal to the largest possible value of $(2/\epsilon)$, so that $\sum_i (\mathcal{R}_i + \mathcal{C}_i)^2 \leq 2\epsilon^2 n (2/\epsilon)^2 = O(n)$. This yields an upper bound of

$\frac{k^3}{n^3}\mathcal{T} = O(\frac{k^3}{n^2})$. Note, that in general, given the bound $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$ for some value $\alpha \leq 1$, then each term $|(\mathcal{R}_i + \mathcal{C}_i)| \leq \frac{2\alpha}{\epsilon}$. On the other hand, $\sum_i |(\mathcal{R}_i + \mathcal{C}_i)| \leq 4\epsilon n$. Thus, once again, $\sum_i |(\mathcal{R}_i + \mathcal{C}_i)|^2$ is maximized by setting $\Theta(\epsilon^2 n/\alpha)$ inner terms equal to $\Theta((\frac{\alpha}{\epsilon})^2)$, giving $\mathcal{T} \leq \alpha n$ for general $\alpha < 1$. Thus, for general $\alpha \leq 1$, we have $\frac{k^3}{n^3}\mathcal{T} = O(\frac{\alpha k^3}{n^2})$.

Next, we bound $\frac{k^2}{n^2}\sum_{i\neq j} c_{i,i}c_{i,j} + \frac{k^2}{n^2}\sum_{i\neq j} c_{i,i}c_{j,i}$ by $\frac{k^2}{n^2}\sum_i y_i^2(\mathcal{R}_i + \mathcal{C}_i)$. As shown above, $|\mathcal{R}_i + \mathcal{C}_i| \leq 2y_i\sqrt{n}$, thus altogether we have

$$\frac{k^2}{n^2}\left(\sum_{i\neq j} c_{i,i}c_{i,j} + \sum_{i\neq j} c_{i,i}c_{j,i}\right) \leq \frac{k^2}{n^2}\sum_i y_i^3\sqrt{n} \tag{9.5}$$

Using that $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$ for $\alpha \leq 1$, and the fact that $\|y\|_2^2 \leq 1$, it follows that $\|y\|_3^3$ is maximized by having $\frac{n\epsilon^2}{\alpha^2}$ terms equal to $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$, which gives an upper bound of $\|y\|_3^3 \leq \frac{\alpha}{\epsilon\sqrt{n}}$. Thus, we can bound the right hand side of Equation 9.5 by $\frac{k^2\alpha}{n^2\epsilon}$, which is $O(k^3/n^2)$ when $\alpha = 1$ using that $k = \Omega(1/\epsilon)$.

Now, we bound $\frac{k^3}{n^3}\sum_{i\neq j\neq u} c_{i,i}c_{j,u}$ by

$$\begin{aligned}
\frac{k^3}{n^3}\sum_{i\neq j\neq u} c_{i,i}c_{j,u} &\leq \frac{k^3}{n^3}\sum_i y_i^2\mathbf{A}_{i,i}\sum_{j\neq u\neq i} y_j\mathbf{A}_{j,u}y_u \\
&\leq \frac{k^3}{n^3}\sum_i y_i^2\mathbf{A}_{i,i}\sum_{j\neq u\neq i} y_j\mathbf{A}_{j,u}y_u \\
&\leq \frac{k^3}{n^3}\sum_i y_i^2\mathbf{A}_{i,i}\left(\epsilon n + O(1)\right) \\
&\leq \frac{\epsilon k^3}{n^2} \\
&= O(\frac{k^2}{n^2})
\end{aligned} \tag{9.6}$$

Also observe that $\sum_{i,j} c_{i,j}^2 \leq \sum_{i,j} y_i^2 y_j^2 = \|y\|_2^4 \leq 1$, so $\sum_{i\neq j} c_{i,j}^2 \leq \sum_{i,j} c_{i,j}^2 \leq 1$, and also $\sum_{i\neq j} c_{i,j}c_{j,i} \leq \sum_{i,j} y_i^2 y_j^2 \leq 1$, which bounds their corresponding terms in the variance by $O(k^2/n^2)$. Finally, we must bound the last term $\frac{k}{n}\sum_i c_{i,i}^2 = \frac{k}{n}\sum_i y_i^4\mathbf{A}_{i,i}^2 \leq \frac{k}{n}\sum_i y_i^4$. Note that $|y_i| \leq 1/(\epsilon\sqrt{n})$ for each i, and $\|y\|_2 \leq 1$. Thus $\sum_i y_i^4$ is maximized when one has $\epsilon^2 n$ terms equal to $1/(\epsilon\sqrt{n})$, and the rest set to 0. So $\sum_i y_i^4 \leq \epsilon^2 n(\frac{1}{\epsilon\sqrt{n}})^4 \leq \frac{1}{\epsilon^2 n}$. In general, if $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$, we have $\sum_i y_i^4 \leq \frac{\epsilon^2 n}{\alpha^2}(\frac{\alpha}{\epsilon\sqrt{n}})^4 \leq \frac{\alpha^2}{\epsilon^2 n}$. Thus we can bound $\frac{k}{n}\sum_i c_{i,i}^2$ by $O(\frac{k^3\alpha^2}{n^2})$

Altogether, this gives

$$\begin{aligned}\mathbf{Var}\left[\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] &\leq O(\frac{k^2}{n^2} + \frac{\alpha k^3}{n^2} + \frac{\alpha k^2}{n^2 \epsilon} + \frac{\alpha^2 k^3}{n^2}) \\ &= O(\frac{k^2}{n^2} + \frac{\alpha k^3}{n^2} + \frac{\alpha^2 k^3}{n^2})\end{aligned} \tag{9.7}$$

which is $O(k^3/n^2)$ in general (where $\alpha \leq 1$), where we assume $k \geq 6/\epsilon$ throughout. □

Corollary 9.3.8 *Let $\delta_i \in \{0,1\}$ be an indicator random variable with $\mathbf{E}[\delta_i] = k/n$. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $\|\mathbf{A}\|_\infty \leq 1$ be any matrix and y a vector such that $|y^\top \mathbf{A} y| \leq c_1 \epsilon n$ for some value $c_1 > 0$, and such that $\|y\|_\infty \leq \frac{\alpha}{\epsilon\sqrt{n}}$ for some $\alpha > 0$. Let $\mathbf{Z} \in \mathbb{R}^n$ be defined by $\mathbf{Z}_i = \mathcal{R}_i(y) + \mathcal{C}_i(y)$ for $i \in [n]$, and suppose we have $\|\mathbf{Z}\|_2^2 \leq c_2 \epsilon n$. Then we have*

$$\mathbf{Var}\left[\sum_{i \neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] \leq O\left(\frac{k^2}{n^2} + \frac{c_1^2 k^4 \epsilon^2}{n^2} + \frac{(c_1 + c_2)\epsilon k^3}{n^2} + \frac{\alpha^2 k^3}{n^2}\right)$$

Proof. We proceed as in Lemma 9.3.7, except that we may remove the terms with $c_{i,j}$ for $i = j$, yielding

$$\begin{aligned}\mathbf{Var}\left[\sum_{i \neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] &\leq \frac{k^2}{n^2}\sum_{i \neq j} c_{i,j}^2 + \frac{k^2}{n^2}\sum_{i \neq j} c_{i,j} c_{j,i} + \frac{k^3}{n^3}\sum_{i \neq j \neq u} c_{i,j} c_{u,j} \\ &+ \frac{k^3}{n^3}\sum_{j \neq i \neq u} c_{i,j} c_{i,u} + \frac{k^3}{n^3}\sum_{i \neq j \neq u} c_{i,j} c_{j,u} + \frac{k^3}{n^3}\sum_{j \neq i \neq u} c_{i,j} c_{u,i} \\ &+ \frac{k^3}{n^3}\sum_{i \neq j \neq u} c_{i,i} c_{j,u} + \frac{k^4}{n^4}\sum_{i \neq j \neq v \neq u} c_{i,j} c_{u,v} - \left(\frac{k^2}{n^2}\sum_{i \neq j} y_i \mathbf{A}_{i,j} y_j\right)^2\end{aligned} \tag{9.8}$$

As in Lemma 9.3.7, we can cancel the term $\frac{k^4}{n^4}\sum_{i \neq j \neq v \neq u} c_{i,j} c_{u,v}$ with a subterm of $-\left(\frac{k^2}{n^2}\sum_{i \neq j} y_i \mathbf{A}_{i,j} y_j\right)^2$, and bound the remaining contribution of $-\left(\frac{k^2}{n^2}\sum_{i \neq j} y_i \mathbf{A}_{i,j} y_j\right)^2$ by individually bounding the other terms in the sum.

First, we can similarly bound the last term by $c_1^2 \epsilon^2 k^4 / n^2$ as needed. Now when bounding

$$\mathcal{T} = \left(\sum_{j \neq i \neq u} c_{i,j} c_{i,u} + \sum_{i \neq j \neq u} c_{i,j} c_{u,j} + \sum_{j \neq i \neq u} c_{i,j} c_{u,i} + \sum_{i \neq j \neq u} c_{i,j} c_{j,u}\right)$$

we first observe that in the proof of Lemma 9.3.7, we only needed a bound on $\|\mathbf{Z}\|_2^2$ to give the bound on $\mathcal{T}$. So by assumption, $\|\mathbf{Z}\|_2^2 \leq c_2 \epsilon n$, which gives a total bound of $\frac{c_2 k^3 \epsilon}{n^2}$ on $\frac{k^3}{n^3}\mathcal{T}$.

Also, we bound we bound $\frac{k^3}{n^3}\sum_{i\neq j\neq u} c_{i,i}c_{j,u}$ by

$$\begin{aligned}
\frac{k^3}{n^3}\sum_{i\neq j\neq u} c_{i,i}c_{j,u} &\leq \frac{k^3}{n^3}\sum_i y_i^2 \mathbf{A}_{i,i} \sum_{j\neq u\neq i} y_j \mathbf{A}_{j,u} y_u \\
&\leq \frac{k^3}{n^3}\sum_i y_i^2 \mathbf{A}_{i,i} \sum_{j\neq u\neq i} y_j \mathbf{A}_{j,u} y_u \\
&\leq \frac{k^3}{n^3}\sum_i y_i^2 \mathbf{A}_{i,i} \left(c_1 \epsilon n + O(1)\right) \\
&\leq \frac{\epsilon c_1 k^3}{n^2}
\end{aligned} \tag{9.9}$$

which is within our desired upper bound. Finally observe that $\sum_{i,j} c_{i,j}^2 \leq \sum_{i,j} y_i^2 y_j^2 = \|y|_2^4 \leq 1$, so $\sum_{i\neq j} c_{i,j}^2 \leq \sum_{i,j} c_{i,j}^2 \leq 1$, and also $\sum_{i\neq j} c_{i,j}c_{j,i} \leq \sum_{i,j} y_i^2 y_j^2 \leq 1$, which bounds their corresponding terms in the variance by $O(k^2/n^2)$, which completes the proof.

□

9.3.3 Improving the complexity to $\tilde{O}(1/\epsilon^2)$

We now demonstrate how to obtain an improved sample complexity of $\tilde{O}(1/\epsilon^2)$ using different scales of sub-sampling, as well as a careful "eigenvector switching" argument. As before, we can assume that $\mathbf{A}$ is symmetric, and $x = \arg\min_{v\in\mathbb{R}^n, \|v\|_2\leq 1} v^\top \mathbf{A} v$ is the smallest eigenvector of $\mathbf{A}$, so that that $\langle x, \mathbf{A}x\rangle = \lambda_{\min}(\mathbf{A}) = -\epsilon n$. Also recall that our algorithms will not need to explicitly know the value $\epsilon = \min_{v\in\mathbb{R}^n, \|v\|_2\leq 1} v^\top \mathbf{A} v/n$, only a lower bound on it, since when run on smaller ϵ our algorithm only samplers larger submatrices. By Proposition 9.3.3, we have $\|x\|_\infty \leq \frac{1}{\epsilon\sqrt{n}}$. We now partition the coordinates of x into *level-sets*, such that all the coordinates x_i^2 within a level set have magnitudes that are close to each other.

Definition 9.3.9. *Given $(\mathbf{A}, x)$, where x is as defined above, define the base level set S as $S = \{i \in [n] \ : \ |x_i|^2 \leq \frac{100}{\epsilon n}\}$, and let $T_a = \{i \in [n] \ : \ \frac{100\cdot 2^{a-1}}{\epsilon n} \leq |x_i|^2 \leq \frac{100\cdot 2^a}{\epsilon n}\}$ for an integer $a \geq 1$.*

We now break the analysis into two possible cases. In the first case, the coordinates in one of the sets T_a contributed a substantial fraction of the "negativeness" of the quadratic form $x_T^\top \mathbf{A} x$,

for some a sufficiently large. Since the sets T_a can become smaller as a increases while still contributing a large fraction of the negative mass, this case can be understood as the negativeness of $x^\top \mathbf{A} x$ being highly concentrated in a small fraction of the matrix, which we must then find to determine that $\mathbf{A}$ is not PSD. In the second case, no such contributing T_a exists, and the negative mass is spread out more evenly across the terms in the quadratic form $x^\top \mathbf{A} x$. If this is the case, we will show that our variance bounds from the prior section can be made to obtain a proof that a single, large sampled principal submatrix $T \subset [n]$ will satisfies $x_T^\top \mathbf{A}_{T\times T} x_T$ will be negative with non-negigible probability.

Formally, we define the two cases as follows:

Case 1: We have $x_S \mathbf{A} x_{T_a} + x_{T_a} \mathbf{A} x_S \leq -\frac{\epsilon n}{10 \log(1/\epsilon)}$ for some a such that $2^a \geq 10^6 \zeta^3$, for some $\zeta = \Theta(\log^2(1/\epsilon))$ with a large enough constant.

Case 2: The above does not hold; namely, we have $x_S \mathbf{A} x_{T_a} + x_{T_a} \mathbf{A} x_S > -\frac{\epsilon n}{10 \log(1/\epsilon)}$ for every $2^a \geq 10^6 \zeta^3$.

Case 1: Varied Subsampling and Eigenvector Switching

In this section, we analyze the **Case 1**, which specifies that $x_S \mathbf{A} x_{T_a} + x_{T_a} \mathbf{A} x_S \leq -\frac{\epsilon n}{10 \log(1/\epsilon)}$ for some T_a such that $2^a \geq 10^6 \zeta^3$, where $\zeta = \Theta(\log^2(1/\epsilon))$ is chosen with a sufficiently large constant. Recall here that $x \in \mathbb{R}^n$ is the (unit) eigenvector associated with $\lambda_{\min}(\mathbf{A}) = -\epsilon n$. We now fix this value a associated with T_a. In order to find a principal submatrix $\mathbf{A}_{T\times T}$ that is not PSD for some sampled subset $T \subset [n]$, we will need to show that $T \cap T_a$ intersects in at least one coordinate.

As discussed in Section 9.2, we will need to switch our analysis from x to a different vector y, in order to have $y_T^\top \mathbf{A}_{T\times T} y_T < 0$ with non-negligible probability conditioned on $|T \cap T_a| \geq 1$. To construct the appropriate vector y, we will first proceed by proving several propositions which bound how the quadratic form $x^\top \mathbf{A} x$ changes as we modify or remove some of the coordinates of x. For the following propositions, notice that by definition of T_b, using the fact that $\|x\|_2^2 \leq 1$, we have that $|T_b| \leq \frac{\epsilon n}{1002^{b-1}}$ for any $b \geq 1$, which in particular holds for $b = a$.

Proposition 9.3.10. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ satisfy $\|\mathbf{A}\|_\infty \leq 1$. Let $S, T \subset [n]$, and let $v \in \mathbb{R}^n$ be any vector such that $\|v\|_2 \leq 1$. Then $|v_S^\top \mathbf{A} v_T| \leq \sqrt{|S| \cdot |T|}$*

Proof. We have $|v_S^\top \mathbf{A} v_T| = |\sum_{i\in S} \sum_{j\in T} v_i \mathbf{A}_{i,j} v_j| \leq \sum_{i\in S} |v_i| \sum_{j\in T} |v_j| \leq \sum_{i\in S} |v_i| \|v_T\|_1 \leq$

$\|v_S\|_1\|v_T\|_1 \le \sqrt{|S||T|}$ as needed. □

Proposition 9.3.11. *Let $\mathbf{A} \in \mathbb{R}^{n\times m}$ satisfy $\|\mathbf{A}\|_\infty \le 1$ for any n, m. and let $v \in \mathbb{R}^n, u \in \mathbb{R}^m$ satisfy $\|u\|_2^2, \|v\|_2^2 \le 1$. Then*

$$\sum_{j=1}^{m}\left(\sum_{i=1}^{n} v_i \mathbf{A}_{i,j} u_j\right)^2 \le n$$

Proof. We have $\left(\sum_{i=1}^n v_i\mathbf{A}_{i,j}u_j\right)^2 \le u_j^2\left(\sum_{i=1}^n |v_i|\right)^2 \le u_j^2\|v\|_1^2$, so the sum can be bounded by $\sum_{j=1}^m u_j^2\|v\|_1^2 = \|v\|_1^2\|u\|_2^2 \le \|v\|_1^2 \le n$ as needed.

□

Proposition 9.3.12. *Let x be as defined above. Then we have $|\langle x_S, \mathbf{A}x_S\rangle| \le 10\epsilon n$.*

Proof. Suppose $\langle x_S, \mathbf{A}x_S\rangle = C\epsilon n$ for a value C with $|C| > 10$. Note that $|\langle x_{[n]\setminus S}, \mathbf{A}x_{[n]\setminus S}\rangle| \le \frac{\epsilon n}{100}$ by Proposition 9.3.10, using that $|[n]\setminus S| = |\cup_{b\ge 1} T_b| \le \frac{\epsilon n}{100}$ (here we use the fact that at most $\frac{\epsilon n}{100}$ coordinates of a unit vector can have squared value larger than $\frac{100}{\epsilon n}$). If $C > 0$, then we must have that $(\langle x_S, \mathbf{A}x_{[n]\setminus S}\rangle + \langle x_{[n]\setminus S}, \mathbf{A}x_S\rangle) \le -(C + 99/100)\epsilon n$ for us to have that $\langle x, \mathbf{A}x\rangle = -\epsilon n$ exactly. Thus if C is positive and larger than 10, it would follow that by setting $v = x_S/2 + x_{[n]\setminus S}$, we would obtain a vector v with $\|v\|_2 \le 1$ such that v has smaller quadratic form with $\mathbf{A}$ than x, namely with $v^\top\mathbf{A}v \le -(C + 99/100)\epsilon n/2 + \epsilon Cn/4 + n\epsilon/100 < -\epsilon n$ using that $C > 10$, which contradicts the optimality of x as the eigenvector for $\lambda_{\min}(\mathbf{A})$. Furthermore, if $C < -10$, then $x_S^\top\mathbf{A}x_S < -10\epsilon$, which again contradicts the optimality of x.

□

Now recall that the total row and column contributions of i are defined as $\mathcal{R}_i(x) = \sum_{j\in[n]\setminus i} x_i\mathbf{A}_{i,j}x_j$ and $\mathcal{C}_i(x) = \sum_{j\in[n]\setminus i} x_j\mathbf{A}_{j,i}x_i$ respectively. In the remainder of the section, we simply write $\mathcal{R}_i = \mathcal{R}_i(x)$ and $\mathcal{C}_i = \mathcal{C}_i(x)$. We now define the contribution of i within the set $S \subset [n]$.

Definition 9.3.13. *Let $S \subset [n]$ be as defined above. Then for any $i \in [n]$, define the row and column contributions of i within S as $\mathcal{R}_i^S = \sum_{j\in S\setminus i} x_i\mathbf{A}_{i,j}x_j$ and $\mathcal{C}_i^S = \sum_{j\in S\setminus i} x_j\mathbf{A}_{j,i}x_i$ respectively.*

Observe from the above definition, we have $\sum_{i\in T_a}(\mathcal{R}_i^S + \mathcal{C}_i^S) = (x_S\mathbf{A}x_{T_a} + x_{T_a}\mathbf{A}x_S) \le -\frac{\epsilon n}{10\log(1/\epsilon)}$, where the inequality holds by definition of Case 1.

Proposition 9.3.14. *We have $\sum_{i\in S}(\mathcal{R}_i^S + \mathcal{C}_i^S)^2 \le 1601\epsilon n$.*

Proof. Let $z^S, z, z^- \in \mathbb{R}^{|S|}$ be vectors defined for $i \in S$ as $z_i^S = \mathcal{R}_i^S + \mathcal{C}_i^S$, $z_i = \mathcal{R}_i + \mathcal{C}_i$, and $z^- = z - z^S$. Notice that our goal is to bound $\|z^S\|_2^2$, which by triangle inequality satisfies $\|z^S\|_2^2 \leq 2\left(\|z\|_2^2 + \|z^-\|_2^2\right)$. First note that

$$\begin{aligned}\|z^-\|_2^2 &= \sum_{i\in S}\left(\sum_{j\notin S} x_i \mathbf{A}_{i,j} x_j + \sum_{j\notin S} x_j \mathbf{A}_{j,i} x_i\right)^2 \\ &\leq 2\sum_{i\in S}\left(\sum_{j\notin S} x_i \mathbf{A}_{i,j} x_j\right)^2 + 2\sum_{i\in S}\left(\sum_{j\notin S} x_j \mathbf{A}_{j,i} x_i\right)^2\end{aligned} \tag{9.10}$$

Using that $[n] \setminus S < \epsilon n/100$, we have by Proposition 9.3.11 that $\sum_{i\in S}\left(\sum_{j\notin S} x_i \mathbf{A}_{i,j} x_j\right)^2 \leq \epsilon n/100$, so $\|z^-\|_2^2 \leq \epsilon n/25$.

We now bound $\|z\|_1 = \sum_{i\in S}|\mathcal{R}_i + \mathcal{C}_i|$. By Fact 9.3.5, we have $\mathcal{R}_i + \mathcal{C}_i \leq 0$ for all $i \in [n]$, which means that $\|z\|_1 \leq \sum_{i\in[n]}|\mathcal{R}_i + \mathcal{C}_i| = |2\langle x, \mathbf{A}x\rangle - 2\sum_{i\in[n]}\mathbf{A}_{i,i}(x_i)^2| \leq 2\epsilon n$. Next, we bound $\|z\|_\infty$. Notice that $|\mathcal{R}_i + \mathcal{C}_i| = 2|x_i\langle \mathbf{A}_{i,*}, x\rangle - \mathbf{A}_{i,i}(x_i)^2| \leq 2\epsilon n (x_i)^2 + 2\mathbf{A}_{i,i}(x_i)^2 < 4\epsilon n (x_i)^2$, using that $\mathbf{A}x = -\epsilon n x$, since x is an eigenvector of $\mathbf{A}$. Since for $i \in S$, it also follows that $(x_i)^2 \leq \frac{100}{\epsilon n}$, thus $\|z\|_\infty \leq 400$ as needed. It follows that $\|z\|_2^2$ is maximized by having $2\epsilon n/400$ coordinates equal to 400, giving $\|z\|_2^2 \leq 2\epsilon n/400(400)^2 = 800\epsilon n$. It follows then that $\|z^S\|_2^2 \leq 1601\epsilon n$ as needed. □

Eigenvector Setup: We now define the "target" direction $y \in \mathbb{R}^n$ which we will use in our analysis for **Case 1**. First, we will need the following definitions. Let

$$D_a^p = \left\{ t \in T_a \ : \ -\frac{2^{p+1}2^a}{\log(1/\epsilon)} \leq \mathcal{R}_t^S + \mathcal{C}_t^S \leq -\frac{2^p 2^a}{\log(1/\epsilon)} \right\}$$

Define the *fill* β of T_a as the value such that $\beta = 2^{-p}$ where $p \geq 1$ is the smallest value of p such that $-|D_a^p|(\frac{2^p 2^a}{\log(1/\epsilon)}) \leq -\frac{\epsilon n}{40\log^2(1/\epsilon)}$. Note that at least one such p for $1 \leq p \leq \log(1/\epsilon)$ must exist. let $T_a^* = D_a^p$ where $\beta = 2^{-p}$. Observe that $x_S \mathbf{A} x_{T_a^*} + x_{T_a^*}\mathbf{A}x_S \leq -\frac{\epsilon n}{40\log^2(1/\epsilon)}$. Finally, we define our target "eigenvector" y as

$$y = x_S + \zeta\beta\left(2^{-a}\cdot x_{T_a}\right) \tag{9.11}$$

where $\zeta = \Theta(\log^2(1/\epsilon))$ is as above, and we also define our target submatrix subsampling size as $\lambda = \frac{2000\beta^2\zeta^2\log(1/\epsilon)}{2^a\epsilon}$. First, we prove that for a random submatrix $\mathbf{A}_{T\times T}$, where $i \in [n]$ is sampled and added to T with probability λ/n, we have that $y^\top \mathbf{A}_{T\times T} y$ is negative in expectation conditioned on $|T \cap T_a^*| \geq 1$.

Lemma 9.3.15. *Suppose we are in case* 1 *with* T_a *contributing such that* $2^a \geq 10^6\zeta^3$. *Let* δ_i *be an indicator variable that we sample coordinate* i, *with* $\mathbf{E}[\delta_i] = \frac{\lambda}{n}$ *and* $\lambda = \frac{2000\beta^2\zeta^2\log(1/\epsilon)}{2^a\epsilon}$. *Then if* $y = x_S + \zeta\beta(2^{-a}x_{T_a})$ *where* $\zeta \geq 100\log^2(1/\epsilon)$, *and if* $t \in T_a^*$, *then*

$$\mathbb{E}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \;\middle|\; \delta_t = 1\right] \leq -\frac{50\zeta\lambda}{n}$$

Proof. First observe $\mathbf{E}[\sum_{i\in S} y_i^2\mathbf{A}_{i,i}\delta_i] \leq \frac{\lambda}{n}\|x\|_2^2 \leq \frac{\lambda}{n}$ since x is a unit vector. Note that $y_S = x_S$ by construction, so we can use Proposition 9.3.12 to bound $|\langle y_S, \mathbf{A}y_S\rangle|$ by $10\epsilon n$, which gives

$$\begin{aligned}
\mathbb{E}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \;\middle|\; \delta_t = 1\right] &\leq \frac{\lambda^2}{n^2}\left(\langle y_S, \mathbf{A}y_S\rangle - \sum_{i\in S} y_i^2\mathbf{A}_{i,i}\delta_i\right) + \frac{\lambda}{n} + \frac{\lambda}{n}\sum_{i\in S}(\mathbf{A}_{t,i} + \mathbf{A}_{i,t})y_iy_t + y_t^2 \\
&\leq \frac{20\lambda^2\epsilon}{n} + \frac{\lambda}{n}\left(1 + \frac{\zeta\beta}{2^a}\left(\mathcal{R}_t^S + \mathcal{C}_t^S\right)\right) + (\frac{\zeta\beta}{2^a})^2(\frac{1002^a}{\epsilon n}) \\
&\leq \frac{20\lambda^2\epsilon}{n} + \frac{\lambda}{n}\left(1 + \frac{\zeta\beta}{2^a}\left(\mathcal{R}_t^S + \mathcal{C}_t^S\right)\right) + \frac{100\zeta^2\beta^2}{2^a\epsilon n}
\end{aligned} \tag{9.12}$$

Now by definition of $i \in T_a^*$, we have $\left(\mathcal{R}_t^S + \mathcal{C}_t^S\right) \leq -\frac{2^a}{\beta\log(1/\epsilon)}$. Thus

$$\mathbb{E}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \;\middle|\; \delta_t = 1\right] \leq \frac{20\lambda^2\epsilon}{n} + \frac{\lambda}{n}\left(1 - \frac{\zeta}{\log(1/\epsilon)}\right) + \frac{100\zeta^2\beta^2}{2^a\epsilon n} \tag{9.13}$$

Setting $\zeta > 100\log^2(1/\epsilon)$, we first note that $\frac{\lambda}{n}\left(1 - \frac{\zeta}{\log(1/\epsilon)}\right) \leq -\frac{99\lambda\zeta}{100n\log(1/\epsilon)}$. Since

$$\lambda = \frac{2000\beta^2\zeta^2\log(1/\epsilon)}{2^a\epsilon} \leq \frac{2000\beta^2}{2^6\zeta\epsilon} \leq \frac{1}{\zeta\epsilon}$$

it follows that $\frac{20\lambda^2\epsilon}{n} \leq \frac{20\lambda}{n\zeta} \leq \frac{20\lambda}{n} < \frac{\lambda\zeta}{5\log(1/\epsilon)n}$. Thus $\frac{10\lambda^2\epsilon}{n} - \frac{99\lambda\zeta}{100n\log(1/\epsilon)} \leq -\frac{3\lambda\zeta}{4n\log(1/\epsilon)}$. So we can

simply and write

$$\begin{aligned}\mathbb{E}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \;\middle|\; \delta_t=1\right] &\leq -\frac{3\lambda\zeta}{4n\log(1/\epsilon)}+\frac{100\zeta^2\beta^2}{2^a\epsilon n}\\ &= -\frac{1500\beta^2\zeta^3}{2^a\epsilon n}+\frac{100\zeta^2\beta^2}{2^a\epsilon n}\\ &\leq -\frac{1400\beta^2\zeta^3}{2^a\epsilon n}\\ &\leq -\frac{50\zeta\lambda}{n}\end{aligned} \tag{9.14}$$

as desired. □

Lemma 9.3.16. *Let δ_i be an indicator variable with $\mathbf{E}[\delta_i]=\lambda/n$. Then*

$$\mathbf{Pr}\left[\left|\sum_{i,j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j - \mathbf{E}\left[\sum_{i,j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j\right]\right| \geq C\frac{\lambda}{n}\right] \leq \frac{49}{50}$$

Where $C>0$ is some constant.

Proof. We can apply Corollary 9.3.8, where we can set the values of c_1, c_2 to be bounded by constants by the results of Proposition 9.3.12 and 9.3.14, and by definition of the set S we can set $\alpha \leq \sqrt{\epsilon}$ for the α in Corollary 9.3.8, and using that $\lambda \leq O(1/\epsilon)$, we obtain:

$$\mathbf{Var}\left[\sum_{i\neq j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j\right] \leq \frac{C\lambda^2}{n^2}$$

for some constant C. Now note that $\mathbf{E}\left[\sum_{i\neq j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j\right] \leq \frac{30\lambda^2\epsilon}{n} \leq O(\frac{\lambda}{n})$, thus by Chebyshev's, with probability $99/100$, we have $|\sum_{i\neq j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j| \leq O(\frac{\lambda}{n})$. Moreover, note that $\sum_i \mathbf{A}_{i,i}y_i^2\delta_i$ can be assumed to be a positive random variable using that $\mathbf{A}_{i,i}=1$, and note the expectation of this variable is $\frac{\lambda}{n}$, and is at most $100\lambda/n$ with probability $99/100$. Thus $|\sum_{i\in S} y_i^2\mathbf{A}_{i,i}\delta_i - \mathbf{E}[\sum_{i\in S} y_i^2\mathbf{A}_{i,i}]| \leq 100\lambda/n$. By a union bound, we have:

$$\mathbf{Pr}\left[\left|\sum_{i,j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j\right| \geq C\frac{\lambda}{n}\right] \leq \frac{49}{50}$$

Where $C=150$. □

Lemma 9.3.17. *Fix any $t \in T_a^*$. Then*

$$\mathbf{Pr}\left[\left|\sum_{i\in S} y_t(y_i\mathbf{A}_{i,t} + \mathbf{A}_{t,i}y_i)\delta_i - \mathbf{E}\left[\sum_{i\in S} y_t(y_i\mathbf{A}_{i,t} + \mathbf{A}_{t,i}y_i)\delta_i\right]\right| \geq \frac{10\lambda}{n}\right] \leq \frac{1}{100}$$

Proof. By independence of the δ_i's

$$\begin{aligned}
\mathbf{Var}\left(\sum_{i\in S}(y_i\mathbf{A}_{i,t}y_t + y_t\mathbf{A}_{t,i}y_i)\delta_i\right) &\leq \frac{\lambda}{n}\sum_i (y_i\mathbf{A}_{i,t}y_t + y_t\mathbf{A}_{t,i}y_i)^2 \\
&\leq \frac{\lambda}{n}\sum_i 2y_t^2y_i^2 \\
&\leq \frac{2\lambda}{n}(\zeta\beta 2^{-a})^2\frac{1002^a}{\epsilon n} \\
&\leq \frac{2\lambda}{n}\left(\frac{100\zeta^2\beta^2}{2^a\epsilon n}\right) \\
&\leq \frac{2\lambda^2}{5n^2}
\end{aligned} \tag{9.15}$$

Now since $\mathbf{E}\left[\sum_{i\in S}(y_i\mathbf{A}_{i,t}y_t + y_t\mathbf{A}_{t,i}y_i)\delta_i\right] \leq -\frac{\lambda}{n}\frac{\zeta}{\log(1/\epsilon)} \leq -\frac{100\lambda}{n}$, the desired result follows by Chebyshev's inequality. □

Lemma 9.3.18. *Fix any $t \in T_a^*$. Then*

$$\mathbf{Pr}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \leq \frac{-25\zeta\lambda}{n} \;\middle|\; \delta_t = 1\right] \geq 24/25$$

Proof. Conditioned on $\delta_t = 1$, we have

$$\begin{aligned}
&\left|\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j\right] - \mathbf{E}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \;\middle|\; \delta_t = 1\right]\right| \\
&\leq \left|\sum_{i\in S} y_t(y_i\mathbf{A}_{i,t} + \mathbf{A}_{t,i}y_i)\delta_i - \mathbf{E}\left[\sum_{i\in S} y_t(y_i\mathbf{A}_{i,t} + \mathbf{A}_{t,i}y_i)\delta_i\right]\right| \\
&+ \left|\sum_{i,j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j - \mathbf{E}\left[\sum_{i,j\in S} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j\right]\right| \\
&\leq C\frac{\lambda}{n}
\end{aligned} \tag{9.16}$$

for some constant $C \leq 200$, where the last fact follows from Lemmas 9.3.16 and 9.3.17 with probability $24/25$. Since $\mathbf{E}\left[\sum_{i,j\in S\cup\{t\}} y_i\mathbf{A}_{i,j}y_j\delta_i\delta_j \;\middle|\; \delta_t = 1\right] \leq -\frac{50\zeta\lambda}{n}$ by Lemma 9.3.15, by

scaling ζ by a sufficiently large constant, the result follows. $\square$

Theorem 115. *Suppose we are in case* 1 *with* T_a *contributing such that* $2^a \geq 10^6\zeta^3$. *Then there is an algorithm that queries at most* $O(\frac{\log^7(1/\epsilon)}{\epsilon^2})$ *entries of* $\mathbf{A}$, *and finds a principal submatrix of* $\mathbf{A}$ *which is not PSD with probability at least* $9/10$ *in the NO case. The algorithm always returns YES on a YES instance.*

Proof. By the above, we just need to sample a expected size $O(\lambda^2)$ submatrix from the conditional distribution of having sampled at least one entry from T_a^*. Since $|T_a^*| \geq \beta/10\frac{\epsilon n}{2^a \log(1/\epsilon)}$, and since $\lambda = \Theta(\frac{\beta^2\zeta^2\log(1/\epsilon)}{2^a\epsilon})$, we see that this requires a total of k samples of expected size $O(\lambda^2)$, where

$$\begin{aligned} k = (n/|T_a^*|)/\lambda &\leq (\frac{2^a 10\log(1/\epsilon)}{\beta\epsilon})(\frac{2^a\epsilon}{\beta^2\zeta^2\log(1/\epsilon)}) \\ &\leq 10\frac{2^{2a}}{\beta^3\zeta^2} \end{aligned} \tag{9.17}$$

Thus the total complexity is $O(k\lambda^2)$, and we have

$$\begin{aligned} k\lambda^2 &\leq 10\frac{2^{2a}}{\beta^3\zeta^2}(\frac{\beta^4\zeta^4\log^2(1/\epsilon)}{2^{2a}\epsilon^2}) \\ &\leq 10\frac{\beta\zeta^2\log^2(1/\epsilon)}{\epsilon^2} \\ &= O(\frac{\zeta^2\log^2(1/\epsilon)}{\epsilon^2}) \end{aligned} \tag{9.18}$$

we use the fact that we can set $\zeta = O(\log^2(1/\epsilon))$. Finally, note that we do not know β or 2^a, but we can guess the value of λ in powers of 2, which is at most $O(\frac{\zeta^2}{\epsilon^2})$, and then set k to be the value such that $k\lambda^2$ is within the above allowance. This blows up the complexity by a $\log(1/\epsilon)$ factor to do the guessing.

$\square$

Case 2: Spread Negative Mass and Main Theorem

In the prior section, we saw that if the quadratic form $x^T\mathbf{A}x$ satisfies the condition for being in Case 1, we could obtain a $\tilde{O}(1/\epsilon^2)$ query algorithm for finding a principal submatrix $A_{T\times T}$ such that $y^\top\mathbf{A}_{T\times T}y < 0$ for some vector y. Now recall that $S = \{i \in [n] \ : \ |x_i|^2 \leq \frac{1}{\epsilon n}\}$, and let

$T_a = \{i \in [n] \ : \ \frac{1002^{a-1}}{\epsilon n} \leq |x_i|^2 \leq \frac{1002^a}{\epsilon n}\}$ for $a \geq 1$. Recall that the definition of Case 1 was that $x_S^\top \mathbf{A} x_{T_a} + x_{T_a}^\top \mathbf{A} x_S \leq -\epsilon n/(10\log(1/\epsilon))$ for some $2^a \geq 10^6\zeta^3$. In this section, we demonstrate that if this condition does not hold, then we will also obtain a $\tilde{O}(1/\epsilon^2)$ query algorithm for the problem.

Thus, suppose now that we are in Case 2; namely that $x_S \mathbf{A} x_{T_a} + x_{T_a} \mathbf{A} x_S > -\epsilon n/(10\log(1/\epsilon))$ for all $2^a \geq 10^6\zeta^3$. Now let $T^+ = \cup_{2^a > 10^6\zeta^3} T_a$ and let $T^- = \cup_{2^a \leq 10^6\zeta^3} T_a$. Let $S^* = S \cup T^-$. We now observe an important fact, which sates that if we are not in Case 1, then $x_{S^*}\mathbf{A}x_{S^*}$ contributes a substantial fraction of the negativeness in the quadratic form.

Fact 9.3.19. *Suppose we are in Case 2: meaning that $x_S^\top \mathbf{A} x_{T_a} + x_{T_a} \mathbf{A} x_S > -\epsilon n/(10\log(1/\epsilon))$ for all $2^a \geq 10^6\zeta^3$. Then we have $x_{S^*}^\top \mathbf{A} x_{S^*} \leq -\epsilon n/2$.*

Proof. Notice that this implies that $x_S^\top \mathbf{A} x_{T^+} + x_{T^+} \mathbf{A} x_S \geq -\epsilon n/10$, since there are at most $\log(1/\epsilon)$ level sets included in T^+ by Proposition 9.3.3. Note since the contribution of $|x_{T^+}^\top \mathbf{A} x_{T^+}| \leq -\epsilon n 10^{-6}/\zeta^3$ and $|x_{T^-}\mathbf{A}x_{T^+} + x_{T^+}\mathbf{A}x_{T^-}| \leq \sqrt{|T^-||T^+|} \leq \epsilon n/100$ by Proposition 9.3.10. Thus if $x^\top \mathbf{A} x \leq -\epsilon n$ to begin with, it follows that we must have

$$\begin{aligned} x_{S^*}^\top \mathbf{A} x_{S^*} &\leq x^\top A x - \Big((x_S^\top \mathbf{A} x_{T^+} + x_{T^+}\mathbf{A}x_S) - (x_{T^+}^\top \mathbf{A} x_{T^+}) - (x_{T^-}\mathbf{A}x_{T^+} + x_{T^+}\mathbf{A}x_{T^-})\Big) \\ &\leq -\epsilon n + \epsilon n/10 + +\epsilon n 10^{-6}/\zeta^3 \epsilon n/100 \\ &< -\epsilon n/2 \end{aligned} \tag{9.19}$$

□

We now proceed by analyzing the result of sampling a principal submatrix from the quadratic form $x_{S^*}^\top \mathbf{A} x_{S^*}$, which by the prior fact is already sufficently negative. Specifically, we will demonstrate that the variance of the standard estimator from Lemma 9.3.7, and specifically Corollary 9.3.8, is already sufficiently small to allow for a single randomly chosen $O(1/\epsilon) \times O(1/\epsilon)$ principal submatrix of $\mathbf{A}$ to have negative quadratic form with x_{S^*} with good probability. In order to place a bound on the variance of this estimator and apply Corollary 9.3.8, we will need to bound the row and column contributions of the quadratic form $x_{S^*}^\top \mathbf{A}_{S^* \times S^*} x_{S^*}$, which we now formally define.

Definition 9.3.20. *For $i \in [n]$, define the row and column contributions of i within S^* as $\mathcal{R}_i^* = \sum_{j \in S^* \setminus i} x_i \mathbf{A}_{i,j} x_j$ and $\mathcal{C}_i^* = \sum_{j \in S^* \setminus i} x_j \mathbf{A}_{j,i} x_i$ respectively.*

Recall that the *total* row and column contributions of i are defined via $\mathcal{R}_i = \sum_{j \in [n] \setminus i} x_i \mathbf{A}_{i,j} x_j$ and $\mathcal{C}_i = \sum_{j \in [n] \setminus i} x_j \mathbf{A}_{j,i} x_i$ respectively, and recall that we have $\mathcal{R}_i + \mathcal{C}_i \leq 0$ for all $i \in [n]$ by Fact 9.3.5

Proposition 9.3.21. *We have* $\sum_{i \in S^*} (\mathcal{R}^*_i + \mathcal{C}^*_i)^2 \leq 10^9 \cdot \zeta^3 \epsilon n$.

Proof. The proof proceeds similarly to Proposition 9.3.14. Let $z^*, z, z^- \in \mathbb{R}^{|S^*|}$ be defined for $i \in S^*$ via $z^*_i = \mathcal{R}^*_i + \mathcal{C}^*_i$, $z_i = \mathcal{R}_i + \mathcal{R}_i$, and $z^- = z - z$. Notice that our goal is to bound $\|z^*\|_2^2$, which by triangle inequality satisfies $\|z^*\|_2^2 \leq 2\left(\|z\|_2^2 + \|z^-\|_2^2\right)$. First note that

$$\begin{aligned} \|z^-\|_2^2 &= \sum_{i \in S^*} \left(\sum_{j \notin S^*} x_i \mathbf{A}_{i,j} x_j + \sum_{j \notin S^*} x_j \mathbf{A}_{j,i} x_i \right)^2 \\ &\leq 2 \sum_{i \in S^*} \left(\sum_{j \notin S^*} x_i \mathbf{A}_{i,j} x_j \right)^2 + 2 \sum_{i \in S^*} \left(\sum_{j \notin S^*} x_j \mathbf{A}_{j,i} x_i \right)^2 \end{aligned} \tag{9.20}$$

Using that $|[n] \setminus S^*| < \epsilon n/100$, we have by Proposition 9.3.11 that $\sum_{i \in S^*} \left(\sum_{j \notin S} x_i \mathbf{A}_{i,j} x_j \right)^2 \leq \epsilon n/100$, so $\|z^-\|_2^2 \leq \epsilon n/25$.

We now bound $\|z\|_1 = \sum_{i \in S} |\mathcal{R}_i + \mathcal{R}_i|$. Recall that we have $\mathcal{R}_i + \mathcal{R}_i \leq 0$ for all $i \in [n]$, which means that $\|z\|_1 \leq \sum_{i \in [n]} |\mathcal{R}_i + \mathcal{R}_i| = |2\langle x, \mathbf{A}x \rangle - 2 \sum_{i \in [n]} \mathbf{A}_{i,i} (x_i)^2| \leq 2\epsilon n$. Next, we bound $\|z\|_\infty$. Notice that $|\mathcal{R}_i + \mathcal{R}_i| = 2|x_i \mathbf{A}_{i,*} x| = 2\epsilon n (x_i)^2 - 2\mathbf{A}_{i,i}(x_i)^2 < 4\epsilon n (x_i)^2$, using that $\mathbf{A}x = -\epsilon n x$, since x is an eigenvectror of $\mathbf{A}$. Since $i \in S^*$, by definition we have $(x_i)^2 \leq \frac{100 \cdot 10^6 \cdot \zeta^3}{\epsilon n}$, thus $\|z\|_\infty \leq 100 \cdot 10^6 \cdot \zeta^3$. It follows that $\|z\|_2^2$ is maximized by having $2\epsilon n/(10^8 \cdot \zeta^3)$ coordinates equal to $10^8 \cdot \zeta^3$, giving $\|z\|_2^2 \leq 2\epsilon n/(10^8 \cdot \zeta^3)(10^8 \cdot \zeta^3)^2 = 2 \cdot 10^8 \cdot \zeta^3 \epsilon n$. It follows then that $\|z\|_2^2 \leq \cdot 10^9 \cdot \zeta^3 \epsilon n$ as needed. □

Theorem 116. *There is an algorithm which, given* $\mathbf{A}$ *with* $\|\mathbf{A}\|_\infty \leq 1$ *such that either* $x^\top \mathbf{A} x \geq 0$ *for all* $x \in \mathbb{R}^n$ *(*`YES` *Case), or* $x^\top \mathbf{A} x \leq -\epsilon n$ *for some* $x \in \mathbb{R}^n$ *with* $\|x\|_2 \leq 1$ *(*`NO` *Case), distinguishes the two cases with probability* $3/4$ *using at most* $\widetilde{O}(\frac{1}{\epsilon^2})$ *queries, and running in time* $\widetilde{O}(1/\epsilon^\omega)$*, where* $\omega < 2.373$ *is the exponent of fast matrix multiplication. Moreover, in the* `YES` *case the, the algorithm always outputs* `YES` *(with probability* 1*), and in the* `NO` *case, the algorithm returns a certificate in the form of a principal submatrix which is not PSD.*

Proof. By Theorem 115 which handles Case 1, we can restrict ourselves to Case 2. Using Fact 9.3.19 as well as Proposition 9.3.21, can apply Corollary 9.3.8 with the vector $y = x_{S^*}$, setting

$c_1 = \Theta(1)$ and $c_2 = \Theta(\zeta^3)$, and $\alpha = O(\sqrt{\epsilon}\zeta^3) = O(\sqrt{\epsilon}\log^6(1/\epsilon))$, to obtain that

$$\mathbf{Var}[\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j] \leq O(\log^{12}(1/\epsilon)\frac{k^2}{n^2})$$

where $k = \tilde{\Theta}(1/\epsilon)$. Since by Proposition 9.3.6 and Fact 9.3.19 , we have $\mathbf{E}\left[\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j\right] \leq \frac{k^2}{4n^2}\langle x_{S^*}, \mathbf{A} x_{S^*}\rangle \leq -\frac{\epsilon k^2}{8n}$, it follows that by repeating the sampling procedure $O(\log^{12}(1/\epsilon))$, by Chebyshev's we will have that at least one sample satisfies $\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j \leq -\frac{\epsilon k^2}{4n}$ with probability $99/100$.

Now note that this random variable does not take into account the diagonal. Thus, it will suffice to bound the contribution of the random variable $\sum_{i\in[n]} \delta_i \mathbf{A}_{i,i}(y_i)^2\ \tilde{O}((1/\epsilon)/n)$. First observe that $\mathbf{E}\left[\sum_{i\in[n]} \delta_i \mathbf{A}_{i,i}(y_i)^2\right] = \frac{k}{n}$. The proof proceeds by a simple bucketing argument; let $\Lambda_i = \{i \in S^* \mid \frac{2^i}{n} \leq (y_i)^2 \leq \frac{2^{i+1}}{n}\}$, and for a single $k \times k$ sampled submatrix, let $T \subset [n]$ be the rows and columns that are sample. Note that $\mathbf{E}\left[|T \cap \Lambda_i|\right] \leq k2^{-i}$, since $|\Lambda_i| \leq 2^{-i}$. Note also that $|\Lambda_i| = 0$ for every i such that $2^i \geq \frac{100^8\zeta^3}{\epsilon}$ by definition of S^* and the fact that y is zero outisde of S^*. Then by Chernoff bounds we have that with probability $\mathbf{Pr}\left[|T \cap \Lambda_i| > \log(1/\epsilon)\max\{k2^{-i}, 1\}\right] \leq 1 - \frac{\epsilon^{10}}{C}$ for some constant C for our choosing. We can then union bound over all $O(\log(1/\epsilon))$ sets Λ_i, to obtain

$$\sum_{i\in[n]} \delta_i \mathbf{A}_{i,i}(y_i)^2 \leq \sum_{i:2^i \leq \frac{100^8\zeta^3}{\epsilon}} \frac{2^{i+1}}{n}|T \cap \Lambda_i| \leq \sum_{i:2^i \leq \frac{100^8\zeta^3}{\epsilon}} \frac{2}{n}\log(1/\epsilon)\max\{k, 2^i\}$$

with probability at least $1 - \frac{\epsilon^9}{C}$. Setting $k = \Theta(\log^6(1/\epsilon)/\epsilon)$, we have that $\sum_{i\in[n]} \delta_i \mathbf{A}_{i,i}(y_i)^2 \leq \sum_{i:2^i \leq \frac{100^8\zeta^3}{\epsilon}} (2/n)\log(1/\epsilon)k = O(\log^2(1/\epsilon)k/n)$. Thus we can condition on $\sum_{i\in[n]} \delta_i \mathbf{A}_{i,i}(y_i)^2 = O(\log^2(1/\epsilon)k/n)$ for all $\tilde{O}(1)$ repetitions of sampling a submatrix. Since at least one sampled submatrix satisfied $\sum_{i\neq j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j \leq -\frac{\epsilon k^2}{4n}$, and since $k = \Theta(\log^6(1/\epsilon)/\epsilon)$, this demonstrates that at least one sampled submatrix will satisfy $\sum_{i,j} y_i \mathbf{A}_{i,j} y_j \delta_i \delta_j < -\frac{\epsilon k^2}{8n}$ as needed in the NO instance. The resulting query complexity is then $O(\log^2(1/\epsilon)k^2) = O(\frac{\log^{24}(1/\epsilon)}{\epsilon^2}) = \widetilde{O}(\frac{1}{\epsilon^2})$ as desired. Finally, for runtime, notice that the main computation is computing the eigenvalues of a $k \times k$ principal submatirx, for $k = \tilde{O}(1/\epsilon)$, which can be carried out in time $\tilde{O}(1/\epsilon^\omega)$ [DDHK07, BVKS19]. □

9.4 PSD Testing with ℓ_2^2 Gap

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a symmetric matrix with eigenvalues $\lambda_{\max} = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = \lambda_{\min}$. In this section, we consider the problem of testing positive semi-definiteness with an ℓ_2^2 gap. Formally, the problem statement is as follows.

Definition 9.4.1 (PSD Testing with ℓ_2^2-Gap.)**.** *Fix, $\epsilon \in (0, 1]$ and let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a symmetric matrix satisfying $\|\mathbf{A}\|_\infty \leq 1$, with the promise that either*

- ***YES Instance**: $\mathbf{A}$ is PSD.*
- ***NO Instance**: $\mathbf{A}$ is ϵ-far from PSD in ℓ_2^2, meaning that $\min_{\mathbf{B} \succeq 0} \|\mathbf{A} - \mathbf{B}\|_F^2 = \sum_{i:\lambda_i < 0} \lambda_i^2 = \epsilon n^2$.*

The PSD Testing problem with ℓ_2^2-gap is to design an algorithm which distinguish these two cases with probability at least $2/3$, using the minimum number of queries possible to the entires of $\mathbf{A}$.

Our algorithm for this problem will query a principal submatrix $\mathbf{A}_{S \times S}$ and return PSD if $\mathbf{A}_{S \times S}$ is PSD, otherwise it will return not PSD. Since all principal submatrices of PSD matrices are PSD, we only need show that if $\mathbf{A}$ is ϵ-far from PSD, then we can find a non-PSD principal submatrix with small size. Note again that this implies that our algorithm will be one-sided. Thus, in the following, we can focus on the case where $\mathbf{A}$ is ϵ-far from PSD. We begin by stating two fundamental observations, which, along with an application of our algorithm from Section 9.3, will allow us to reduce the problem of PSD testing with ℓ_2 gap to the problem of testing certain functions of the *singular values* of $\mathbf{A}$.

Proposition 9.4.2 (PSD matrices are top heavy)**.** *Fix any $n \in \mathbb{N}$, $1 \leq k \leq n$, and $\mathbf{D} \in \mathbb{R}^{n \times n}$. Then if $\mathbf{D}$ is PSD, we have*

$$\sum_{i>k} \sigma_i(\mathbf{D})^2 \leq \frac{1}{k} \left(\mathit{Tr}(\mathbf{D})\right)^2$$

In particular, if $\mathbf{D}$ has bounded entries $\|\mathbf{D}\|_\infty \leq 1$, we have $\sum_{i>k} \sigma_i(\mathbf{D})^2 \leq \frac{1}{k} n^2$.

Proof. We first show that $\sigma_k(\mathbf{D}) \leq k^{-1}\mathrm{Tr}(\mathbf{D})$. To see this, suppose $\sigma_k(\mathbf{D}) > k^{-1}\mathrm{Tr}(\mathbf{D})$. Then because $\mathbf{D}$ is PSD, we would have $\sum_i \sigma_i = \sum_i \lambda_i = \mathrm{Tr}(\mathbf{A}) > k \cdot k^{-1}\mathrm{Tr}(\mathbf{D})$, a contradiction. Thus, $\sigma_i(\mathbf{D}) \leq k^{-1}\mathrm{Tr}(\mathbf{D})$ for all $i \geq k$. Using this and the bound $\sum_{i>k} \sigma_i(\mathbf{D}) \leq \mathrm{Tr}(\mathbf{D})$, it follows that the quantity $\sum_{i>k} \sigma_i(\mathbf{D})^2$ is maximimized by having k singular values equal to $\mathrm{Tr}(\mathbf{D})/k$, yielding $\sum_{i>t} \sigma_i(\mathbf{D})^2 \leq k \cdot (\mathrm{Tr}(\mathbf{D})/k)^2 = k^{-1}(\mathrm{Tr}(\mathbf{D}))^2$ as needed. □

Proposition 9.4.3. *Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be a symmetric matrix such that $\|\mathbf{D}\|_\infty \leq 1$, and let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ be its singular values. Suppose $\mathbf{D}$ is at least ϵ-far in L_2 from PSD, so that $\sum_{i:\lambda_i(\mathbf{D})<0} \lambda_i^2(\mathbf{D}) \geq \epsilon n^2$, and suppose further that $\min_i \lambda_i(\mathbf{D}) > -\frac{1}{2k}n$ for any $k \geq \frac{2}{\epsilon}$. Then we have*

$$\sum_{i>k} \sigma_i^2(\mathbf{D}) > \frac{\epsilon}{2} n^2$$

Proof. Let $W \subseteq [n]$ be the set of values $i \in [n]$ such that $\lambda_i < 0$. Let $W' \subseteq [n]$ be the set of values $i \in [n]$ such that $\sigma_i < \frac{1}{2k}n$. By assumption: $\sum_{i \in W'} \sigma_i^2 \geq \sum_{i \in W} \lambda_i^2 \geq \epsilon n^2$. Now $\sum_{i \in W'} \sigma_i^2 = \sum_{i \in W', i \leq k} \sigma_i^2 + \sum_{i \in W', i > k} \sigma_i^2$, so the fact that $|\sigma_i| \leq (1/2k)n$ for every $i \in W'$, we have that $\sum_{i \in W', i \leq k} \sigma_i^2 \leq k(n/(2k))^2 = n^2/4k < \epsilon n^2/2$. Thus we must have $\sum_{i \in W, i > t} \sigma_i^2 > \epsilon n^2/2$, giving

$$\begin{aligned} \sum_{i>k} \sigma_i^2 &\geq \sum_{i \in W', i > k} \sigma_i^2 \\ &> \epsilon n^2/2 \end{aligned} \tag{9.21}$$

as required. □

9.4.1 Analysis of the Algorithm

Our analysis will require several tools, beginning with the following interlacing lemma.

Lemma 9.4.4 (Dual Lidskii Inequality, [Tao11] Chapter 1.3)**.** *Let $\mathbf{M}_1, \mathbf{M}_2$ be $t \times t$ symmetric Matrices, and fix $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. Then we have*

$$\sum_{j=1}^{k} \lambda_{i_j}(\mathbf{M}_1 + \mathbf{M}_2) \geq \sum_{j=1}^{k} \lambda_{i_j}(\mathbf{M}_1) + \sum_{j=1}^{k} \lambda_{n-j+1}(\mathbf{M}_2)$$

We will also need the following result of Rudelson and Vershynin [RV07] on the decay of spectral norms of random submatrices.

Proposition 9.4.5 ([RV07])**.** *Let $\mathbf{A} \in \mathbb{R}^{n \times m}$ be a rank r matrix with maximum Euclidean row norm bounded by M, in other words $\max_i |(\mathbf{A}\mathbf{A}^\top)_{i,i}| \leq M$. Let $Q \subset [n]$ be a random subset of rows of $\mathbf{A}$ with expected cardinality q. Then there is a fixed constant $\kappa \geq 1$ such that*

$$\mathbf{E}\left[\|\mathbf{A}_{Q\times[m]}\|_2\right] \leq \kappa(\sqrt{\delta}\|\mathbf{A}\|_2 + \sqrt{\log q}M)$$

Finally, we will need a generalized Matrix Chernoff bound for the interior eigenvalues of sums of random matrices, which was derived by Gittens and Tropp [GT11].

Theorem 117 (Interior Eigenvalue Matrix Chernoff, Theorem 4.1 of [GT11])**.** *Consider a finite sequence $\{\mathbf{X}_j\}$ of independent, random, positive-semidefinite matrices with dimension m, and assume that $\|\mathbf{X}_j\|_2 \leq L$ for some value L almost surely. Given an integer $k \leq n$, define*

$$\mu_k = \lambda_k\left(\sum_j \mathbf{E}\left[\mathbf{X}_j\right]\right)$$

then we have the tail inequalities

$$\begin{cases} \mathbf{Pr}\left[\lambda_k(\sum_j \mathbf{X}_j) \geq (1+\delta)\mu_k\right] \leq (n-k+1)\cdot\left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^{\mu_k/L} & \text{for } \delta > 0 \\ \mathbf{Pr}\left[\lambda_k(\sum_j \mathbf{X}_j) \leq (1-\delta)\mu_k\right] \leq k\cdot\left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu_k/L} & \text{for } \delta \in [0,1) \end{cases} \tag{9.22}$$

The Algorithm. Our first step is to run the ℓ_∞-gap algorithm of Section 9.3 with $\epsilon_0 = \frac{2}{k}$, where we set $k = \frac{2\cdot 400^2\kappa^4}{\epsilon}$, where $\kappa \geq 1$ is the constant in Proposition 9.4.5. This allows us to assume that $\lambda_i \geq -\epsilon_0 n/1000 \geq -\frac{1}{2k}n$ for all i, otherwise we have a $\widetilde{O}(1/\epsilon^2)$-query algorithm from the Section 9.3, and since our target complexity is $\widetilde{O}(1/\epsilon^4)$, we can safely disregard the cost of running this algorithm in parallel. We begin by demonstrating that the Frobenius norm of $\mathbf{SA}$ is preserved (up to scaling), where $\mathbf{S}$ is a random row sampling matrix with sufficiently many rows.

Proposition 9.4.6. *Let $\mathbf{M} \in \mathbb{R}^{m\times m}$. Fix $t \geq 1$ and let $\mathbf{S}$ be a row sampling matrix which samples each row of $\mathbf{M}$ with probability $p = \frac{t}{m}$, and let $\mathbf{S} \in \mathbb{R}^{t_0\times m}$ be a row sampling matrix drawn from this distribution, where $\mathbf{E}\left[t_0\right] = t$. Then we have*

$$\mathbf{E}\left[\frac{1}{p}Tr(\mathbf{SMS}^\top)\right] = \sum_i \lambda_i(\mathbf{M}) = Tr(\mathbf{M})$$

and

$$Var\left(\frac{1}{p}Tr(\mathbf{SMS}^\top)\right) \leq \frac{m}{t}\sum_i \mathbf{M}_{i,i}^2$$

Proof. For $i \in [m]$, let $\delta_i \in \{0,1\}$ indicate that we sample row i. We have $\mathbf{E}\left[\mathrm{Tr}(\mathbf{SMS}^\top)\right] = \frac{1}{p}\mathbf{E}\left[\sum_{i=1}^n \delta_i \mathbf{M}_{i,i}\right] = \mathrm{Tr}(\mathbf{M})$. Moreover,

$$\begin{aligned}\mathrm{Var}\left(\frac{1}{p}\mathrm{Tr}(\mathbf{SMS}^\top)\right) &\leq \frac{1}{p^2}\sum_{i=1}^n \delta_i \mathbf{M}_{i,i} - (\mathrm{Tr}(\mathbf{M}))^2 \\ &\leq \sum_{i\neq j}\mathbf{M}_{i,i}\mathbf{M}_{j,j} + \frac{1}{p}\sum_i \mathbf{M}_{i,i}^2 - (\mathrm{Tr}(\mathbf{M}))^2 \\ &\leq \frac{1}{p}\sum_i \mathbf{M}_{i,i}^2\end{aligned} \tag{9.23}$$

as stated. □

We now fix $t = \Theta(\log(1/\epsilon)/\epsilon^2)$, and draw row independent sampling matrices $\mathbf{S}, \mathbf{T}$ with an expected t rows. Let $S, T \subset [n]$ be the rows and columns sampled by $\mathbf{S}, \mathbf{T}^\top$ respectively. We then compute $\mathbf{Z} = \mathbf{SAT}^\top$ with an expected $O(t^2)$ queries. Finally, we query the principal submatrix $\mathbf{A}_{(S\cup T)\times(S\cup T)}$, and test whether $\mathbf{A}_{(S\cup T)\times(S\cup T)}$ is PSD. Clearly if $\mathbf{A}$ is PSD, so is $\mathbf{A}_{(S\cup T)\times(S\cup T)}$, so it suffices to anaylzie the `NO` case, which we do in the remainder.

Lemma 9.4.7. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be ϵ-far from PSD with $\|\mathbf{A}\|_\infty \leq 1$. Then let $\mathbf{Z} = \mathbf{SAT}^\top$ be samples as described above, so that $\mathbf{Z}$ has an expected $t = \Theta(\log(1/\epsilon)/\epsilon^2)$ rows and columns, where t is scaled by a larger enough constant, and let $k = \frac{2\cdot 400^2\kappa^4}{\epsilon}$, where $\kappa \geq 1$ is the constant in Proposition 9.4.5. Suppose further that $\sigma_{k+1}(\mathbf{A}) \leq 10n/k$. Then with probability $19/20$, we have*

$$\frac{n^2}{t^2}\sum_{i>k}\sigma_i^2(\mathbf{Z}) > \epsilon n^2/16$$

Proof. Now write $\mathbf{A} = \mathbf{U}\Lambda\mathbf{V}^\top$, $\mathbf{A}_k = \mathbf{U}\Lambda_k\mathbf{V}^\top, \mathbf{A}_{-k} = \mathbf{U}\Lambda_{-k}\mathbf{V}^\top$. Then $\mathbf{A} = \mathbf{A}_k + \mathbf{A}_{-k}$, and the rows of $\mathbf{A}_k$ are orthogonal to the rows of $\mathbf{A}_{-k}$. Note that this implies that $\|\mathbf{A}_{i,*}\|_2^2 = \|(\mathbf{A}_k)_{i,*}\|_2^2 + \|(\mathbf{A}_{-k})_{i,*}\|_2^2$ for each $i \in [n]$ by the Pythagorean theorem, and since $\|\mathbf{A}\|_\infty \leq 1$ we have $\|(\mathbf{A}_{-k})_{i,*}\|_2^2 \leq n$.

Now set $\mathbf{M}_1 = \mathbf{S}\mathbf{A}_k\mathbf{A}_k^\top\mathbf{S}^\top$, and $\mathbf{M}_2 = \mathbf{S}\mathbf{A}_{-k}\mathbf{A}_{-k}^\top\mathbf{S}^\top$. Notice that $\mathbf{M}_1 + \mathbf{M}_2 = \mathbf{S}(\mathbf{A}_k\mathbf{A}_k^\top + \mathbf{A}_{-k}\mathbf{A}_{-k}^\top)\mathbf{S}^\top = \mathbf{SAA}^\top\mathbf{S}^\top$, using the fact that the rows and columns of $\mathbf{A}_k$ are orthogonal to the rows and columns (respectively) of $\mathbf{A}_{-k}$. Let $p = \frac{t}{n}$ be the row sampling probability. Now suppose $\|(\mathbf{A}_{-k})\|_F^2 = \alpha n^2$. Note that we have shown that $\alpha > \epsilon/2$. By Proposition 9.4.6, we have $\mathbf{E}\left[\mathrm{Tr}(\mathbf{M}_2)/p\right] = \sum_{i>k} = \alpha n^2 > \epsilon n^2/2$ for some $\alpha \geq \epsilon/2$, where the last inequality follows from Proposition 9.4.3. Moreover, we have

$$\begin{aligned}\text{Var}\left(\frac{1}{p}\text{Tr}(\mathbf{M}_2)\right) &\leq \frac{1}{p}\sum_i (\mathbf{M}_2)_{i,i}^2 \\ &= \frac{1}{p}\sum_i \|(\mathbf{A}_{-k})_{i,*}\|_2^4\end{aligned} \tag{9.24}$$

It follows that since each row satisfies $\|(\mathbf{A}_{-k})_{i,*}\|_2^2 \leq n$ and $\|(\mathbf{A}_{-k})\|_F^2 = \alpha n^2$., the quantity $\sum_i \|(\mathbf{A}_{-k})_{i,*}\|_2^4$ is maximized having αn rows with squared norm equal to n. This yields

$$\begin{aligned}\text{Var}\left(\frac{1}{p}\text{Tr}(\mathbf{M}_2)\right) &\leq \frac{1}{p}\sum_i 2\alpha n \cdot n^2 \\ &\leq 2\frac{\alpha n^4}{t} \\ &\leq \frac{\alpha^2}{100^2}n^4\end{aligned} \tag{9.25}$$

Where in the last line, we used that $t > \frac{4\cdot 100^2}{\epsilon_0} \geq \frac{2\cdot 100^2}{\alpha}$. Then by Chebyshev's inequality, with probability $99/100$, we have $\frac{1}{p}\text{Tr}(\mathbf{M}_2) > \alpha n^2 - (\alpha/10)n^2 = (9/10)\alpha n^2 \geq (9/20)\epsilon n^2$. Call this event $\mathcal{E}_1$, and condition on it now. Next, by Proposition 9.4.5, since $\sigma_{k+1}(\mathbf{A}) \leq 10n/k$ we have $\mathbf{E}\left[\|\mathbf{S}\mathbf{A}_{-k}\|_2\right] \leq \kappa(10\sqrt{tn}/k + \sqrt{2\log(1/\epsilon)}\sqrt{n}) < 20\kappa\sqrt{tn}/k$. Then by Markovs, we have $\|\mathbf{S}\mathbf{A}_{-k}\|_2^2 = \|\mathbf{M}_2\|_2 \leq 200^2\kappa^2 tn/k^2$ with probability $99/100$, which we condition on now, and call this event $\mathcal{E}_2$. Then by the Dual Lidskii inequality 9.4.4, we have

$$\begin{aligned}\frac{1}{p}\sum_{j>k}\lambda_j(\mathbf{M}_1+\mathbf{M}_2) &\geq \frac{1}{p}\left(\sum_{j>k}\lambda_j(\mathbf{M}_2)\right) \\ &\geq \frac{1}{p}(\text{Tr}(\mathbf{M}_2) - k\|\mathbf{M}_2\|_2) \\ &\geq (9/20)\epsilon n^2 - 200^2\kappa^2 n^2/k \\ &\geq \epsilon n^2/4\end{aligned} \tag{9.26}$$

using that $k > \frac{2\cdot 400^2\kappa^4}{\epsilon}$. Now let $\mathbf{W} = \frac{1}{\sqrt{p}}(\mathbf{S}\mathbf{A})^\top$, and note that we took the transpose, so $\mathbf{W}$ has n rows and t_1 columns, where $\mathbf{E}\left[t_1\right] = t$. Now by Chernoff bounds, with probability $99/100$ we have $t_2 \leq 2t$; call this event $\mathcal{E}_3$ and condition on it now. The above demonstrates that $\frac{1}{p}\sum_{j>k}\lambda_j(\mathbf{M}_1+\mathbf{M}_2) = \sum_{j>k}\lambda_j^2(\mathbf{W}) \leq \epsilon n^2/4$. Now note that $\sigma_{k+1}(\mathbf{W}) = \frac{1}{\sqrt{p}}(\sigma_{k+1}(\mathbf{S}\mathbf{A}_k + \mathbf{S}\mathbf{A}_{-k}) < \frac{1}{\sqrt{p}}\|\mathbf{S}\mathbf{A}_{-k}\|_2 \leq 200\kappa n/k$, where we used the Weyl inequality for singular values: namely that for any two matrices $\mathbf{A}, \mathbf{B}$ and value i, $|\sigma_i(\mathbf{A}+\mathbf{B}) - \sigma_i(\mathbf{A})| \leq \|\mathbf{B}\|_2$, and using that

$\mathbf{SA}_k$ is rank at most k, so $\sigma_{k+1}(\mathbf{SA}_k) = 0$.

Now draw a random row sampling matrix $\mathbf{T}$ with an expected t rows, and write $\mathbf{N}_1 = \mathbf{T}\mathbf{W}_k\mathbf{W}_k^\top\mathbf{T}$ and $\mathbf{N}_2 = \mathbf{T}\mathbf{W}_{-k}\mathbf{W}_{-k}^\top\mathbf{T}$, and note again that $\mathbf{N}_1 + \mathbf{N}_2 = \mathbf{T}\mathbf{W}\mathbf{W}^\top\mathbf{T}$. Moreover, the rows of $\mathbf{W}_k$ live in a subspace orthogonal to the rows of $\mathbf{W}_{-k}$, so again by the Pythagorean theorem and boundedness of the entries in $\mathbf{A}$, we have $\|(\mathbf{W}_{-k})_{i,*}\|_2^2 \le \frac{1}{p}t_1 \le 2n$ for all $i \in [n]$. Then by Proposition 9.4.6, we have $\mathbf{E}\left[\mathrm{Tr}(\mathbf{N}_2)/p\right] = \|\mathbf{W}_{-k}\|_F^2 = \alpha n^2 \ge \epsilon n^2/4$, and

$$\begin{aligned}\mathrm{Var}\left(\frac{1}{p}\mathrm{Tr}(\mathbf{N}_2)\right) &\le \frac{1}{p}\sum_{i=1}^n \|(\mathbf{W}_{-k})_{i,*}\|_2^4 \\ &\le \frac{1}{p}n^3 \\ &\le \frac{1}{t}n^4 \\ &\le \frac{\epsilon^2}{100^2}n^4\end{aligned} \tag{9.27}$$

Then by Chebyshev's inequality, with probability $99/100$, we have $\frac{1}{p}\mathrm{Tr}(\mathbf{N}_2) > \epsilon n^2/4 - (\epsilon/10)n^2 = \epsilon n^2/8$. Call this event $\mathcal{E}_4$, and condition on it now. Now as shown above, we have $\|W_{-k}\|_2 \le 200\kappa n/k$, thus by Proposition 9.4.5 we have $\mathbf{E}\left[\|\mathbf{T}\mathbf{W}_{-k}\|_2\right] \le \kappa(200\kappa\sqrt{tn}/k + 4\sqrt{\log(1/\epsilon)}\sqrt{n}) \le 400\kappa^2\sqrt{tn}$, again where we take $t = \Theta(\frac{\log(1/\epsilon)}{\epsilon^2})$ with a large enough constant. Then by Markov's inequality, with probability $99/100$ we have $\|\mathbf{N}_2\|_2 \le 400^2\kappa^4 n^2/k^2$, and again by the Dual Lidskii inequality 9.4.4, we have

$$\begin{aligned}\frac{1}{p}\sum_{j>k}\lambda_j(\mathbf{N}_1 + \mathbf{N}_2) &\ge \frac{1}{p}\left(\sum_{j>k}\lambda_j(\mathbf{N}_2)\right) \\ &\ge \frac{1}{p}(\mathrm{Tr}(\mathbf{N}_2) - k\|\mathbf{N}_2\|_2) \\ &\ge \epsilon n^2/8 - 400^2\kappa^4 n^2/k \\ &\ge \epsilon n^2/16\end{aligned} \tag{9.28}$$

Using that $k \ge \frac{2\cdot 400^2\kappa^4}{\epsilon}$. Note moreover that

$$\frac{1}{p}\sum_{j>k}\lambda_j(\mathbf{N}_1 + \mathbf{N}_2) = \frac{1}{p}\sum_{j>k}\sigma_j^2(\mathbf{TW}) = \frac{1}{p^2}\sum_{j>k}\sigma_j^2(\mathbf{SA}^\top\mathbf{T}^\top)$$

Using that $\mathbf{A} = \mathbf{A}^\top$ so that $\mathbf{Z} = \mathbf{SA}^\top\mathbf{T}^\top$ we conclude that $\frac{1}{p^2}\sum_{i>k}\sigma_i^2(\mathbf{Z}) = \frac{n^2}{t^2}\sum_{i>k}\sigma_i^2(\mathbf{Z}) > \epsilon n^2/16$ as desired. Note that we conditioned on $\mathcal{E}_i$ for $i = 1, 2, 3, 4, 5$, each of which held with probability $99/100$, thus the result holds with probability $19/20$ by a union bound.

□

We will now address the case where $\sigma_k(\mathbf{A}) > 10n/k$.

Lemma 9.4.8. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be ϵ-far from PSD with $\|\mathbf{A}\|_\infty \leq 1$. Then let $\mathbf{Z} = \mathbf{SAT}^\top$ be samples as described above, so that $\mathbf{Z}$ has an expected $t = \Theta(\log(1/\epsilon)/\epsilon^2)$ rows and columns, where t is scaled by a larger enough constant, and let $k = \frac{2\cdot 400^2\kappa^4}{\epsilon}$, where $\kappa \geq 1$ is the constant in Proposition 9.4.5. Suppose further that $\sigma_k(\mathbf{A}) > 10n/k$. Then with probability $49/50$, we have*

$$\frac{n}{t}\sigma_k(\mathbf{Z}) \geq 8n/k$$

Proof. The proof is by application of Theorem 117 twice. We first generate a random row sampling matrix $\mathbf{S}$ with an expected t rows, and bound $\lambda_k((\mathbf{SA})^\top\mathbf{SA}) = \sigma_k^2(\mathbf{SA})$. Let $\mathbf{X}_j \in \mathbb{R}^{n\times n}$ be a random variable such that $\mathbf{X}_j = \mathbf{A}_{(j)}^\top\mathbf{A}_{(j)}$, where $\mathbf{A}_{(j)}$ is the j-th row of $\mathbf{A}$ that was sampled in $\mathbf{S}$. Then $\sum_j \mathbf{X}_j = (\mathbf{SA})^\top\mathbf{SA}$, and $\mathbf{E}\left[\mathbf{X}_j\right] = \frac{t}{n}\sum_{j=1}^n \mathbf{A}_j^\top\mathbf{A}_j = \frac{t}{n}\mathbf{AA}^\top$, where $\mathbf{A}_j$ is the j-th row of $\mathbf{A}$. Moreover, note that $\|\mathbf{X}_j\|_2 \leq \max_i \|\mathbf{A}_{i,*}\|_2^2 \leq n$ for all j, by the boundedness of $\mathbf{A}$. Thus note that $\mu_k = \lambda_k((t/n)\mathbf{A}^\top\mathbf{A}) \geq (t/n)100n^2/k^2 = \frac{100tn}{k^2}$. Thus by the Interior Matrix Chernoff Bound 117, we have that for some constant c:

$$\begin{aligned}\mathbf{Pr}\left[\lambda_k((\mathbf{SA})^\top\mathbf{SA}) \leq .9\mu_k\right] &\leq k\cdot c^{\mu_k/L}\\ &\leq k\cdot c^{\frac{100tn}{k^2}\cdot\frac{1}{n}}\\ &\leq k\cdot e^{-100\log(k)}\\ &\leq 1/1000\end{aligned} \tag{9.29}$$

Where we use $t = \Theta(\frac{\log(1/\epsilon)}{\epsilon^2})$ with a large enough constant. Also condition on the fact that $\mathbf{S}$ has at most $2t$ rows, which holds with probability $999/1000$. Call the union of the above two event $\mathcal{E}_1$, which holds with probability $99/100$, and condition on it now. Given this, we have $\sigma_k^2(\mathbf{SA}) \geq \frac{90tn}{k^2}$. Now again, let $\mathbf{Y}_j = (\mathbf{SA})_{(j)}(\mathbf{SA})_{(j)}^\top$, where $(\mathbf{SA})_{(j)}$ is the j-th column of $\mathbf{SA}$ sampled by the column sampling matrix $\mathbf{T}$. Let $\mathbf{M} = (\mathbf{SA})^\top$. Then again we have $\|\mathbf{Y}_j\|_2 \leq 2t$, using that $\mathbf{SA}$ has at most $2t$ rows, and each entry is bounded by 1. Moreover, $\sum_j \mathbf{Y}_j = \mathbf{TMM}^\top\mathbf{T}^\top$ We also have $\lambda_k(\mathbf{E}\left[\sum_j \mathbf{Y}_j\right]) = \lambda_k(\frac{t}{n}\mathbf{MM}^\top) > \frac{90t^2}{k^2}$. Applying the Interior Matrix Chernoff Bound again, we have that for some constant c:

$$\begin{aligned}\mathbf{Pr}\left[\lambda_k(\mathbf{T}(\mathbf{SA})^\top(\mathbf{SA})\mathbf{T}^\top) \le .9\mu_k\right] &\le k \cdot c^{\mu_k/L} \\ &\le k \cdot c^{\frac{90t^2}{k^2}\cdot\frac{1}{2t}} \\ &\le k \cdot e^{-100\log(k)} \\ &\le 1/1000\end{aligned} \tag{9.30}$$

Call the above event $\mathcal{E}_2$. Conditioned on $\mathcal{E}_\infty \cup \mathcal{E}_2$, which hold together with probability $49/50$, we have that $\sigma_k(\mathbf{SAT}^\top) \le .9\sqrt{\frac{90t^2}{k^2}} > 8t/k$. Since $\mathbf{Z} = \mathbf{SAT}^\top$, we have $\frac{n}{t}\sigma_k(\mathbf{Z}) > 8n/k$ as needed.

□

Theorem 118. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be ϵ-far from PSD with $\|\mathbf{A}\|_\infty \le 1$. Then if $S, T \subset [n]$ are random subsets with expected each size $t = O(\log(1/\epsilon)/\epsilon^2)$, then with probability $9/10$ the principal submatrixx $\mathbf{A}_{(S\cup T)\times(S\cup T)}$ is not PSD.*

Proof. First, by Chernoff bounds, with probability $99/100$ we have $|S \cup T| \le |S| + |T| \le 4t$, which we call $\mathcal{E}_1$ and condition on now. First, consider the case that $\sigma_k(\mathbf{A}) \le 10n/k$, where $k = \frac{2\cdot 400^2\kappa^4}{\epsilon}$. Then by Lemma 9.4.7, with probability $19/20$, we have that $\sum_{i>k}\sigma_i^2(\mathbf{A}_{S\times T}) > \epsilon t^2/16$. Now we first prove the following claim:

Claim 9.4.9. *Let $\mathbf{Z} \in \mathbb{R}^{n\times m}$ be any matrix, and let $\tilde{\mathbf{Z}}$ be a rectangular submatrix of $\mathbf{Z}$. for any Let $\mathbf{Z}_k, \tilde{\mathbf{Z}}_k$ be the truncated SVD of $\mathbf{Z}, \tilde{\mathbf{Z}}$ respectively, for any $1 \le k \le \min\{n, m\}$. Then we have*

$$\|\mathbf{Z} - \mathbf{Z}_k\|_F^2 \ge \|\tilde{\mathbf{Z}} - \tilde{\mathbf{Z}}_k\|_F^2$$

Proof. Note that $\|\mathbf{Z}-\mathbf{Z}_k\|_F^2 \ge \|\tilde{\mathbf{Z}}_k - \mathbf{Z}'_k\|_F^2$, where $\mathbf{Z}'_k$ is the matrix $\mathbf{Z}_k$ restricted to the submatrix containing $\tilde{\mathbf{Z}}$. But $\tilde{\mathbf{Z}}_k$ is the *best* rank-k approximation to $\tilde{\mathbf{Z}}$, so $\|\tilde{\mathbf{Z}} - \tilde{\mathbf{Z}}_k\|_F^2 = \min_{\mathbf{B}\text{rank-k}}\|\tilde{\mathbf{Z}} - \mathbf{B}\|_F^2 \le \|\tilde{\mathbf{Z}}_k - \mathbf{Z}'_k\|_F^2$, using the fact that a submatrix of a rank-k matrix is at most rank k. □

It follows that $\|\mathbf{A}_{(S\cup T)\times(S\cup T)} - (\mathbf{A}_{(S\cup T)\times(S\cup T)})_k\|_F^2 = \sum_{j>k}\sigma_j^2(\mathbf{A}_{(S\cup T)\times(S\cup T)}) \ge \sum_{j>k}\sigma_j^2(\mathbf{A}_{S\times T}) > \epsilon t^2/16 > \epsilon|S\cup T|^2/256$. But note that if $\mathbf{A}_{(S\cup T)\times(S\cup T)}$ was PSD, then we would have $\sum_{j>k}\sigma_j^2(\mathbf{A}_{S\times T}) <\le \frac{16}{k}t^2$, which is a contradiction since $k = \frac{2\cdot 400^2\kappa^4}{\epsilon} > \frac{100^2}{\epsilon}$.

Now consider the case that $\sigma_k(\mathbf{A}) > 10n/k$. Then by Lemma 9.4.8, we have $\sigma_k((\mathbf{A}_{S\times T}) \ge 8t/k$ with probability at least $49/50$. Then $\|\mathbf{A}_{S\times T}\|_{\mathcal{S}_1} \ge \sum_{i=1}^k \sigma_i((\mathbf{A}_{S\times T}) \ge 8t$. Using the fact

that the Schatten norm of a matrix is always at least as large as the Schatten norm of any submatrix (this follows from the fact that the singular values of the submatrix are point-wise dominated by the larger matrix, see Theorem 1 [Tho72]), we have $\|\mathbf{A}_{(S\cup T)\times(S\cup T)}\|_{\mathcal{S}_1} \geq 8t$. But note that if $\mathbf{A}_{(S\cup T)\times(S\cup T)}$ was PSD, then we would have $\|\mathbf{A}_{(S\cup T)\times(S\cup T)}\|_{\mathcal{S}_1} = \mathrm{Tr}(\mathbf{A}_{(S\cup T)\times(S\cup T)}) \leq |S\cup T| \leq 4t$, which is a contradiction. This completes the proof of the theorem. □

Theorem 119. *Fix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with $\|A\|_\infty \leq 1$. There is a non-adaptive sampling algorithm that, with probability $9/10$, correctly distinguishes the case that $\mathbf{A}$ is PSD from the case that $\mathbf{A}$ is ϵ-far from PSD in ℓ_2, namely that $\sum_{i:\lambda_i(\mathbf{A})<0} \frac{\lambda_i^2(\mathbf{A})}{n^2} \geq \epsilon$. The algorithm queries a total of $O(\frac{\log^2(1/\epsilon)}{\epsilon^4})$ entries of $\mathbf{A}$, and always correctly classifies $\mathbf{A}$ as PSD if $\mathbf{A}$ is indeed PSD. Moreover, the algorithm runs in time $\tilde{O}(1/\epsilon^{2\omega})$, where $\omega < 2.373$ is the exponent of fast matrix multiplication.*

Proof. We first apply the algorithm of Section 9.3 with $\epsilon_0 = \frac{2}{k}$, which as discussed allows us to assume that $\lambda_i \geq -\epsilon_0 n/1000 \geq -\frac{1}{2k}n$ for all i. The cost of doing so is $\widetilde{\Theta}(1/\epsilon^2)$ queries, and this algorithm also yields one-sided error as desired. The remainder of the theorem follows directly from Theorem 118, using that all principal submatrices of PSD matrices are PSD. Finally, for runtime, notice that the main computation is computing the eigenvalues of a $k \times k$ principal submatirx, for $k = \tilde{O}(1/\epsilon^2)$, which can be carried out in time $\tilde{O}(1/\epsilon^{2\omega})$ [DDHK07, BVKS19]. □

9.5 Lower Bounds

9.5.1 Lower Bound for PSD Testing with ℓ_∞ Gap

We begin by demonstrating a $O(1/\epsilon^2)$ lower bound for the problem of testing postive semi-definiteness with an ℓ_∞ gap. Our lower bound holds even when the algorithm is allowed to adaptively sample entryies of $\mathbf{A}$.

Theorem 120. *Any adaptive or non-adaptive algorithm which receives query access to $A \in \mathbb{R}^{n\times n}$ with $\|\mathbf{A}\|_\infty \leq 1$, and distinguishes with probability at least $2/3$ whether*

- *$\mathbf{A}$ is PSD.*
- *$x^T\mathbf{A}x < -\epsilon n$ for some unit vector $x \in \mathbb{R}^n$ and $\epsilon \in (0,1)$*

must make $\Omega(1/\epsilon^2)$ queries to A.

Proof. We construct two distributions $\mathcal{D}_1, \mathcal{D}_2$ over matrices, and draw the input A from the mixture $(\mathcal{D}_1 + \mathcal{D}_2)/2$. $\mathcal{D}_1$ is supported on one matrix: the zero matrix $\mathbf{0}^{n\times n}$, which is PSD. Now set $t = 2\epsilon^2 n$ and let $B \in \mathbb{R}^{n\times n}$ be the matrix given by

$$\mathbf{B} = \begin{bmatrix} 0 & -\mathbf{1}^{n-t\times t} \\ -\mathbf{1}^{t\times n-t} & -\mathbf{1}^{t\times t} \end{bmatrix}$$

Where $-\mathbf{1}^{n\times m}$ is the $n \times m$ matrix consisting of a -1 in each entry. Now let $x \in \mathbb{R}^{n\times n}$ be defined by $x_i = 1$ for $i = 1, 2, \ldots, n-t$, and let $x_j = 1/\epsilon$ for $j > n-t$. Then note that $x^T\mathbf{B}x < -\frac{1}{\epsilon} \cdot 2\epsilon^2 n^2 < -\epsilon n\|x\|_2^2$, thus $\mathbf{B}$ is ϵ-far from PSD in ℓ_∞ gap. To sample $A \sim \mathcal{D}_1$, we set $\mathbf{A} = \mathbf{P}_\Sigma \mathbf{B}\mathbf{P}_\sigma^T$, where $\mathbf{P}_\sigma$ is a randomly drawn permutation matrix, namely $\sigma \sim S_n$ uniformly at random. Notice that to distinguish $A \sim \mathcal{D}_1$ from $A \sim \mathcal{D}_2$, the algorithm must read a non-zero entry. By Yao's min-max principle, we can assume that there is a deterministic algorithm that solves the problem with probability $2/3$ over the randomness of the distribution. Fix any $k < 1/(100\epsilon^2)$, and let $s_1, s_2, \ldots s_k$ be the adaptive sequence of entries it would sample if $A_{s_i} = 0$ for each $i = 1, 2, \ldots, k$. Then then the probability that any of the the s_i's land in a row or a column of $\mathbf{A} = \mathbf{P}_\Sigma \mathbf{B}\mathbf{P}_\sigma^T$ with non-zero entries is at most $1/50$. Thus with probability $49/50$ under input from $A \sim \mathcal{D}_2$, the algorithm will output the same value had A been the all zero matrix. Thus the algorithm succeeds with probability at most $51/100$ when A is drawn from the mixture, demonstrating that $\Omega(1/\epsilon^2)$ samples are required for probability $2/3$ of success. □

9.5.2 Lower Bound for PSD Testing with ℓ_2 Gap

We now present our main lower bound for PSD testing. Our result relies on the construction of explicit graphs with gaps in their spectrum, which have the property that they are indistinguishable given only a small number of queries to their adjacency matrices. Our lower bound is in fact a general construction, which will also result in lower bounds for testing the Schatten 1 norm, Ky-Fan norm, and cost of the best rank k approximation.

Roadmap In the following, we will first introduce the notation and theory required for the section, beginning with the notion of subgraph equivalence of matrices. We then construct our hard distributions $\mathcal{D}_1, \mathcal{D}_2$, and prove our main conditional results, Lemma 9.5.5, which demonstrates a lower bound for these hard distributions conditioned on the existence of certain pairs of subgraph equivalent matrices. Finally, we prove the existence of such matrices, which is carried out in the following Section 9.5.2. Putting these pieces together, we obtain our main lower bound in Theorem 123.

Preliminaries and Notation In the following, it will be useful to consider *signed graphs*. A signed graph Σ is a pair $(|\Sigma|, s)$, where $|\Sigma| = (V, E)$ is a simple graph, called the *underlying graph*, and $s : E \to \{1, -1\}$ is the *sign function*. We will sometimes abbreviate the signs equivalently as $\{+, -\}$. We will write E^+, E^- to denote the set of positive and negative edges. If Σ is a signed graph, we will often write $\Sigma = (V(\Sigma), E(\Sigma))$, where $E(\Sigma)$ is a set of *signed* edges, so $E(\Sigma) \subset \binom{|V(\Sigma)|}{2} \times \{+, -\}$ with the property that for each $e \in \binom{|V(\Sigma)|}{2}$, at most one of $(e, +,), (e, -)$ is contained in $E(\Sigma)$. For a signed graph G on n vertices, let $\mathbf{A}_G \in \{1, 0, -1\}^{n \times n}$ be its adjacency matrix, where $(\mathbf{A}_G)_{i,j}$ is the sign of the edge $e = (v_i, v_j)$ if $e \in E(G)$, and is 0 otherwise.

For a graph H, let $\|H\|$ denote the number of vertices in H. For any simple (unsigned) graph G, let $\overline{G}$ be the signed graph obtained by having $E^+(\overline{G}) = E(G)$, and $E^-(\overline{G}) = \binom{|V|}{2} \setminus E(G)$. In other words, $\overline{G}$ is the complete signed graph obtained by adding all the edges in the complement of G with a negative sign, and giving a positive sign the edges originally in G. We remark that the negation of the adjacency matrix of $\overline{G}$ is known as the *Seidel matrix* of G. In what follows, we will often not differentiate between a signed graph G and its signed adjacency matrix $\mathbf{A}_G$. For graphs G, H, let $G \oplus H$ denote the disjoint union of two graphs G, H We will assume familiarity with basic group theory. For groups G, H, we write $H \leq G$ if H is a subgroup of G. For a set T, let 2^T denote the power set of T. Throughout, let S_n denote the symmetric group on n letters. For two signed graphs Σ, H, let $\mathcal{F}_H(\Sigma) = \{G = (V(\Sigma), E(G)) \mid E(G) \subseteq E(\Sigma), G \cong H\}$ be the set of signed subgraphs of Σ isomorphic to H. For a permutation $\sigma \in S_n$, we write $\mathbf{P}_\sigma \in \mathbb{R}^{n \times n}$ to denote the row permutation matrix associated with σ. For $k \geq 3$, let C_k denote the cycle graph on k vertices.

For signed graphs G, H, a signed graph isomorphism (or just isomorphism) is a graph isomorphism that preserve the signs of the edges. For any set $U \subset [n] \times [n]$ and matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, we write $\mathbf{A}_U$ to denote the matrix obtained by setting the entries $(A_U)_{i,j} = A_{i,j}$ for $(i, j) \in U$, and $(\mathbf{A}_U)_{i,j} = 0$ otherwise. A set $U \subset [n] \times [n]$ is called symmetric if $(i, j) \in U \iff (j, i) \in U$. We call U simple if it does not contain any elements of the form (i, i). We will sometimes refer to a simple symmetric U by the underlying simple undirected graph induced U.

Subgraph Equivalence We now formalize the indistinguishably property which we will require. For matrices $\mathbf{A}, \mathbf{B}$, when thought of as adjacency matrices of graphs, this property can be thought of as a more general version of "locally indistinguishability", in the sense that, for any small subgraph H of $\mathbf{A}$, there is a *unique* subgraph of $\mathbf{B}$ that is isomorphic to H. The following definition is more general, in the sense that a subgraph can also have "zero valued edges", corresponding to the fact that an algorithm can learn of the non-existence of edges, as well as their

existence.

Definition 9.5.1 (Sub-graph Equivalence)**.** *Fix any family $\mathcal{U}$ of symmetric subsets $\mathcal{U} = \{U_i\}_i \in 2^{[n]\times[n]}$, and let $\Gamma \leq S_n$ be a subgroup of the symmetric group on n letters. Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$. Then we say that $\mathbf{A}$ is $(\mathcal{U}, \Gamma)$-subgraph isomorphic to $\mathbf{B}$, and write $\mathbf{A} \cong_{\mathcal{U},\Gamma} \mathbf{B}$, if for every $U_i \in \mathcal{U}$ there is a bijection $\psi_i : \Gamma \to \Gamma$ such that*

$$\left(\mathbf{P}_\sigma \mathbf{A} \mathbf{P}_\sigma^T\right)_{U_i} = \left(\mathbf{P}_{\psi_i(\sigma)} \mathbf{B} \mathbf{P}_{\psi_i(\sigma)}^T\right)_{U_i}$$

for all $\sigma \in \Gamma$. If G, H are two signed graphs on n vertices with adjacency matrices $\mathbf{A}_G, \mathbf{A}_H$, then we say that G is $(\mathcal{U}, \Gamma)$-subgraph equivalent to H, and write $G \cong_{\mathcal{U},\Gamma} H$, if $\mathbf{A}_G \cong_{\mathcal{U},\Gamma} \mathbf{A}_H$.

Note we do not require the U_i's to be simple in the above definition. At times, if $\Gamma = S_n$, then we may omit Γ and just write $G \cong_{\mathcal{U}} H$ or $\mathbf{A} \cong_{\mathcal{U}} \mathbf{B}$.

Example 121. Let G, H be arbitrary graphs on n vertices, and let each $\mathcal{U} = \{U_i\}$ be a simple graph consisting of a single edge. Then $G \cong_{\mathcal{U},S_n} H$ if and only if $|E(G)| = |E(H)|$.

Example 122. Let G, H be arbitrary graphs on n vertices, and let $\mathcal{U} = \{U_i\}$ be a single graph, where U_i is a triangle on any three vertices. Then $G \cong_{\mathcal{U},S_n} H$ if and only if the number of induced subgraphs on three vertices that are triangles, wedges, and single edges, are each the the same in G as in H.

In what follows, we will consider graphs that are $\mathcal{U}$ subgraph isomorphic, for a certain family of classes $\mathcal{U}$, which we now define. In what follows, recall that the matching number $\nu(G)$ of a graph G is the size of a maximum matching in G, or equivalently the maximum size of any subset of pairwise vertex disjoint edges in G.

Definition 9.5.2. *For $1 \leq t \leq n$, let $\mathcal{U}_n^t$ be the set of all undirected, possibly non-simple graphs U_i on n vertices, with the property that after removing all self-loops, U_i does not contains any set of t vertex disjoint edges. Equivalently, after removing all self-loops from U_i, the matching number $\nu(U_i)$ of U_i is less than t.*

In other words, $\mathcal{U}_n^t$ is the set of graphs with no set of t pair-wise non-adjacent edges $e_1, \ldots, e_t$ such that each e_i is not a self loop. Notice by the above definition that $\mathcal{U}_n^t \subset \mathcal{U}_n^{t+1}$. We will also need the following definition.

Definition 9.5.3. *For any $n, m \leq 1$, let $\Gamma_{n,m} \leq S_{nm}$ be the subgroup defined $\Gamma_{n,m} = \{\sigma \in$*

$S_{nm} \mid \sigma(i,j) = (\pi(i), j),\ \pi \in S_n\}$, *where the tuple* $(i,j) \in [n] \times [m]$ *indexes into* $[nm]$ *in the natural way.*

Notice in particular, if $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{D} \in \mathbb{R}^{m \times m}$, then we have

$$\{\mathbf{P}_\sigma(\mathbf{A} \otimes \mathbf{D})\mathbf{P}_\sigma^T \mid \sigma \in \Gamma_{n,m}\} = \{(\mathbf{P}_\pi \otimes \mathbb{I}_m)(\mathbf{A} \otimes \mathbf{D})(\mathbf{P}_\pi \otimes \mathbb{I}_m)^T \mid \pi \in S_n\}$$

Note also by elementary properties of Kronecker products, we have $(\mathbf{P}_\pi \otimes \mathbb{I}_m)(\mathbf{A} \otimes \mathbf{D})(\mathbf{P}_\pi \otimes \mathbb{I}_m)^T = (\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T) \otimes \mathbf{D}$. For such a $\sigma \in \Gamma_{n,m}$, we write $\sigma = \pi \otimes \mathrm{id}$, where $\pi \in S_n$

Lemma 9.5.4. *Fix any* $t, m \geq 1$*, and let* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ *be matrices with* $\mathbf{A} \cong_{\mathcal{U}_n^t, S_n} \mathbf{B}$*, where* $\mathcal{U}_n^t$ *is defined as above, and let* $\mathbf{T} \in \mathbb{R}^{m \times m}$ *be any matrix. Then* $\mathbf{A} \otimes \mathbf{T} \cong_{\mathcal{U}_{nm}^t, \Gamma_{n,m}} \mathbf{B} \otimes \mathbf{T}$*, where* $\Gamma \leq S_{nm}$ *is as defined above.*[10]

Proof. Fix any $U_i' \in \mathcal{U}_{nm}^t$. Note that every edge of U_i' corresponds to a unique edge of a graph on n vertices. This can be seen as every edge of U_i' is of the form $((i_1, j_1), (i_2, j_2))$ where $i_1, i_2 \in [n], j_1, j_2 \in [m]$, which corresponds to the edge $(i_1, i_2) \in [n] \times [n]$. So let $U_i \subset [n] \times [n]$ be the set of all such edges induced by the edges of U_i'. Observe, of course, that many distinct edges of U_i' could result in the same edge of U_i. We claim that $U_i \in \mathcal{U}_n^t$. Suppose this was not the case, and let $e_1, \ldots, e_t \in U_i$ be vertex disjoint non-self loop edges, where $e_\ell = (i_\ell, j_\ell)$, $i_\ell \neq j_\ell$. Then for each $\ell \in [t]$, there must be at least one edge $e_\ell' \in U_i'$ such that $e_\ell' = ((i_\ell, a_\ell)(j_\ell, b_\ell)) \in U_i'$, and we can fix e_ℓ' to be any such edge. Then since each vertex $i_\ell \in [n]$ occured in at most one edge of $e_1, \ldots, e_t$ by assumption, it follows that each vertex $(i_\ell, j_\ell) \in [n] \times [m]$ occurs at most once in $e_1', \ldots, e_t'$, which contradictts the fact that the $U_i' \in \mathcal{U}_{nm}^t$.

Now that we have $U_i \in \mathcal{U}_n^t$, since $\mathbf{A} \cong_{\mathcal{U}_n^t, S_n} \mathbf{B}$ we have a bijection function $\psi_i : S_n \to S_n$ such that $\left(\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T\right)_{U_i} = \left(\mathbf{P}_{\psi_i(\pi)} \mathbf{B} \mathbf{P}_{\psi_i(\pi)}^T\right)_{U_i}$. We now define the mapping $\hat{\psi}_i : \Gamma_{n,m} \to \Gamma_{n,m}$ by $\hat{\psi}_i(\pi \otimes \mathrm{id}) = \psi_i(\pi) \otimes \mathrm{id}$, and show that it satisfies the conditions of Definition 9.5.1. Now note that each $\sigma = \pi \otimes \mathrm{id} \in \Gamma_{n,m}$ satisfies $\mathbf{P}_\sigma = \mathbf{P}_\pi \otimes \mathbb{I}$, and so $\mathbf{P}_\sigma(\mathbf{A} \otimes \mathbf{T})\mathbf{P}_\sigma^T = \mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T \otimes \mathbf{T}$.

We now claim that for any $U_i' \in \mathcal{U}_{nm}^t$, if we construct $U_i \in \mathcal{U}_n^t$ as above, we have that for any matrix $\mathbf{Z} \in \mathbb{R}^{n \times n}$ the non-zero entries of $(\mathbf{Z})_{U_i} \otimes \mathbf{T}$ contain the non-zero entries of $(\mathbf{Z} \otimes \mathbf{T})_{U_i'}$. As a consequence, if $(\mathbf{Z})_{U_i} \otimes \mathbf{T} = (\mathbf{Y})_{U_i} \otimes \mathbf{T}$ for some other matrix $\mathbf{Y} \in \mathbb{R}^{n \times n}$, we also have $(\mathbf{Z} \otimes \mathbf{T})_{U_i'} = (\mathbf{Y} \otimes \mathbf{T})_{U_i'}$. But the claim in question just follows from the construction of U_i', since for every entry $((i_1, j_1), (i_2, j_2)) \in U_i'$ we added the entry $(i_1, i_2) \in U_i$. Now since we have

[10] Note that this fact extends naturally to tensoring with rectangular matrices $\mathbf{T}$.

that $\left(\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T\right)_{U_i} = \left(\mathbf{P}_{\psi_i(\pi)} \mathbf{B} \mathbf{P}_{\psi_i(\pi)}^T\right)_{U_i}$, we also obtain

$$\left(\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T\right)_{U_i} \otimes \mathbf{T} = \left(\mathbf{P}_{\psi_i(\pi)} \mathbf{B} \mathbf{P}_{\psi_i(\pi)}^T\right)_{U_i} \otimes \mathbf{T}$$

which as just argued implies that

$$\left(\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T \otimes \mathbf{T}\right)_{U_i'} = \left(\mathbf{P}_{\psi_i(\pi)} \mathbf{B} \mathbf{P}_{\psi_i(\pi)}^T \otimes \mathbf{T}\right)_{U_i'}$$

Since $\left(\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^T \otimes \mathbf{T}\right)_{U_i'} = \left(\mathbf{P}_\sigma (\mathbf{A} \otimes \mathbf{T}) \mathbf{P}_\sigma^T\right)_{U_i'}$ and $(\mathbf{P}_{\psi_i(\pi)} \mathbf{B} \mathbf{P}_{\psi_i(\pi)}^T \otimes \mathbf{T})_{U_i'} = (\mathbf{P}_{\hat{\psi}(\sigma)} (\mathbf{B} \otimes \mathbf{T}) \mathbf{P}_{\hat{\psi}(\sigma)}^T)_{U_i'}$, it follows that $\mathbf{A} \otimes \mathbf{T} \cong_{\mathcal{U}_{nm}^t, \Gamma_{n,m}} \mathbf{B} \otimes \mathbf{T}$ as required.

□

The Hard Instance. We now describe now distributions, $\mathcal{D}_1, \mathcal{D}_2$, supported on $n \times n$ matrices $\mathbf{A}$ and paramterized by a value $k \geq 1$, such that distinguishing $\mathcal{D}_1$ from $\mathcal{D}_2$ requires $\Omega(k^2)$ samples. The distributions are parameterized by *three* matrices, $(\mathbf{B}, \mathbf{D}, \mathbf{Z})$, which are promised to satisfy the properties that $\mathbf{B}, \mathbf{D} \in \mathbb{R}^{d \times d}$ with $\mathbf{B} \cong_{\mathcal{U}_d^t, S_d} \mathbf{D}$ for some $t \leq d$, and $\mathbf{Z} \in \mathbb{R}^{m \times m}$, where $m = n/(dk)$. Also define $\widetilde{\mathbf{B}} = \mathbf{B} \otimes \mathbf{Z}$, $\widetilde{\mathbf{D}} = \mathbf{D} \otimes \mathbf{Z}$. We now define the distribution. We first define $\mathcal{D}_1$. In $\mathcal{D}_1$, we select a random partition of $[n]$ into $L_1, \ldots, L_k$, where each $|L_i| = n/k$ exactly. Then for each $i \in [k]$, we select a uniformly random $\sigma_i \in \Gamma_{d,m}$ and set $\mathbf{A}_{L_i \times L_i} = \mathbf{P}_{\sigma_i} \widetilde{\mathbf{B}} \mathbf{P}_{\sigma_i}^T$, and the remaining elements of $\mathbf{A}$ are set to 0. In $\mathcal{D}_2$, we perform the same procedure, but set $\mathbf{A}_{L_i \times L_i} = \mathbf{P}_{\sigma_i} \widetilde{\mathbf{D}} \mathbf{P}_{\sigma_i}^T$. So if $\mathbf{A} \sim \frac{\mathcal{D}_1 + \mathcal{D}_2}{2}$, then $\mathbf{A}$ is block-diagonal, with each block having size n/k. We first demonstrate that for any matrices $(\mathbf{B}, \mathbf{D}, \mathbf{Z})$ satisfying the above properties, distinguishing these distributions requires $\Omega(k^2)$ samples. We assume in the following that dk divides n, which will be without loss of generality since we can always embed a small instance of the lower bound with size n' such that $n/2 < n - dk \leq n' \leq n$, and such that dk divides n.

Lemma 9.5.5. *Fix any $1 \leq k, d \leq n$. Let $(\mathbf{B}, \mathbf{D}, \mathbf{Z})$ be any three matrices such that $\mathbf{B}, \mathbf{D} \in \mathbb{R}^{d \times d}$, $\mathbf{B} \cong_{\mathcal{U}_d^t, S_d} \mathbf{D}$ where $t = \log k$, and $\mathbf{Z} \in \mathbb{R}^{m \times m}$, where $m = n/(dk)$. Then any non-adaptive sampling algorithm which receives $\mathbf{A} \sim \frac{\mathcal{D}_1 + \mathcal{D}_2}{2}$ where the distributions are defined by the tuple $(\mathbf{B}, \mathbf{D}, \mathbf{Z})$ as above, and distinguishes with probability at least $2/3$ whether $\mathbf{A}$ was drawn from $\mathcal{D}_1$ or $\mathcal{D}_2$ must sample $\Omega(k^2)$ entries of $\mathbf{A}$.*

Proof. We show that any algorithm cannot distinguish $\mathcal{D}_1$ from $\mathcal{D}_2$ with probability greater than $2/3$ unless it makes at least $\ell > C \cdot k^2$ queries, for some constant $C > 0$. So suppose the algorithm

makes at most $C \cdot k^2/100$ queries in expectation and is correct with probability $2/3$. Then by Markov's there is a algorithm that always makes at most $\ell = Ck^2$ queries which is correct with probability $3/5$. By Yao's min-max principle, there is a determinstic algorithm making this many queries which is correct with probability $3/5$ over the distribution $\frac{\mathcal{D}_1+\mathcal{D}_2}{2}$. So fix this algorithm, which consists of a single subset $U \subset [n] \times [n]$ with $|U| = \ell$.

We now generate the randomness used to choose the partition $L_1, \ldots, L_k$ of $[n]$. Let $U_i = U \cap L_i \times L_i = \{(i,j) \in U \mid i,j \in L_i\}$. Let $\mathcal{E}_i$ be the event that $U_i \in \mathcal{U}^t_{md}$. We first bound $\mathbf{Pr}[\neg\mathcal{E}_i]$, where the probability is over the choice of the partition $\{L_i\}_{i\in[k]}$. For $\neg\mathcal{E}_i$, there must be t pairwise vertex disjoint non-self loop edges $e_1, \ldots, e_t \in U$ such that $e_j = (a_j, b_j)$ and $a_j, b_j \in L_i$. In other words, we must have $2t$ distinct vertices $a_1, b_1, \ldots, a_t, b_t \in L_i$. For a fixed vertex $v \in [n]$, this occurs with probability $1/k$, and the probability that another $u \in [n] \cap L_i$ conditioned on $v \in [n]$ is strictly less than $1/k$ as have have $|L_i| = n/k$ exactly. Thus, the probability that all $2t$ vertices are contained in L_i can then be bounded $\frac{1}{k^{2t}}$. Now there are $\binom{|U|}{t} \leq \ell^t$ possible choices of vertex disjoint edges $e_1, \ldots, e_t \in U$ which could result in $\mathcal{E}_i$ failing to hold, thus $\mathbf{Pr}[\mathcal{E}_i] \geq 1 - \frac{\ell^t}{k^{2t}}$ and by a union bound

$$\begin{aligned}
\mathbf{Pr}\left[\cap_{i=1}^k \mathcal{E}_i\right] &\geq 1 - \frac{\ell^t}{k^{2t-1}} \\
&\geq 1 - \frac{C^t k^{2t}}{k^{2t-1}} \\
&\geq 1 - C^t k \\
&\geq \frac{99}{100}
\end{aligned} \tag{9.31}$$

Where in the last line, we took $C \leq 1/10$ and used the fact that $t = \log(k)$. Then if $\mathcal{E} = \cap_{i=1}^k \mathcal{E}_i$, we have $\mathbf{Pr}[\mathcal{E}] > 99/100$, which we condition on now, along with any fixing of the L_i's that satisfies $\mathcal{E}$. Conditioned on this, it follows that $U_i \in \mathcal{U}_{md}$ for each $i \in [k]$. Using that $B \cong_{\mathcal{U}^t_d, S_n} D$, we can and apply Lemma 9.5.4 to obtain $\mathbf{B} \otimes \mathbf{Z} \cong_{\mathcal{U}^t_{dm}, \Gamma_{d,m}} \mathbf{D} \otimes \mathbf{Z}$. Thus, for each $i \in [k]$ we can obtain a bijection function $\psi_i : \Gamma_{d,m} \to \Gamma_{d,m}$ such that $(\mathbf{P}_{\sigma_i} \widetilde{\mathbf{B}} \mathbf{P}^T_{\sigma_i})_{U_i} = (\mathbf{P}_{\psi_i(\sigma_i)} \widetilde{\mathbf{D}} \mathbf{P}^T_{\psi(\sigma_i)})_{U_i}$ for each $\sigma_i \in \Gamma_{d,m}$. Thus we can create a coupling of draws from $\mathcal{D}_1$ with those from $\mathcal{D}_2$ conditioned on $\mathcal{E}$, so for any possible draw from the remaining randomness of $\mathcal{D}_1$, which consists only of drawing some $(\sigma_1, \ldots, \sigma_k) \in S_d^k$ generating a matrix $\mathbf{A}_1$, we have a unique corresponding draw $(\psi_1(\sigma_1), \ldots, \psi_k(\sigma_k)) \in S_d^k$ of the randomness in $\mathcal{D}_2$ which generates a matrix $\mathbf{A}_2$, such that $(\mathbf{A}_1)_U = (\mathbf{A}_2)_U$. Thus conditioned on $\mathcal{E}$, any algorithm is correct on $\frac{\mathcal{D}_1+\mathcal{D}_2}{2}$ with probability exactly $1/2$. Since $\mathcal{E}$ occured with probability $99/100$, it follows than the algorithm is correct with probability $51/100 < 3/5$, which is a contradiction. Thus we must have $\ell \geq Ck^2 = \Omega(k^2)$

as needed.

□

We are now ready to introduce our construction of the matrices as required in the prior lemma. Recall that $k \geq 3$, let C_k denote the cycle graph on k vertices.

Fact 9.5.6. *Fix any $n \geq 3$. We have $\lambda_{\min}(C_{2n+1}) = -2 + \Theta(1/n^2)$ and $\lambda_{\min}(C_n \oplus C_{n+1}) = -2$.*

Proof. The eigenvalues of the cycle C_ℓ are given by $2\cos(\frac{2\pi t}{\ell})$ [Chu96] for $t = 0, \ldots, \ell - 1$, which yields the result using the fact that $\cos(\pi(1+\epsilon)) = 1 + \Theta(\epsilon^2)$ for small ϵ □

Proposition 9.5.7. *Fix any $n = n_1 + n_2$. For any $t \leq \min\{n_1, n_2\}/4$, we have $C_n \cong_{\mathcal{U}_n^t, S_n} C_{n_1} \oplus C_{n_2}$*

Proof. We begin by fixing any set $U_i \in \mathcal{U}_n^t$. For any signed graph Σ and graph G on n vertices such that the maximum set of vertex disjoint edges in Σ is $t \leq \min\{n_1, n_2\}/4$, let $\mathcal{H}_\Sigma(G) = \{\sigma \in S_n \mid \mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{A}_H, H \subset G\}$ and let $\mathcal{H}_\Sigma^{-1}(G) = \{\sigma \in S_n \mid \mathbf{A}_\Sigma = \mathbf{P}_\sigma \mathbf{A}_H \mathbf{P}_\sigma^T, H \subset G\}$. By Corollary 9.5.11, we have $|\mathcal{H}_\Sigma(\overline{C}_n)| = |\mathcal{H}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})|$ whenever $|\Sigma|$ has no set of at least $\min\{n_1, n_2\}/4$ vertex disjoint edges. Since S_n is a group and has unique inverses, we also have$|\mathcal{H}_\Sigma^{-1}(\overline{C}_n)| = |\mathcal{H}_\Sigma(\overline{C}_n)| = |\mathcal{H}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})| = |\mathcal{H}_\Sigma^{-1}(\overline{C_{n_1} \oplus C_{n_2}})|$.

We now define a function $\psi_i : S_n \to S_n$ such that $(\mathbf{P}_\sigma \mathbf{A}_{\overline{C}_n} \mathbf{P}_\sigma^T)_{U_i} = (\mathbf{P}_{\psi_i(\sigma)} \mathbf{A}_{\overline{C_{n_1} \oplus c_{n_2}}} \mathbf{P}_{\psi_i(\sigma)}^T)_{U_i}$ for every $\sigma \in S_n$. Now fix any signed graph Σ such that $\Sigma = (\mathbf{P}_\sigma \mathbf{A}_{\overline{C}_n} \mathbf{P}_\sigma^T)_{U_i}$ for some $\sigma \in S_n$. Note that the set of $\pi \in S_n$ such that $\Sigma = (\mathbf{P}_\pi A_{\overline{C}_n} \mathbf{P}_\pi^T)_{U_i}$ is precisely $\mathcal{H}_\Sigma^{-1}(\overline{C}_n)$. Similarly, the set $\pi \in S_n$ such that $\Sigma = (\mathbf{P}_\pi A_{\overline{C_{n_1} \oplus C_{n_2}}} \mathbf{P}_\pi^T)_{U_i}$ is precisely $\mathcal{H}_\Sigma^{-1}(\overline{C_{n_1} \oplus C_{n_2}})$. Also, by construction of $\mathcal{U}_n^t$, we know that the maximum set of vertex disjoint edges in U_i, and therefore in Σ is $t \leq \min\{n_1, n_2\}/4$, So by the above, we know there is a bijection $\psi_i^\Sigma : \mathcal{H}_\Sigma^{-1}(\overline{C}_n) \to \mathcal{H}_\Sigma^{-1}(\overline{C_{n_1} \oplus C_{n_2}})$ for every such realizable matrix Σ. Taking $\psi_i(\sigma) = \psi_i^{(\mathbf{P}_\sigma \mathbf{A}_{\overline{C}_n} \mathbf{P}_\sigma^T)_{U_i}}(\sigma)$ satisfies the desired properties for $\overline{C_n} \cong_{\mathcal{U}_n^t, S_n} \overline{C_{n_1} \oplus C_{n_2}}$. Notice that this implies that $C_n \cong_{\mathcal{U}_n^t, S_n} C_{n_1} \oplus C_{n_2}$, since C_n and $C_{n_1} \oplus C_{n_2}$ are both obtained from obtained $\overline{C_n}$ and $\overline{C_{n_1} \oplus C_{n_2}}$ by changing every entry with the value -1 to 0.

□

Proposition 9.5.8. *Fix any $t > 1$, and set either $d_0 = 4t$, and $d = 2d_0 + 1$. Set $\mathbf{B} = 1/2(\mathbf{A}_{C_d} + \lambda \mathbb{I}_d)$ and $\mathbf{D} = 1/2(\mathbf{A}_{C_{d_0} \oplus C_{d_0+1}} + \lambda \mathbb{I}_d)$, where $\lambda = -2\cos(\frac{2\pi d_0}{2d_0+1})$. Then we have that $\mathbf{B}$ is PSD, $\lambda_{\min}(\mathbf{D}) < -\delta$ where $\delta = \Theta(1/d^2)$, $\|\mathbf{B}\|_\infty, \|\mathbf{D}\|_\infty \leq 1$, and $\mathbf{B} \cong_{\mathcal{U}_d^t, S_d} \mathbf{D}$.*

Proof. By Proposition 9.5.7, we know $C_{2d_0+1} \cong_{\mathcal{U}^t_{2d_0+1}, S_{2d_0+1}} C_{d_0} \oplus C_{d_0+1}$, so to show subgraph equivalence suffices to show that adding $\lambda \mathbb{I}_{2d_0+1}$ to both C_{2d_0+1} and $C_{d_0} \oplus C_{d_0+1}$ does not effect the fact that they are $\mathcal{U}^t_{d_0}, S_{d_0}$ subgraph-equivalent. But note that this fact is clear, since we have only changed the diagonal which is still equal to λ everywhere for both $\mathbf{B}, \mathbf{D}$. Namely, for any $\sigma, \pi \in S_{2d_0+1}$ and $i \in [2d_0+1]$ we have $\left(\mathbf{P}_\sigma \mathbf{B} \mathbf{P}_\sigma^T\right)_{(i,i)} = \left(\mathbf{P}_\pi \mathbf{D} \mathbf{P}_\pi^T\right)_{(i,i)} = \lambda$, thus the subgraph equivalence between C_{2d_0+1} and $C_{d_0} \oplus C_{d_0+1}$ still holds using the same functions ψ_i as required for $C_{2d_0+1} \cong_{\mathcal{U}^t_{2d_0+1}, S_{2d_0+1}} C_{d_0} \oplus C_{d_0+1}$. Note that the L_∞ bound on the entries follows from the fact that adjacency matrices are bounded by 1 and zero on the diagonal, $\lambda \leq 2$, and we scale each matrix down by $1/2$. Next, by Fact 9.5.6, we know that $\mathbf{B}$ is PSD and $\lambda_{\min}(\mathbf{D}) = -\Theta(\frac{1}{d^2})$, which holds still after scaling by $1/2$, and completes the proof. □

We now state our main theorem, which is direct result of instantiating the general lower bound of Lemma 9.5.5 with the matrices as described above in Proposition 9.5.8.

Theorem 123. *Any non-adaptive sampling algorithm which solves with probability at least $2/3$ the PSD testing problem with ϵ-ℓ_2^2 gap must query at least $\widetilde{\Omega}(\frac{1}{\epsilon^2})$ entries of the input matrix.*

Proof. Set $k = C\frac{1}{\epsilon \log^6(1/\epsilon)}$ for a small enough constant $C > 0$. Also set $t = \log k$, $d_0 = 4t$. $d = 2d_0 + 1$, and as before set $m = n/(dk)$. We first apply Lemma 9.5.5 with $\mathbf{Z} = \mathbf{1}^{m \times m}$, and the matrices $\mathbf{B} = 1/2(\mathbf{A}_{C_d} + \lambda \mathbb{I}_d)$ and $\mathbf{D} = 1/2(\mathbf{A}_{C_{d_0} \oplus C_{d_0+1}} + \lambda \mathbb{I}_d)$ from Proposition 9.5.8, where $\lambda = -2\cos(\frac{2\pi d_0}{2d_0+1})$. Then by Lemma 9.5.5, using that $\mathbf{B} \cong_{\mathcal{U}^t_{2d_0+1}, S_{2d_0+1}} \mathbf{D}$ via Proposition 9.5.8, it follows that any non-adaptive sampling algorithm that distinguishes $\mathcal{D}_1$ from $\mathcal{D}_2$ requires $\Omega(k^2)$ samples.

We now demonstrate every instance of $\mathcal{D}_1$ and $\mathcal{D}_2$ satisfy the desired ℓ_2^2-gap as defined in Problem 2. First, since the eigenvalues of the Kronecker product $\mathbf{Y} \otimes \mathbf{Z}$ of any matrices $\mathbf{Y}, \mathbf{Z}$ are all pairwise eigenvalues of the matrices $\mathbf{Y}, \mathbf{Z}$, it follows that $\widetilde{\mathbf{B}}$ is PSD as $\mathbf{B}$ is PSD by Proposition 9.5.8 and and $\mathbf{1}^{m \times m} = \mathbf{1}^m (\mathbf{1}^m)^T$ is PSD. By the same fact and Proposition 9.5.8, since $\lambda_1(\mathbf{1}^{m \times m}) = m$, we have that $\lambda_{\min}(\widetilde{\mathbf{D}}) = -\Theta(m/(d^2)) = -\Theta(\frac{n}{d^3 k})$. Now note that if $\mathbf{A}_1 \sim \mathcal{D}_1$, then $\mathbf{A}_1$ is a block-diagonal matrix where each block is PSD, thus $\mathbf{A}_1$ is PSD. Note also that if $\mathbf{A}_2 \sim \mathcal{D}_2$, then $\mathbf{A}_2$ is a block-diagonal matrix where each block is has an eigenvalue smaller than $-C'\frac{n}{d^3 k}$ for some constant $C' > 0$. Since the eigenvalues of a block diagonal matrix are the union of the eigenvalues of the blocks, it follows that

$$\begin{aligned}\sum_{i:\lambda_i(\mathbf{A}_2)<0}(\lambda_i(\mathbf{A}_2))^2 &= \sum_{i=1}^{k}\lambda_{\min}(\widetilde{\mathbf{D}})^2 \\ &\geq k(C'\frac{n}{d^3k})^2 \\ &= \left(\frac{(C')^2\log^6(1/\epsilon)}{C(8\log(\frac{C}{\epsilon\log^6(1/\epsilon)})+1)^6}\right)\cdot\epsilon n^2 \\ &\geq \epsilon n^2\end{aligned} \tag{9.32}$$

Where the last inequality follows from setting the constant $C = \frac{(C')^2}{100^6}$ so that

$$\begin{aligned}\frac{(C')^2\log^6(1/\epsilon)}{C\left(8\log(\frac{C}{\epsilon\log^6(1/\epsilon)})+1\right)^6} &= \frac{100^6\log^6(1/\epsilon)}{\left(8\log(\frac{(C')^2}{100^6\epsilon\log^6(1/\epsilon)})+1\right)^6} \\ &\geq \frac{100^6\log^6(1/\epsilon)}{\left(16\log(\frac{1}{\epsilon})\right)^6} \\ &> 1\end{aligned} \tag{9.33}$$

and using that the first inequality above holds whenever $\left(\frac{1}{\epsilon}\right)^{16} \leq \left(\frac{(C')^2}{3\cdot 100^6\epsilon\log^6(1/\epsilon)}\right)^8$, which is true so long as $\epsilon < C_0$ for some constant C_0. Note that if $\epsilon > C_0$, then a lower bound of $\Omega(1) = \Omega(1/\epsilon^2)$ follows from the one heavy eigenvalue ℓ_∞ gap lower bound. Thus $\mathbf{A}_1, \mathbf{A}_2$ satisfies the ϵ-L_2 gap property as needed, which completes the proof.

□

$C_{n_1+n_2}$ is Subgraph Equivalent to $C_{n_1} \oplus C_{n_2}$

In this section, we demonstrate the subgraph equivalence of the the cycle $C_{n_1+n_2}$ and union of cycles $C_{n_1} \oplus C_{n_2}$. In order to refer to edges which are not in the cycles $C_{n_1+n_2}$ and $C_{n_1} \oplus C_{n_2}$, it will actually be convenient to show that $\overline{C_{n_1+n_2}}$ is subgraph equivelant to $\overline{C_{n_1} \oplus C_{n_2}}$, where recall that $\overline{G}$ for a simple graph G is the result of adding negative edges to G for each edge $e = (u, v) \notin E(G)$. Equivalently, the adjacency matrix of $\overline{G}$ is the result of replacing the 0's on the off-diagonal of A_G with -1's. Notice that, by the definition of subgraph equivalence, it does not matter whether these values are set to 0 or to -1.

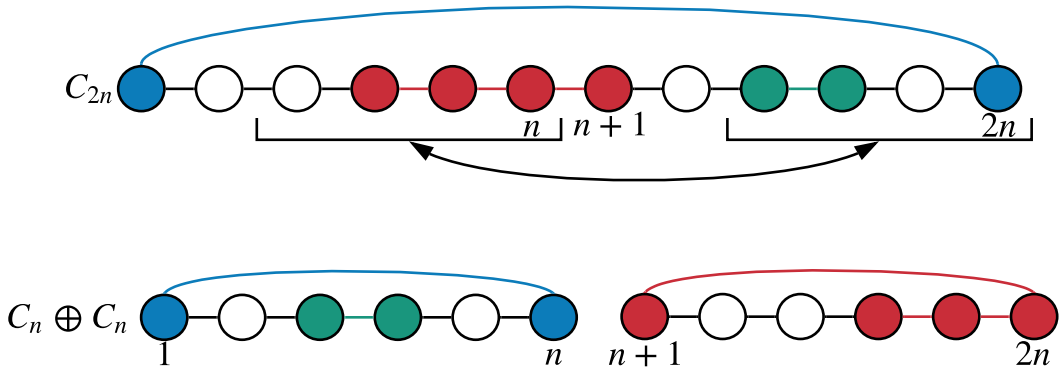


Figure 9.2: An illustration of the bijection in Lemma 9.5.9, when H only contains positive edges. The three colored paths represent the graph H, which must be mapped from C_{2n} to $C_n \oplus C_n$. Since the paths intersect the edges $(2n, 1)$ and $(n, n+1)$ to be cut, we must first swap the last four vertices $\{n-4, \ldots, n\}$ and $\{2n-4, \ldots, 2n\}$ of C_{2n} before the two splitting points $n, 2n$, and then cut the cycle. Note that four is the smallest number of vertices which can be swapped, without swapping in the middle of a path of H.

Overview of the bijection. We now intuitively describe the bijection of Lemma 9.5.9, which demonstrates that for any singed graph Σ such that any set of pairwise vertex disjoint edges $\{e_1, \ldots, e_k\}$ (i.e. any matching) in Σ has size at most $k \leq \min\{n_1, n_2\}/4$, the number of subgraphs of $\overline{C_{n_1+n_2}}$ isomorphic to Σ is the same as the number of subgraphs of $\overline{C_{n_1} \oplus C_{n_2}}$ isomorphic to Σ. So let H be any subgraph of $\overline{C_{n_1+n_2}}$ that is isomorphic to Σ. For simplicity, let $n_1 = n_2$, and suppose H contains only positive edges, so that H is actually a subgraph of the unsigned cycle C_{2n}. Since Σ has at most $n/4$ edges, $\Sigma \cong H$ must be a collection of disjoint paths. So the problem can be described as an arrangement problem: for each arrangement H of Σ in C_{2n}, map it to a unique arrangement H' of Σ in $C_n \oplus C_n$.

We would like to construct such a mapping by "splitting" the big cycle C_{2n} into two smaller cycles, see Figure 9.2 for an example. Specifically, we could split the cycle C_{2n} down the middle, cutting the edges $(n, n+1)$, and $(n, 1)$, and instead connecting the first vertex to the n-th and the $n+1$-st to the $2n$-th. Now if H does not contain either of the cut edges, then the resulting collection of paths will be an isomorphic copy of H living inside of $C_n \oplus C_n$. However, if H does contain such an edge, we cannot cut the cycle here, as the resulting paths inside of $C_n \oplus C_n$ would not be isomorphic. For example see Figure 9.2, where if we just cut the edge between $(n, n+1)$ and rerouted it to $(n, 1)$, then the red cycle with 4 vertices would be disconnected into a cycle of length three, and an isolated vertex. To handle this, before cutting and rerouting the edges $(n, n+1)$ and $(2n, 1)$, we first *swap* the last i vertices before the cutting points, for some

i. Namely, we swap the vertices $(n-i, n-i+1, \ldots, n)$ with $(2n-i, 2n-i+1, \ldots, 2n)$ and *then* split the graph at the edges $(n, n+1)$ and $(2n, 1)$. For the resulting graphs to be isomorphic, we cannot swap in the middle of a path, thus the value i is chosen as the *smallest* $i \geq 0$ such that the edges $(n-i-1, n-i)$ and $(2n-i-1, 2n-i)$ do not exist in any path of H. Moreover, such an i must exist, so long as H has fewer than $\min\{n_1, n_2\}$ edges (the stronger bound of $\min\{n_1, n_2\}/4$ is only needed for the more general case, where negative edges are included).

One can show that this mapping is actually an involution; namely, given the collection of paths H' in $C_n \oplus C_n$ which are obtained from applying the function on H, one can similarly find the smallest $i \geq 0$ such that the edges $(n-i-1, n-i)$ and $(2n-i-1, 2n-i)$ are not in H', which must in fact be the same value of i used when mapping H! Then, by swapping the last i vertices before n and $2n$, and then reconnecting $C_n \oplus C_n$ into a single cycle, one obtains the original graph H. From this, demonstrating bijectivity becomes relatively straightforward. Extending this to the case where H is allowed to contain negative edges of $\overline{C_{2n}}$ follows similar steps, albiet with a stronger condition on the choice of i. The full proof is now presented below.

Lemma 9.5.9. *Fix any $n = n_1 + n_2$. Fix any simple graph $|\Sigma|$, such that any set of vertex disjoint edges $\{e_1, \ldots, e_k\}$ in $|\Sigma|$ has size at most $k \leq \min\{n_1, n_2\}/4$, and let $\Sigma = (|\Sigma|, \sigma)$ be any signing of $|\Sigma|$. Let $\mathcal{F}_\Sigma(\overline{C_n})$ denote the set of subgraphs of $\overline{C_n}$ isomorphic to $|\Sigma|$, and similarly define $\mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$. Then we have*

$$\left|\mathcal{F}_\Sigma(\overline{C_n})\right| = \left|\mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})\right|$$

Proof. Order the vertices's of the cycle $C_n = \{1, 2, \ldots, n\}$, which we will describe as the same vertex set for $C_{n_1} \oplus C_{n_2}$, where $\{1, \ldots, n_1\}$ are the vertices of the first cycle C_{n_1} and $\{n_1+1, \ldots, n\}$ are the vertices of C_{n_2}. We derive a bijection $\varphi : \mathcal{F}_\Sigma(\overline{C_n}) \to \mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$. We describe a point $\mathbf{X} \in \mathcal{F}_H(\overline{C_n}) \cup \mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$ by its (signed) adjacency matrix $\mathbf{X} \in \{-1, 0, 1\}^{n \times n}$. Namely, $\mathbf{X} \in \{-1, 0, 1\}^{n \times n}$ is any matrix obtained by setting a subset of the entries of $\mathbf{A}_{\overline{C_{n_1+n_2}}}$ or $\mathbf{A}_{\overline{C_{n_1} \oplus C_{n_2}}}$ equal to 0, such that the signed graph represented by $\mathbf{X}$ is isomorphic to Σ. In this following, we will always modularly interpret the vertex $v_{n+i} = v_i$ for $i \geq 1$.

Thus, we can now think of φ as being defined on the subset of the matrices $\{-1, 0, 1\}^{n \times n}$ given by the adjacency matrices of signed graphs in $\mathcal{F}_\Sigma(\overline{C_n})$. In fact, it will useful to define φ on a larger domain. Let $\mathcal{D} \subset \{-1, 0, 1\}^{n \times n}$ be the set of all adjacency matrices for signed graphs G with the property that any set of vertex disjoint edges $\{e_1, \ldots, e_k\}$ in G size at most $k \leq \min\{n_1, n_2\}/4$. Notice that $\mathcal{D}$ contains both $\mathcal{F}_\Sigma(\overline{C_n})$ and $\mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$. For a given

$\mathbf{X} \in \mathcal{D}$, we will define $\varphi(\mathbf{X}) = \mathbf{P}_{\sigma_\mathbf{X}}\mathbf{X}\mathbf{P}_{\sigma_\mathbf{X}}^T$ for some permutation $\sigma_\mathbf{X}$. Since the graph of $\mathbf{P}_{\sigma_\mathbf{X}}\mathbf{X}\mathbf{P}_{\sigma_\mathbf{X}}^T$ is by definition isomorphic to $\mathbf{X}$, it follows that $\mathbf{P}_{\sigma_\mathbf{X}}\mathbf{X}\mathbf{P}_{\sigma_\mathbf{X}}^T \in \mathcal{D}$, thus φ maps $\mathcal{D}$ into $\mathcal{D}$. So in order to define the mapping $\varphi(\mathbf{X})$, it suffices to define a function $\phi : \mathcal{D} \to S_n$ mapping into the symmetric group so that $\varphi(\mathbf{X}) = \mathbf{P}_{\phi(\mathbf{X})}\mathbf{X}\mathbf{P}_{\phi(\mathbf{X})}^T$.

For $i = 0, 1, 2, \ldots, \min_{n_1,n_2} -1$, define the permutation $\sigma_i \in S_n$ as follows. For $j \in \{0, 1, \ldots, n_1 - i\} \cup \{n_1 + 1, \ldots, n - i\}$, we set $\sigma_i(j) = j$. If $i > 0$, then for each $0 \leq j < i$, we set $\sigma_i(n_1 - j) = n - j$ and $\sigma_i(n - j) = n_1 - j$. In other words, the function σ_i swaps the last $\max\{0, i - 1\}$ vertices before the spliting points n_1, n of the cycle. Notice that σ_i is an involution, so $\sigma_i(\sigma_i) = \text{id}$ and $\sigma_i = \sigma_i^{-1}$.

We now define our bijection φ. For $\mathbf{X} \in \mathcal{D}$, let $i(\mathbf{X})$ be the smallest value of $i \geq 0$ such that $\mathbf{X}_{n_1-i,n_1-i+1} = \mathbf{X}_{n-i,n-i+1} = \mathbf{X}_{n-i,n_1-i+1} = \mathbf{X}_{n_1-i,n-i+1} = 0$. Equivalently, $i(\mathbf{X})$ is the smallest value of $i \geq 0$ such that none of the four edges of the cycle $c_i = (v_{n_1-i}, v_{n_1-i+1}, v_{n-i}, v_{n-i+1})$ exist in $\mathbf{X}$. We then define $\varphi(\mathbf{X}) = \sigma_{i(\mathbf{X})} = \sigma_\mathbf{X}$, so that $\varphi(\mathbf{X}) = \mathbf{P}_{\sigma_{i(\mathbf{X})}}\mathbf{X}\mathbf{P}_{\sigma_{i(\mathbf{X})}}^T$. Note that if the maximum number of vertex disjoint edges in $\mathbf{X}$ is at most $\min\{n_1, n_2\}/4$, then $i(\mathbf{X})$ must always exist and is at most $\min\{n_1, n_2\}/2 + 1$. This can be seen by the fact that for each i such that $i(\mathbf{X}) > i + 1$, there must be at least one edge with endpoints in the set $\{v_{n_1-i}, v_{n_1-i+1}, v_{n-i}, v_{n-i+1}\}$, thus for each $i \geq 0$ with $i < i(\mathbf{X})$ we can assign an edge e_i, such that $e_0, e_2, e_4, \ldots, e_{i(\mathbf{X})-1}$ are vertex disjoint.

We must first argue that if $\mathbf{X} \in \mathcal{F}_\Sigma(\overline{C_n})$, then $\varphi(\mathbf{X}) \in \mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$, namely that the function maps into the desired co-domain. To do this, we must show that for every (i, j) with $(\mathbf{P}_{\sigma_\mathbf{X}}\mathbf{X}\mathbf{P}_{\sigma_\mathbf{X}}^T)_{i,j} \neq 0$, we have $(\mathbf{P}_{\sigma_\mathbf{X}}\mathbf{X}\mathbf{P}_{\sigma_\mathbf{X}}^T)_{i,j} = (\mathbf{A}_{\overline{C_{n_1} \oplus C_{n_2}}})_{i,j}$. This is equivalent to showing that for any signed edge $e = (v_i, v_j) \in \mathbf{X}$, v_i, v_j are connected in C_n if and only if $v_{\sigma_\mathbf{X}(i)}, v_{\sigma_\mathbf{X}(j)}$ are connected in $C_{n_1} \oplus C_{n_2}$. In the proof of this fact, we will only use that $\mathbf{X} \in \mathcal{D}$.

So suppose v_i, v_j were connected in C_n, and wlog $j > i$. First suppose that $i \notin \{n, n_1\}$. Then we have $j = i + 1$. Since $e = (v_i, v_j) \in \mathbf{X}$ is an edge of the subgraph, we know $i \notin \{n - i(\mathbf{X}), n_1 - i(\mathbf{X})\}$ by construction of $i(\mathbf{X})$. Thus $(v_{\sigma_\mathbf{X}(i)}, v_{\sigma_\mathbf{X}(j)}) = (v_{i'}, v_{i'+1})$ for some $i' \notin \{n_1, n\}$, which is always an edge of $C_{n_1} \oplus C_{n_2}$. If $i = n$, then $j = 1$, and we have $i(\mathbf{X}) > 0$, so $\sigma(i) = n_1$ and $\sigma(j) = 1$, and (v_{n_1}, v_1) is an edge of $C_{n_1} \oplus C_{n_2}$. Similarly, if $i = n_1$, then $j = n_1 + 1$, and since again necessarily $i(\mathbf{X}) > 0$ we have $\sigma(i) = n, \sigma(j) = n_1 + 1$, and (v_n, v_{n_1+1}) is an edge of $C_{n_1} \oplus C_{n_2}$. We now consider the case where $(v_i, v_j) \in \mathbf{X}$ is not an edge in C_n. Suppose for the sake of contradiction that $(v_{\sigma_\mathbf{X}(i)}, v_{\sigma_\mathbf{X}(j)})$ is an edge in $C_{n_1} \oplus C_{n_2}$. WLOG, i, j are in the first cycle C_{n_1}. We can write $\sigma_\mathbf{X}(i) = i', \sigma_\mathbf{X}(j) = i' + 1$ for some $i' \in \{1, 2, \ldots, n_1\}$, where $i' + 1$ is interpreted as 1 if $i' = n_1$. If $i' \leq i(\mathbf{X}) - 1$, then both $i' = i$ and $i' + 1 = i + 1 = j$, but (v_i, v_{i+1}) is also connected in C_n. If $i' \geq i(\mathbf{X}) + 1$, then $i' = i + n_2$

and $i' + 1 = i + n_2 + 1$ (where $i + n_2 + 1$ is interpreted modularly as 1 if $i = n_1$), and again v_{i+n_2} and v_{i+n_2+1} are connected in C_n. Finally, if $i' = i(\mathbf{X})$, then $i = i'$ and $j = i + n_2 + 1$, but then we cannot have $(v_i, v_j) \in \mathbf{X}$ by construction of $i(\mathbf{X})$, which completes the of the claim that φ maps $\mathcal{F}_\Sigma(\overline{C_n})$ into $\mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$.

We now show that φ is injective. To do this, we show that $\varphi(\varphi(\mathbf{X})) = \mathbf{X}$ for any $\mathbf{X} \in \mathcal{D}$ – namely that φ is an involution on $\mathcal{D}$. This can be seen by showing that we always have $i(\mathbf{X}) = i(\varphi(\mathbf{X}))$. To see this, observe that $i(\mathbf{X})$ is defined as the first $i \geq 0$ such that none of the four edges of the cycle $c_i = (v_{n_1-i}, v_{n_1-i+1}, v_{n-i}, v_{n-i+1})$ exist in $\mathbf{X}$. Thus it suffices to show that for each $\min\{n_1, n_2\} - 1 > i \geq 0$, the number of edges in c_i is preserved after permuting the vertices by $\sigma_{i(\mathbf{X})}$. To see this, note that if $i(x) > i$, then

$$(\sigma_{i(\mathbf{X})}(v_{n_1-i}), \sigma_{i(\mathbf{X})}(v_{n_1-i+1}), \sigma_{i(\mathbf{X})}(v_{n-i}), \sigma_{i(\mathbf{X})}(v_{n-i+1})) = (v_{n-i}, v_{n-i+1}, v_{n_1-i}, v_{n_1-i+1})$$

which is the same cycle. If $i(\mathbf{X}) < i$, then $\sigma_{i(\mathbf{X})}$ does not move any of the vertices in c_i. Finally, if $i(\mathbf{X}) = i$, then

$$(\sigma_{i(\mathbf{X})}(v_{n_1-i}), \sigma_{i(\mathbf{X})}(v_{n_1-i+1}), \sigma_{i(\mathbf{X})}(v_{n-i}), \sigma_{i(\mathbf{X})}(v_{n-i+1})) = (v_{n_1-i}, v_{n-i+1}, v_{n-i}, v_{n_1-i+1})$$

which again is the same cycle c_i (just with the ordering of the vertices reversed). So $\varphi(\varphi(\mathbf{X})) = \mathbf{X}$ for any $\mathbf{X} \in \mathcal{D}$, so in particular $\varphi : \mathcal{F}_\Sigma(\overline{C_n}) \to \mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$ is injective.

To show surjectivity, it suffices to show that if $\mathbf{X} \in \mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$ then $\varphi(\mathbf{X}) \in \mathcal{F}_\Sigma(\overline{C_n})$. Namely, that φ can also be defined as a valid function $\varphi : \mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}}) \to \mathcal{F}_\Sigma(\overline{C_n})$. Again, this is equivalent to showing that for any signed edge $e = (v_i, v_j) \in \mathbf{X}$, v_i, v_j are connected in $C_{n_1} \oplus C_{n_2}$ if and only if $v_{\sigma_\mathbf{X}(i)}, v_{\sigma_\mathbf{X}(j)}$ are connected in C_n. Since $\sigma_\mathbf{X}$ is an involution, this is the same as asking that for any signed edge $e = (v_i, v_j) \in \mathbf{X}$, $v_{\sigma_\mathbf{X}(\sigma_\mathbf{X}(i))}, v_{\sigma_\mathbf{X}(\sigma_\mathbf{X}(j))}$ are connected in $C_{n_1} \oplus C_{n_2}$ if and only if $v_{\sigma_\mathbf{X}(i)}, v_{\sigma_\mathbf{X}(j)}$ are connected in C_n. Setting $i' = \sigma_\mathbf{X}(i), j' = \sigma_\mathbf{X}(j)$, this states that for all signed edges $(v_{i'}, v_{j'}) \in \mathbf{P}_{\sigma_\mathbf{X}}\mathbf{X}\mathbf{P}_{\sigma_\mathbf{X}} = Y \in \mathcal{D}$, we have that $v_{i'}, v_{j'}$ are connected in C_n if and only if $v_{\sigma_\mathbf{X}(i')}, v_{\sigma_\mathbf{X}(j')}$ are connected in $C_{n_1} \oplus C_{n_2}$. But as shown above, we have that $i(\mathbf{X}) = i(\varphi(\mathbf{X}))$, so $\sigma_\mathbf{X} = \sigma_Y$, and then this fact was already proven above for any $Y \in \mathcal{D}$, which completes the proof.

□

Now for any signed graph Σ on n vertices, let $\mathbf{A}_\Sigma$ be its adjacency matrix. Note that we can equivalently define via $\mathcal{F}_\Sigma(\overline{C_n}) = \{H \subseteq \overline{C_n}, \mid \mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{A}_H, \sigma \in S_n\}$. Here $H \subseteq \overline{C_n}$ means H is a subgraph of $\overline{C}_n$. On the other hand, we may be interested in the potentially much

larger set of all possible permutations σ such that $\mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{A}_H$ for some $H \subset \overline{C_n}$. So define $\mathcal{H}_\Sigma(\overline{C_n}) = \{\sigma \mid \mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{A}_H, H \subset \overline{C_n}, \sigma \in S_n\}$. It is not difficult to show that $|\mathcal{H}_\Sigma(\overline{C_n})| = |\text{Aut}(\Sigma)||\mathcal{F}_\Sigma(\overline{C_n})|$, where $\text{Aut}(\Sigma)$ is the set of (signed) graph automorphisms of Σ.

Fact 9.5.10. *We have $|\mathcal{H}_\Sigma(\overline{C_n})| = |Aut(\Sigma)||\mathcal{F}_\Sigma(\overline{C_n})|$.*

Proof. Fix any $H \subset \overline{C_n}$ such that $\mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{A}_H$ for some $\sigma \in S_n$. We show that there are exactly $|\text{Aut}(\Sigma)|$ elements $\sigma' \in S_n$ such that $\mathbf{P}_{\sigma'} \mathbf{A}_\Sigma \mathbf{P}_{\sigma'}^T = \mathbf{A}_H$. By definition, $\text{Aut}(\Sigma)$ is the set of permutations $\pi \in S_n$ with $\mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{A}_\Sigma$. For every $\pi \in \text{Aut}(\Sigma)$, we have $\mathbf{P}_\sigma \mathbf{P}_\pi \mathbf{A}_\Sigma \mathbf{P}_\pi \mathbf{P}_\sigma^T = \mathbf{P}_{\sigma\circ\pi} \mathbf{A}_\Sigma \mathbf{P}_{\sigma\circ\pi}^T = \mathbf{A}_H$, and moreover the set of elements $|\{\sigma \circ \pi \mid \pi \in \text{Aut}(\Sigma)\}| = |\text{Aut}(\Sigma)|$ since S_n is a group. Now suppose we have some $\lambda \in S_n$ such that $\mathbf{P}_\lambda \mathbf{A}_\Sigma \mathbf{P}_\lambda^T = \mathbf{A}_H$ and $\lambda \notin \{\sigma \circ \pi \mid \pi \in \text{Aut}(\Sigma)\}$. Then $\mathbf{P}_\sigma \mathbf{A}_\Sigma \mathbf{P}_\sigma^T = \mathbf{P}_\lambda \mathbf{A}_\Sigma \mathbf{P}_\lambda^T$, so $\mathbf{P}_{\sigma^{-1}\circ\lambda} \mathbf{A}_\Sigma \mathbf{P}_{\sigma^{-1}\circ\lambda}^T = \mathbf{A}_\Sigma$, which by definition implies that $\sigma^{-1} \circ \lambda = x$ for some $x \in \text{Aut}(\Sigma)$. Thus $\lambda = \sigma \circ x \in \{\sigma \circ \pi \mid \pi \in \text{Aut}(\Sigma)\}$, which is a contradiction. □

Corollary 9.5.11. *Fix any $n = n_1 + n_2$. Fix any simple graph $|\Sigma|$, such that any set of vertex disjoint edges $\{e_1, \ldots, e_k\}$ in $|\Sigma|$ has size at most $k \leq \min\{n_1, n_2\}/4$, and let $\Sigma = (|\Sigma|, \sigma)$ be any signing of $|\Sigma|$. Let $\mathcal{F}_\Sigma(\overline{C_n})$ denote the set of subgraphs of C_n isomorphic to $|\Sigma|$, and similarly define $\mathcal{F}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})$. Then we have*

$$\left|\mathcal{H}_\Sigma(\overline{C_n})\right| = \left|\mathcal{H}_\Sigma(\overline{C_{n_1} \oplus C_{n_2}})\right|$$

9.5.3 Lower Bounds for Schatten, Ky-Fan, and Tail Error Testing

In this section, we demonstrate how our construction of subgraph equivalent matrices with gaps in their spectrum result in lower bounds for a number of other spectral testing problems via Lemma 9.5.5. We begin by proving a lower bound for testing Schatten norms. To do this, we must first demonstrate that there is a gap in the Schatten 1 norm between a cycle and the union of two disjoint cycles.

Fact 9.5.12 (Theorem 1 of [Kna09])**.** *Fix any $a, b, n \in \mathbb{R}$ with $\sin(b/2) \neq 0$. Then we have*

$$\sum_{k=0}^{n-1} \cos(a + kb) = \frac{\sin(\frac{nb}{2})}{\sin(\frac{b}{2})} \cos\left(a + \frac{(n-1)b}{2}\right)$$

Proposition 9.5.13. *Fix any $d \geq 6$ be any integer divisible by 4. Then*

$$\|C_d\|_{\mathcal{S}_1} = 4 \cdot \frac{\cos(\pi/d)}{\sin(\pi/d)}$$

Proof. By [Chu96], for any $d \geq 3$ the eigenvalues of C_d are given by $2 \cdot \cos(\frac{2\pi j}{d})$ for $j = 0, 1, \ldots, d-1$. Let $a_1 = \lfloor d/4 \rfloor, a_2 = \lfloor 3d/4 \rfloor, a_3 = d - a_2 - 1$.

$$\begin{aligned}
\|C_d\|_1 &= 2 \sum_{j=0}^{d-1} \left| \cos\left(\frac{2\pi j}{d}\right) \right| \\
&= 2 \left(\sum_{j=0}^{a_1} \cos\left(\frac{2\pi j}{d}\right) - \sum_{j=a_1+1}^{a_2} \cos\left(\frac{2\pi j}{d}\right) + \sum_{j=a_2+1}^{d-1} \cos\left(\frac{2\pi j}{d}\right) \right) \\
&= 2 \left(\sum_{j=-a_3}^{a_1} \cos\left(\frac{2\pi j}{d}\right) - \sum_{j=a_1+1}^{a_2} \cos\left(\frac{2\pi j}{d}\right) \right)
\end{aligned} \tag{9.34}$$

We analyze each term in the above via Fact 9.5.12. Firstly:

$$\begin{aligned}
\sum_{j=-a_3}^{a_1} \cos\left(\frac{2\pi j}{d}\right) &= \sum_{j=0}^{a_1+a_3} \cos\left(\frac{2\pi j}{d} - \frac{2\pi a_3}{d}\right) \\
&= \frac{\sin((a_1+a_3+1)\pi/d)}{\sin(\pi/d)} \cos\left(\frac{(a_1+a_3)\pi}{d} - \frac{2\pi a_3}{d}\right)
\end{aligned} \tag{9.35}$$

Note that if d is divisible by 4, the above becomes $2\cos(\pi/d)/\sin(\pi/d)$. Next, for the second term, we have

$$\begin{aligned}
\sum_{j=a_1+1}^{a_2} \cos\left(\frac{2\pi j}{d}\right) &= \sum_{j=0}^{a_2-a_1-1} \cos\left(\frac{2\pi j}{d} - \frac{2\pi(a_1+1)}{d}\right) \\
&= \frac{\sin((a_2-a_1)\pi/d)}{\sin(\pi/d)} \cos\left(\frac{(a_2-a_1-1)\pi}{d} - \frac{2\pi(a_1+1)}{d}\right)
\end{aligned} \tag{9.36}$$

Again, note that if d is divisible by 4, the above becomes $2\cos(\pi/d)/\sin(\pi/d)$. Putting these two equations together, we have that

$$\|C_d\|_1 = 4 \cdot \frac{\cos(\pi/d)}{\sin(\pi/d)}$$

□

Proposition 9.5.14. *Fix any d larger than some constant. Then we have*

$$|\|\mathbf{C}_{8d}\|_{\mathcal{S}_1} - \|\mathbf{C}_{4d} \oplus \mathbf{C}_{4d}\|_{\mathcal{S}_1}| \gtrsim \frac{1}{d^3}$$

Proof. By the prior Lemma, we have $\|\mathbf{C}_d\|_{\mathcal{S}_1} = 4\cot(\pi/d)$ for any d divisible by 4. Thus using the Taylor expansion of cotangent, we have

$$\|\mathbf{C}_{8d}\|_{\mathcal{S}_1} = 4\left(\frac{8d}{\pi} + \frac{\pi}{24d} + \frac{\pi^3}{45 \cdot 512 \cdot d^3} + O(1/d^5)\right) \tag{9.37}$$

and

$$\begin{aligned} \|\|\mathbf{C}_{4d} \oplus \mathbf{C}_{4d}\|_{\mathcal{S}_1} &= 2\|\mathbf{C}_{4d}\|_{\mathcal{S}_1} \\ &= 4\left(\frac{8d}{\pi} + \frac{\pi}{24d} + \frac{\pi^3}{45 \cdot 128 \cdot d^3} + O(1/d^5)\right) \end{aligned} \tag{9.38}$$

Thus

$$|\|\mathbf{C}_{8d}\|_{\mathcal{S}_1} - \|\mathbf{C}_{4d} \oplus \mathbf{C}_{4d}\|_{\mathcal{S}_1}| \gtrsim \frac{1}{d^3} \tag{9.39}$$

□

Theorem 124. *Fix any $\frac{1}{\sqrt{n}} \leq \epsilon \leq 1$. Then given $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $\|\mathbf{A}\|_\infty \leq 1$, any non-adaptive sampling algorithm which distinguishes between the cases*

1. $\|\mathbf{A}\|_{\mathcal{S}_1} > \epsilon_0 n^{1.5}$
2. $\|\mathbf{A}\|_{\mathcal{S}_1} < \epsilon_0 n^{1.5} - \epsilon n^{1.5}$

with probability at least $3/4$, where $\epsilon_0 = \widetilde{\Theta}(\epsilon)$, must query at least $\tilde{\Omega}(1/\epsilon^4)$ entries of $\mathbf{A}$.

Proof. We use the hard instance $\mathcal{D}_1, \mathcal{D}_2$ as earlier. Set $k = C\frac{1}{\epsilon^2 \log^9(1/\epsilon)}$, $t = \log k$, and $d = 4k$, and $m = n/(dk)$. We instantiate the matrices $(\mathbf{B}, \mathbf{D}, \mathbf{Z})$ in the hard instance via $\mathbf{B} = C_{2d}, \mathbf{D} = \mathbb{C}_d \oplus C_d$, and let $\mathbf{Z} = \delta_{i,j}$ for $i \leq j$, where $\delta_{i,j} \in \{-1, 1\}$ are i.i.d. Bernoulli random variables, so that $\mathbf{Z} \in \mathbb{R}^{m \times m}$ is a symmetric random Bernoulli matrix. Using the fact that $\|\mathbf{Z}\|_2 \leq O(\sqrt{n})$ with high probability [Ver10], along with the fact that $\|\mathbf{Z}\|_F^2 = n^2$ deterministically, we have

that $\|\mathbf{Z}\|_{\mathcal{S}_1} > C_1 m^{1.5}$ with non-zero probability for some constant $C_1 > 0$, as the former two facts imply that $\mathbf{Z}$ has $\Omega(n)$ eigenvalues with magnitude $\Theta(\sqrt{n})$. Thus, we can deterministically fix $\mathbf{Z}$ to be such a matrix with $\{1, -1\}$ entries such that $\|\mathbf{Z}\|_{\mathcal{S}_1} \geq C_1 m^{1.5}$. Given this, we have $\|\widetilde{\mathbf{B}}\|_{\mathcal{S}_1} = \|\mathbf{B} \otimes \mathbf{Z}\|_{\mathcal{S}_1} = \|\mathbf{B}\|_{\mathcal{S}_1} \cdot \|\mathbf{Z}\|_{\mathcal{S}_1}$, and so by Proposition 9.5.14, we have

$$\left| \|\widetilde{\mathbf{B}}\|_{\mathcal{S}_1} - \|\widetilde{\mathbf{D}}\|_{\mathcal{S}_1} \right| \geq C_0 \frac{m^{1.5}}{d^3}$$

for some absolute constant $C_0 \geq 0$. Note also that we have $\|\mathbf{B}\|_{\mathcal{S}_1} > \Omega(d)$, where we use the fact that a constant fraction of the eigenvalues $2 \cdot \cos(\frac{2\pi j}{d})$ for $j = 0, 1, \ldots, d-1$ of $\mathbf{B}$ are $\Omega(1)$. Thus we have $\|\widetilde{\mathbf{B}}\|_{\mathcal{S}_1} = dm^{1.5}$.

Now by Proposition 9.5.7, we obtain that $\mathbf{B} \cong_{\mathcal{U}_{2d}^t, S_{2d}} \mathbf{D}$, and thus $\widetilde{\mathbf{B}} = \mathbf{B} \otimes \mathbf{Z} \cong_{\mathcal{U}_{2d}^t, \Gamma_{2d,2dm}} \mathbf{D} \otimes \mathbf{Z} = \widetilde{\mathbf{D}}$ by Lemma 9.5.4. Thus by Lemma 9.5.5, we have that distinguishing $\mathcal{D}_1$ from $\mathcal{D}_1$ requires $\Omega(k^2) = \tilde{O}(1/\epsilon^4)$ samples for any non-adaptive algorithm. It suffices then to show that if $\mathbf{A}_1 \sim \mathcal{D}_1$ and $\mathbf{A}_2 \sim \mathcal{D}_2$, then we have the desired gap in Schatten norms. We have

$$\begin{aligned} |\|\mathbf{A}_1\|_{\mathcal{S}_1} - \|\mathbf{A}_2\|_{\mathcal{S}_1}| &\geq \sum_{i=1}^{k} C_0 \frac{m^{1.5}}{d^3} \\ &\geq C_0 \frac{n^{1.5}}{d^{4.5} k^{1/2}} \\ &\geq \epsilon n^{1.5} \end{aligned} \tag{9.40}$$

Where the last inequality follows setting C large enough, and assuming that $1/\epsilon$ is larger than some constant as in Theorem 123. Again, if $1/\epsilon$ is not larger than some constant, a $\Omega(1)$ lower bound always applies, since an algorithm must read at least one entry of the matrix to have any advantage. Now note that we also have $\|\mathbf{A}_1\|_{\mathcal{S}_1} = k\|\widetilde{\mathbf{B}}\|_{\mathcal{S}_1} = kdm^{1.5} = n^{1.5}/\sqrt{dk} = \widetilde{\Theta}(\epsilon n^{1.5})$ as desired. To complete the proof, we can scale down all the entries of the input matrix by $1/2$, which results in the required bounded entry property, and only changes the gap by a constant factor. □

We now present our lower bound for testing Ky-Fan norms. Recall that for a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $1 \leq s \geq n$, the Ky-Fan s norm is defined as $\|\mathbf{A}\|_{KF(s)} = \sum_{i=1}^{k} \sigma_i(\mathbf{A})$, where $\sigma_i(\mathbf{A})$ is the i-th singular value of $\mathbf{A}$.

Theorem 125. *Fix any $1 \leq s \leq n/(\operatorname{poly}\log n)$. Then there exists a fixed constant $c > 0$ such that given $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $\|\mathbf{A}\|_\infty \leq 1$, any non-adaptive sampling algorithm which distinguishes*

between the cases

1. $\|\mathbf{A}\|_{KF(s)} > \frac{c}{\log(s)}n$
2. $\|\mathbf{A}\|_{KF(s)} < (1-\epsilon_0)\frac{c}{\log(s)}n$

with probability at least $3/4$*, where* $\epsilon_0 = \Theta(1/\log^2(s))$*, must query at least* $\tilde{\Omega}(s^2)$ *entries of* $\mathbf{A}$*.*[11]

Proof. The proof is nearly the same as the usage of the hard instance in Theorem 123. Set $k = s$, and let $d_0 = \Theta(\log s)$ and $d = 2d_0 + 1$. We apply Lemma 9.5.5 with the hard instance as instantiated with $\mathbf{Z} = \mathbf{1}^{m\times m}$, and the matrices $\mathbf{B} = 1/4(\mathbf{A}_{C_d} - 2\mathbb{I}_d)$ and $\mathbf{D} = 1/4(\mathbf{A}_{C_{d_0}\oplus C_{d_0+1}} - 2\mathbb{I}_d)$. Notice that since the eigenvalues of C_d are given by $2\cdot\cos(\frac{2\pi j}{d})$ for $j = 0, 1, \ldots, d-1$ [Chu96], we have $\lambda_{\min}(\mathbf{A}_{C_d}) = -2\cos(\frac{2\pi d_0}{2d_0+1}) = -2 + \Theta(1/\log^2(1/\epsilon))$, $\lambda_{\min}(\mathbf{A}_{C_{d_0}\oplus C_{d_0+1}}) = -2$, and $\lambda_{\max}(\mathbf{A}_{C_d}) = \lambda_{\max}(\mathbf{A}_{C_{d_0}\oplus C_{d_0+1}}) = 2$. Thus $\|\mathbf{D}\|_2 = 4$ and $\|\mathbf{B}\|_2 = 4 - \Theta(1/d^2)$, and moreover $\|\mathbf{D}\otimes\mathbf{Z}\|_2 = 4m$ $\|\mathbf{B}\otimes\mathbf{Z}\|_2 = 4m(1-\Theta(1/\log^2(1/\epsilon)))$. Thus if $\mathbf{A}_1 \sim \mathcal{D}_1$, we have $\|\mathbf{A}_1\|_{KF(s)} > \sum_{i=1}^{k} 4m = 4km$, and $\|\mathbf{A}_2\|_{KF(s)} < 4km(1-\Theta(1/\log^2(1/\epsilon)))$. The proof then follows from the $\Omega(k^2)$ lower bound for this hard instance via Lemma 9.5.5.

□

We now present our lower bound for testing the magnitude of the s-tail $\|\mathbf{A} - \mathbf{A}_s\|_F^2$, where $\mathbf{A}_s = \mathbf{U}\Sigma_s\mathbf{V}^T$ is the truncated SVD (the best rank-s approximation to $\mathbf{A}$). Note that $\|\mathbf{A} - \mathbf{A}_s\|_F^2 = \sum_{j>s}\sigma_j^2(\mathbf{A})$.

Theorem 126. *Fix any* $1 \le s \le n/(\operatorname{poly}\log n)$*. Then there exists a fixed constant* $c > 0$ *(independent of* ϵ*), such that given* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *with* $\|\mathbf{A}\|_\infty \le 1$*, any non-adaptive sampling algorithm which distinguishes between the cases*

1. $\|\mathbf{A} - \mathbf{A}_s\|_F^2 > \frac{c}{\log(s)}\cdot\frac{n^2}{s}$
2. $\|\mathbf{A} - \mathbf{A}_s\|_F^2 < (1-\epsilon_0)\cdot\frac{c}{\log(s)}\cdot\frac{n^2}{s}$

with probability at least $3/4$*, where* $\epsilon_0 = \widetilde{\Theta}(1)$*, must query at least* $\tilde{\Omega}(s^2)$ *entries of* $\mathbf{A}$*.*

Proof. We set $s = k$, and use the same hard instance as in Theorem 125 above. Note that if $\mathcal{D}_1, \mathcal{D}_2$ are defined as in Theorem 125, if $\mathbf{A}_1 \sim \mathcal{D}_1$, $\mathbf{A}_s \sim \mathcal{D}_s$, we have $\sum_{i=1}^{s}\lambda_i(\mathbf{A}_1) = s(4m)^2 = 16n^2/(sd^2)$ and $\sum_{i=1}^{s}\lambda_i(\mathbf{A}_2) = 16n^2/(sd^2)(1-\Theta(1/\log^2 s))$. Now note that $\|\mathbf{A}_1\|_F^2 = \|\mathbf{A}_2\|_F^2 = kdm^2 = n^2/(dk) = n^2/(ds)$, using that each of the single cycle and union of two smaller cycles has d edges, so the Frobenius norm of each block is dm^2 in both cases.

[11] $\widetilde{\Omega}$ hides $\log(s)$ factors here.

Using that $d = \Theta(\log s)$, we have that if $\|(\mathbf{A}_1) - (\mathbf{A}_1)s\|_F^2 > n^2/(ds) - 16n^2/(sd^2) = c\frac{n^2}{s\log(s)}$ for some constant $c > 0$, and $\|(\mathbf{A}_2) - (\mathbf{A}_2)s\|_F^2 > n^2/(ds) - 16n^2/(sd^2)(1 - \Theta(1/\log^2 s)) = c\frac{n^2}{s\log(s)} + \widetilde{\Theta}(\frac{n^2}{s})$, which completes the proof after applying Lemma 9.5.5. □

9.5.4 Lower Bound For Estimating Ky-Fan of $\mathbf{A}\mathbf{A}^T$ via Submatrices

In this section, we demonstrate a $\Omega(1/\epsilon^4)$ query lower bound for algorithms which estimate the quantity $\sum_{i=1}^k \sigma_i^2(\mathbf{A}) = \|\mathbf{A}\mathbf{A}^T\|_{KF(k)}$ for any $k \geq 1$ by querying a sub-matrix. The following lemma as a special case states that for $\epsilon = \Theta(1/\sqrt{n})$, additive ϵn^2 approximation of $\|\mathbf{A}\mathbf{A}^T\|_{KF(k)}$ requires one to read the entire matrix $\mathbf{A}$.

Lemma 9.5.15. *Fix any $1 \leq k \leq n$, and fix any $\frac{100}{\sqrt{n}} \leq \epsilon \leq 1/4$. Any algorithm that queries a submatrix $\mathbf{A}_{S\times T}$ of $\mathbf{A} \in \mathbb{R}^{n\times n}$ with $\|\mathbf{A}\|_\infty \leq 1$ and distinguishes with probability at least $4/5$ between the case that either:*

- $\sum_{i=1}^k \sigma_i^2(\mathbf{A}) > n^2/2 + \epsilon n^2$.
- $\sum_{i=1}^k \sigma_i^2(\mathbf{A}) \leq n^2/2$

must make $|S| \cdot |T| = \Omega(1/\epsilon^4)$ queries to the matrix $\mathbf{A}$.

Proof. We design two distributions $\mathcal{D}_1, \mathcal{D}_2$. If $\mathbf{A}_1 \sim \mathcal{D}_1$, we independently set each row of $\mathbf{A}_1$ equal to the all $1's$ vector with probability $p_1 = 1/2 + 2\epsilon$, and then return either $\mathbf{A} = \mathbf{A}_1$ or $\mathbf{A} = \mathbf{A}_1^T$ with equal probability. If $\mathbf{A}_2 \sim \mathcal{D}_2$, we independently set each row of $\mathbf{A}_2$ equal to the all $1's$ vector with probability $p_2 = 1/2 - 2\epsilon$, and then return either $\mathbf{A} = \mathbf{A}_2$ or $\mathbf{A} = \mathbf{A}_2^T$ with equal probability. Our hard instance then draws $\mathbf{A} \sim \frac{\mathcal{D}_1+\mathcal{D}_2}{2}$ from the mixture. First note that in both cases, we have $\|\mathbf{A}\|_2^2 = \|\mathbf{A}\|_F^2 = \sum_{i=1}^k \sigma_i^2(\mathbf{A})$, since the matrix is rank 1. Since $\frac{100}{\sqrt{n}} \geq \epsilon$, by Chernoff bounds, we have that if $\mathbf{A}_1 \sim \mathcal{D}_1$ then $\sum_{i=1}^k \sigma_i^2(\mathbf{A}) > n^2/2 + \epsilon n^2$ with probability at least $99/100$. Similarly, we have that if $\mathbf{A}_2 \sim \mathcal{D}_2$ then $\sum_{i=1}^k \sigma_i^2(\mathbf{A}) \leq n^2$ with probability at least $99/100$.

Now suppose that such an algorithm sampling $|S| \cdot |T| < \frac{c^2}{\epsilon^4}$ entries exists, for some constant $c > 0$. Then by Yao's min-max principle, there is a fixed submatrix $S, T \subset [n]$ such that, with probability $9/10$ over the distribution $\frac{\mathcal{D}_1+\mathcal{D}_2}{2}$, the algorithm correctly distinguishes $\mathcal{D}_1$ from $\mathcal{D}_2$ given only $A_{S\times T}$. Suppose WLOG that $|S| \leq \frac{c}{\epsilon^2}$. Then consider the case only when $\mathbf{A}_1$ or $\mathbf{A}_2$ is returned by either of the distributions, and not their transpose, which occurs with probability at least $1/2$. Then $A_{S\times T}$ is just a set of $|S|$ rows, each of which are either all 0's or all 1's. Moreover, each row is set to being the all 1's row independently with probability p_1 in the case of $\mathcal{D}_2$, and

p_2 in the case of $\mathcal{D}_2$. Thus, by Independence across rows, the behavior of the algorithm can be assumed to depend only on the number of rows which are set to 1. Thus, in the case of $\mathcal{D}_1$ the algorithm receives $X_1 \sim \texttt{Bin}(|S|, p_1)$ and in $\mathcal{D}_2$ the algorithm receives $X_2 \sim \texttt{Bin}(|S|, p_2)$. Then if $d_{TV}(X_1, X_2)$ is the total variational distance between X_1, X_2, then by Equation 2.15 of [AJ06], assuming that $\epsilon\sqrt{|S|}$ is smaller than some constant (which can be obtained by setting c small enough), we have

$$d_{TV}(X_1, X_2) \leq O(\epsilon\sqrt{|S|})$$

Which is at most $1/100$ for c a small enough constant. Thus any algorithm can correctly distinguish these two distributions with advantage at most $1/100$. Since we restricted our attention to the event when rows were set and not columns, and since we conditioned on the gap between the norms which held with probability $99/100$, it follows that the algorithm distinguishes $\mathcal{D}_1$ from $\mathcal{D}_2$ with probability at most $1/2 + 1/4 + (2/100) < 4/5$, which completes the proof. □

9.6 Proof of Eigenvalue Identity

Proposition 9.6.1. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be any real symmetric matrix. Then $\min_{\mathbf{B}\succeq 0} \|\mathbf{A} - \mathbf{B}\|_F^2 = \sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{A})$.*

Proof. Let g_i be the eigenvector associated with $\lambda_i = \lambda_i(\mathbf{A})$. First, setting $\mathbf{B} = \sum_{i:\lambda_i(\mathbf{A})\geq 0} \lambda_i g_i g_i^\top$, which is a PSD matrix, we have $\|\mathbf{A} - \mathbf{B}\|_F^2 = \|\sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i(g_i g_i^\top \mathbf{A})\|_2^2 = \sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{A})$, where the second equality follows from the Pythagorean Theorem, which proves that $\min_{\mathbf{B}\succeq 0} \|\mathbf{A}-\mathbf{B}\|_F^2 \leq \sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{A})$. To see the other direction, fix any PSD matrix $\mathbf{B}$, and let $\mathbf{Z} = \mathbf{B} - \mathbf{A}$. Then $\mathbf{Z} + \mathbf{A} \succeq 0$, where $\succeq$ is the Lowner ordering, thus $\mathbf{Z} \succeq -\mathbf{A}$, which by definition implies that $x^\top \mathbf{Z} x \geq -x^\top \mathbf{A} x$ for all $x \in \mathbb{R}^n$. Then by the Courant-Fischer variational characterization of eigenvalues, we have that $\lambda_i(\mathbf{Z}) \geq -\lambda_i(\mathbf{A})$ for all i. In particular, $|\lambda_i(\mathbf{Z})| \geq |\lambda_i(\mathbf{A})|$ for all i such that $\lambda_i(\mathbf{A}) < 0$. Thus $\|\mathbf{Z}\|_F^2 = \sum_i \lambda_i^2(\mathbf{Z}) \geq \sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{Z}) \geq \sum_{i:\lambda_i(\mathbf{A})<0} \lambda_i^2(\mathbf{A})$, which completes the proof.

□

Chapter 10

Kronecker Product Regression and Low-Rank Approximation

We now shift our perspective from the property testing framework to a more traditional computational model, where instead of minimizing the number of queries we make to the matrix, our goal is to solve some numerical linear algebra task as quickly as possible. For the remainder of Part II, we will focus on two fundamental linear algebraic problems: *regression* and *low-rank approximation*. In this section, we study the problem of solving ℓ_p regression and low rank approximation when the design matrix is a *Kronecker product* of several smaller matrices. The materials in this chapter are based on our paper [DJS+19].

In the traditional setting for over-constrained ℓ_p regression, we are given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with $n \gg d$, and a vector $b \in \mathbb{R}^d$, and are tasked with finding a x solution to the optimization problem:

$$\min_{x \in \mathbb{R}^d} \|\mathbf{A}x - b\|_p$$

While in general this problem requires $\Omega(\mathrm{nnz}(\mathbf{A}))$ time [W+14][1], in recent years a prolific line of work has applied sketching to solve regression and low rank approximation in *sublinear* time when the design matrix $\mathbf{A}$ admits nice structural properties [SWZ16, LHW17, DSSW18, SWZ19, SW19, BW18, BCW20, AKK+20a]. Common examples include when $\mathbf{A}$ is a PSD or Vandermonde matrix, or is given as a tensor product of several smaller matrices.

This focus of this chapter falls under the umbrella of the above line of work. Specifically, we study the Kronecker product regression problem, in which the design matrix is a Kronecker

[1] recall $\mathrm{nnz}(\mathbf{A})$ is the number of non-zero entries in $\mathbf{A}$, i.e., the *input sparsity*.

product of two or more matrices. Namely, $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q$, and one is given as input the matrices $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$. Note that for such a product, we have $\mathbf{A} \in \mathbb{R}^{n \times d}$ where $n = n_1 \cdots n_q$ and $d = d_1 \cdots d_q$, so the design matrix grows exponentially in the number of factors. Even computing such a product, let alone solving regression with it, is extremely expensive computationally.

The study of fast regression on such matrices was initiated by Diao, Song, Sun, and Woodruff [DSSW18], who gave an algorithm which runs in time faster than forming the Kronecker product $\mathbf{A} \in \mathbb{R}^{n_1 \cdots n_q \times d_1 \cdots d_q}$. Specifically, for $p = 2$ they achieve a running time of $O(\sum_{i=1}^q \text{nnz}(\mathbf{A}_i) + \text{nnz}(b))$, which is sublinear in the sparsity of $\mathbf{A}$, but may still require $\Omega(n)$ time due to the depenedency on $\text{nnz}(b)$. For $1 \leq p < 2$, their runtime is suffers additional polynomial factors; for instance, for $p = 1$, $q = 2$ and $n_1 = n_2$, their runtime is $O(n_1^{3/2}\text{poly}(d_1 d_2) + \text{nnz}(b))$.

In this chapter, we provide significantly faster algorithms for Kronecker product regression. For $p = 2$, our running time is $O(\sum_{i=1}^q \text{nnz}(\mathbf{A}_i))$, which has no dependence on $\text{nnz}(b)$. For $1 \leq p < 2$, our running time is $O(\sum_{i=1}^q \text{nnz}(\mathbf{A}_i) + \text{nnz}(b))$, which matches the prior best running time for $p = 2$. Additionally, we study the related all-pairs regression problem, where given $\mathbf{A} \in \mathbb{R}^{n \times d}, b \in \mathbb{R}^n$, one wants to solve $\min_{x \in \mathbb{R}^d} \|\bar{\mathbf{A}}x - \bar{b}\|_p$, where $\bar{\mathbf{A}} \in \mathbb{R}^{n^2 \times d}$ and $\bar{b} \in \mathbb{R}^{n^2}$ consist of all pairwise differences of the rows of $\mathbf{A}$ and b, respectively. We give an $O(\text{nnz}(A))$ time algorithm for $p \in [1, 2]$, improving the $\Omega(n^2)$ time required to form $\bar{\mathbf{A}}$. Finally, we initiate the study of Kronecker product low rank where the goal is to output a low rank approximation to a Kronecker product matrix $\mathbf{A} = \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q$. We give algorithms which run in $O(\sum_{i=1}^q \text{nnz}(\mathbf{A}_i))$ time, significantly faster than computing $\mathbf{A}$.

Highlighted Contributions

The main contributions of this chapter as as follows:

- We design the first *input-sparsity* runtime algorithms for Kronecker product ℓ_p regression for $1 \leq p < 2$, and low-rank approximation (Section 10.2)
- We improve the state of the art runtime for Kronecker product ℓ_2 regression by removing the dependency on $\text{nnz}(b)$ from the runtime (Section 10.2.2)
- We initiate the study of Kronecker product low rank approximation, and give input sparsity $O(\sum_{i=1}^q \text{nnz}(\mathbf{A}_i))$ time algorithms for this problem (Section 10.4).

10.1 Background

In the q-th order Kronecker product regression problem, one is given matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_q$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, as well as a vector $b \in \mathbb{R}^{n_1 n_2 \cdots n_q}$, and the goal is to obtain a solution to the optimization problem:

$$\min_{x \in \mathbb{R}^{d_1 d_2 \cdots d_q}} \|(\mathbf{A}_1 \otimes \mathbf{A}_2 \cdots \otimes \mathbf{A}_q)x - b\|_p$$

Kronecker product regression is a special case of ordinary regression in which the design matrix is highly structured. Such matrices naturally arise in applications such as spline regression, signal processing, and multivariate data fitting. We direct the reader to [VL92, VLP93, GVL13, DSSW18] for further background and applications of Kronecker product regression. Our focus is on the *over-constrained regression* setting, when $n_i \gg d_i$ for each i, and so the goal is to have a small running time dependence on the n_i's. This is the primary setting in which such Kronecker products arise, and is the standard setting for the application of sketching techniques to big-data NLA tasks (we direct the reader to[Mah11, W+14] for surveys)

The results of Diao, Song, Sun, and Woodruff [DSSW18] utilized sketching techniques to output an $x \in \mathbb{R}^{d_1 d_2 \cdots d_q}$ with objective function at most $(1+\epsilon)$-times larger than optimal, for both least squares (i.e., $p = 2$) and least absolute deviation (i.e., $p = 1$) Kronecker product regression. For least squares regression, the algorithm of [DSSW18] achieves $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{nnz}(b) + \mathrm{poly}(d/\epsilon))$ time.

Observe that explicitly forming the matrix $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q$ would take $\prod_{i=1}^q \mathrm{nnz}(\mathbf{A}_i)$ time, which can be as large as $\prod_{i=1}^q n_i d_i$, and so the results of [DSSW18] offer a large computational advantage. Unfortunately, since $b \in \mathbb{R}^{n_1 n_2 \cdots n_q}$, we can have $\mathrm{nnz}(b) = \prod_{i=1}^q n_i$, and therefore $\mathrm{nnz}(b)$ is likely to be the dominant term in the running time. This leaves open the question of whether it is possible to solve this problem in time *sub-linear* in $\mathrm{nnz}(b)$, with a dominant term of $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i))$.

For least absolute deviation regression, the bounds of [DSSW18] achieved are still an improvement over computing $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q$, though worse than the bounds for least squares regression. The authors focus on $q = 2$ and the special case $n = n_1 = n_2$. Here, they obtain a running time of $O(n^{3/2}\mathrm{poly}(d_1 d_2/\epsilon) + \mathrm{nnz}(b))$[2]. This leaves open the question of whether an *input-sparsity* $O(\mathrm{nnz}(\mathbf{A}_1) + \mathrm{nnz}(\mathbf{A}_2) + \mathrm{nnz}(b) + \mathrm{poly}(d_1 d_2/\epsilon))$ time algorithm exists.

[2]We remark that while the $\mathrm{nnz}(b)$ term is not written in the Theorem of [DSSW18], their approach of leverage score sampling from a well-conditioned basis requires one to sample from a well conditioned basis of $[\mathbf{A}_1 \otimes \mathbf{A}_2, b]$ for a subspace embedding. As stated, their algorithm only sampled from $[\mathbf{A}_1 \otimes \mathbf{A}_2]$. To fix this omission, their algorithm would require an additional $\mathrm{nnz}(b)$ time to leverage score sample from the augmented matrix.

All-Pairs Regression. In addition to ℓ_p regression we also study the related all-pairs regression problem. Given $\mathbf{A} \in \mathbb{R}^{n\times d}, b \in \mathbb{R}^n$, the goal is to approximately solve the ℓ_p regression problem

$$\min_x \|\bar{\mathbf{A}}x - \bar{b}\|_p$$

where $\bar{\mathbf{A}} \in \mathbb{R}^{n^2\times d}$ is the matrix formed by taking all pairwise differences of the rows of $\mathbf{A}$ (and $\bar{b}$ is defined similarly). Note that one can write $\bar{\mathbf{A}}$ as the difference of Kronecker products: $\bar{\mathbf{A}} = \mathbf{A} \otimes \mathbf{1}^n - \mathbf{1}^n \otimes \mathbf{A}$, where $\mathbf{1}^n \in \mathbb{R}^n$ is the all ones vector. For $p = 1$, this is known as the *rank regression estimator*, which has a long history in statistics. It is closely related to the Wilconxon rank test [WL09] in statistics, and enjoys the desirable property of being robust with substantial efficiency gain with respect to heavy-tailed random errors, while maintaining high efficiency for Gaussian errors [WKL09, WL09, WPB+18, Wan19]. Recently, the all-pairs loss function was also used by [WPB+18] as an alternative approach to overcoming the challenges of tuning parameter selection for the Lasso algorithm. Unfortunately, the rank regression estimator is computationally intensive to compute, even for moderately sized data, since the standard procedure (for $p = 1$) is to solve a linear program with $O(n^2)$ constraints. In this work, we demonstrate the first highly efficient algorithm for this estimator.

Low-Rank Approximation. Lastly, we extend our techniques to the Low Rank Approximation (LRA) problem for Kronecker product matrices.. Here, given a large data matrix $\mathbf{A}$, the goal is to find a low rank matrix $\mathbf{B}$ which well-approximates $\mathbf{A}$. LRA is useful in numerous applications, such as compressing massive datasets to their primary components for storage, denoising, and fast matrix-vector products. Thus, designing fast algorithms for approximate LRA has become a large and highly active area of research; see [W+14] for a survey.

Given q matrices $\mathbf{A}_1, \cdots, \mathbf{A}_q$ where $\mathbf{A}_i \in \mathbb{R}^{n_i\times d_i}$, $n_i \gg d_i$, $\mathbf{A} = \otimes_{i=1}^q \mathbf{A}_i$, the goal is to output a rank-k matrix $\mathbf{B} \in \mathbb{R}^{n\times d}$ such that $\|\mathbf{B} - \mathbf{A}\|_F^2 \leq (1+\epsilon)\,\mathrm{OPT}_k$, where OPT_k is the cost of the best rank-k approximation, $n = n_1 \cdots n_q$, and $d = d_1 \cdots d_q$. Note that the fastest general purpose algorithms for this problem run in time $O(\mathrm{nnz}(\mathbf{A}) + \mathrm{poly}(dk/\epsilon))$ [CW17]. However, as in regression, if $\mathbf{A} = \otimes_{i=1}^q \mathbf{A}_i$, we have $\mathrm{nnz}(A) = \prod_{i=1}^q \mathrm{nnz}(\mathbf{A}_i)$, which grows very quickly. Instead, one might also hope to obtain a running time of $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{poly}(dk/\epsilon))$.

10.1.1 Contributions

Our main contribution is an input sparsity time $(1+\epsilon)$-approximation algorithm to Kronecker product regression for every $p \in [1, 2]$, and $q \geq 2$. Given $\mathbf{A}_i \in \mathbb{R}^{n_i\times d_i}$, $i = 1, \ldots, q$, and $b \in \mathbb{R}^n$

where $n = \prod_{i=1}^{q} n_i$, together with accuracy parameter $\epsilon \in (0, 1/2)$ and failure probability $\delta > 0$, the goal is to output a vector $x' \in \mathbb{R}^d$ where $d = \prod_{i=1}^{q} d_i$ such that

$$\|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b\|_p \leq (1+\epsilon) \min_x \|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - b\|_p$$

holds with probability at least $1 - \delta$. In this chapter, we prove the following two Theorems:

Theorem 127 (Restatement of Theorem 131, Kronecker product ℓ_2 regression)**.** *Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal row sampling matrix generated via Proposition 10.2.4, with $m = \Theta(1/(\delta\epsilon^2))$ non-zero entries, and let $\mathbf{A} = \otimes_{i=1}^{q} \mathbf{A}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, and $b \in \mathbb{R}^n$, where $n = \prod_{i=1}^{q} n_i$ and $d = \prod_{i=1}^{q} d_i$. Then we have let $\hat{x} = \arg\min_{x \in \mathbb{R}^d} \|\mathbf{D}\mathbf{A}x - \mathbf{D}b\|_2$, and let $x^* = \arg\min_{x' \in \mathbb{R}^d} \|\mathbf{A}x - b\|_2$. Then with probability $1 - \delta$, we have*

$$\|\mathbf{A}\hat{x} - b\|_2 \leq (1+\epsilon)\|\mathbf{A}x^* - b\|_2$$

Moreover, the total runtime requires to compute $\hat{x}$ is

$$\tilde{O}\left(\sum_{i=1}^{q} \mathrm{nnz}(\mathbf{A}_i) + \mathrm{poly}(dq/(\delta\epsilon))\right).$$

Theorem 128 (Restatement of Theorem 132, Kronecker product ℓ_p regression)**.** *Fix $1 \leq p < 2$. Then for any constant $q = O(1)$, given matrices $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_q$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, let $n = \prod_{i=1}^{q} n_i$, $d = \prod_{i=1}^{q} d_i$. Let $\hat{x} \in \mathbb{R}^d$ be the output of Algorithm 8. Then*

$$\|(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)\hat{x} - b\|_p \leq (1+\epsilon) \min_{x \in \mathbb{R}^n} \|(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x - b\|_p$$

holds with probability at least $1 - \delta$. In addition, our algorithm takes

$$\widetilde{O}\left(\left(\sum_{i=1}^{q} \mathrm{nnz}(\mathbf{A}_i) + \mathrm{nnz}(b) + \mathrm{poly}(d\log(1/\delta)/\epsilon)\right)\log(1/\delta)\right)$$

time to output $\widehat{x} \in \mathbb{R}^d$.

Observe that in both cases, this running time is significantly faster than the time to write down $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q$. For $p = 2$, up to logarithmic factors, the running time is the same as the time required to simply read each of the $\mathbf{A}_i$. Notice further that this is *sub-linear* in the input size, since it does not depend on $\mathrm{nnz}(b)$. Moreover, in the setting $p < 2$, $q = 2$ and $n_1 = n_2$ considered in [DSSW18], our algorithm offers a substantial improvement over their running time of $O(n^{3/2}\mathrm{poly}(d_1 d_2/\epsilon))$. In addition, we empirically evaluate our Kronecker product regression

algorithm on exactly the same datasets as those used in [DSSW18]. For $p \in \{1, 2\}$, the accuracy of our algorithm is nearly the same as that of [DSSW18], while the running time is significantly faster.

For the all-pairs (or rank) regression problem, we first note that for $\mathbf{A} \in \mathbb{R}^{n \times d}$, one can rewrite $\bar{\mathbf{A}} \in \mathbb{R}^{n^2 \times d}$ as the difference of Kronecker products $\bar{\mathbf{A}} = \mathbf{A} \otimes \mathbf{1}^n - \mathbf{1}^n \otimes \mathbf{A}$ where $\mathbf{1}^n \in \mathbb{R}^n$ is the all ones vector. Since $\bar{\mathbf{A}}$ is not a Kronecker product itself, our earlier techniques for Kronecker product regression are not directly applicable. Therefore, we utilize new ideas, in addition to careful sketching techniques, to obtain an $\widetilde{O}(\mathrm{nnz}(A) + \mathrm{poly}(d/\epsilon))$ time algorithm for $p \in [1, 2]$, which improves substantially on the $O(n^2 d)$ time required to even compute $\bar{\mathbf{A}}$, by a factor of at least n. Formally, our result for all-pairs regression is as follows:

Theorem 129 (Restatement of Theorem 133)**.** *Given $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$, for $p \in [1, 2]$ there is an algorithm for the All-Pairs Regression problem that outputs $\hat{x} \in \mathbb{R}^d$ such that with probability $1 - \delta$ we have*

$$\|\bar{\mathbf{A}}\hat{x} - \bar{b}\|_p \leq (1 + \epsilon) \min_{x \in \mathbb{R}^d} \|\bar{\mathbf{A}}x - \bar{b}\|_p$$

Where $\bar{\mathbf{A}} = \mathbf{A} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{A} \in \mathbb{R}^{n^2 \times d}$ and $\bar{b} = b \otimes \mathbf{1} - \mathbf{1} \otimes b \in \mathbb{R}^{n^2}$. For $p < 2$, the running time is $\widetilde{O}(nd + \mathrm{poly}(d/(\epsilon\delta)))$, and for $p = 2$ the running time is $O(\mathrm{nnz}(A) + \mathrm{poly}(d/(\epsilon\delta)))$.

Our main technical contribution for both our ℓ_p regression algorithm and the rank regression problem is a novel and highly efficient ℓ_p sampling algorithm. Specifically, for the rank-regression problem we demonstrate, for a given $x \in \mathbb{R}^d$, how to independently sample s entries of a vector $\bar{\mathbf{A}}x = y \in \mathbb{R}^{n^2}$ from the ℓ_p distribution $(|y_1|^p/\|y\|_p^p, \ldots, |y_{n^2}|^p/\|y\|_p^p)$ in $\widetilde{O}(nd + \mathrm{poly}(ds))$ time. For the ℓ_p regression problem, we demonstrate the same result when $y = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - b \in \mathbb{R}^{n_1 \cdots n_q}$, and in time $\widetilde{O}(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{nnz}(b) + \mathrm{poly}(ds))$. This result allows us to sample a small number of rows of the input to use in our sketch. Our algorithm draws from a large number of disparate sketching techniques, such as the dyadic trick for quickly finding heavy hitters [CM05, KNPW11b, LNNT16, NS19], and the precision sampling framework from the streaming literature [AKO11].

For the Kronecker Product Low-Rank Approximation (LRA) problem, we give an input sparsity $O(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{poly}(dk/\epsilon))$-time algorithm which computes a rank-k matrix $\mathbf{B}$ such that $\|\mathbf{B} - \otimes_{i=1}^q \mathbf{A}_i\|_F^2 \leq (1 + \epsilon) \min_{\mathrm{rank}-k\ B'} \|\mathbf{B}' - \otimes_{i=1}^q \mathbf{A}_i\|_F^2$. Note again that the dominant term $\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i)$ is substantially smaller than the $\mathrm{nnz}(A) = \prod_{i=1}^q \mathrm{nnz}(\mathbf{A}_i)$ time required to write down the Kronecker Product A, which is also the running time of state-of-the-art general purpose

LRA algorithms [CW17, MM13, NN13]. Thus, our results demonstrate that substantially faster algorithms for approximate LRA are possible for inputs with a Kronecker product structure.

Theorem 130 (Restatement of Theorem 134)**.** *For any constant $q \geq 2$, there is an algorithm which runs in time $O(\sum_{i=1}^{q} \mathrm{nnz}(\mathbf{A}_i) + d\mathrm{poly}(k/\epsilon))$ and outputs a rank k-matrix $\mathbf{B}$ in factored form such that $\|\mathbf{B} - \mathbf{A}\|_F \leq (1+\epsilon)\,\mathrm{OPT}_k$ with probability $9/10$.*

Our technical contributions employed towards the proof of Theorem 134 involve demonstrating that useful properties of known sketching matrices hold also for the Kronecker product of these matrices. Specifically, we demonstrate the Kronecker products of the well-known *count-sketch* matrices satisfy the property of being *Projection Cost Preserving Sketches* (PCP). By properties of the Kronecker product, we can quickly apply such a sketching matrix to the input matrix A, and the PCP property will allow us to bound the cost of the best low rank approximation obtained via the sketch.

10.1.2 Useful Sketches and Well Conditioned Bases

In this section, we introduce several useful sketching primitives which will be used throughout this chapter.

Stable Transformations As we did in Part I, we will make substantial use of the p-stable distribution $\mathcal{D}_p$ in this chapter (see background on p-stables in Section 2.2.2). Recall that $\mathcal{D}_p$ has the property that if $z_1, \ldots, z_n \sim D_p$ are i.i.d., and $a \in \mathbb{R}^n$, then $\sum_{i=1}^{n} z_i a_i \sim z\|a\|_p$ where $\|a\|_p = (\sum_{i=1}^{n} |a_i|^p)^{1/p}$, and $z \sim \mathcal{D}_p$.

Definition 10.1.1 (Dense p-stable Transform, [CDMI+13, SW11])**.** *Let $p \in [1,2]$. Let $\mathbf{S} = \sigma \cdot \mathbf{C} \in \mathbb{R}^{m \times n}$, where σ is a scalar, and each entry of $\mathbf{C} \in \mathbb{R}^{m \times n}$ is chosen independently from $\mathcal{D}_p$.*

We will also need a sparse version of the above.

Definition 10.1.2 (Sparse p-Stable Transform, [MM13, CDMI+13])**.** *Let $p \in [1,2]$. Let $\mathbf{\Pi} = \sigma \cdot \mathbf{SC} \in \mathbb{R}^{m \times n}$, where σ is a scalar, $\mathbf{S} \in \mathbb{R}^{m \times n}$ has each column chosen independently and uniformly from the m standard basis vectors of $\mathbb{R}^m$, and $\mathbf{C} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with diagonals chosen independently from the standard p-stable distribution. For any matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, $\mathbf{\Pi A}$ can be computed in $O(\mathrm{nnz}(\mathbf{A}))$ time.*

One nice property of p-stable transformations is that they provide *low-distortion ℓ_p embeddings*.

Lemma 10.1.3 (Theorem 1.4 of [WW19]; see also Theorem 2 and 4 of [MM13] for earlier work[3]). *Fix $\mathbf{A} \in \mathbb{R}^{n\times d}$, and let $\mathbf{S} \in \mathbb{R}^{k\times n}$ be a sparse or dense p-stable transform for $p \in [1,2)$, with $k = \Theta(d^2/\delta)$. Then with probability $1-\delta$, for all $x \in \mathbb{R}^d$:*

$$\|\mathbf{A}x\|_p \leq \|\mathbf{S}\mathbf{A}x\|_p \leq O(d\log d)\|\mathbf{A}x\|_p$$

We simply call a matrix $\mathbf{S} \in \mathbb{R}^{k\times n}$ a low distortion ℓ_p embedding for $\mathbf{A} \in \mathbb{R}^{n\times d}$ if it satisfies the above inequality for all $x \in \mathbb{R}^d$.

Leverage Scores & Well Condition Bases. We first recall the definition of ℓ_2 leverage scores from Chapter 2.

Definition 2.4.1 (Leverage Scores). *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be its singular value decomposition. For each $i \in [n]$, we define the i-th leverage score of $\mathbf{A}$, denoted τ_i, by $\tau_i = \|\mathbf{U}_{i,*}\|_2^2$. Equivalently, of $\mathbf{A} = \mathbf{Q}\mathbf{R}$ is the QR-decomposition of $\mathbf{A}$, we have $\mathbf{A}\mathbf{R}^{-1} = \mathbf{U}$, thus $\tau_i = \|(\mathbf{A}\mathbf{R}^{-1})_{i,*}\|_2^2$.*

We now introduce the notion of a *well-conditioned basis* for a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$.

Definition 10.1.4 ((ℓ_p, α, β) Well-Conditioned Basis, [Cla05]). *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, we say $\mathbf{U} \in \mathbb{R}^{n\times d}$ is an (ℓ_p, α, β) well-conditioned basis for the column span of $\mathbf{A}$ if the columns of $\mathbf{U}$ span the columns of $\mathbf{A}$, and if for any $x \in \mathbb{R}^d$, we have $\alpha\|x\|_p \leq \|\mathbf{U}x\|_p \leq \beta\|x\|_p$, where $\alpha \leq 1 \leq \beta$. If $\beta/\alpha = d^{O(1)}$, then we simply say that $\mathbf{U}$ is an ℓ_p well conditioned basis for $\mathbf{A}$.*

The following fact demonstrates that the left singular vectors of $\mathbf{A}$ yield a ℓ_p well-conditioned basis for $\mathbf{A}$.

Fact 10.1.5 ([WW19, MM13]). *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$, and let $\mathbf{S}\mathbf{A} \in \mathbb{R}^{k\times d}$ be a low distortion ℓ_p embedding for $\mathbf{A}$ (see Lemma 10.1.3), where $k = O(d^2/\delta)$. Let $\mathbf{S}\mathbf{A} = \mathbf{Q}\mathbf{R}$ be the QR decomposition of $\mathbf{S}\mathbf{A}$. Then $\mathbf{A}\mathbf{R}^{-1}$ is an ℓ_p well-conditioned basis with probability $1-\delta$.*

[3] In discussion with the authors of these works, the original $O((d\log d)^{1/p})$ distortion factors stated in these papers should be replaced with $O(d\log d)$; as we do not optimize the poly(d) factors in our analysis, this does not affect our bounds.

10.2 Kronecker Product Regression

We first introduce our algorithm for $p = 2$. Our algorithm for $1 \leq p < 2$ is given in Section 10.2.2. Our regression algorithm for $p = 2$ is formally stated in Algorithm 7. Recall that our input design matrix is $\mathbf{A} = \otimes_{i=1}^{q} \mathbf{A}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, and we are also given $b \in \mathbb{R}^{n_1 \cdots n_q}$. Let $n = \prod_{i=1}^{q} n_i$ and $d = \prod_{i=1}^{q} d_i$. The crucial insight of the algorithm is that one can approximately compute the leverage scores of $\mathbf{A}$ given only good approximations to the leverage scores of each $\mathbf{A}_i$. Applying this fact gives a efficient algorithm for sampling rows of $\mathbf{A}$ with probability proportional to the leverage scores. Following standard arguments, we will show that by restricting the regression problem to the sampled rows, we can obtain our desired $(1 \pm \epsilon)$-approximate solution efficiently.

Our main theorem for this section is stated below. A full proof of the theorem can be found Section 10.2.1.

Theorem 131 (Kronecker product ℓ_2 regression)**.** *Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal row sampling matrix generated via Proposition 10.2.4, with $m = \Theta(d/(\delta\epsilon^2))$ non-zero entries, and let $\mathbf{A} = \otimes_{i=1}^{q} \mathbf{A}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, and $b \in \mathbb{R}^n$, where $n = \prod_{i=1}^{q} n_i$ and $d = \prod_{i=1}^{q} d_i$. Then let $\hat{x} = \arg\min_{x \in \mathbb{R}^d} \|\mathbf{D}\mathbf{A}x - \mathbf{D}b\|_2$, and let $x^* = \arg\min_{x' \in \mathbb{R}^d} \|\mathbf{A}x - b\|_2$. Then with probability $1 - \delta$, we have*

$$\|\mathbf{A}\hat{x} - b\|_2 \leq (1 + \epsilon)\|\mathbf{A}x^* - b\|_2.$$

Moreover, the total running time required to compute $\hat{x}$ is $\tilde{O}(\sum_{i=1}^{q} \mathrm{nnz}(\mathbf{A}_i) + (dq/(\delta\epsilon))^{O(1)})$.[4].

10.2.1 Kronecker Product ℓ_2 Regression

We now prove the correctness of our ℓ_2 Kronecker product regression algorithm. Specifically, we prove Theorem 131. To prove correctness, we need to establish several facts about the leverage scores of a Kronecker product.

Proposition 10.2.1. *Let $\mathbf{U}_i \in \mathbb{R}^{n_i \times d_i}$ be an orthonormal basis for $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$. Then $\mathbf{U} = \otimes_{i=1}^{q} \mathbf{U}_i$ is an orthonormal basis for $\mathbf{A} = \otimes_{i=1}^{q} \mathbf{A}_i$.*

[4]We remark that the exponent of d in the runtime can be bounded by 3. To see this, first note that the main computation taking place is the leverage score computation from Proposition 10.2.3. For a q input matrices, we need to generate the leverage scores to precision $\Theta(1/q)$, and thus the complexity from running Proposition 10.2.3 to approximate leverage scores is $O(d^3/q^4)$ by the results of [CW17]. The remaining computation is to compute the pseudo-inverse of a $d/\epsilon^2 \times d$ matrix, which requires $O(d^3/\epsilon^2)$ time, so the additive term in the Theorem can be replaced with $O(d^3/\epsilon^2 + d^3/q^4)$.

Algorithm 7: Our ℓ_2 Kronecker Product Regression Algorithm

ℓ_2 **Kronecker Regression**$(\{\mathbf{A}_i, n_i, d_i\}_{i\in[q]}, b)$ //Theorem 131

$d \leftarrow \prod_{i=1}^q d_i$, $n \leftarrow \prod_{i=1}^q n_i$, $m \leftarrow \Theta(d/(\delta\epsilon^2))$.

Compute approximate leverage scores $\tilde{\tau}_i(\mathbf{A}_j)$ for all $j \in [q]$, $i \in [n_j]$.

//Proposition 10.2.3

Construct diagonal leverage score sampling matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$, with m non-zero entries

//Proposition 10.2.4

Compute (via the psuedo-inverse)

$$\widehat{x} = \arg\min_{x\in\mathbb{R}^d} \|\mathbf{D}(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x - \mathbf{D}b\|_2$$

Return $\widehat{x}$

Proof. Note that the column norm of each column of $\mathbf{U}$ is the product of column norms of the $\mathbf{U}_i$'s, which are all 1. Thus $\mathbf{U}$ has unit norm columns. It suffices then to show that all the singular values of $\mathbf{U}$ are 1 or -1, but this follows from the fact that the singular values of $\mathbf{U}$ are the product of singular values of the $\mathbf{U}_i$'s, which completes the proof. □

Corollary 10.2.2. *Let $\mathbf{A} = \otimes_{i=1}^q \mathbf{A}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i\times d_i}$. Fix any $\vec{i} = (i_1, \ldots, i_q) \in [n_1] \times [n_2] \times \cdots \times [n_q]$, and let $\vec{i}$ index into a row of $\mathbf{A}$ in the natural way. Then the $\vec{i}$-th leverage score of $\mathbf{A}$ is equal to $\prod_{j=1}^q \tau_{i_j}(\mathbf{A}_j)$, where $\tau_t(\mathbf{B})$ is the t-th leverage score of a matrix $\mathbf{B}$.*

Proof. Note $\mathbf{U} = \otimes_{i=1}^q \mathbf{U}_i$ is an orthonormal basis for $\mathbf{A} = \otimes_{i=1}^q \mathbf{A}_i$ by the prior Proposition. Now if $\mathbf{U}_{\vec{i},*}$ is the $\vec{i}$-th row of $\mathbf{U}$, then by fundamental properties of Kronecker products [VL00], we have $\|\mathbf{U}_{\vec{i},*}\|_2 = \prod_{j=1}^q \|(\mathbf{U}_j)_{i_j,*}\|_2$, which completes the proof. Note here that we used the fact that leverage scores are independent of the choice of orthonormal basis [W+14]. □

Proposition 10.2.3 (Theorem 29 of [CW17])**.** *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $\tau \in \mathbb{R}^n$ be the ℓ_2 leverage scores of $\mathbf{A}$. Then there is an algorithm which computes values $\tilde{\tau}_1, \tilde{\tau}_2, \ldots, \tilde{\tau}_n$ such that $\tilde{\tau}_i = (1 \pm \epsilon)\tau_i$ simultaneously for all $i \in [n]$ with probability $1 - 1/n^c$ for any constant $c \geq 1$. The runtime is $\tilde{O}(\mathrm{nnz}(\mathbf{A}) + d^3/\epsilon^2)$.*

Proposition 10.2.4. *Given $\mathbf{A} = \otimes_{i=1}^q \mathbf{A}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i\times d_i}$, set $p_i = \frac{\tau_i(\mathbf{A})}{\sum_j \tau_j(\mathbf{A})}$. Then there is an algorithm which, with probability $1 - 1/n^c$ for any constant $c \geq 1$, outputs a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ with m non-zeros entries, such that $\mathbf{D}_{i,i} = 1/(m\tilde{p}_i)$ with probability $\tilde{p}_i$, and $\mathbf{D}_{i,i}$ is zero otherwise, for some $\tilde{p}_i \in (1 \pm 1/10)p_i(\mathbf{A})$. The time required is $\tilde{O}(\sum_{i=1}^q \mathrm{nnz}(\mathbf{A}_i) + \mathrm{poly}(dq/\epsilon) + mq)$.*

Proof. By Proposition 10.2.3, we can compute approximate leverage scores of each $\mathbf{A}_i$ up to

error $\Theta(1/q)$ in time $\tilde{O}(\mathrm{nnz}(\mathbf{A}_i) + \mathrm{poly}(d/\epsilon))$ with high probability. To sample a leverage score from $\mathbf{A}$, it suffices to sample one leverage score from each of the $\mathbf{A}_i$'s by Corollary 10.2.2. The probability that a given row $\vec{i} = (i_1, \ldots, i_q) \in [n_1] \times [n_2] \times \cdots \times [n_q]$ of $\mathbf{A}$ is chosen is $\prod_{j=1}^{q} \tilde{\tau}(\mathbf{A}_j)_{i_j} = (1 \pm \Theta(1/q))^q \tau_{\vec{i}}(\mathbf{A}) = (1 \pm 1/10)\tau_{\vec{i}}(\mathbf{A})$ as needed. Obtaining a sample takes $\tilde{O}(1)$ time per $\mathbf{A}_i$ (since a random number needs to be generated to $O(\log(n))$-bits of precision in expectation and with high probability to obtain this sample), thus $O(q)$ time overall, so repeating the sampling M times gives the desired additive mq runtime. □

The $q = 1$ version of the following result can be found in [CW17, SWZ19].

Proposition 10.2.5. *Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal row sampling matrix generated via Proposition 10.2.4, with $m = \Theta(1/(\delta\epsilon^2))$ non-zero entries. Let $\mathbf{A} = \otimes_{i=1}^{q} \mathbf{A}_i$ as above, and let $\mathbf{U} \in \mathbb{R}^{n \times r}$ be an orthonormal basis for the column span of $\mathbf{A}$, where $r = \mathrm{rank}(\mathbf{A})$. Then for any matrix $\mathbf{B}$ with n rows, we have*

$$\mathbf{Pr}\left[\|\mathbf{U}^\top \mathbf{D}^\top \mathbf{D}\mathbf{B} - \mathbf{U}^\top \mathbf{B}\|_F \leq \epsilon \|\mathbf{U}\|_F \|\mathbf{B}\|_F\right] \geq 1 - \delta$$

Proof. By definition of leverage scores and Proposition 10.2.4, $\mathbf{D}$ is a matrix which sample each row $\mathbf{U}_{i,*}$ of $\mathbf{U}$ with probability at least $(9/10)\|\mathbf{U}_{i,*}\|_2/\|\mathbf{U}\|_F$. Taking the average of m such rows, we obtain the approximate matrix product result with error $O(1/\sqrt{\delta m})$ with probability $1 - \delta$ by Theorem 2.1 of [KV17]. □

We now ready to prove the main theorem of this section, Theorem 131

Theorem 131 (Kronecker product ℓ_2 regression)**.** *Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal row sampling matrix generated via Proposition 10.2.4, with $m = \Theta(1/(\delta\epsilon^2))$ non-zero entries, and let $\mathbf{A} = \otimes_{i=1}^{q} \mathbf{A}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, and $b \in \mathbb{R}^n$, where $n = \prod_{i=1}^{q} n_i$ and $d = \prod_{i=1}^{q} d_i$. Then we have let $\hat{x} = \arg\min_{x \in \mathbb{R}^d} \|DAx - \mathbf{D}b\|_2$, and let $x^* = \arg\min_{x' \in \mathbb{R}^d} \|\mathbf{A}x - b\|_2$. Then with probability $1 - \delta$, we have*

$$\|\mathbf{A}\hat{x} - b\|_2 \leq (1 + \epsilon)\|\mathbf{A}x^* - b\|_2$$

Moreover, the total runtime requires to compute $\hat{x}$ is

$$\tilde{O}\left(\sum_{i=1}^{q} \mathrm{nnz}(\mathbf{A}_i) + \mathrm{poly}(dq/(\delta\epsilon))\right).$$

Proof. Let $\mathbf{U}$ be an orthonormal basis for the column span of $\mathbf{A}$. By Lemma 3.3 of [CW09],

we have $\|\mathbf{A}(\hat{x} - x^*)\|_2 \leq 2\sqrt{\epsilon}\|\mathbf{A}x^* - b\|_2$. Note that while Lemma 3.3 of [CW09] uses a different sketching matrix $\mathbf{D}$ than us, the only property required for the proof of Lemma 3.3 is that $|\mathbf{U}^\top\mathbf{D}^\top\mathbf{D}\mathbf{B} - \mathbf{U}^\top\mathbf{B}\|_F \leq \sqrt{\epsilon/d}\|\mathbf{A}\|_F\|\mathbf{B}\|_F$ with probability at least $1 - \delta$ for any fixed matrix $\mathbf{B}$, which we obtain by Proposition 10.2.5 by having $O(d/(\delta\epsilon^2))$ non-zeros on the diagonal of $\mathbf{D}$). By the normal equations, we have $\mathbf{A}^\top(\mathbf{A}x^* - b) = 0$, thus $\langle A(\hat{x} - x^*), (\mathbf{A}x^* - b)\rangle = 0$, and so by the Pythagorean theorem we have

$$\|\mathbf{A}\hat{x} - b\|_2^2 = \|\mathbf{A}x^* - b\|_2^2 + \|\mathbf{A}(\hat{x} - x^*)\|_2^2 \leq (1 + 4\epsilon)\|\mathbf{A}x^* - b\|_2^2$$

Which completes the proof after rescaling of ϵ. The runtime required to obtain the matrix $\mathbf{D}$ is $\tilde{O}(\sum_{i=1}^q \text{nnz}(\mathbf{A}_i) + \text{poly}(dq/\epsilon))$ by Proposition 10.2.4, where we set $\mathbf{D}$ to have $m = \Theta(d/(\delta\epsilon^2))$ non-zero entries on the diagonal. Once $\mathbf{D}$ is obtained, one can compute $\mathbf{D}(\mathbf{A} + b)$ in time $O(md)$, thus the required time is $O(\delta^{-1}(d/\epsilon)^2)$. Finally, computing $\hat{x}$ once $\mathbf{DA}, \mathbf{D}b$ are computed requires a single pseudo-inverse computation, which can be carried out in $O(\delta^{-1}d^3/\epsilon^2)$ time (since $\mathbf{DA}$ now has only $O(\delta^{-1}(d/\epsilon)^2)$ rows).

□

10.2.2 Kronecker Product ℓ_p Regression

We now consider ℓ_p regression for $1 \leq p < 2$. Our algorithm is stated formally in Algorithm 8. Our high level approach follows that of [DDH+09]. Namely, we first obtain a vector x' which is an $O(1)$-approximate solution to the optimal solution. This is done by first constructing (implicitly) a matrix $\mathbf{U} \in \mathbb{R}^{n\times d}$ that is a well-conditioned basis for the design matrix $\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q$. We then efficiently sample rows of $\mathbf{U}$ with probability proportional to their ℓ_p norm (which must be done without even explicitly computing most of $\mathbf{U}$). We then use the results of [DDH+09] to demonstrate that solving the regression problem constrained to these sampled rows gives a solution $x' \in \mathbb{R}^d$ such that

$$\|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b\|_p \leq 8 \min_{x\in\mathbb{R}^d} \|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b\|_p$$

We define the *residual error* $\rho = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b \in \mathbb{R}^n$ of x'. Our goal is to sample additional rows $i \in [n]$ with probability proportional to their residual error $|\rho_i|^p/\|\rho\|_p^p$, and solve the regression problem restricted to the sampled rows. However, we cannot afford to compute even a small fraction of the entries in ρ (even when b is dense, and certainly not when b is sparse). So to carry out this sampling efficiently, we design an involved, multi-part sketching

and sampling routine (described in Section 19). This sampling technique is the main technical contribution of this section, and relies on a number of techniques, such as the Dyadic trick for quickly finding heavy hitters from the streaming literature, and a careful pre-processing step to avoid a $\mathrm{poly}(d)$-blow up in the runtime. Given these samples, we can obtain the solution $\hat{x}$ after solving the regression problem on the sampled rows, and the fact that this gives a $(1+\epsilon)$ approximate solution will follow from Theorem 6 of [DDH+09].

Algorithm 8: Our ℓ_p Kronecker Product Regression Algorithm, $1 \leq p < 2$

$O(1)$-approximate ℓ_p Regression$(\{\mathbf{A}_i, n_i, d_i\}_{i\in[q]})$ //Theorem 132
$d \leftarrow \prod_{i=1}^{q} d_i$, $n \leftarrow \prod_{i=1}^{q} n_i$.
for $i = 1, \ldots, q$ **do**
  $s_i \leftarrow O(qd_i^2)$
  Generate sparse p-stable transform $\mathbf{S}_i \in \mathbb{R}^{s_i \times n}$ (def 10.1.2) //Lemma 10.1.3
  Take the QR factorization of $\mathbf{S}_i\mathbf{A}_i = \mathbf{Q}_i\mathbf{R}_i$ to obtain $\mathbf{R}_i \in \mathbb{R}^{d_i \times d_i}$ //Fact 10.1.5
  Let $\mathbf{Z} \in \mathbb{R}^{d\times\tau}$ be a dense p-stable transform for $\tau = \Theta(\log(n))$ //Definition 10.1.1
  for $j = 1, \ldots, n_i$ **do**
    Set
$$a_{i,j} \leftarrow \mathrm{median}_{\eta\in[\tau]} \left\{ \left(\frac{|(\mathbf{A}_i\mathbf{R}_i^{-1}\mathbf{Z})_{j,\eta}|}{\theta_p} \right)^p \right\}$$
    where θ_p is the median of $\mathcal{D}_p$.
Define a distribution $\mathcal{D} = \{q'_1, q'_1, \ldots, q'_n\}$ by $q'_{\sum_{i=1}^{q} j_i \prod_{l=1}^{j-1} n_l} = \prod_{i=1}^{q} a_{i,j_i}$.
Let $\mathbf{\Pi} \in \mathbb{R}^{n\times n}$ denote a diagonal sampling matrix, where $\mathbf{\Pi}_{i,i} = 1/q_i^{1/p}$ with probability $q_i = \min\{1, r_1 q'_i\}$ and 0 otherwise, where $r_1 = \Theta(d^3/\epsilon^2)$. [DDH+09]
Let $x' \in \mathbb{R}^d$ denote the solution of
$$\min_{x\in\mathbb{R}^d} \|\mathbf{\Pi}(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x - \mathbf{\Pi}b\|_p$$
Return x' //x' is an $O(1)$ approx: Lemma 10.2.10
$(1+\epsilon)$**-approximate ℓ_p Regression:** $x' \in \mathbb{R}^d$
Implicitly define $\rho = (\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x' - b \in \mathbb{R}^n$
Via Lemma 10.2.14, compute a diagonal sampling matrix $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ such that $\mathbf{\Sigma}_{i,i} = 1/\alpha_i^{1/p}$ with probability $\alpha_i = \min\{1, \max\{q_i, r_2|\rho_i|^p/\|\rho\|_p^p\}\}$ where $r_2 = \Theta(d^3/\epsilon^3)$.
Compute $\widehat{x} = \arg\min_{x\in\mathbb{R}^d} \|\mathbf{\Sigma}(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q) - \mathbf{\Sigma}b\|_p$ (via convex optimization methods, e.g., [BCLL18, AKPS19, LSZ19])
Return $\widehat{x}$

The ℓ_p Regression Algorithm

We now give a complete proof of Theorem 132. Our high level approach follows that of [DDH+09]. Namely, we first obtain a vector x' which is a $O(1)$ approximate solution to the

optimal, and then use the *residual error* $\rho \in \mathbb{R}^d$ of x' to refine x' to a $(1 \pm \epsilon)$ approximation $\widehat{x}$. The fact that x' is a constant factor approximation follows from our Lemma 10.2.10. Given x', by Lemma 10.2.14 we can efficiently compute the matrix $\mathbf{\Sigma}$ which samples from the coordinates of the residual error $\rho = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b$ in the desired runtime. The sampling lemma is the main technical lemma, and requires a careful multi-part sketching and sampling routine. Given this $\mathbf{\Sigma}$, the fact that $\hat{x}$ is a $(1+\epsilon)$ approximate solution follows directly from Theorem 6 of [DDH+09]. Our main theorem and its proof is stated below. The proof will utilize the lemmas and sampling algorithm developed in the secitons which follow.

Theorem 132 (Main result, ℓ_p $(1+\epsilon)$-approximate regression)**.** *Fix $1 \leq p < 2$. Then for any constant $q = O(1)$, given matrices $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_q$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, let $n = \prod_{i=1}^q n_i$, $d = \prod_{i=1}^q d_i$. Let $\hat{x} \in \mathbb{R}^d$ be the output of Algorithm 8. Then*

$$\|(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)\hat{x} - b\|_p \leq (1+\epsilon) \min_{x \in \mathbb{R}^n} \|(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x - b\|_p$$

holds with probability at least $1 - \delta$. In addition, our algorithm takes

$$\widetilde{O}\left(\left(\sum_{i=1}^q \operatorname{nnz}(\mathbf{A}_i) + \operatorname{nnz}(b) + \operatorname{poly}(d \log(1/\delta)/\epsilon)\right) \log(1/\delta)\right)$$

time to output $\widehat{x} \in \mathbb{R}^d$.

Proof. By Lemma 10.2.10, the output x' in line 16 of Algorithm 8 is an 8 approximation of the optimal solution, and x' is obtained in time $\widetilde{O}(\sum_{i=1}^q \operatorname{nnz}(\mathbf{A}_i) + (dq/\epsilon)^{O(1)})$. We then obtain the residual error $\rho = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b$ (implicitly). By Theorem 6 of [DDH+09], if we let $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ be a row sampling matrix where $\mathbf{\Sigma}_{i,i} = 1/\alpha_i^{1/p}$ with probability $\alpha_i = \min\{1, \max\{q_i, r_2 \frac{|\rho_i|^p}{\|\rho\|_p^p}\}$, where q_i is the row sampling probability used in the sketch $\mathbf{\Pi}$ from which x' was obtained, and $r_2 = O(d^3/\epsilon^2 \log(1/\epsilon))$, then the solution to $\min_x \|\mathbf{\Sigma}(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - \mathbf{\Sigma}b\|_p$ will be a $(1+\epsilon)$ approximately optimal solution. By Lemma 10.2.14, we can obtain such a matrix $\mathbf{\Sigma}$ in time $\widetilde{O}(\sum_{i=1}^q \operatorname{nnz}(\mathbf{A}_i) + q \operatorname{nnz}(b) + (d \log(n)/(\epsilon\delta)^{O(q^2)})$, which completes the proof of correctness. Finally, note that we can solve the sketched regression problem $\min_x \|\mathbf{\Sigma}(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - \mathbf{\Sigma}b\|_p$ which has $O((d \log(n)/\epsilon)^{O(q^2)}(1/\delta))$ constraints and d variables in time $O((d \log(n)/\epsilon)^{O(q^2)}(1/\delta))$ using linear programming for $p = 1$, or more generally interior point methods for convex programming for $p > 1$ (see [BCLL18, AKPS19, LSZ19] for

the recent development of ℓ_p solvers).

Now to boost the failure probability from a $O(1/\delta)$ to $\log(1/\delta)$ dependency, we do the following. We run the above algorithm with $\delta = 1/10$, so that our output $\hat{x} \in \mathbb{R}^d$ is a $(1+\epsilon)$ approximation with probability $9/10$, and we repeat this r times with $r = O(\log(1/\delta))$ time to obtain $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_r$,, and then we repeat another r times to obtain distinct sampling matrices $\mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_r$ (note that $\mathbf{\Sigma}_i$ is not the sampling matrix associated to $\hat{x}_i$ in this notation, and comes from a distinct repetition of the above algorithm). This blows up the overall runtime by $O(\log(1/\delta))$. Now for any vector $x \in \mathbb{R}^d$, define the random variable

$$\mathbf{X}_i = |\mathbf{\Sigma}_{i,i}(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)_{i,*}x - b_i|^p$$

Clearly $\mathbf{E}\left[\sum_i \mathbf{X}_i\right] = \|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - b\|_p^p$. Moreover, we can bound

$$\mathbf{E}\left[\sum_i \mathbf{X}_i^2\right] \lesssim \mathrm{poly}(d)\left(\mathbf{E}\left[\sum_i \mathbf{X}_i\right]\right)^2 / r_2$$

which can be seen from the proof of Lemma 9 in [DDH+09], which contains a computation. Setting $r_2 = \mathrm{poly}(d)$ large enough, by Chebyshev's we have that each $\mathbf{\Sigma}_i$ preserves the cost of a fixed vector x_j with probability $99/100$. so with probability $1-\delta$, after a union bound, for all $i \in [r]$ we have that

$$\mathrm{median}_j \|\mathbf{\Sigma}_j(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)\hat{x}_i - \mathbf{\Sigma}_j b\|_p = (1 \pm \epsilon)\|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)\hat{x}_i - b\|_p$$

Thus we now have $(1+\epsilon)$-error approximations of the cost of each $\hat{x}_i$, and we can output the $\hat{x}_i$ with minimal cost. By Chernoff bounds, at least one $\hat{x}_i$ will be a $(1+\epsilon)$ optimal solution, so by a union bound we obtain the desired result with probability $1-2\delta$ as needed.

□

We start by defining a tensor operation which will be useful for our analysis.

Definition 10.2.6 ($((\cdot,\ldots,\cdot),\cdot)$ operator for tensors and matrices)**.** *Given tensor* $\mathbf{A} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_q}$ *and matrices* $\mathbf{B}_i \in \mathbb{R}^{n_i \times d_i}$ *for* $i \in [q]$*, we define the tensor* $((\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_q), A) \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_q}$*:*

$$((\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_q), A)_{i_1,\ldots,i_q} = \sum_{i_1'=1}^{d_1}\sum_{i_2'=1}^{d_2}\cdots\sum_{i_q'=1}^{d_q}\mathbf{A}_{i_1',i_2',\ldots,i_q'}\prod_{\ell=1}^{q}(\mathbf{B}_\ell)_{i_\ell,i_\ell'}$$

Observe for the case of $q = 2$, we just have $((\mathbf{B}_1, \mathbf{B}_2), \mathbf{A}) = \mathbf{B}_1 A \mathbf{B}_2^\top \in \mathbb{R}^{n_1 \times n_2}$.

Using the above notation, we first prove a result about reshaping tensors.

Lemma 10.2.7 (Reshaping)**.** *Given matrices $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_q \in \mathbb{R}^{n_i \times d_i}$ and a tensor $\mathbf{B} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_q}$, let $n = \prod_{i=1}^q n_i$ and let $d = \prod_{i=1}^d d_i$. Let b denote the vectorization of $\mathbf{B}$. For any tensor $\mathbf{X} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_q}$, we have $\|((\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_q), \mathbf{X}) - \mathbf{B}\|_\xi$ is equal to $\|(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x - b\|_\xi$ where ξ is any entry-wise norm (such as an ℓ_p-norm) and x is the vectorization of X. See Definition 10.2.6 of the $((\cdot, \ldots, \cdot), \cdot)$ tensor operator.*

Observe, for the case of $q = 2$, this is equivalent to the statement that

$$\|\mathbf{A}_1 \mathbf{X} \mathbf{A}_2^\top - \mathbf{B}\|_\xi = \|(\mathbf{A}_1 \otimes \mathbf{A}_2)x - b\|_\xi$$

Proof. For the pair $x \in \mathbb{R}^d$, $\mathbf{X} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_q}$, the connection is the following: $\forall i_1 \in [d_1], \ldots, i_q \in [d_q]$,

$$x_{i_1 + \sum_{l=2}^q (i_l - 1) \cdot \prod_{t=1}^{l-1} d_t} = \mathbf{X}_{i_1, \cdots, i_q}.$$

Similarly, for $b \in \mathbb{R}^n$, $\mathbf{B} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_q}$, for any $j_1, \in [n_1], \ldots, j_q \in [n_q]$,

$$b_{j_1 + \sum_{l=2}^q (j_l - 1) \cdot \prod_{t=1}^{l-1} n_t} = \mathbf{B}_{j_1, j_2, \cdots, j_q}.$$

For simplicity, for any $(i_1, \ldots, i_q) \in [d_1] \times \cdots \times [d_q]$ and $(j_1, \ldots, j_q) \in [n_1] \times \cdots \times [n_q]$ we define $\vec{i} = i_1 + \sum_{l=2}^q (i_l - 1) \cdot \prod_{t=1}^{l-1} d_t$ and similarly $\vec{j} = j_1 + \sum_{l=2}^q (j_l - 1) \cdot \prod_{t=1}^{l-1} n_t$. Then we can simplify the above relation and write $x_{\vec{i}} = \mathbf{X}_{i_1, i_2, \cdots, i_q}$, and $b_{\vec{j}} = \mathbf{B}_{j_1, j_2, \cdots, j_q}$.

For a matrix $\mathbf{Z}$, let $\mathbf{Z}_{i,*}$ denote the i-th row of Z. We consider the $\vec{j}$-th entry of $(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x$,

$$\begin{aligned}
((\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)x)_{\vec{j}} &= \left\langle (\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)_{\vec{j},*} \cdot x \right\rangle \\
&= \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} \cdots \sum_{i_q=1}^{d_q} \left(\prod_{l=1}^q (\mathbf{A}_l)_{j_l, i_l} \right) \cdot x_{\vec{i}} \\
&= \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} \cdots \sum_{i_q=1}^{d_q} \left(\prod_{l=1}^q (\mathbf{A}_l)_{j_l, i_l} \right) \cdot \mathbf{X}_{i_1, i_2, \cdots, i_q} \\
&= ((\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_q), X)_{j_1, \ldots, j_q}.
\end{aligned}$$

Where the last equality is by Definition (10.2.6). Since we also have $b_{\vec{j}} = \mathbf{B}_{j_1,\ldots,j_q}$, this completes the proof of the Lemma. □

Sampling From an ℓ_p-Well-Conditioned Base

In this Section, we discuss the first half of Algorithm 8 which computes $x' \in \mathbb{R}^d$, which we will show is a $O(1)$-approximate solution to the optimal. First note that by Lemma 10.1.3 together with fact 10.1.5, we know that $\mathbf{A}_i\mathbf{R}_i^{-1}$ is an ℓ_p well conditioned basis for $\mathbf{A}_i$ (recall this means that $\mathbf{A}_i\mathbf{R}_i^{-1}$ is a (α, β, p) well conditioned basis for $\mathbf{A}$, and $\beta/\alpha = d_i^{O(1)}$) with probability $1 - O(1/q)$, and we can then union bound over this occurring for all $i \in [q]$. Given this, we now prove that $(\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \mathbf{A}_2\mathbf{R}_2^{-1} \otimes \cdots \otimes \mathbf{A}_q\mathbf{R}_q^{-1})$ is a well conditioned basis for $(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_q)$.

Lemma 10.2.8. *Let $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$ and $\mathbf{R}_i \in \mathbb{R}^{d_i \times d_i}$. Then if $\mathbf{A}_i\mathbf{R}_i^{-1}$ is a (α_i, β_i, p) well-conditioned basis for $\mathbf{A}_i$ for $i = 1, 2, \ldots, q$, we have for all $x \in \mathbb{R}^{d_1 \cdots d_q}$:*

$$\prod_{i=1}^{q} \alpha_i \|x\|_p \leq \|(\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \mathbf{A}_2\mathbf{R}_2^{-1} \otimes \cdots \otimes \mathbf{A}_q\mathbf{R}_q^{-1})x\|_p \leq \prod_{i=1}^{q} \beta_i \|x\|_p$$

Proof. We first consider the case of $q = 2$. We would like to prove

$$\alpha_1\alpha_2\|x\|_p \leq \|(\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \mathbf{A}_2\mathbf{R}_2^{-1})x\|_p \leq \beta_1\beta_2\|x\|_p,$$

First note, by the reshaping Lemma 10.2.7, this is equivalent to

$$\alpha_1\alpha_2\|\mathbf{X}\|_p \leq \|\mathbf{A}_1\mathbf{R}_1^{-1}\mathbf{X}(\mathbf{R}_2^{-1}\mathbf{A}_2)^\top\|_p \leq \beta_1\beta_2\|\mathbf{X}\|_p.$$

Where $\mathbf{X} \in \mathbb{R}^{d_1 \times d_2}$ is the tensorization of x. We first prove one direction. Let $\mathbf{U}_1 = \mathbf{A}_1\mathbf{R}_1^{-1}$ and $\mathbf{U}_2 = \mathbf{A}_2\mathbf{R}_2^{-1}$. We have

$$\begin{aligned}
\|\mathbf{U}_1\mathbf{X}\mathbf{U}_2^\top\|_p^p &= \sum_{i_2=1}^{n_2} \|\mathbf{U}_1(\mathbf{X}\mathbf{U}_2^\top)_{i_2}\|_p^p \\
&\leq \sum_{i_2=1}^{n_2} \beta_1^p \|(\mathbf{X}\mathbf{U}_2^\top)_{i_2}\|_p^p \\
&= \beta_1^p \|\mathbf{X}\mathbf{U}_2^\top\|_p^p \\
&\leq \beta_1^p\beta_2^p\|\mathbf{X}\|_p^p,
\end{aligned}$$

where the first step follows from rearranging, the second step follows from the well-conditioned

property of $\mathbf{U}_1$, the third step follows from rearranging again, the last step follows from the well-conditioned property of $\mathbf{U}_2$. Similarly, we have

$$\begin{aligned}\|\mathbf{U}_1\mathbf{X}\mathbf{U}_2^\top\|_p^p &= \sum_{i_2=1}^{n_2}\|\mathbf{U}_1(\mathbf{X}\mathbf{U}_2^\top)_{i_2}\|_p^p \\ &\geq \sum_{i_2=1}^{n_2}\alpha_1^p\|(\mathbf{X}\mathbf{U}_2^\top)_{i_2}\|_p^p \\ &= \alpha_1^p\|\mathbf{X}\mathbf{U}_2^\top\|_p^p \\ &\geq \alpha_1^p\alpha_2^p\|\mathbf{X}\|_p^p,\end{aligned}$$

where again the first step follows from rearranging, the second step follows from the well-conditioned property of $\mathbf{U}_1$, the third step follows from rearranging again, the last step follows from the well-conditioned property of $\mathbf{U}_2$.

In general, for arbitrary $q \geq 2$, similarly using our reshaping lemma, we have

$$\begin{aligned}\|(\otimes_{i=1}^q(\mathbf{A}_i\mathbf{R}_i^{-1}))x\|_p &\geq \prod_{i=1}^q\alpha_i\|x\|_p, \\ \|(\otimes_{i=1}^q(\mathbf{A}_i\mathbf{R}_i^{-1}))x\|_p &\leq \prod_{i=1}^q\beta_i\|x\|_p.\end{aligned}$$

□

Putting this together with fact 10.1.5, and noting $d = d_1 \cdots d_q$, we have

Corollary 10.2.9. *Let $\mathbf{A}_i\mathbf{R}_i^{-1}$be as in algorithm 8. Then we have for all $x \in \mathbb{R}^{d_1\cdots d_q}$:*

$$(1/d)^{O(1)}\|x\|_p \leq \|(\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \cdots \otimes \mathbf{A}_q\mathbf{R}_q^{-1})x\|_p \leq d^{O(1)}\|x\|_p,$$

In other words, $(\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \cdots \otimes \mathbf{A}_q\mathbf{R}_q^{-1})$ is a well conditioned ℓ_p basis for $(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)$

From this, we can obtain the following result.

Lemma 10.2.10. *Let $x' \in \mathbb{R}^d$ be the output of the $O(1)$-Approximate ℓ_p Regression Procedure in Algorithm 8. Then with probability $99/100$ we have*

$$\|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b\|_p \leq 8\min_x\|(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - b\|_p.$$

Moreover, the time required to compute x' is $\tilde{O}(\sum_{i=1}^q \operatorname{nnz}(\mathbf{A}_i) + \operatorname{poly}(dq/\epsilon))$.

Proof. By Theorem 6 of [DDH+09], if we let $\mathbf{\Pi}$ be a diagonal row sampling matrix such that $\mathbf{\Pi}_{i,i} = 1/q_i^{1/p}$ with probability $q_i \geq \min\{1, r_1 \frac{\|\mathbf{U}_{i,*}\|_p^p}{\|\mathbf{U}\|_p^p}\}$, where $\mathbf{U}$ is a ℓ_p well-conditioned basis for $(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)$ and $r_1 = O(d^3)$, then the solution x' to

$$\min_x \|\mathbf{\Pi}((\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - b\|$$

will be a 8-approximation. Note that we can solve the sketched regression problem $\min_x \|\mathbf{\Pi}((\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b\|$ which has $O(\text{poly}(d/\epsilon))$ constraints and d variables in time $\text{poly}(d/\epsilon)$ using linear programming for $p = 1$, or more generally interior point methods for convex programming for $p > 1$ (see [BCLL18, AKPS19, LSZ19] for the recent development of ℓ_p solvers).

Then by Corollary 10.2.9, we know that setting $\mathbf{U} = (\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \cdots \otimes \mathbf{A}_q\mathbf{R}_q^{-1})$ suffices, so now we must sample rows of $\mathbf{U}$. To do this, we must approximately compute the norms of the rows of $\mathbf{U}$. Here, we use the fact that $\|\cdot\|_p^p$ norm of a row of $(\mathbf{A}_1\mathbf{R}_1^{-1} \otimes \cdots \otimes \mathbf{A}_q\mathbf{R}_q^{-1})$ is the product of the row norms of the $\mathbf{A}_i\mathbf{R}_i^{-1}$ that correspond to that row. Thus it suffices to sample a row j_i from each of the $\mathbf{A}_i\mathbf{R}_i^{-1}$'s with probability at least $\min\{1, r_1\|(\mathbf{A}_i\mathbf{R}_i^{-1})_{j_i,*}\|_p^p/\|\mathbf{A}_i\mathbf{R}_i^{-1}\|_p^p\}$ for each $i \in [q]$.

To do this, we must estimate all the row norms $\|(\mathbf{A}_i\mathbf{R}_i^{-1})_{j_i,*}\|_p^p$ to $(1 \pm 1/10)$ error. This is done in steps $7 - 10$ of Algorithm 8, which uses dense p-stable sketches $Z \in \mathbb{R}^{d\times\tau}$, and computes $(\mathbf{A}_i\mathbf{R}_i^{-1}\mathbf{Z})$, where $\tau = \Theta(\log(n))$. Note that computing $\mathbf{R}_i^{-1}Z \in \mathbb{R}^{d\times\tau}$ requires $\tilde{O}(d^2)$. Once computed, $\mathbf{A}_i(\mathbf{R}_i^{-1}\mathbf{Z})$ can be computed in $\widetilde{O}(\text{nnz}(\mathbf{A}_i))$ time. We then take the median of the coordinates of $(\mathbf{A}_i\mathbf{R}_i^{-1}\mathbf{Z})$ (normalized by the median of the p-stable distribution $\mathcal{D}_p$, which can be efficiently approximated to $(1 \pm \epsilon)$ in $O(\text{poly}(1/\epsilon))$ time, see Appendix A.2 of [KNW10a] for details) as our estimates for the row norms. This is simply the Indyk median estimator [Ind06] given in Theorem 8, and gives a $(1 \pm 1/10)$ estimate $a_{i,j}$ of all the row norms $\|(\mathbf{A}_i\mathbf{R}_i^{-1})_{j,*}\|_p^p$ with probability $1 - 1/\text{poly}(n)$. Then it follows by Theorem 6 of [DDH+09] that x' is a 8-approximation of the optimal solution with probability $99/100$ (note that we amplified the probability by increasing the sketch sizes $\mathbf{S}_i$ by a constant factor), which completes the proof.

□

ℓ_p Sampling From the Residual of a $O(1)$-factor Approximation

By Lemma 10.2.10 in the prior section, we know that the x' first returned by the in algorithm 8 is a 8-approximation. We now demonstrate how we can use this $O(1)$ approximation to obtain a $(1 + \epsilon)$ approximation. The approach is again to sample rows of $(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)$. But instead

of sampling rows with the well-conditioned leverage scores q_i, we now sample the i-th row with probability $\alpha_i = \min\{1, \max\{q_i, r_2|\rho_i|^p/\|\rho\|_p^p\}\}$, where $\rho = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b \in \mathbb{R}^n$ is the *residual error* of the $O(1)$-approximation x'. Thus we must now determine how to sample quickly from the residuals $|\rho_i|^p/\|\rho\|_p^p$. Our sampling algorithm will need a tool originally developed in the streaming literature.

Count-sketch for heavy hitters with the Dyadic Trick. We now introduce a sketch $\mathbf{S}$ which finds the ℓ_2 heavy hitters in a vector x efficently. This sketch $\mathbf{S}$ is known as count-sketch for heavy hitters with the Dyadic Trick. To build $\mathbf{S}$ we first stack $\Theta(\log(n))$ copies of the *count sketch matrix* $\mathbf{S}^i \in \mathbb{R}^{k' \times n}$ [CW17]. The matrix $\mathbf{S}^i$ is precisely a single "repetition" of the sketch from Section 2.3.2, and is constructed as follows. $\mathbf{S}^i$ has exactly one non-zero entry per column, which is placed in a uniformly random row, and given the value 1 or -1 uniformly at random. For $\mathbf{S}^i$, let $h_i : [n] \to [k']$ be such that $h_i(t)$ is the row with the non-zero entry in the t-th column of $\mathbf{S}^i$, and let $g_i : [n] \to \{1, -1\}$ be such that the value of that non-zero entry is $g_i(t)$. Note that the h_i, g_i can be implemented as 4-wise independent hash functions.

Fix any $x \in \mathbb{R}^n$. Then given $\mathbf{S}^1 x, \mathbf{S}^2 x, \cdots, \mathbf{S}^{\Theta(\log(n))} x$, we can estimate the value of any coordinate x_j by

$$\text{median}_{i \in \Theta \log(n)} \{g_i(j)(\mathbf{S}^i x)_{h_i(j)}\}$$

By Theorem 9, this gives an estimate of x_j with additive error $\Theta(1/\sqrt{k'})\|x\|_2$ with probability $1 - 1/\text{poly}(n)$ for all $j \in [n]$. However, naively, to find the heaviest coordinates in x, that is all coordinates x_j with $|x_j| \geq \Theta(1/\sqrt{k'})\|x\|_2$, one would need to query $O(n)$ estimates. This is where the Dyadic trick comes in [CM05]. We repeat the above process $\Theta(\log(n))$ times, with matrices $\mathbf{S}^{(i,j)}$, for $i, j \in \Theta(\log(n))$. Importantly, however, in $\mathbf{S}^{(i,j)}$, for all $t, t' \in [n]$ such that the first j most significant bits in their binary identity representation are the same, we set $h_{(i,j)}(t) = h_{(i,j)}(t')$, effectively collapsing these identities to one. To find a heavy item, we can then query the values of the *two* identities from $\mathbf{S}^{(1,1)}, \mathbf{S}^{(2,1)}, \cdots, \mathbf{S}^{(\Theta(\log(n)),1)}$, and recurse into all the portions which have size at least $\Theta(1/\sqrt{k'})\|x\|_2$. It is easy to see that we recurse into at most $O(k')$ such pieces in each of the $\Theta(\log(n))$ levels, and it takes $O(\log(n))$ time to query a single estimate, from which the desired runtime of $O(k' \log^2(n))$ is obtained. For a further improvement on size k of the overall sketched required to quickly compute Q, see [LNNT16]. We summarize this construction below in definition 10.2.11.

Definition 10.2.11 (Count-sketch for heavy hitters with Dyadic Trick [CCFC02b, LNNT16])**.** *There is a randomized sketch $\mathbf{S} \in \mathbb{R}^{k \times n}$ with $k = O(\log^2(n)/\epsilon^2)$ such that, for a fixed vector $x \in \mathbb{R}^n$, given $\mathbf{S}x \in \mathbb{R}^k$, one can compute a set $Q \subset [n]$ with $|Q| = O(1/\epsilon^2)$ such that*

$\{i \in [n] \mid |x_i| \geq \epsilon \|x\|_2\} \subseteq Q$ *with probability* $1 - 1/\mathrm{poly}(n)$. *Moreover,* $\mathbf{S}x$ *can be computed in* $O(\log^2(n)\,\mathrm{nnz}(x))$ *time. Given* $\mathbf{S}x$*, the set* Q *can be computed in time* $O(k)$.

We begin with some notation. For a vector $y \in \mathbb{R}^n$, where $n = n_1 \cdots n_q$, one can index any entry of y_i via $\vec{i} = (i_1, i_2, \cdots, i_q) \in [n_1] \times \cdots \times [n_q]$ via $i = i_1 + \sum_{j=2}^{q} (i_j - 1) \prod_{l=1}^{i_j - 1} n_l$. It will useful to index into such a vector y interchangably via a vector $y_{\vec{i}}$ and an index y_j with $j \in [n]$. For any set of subsets $T_i \subset [n_i]$, we can define $y_{T_1 \times \cdots T_q} \in \mathbb{R}^n$ as y restricted to the $\vec{i} \in T_1 \times \cdots \times T_q$. Here, by restricted, we mean the coordinates in y that are not in this set are set equal to 0. Similarly, for a $y \in \mathbb{R}^{n_i}$ and $\mathbf{S} \subset [n_i]$, we can define y_S as y restricted to the coordinates in $\mathbf{S}$. Note that in Algorithm 9, $\mathbb{I}_n$ denotes the $n \times n$ identity matrix for any integer n. We first prove a proposition on the behavior of Kronecker products of p-stable vectors, which we will need in our analysis.

Proposition 10.2.12. *Let* $\mathbf{Z}_1, \mathbf{Z}_2, \cdots, \mathbf{Z}_q$ *be independent vectors with entries drawn i.i.d. from the* p*-stable distribution, with* $\mathbf{Z}_i \in \mathbb{R}^{n_i}$*. Now fix any* $i \in [q]$*, and any* $x \in \mathbb{R}^n$*, where* $n = n_1 n_2 \cdots n_q$*. Let* $e_j \in \mathbb{R}^{n_i}$ *be the* j*-th standard basis column vector for any* $j \in [n_i]$*. Let* $\Gamma(i,j) = [n_1] \times [n_2] \times \cdots \times [n_{i-1}] \times \{j\} \times [n_{i+1}] \times \cdots \times [n_q]$*. Define the random variable*

$$\mathcal{X}_{i,j}(x) = |(\mathbf{Z}_1 \otimes \mathbf{Z}_1 \otimes \cdots \otimes \mathbf{Z}_{i-1} \otimes e_j^\top \otimes \mathbf{Z}_{i+1} \otimes \cdots \otimes \mathbf{Z}_q)x|^p.$$

Then for any $\lambda > 1$*, with probability at least* $1 - O(q/\lambda)$ *we have*

$$\|x_{\Gamma(i,j)}\|_p^p / \lambda^q \leq \mathcal{X}_{i,j}(x) \leq (\lambda \log(n))^q \|x_{\Gamma(i,j)}\|_p^p$$

Proof. First observe that we can reshape $y = x_\Gamma \in \mathbb{R}^m$ where $m = n/n_i$, and re-write this random variable as $\mathcal{X}_{i,j}(x) = |(\mathbf{Z}_1 \otimes \mathbf{Z}_2 \otimes \cdots \otimes \mathbf{Z}_{q-1})y|^p$. By reshaping Lemma 10.2.7, we can write this as $|(\mathbf{Z}_1 \otimes \mathbf{Z}_2 \otimes \cdots \otimes \mathbf{Z}_{q-2})\mathbf{Y}\mathbf{Z}_{q-1}^\top|^p$, where $\mathbf{Y} \in \mathbb{R}^{m/n_{q-1} \times n_{q-1}}$. We first prove a claim. In the remainder of the proof, for a matrix $\mathbf{A}$, let $\|\mathbf{A}\|_p^p = \sum_{i,j} |\mathbf{A}_{i,j}|^p$.

Claim 10.2.13. *Let* Z *be any* p*-stable vector and* $\mathbf{X}$ *a matrix. Then for any* $\lambda > 1$*, with probability* $1 - O(1/\lambda)$*, we have*

$$\lambda^{-1} \|\mathbf{X}\|_p^p \leq \|\mathbf{X}Z\|_p^p \leq \log(n)\lambda \|\mathbf{X}\|_p^p.$$

Proof. By p-stability, each entry of $|(\mathbf{X}`Z)_i|^p$ is distributed as $|z_i|^p \|\mathbf{X}_{i,*}\|_p^p$, where z_i is again p-stable (but the $z_i's$ are not independent). By Fact 2.2.8 (see also Chapter 1.5 of [Nol]), p-stables have tails that decay at the rate $\Theta(1/x^p)$ thus $\mathbf{Pr}\left[|z_i|^p > x\right] = O(1/x)$ for any $x > 0$. We can

condition on the fact that $z_i < \lambda \cdot n^{10}$ for all i, which occurs with probability at least $1 - n^{-9}/\lambda$ by a union bound. Conditioned on this, we have $\mathbf{E}[|z_i|^p] = O(\log(n))$ (this can be seen by integrating over the truncated tail $O(1/x)$), and the upper bound then follows from a application of Markov's inequality.

For the lower bound Let Y_i be an indicator random variable indicating the event that $|z_i|^p < 2/\lambda$. Now p-stables are anti-concentrated, namely, their pdf is upper bounded by a constant everywhere. It follows that $\mathbf{Pr}[Y_i] < c/\lambda$ for some constant c. By Markov's inequality

$$\mathbf{Pr}\left[\sum_i Y_i \|\mathbf{X}_{i,*}\|_p^p > \|X\|_p^p/2\right] < O(1/\lambda)$$

Conditioned on this, the remaining $\|\mathbf{X}\|_p^p/2$ of the ℓ_p mass shrinks by less than a $2/\lambda$ factor, thus $\|\mathbf{X}Z\|_p^p > (\|\mathbf{X}\|_p^p/2)(2/\lambda) = \|\mathbf{X}\|_p^p/\lambda$ as needed. □

By the above claim, we have $\|\mathbf{Y}\|_p/\lambda^{1/p} \leq \|\mathbf{Y}\mathbf{Z}_{q-1}^\top\|_p \leq (\log(n)\lambda)^{1/p}\|\mathbf{Y}\|_p$ with probability $1 - O(1/\lambda)$. Given this, we have $\mathcal{X}_{i,j}(x) = |(\mathbf{Z}_1 \otimes \mathbf{Z}_2 \otimes \cdots \otimes \mathbf{Z}_{q-2})y'|^p$, where $\|\mathbf{Y}\|_p/\lambda^{1/p} \leq \|y'\|_p \leq (\log(n)\lambda)^{1/p}\|\mathbf{Y}\|_p$. We can inductively apply the above argument, each time getting a blow up of $(\log(n)\lambda)^{1/p}$ in the upper bound and $(1/\lambda)^p$ in the lower bound, and a failure probability of $(1/\lambda)$. Union bounding over all q steps of the induction, the proposition follows.

□

Lemma 10.2.14. *Fix any $r_2 \geq 1$, and suppose that $x' = \min_x \|\mathbf{\Pi}(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x - \mathbf{\Pi}b\|_p$ and $\mathbf{\Pi} \in \mathbb{R}^{n \times n}$ is a row sampling matrix such that $\mathbf{\Pi}_{i,i} = 1/q_i^{1/p}$ with probability q_i. Define the residual error $\rho = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b \in \mathbb{R}^n$. Then Algorithm 9, with probability $1 - \delta$, succeeds in outputting a row sampling matrix $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ such that $\mathbf{\Sigma}_{i,i} = 1/\alpha_i^{1/p}$ with probability $\alpha_i = \min\{1, \max\{q_i, r_3|\rho_i|^p/\|\rho\|_p^p\}\}$ for some $r_3 \geq r_2$, and otherwise $\mathbf{\Sigma}_{i,i} = 0$. The algorithm runs in time*

$$\widetilde{O}\left(\sum_{i=1}^q \operatorname{nnz}(\mathbf{A}_i) + q\operatorname{nnz}(b) + (r_2\log(n)/\delta)^{O(q^2)}\right).$$

Proof. The algorithm is given formally in Figure 9. We analyze the runtime and correctness here.

Proof of Correctness. The approach of the sampling algorithm is as follows. Recall that we can index into the coordinates of $\rho \in \mathbb{R}^n$ via $\vec{a} = (a_1, \ldots, a_q)$ where $a_i \in [n_i]$. We build the

Algorithm 9: Algorithm to ℓ_p sample $\Theta(r_2)$ entires of $\rho = (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)x' - b$

$\rho, r_2\ T \leftarrow \emptyset$ //T is the sample set to return

$r_3 \leftarrow \Theta(r_2 \log^{q^2}(n)/\delta)$

Generate i.i.d. p-stable vectors $Z^{1,j}, Z^{2,j}, \ldots, Z^{q,j} \in \mathbb{R}^n$ for $j \in [\tau]$ where $\tau = \Theta(\log n)$

Pre-compute and store $Z^{i,j}\mathbf{A}_i \in \mathbb{R}^{1\times d_i}$ for all $i \in [q]$ and $j \in [\tau]$

For each $i \in [q]$, generate an independent copy $\mathbf{S}^i \in \mathbb{R}^{k\times n_i}$ of count-sketch for heavy hitters from Definition 10.2.11, where $k = O(\log^2(n) r_3^{O(1)})$.

for $t = 1, 2, \ldots, r_3$ **do**

  $s = (s_1, \ldots, s_q) \leftarrow (\emptyset, \ldots, \emptyset)$ //next sample to return

  $w^j \leftarrow \left((\mathbb{I}_{n_1}) \otimes (\bigotimes_{k=2}^q Z^{k,j})\rho\right) \in \mathbb{R}^{n_1}$ //$\mathbb{I}_n \in \mathbb{R}^{n\times n}$ is the identity matrix

  Define $w \in \mathbb{R}^{n_1}$ by $w_l = \text{median}_{j\in[\tau]}\{|w_l^j|\}$ for $l \in [n_1]$

  Sample $j^* \in [n_1]$ from the distribution $\left(\frac{|w_1|^p}{\|w\|_p^p}, \frac{|w_2|^p}{\|w\|_p^p}, \ldots, \frac{|w_{n_1}|^p}{\|w\|_p^p}\right)$, and set $s_1 \leftarrow j^*$

  for $i = 2, \ldots, q$ **do**

    for $j \in [\tau]$ **do**

      Write $e_{a_k}^\top \in \mathbb{R}^{1\times n_k}$ as the standard basis vector

      $v_i^j \leftarrow \mathbf{S}^i\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (\mathbb{I}_{n_i}) \otimes (\bigotimes_{k=i+1}^q Z^{k,j})\rho\right) \in \mathbb{R}^k$

      Compute heavy hitters $H_{i,j} \subset [n_i]$ from v_i^j //Definition 10.2.11

      $\beta_i^j \leftarrow \left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (\bigotimes_{k=i}^q Z^{k,j})\rho\right) \in \mathbb{R}$

    Define $\beta_i \in \mathbb{R}^{k'}$ by $\beta_i = \text{median}_{j\in[\tau]}\{|\beta_i^j|^p\}$, and $H_i = \cup_{j=1}^\tau H_{i,j}$

    $\gamma_i \leftarrow \text{median}_{j\in[\tau]}\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes Z_{[n_i]\setminus H_i}^{i,j} \otimes (\bigotimes_{k=i+1}^q Z^{k,j})\rho\right) \in \mathbb{R}$

    if *with probability* $1 - \gamma_i/\beta_i$ **then**

      Draw $\xi \in H_i$ with probability:

$$\lambda_\xi = \frac{\text{median}_{j\in\tau}\left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (e_\xi^\top) \otimes (\bigotimes_{k=i+1}^q Z^{k,j})\rho\right)\right|^p}{\sum_{\xi'\in H_i}\text{median}_{j\in\tau}\left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (e_{\xi'}^\top) \otimes (\bigotimes_{k=i+1}^q Z^{k,j})\rho\right)\right|^p}$$

      Set $s_i \leftarrow \xi$ //else, s_i is sampled as a heavy hitter

    else

      //s_i was not sampled as a heavy hitter

      Randomly partition $[n_i]$ into $\Omega_1^i, \Omega_2^i, \ldots, \Omega_\eta^i$ with $\eta = \Theta(r_3^2)$.

      Sample $t \sim [\eta]$ uniformly at random.

      for $j \in \Omega_t \setminus H_i$ **do**

        Set $\theta_j = \text{median}_{l\in[\tau]}\left(|(\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (e_j^\top) \otimes (\bigotimes_{k=i+1}^q Z^{k,l})\rho|^p\right)$

      Sample $s_i \leftarrow j^*$ from the distribution $\left\{\frac{\theta_j}{\sum_{j'\in\Omega_t\setminus H_i}\theta_{j'}}\right\}_{j\in\Omega_t\setminus H_i}$

  $T \leftarrow S \cup s$ where $s = (s_1, \ldots, s_q)$

Return sample set T

coordinates of $\vec{a}$ one by one. To sample a $\vec{a} \in \prod_{i=1}^{q}[n_i]$, we can first sample $a_1 \in [n_1]$ from the distribution $\mathbf{Pr}[a_1 = j] = \sum_{\vec{u}:u_1=j} |\rho_{\vec{u}}|^p / (\sum_{\vec{u}} |\rho_{\vec{u}}|^p)$. Once we fix a_1, we can sample a_2 from the conditional distribution distribution $\mathbf{Pr}[a_2 = j] = \sum_{\vec{u}:u_2=j,u_1=a_1} |\rho_{\vec{u}}|^p / (\sum_{\vec{u}:u_1=a_1} |\rho_{\vec{u}}|^p)$, and so on. For notation, given a vector $\vec{a} = (a_1, \ldots, a_{i-1})$, let $\Delta(\vec{a}) = \{\vec{u} \in [n_1] \times \cdots \times [n_q] \mid a_j = y_j \text{ for all } j = 1, 2, \ldots, i-1\}$. Then in general, when we have sampled $\vec{a} = (a_1, \ldots, a_{i-1})$ for some $i \leq q$, we need to sample $a_i \leftarrow j \in [n_k]$ with probability

$$\mathbf{Pr}[a_i = j] = \sum_{\vec{u} \in \Delta(\vec{a}):u_i=j} |\rho_{\vec{u}}|^p / \left(\sum_{\vec{u} \in \Delta(\vec{a})} |\rho_{\vec{u}}|^p \right).$$

We repeat this process to obtain the desired samples. Note that to sample efficiently, we will have to compute these aforementioned sampling probabilities approximately. Because of the error in approximating, instead of returning r_2 samples, we over-sample and return $r_3 = \Theta(r_2 \log^{q^2}(n))$ samples.

The first step is of the algorithm is to generate the p-stable vectors $Z^{i,j} \in \mathbb{R}^{n_i}$ for $i \in [q]$ and $j = 1, 2, \ldots, \Theta(\log(n))$. We can pre-compute and store $Z^{i,j}\mathbf{A}_i$ for $i \in [q]$, which takes $\tilde{O}(\sum_{i=1}^{q} \operatorname{nnz}(\mathbf{A}_i))$ time. We set $w^j \leftarrow \left((\mathbb{I}_{n_1}) \otimes (\bigotimes_{k=2}^{q} Z^{k,j})\rho\right) \in \mathbb{R}^{n_1}$ and define $w \in \mathbb{R}^{n_1}$ by $w_l = \operatorname{median}_{j \in [\tau]}\{|w_l^j|\}$ for $l \in [n_1]$. Observe that w_l^j is an estimate of $\sum_{\vec{u}:u_1=l} |\rho_{\vec{u}}|^p$. By Proposition 10.2.12, it is a $(c\log(n))^q$ approximation with probability at least $3/4$ for some constant c. Taking the median of $\Theta(\log(n))$ repetitions, we have that

$$c^{-q} \cdot \sum_{\vec{u}:u_1=l} |\rho_{\vec{u}}|^p \leq |w_l|^p \leq (c\log(n))^q \cdot \sum_{\vec{u}:u_1=l} |\rho_{\vec{u}}|^p$$

with probability $1 - 1/\operatorname{poly}(n)$, and we can then union bound over all such estimates every conducted over the course of the algorithm. We call the above estimate $|w_l|^p$ a $O((c\log(n))^q)$-error estimate of $\sum_{\vec{u}:u_1=l} |\rho_{\vec{u}}|^p$. Given this, we can correctly and independently sample the first coordinate of each of the $\Theta(r_3)$ samples. We now describe how to sample the i-th coordinate. So in general, suppose we have sampled $(a_1, ..., a_{i-1})$ so far, and we need to now sample $a_i \in [n_i]$ conditioned on $(a_1, ..., a_{i-1})$. We first consider

$$W^{i,k} = \left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (\mathbb{I}_{n_i}) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j})\rho \right) \in \mathbb{R}^{n_i}$$

Note that the j-th coordinate $W_j^{i,k}$ for $W^{i,k}$ is an estimate of $\sum_{\vec{u} \in \Delta(\vec{a}):u_i=j} |\rho_{\vec{u}}|^p$. Again by By Proposition 10.2.12, with probability $1 - 1/\operatorname{poly}(n)$, we will have $|W_j^{i,k}|^p$ is a $O((c\log(n))^q)$-

error estimate of $\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p$ or at least one $k\in[\tau]$. Our goal will now be to find all $j\in[n_i]$ such that $\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p \geq \Theta((c\log(n))^q/r_3^8)\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p$. We call such a j a *heavy hitter*.

Let $Q_i\subset[n_i]$ be the set of heavy hitters. To find all the heavy hitters, we use the count-sketch for heavy hitters with the Dyadic trick of definition 10.2.11. We construct this count-sketch of def 10.2.11 $\mathbf{S}^i\in\mathbb{R}^{k'\times n_i}$ where $k'=O(\log^2(n)r_3^{16})$. We then compute $\mathbf{S}^iW^{i,k}$, for $k=1,2,\ldots,\tau$, and obtain the set of heavy hitters $h\in H_{i,k}\subset[n_i]$ which satisfy $|W^{i,k}_j|^p\geq\Theta(1/r_3^8)\|W^{i,k}\|_p^p$. By the above discussion, we know that for each $j\in Q_i$, we will have $|W^{i,k}_j|^p\geq\Theta(1/r_3^{16})\|W^{i,k}\|_p^p$ for at least one $k\in[\tau]$ with high probability. Thus $H_i=\cup_{k=1}^{\tau}H_{i,k}\supseteq Q_i$.

We now will decide to either sample a heavy hitter $\xi\in H_i$, or a non-heavy hitter $\xi\in[n_i]\setminus H_i$. By Proposition 10.2.12, we can compute a $O((c\log(n))^{-q})$-error estimate

$$\beta_i=\text{median}_{j\in[\tau]}\left|\left((\bigotimes_{k=1}^{i-1}e_{a_k}^\top)\otimes(\bigotimes_{k=i}^{q}Z^{k,j})\rho\right)\right|^p$$

of $\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p$, meaning:

$$O(c^{-q})\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p\leq\beta_i\leq O((c\log n)^q)\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p.$$

Again, by Proposition 10.2.12, we can compute a $O((c\log(n))^{-q})$-error estimate

$$\gamma_i=\text{median}_{j\in[\tau]}\left((\bigotimes_{k=1}^{i-1}e_{a_k}^\top)\otimes Z^{i,j}_{[n_i]\setminus H_i}\otimes(\bigotimes_{k=i+1}^{q}Z^{k,j})\rho\right)$$

of $\sum_{h\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p$. It follows that

$$O(c^{-2q})\frac{\sum_{h\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p}{\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p}\leq\frac{\gamma_i}{\beta_i}\leq O((c\log n)^{2q})\frac{\sum_{h\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p}{\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p}$$

In other words, γ_i/β_i is a $O((c\log(n))^{2q})$-error approximation of the true probability that we should sample a non-heavy item. Thus with probability $1-\gamma_i/\beta_i$, we choose to sample a heavy item.

To sample a heavy item, for each $\xi\in H_i$, by Proposition 10.2.12, we can compute an

$O((c\log(n))^{-q})$-error estimate

$$\text{median}_{j\in\tau}\left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top)\otimes(e_\xi^\top)\otimes(\bigotimes_{k=i+1}^{q} Z^{k,j})\rho\right)\right|^p$$

of $\sum_{\vec{u}\in\Delta(\vec{a}):u_i=\xi}|\rho_{\vec{u}}|^p$, meaning

$$\begin{aligned}O(c^{-q})\sum_{\vec{u}\in\Delta(\vec{a}):u_i=\xi}|\rho_{\vec{u}}|^p &\leq \text{median}_{j\in\tau}\left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top)\otimes(e_\xi^\top)\otimes(\bigotimes_{k=i+1}^{q} Z^{k,j})\rho\right)\right|^p\\ &\leq O((c\log n)^q)\sum_{\vec{u}\in\Delta(\vec{a}):u_i=\xi}|\rho_{\vec{u}}|^p\end{aligned}$$

Thus we can choose to sample a heavy item $\xi\in H_i$ from the distribution given by

$$\mathbf{Pr}\left[\text{sample } a_i\leftarrow\xi\right]=\frac{\text{median}_{j\in\tau}\left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top)\otimes(e_\xi^\top)\otimes(\bigotimes_{k=i+1}^{q} Z^{k,j})\rho\right)\right|^p}{\sum_{\xi'\in H_i}\text{median}_{j\in\tau}\left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top)\otimes(e_{\xi'}^\top)\otimes(\bigotimes_{k=i+1}^{q} Z^{k,j})\rho\right)\right|^p}$$

Which gives a $O((c\log(n))^{2q})$-error approximation to the correct sampling probability for a heavy item.

In the second case, with probability γ_i/β_i, we choose to not sample a heavy item. In this case, we must now sample a item from $[n_i]\setminus H_i$. To do this, we partition $[n_i]$ randomly into $\Omega_1,\ldots,\Omega_\eta$ for $\eta=1/r_3^2$. Now there are two cases. First suppose that we have

$$\frac{\sum_{j\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p}{\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p}\leq\Theta(1/r_3^3)$$

Now recall that γ_i/β_i was a $O((c\log(n))^{2q})$-error estimate of the ratio on the left hand side of the above equation, and γ_i/β_i was the probability with which we choose to sample a non-heavy hitter. Since we only repeat the sampling process r_3 times, the probability that we ever sample a non-heavy item in this case is at most $\Theta(q(c\log(n))^{2q}/r_3^2)<\Theta(q/r_3)$, taken over all possible repetitions of this sampling in the algorithm. Thus we can safely ignore this case, and condition on the fact that we never sample a non-heavy item in this case.

Otherwise,

$$\sum_{j\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p > \Theta(1/r_3^3)\sum_{\vec{u}\in\Delta(\vec{a})}|\rho_{\vec{u}}|^p,$$

and it follows that

$$\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j'}|\rho_{\vec{u}}|^p \le r_3^{-5}\cdot\sum_{j\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p$$

for all $j'\in[n_i]\setminus H_i$, since we removed all $\Theta(1/r_3^8)$ heavy hitters from $[n_i]$ originally. Thus by Chernoff bounds, with high probability we have that

$$\sum_{j\in\Omega_i\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p = \Theta\left(\eta^{-1}\cdot\sum_{j\in[n_i]\setminus H_i}\sum_{\vec{u}\in\Delta(\vec{a}):u_i=j}|\rho_{\vec{u}}|^p\right),$$

which we can union bound over all repetitions.

Given this, by choosing $t\sim[\eta]$ uniformly at random, and then choosing $j\in\Omega_t\setminus H_i$ with probability proportional to its mass in $\Omega_t\setminus H_i$, we get a $\Theta(1)$ approximation of the true sampling probability. Since we do not know its exact mass, we instead sample from the distribution

$$\left\{\frac{\theta_j}{\sum_{j'\in\Omega_t\setminus H_i}\theta_{j'}}\right\}_{j\in\Omega_t\setminus H_i},$$

where

$$\theta_j = \text{median}_{l\in[\tau]}\left(\left|(\bigotimes_{k=1}^{i-1}e_{a_k}^\top)\otimes(e_j^\top)\otimes(\bigotimes_{k=i+1}^{q}Z^{k,l})\rho\right|^p\right)$$

Again by Proposition 10.2.12, this gives a $O((c\log(n))^{2q})$-error approximation to the correct sampling probability. Note that at each step of sampling a coorindate of $\vec{a}$ we obtained at most $O((c\log(n))^{2q})$-error in the sampling probability. Thus, by oversampling by a $O((c\log(n))^{2q^2})$ factor, we can obtain the desired sampling probabilities. This completes the proof of correctness. Note that to improve the failure probability to $1-\delta$, we can simply scale r_3 by a factor of $1/\delta$.

Proof of Runtime. We now analyze the runtime. At every step $i=1,2,\ldots,q$ of the sampling, we compute $v_i^j\leftarrow\mathbf{S}^i\left((\bigotimes_{k=1}^{i-1}e_{a_k}^\top)\otimes(\mathbb{I}_{n_i})\otimes(\bigotimes_{k=i+1}^{q}Z^{k,j})\rho\right)\in\mathbb{R}^{n_i}$ for $j=1,2,\ldots\Theta(\log(n))$.

This is equal to

$$\mathbf{S}^i \left((\bigotimes_{k=1}^{i-1} (\mathbf{A}_k)_{a_k,*}) \otimes (\mathbf{A}_i) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j} \mathbf{A}_k) x' - (\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (\mathbb{I}_{n_i}) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j}) b \right)$$

We first consider the term inside of the parenthesis (excluding $\mathbf{S}^i$). Note that the term $(\bigotimes_{k=i+1}^{q} Z^{k,j} \mathbf{A}_k)$ was already pre-computed, and is a vector of length at most d, this this requires a total of $\tilde{O}(\sum_{i=1}^{q} \operatorname{nnz}(\mathbf{A}_i) + d)$ time. Note that these same values are used for every sample. Given this pre-computation, we can rearrage the first term to write $(\bigotimes_{k=1}^{i-1} (\mathbf{A}_k)_{a_k,*}) \otimes (\mathbf{A}_i) \mathbf{X}' (\bigotimes_{k=i+1}^{q} Z^{k,j} \mathbf{A}_k)^\top$ where $\mathbf{X}'$ is a matrix formed from x' so that x' is the vectorization of $\mathbf{X}'$ (this is done via reshaping Lemma 10.2.7). The term $y = \mathbf{X}' (\bigotimes_{k=i+1}^{q} Z^{k,j} \mathbf{A}_k)^\top$ can now be computed in $O(d)$ time, and then we reshape again to write this as $(\bigotimes_{k=1}^{i-1} (\mathbf{A}_k)_{a_k,*}) Y \mathbf{A}_i^\top$ where Y again is a matrix formed from y. Observe that $\zeta = \operatorname{vec}(\bigotimes_{k=1}^{i-1} (\mathbf{A}_k)_{a_k,*} \mathbf{Y}) \in \mathbb{R}^{d_i}$ can be computed in time $O(qd)$, since each entry is a dot product of a column $\mathbf{Y}_{*,j} \in \mathbb{R}^{d_1 \cdot d_2 \cdots d_{i-1}}$ of Y with the $d_1 \cdot d_2 \cdots d_{i-1}$ dimensional vector $\bigotimes_{k=1}^{i-1} (\mathbf{A}_k)_{a_k,*}$, which can be formed in $O(d_1 \cdot d_2 \cdots d_{i-1} q)$ time, and there are a total of d_i columns of Y.

Given this, The first entire term $\mathbf{S}^i (\bigotimes_{k=1}^{i-1} (\mathbf{A}_k)_{a_k,*}) \otimes (\mathbf{A}_i) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j} \mathbf{A}_k) x'$ can be rewritten as $\mathbf{S}^i \mathbf{A}_i \zeta$, where $\zeta = \zeta_{\vec{a}} \in \mathbb{R}^{d_i}$ can be computed in $O(dq)$ time for each sample $\vec{a}$. Thus if we recompute the value $\mathbf{S}_i \mathbf{A}_i \in \mathbb{R}^{k \times n}$, where $k = \widetilde{O}(r_3^{16})$, which can be done in time $\widetilde{O}(\operatorname{nnz} \mathbf{A}_i)$, then every time we are sampling the i-th coordinate of some $\vec{a}$, computing the value of $\mathbf{S}^i \mathbf{A}_i \zeta_{\vec{a}}$ can be done in time $O(k d_i^2) = r_3^{O(1)}$.

We now consider the second term. We perform similar trick, reshaping $b \in \mathbb{R}^n$ into $\mathbf{B} \in \mathbb{R}^{(n_1 \cdots n_i) \times (n_i \cdots n_q)}$ and writing this term as $((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (\mathbb{I}_{n_i})) \mathbf{B} (\bigotimes_{k=i+1}^{q} Z^{k,j})^\top$ and computing $b' = \mathbf{B} (\bigotimes_{k=i+1}^{q} Z^{k,j})^\top \in \mathbb{R}^{(n_1 \cdots n_i)}$ in $\operatorname{nnz}(\mathbf{B}) = \operatorname{nnz}(b)$ time. Let $\mathbf{B}' \in \mathbb{R}^{(n_1 \cdots n_{i-1}) \times n_i}$ be such that $\operatorname{vec}(\mathbf{B}') = b'$, and we reshape again to obtain $(\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \mathbf{B}' (\mathbb{I}_{n_i}) = (\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \mathbf{B}'$ Now note that so far, the value $\mathbf{B}'$ did not depend on the sample $\vec{a}$ at all. Thus for each $i = 1, 2, \ldots, q$, $\mathbf{B}'$ (which depends only on i) can be pre-computed in $\operatorname{nnz}(b)$ time. Given this, the value $(\bigotimes_{k=1}^{i-1} e_{a_k}^\top) B'$ is just a row $\mathbf{B}'_{(a_1,\ldots,a_k),*}$ of $\mathbf{B}'$ (or a column of $(\mathbf{B}')^\top$). We first claim that $\operatorname{nnz}(\mathbf{B}') \leq \operatorname{nnz}(b) = \operatorname{nnz}(\mathbf{B})$. To see this, note that each entry of $\mathbf{B}'$ is a dot product $\mathbf{B}_{j,*} (\bigotimes_{k=i+1}^{q} Z^{k,j})^\top$ for some row $\mathbf{B}_{j,*}$ of $\mathbf{B}$, and moreover there is a bijection between these dot products and entries of $\mathbf{B}'$. Thus for every non-zero entry of $\mathbf{B}'$, there must be a unique non-zero row (and thus non-zero entry) of $\mathbf{B}$. This gives a bijection from the support of $\mathbf{B}'$ to the support of $\mathbf{B}$ (and thus b) which completes the claim. Since $\mathbf{S}^i (\mathbf{B}'_{(a_1,\ldots,a_k),*})^\top$ can be computed in $\tilde{O}(\operatorname{nnz}(\mathbf{B}'_{(a_1,\ldots,a_k),*}))$ time, it follows that $\mathbf{S}^i (\mathbf{B}'_{(a_1,\ldots,a_k),*})^\top$ can be computed for all rows $(\mathbf{B}'_{(a_1,\ldots,a_k),*})$ of $\mathbf{B}$ in $\widetilde{O}(\operatorname{nnz}(b))$ time. Given this precomputation, we note

that $(\mathbb{I}_{n_i}) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j})b$ is just $\mathbf{S}^i(\mathbf{B}'_{(a_1,\ldots,a_k),*})^\top$ for some $(a_1,\ldots,a_k)$, which has already been pre-computed, and thus requires no addition time per sample. Thus, given a total of $\widetilde{O}(\sum_{i=1}^{q} \operatorname{nnz}(\mathbf{A}_i) + q\operatorname{nnz}(b) + r_3^{O(1)})$ pre-processing time, for each sample we can compute v_i^j for all $i \in [q]$ and $j \in [\tau]$ in $\widetilde{O}(r_3^{O(1)})$ time, and thus $\widetilde{O}(r_3^{O(1)})$ time over all r_3 samples.

Given this, the procedure to compute the heavy hitters $H_{i,j}$ takes $\widetilde{O}(r_3^{16})$ time by Definition 10.2.11 for each sample and $i \in [q], j \in [\tau]$. By a identical pre-computation and rearrangement argument as above, each β_i^j (and thus β_i) can be computed in $\widetilde{O}(r_3^{O(1)})$ time per sample after pre-computation. Now note that γ_i is simply equal to

$$\operatorname{median}_{j\in[\tau]} \left(\beta_i^j - (\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (Z_{H_i}^{k,j}) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j})\rho \right).$$

Since $Z_{H_i}^{k,j}$ is sparse, the above can similar be computed in $O(d|H_i|) = \widetilde{O}(r_3^{O(1)})$ time per sample after pre-computation. To see this, note that the b term of $(\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (Z_{H_i}^{k,j}) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j})\rho$ can be written as $(\bigotimes_{k=1}^{i-1} e_{a_k}^\top)\mathbf{B}'''(Z_{H_i}^{k,j})^\top$, where $\mathbf{B}''' \in \mathbb{R}^{n_1\cdots n_{i-1}\times n_i}$ is a matrix that has already been pre-computed and does not depend on the given sample. Then this quantity is just the dot product of a row of $\mathbf{B}'''$ with $(Z_{H_i}^{k,j})^\top$, but since $(Z_{H_i}^{k,j})$ is $|H_i|$-sparse, so the claim for the b term follows. For the $(\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_q)$ term, just as we demonstrated in the discussion of computing v_i^j, note that this can be written as $(\bigotimes_{k=1}^{i-1}(\mathbf{A}_k)_{a_k,*})Y((\mathbf{A}_i)_{H_i,*})^\top$ for some matrix $Y \in \mathbb{R}^{d_1\cdots d_i \times d_{i-1}}$ that has already been precomputed. Since $(\mathbf{A}_i)_{H_i,*}$ only has $O(|H_i|)$ non-zero rows, this whole product can be computed in time $O(d|H_i|)$ as needed.

Similarly, we can compute the sampling probabilities

$$\mathbf{Pr}\left[\text{sample } a_i \leftarrow j\right] = \frac{\operatorname{median}_{j\in\tau} \left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (e_\xi^\top) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j})\rho\right)\right|^p}{\sum_{\xi'\in H_i} \operatorname{median}_{j\in\tau} \left|\left((\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (e_{\xi'}^\top) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,j})\rho\right)\right|^p}$$

for each every item $\zeta \in H_i$ in $\widetilde{O}(r_3^{O(1)})$ time after pre-computation, and note $|H_i| = \widetilde{O}(r_3^{O(1)})$ by definition 10.2.11. Thus the total time to sample a heavy hitter in a given coordinate $i \in [q]$ for each sample $\widetilde{O}(r_3^{O(1)})$ per sample, for an overall time of $\widetilde{O}(qr_3^{O(1)})$ over all samples and $i \in [q]$.

Finally, we consider the runtime for sampling a non-heavy item. Note that $|\Omega_t| = O(n_i/\eta)$ with high probability for all $t \in [\eta]$ by chernoff bounds. Computing each

$$\theta_j = \operatorname{median}_{l\in[\tau]} \left(\left| (\bigotimes_{k=1}^{i-1} e_{a_k}^\top) \otimes (e_j^\top) \otimes (\bigotimes_{k=i+1}^{q} Z^{k,l})\rho \right|^p \right)$$

takes $O(qd)$ time after pre-computation, and so we spend a total of $O(qdn_i/\eta)$ time sampling an item from $\Omega_t \setminus H_i$. Since we only ever sample a total of r_3 samples, and $\eta = \Theta(r_3^2)$, the total time for sampling non-heavy hitters over the course of the algorithm in coordinate i is $o(n_i) = o(\text{nnz}(\mathbf{A}_i))$ as needed, which completes the proof of the runtime.

Computing the Sampling Probabilities α_i The above arguments demonstrate how to sample efficiently from the desired distribution. We now must describe how the sampling probabilities α_i can be computed. First note, for each sample that is sampled in the above way, at every step we compute exactly the probability with which we decide to sample a coordinate to that sample. Thus we know exactly the probability that we choose a sample, and moreover we can compute each q_i in $O(d)$ time as in Lemma 10.2.10. Thus we can compute the maximum of q_i and this probability exactly. For each item sampled as a result of the leverage score sampling probabilities q_i as in Lemma 10.2.10, we can also compute the probability that this item was sampled in the above procedure, by using the same sketching vectors $Z^{i,k}$ and count-sketches $\mathbf{S}^i$. This completes the proof of the Lemma.

□

10.3 All-Pairs Regression

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$, let $\bar{\mathbf{A}} \in \mathbb{R}^{n^2 \times d}$ be the matrix such that $\bar{\mathbf{A}}_{i+(j-1)n,*} = \mathbf{A}_{i,*} - \mathbf{A}_{j,*}$, and let $\bar{b} \in \mathbb{R}^{n^2}$ be defined by $\bar{b}_{i+(j-1)n} = b_i - b_j$. Thus, $\bar{\mathbf{A}}$ consists of all pairwise differences of rows of $\mathbf{A}$, and $\bar{b}$ consists of all pairwise differences of rows of b. The ℓ_p all pairs regression problem on the inputs $\mathbf{A}, b$ is to solve $\min_{x \in \mathbb{R}^d} \|\bar{\mathbf{A}}x - \bar{b}\|_p$.

First note that this problem has a close connection to Kronecker product regression. Namely, the matrix $\bar{\mathbf{A}}$ can be written $\bar{\mathbf{A}} = \mathbf{A} \otimes \mathbf{1}^n - \mathbf{1}^n \otimes \mathbf{A}$, where $\mathbf{1}^n \in \mathbb{R}^n$ is the all 1's vector. Similarly, $\bar{b} = b \otimes \mathbf{1}^n - \mathbf{1}^n \otimes b$. For simplicity, we now drop the superscript and write $\mathbf{1} = \mathbf{1}^n$.

Our algorithm is given formally in Figure 10. We generate sparse p-stable sketches $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{R}^{k \times n}$ (Definition 10.1.2), where $k = (d/(\epsilon\delta))^{O(1)}$. We compute $\mathbf{M} = (\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F}) = \mathbf{S}_1\mathbf{F} \otimes \mathbf{S}_2\mathbf{1} - \mathbf{S}_1\mathbf{1} \otimes \mathbf{S}_2\mathbf{F}$, where $\mathbf{F} = [\mathbf{A}, b]$. We then take the $\mathbf{QR}$ decomposition $\mathbf{M} = \mathbf{QR}$. Finally, we sample rows of $(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{R}^{-1}$ with probability proportional to their ℓ_p norms. This is done by an involved sampling procedure described in Lemma 10.3.1, which is similar to the sampling procedure used in the proof of Theorem 132. Finally, we solve the regression problem $\min_x \|\mathbf{\Pi}(\bar{\mathbf{A}}x - \bar{b})\|_p$, where $\mathbf{\Pi}$ is the diagonal row-sampling matrix constructed by the

sampling procedure. We summarize the guarantee of our algorithm in the following theorem.

Theorem 133. *Given* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *and* $b \in \mathbb{R}^n$*, for* $p \in [1,2]$*, let* $\bar{\mathbf{A}} = A \otimes \mathbf{1} - \mathbf{1} \otimes A \in \mathbb{R}^{n^2\times d}$ *and* $\bar{b} = b \otimes \mathbf{1} - \mathbf{1} \otimes b \in \mathbb{R}^{n^2}$*. Then there is an algorithm for that outputs* $\hat{x} \in \mathbb{R}^d$ *such that with probability* $1-\delta$ *we have* $\|\bar{\mathbf{A}}\hat{x} - \bar{b}\|_p \leq (1+\epsilon)\min_{x\in\mathbb{R}^d} \|\bar{\mathbf{A}}x - \bar{b}\|_p$*. The running time is* $\widetilde{O}(\mathrm{nnz}(\mathbf{A}) + (d/(\epsilon\delta))^{O(1)})$.

Algorithm 10: Our All-Pairs Regression Algorithm

All-Pairs Regression:$\mathbf{A}, b$
$\mathbf{F} = [\mathbf{A}, b] \in \mathbb{R}^{n\times d+1}$. $r \leftarrow \mathrm{poly}(d/\epsilon)$
Generate $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{R}^{k\times n}$ sparse p-stable transforms for $k = \mathrm{poly}(d/(\epsilon\delta))$.
Sketch $(\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})$.
Compute QR decomposition: $(\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F}) = \mathbf{QR}$.
Let $\mathbf{M} = (\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{R}^{-1}$, and $\sigma_i = \|\mathbf{M}_{i,*}\|_p^p / \|M\|_p^p$.
Obtain row sampling diagonal matrix $\mathbf{\Pi} \in \mathbb{R}^{n\times n}$ such that $\mathbf{\Pi}_{i,i} = 1/\widetilde{q}_i^{\,1/p}$ independently with probability $q_i \geq \min\{1, r\sigma_i\}$, where $\widetilde{q}_i = (1 \pm \epsilon^2)q_i$.
//Lemma 10.3.1 **Return** $\hat{x}$, where $\hat{x} = \arg\min_{x\in\mathbb{R}^d} \|\mathbf{\Pi}(\bar{\mathbf{A}}x - \bar{b})\|_p$.

The theorem crucially utilizes our fast ℓ_p sampling routine, which is described in Figure 15 in the supplementary. A full discussion and proof of the lemma can be found in the supplementary material 10.3.2.

Lemma 10.3.1 (Fast ℓ_p sampling)**.** *Given* $R \in \mathbb{R}^{d+1\times d+1}$ *and* $\mathbf{F} = [\mathbf{A}, b] \in \mathbb{R}^{n\times d+1}$*, there is an algorithm that, with probability* $1 - n^{-c}$ *for any constant* c*, produces a diagonal matrix* $\mathbf{\Pi} \in \mathbb{R}^{n^2\times n^2}$ *such that* $\mathbf{\Pi}_{i,i} = 1/\widetilde{q}_i^{\,1/p}$ *with probability* $q_i \geq \min\{1, r\|\mathbf{M}_{i,*}\|_p^p / \|M\|_p^p\}$ *and* $\mathbf{\Pi}_{i,i} = 0$ *otherwise, where* $r = \mathrm{poly}(d/\epsilon)$ *and* $\mathbf{M} = (\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{R}^{-1}$*, and* $\tilde{q}_i = (1 \pm \epsilon^2)q_i$ *for all* $i \in [n^2]$*. The total time required is* $\widetilde{O}(\mathrm{nnz}\, A + \mathrm{poly}(d/\epsilon))$.

10.3.1 Analysis of All-Pairs Regression Algorithm

In this section, we prove the correctness of our all-pairs regression algorithm 10. Our main theorem, Theorem 133, relies crucially on the sample routine developed in Section 10.3.2. We first prove the theorem which utilizes this routine, and defer the description and proof of the routine to Section 10.3.2.

Recall first the high level description of our algorithm (given formally in Figure 10). We pick $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{R}^{k\times n}$and S are sparse p-stable sketches. We then compute $\mathbf{M} = (\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F}) = \mathbf{S}_1\mathbf{F} \otimes \mathbf{S}_2\mathbf{1} - \mathbf{S}_1\mathbf{1} \otimes \mathbf{S}_2\mathbf{F}$, where $\mathbf{F} = [\mathbf{A}, b]$. We then take the $\mathbf{QR}$ decomposition

$\mathbf{M} = \mathbf{QR}$. Finally, we sample rows of $(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{R}^{-1}$ with probability proportional to their ℓ_p norms. This is done by the sampling procedure described in Section 10.3.2. Finally, we solve the regression problem $\min_x \|\mathbf{\Pi}(\bar{\mathbf{A}}x - \bar{b})\|_p$, where $\mathbf{\Pi}$ is the diagonal row-sampling matrix constructed by the sampling procedure.

We begin by demonstrating that $\mathbf{S}_1 \otimes \mathbf{S}_2$ is a $\mathrm{poly}(d)$ distortion embedding for the column span of $[\bar{\mathbf{A}}, \bar{b}]$.

Lemma 10.3.2. *Let $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{R}^{k \times n}$ be sparse p-stable transforms, where $k = \mathrm{poly}(d/(\epsilon\delta))$. Then for all $x \in \mathbb{R}^{d+1}$, with probability $1 - \delta$ we have*

$$1/O(d^4 \log^4 d) \cdot \|[\bar{\mathbf{A}}, \bar{b}]x\|_p \leq \|(\mathbf{S}_1 \otimes \mathbf{S}_2)[\bar{\mathbf{A}}, \bar{b}]x\|_p \leq O(d^2 \log^2 d) \cdot \|[\bar{\mathbf{A}}, \bar{b}]x\|_p.$$

Proof. Let $\mathbf{F} = [\mathbf{A}, b]$. Then a basis for the columns of $[\bar{\mathbf{A}}, \bar{b}]$ is given by $\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F}$. We first condition on both $\mathbf{S}_1, \mathbf{S}_2$ being a low-distortion embedding for the $d + 2$ dimensional column-span of $[\mathbf{F}, \mathbf{1}]$. Note that this holds with large constant probability by 10.1.3.

So for any $x \in \mathbb{R}^{d+1}$, we first show the upper bound

$$\begin{aligned}
\|(\mathbf{S_1} \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})x\|_p &= \|(\mathbf{S}_1 F \otimes \mathbf{S}_2\mathbf{1})x - (\mathbf{S}_1\mathbf{1} \otimes \mathbf{S}_2 F)\|_p \\
&= \|\mathbf{S}_1 F x \mathbf{1}^\top \mathbf{S}_2^\top - \mathbf{S}_1 \mathbf{1} x^\top \mathbf{F}^\top \mathbf{S}_2^\top\|_p \\
&= \|\mathbf{S}_1(F x \mathbf{1}^\top - \mathbf{1} x^\top \mathbf{F}^\top)\mathbf{S}_2^\top\|_p \\
&\leq O(d \log d) \cdot \|(\mathbf{F} x \mathbf{1}^\top - \mathbf{1} x^\top \mathbf{F}^\top)\mathbf{S}_2^\top\|_p \\
&\leq O(d^2 \log^2 d) \cdot \|\mathbf{F} x \mathbf{1}^\top - \mathbf{1} x^\top \mathbf{F}^\top\|_p \\
&= O(d^2 \log^2 d) \cdot \|(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})x\|_p,
\end{aligned}$$

where the first equality follows by properties of the Kronecker product [VL00], the second by reshaping Lemma 10.2.7. The first inequality follows from the fact that each column of $(Fx\mathbf{1}^\top - \mathbf{1}x^\top\mathbf{F}^\top)\mathbf{S}_2^\top$ is a vector in the column span of $[\mathbf{F}, \mathbf{1}]$, and then using that $\mathbf{S}_1$ is a low distortion embedding. The second inequality follows from the fact that each row of $(\mathbf{F}x\mathbf{1}^\top - \mathbf{1}x^\top\mathbf{F}^\top)$ is a vector in the *column span* of $[\mathbf{F}, \mathbf{1}]$, and similarly using that $\mathbf{S}_2$ is a low distortion embedding. The final inequality follows from reshaping. Using a similar sequence of inequalities, we get the matching lower bound as desired. □

We now prove our main theorem.

Theorem 133 *Given $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$, for $p \in [1,2]$ there is an algorithm for the All-Pairs Regression problem that outputs $\hat{x} \in \mathbb{R}^d$ such that with probability $1-\delta$ we have*

$$\|\bar{\mathbf{A}}\hat{x} - \bar{b}\|_p \leq (1+\epsilon) \min_{x \in \mathbb{R}^d} \|\bar{\mathbf{A}}x - \bar{b}\|_p$$

Where $\bar{\mathbf{A}} = A \otimes \mathbf{1} - \mathbf{1} \otimes A \in \mathbb{R}^{n^2 \times d}$ and $\bar{b} = b \otimes \mathbf{1} - \mathbf{1} \otimes b \in \mathbb{R}^{n^2}$. For $p < 2$, the running time is $\widetilde{O}(nd + (d/(\epsilon\delta))^{O(1)})$, and for $p = 2$ the running time is $O(\mathrm{nnz}(\mathbf{A}) + (d/(\epsilon\delta))^{O(1)})$.

Proof. We first consider the case of $p = 2$. Here, we can use the fact that the TENSORSKETCH random matirx $S \in \mathbb{R}^{k \times n}$ is a subspace embedding for the column span of $[\bar{\mathbf{A}}, \bar{b}]$ when $k = \Theta(d/\epsilon^2)$ [DSSW18], meaning that $\|\mathbf{S}[\bar{\mathbf{A}}, \bar{b}]\|_2 = (1 \pm \epsilon)\|[\bar{\mathbf{A}}, \bar{b}]x\|_2$ for all $x \in \mathbb{R}^{d+1}$ with probability $9/10$. Moreover, $\mathbf{S}\bar{\mathbf{A}}$ and $\mathbf{S}\bar{b}$ can be computed in $O(\mathrm{nnz}(\mathbf{A}) + \mathrm{nnz}(b)) = O(\mathrm{nnz}(\mathbf{A}))$ by [DSSW18] since they are the difference of Kronecker products. As a result, we can simply solve the regression problem $\hat{x} = \arg\min_x \|\mathbf{S}\bar{\mathbf{A}}x - \mathbf{S}\bar{b}\|_2$ in $\mathrm{poly}(kd)$ time to obtain the desired $\hat{x}$.

For $p < 2$, we use the algorithm in Figure 10, where the crucial leverage score sampling procedure to obtain $\mathbf{\Pi}$ in step 7 of Figure 10 is described in Lemma 10.3.1. Our high level approach follows the general ℓ_p sub-space embedding approach of [DDH+09]. Namely, we first compute a low-distortion embedding $(\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})$. By Lemma 10.3.2, using sparse-p stable transformations $\mathbf{S}_1, \mathbf{S}_2$, we obtain the desired $\mathrm{poly}(d)$ distortion embedding into $\mathbb{R}^{k^2}$, where $k = \mathrm{poly}(d/\epsilon)$. Note that computing $(\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})$ can be done in $O(\mathrm{nnz}(\mathbf{A}) + \mathrm{nnz}(b) + n)$ time using the fact that $(\mathbf{S}_1 \otimes \mathbf{S}_2)(\mathbf{F} \otimes \mathbf{1}) = \mathbf{S}_1 F \otimes \mathbf{S}_2 \mathbf{1}$. As shown in [DDH+09], it follows that $\mathbf{M} = (\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{R}^{-1}$ is an ℓ_p well-conditioned basis for the column span of $(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})$ (see definition 10.1.4). Then by Theorem 5 of [DDH+09], if we let $\hat{\mathbf{\Pi}}$ be the diagonal row sampling matrix such that $\hat{\mathbf{\Pi}}_{i,i} = 1/q_i^{1/p}$ for each i with probability $q_i \geq \min\{1, r\|\mathbf{M}_{i,*}\|_p^p / \|M\|_p^p\}$ (and $\hat{\mathbf{\Pi}}_{i,i} = 0$ otherwise) for $r = \mathrm{poly}(d \log(1/\delta)/\epsilon)$, then with probability $1-\delta$ we have

$$\|\hat{\mathbf{\Pi}}(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})x\|_p = (1 \pm \epsilon)\|(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})x\|_p$$

for all $x \in \mathbb{R}^{d+1}$. First assume that we had such a matrix.

Since $(\bar{\mathbf{A}}x - \bar{b})$ is in the column span of $(\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})$ for any $x \in \mathbb{R}^{d+1}$, it follows that $\|\hat{\mathbf{\Pi}}(\bar{\mathbf{A}}x - \bar{b})\|_p = (1 \pm \epsilon)\|(\bar{\mathbf{A}}x - \bar{b})\|_p$ for all $x \in \mathbb{R}^d$, which completes the proof of correctness. By Lemma 10.3.1, we can obtain a row sampling matrix $\mathbf{\Pi}$ in time $\widetilde{O}(nd + \mathrm{poly}(d/\epsilon))$, except that the entries of $\mathbf{\Pi}$ are instead equal to either 0 or $1/\widetilde{q}_i^{1/p}$ where $\widetilde{q}_i = (1 \pm \epsilon^2)q_i$. Now let $\hat{\mathbf{\Pi}}$

be the idealized row sampling matrices from above, with entries either 0 or $1/q_i^{1/p}$ as needed for Theorem 5 of [DDH+09]. Note that for any matrix $\mathbf{Z}$ each row of $\hat{\mathbf{\Pi}}\mathbf{Z}x$ is equal to $\mathbf{\Pi}\mathbf{Z}x$ times some constant $1-\epsilon^2 < c < 1+\epsilon^2$. It follows that $\|\mathbf{\Pi}(\bar{\mathbf{A}}x - \bar{b})\|_p = (1 \pm \epsilon^2)\|\hat{\mathbf{\Pi}}(\bar{\mathbf{A}}x - \bar{b})\|_p$ for all $x \in \mathbb{R}^d$, and thus the objective function is changed by at most a $(1 \pm \epsilon^2)$ term, which is simply handled by a constant factor rescaling of ϵ.

Finally, we can solve the sketched regression problem $\|\mathbf{\Pi}(\bar{\mathbf{A}}x - \bar{b})\|_p$ which has $\mathrm{poly}(d/\epsilon)$ constraints and d variables in time $\mathrm{poly}(d/\epsilon)$ using linear programming for $p = 1$, or more generally interior point methods for convex programming for $p > 1$ (see [BCLL18, AKPS19, LSZ19] for the recent development of ℓ_p solver. Finally, the failure probability bound holds by union bounding over all the aforementioned results, and noting that the lowest probability event was the even that $\mathbf{S}_1 \otimes \mathbf{S}_2$ was a low distortion embedding via Lemma 10.3.2. This completes the proof of the theorem.

□

10.3.2 Proof of Fast Sampling Lemma 10.3.1

We now provide a full proof of the main technical lemma of Section 10.3. The sampling algorithm is given formally in Algorithm 15. The following proof of Lemma 10.3.1 analyzes each step in the process, demonstrating both correctness and the desired runtime bounds.

Lemma 10.3.1 *Given $\mathbf{R} \in \mathbb{R}^{(d+1)\times(d+1)}$ and $\mathbf{F} = [\mathbf{A}, b] \in \mathbb{R}^{n\times(d+1)}$, there is an algorithm that, with probability $1-\delta$ for any $\delta > n^{-c}$ for any constant c, produces a diagonal matrix $\mathbf{\Pi} \in \mathbb{R}^{n^2\times n^2}$ such that $\mathbf{\Pi}_{i,i} = 1/\tilde{q}_i^{1/p}$ with probability $q_i \geq \min\{1, r\|\mathbf{M}_{i,*}\|_p^p/\|\mathbf{M}\|_p^p\}$ and $\mathbf{\Pi}_{i,i} = 0$ otherwise, where $r = \mathrm{poly}(d/\epsilon)$ and $\mathbf{M} = (\mathbf{F}\otimes\mathbf{1} - \mathbf{1}\otimes\mathbf{F})\mathbf{R}^{-1}$, and $\tilde{q}_i = (1\pm\epsilon^2)q_i$ for all $i \in [n^2]$. The total time required is $\widetilde{O}(\mathrm{nnz}\, A + \mathrm{poly}(d/\epsilon))$.*

Proof. Our proof proceeds in several steps. We analyze the runtime concurrently with out analysis of correctness.

Reducing the number of Columns of $\mathbf{R}^{-1}$. We begin by generating a matrix $\mathbf{G} \in \mathbb{R}^{(d+1)\times\xi}$ of i.i.d. $\mathcal{N}(0, 1/\sqrt{\xi})$ Gaussian random variables. We then compute $Y \leftarrow \mathbf{R}^{-1}G$ in $\widetilde{O}(d^2)$ time. We first claim that it suffices to instead ℓ_p sample rows of $\mathbf{C} = (\mathbf{F}\otimes\mathbf{1} - \mathbf{1}\otimes\mathbf{F})Y = MG$. Note that each entry $|\mathbf{C}_{i,j}|^p$ is distributed as $g^p\|\mathbf{M}_{i,*}\|_2^p$ where $\mathbf{G}$ $\mathcal{N}(0, 1/\sqrt{\xi})$ Gaussian, which holds by the 2-stability of Gaussian random variables. Note that $\mathbf{E}[|g|^p] = \Theta(1/\xi)$, so

Algorithm 11: Algorithm to ℓ_p sample $\Theta(r)$ rows of $\mathbf{M} = (\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{R}^{-1}$

ℓ_p **sample:** $\mathbf{F} = [\mathbf{A}, b] \in \mathbb{R}^{n \times d}, \mathbf{R}^{-1} \in \mathbb{R}^{d+1 \times d+1}, r$

Generate a matrix $\mathbf{G} \in \mathbb{R}^{d+1 \times \xi}$ of i.i.d. $\mathcal{N}(0, 1/\sqrt{\xi})$ Gaussian random variables, with $\xi = \Theta(\log(n))$. Set $\mathbf{Y} \leftarrow \mathbf{R}^{-1}\mathbf{G} \in \mathbb{R}^{d+1 \times \xi}$, and $\mathbf{C} \leftarrow (\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})\mathbf{Y}$

Reshape i-th column $\mathbf{C}_{*,i}$ into $(\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}(\mathbf{Y}_{*,i})^\top\mathbf{F}^\top) \in \mathbb{R}^{n \times n}$

Generate $Z \in \mathbb{R}^{t \times n}$ i.i.d. p-stable for $t = \Theta(\log(n))$ //Definition 10.1.1

For all $(i, l) \in [\xi] \times [n]$, set

$$\sigma_{i,l} \leftarrow \mathrm{median}_{\tau \in [t]} \left(\frac{\left| (Z(\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}(\mathbf{Y}_{*,i})^\top\mathbf{F}^\top)_{\tau,l} \right|^p}{(\mathrm{median}(\mathcal{D}_p))^p} \right)$$

Set $W^{(i,l)} \leftarrow (\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}\mathbf{Y}_{*,i}^\top\mathbf{F}^\top)_{*,l} = \mathbf{F}\mathbf{Y}_{*,i} - \mathbf{1}(\mathbf{F}\mathbf{Y})_{l,i} \in \mathbb{R}^n$

for $j = 1, \ldots, \Theta(r)$ **do**

  Sample (i, l) from distribution $\sigma_{i,l} / \left(\sum_{i',l'} \sigma_{i',l'} \right)$

$T \leftarrow$ multi-set of samples (i, l)

Generate $\mathbf{S}_0 \in \mathbb{R}^{k \times n}$ $S \in \mathbb{R}^{k' \times n}$ count-sketches for heavy hitters with $k = r^{O(1)}, k' = k^{O(1)}$ and Generate $u_1, \ldots, u_n$ i.i.d. exponential variables. //Definition 10.2.11

$\mathbf{D} \leftarrow \mathtt{Diag}(1/u_1^{1/p}, \ldots, 1/u_n^{1/p}) \in \mathbb{R}^{n \times n}$.

for *each sample* $(i, l) \in T$ **do**

  Compute $\mathbf{S}_0 W^{(i,l)}$ and obtain set of heavy hitters $Q_0^{(i,l)} \subset [n]$. Then compute $W_j^{(i,l)}$ exactly for all $j \in Q_0^{(i,l)}$, to obtain true heavy hitters $H^{(i,l)}$. Next, compute the value

$$\alpha_{i,l} \leftarrow \mathrm{median}_{\tau \in [t]} \left(\frac{\left| Z_{\tau,*} W^{(i,l)} - \sum_{\zeta \in H^{(i,l)}} Z_{\tau,\zeta} W_\zeta^{(i,l)} \right|^p}{(\mathrm{median}(\mathcal{D}_p))^p} \right)$$

  if *With prob* $1 - \alpha_{(i,l)}/\sigma_{(i,l)}$, *sample a heavy item* $j^* \leftarrow j$ **then**

    Sample a heavy item $j^* \leftarrow j$ from the distribution $|W_j^{(i,l)}|^p / \sum_{j \in H_{(i,l)}} |W_j^{(i,l)}|^p$.

    Return The row $((l-1)n + j^*)$ //Note that $\mathbf{C}_{(l-1)n+j^*,*}$ contains $W_{j^*}^{(i,l)}$

  else

    Randomly partition $[n]$ into $\Omega_1, \Omega_2, \ldots, \Omega_\eta$ with $\eta = \Theta(r^4/\epsilon^4)$, and sample $t \sim [\eta]$ uniformly at random.

    Compute $\mathbf{S}(\mathbf{D}W^{(i,l)})_{\Omega_t \setminus H^{(i,l)}}$, and set $Q^{(i,l)} \subset \Omega_t \setminus H^{(i,l)}$ of heavy hitters.

    $j^* \leftarrow \arg\max_{j \in Q^{(i,l)}} (\mathbf{D}W^{(i,l)})_j$

    Return The row $((l-1)n + j^*)$ //Note that $\mathbf{C}_{(l-1)n+j^*,*}$ contains $W_{j^*}^{(i,l)}$

$\mathbf{E}\left[\|\mathbf{C}_{i,*}\|_p^p\right] = \|\mathbf{M}_{i,*}\|_2^p$, and by sub-exponential concentration (see Chapter 2 of [Wai19]), we have that $\|\mathbf{C}_{i,*}\|_p^p = (1 \pm 1/10)\|\mathbf{M}_{i,*}\|_2^p$ with probability $1 - 1/\mathrm{poly}(n)$, and we can union bound over this holding for all $i \in [n^2]$. By relationships between the p norms, we have $\|\mathbf{M}_{i,*}\|_p^p/d < \|\mathbf{M}_{i,*}\|_2^p < \|\mathbf{M}_{i,*}\|_p^p$, thus this changes the overall sampling probabilities by a factor between $\Theta(1/d^2)$ and $\Theta(d^2)$. Thus, we can safely oversample by this factor (absorbing it into the value of r) to compensate for this change in sampling probabilities.

Sampling a row from $\mathbf{C}$. To sample a row from $\mathbf{C}$, the approach will be to sample an entry $\mathbf{C}_{i,j}$ of $\mathbf{C}$ with probability proportional to $\|\mathbf{C}_{i,j}\|_p^p/\|\mathbf{C}\|_p^p$. For every (i,j) sampled, we sample the entire i-th row of j, so that the j-th row is indeed sampled with probability proportional to its norm. Thus, it suffices to sample entries of $\mathbf{C}$ such that each $\mathbf{C}_{i,j}$ is chosen with probability at least $\min\{1, r\|\mathbf{C}_{i,j}\|_p^p/\|\mathbf{C}\|_p^p\}$. First note that the i-th column of $\mathbf{C} = (\mathbf{F} \otimes \mathbf{1} - \mathbf{1} \otimes \mathbf{F})Y$ can be rearranged into a $n \times n$ matrix via Lemma 10.2.7, given by $(\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}\mathbf{Y}_{*,i}^\top\mathbf{F}^\top)$. To ℓ_p sample a coordinate from $\mathbf{C}$, it suffices to first ℓ_p sample a column of one of the above matrices, and then ℓ_p sample an entry from that column.

To do this, we first compute $\mathbf{F}\mathbf{Y} \in \mathbb{R}^{n \times \xi}$, which can be done in time $\widetilde{O}(\mathrm{nnz}(\mathbf{A}))$ because Y only has $\xi = \Theta(\log(n))$ columns. We then compute $Z(\mathbf{F}Y_{*,i}\mathbf{1}^\top - \mathbf{1}Y_{*,i}^\top\mathbf{F}^\top) \in \mathbb{R}^{1 \times n}$ for all $i \in [d]$, where $Z \in \mathbb{R}^{1 \times n}$ is a fixed vector of i.i.d. p-stable random variables. Once $\mathbf{F}\mathbf{Y}$ has been computed, for each $i \in [\xi]$ it takes $O(n)$ time to compute this n-dimensional vector, thus the total time required to compute all ξ vectors is $\widetilde{O}(n)$. We repeat this process $t = O(\log(n))$ times with different p-stable vectors $Z^1, \ldots, Z^\top$, and take the median of each coordinate of $Z^j(\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}\mathbf{Y}_{*,i}^\top\mathbf{F}^\top) \in \mathbb{R}^n$, $j \in [t]$, divided by the median of the p-stable distribution (which can be approximated to $(1 \pm \epsilon)$ error in $\mathrm{poly}(1/\epsilon)$ time, see Appendix A.2 of [KNW10a] for details of this). This is done in Step 7 of Algorithm 15. It is standard it this gives a $(1 \pm 1/10)$ approximation the the norm $\|(\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}Y_{*,i}^\top\mathbf{F}^\top)_{*,l}\|_p$ for each $i \in [d], l \in [n]$ with probability $1 - 1/\mathrm{poly}(n)$ (See the Indyk median estimator of Theorem 8).

Now let $\sigma_{i,l}$ be our estimate of the norm $\|(\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}Y_{*,i}^\top\mathbf{F}^\top)_{*,l}\|_p$, for all $i \in [\xi]$ and $l \in [n]$. We now sample a columns $(i,l) \in [\xi] \times [n]$, where each (i,l) is chosen with probability $\sigma_{i,l}/(\sum_{i',l'} \sigma_{i',l'})$. We repeat this process $\Theta(r)$ times, to obtain a *multi-set* $T \subset [\xi] \times [n]$ of sampled columns (i,l). We stress that T is a multi-set, because the same column (i,l) may have been chosen for multiple samples, and each time it is chosen we must independently sample one of the entries of that column. For any $(i,l) \in T$, we define $W^{(i,l)} = (\mathbf{F}\mathbf{Y}_{*,i}\mathbf{1}^\top - \mathbf{1}Y_{*,i}^\top\mathbf{F}^\top)_{*,l} = (\mathbf{F}\mathbf{Y}_{*,i} - \mathbf{1}(\mathbf{F}\mathbf{Y})_{l,i})$.

ℓ_p Sampling an entry from $W^{(i,l)}$. Now fix any $(i,l) \in T$. We show how to ℓ_p sample an entry from the vector $W^{(i,l)} \in \mathbb{R}^n$. In other words, for a given $j \in [n]$, we want to sample $W_j^{(i,l)} \in [n]$ with probability at least $r|W_j^{(i,l)}|^p/\|W^{(i,l)}\|_p^p$. We do this in two steps. First, let $\mathbf{S}_0 \in \mathbb{R}^{k \times n}$ be the count-sketch for heavy hitters of definition 10.2.11, where $k = \text{poly}(r)$. Note that we can compute $\mathbf{S}_0 FY$ and $\mathbf{S}_0\mathbf{1}$ in time $\tilde{O}(n)$, since $\mathbf{FY} \in \mathbb{R}^{n \times \xi}$. Once this is done, for each $(i,l) \in T$ we can compute $\mathbf{S}_0 W^{(i,l)}$ in $O(k)$ time by computing $(\mathbf{S}_0\mathbf{1}(\mathbf{FY})_{l,i})$ (note that $\mathbf{FY}$ and $\mathbf{S}_0\mathbf{1}$ are already computed), and subtracting it off from the i-th column of $\mathbf{S}_0 FY$, so the total time is $\tilde{O}(n + \text{poly}(d/\epsilon))$ to compute $\mathbf{S}_0 W^{(i,l)}$ for all $(i,l) \in |T|$. Now we can obtain the set $Q_0^{(i,l)} \subset [n]$ containing all the $\tilde{\Omega}(1/\sqrt{k})$ heavy hitters in $W^{(i,l)}$ with high probability. We can then explicitly compute the value of $W_j^{(i,l)}$ for all $j \in Q_0^{(i,l)}$, and exactly compute the set

$$H^{(i,l)} = \left\{ j \in [n] \;\middle|\; |W_j^{(i,l)}|^p > \beta/r^{16}\|W^{(i,l)}\|_p^p \right\},$$

all in $\tilde{O}(k)$ time via definition 10.2.11, where $\beta > 0$ is a sufficiently small constant (here we use the fact that $|x|_p \geq |x|_2$ for $p \leq 2$). Note that we use the same sketch $\mathbf{S}_0$ to compute all sets $Q_0^{(i,l)}$, and union bound the event that we get the heavy hitters over all $\text{poly}(d/\epsilon)$ trails.

We are now ready to show how we sample an index from $W^{(i,l)}$. First, we estimate the total ℓ_p norm of the items in $[n_i] \setminus H^{(i,l)}$ (again with the Indyk median estimator), and call this $\alpha_{(i,l)}$ as in Algorithm 15, which can be computed in $O(|H^{(i,l)}|)$ additional time (by subtracting off the $|H^{(i,l)}|$ coordinates $ZW_\zeta^{(i,l)}$ for all heavy hitters $\zeta \in H^{(i,l)}$ from our estimate $\sigma_{(i,l)}$), and with probability $\alpha_{(i,l)}/\sigma_{(i,l)}$, we choose to sample one of the items of $H^{(i,l)}$, which we can then sample from the distribution $|W_j^{(i,l)}|^p/(\sum_{j \in H^{(i,l)}} |W_j^{(i,l)}|^p)$. Since all the $\sigma_{(i,l)}, \alpha_{(i,l)}$'s were constant factor approximations, it follows that we sampled such an item with probability $\Omega(r|W_{j'}^{(i,l)}|^p/\|C\|_p^p)$ as needed. Otherwise, we must sample an entry from $[n] \setminus H^{(i,l)}$. To do this, we first randomly partition $[n]$ into $\eta = \Theta(r^4/\epsilon^4)$ subsets $\Omega_1, \Omega_2, \ldots, \Omega_\eta$.

We now make the same argument made in the proof of Lemma 10.2.14, considering two cases. In the first case, the ℓ_p mass of $[n] \setminus H^{(i,l)}$ drops by a $1/r^2$ factor after removing the heavy hitters. In this case, $\alpha_{(i,l)}/\sigma_{(i,l)} = O(1/r^2)$, thus we will never *not* sample a heavy hitter with probability $1 - O(1/r)$, which we can safely ignore. Otherwise, the ℓ_p drops by less than a $1/r^2$ factor, and it follows that all remaining items must be at most a β/r^{14} heavy hitter over the remaining coordinates $[n] \setminus H^{(i,l)}$ (since if they were any larger, they would be β/r^{16} heavy hitters in $[n]$, and would have been removed in $H^{(i,l)}$). Thus we can assume we are in the second case. So by Chernoff bounds, we have $\sum_{j \in \Omega_t} |W_j^{(i,l)}|_p = \Theta(\frac{1}{\eta}\sum_{j \in [n] \setminus H^{(i,l)}} |W_j^{(i,l)}|_p)$ with probability greater than $1 - \exp(-\Omega(r))$. We can then union bound over this event occurring for all $t \in [\eta]$

and all $(i,l) \in T$. Given this, if we uniformly sample a $t \sim [\eta]$, and then ℓ_p sample a coordinate $j \in \Omega_t$, we will have sampled this coordinate with the correct probability up to a constant factor. We now sample such a t uniformly from η.

To do this, we generate a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, where $\mathbf{D}_{i,i} = 1/u_i^{1/p}$, where $u_1, \ldots, u_n$ are i.i.d. exponential random variables. For any set $\Gamma \subset [n]$, let $\mathbf{D}_\Gamma$ be D with all diagonal entries (j,j) such that $j \notin \Gamma$ set equal to 0. Now let $S \in \mathbb{R}^{k' \times n}$ be a second instance of count-sketch for heavy hitters of definition 10.2.11, where we set $k' = \text{poly}(k)$ from above. It is known that returning $j^* = \arg\max_{j \in \Omega_t \setminus H^{(i,l)}} |(\mathbf{D}W^{(i,l)})_j|$ is a perfect ℓ_p sample from $\Omega_t \setminus H^{(i,l)}$ [JW18b]. Namely, $\mathbf{Pr}[j^* = j] = |W_j^{(i,l)}|^p / \|W_{\Omega_t \setminus H^{(i,\ell)}}\|_p^p$ for any $j \in \Omega_t \setminus H^{(i,\ell)}$. Thus it will suffice to find this j^*. To find j^*, we compute $S(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}$. Note that since $\mathbf{FY}$ has already been computed, to do this we need only compute $S\mathbf{D}_{\Omega_t \setminus H^{(i,\ell)}} FY_{*,i}$ and $S\mathbf{D}_{\Omega_t \setminus H^{(i,\ell)}} \mathbf{1}(\mathbf{FY})_{\ell,i}$, which takes total time $\tilde{O}(|\Omega_t \setminus H^{(i,\ell)}|) = \tilde{O}(n/\eta)$. We then obtain a set $Q^{(i,l)} \subset \Omega_t \setminus H^{(i,\ell)}$ which contains all j with $|(\mathbf{D}W^{(i,l)})_j| \geq \tilde{\Omega}(1/\sqrt{k'}) \|(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}\|_2$.

As noted in [JW18b], the value $\max_{j \in \Omega_t \setminus H^{(i,l)}} |(\mathbf{D}W^{(i,l)})_j|$ is distributed identically to

$$\|W_{\Omega_t \setminus H^{(i,\ell)}}\|_p / u^{1/p}$$

where u is again an exponential random variable. Since exponential random variables have tails that decay like $e^{-\Omega(x)}$, it follows that with probability $1 - \exp(-\Omega(r))$ that we have

$$\max_{j \in \Omega_t \setminus H^{(i,l)}} |(\mathbf{D}W^{(i,l)})_j| = \Omega(\|W_{\Omega_t \setminus H^{(i,\ell)}}\|_p / r),$$

and we can then union bound over the event that this occurs for all $(i,l) \in T$ and Ω_t. Given this it follows that $(\mathbf{D}W^{(i,l)})_{j^*} = \Omega(\|W_{\Omega_t \setminus H^{(i,\ell)}}\|_p / r)$. Next, for any constant $c \geq 2$, by Proposition 1 of [JW18b], we have $\|((\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}})_{\text{tail}(c\log(n))}\|_2 = \tilde{O}(\|W^{(i,l)}_{\Omega_t \setminus H^{(i,\ell)}}\|_p)$ with probability $1 - n^{-c}$, where for a vector x, $x_{\text{tail}(t)}$ is x but with the top t largest (in absolute value) entries set equal to 0. Since there are at most $c\log(n)$ coordinates in $(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}$ not counted in $((\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}})_{\text{tail}(c\log(n))}$, and since $(\mathbf{D}W)_{j^*}$ is the largest coordinate in all of $(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}$, by putting together all of the above it follows that $(\mathbf{D}W)_{j^*}$ is a $\tilde{\Omega}(1/r)$-heavy hitter in $(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}$. Namely, that $|(\mathbf{D}W)_{j^*}| \geq \tilde{\Omega}(\|(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}\|_2 / r)$. Thus, we conclude that $j^* \in Q^{(i,l)}$.

Given that $j^* \in Q^{(i,l)}$, we can then compute the value $(\mathbf{D}\mathbf{W}^{(i,l)})_j = \mathbf{D}_{j,j}(\mathbf{FY}_{j,i} - \mathbf{FY}_{l,i})$ in $O(1)$ time to find the maximum coordinate j^*. Since $|Q^{(i,l)}| = O(k') = O(\text{poly}(d/\epsilon))$, it follows that the total time required to do this is $\tilde{O}(n/\eta + \text{poly}(d/\epsilon))$. Since we repeat this process for each $(i,l) \in T$, and $|T| = \Theta(r)$ whereas $\eta = \Theta(r^4)$, it follows that the total runtime for this

step is $\tilde{O}(n/r^3 + \mathrm{poly}(d/\epsilon))$. By [JW18b], the result is a perfect ℓ_p sample from $(\mathbf{D}W)_{\Omega_t \setminus H^{(i,\ell)}}$, which is the desired result. To complete the proof, we note that the only complication that remains is that we utilize the same scaling matrix $\mathbf{D}$ to compute the sampled used in each of the columns $W^{(i,l)}$ for each $(i,l) \in T$. However, note that for $t \neq t'$, we have that $\mathbf{D}_{\Omega_t}$ and $\mathbf{D}_{\Omega_t}$ are independent random variables. Thus it suffices to condition on the fact that the $t \in [\eta]$ that is sampled for each of the $|T|$ repetitions of sampling a Ω_t are distinct. But this occurs with probability at least $1/r$, since $|T| = \Theta(r)$ and $\eta = \Theta(r^4)$. Conditioned on this, all $|T|$ samples are independent, and each sample is an entry $\mathbf{C}_{i,j}$ of $\mathbf{C}$ such that the probability that a given (i,j) is chosen is $|\mathbf{C}_{i,j}|^p/\|\mathbf{C}\|_p^p$. Repeating this sampling $\Theta(r)$ times, we get that each $\mathbf{C}_{i,j}$ is sampled with probability at least $\min\{1, r|\mathbf{C}_{i,j}|^p/\|\mathbf{C}\|_p^p\}$, which completes the proof of correctness. Note that the dominant runtime of the entire procedure was $\widetilde{O}(\mathrm{nnz}(\mathbf{A}) + \mathrm{poly}(d/\epsilon))$ as stated, and the probability of success was $1 - \exp(-r) + 1/\mathrm{poly}(n)$, which we can be amplified to any $1 - \delta$ for $\delta > 1/n^c$ for some constant c by increasing the value of r by $\log(1/\delta)$ and the number of columns of the sketch $\mathbf{G}$ to $\log(1/\delta)$, which does not effect the $\widetilde{O}(\mathrm{nnz}(\mathbf{A}) + \mathrm{poly}(d/\epsilon))$ runtime.

Computing approximations $\tilde{q}_i$ for q_i. It remains now how to compute the approximate sampling probabilities $\tilde{q}_i$ for $\Theta(r)$ rows of $\mathbf{C}$ that were sampled. Note that to sample an entry, in $\mathbf{C}$, we first sampled the $n \times 1$ submatrix $W^{(i,l)}$ of $\mathbf{C}$ which contained it, where the probability that we sample this submatrix is known to us. Next, if the entry of $\mathbf{C}$ was a heavy hitter in $W^{(i,l)}$, we exactly compute the probability that we sample this entry, and sample it with this probability. If the entry j of $W^{(i,l)}$ is not a heavy hitter, we first sample an Ω_t uniformly with probability exactly $1/\eta$. The last step is sampling a coordinate from $W^{(i,l)}_{\Omega_t \setminus H^{(i,l)}}$ via exponential scaling. However, we do not know the exact probability of this sampling, since this will be equal to $|W^{(i,l)}_j|^p/\|W^{(i,l)}_{\Omega_t \setminus H^{(i,l)}}\|_p^p$, and we do not know $\|W^{(i,l)}_{\Omega_t \setminus H^{(i,l)}}\|_p^p$ exactly. Instead, we compute it approximately to error $(1 \pm \epsilon^2)$ as follows. For each $(i,l) \in T$ and $\alpha = 1, 2, \ldots, \Theta(\log(n)/\epsilon^4)$, we compute $Z^{(\alpha)} W^{(i,l)}_{\Omega_t \setminus H^{(i,l)}}$, where $Z \in \mathbb{R}^{1 \times |\Omega_t \setminus H^{(i,l)}|}$ is a vector of p-stable random variables. Again, we use the Indyk median estimator, taking the median of these $\Theta(\log(n)/\epsilon^4)$ repetitions, to obtain an estimate of $\|W^{(i,l)}_{\Omega_t \setminus H^{(i,l)}}\|_p^p$ with high probability to $(1 \pm \epsilon^2)$ relative error. Each repetition requires $O(|\Omega_t \setminus H^{(i,l)}|)$ additional time, and since $|\Omega_t \setminus H^{(i,l)}||T| = o(\epsilon^4 n/r^3)$, it follows that the total computational time is at most an additive $o(n)$, thus computing the $\tilde{q}_i$'s to error $(1 \pm \epsilon^2)$ does not effect the overall runtime.

□

10.4 Low Rank Approximation of Kronecker Product Matrices

We now consider low rank approximation of Kronecker product matrices. Given q matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_q$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times d_i}$, the goal is to output a rank-k matrix $\mathbf{B} \in \mathbb{R}^{n \times d}$, where $n = \prod_{i=1}^{q} n_i$ and $d = \prod_{i=1}^{q} d_i$, such that $\|\mathbf{B} - \mathbf{A}\|_F \leq (1+\epsilon)\,\mathrm{OPT}_k$, where $\mathrm{OPT}_k = \min_{\mathtt{rank}-k\ \mathbf{A}'} \|\mathbf{A}' - \mathbf{A}\|_F$, and $A = \otimes_{i=1}^{q} \mathbf{A}_i$. Our approach employs the Count-Sketch distribution of matrices [CW17]. A count-sketch matrix $\mathbf{S}$ is generated as follows. Each column of $\mathbf{S}$ contains exactly one non-zero entry. The non-zero entry is placed in a uniformly random row, and the value of the non-zero entry is either 1 or -1 chosen uniformly at random.

Our algorithm is as follows. We sample q independent Count-Sketch matrices $\mathbf{S}_1, \ldots \mathbf{S}_q$, with $\mathbf{S}_i \in \mathbb{R}^{k_i \times n_i}$, where $k_1 = \cdots = k_q = \Theta(qk^2/\epsilon^2)$. We then compute $\mathbf{M} = (\otimes_{i=1}^{q} \mathbf{S}_i)\mathbf{A}$, and let $\mathbf{U} \in \mathbb{R}^{k \times d}$ be the top k right singular vectors of M. Finally, we output $\mathbf{B} = \mathbf{A}\mathbf{U}^\top\mathbf{U}$ in factored form (as $q+1$ separate matrices, $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_q, \mathbf{U}$), as the desired rank-$k$ approximation to $\mathbf{A}$. The following theorem demosntrates the correctness of this algorithm.

Theorem 134. *For any constant $q \geq 2$, there is an algorithm which runs in time $O(\sum_{i=1}^{q} \mathrm{nnz}(\mathbf{A}_i) + d\mathrm{poly}(k/\epsilon))$ and outputs a rank k-matrix $\mathbf{B}$ in factored form such that*

$$\|\mathbf{B} - \mathbf{A}\|_F \leq (1+\epsilon)\,\mathrm{OPT}_k$$

with probability $9/10$. [5]

Proof. By Lemma 10.4.4, we have $(1-\epsilon)\|\mathbf{A} - \mathbf{A}\mathbf{P}\|_F^2 \leq \|\mathbf{M} - \mathbf{M}\mathbf{P}\|_F^2 + c \leq (1+\epsilon)\|\mathbf{A} - \mathbf{A}\mathbf{P}\|_F^2$ for all rank k projection matrices $\mathbf{P}$. In particular, we have

$$\min_P (1+\epsilon)\|\mathbf{A} - \mathbf{A}\mathbf{P}\|_F^2 + c = (1+\epsilon)\,\mathrm{OPT}_k^2$$

where the minimum is taken over all rank k projection matrices. The minimizer P on the LHS is given by the projection onto the top k singular space of $\mathbf{M}$. Namely, $\mathbf{M}\mathbf{P} = \mathbf{M}\mathbf{U}^\top\mathbf{U}$ where $\mathbf{U}$ is the top k singular row vectors of $\mathbf{M}$. Thus $\|\mathbf{M} - \mathbf{M}\mathbf{U}^\top\mathbf{U}\|_F^2 + c \leq (1+\epsilon)\,\mathrm{OPT}_k^2$. Moreover, we have $\|\mathbf{A} - \mathbf{A}\mathbf{U}^\top\mathbf{U}\|_F^2 \leq (1+2\epsilon)(\|\mathbf{M} - \mathbf{M}\mathbf{U}^\top\mathbf{U}\|_F^2 + c) \leq (1+4\epsilon)\,\mathrm{OPT}_k^2$. Thus $\|\mathbf{A} - \mathbf{A}\mathbf{U}^\top\mathbf{U}\|_F \leq (1+O(\epsilon))\,\mathrm{OPT}_k$ as needed.

[5] To amplify the probability, we can sketch $\mathbf{A}$ and $\mathbf{A}\mathbf{U}^\top\mathbf{U}$ with a sparse JL matrix (e.g., Lemma 10.4.3 with $k_i = \Theta(qk^2/(\delta\epsilon^2))$ for each i) in input sparsity time to estimate the cost of a given solution. We can then repeat $\log(1/\delta)$ times and take the minimum to get failure probability $1-\delta$.

For runtime, note that we first must compute $\mathbf{M} = (\otimes_{i=1}^{q} \mathbf{S}_i)(\mathbf{A}_1 \otimes \mathbf{A}_2) = \mathbf{S}_1\mathbf{A}_1 \otimes \cdots \otimes \mathbf{S}_q\mathbf{A}_q$. Now $\mathbf{S}_i\mathbf{A}_i$ can be computed in $O(\text{nnz}(\mathbf{A}_i))$ time for each i [CW17]. One all $\mathbf{S}_i\mathbf{A}_i$ are computed, their Kronecker product can be computed in time $O(qk_1k_2 \cdots k_q d) = \text{poly}(kd/\epsilon)$. Given $\mathbf{M} \in \mathbb{R}^{k_1 \cdots k_q \times d}$, the top k singular vectors U can be computed by computing the SVD of $\mathbf{M}$, which is also done in time $\text{poly}(kd/\epsilon)$. Once $\mathbf{U}$ is obtained, the algorithm can terminate, which yields the desired runtime. □

To complete the proof of the main theorem, we will need to prove Lemma 10.4.4. To do this, we begin by recalling the definition of a subspace embedding, and introducing the approximate matrix product property:

Definition 10.4.1 (Subspace embedding)**.** *A random matrix $\mathbf{S}$ is called a ϵ-subspace embedding for a rank k subspace $\mathcal{V}$ we have simultaneously for all $x \in \mathcal{V}$ that*

$$\|\mathbf{S}x\|_2 = (1 \pm \epsilon)\|x\|_2.$$

Definition 10.4.2 (Approximate matrix product)**.** *A random matrix $\mathbf{S}$ satisfies the ϵ-approximate matrix product property if, for any fixed matrices A, B, of the appropriate dimensions, we have*

$$\mathbf{Pr}\left[\|\mathbf{A}^\top\mathbf{S}^\top\mathbf{S}\mathbf{B} - \mathbf{A}^\top\mathbf{B}\|_F \leq \epsilon\|\mathbf{A}\|_F\|\mathbf{B}\|_F\right] \geq 9/10.$$

We now show that $\mathbf{S}$ is both a subspace embedding and satisfies approximate matrix product, where $\mathbf{S} = \otimes_{i=1}^{q} \mathbf{S}_i$ and $\mathbf{S}_i \in \mathbb{R}^{k_i \times n_i}$ are count-sketch matrices.

Lemma 10.4.3. *If $\mathbf{S} = (\otimes_{i=1}^{q} \mathbf{S}_i)$ with $\mathbf{S}_i \in \mathbb{R}^{k_i \times n_i}$, $k_1 = k_2 = \cdots = k_q = \Theta(qk^2/\epsilon^2)$, then $\mathbf{S}$ is an ϵ-subspace embedding for any fixed k dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$ with probability $9/10$, and also satisfies the (ϵ/k)-approximate matrix product property.*

Proof. We first show that $\mathbf{S}$ satisfies the $O(\epsilon/k, 1/10, 2)$-JL moment property. Here, the (ϵ, δ, ℓ)-JL moment property means that for any fixed $x \in \mathbb{R}^n$ with $\|x\|_2 = 1$, we have $\mathbf{E}\left[(\|Sx\|_2^2 - 1)^2\right] \leq \epsilon^\ell\delta$, which will imply approximate matrix product by the results of [KN14].

We prove this by induction on q. Let $\bar{k} = k_1$. First suppose $\mathbf{S} = (\mathbf{Q} \otimes \mathbf{T})$, where $\mathbf{Q} \in \mathbb{R}^{k_1 \times n_1}$ is a count-sketch, and $\mathbf{T} \in \mathbb{R}^{k' \times n'}$ is any random matrix which satisfies $\mathbf{E}\left[\|\mathbf{T}x\|_2^2\right] = \|x\|_2^2$ ($\mathbf{T} \in \mathbb{R}^{k' \times n'}$ is unbiased), and $\mathbf{E}\left[(\|\mathbf{T}x\|_2 - 1)^2\right] \leq 1 + c/\bar{k}$ for some value $c < \bar{k}$. Note that both of these properties are satisfied with $c = 4$ if $\mathbf{T} \in \mathbb{R}^{k_2 \times n_2}$ is itself a count-sketch matrix [CW17]. Moreover, these are the only properties we will need about T, so we will. We now prove that

$\mathbf{E}\left[\|(\mathbf{S}\otimes\mathbf{T})x\|_2^2\right]=1$ and $\mathbf{E}\left[\|(\mathbf{S}\otimes\mathbf{T})x\|_2^4\right]\leq 1+(c+4)/\bar{k}$ for any unit vector x.

Fix any unit $x\in\mathbb{R}^n$ now (here $n=n_1n'$), and let $x^j\in\mathbb{R}^{n'}$ be the vector obtained by restricted x to the coordinates jn_1+1 to $(j+1)n_1$. For any $i\in[k_1], j\in[k']$, let $i_j=(i-1)k'+j$. Let $h_{\mathbf{Q}}(i)\in[k_1]$ denote the row where the non-zero entry in the i-th column is placed in $\mathbf{Q}$. Let $\sigma_{\mathbf{Q}}(i)\in\{1,-1\}$ denote the sign of the entry $Q_{h_{\mathbf{Q}}(i),i}$. Let $\delta_{\mathbf{Q}}(i,j)$ indicate the event that $h_{\mathbf{Q}}(i)=j$. First note that

$$\begin{aligned}
\mathbf{E}\left[\sum_{i,j}((\mathbf{Q}\otimes\mathbf{T})x)_{i_j}^2\right] &= \mathbf{E}\left[\sum_{i=1}^{k_1}\sum_{j=1}^{k'}\left(\sum_{\tau=1}^{n_1}\delta_{\mathbf{Q}}(\tau,i)\sigma_{\mathbf{Q}}(\tau)(\mathbf{T}x^\tau)_j\right)^2\right]\\
&= \mathbf{E}\left[\sum_{i=1}^{k_1}\sum_{j=1}^{k'}\sum_{\tau=1}^{n_1}\delta_{\mathbf{Q}}(\tau,i)(\mathbf{T}x^\tau)_j^2\right]\\
&= \mathbf{E}\left[\sum_{\tau=1}^{n_1}\sum_{i=1}^{k_1}\sum_{j=1}^{k'}\delta_{\mathbf{Q}}(\tau,i)(\mathbf{T}x^\tau)_j^2\right]\\
&= \mathbf{E}\left[\sum_{\tau=1}^{n_1}\|\mathbf{T}x^\tau\|_2^2\right]\\
&= \|x\|_2^2
\end{aligned}$$

Where the last equality follows because count-sketch $\mathbf{T}$ is unbiased for the base case, namely that $\mathbf{E}\left[\|\mathbf{T}x\|_2^2\right]=\|x\|_2^2$ for any x [W+14], or by induction. We now compute the second moment,

$$\begin{aligned}
\mathbf{E}\left[\left(\sum_{i,j}((\mathbf{Q}\otimes\mathbf{T})x)_{i_j}^2\right)^2\right] &= \mathbf{E}\left[\left(\sum_{i,j}\left(\sum_{\tau=1}^{n_1}\delta_{\mathbf{Q}}(\tau,i)\sigma_{\mathbf{Q}}(\tau)(\mathbf{T}x^\tau)_j\right)^2\right)^2\right]\\
&= \mathbf{E}\left[\left(\sum_{i,j}\sum_{\tau_1,\tau_2}^{n_1}\delta_{\mathbf{Q}}(\tau_1,i)\sigma_{\mathbf{Q}}(\tau_1)(\mathbf{T}x^{\tau_1})_j\delta_{\mathbf{Q}}(\tau_2,i)\sigma_{\mathbf{Q}}(\tau_2)(\mathbf{T}x^{\tau_2})_j\right)^2\right]\\
&= \sum_{\tau_1,\tau_2,\tau_3,\tau_4}^{n_1}\mathbb{E}\left[\left(\sum_{i,j}\delta_{\mathbf{Q}}(\tau_1,i)\sigma_{\mathbf{Q}}(\tau_1)(\mathbf{T}x^{\tau_1})_j\delta_{\mathbf{Q}}(\tau_2,i)\sigma_{\mathbf{Q}}(\tau_2)(\mathbf{T}x^{\tau_2})_j\right)\right.\\
&\quad\left.\cdot\left(\sum_{i,j}\delta_{\mathbf{Q}}(\tau_3,i)\sigma_{\mathbf{Q}}(\tau_3)(\mathbf{T}x^{\tau_3})_j\delta_{\mathbf{Q}}(\tau_4,i)\sigma_{\mathbf{Q}}(\tau_4)(\mathbf{T}x^{\tau_4})_j\right)\right].
\end{aligned}$$

We now analyze the above expectation. There are several cases for the expectation of each term.

First, we bound the sum of the expectations when $t_1 = t_2 = t_3 = t_4$ by

$$\begin{aligned}
&\sum_{\tau=1}^{n_1} \mathbb{E}\left[\left(\sum_{i,j} \delta_{\mathbf{Q}}(\tau,i)\sigma_{\mathbf{Q}}(\tau)(\mathbf{T}x^{\tau})_j \delta_{\mathbf{Q}}(\tau,i)\sigma_{\mathbf{Q}}(\tau)(\mathbf{T}x^{\tau})_j\right)\right. \\
&\left.\cdot\left(\sum_{i,j} \delta_{\mathbf{Q}}(\tau,i)\sigma_{\mathbf{Q}}(\tau)(\mathbf{T}x^{\tau})_j \delta_{\mathbf{Q}}(\tau,i)\sigma_{\mathbf{Q}}(\tau)(\mathbf{T}x^{\tau})_j\right)\right] \\
\leq &\sum_{\tau=1}^{n_1} \mathbb{E}\left[\|\mathbf{T}x^{\tau}\|_2^4\right] = 1 + c/\bar{k}
\end{aligned}$$

Where the last equation follows from the variance of count-sketch [CW17] for the base case, or by induction for $q \geq 3$. We now bound the sum of the expectations when $t_1 = t_2 \neq t_3 = t_4$ by

$$\begin{aligned}
&\sum_{\tau_1 \neq \tau_2} \mathbb{E}\left[\left(\sum_{i,j} \delta_{\mathbf{Q}}(\tau_1,i)\sigma_{\mathbf{Q}}(\tau_1)(\mathbf{T}x^{\tau_1})_j \delta_{\mathbf{Q}}(\tau_1,i)\sigma_{\mathbf{Q}}(\tau_1)(\mathbf{T}x^{\tau_1})_j\right)\right. \\
&\left.\cdot\left(\sum_{i,j} \delta_{\mathbf{Q}}(\tau_2,i)\sigma_{\mathbf{Q}}(\tau_2)(\mathbf{T}x^{\tau_2})_j \delta_{\mathbf{Q}}(\tau_2,i)\sigma_{\mathbf{Q}}(\tau_2)(\mathbf{T}x^{\tau_2})_j\right)\right] \\
&\leq \sum_{\tau_1 \neq \tau_2} \mathbf{E}\left[\|\mathbf{T}x^{\tau_1}\|_2^2 \|\mathbf{T}x^{\tau_2}\|_2^2/k_1\right] \\
&\leq \mathbf{E}\left[\|\mathbf{T}x\|_2^4/k_1\right] \leq (1 + c/\bar{k})/k_1.
\end{aligned}$$

We can similarly bound the sum of the terms with $t_1 = t_3 \neq t_2 = t_4$ and $t_1 = t_4 \neq t_3 = t_2$ by $(1 + c/\bar{k})/k_1$, giving a total bound on the second moment of

$$\mathbf{E}\left[\|(\mathbf{Q} \otimes \mathbf{T})x\|_2^4\right] \leq 1 + c/\bar{k} + 3(1 + c/\bar{k})/k_1) \leq 1 + (4 + c)/\bar{k}$$

since any term with a $t_i \notin \{t_1, t_2, t_3, t_4\} \setminus \{t_i\}$ immediately has expectation 0. By induction, it follows that $\mathbf{E}\left[(\otimes_{i=1}^q \mathbf{S}_i)x\|_2^2\right] = 1$ for any unit x, and $\mathbf{E}\left[(\otimes_{i=1}^q \mathbf{S}_i)x\|_2^4\right] \leq 1 + (4q + c)/\bar{k}$, where c is the constant from the original variance of count-sketch. Setting $\bar{k} = k_1 = \cdots = k_q = \Theta(qk^2/\epsilon^2)$ with a large enough constant, this completes the proof that $\mathbf{S} = (\otimes_{i=1}^q \mathbf{S}_i)$ has the $O(\epsilon/k, 1/10, 2)$-JL moment property. Then by Theorem 21 of [KN14], we obtain the approximate matrix product property:

$$\mathbf{Pr}\left[\|\mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{B} - \mathbf{A}^\top \mathbf{B}\|_F \leq O(\epsilon/k)\|\mathbf{A}\|_F \|\mathbf{B}\|_F\right] \geq 9/10$$

for any two matrices $\mathbf{A}, \mathbf{B}$. Letting $\mathbf{A} = \mathbf{B}^\top = \mathbf{U}$ where $\mathbf{U} \in \mathbb{R}^{n \times k}$ is a orthogonal basis for any k-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$, it follows that

$$\|\mathbf{U}^\top \mathbf{S}^\top \mathbf{S}\mathbf{U} - \mathbb{I}_k\|_F \leq O(\epsilon/k)\|U\|_F^2 \leq O(\epsilon),$$

where the last step follows because U is orthonormal, so $\|U\|_F^2 = k$. Since the Frobenius norm upper bounds the spectral norm $\|\cdot\|_2$, we have $\|\mathbf{U}^\top \mathbf{S}^\top \mathbf{S}\mathbf{U} - \mathbb{I}_k\|_2 \leq O(\epsilon)$, from which it follows that all the eigenvalues of $\mathbf{U}^\top \mathbf{S}^\top \mathbf{S}\mathbf{U}$ are in $(1 - O(\epsilon), 1 + O(\epsilon))$, which implies $\|\mathbf{S}\mathbf{U}x\|_2 = (1 \pm O(\epsilon))\|x\|_2$ for all $x \in \mathbb{R}^n$, so for any $y \in \mathcal{V}$, let x_y be such that $y = \mathbf{U}x_y$, and then

$$\|\mathbf{S}y\|_2 = \|\mathbf{S}\mathbf{U}x_y\|_2 = (1 \pm O(\epsilon))\|x_y\|_2 = (1 \pm O(\epsilon))\|\mathbf{U}x_y\|_2 = (1 \pm O(\epsilon))\|y\|_2,$$

which proves that $\mathbf{S}$ is a subspace embedding for $\mathcal{V}$ (not the second to last inequality holds because U is orthonormal). □

Finally, we are ready to prove Lemma 10.4.4.

Lemma 10.4.4. *Let $\mathbf{S} = (\otimes_{i=1}^q \mathbf{S}_i)$ with $\mathbf{S}_i \in \mathbb{R}^{k_i \times n_i}$, $k_1 = k_2 = \cdots = k_q = \Theta(qk^2/\epsilon^2)$. Then with probability $9/10$ $\mathbf{SA}$ is a Projection Cost Preserving Sketch (*PCPSKETCH*) for A, namely for all rank k orthogonal projection matrix $P \in \mathbb{R}^{d \times d}$,*

$$(1 - \epsilon)\|\mathbf{A} - \mathbf{AP}\|_F^2 \leq \|\mathbf{SA} - \mathbf{SAP}\|_F^2 + c \leq (1 + \epsilon)\|\mathbf{A} - \mathbf{AP}\|_F^2$$

where $c \geq 0$ is some fixed constant independent of $\mathbf{P}$ (but may depend on A and $\mathbf{SA}$).

Proof. To demonstrate that $\mathbf{SA}$ is a PCPSKETCH, we show that the conditions of Lemma 10 of [CEM+15] hold, which imply this result. Our result follows directly from Theorem 12 of [CEM+15]. Note that all that is needed (as discussed below the theorem) for the proof is that $\mathbf{S}$ is an ϵ-subspace embedding for a fixed k-dimensional subspaces, and that $\mathbf{S}$ satisfies the $(\epsilon/\sqrt{k})$ approximate matrix product property. By Lemma 10.4.3, we have both ϵ-subspace embedding for $\mathbf{S}$ as well as a stronger (ϵ/k) approximate matrix product property. Thus Theorem 12 holds for the random matrix $\mathbf{S}$ when $k_1 = k_2 = \cdots = k_q = \Theta(qk^2/\epsilon^2)$, which completes the proof.

□

10.5 Evaluation

In our numerical simulations, we compare our algorithms to two baselines: (1) brute force, i.e., directly solving regression without sketching, and (2) the methods based sketching developed in [DSSW18]. All methods were implemented in Matlab on a Linux machine. We remark that in our implementation, we simplified some of the steps of our theoretical algorithm, such as the

residual sampling algorithm (Alg. 9). We found that in practice, even with these simplifications, our algorithms already demonstrated substantial improvements over prior work.

Following the experimental setup in [DSSW18], we generate matrices $\mathbf{A}_1 \in \mathbb{R}^{300\times 15}$, $\mathbf{A}_2 \in \mathbb{R}^{300\times 15}$, and $b \in \mathbb{R}^{300^2}$, such that all entries of $\mathbf{A}_1, \mathbf{A}_2, b$ are sampled i.i.d. from a normal distribution. Note that $\mathbf{A}_1 \otimes \mathbf{A}_2 \in \mathbb{R}^{90000\times 225}$. We define T_{bf} to be the time of the brute force algorithm, T_{old} to be the time of the algorithms from [DSSW18], and T_{ours} to be the time of our algorithms. We are interested in the time ratio with respect to the brute force algorithm and the algorithms from [DSSW18], defined as, $r_t = T_{\text{ours}}/T_{\text{bf}}$, and $r'_t = T_{\text{ours}}/T_{\text{old}}$. The goal is to show that our methods are significantly faster than both baselines, i.e., both r_t and r'_t are significantly less than 1.

We are also interested in the quality of the solutions computed from our algorithms, compared to the brute force method and the method from [DSSW18]. Denote the solution from our method as x_{our}, the solution from the brute force method as x_{bf}, and the solution from the method in [DSSW18] as x_{old}. We define the relative residual percentage $r_e$and r'_e to be:

$$r_e = 100\frac{|\|\mathcal{A}x_{\text{ours}} - b\| - \|\mathcal{A}x_{\text{bf}} - b\||}{\|\mathcal{A}x_{\text{bf}} - b\|}, \quad r'_e = 100\frac{|\|\mathcal{A}x_{\text{old}} - b\| - \|\mathcal{A}x_{\text{bf}} - b\||}{\|\mathcal{A}x_{\text{bf}} - b\|}$$

Where $\mathcal{A} = \mathbf{A}_1 \otimes \mathbf{A}_2$. The goal is to show that r_e is close zero, i.e., our approximate solution is comparable to the optimal solution in terms of minimizing the error $\|\mathcal{A}x - b\|$.

Throughout the simulations, we use a moderate input matrix size so that we can accommodate the brute force algorithm and to compare to the exact solution. We consider varying values of m, where , denotes the size of the sketch (number of rows) used in either the algorithms of [DSSW18] or the algorithms in this paper. We also include a column m/n in the table, which is the ratio between the size of the sketch and the original matrix $\mathbf{A}_1 \otimes \mathbf{A}_2$. Note in this case that $n = 90000$.

Simulation Results for ℓ_2 We first compare our algorithm, Alg. 7, to baselines under the ℓ_2 norm. In our implementation, $\min_x \|\mathbf{A}x - b\|_2$ is solved by Matlab backslash $A\backslash b$. Table 10.1 summarizes the comparison between our approach and the two baselines. The numbers are averaged over 5 random trials. First of all, we notice that our method in general provides slightly less accurate solutions than the method in [DSSW18], i.e., $r_e > r'_e$ in this case. However, comparing to the brute force algorithm, our method still generates relatively accurate solutions, especially when m is large, e.g., the relative residual percentage w.r.t. the optimal solution is around 1% when $m \approx 16000$. On the other hand, as suggested by our theoretical improvements

Table 10.1: Results for ℓ_2 and ℓ_1-regression with respect to different sketch sizes m.

	m	m/n	r_e	r'_e	r_t	r'_t
ℓ_2	8100	.09	2.48%	1.51%	0.05	0.22
	12100	.13	1.55%	0.98%	0.06	0.24
	16129	.18	1.20%	0.71%	0.07	0.08
ℓ_1	2000	.02	7.72%	9.10%	0.02	0.59
	4000	.04	4.26%	4.00%	0.03	0.75
	8000	.09	1.85%	1.6%	0.07	0.83
	12000	.13	1.29%	0.99%	0.09	0.79
	16000	.18	1.01%	0.70%	0.14	0.90

for ℓ_2, our method is significantly faster than the method from [DSSW18], consistently across all sketch sizes m. Note that when $m \approx 16000$, our method is around 10 times faster than the method in [DSSW18]. For small m, our approach is around 5 times faster than the method in [DSSW18].

Simulation Results for ℓ_1 We compare our algorithm, Alg. 8, to two baselines under the ℓ_1-norm. The first is a brute-force solution, and the second is the algorithm for [DSSW18]. For $\min_x \|\mathbf{A}x - b\|_1$, the brute for solution is obtained via a Linear Programming solver in Gurobi [GO16]. Table 10.1 summarizes the comparison of our approach to the two baselines under the ℓ_1-norm. The statistics are averaged over 5 random trials. Compared to the Brute Force algorithm, our method is consistently around 10 times faster, while in general we have relative residual percentage around 1%. Compared to the method from [DSSW18], our approach is consistently faster (around 1.3 times faster). Note our method has slightly higher accuracy than the one from [DSSW18] when the sketch size is small, but slightly worse accuracy when the sketch size increases.

Chapter 11

In-Database Regression

In the previous chapter, we demonstrated the power of sketching as a tool for accelerating numerical linear algebra algorithms when the design matix $\mathbf{A}$ admits nice structural properties. In Chapter 10, we studied the case where $\mathbf{A}$ was a Kronecker product of several smaller matrices, and designed algorithms for optimization tasks such as regression and low rank approximation. In this Chapter, we will study algorithms for regression when $\mathbf{A}$ has a more constrained structure; namely, when $\mathbf{A}$ is a *database join* of several smaller tables. Specifically, we study design matrices $\mathbf{A} = \mathbf{J}$ which are an arbitrary database join on tables $\mathbf{T}_1, \ldots, \mathbf{T}_m$, with $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, written as $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m$. The materials in this chapter are based on our work [JSWY21].

While join queries share many similarities to Kronecker products, they are structurally more complicated. Specifically, join queries do not admit many of the nice algebraic properties enjoyed by Kronecker products. On the other hand, like Kronecker products, the size of the join $\mathbf{J}$ can be exponentially large in the number of component tables $\mathbf{T}_i$. Specifically, $\mathbf{J}$ itself can have as many as $O(n_1 n_2 \cdots n_m)$ rows, and at most $\sum_i d_i$ columns.

Given the results of Chapter 10, a natural question is whether one can perform common linear algebraic tasks, such as regression, on database joins without explicitly computing $\mathbf{J}$. As we have seen, a crucial ingredient for such tasks is the design of a *subspace embedding* (SE) for $\mathbf{J}$ (Definition 2.4.2), which significantly reduces the number of rows in $\mathbf{J}$ while preserving lengths of vectors in the span. Recall that subspace embeddings are the primary tool for obtaining extremely efficient solutions for many linear-algebraic tasks, such as least squares regression and low rank approximation (see Section 2.4 for further background on subspace embeddings). The gold standard for obtaining such an embedding, as in Chapter 10, would be to design *input-*

sparsity runtime algorithms, where the sparsity is that of the component tables $\mathbf{T}_i$: namely a runtime of $O(\sum_i \text{nnz}(\mathbf{T}_i))$. Specifically, we ask:

Is it possible to apply a subspace embedding to a join, without having to explicitly form the join?

In this work, we accomplish precely this, by giving input-sparsity algorithms for computing subspace embeddings on joins of two-tables, $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, with running time bounded by $O(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2))$. We show that our SEs result in input sparsity time algorithms for high accuracy regression, significantly improving upon the running time of prior FAQ-based methods for regression. We also extend our results to arbitrary joins for the ridge regression problem, considerably improving the running time of prior methods for database joins. Empirically, we apply our method to real datasets and show that it is significantly faster than existing algorithms.

Highlighted Contributions

The main contributions of this chapter as as follows:

- We design the first algorithms for subspace embeddings and regression on database joins $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ of two joins, which moreover run in input sparsity time (Section 11.3).

- We give a framework to extend our algorithms to more general join queries, considerably improving the running time of prior state of the art methods (Section 11.4).

- We empirically evaluate both of our algorithms several important datasets, including the LastFM [CBK11] and MovieLens [HK15] datasets. Both our methods outperform prior state of the art FAQ based algorithms (developed in [AKNR16]) which compute exact solutions for regression, while maintaining extremely low relative error rates at or below 1% (Section 11.5).

11.1 Background

A canonical and important example of structured matrices is the database join. The prevalence of such joins in modern algorithmic tasks has motivated companies and teams such as Relation-

alAI [Rel] and Google's Bigquery ML [Big] to design databases that are capable of handling machine learning queries. Unfortunately, as noted earlier, in general the size of database joins grows exponentially with the number of component tables. Moreover, for many fundamental linear algebraic tasks such as regression, there are still no theoretically efficient algorithms for handling large database joins. Here, an efficient algorithm would produce good solutions without running the computationally intractable task of computing the join $\mathbf{J}$ itself. The design of such algorithms is the focus of this chapter.

Recall that in the database join setting, we have m tables $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_m$, where $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, and the database join $\mathbf{J}$ over a specified set of columns is denoted

$$\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m \in \mathbb{R}^{N \times d}$$

Notice that the actual set of columns, namely the structure of the join query, is not specified in the above notation, and must therefore be specified contextually. In this notation, we point out that the number N of rows of the join $\mathbf{J}$ can be as large as $n_1 n_2 \cdots n_m$.

We do note that it is possible to perform various operations on a join in sublinear (in the size of the join) time, using database algorithms that are developed for *Functional Aggregation Queries* (FAQ) [AKNR16, AKNN+18]. Specifically, using FAQ-based techniques, one can compute the covariance matrix $\mathbf{J}^T\mathbf{J}$ in time $O(d^4 mn \log(n))$ for an acyclic join,[1] where $n = \max(n_1, \ldots, n_m)$, after which one can solving least squares regression can be accomplished in poly(d) time. While this can be significantly faster than the $\Omega(Nd)$ time required to compute the actual join, it is still significantly larger than the input description size $\sum_i \text{nnz}(\mathbf{T}_i)$, which even if the tables are dense, is at most dmn. When the tables are sparse, e.g., $O(n)$ non-zero entries in each table, one could even hope for a running time close to $O(mn)$.

We note that the lack of input-sparsity time algorithms for regression on joins is further exacerbated in the presence of categorical features. Indeed, it is a common practice to convert categorical data to their so-called *one-hot encoding* before optimizing any statistical model. Such an encoding creates one column for each possible value of the categorical feature and only a single column is non-zero for each row. Thus, in the presence of categorical features, the tables in the join are extremely high dimensional and extremely sparse, and would benefit even more greatly from input sparsity algorithms.

In addition, since the data is high-dimensional, a common technique in regression is to regularize to avoid overfitting. A particularly popular form of regularized is *ridge regression*. Thus,

[1] See definition in Section 11.4.

in addition to ordinary regression, one could also ask if it is possible to efficently implement ridge regression on joins. Specifically, one could ask if it is possible to design input-sparsity time algorithms on joins for regression or ridge regression.

Can one design input-sparsity time algorithms on joins for both regression and ridge regression?

11.1.1 Our Contributions

We start by describing our results for least squares regression in the important case when the join is on two tables. We note that the two-table case is a very well-studied case, see, e.g., [AMS96, AGMS99, GWWZ15]. Moreover, one can always reduce to the case of a two-table join by precomputing part of a general join. The following two theorems state our results for computing a subspace embedding and for solving regression on the join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ of two tables. Our results demonstrate a substantial improvement over all prior algorithms for this problem. In particular, they answer the two questions above, showing that it is possible to compute a subspace embedding in input-sparsity time, and that regression can also be solved in input-sparsity time.

Previous to our work, the fastest algorithm for linear regression on two tables has a worst-case time complexity of $\tilde{O}(nd + nD^2)$, where $n = \max(n_1, n_2)$, d is the number of columns of the (non-encoded) join, and D is the dimensionality of the data after encoding the categorical data. Note that in the case of numerical (non-categorical) data, we have $D = d$ since there is no one-hot encoding and the time complexity is $O(nd^2)$. This complexity can be further improved to $O(nd^{\omega-1})$ where $\omega < 2.373$ is the exponent of fast matrix multiplication; note that this is the same as the fastest known time complexity for exact linear regression on a single table. In the case of categorical features (sparse data), using sparse tensors, the D^2 factor in the complexity can be replaced by $\Omega(d^2)$ [AKNN+18].

We state two results with differing leading terms and low-order additive terms, as one may be more useful than the other depending on whether the input tables are dense or sparse.

Theorem 135 (In-Database Subspace Embedding)**.** *Suppose $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{N \times d}$ is a join of two tables, where $\mathbf{T}_1 \in \mathbb{R}^{n_1 \times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2 \times d_2}$. Then Algorithm 12 outputs a sketching matrix $\mathbf{S}^* \in \mathbb{R}^{k \times N}$ such that with probability $9/10$ we have that $\mathbf{S}^*$ is an ϵ-subspace embedding for $\mathbf{J}$. In other words, we have*

$$\|\mathbf{S}^*\mathbf{J}x\|_2^2 = (1 \pm \epsilon)\|\mathbf{J}x\|_2^2$$

simultaneously for all $x \in \mathbb{R}^d$ with probability[2] *at least* $9/10$. *The running time to return* $\mathbf{S}^*\mathbf{J}$ *is the minimum of* $\tilde{O}((n_1+n_2)d/\epsilon^2+d^3/\epsilon^2)$ *and* $\tilde{O}((\text{nnz}(\mathbf{T}_1)+\text{nnz}(\mathbf{T}_2))/\epsilon^2+(n_1+n_2)/\epsilon^2+d^5/\epsilon^2)$.[3] *In the former case, we have* $k = \tilde{O}(d^2/\epsilon^2)$, *and in the latter case we have* $k = \tilde{O}(d^4/\epsilon^2)$.

Next, by following a standard reduction from a subspace embedding to an algorithm for regression, we obtain extremely efficient machine precision regression algorithms for two-table database joins.

Theorem 136 (Machine Precision Regression)**.** *Suppose* $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{N\times d}$ *is a join of two tables, where* $\mathbf{T}_1 \in \mathbb{R}^{n_1\times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2\times d_2}$. *Let* $\mathbf{U} \subseteq [d]$ *be any subset, and let* $\mathbf{J}_U \in \mathbb{R}^{N\times|U|}$ *be* $\mathbf{J}$ *restricted to the columns in* U, *and let* $b \in \mathbb{R}^N$ *be any column of the join* $\mathbf{J}$. *Then there is an algorithm which outputs* $\hat{x} \in \mathbb{R}^{|U|}$ *such that with probability* $9/10$[4] *we have*

$$\|\mathbf{J}_U\hat{x} - b\|_2 \leq (1+\epsilon)\min_{x\in\mathbb{R}^{|U|}} \|\mathbf{J}_U x - b\|_2$$

The running time required to compute $\hat{x}$ *is the minimum of* $\tilde{O}(((n_1+n_2)d+d^3)\log(1/\epsilon))$ *and* $\tilde{O}((\text{nnz}(\mathbf{T}_1)+\text{nnz}(\mathbf{T}_2)+d^5)\log(1/\epsilon))$.

General Joins We next consider arbitrary joins on more than two tables. In this case, we primarily focus on the ridge regression problem $\min_x \|\mathbf{J}x - b\|_2^2 + \lambda\|x\|_2^2$, for a regularization parameter λ. This problem is a popular regularized variant of regression and was considered in the context of database joins in [AKNN+18]. We introduce a general framework to apply sketching methods over arbitrary joins in Section 11.4; our method is able to take a sketch with certain properties as a black box, and can be applied both to TensorSketch [ANW14, Pag13, PP13], as well as recent improvements to this sketch [AKK+20b, WZ20a] for certain joins. Unlike previous work, which required computing $\mathbf{J}^T\mathbf{J}$ exactly, we show how to use sketching to approximate this up to high accuracy, where the number of entries of $\mathbf{J}^T\mathbf{J}$ computed depends on the so-called statistical dimension of $\mathbf{J}$, which can be much smaller than the data dimension D.

[2]We remark that using standard techniques for amplifying the success probability of an SE (see Section 2.3 of [W+14]) one can boost the success probability to $1-\delta$ by repeating the entire algorithm $O(\log\delta^{-1})$ times, increasing the running time by a multiplicative $O(\log\delta^{-1})$ factor. One must then compute the SVD of each of the $O(\log\delta^{-1})$ sketches, which results in an additive $O(kd^{\omega-1}\log\delta^{-1})$ term in the running time, where $\omega \approx 2.376$ is the exponent of matrix multiplication. Note that this additive dependence on d is only slightly ($\approx d^{.376}$) larger than the dependence required for constant probability as stated in the theorem.

[3]We use $\tilde{O}$ notation to omit factors of $\log(N)$.

[4]The probability of success here is the same as the probability of success of constructing a subspace embedding; see earlier footnote about amplifying this success probability.

Evaluation Empirically, we compare our algorithms on various databases to the previous best FAQ-based algorithm of [AKNN+18], which computes each entry of the covariance matrix $\mathbf{J}^T\mathbf{J}$. For two-table joins, we focus on the standard regression problem. We use the algorithm described in Section 11.3, replacing the Fast Tensor-Sketch with Tensor-Sketch for better practical performance. For general joins, we focus on the ridge regression problem; such joins can be very high dimensional and ridge regression helps to prevent overfitting. We apply our sketching approach to the entire join and obtain an approximation to it, where our complexity is in terms of the statistical dimension rather than the actual dimension D. Our results demonstrate significant speedups over the previous best algorithm, with only a small sacrifice in accuracy. For example, for the join of two tables in the MovieLens data set, which has 23 features, we obtain a 10-fold speedup over the FAQ-based algorithm, while maintaining a 0.66% relative error. For the natural join of three tables in the real MovieLens data set, which is a join with 24 features, we obtain a 3-fold speedup over the FAQ-based algorithm with only 0.28% MSE relative error. Further details can be found in Section 11.5.

11.1.2 Related In-Database Machine Learning Work

The work of [AKNR16] introduced Inside-out, a polynomial time algorithm for calculating functional aggregation queries (FAQs) over joins without performing the joins themselves, which can be utilized to train various types of machine learning models. The Inside-Out algorithm builds on several earlier papers, including [AM00, Dec96, KW08, GM06]. Relational linear regression, singular value decomposition, and factorization machines are studied extensively in multiple prior works [Ren13, KNP15, SOC16, KNN+18, AKNN+18, KNPZ16, ELB+17, KJY+15]. The best known time complexity for training linear regression when the features are continuous, is $O(d^4mn^{\text{fhtw}}\log(n))$ where fhtw is the fractional hypertree width of the join query. Note that fhtw is 1 for acyclic joins. For categorical features, the time complexity is $O(d^2mn^{\text{fhtw}+2})$ in the worst-case; however, [AKNN+18] uses sparse computation of the results to reduce this time depending on the join instance. In the case of polynomial regression, the calculation of pairwise interactions among the features can be time-consuming and it is addressed in [AKNN+18, LCK19]. Relational support vector machines with Gaussian kernels are studied in [YGL+]. In [CK19], a relational algorithm is introduced for Independent Gaussian Mixture Models, which can be used for kernel density estimation by estimating the underlying distribution of the data.

11.2 Preliminaries on Database Joins and Relevant Sketches

We first introduce the notion of a *block* of a join, which will be important in our analysis. Let $\mathbf{T}_1, \ldots, \mathbf{T}_m$ be tables, with $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$. Let $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m \in \mathbb{R}^{N \times d}$ be an arbitrary join on the tables $\mathbf{T}_i$. Let Q be the subset of columns which are contained in at least two tables, e.g., the columns which are joined upon. For any subset U of columns and any table T containing a set of columns U', let $T|_U$ be the projection of T onto the columns in $U \cap U'$. Similarly define $r|_U$ for a row r. Let C be the set of columns in $\mathbf{J}$, and let $C_j \subset C$ be the columns contained in T_j. Define the set of *blocks* $\mathcal{B} = \mathcal{B}(\mathbf{J}) \subset \mathbb{R}^{|Q|}$ of the join to be the set of distinct rows in the projection of $\mathbf{J}$ onto Q. In other words, $\mathcal{B}$ is the set of distinct rows which occur in the restriction of $\mathbf{J}$ to the columns being joined on. For any $j \in [m]$, let $\hat{\mathbf{T}}_j \in \mathbb{R}^{n_j \times d}$ be the embedding of the rows of $\mathbf{T}_j$ into the join $\mathbf{J}$, obtained by padding $\mathbf{T}_j$ with zero-valued columns for each column not contained in $\mathbf{T}_j$, and such that for any column c contained in more than one $\mathbf{T}_j$, we define the matrices $\hat{\mathbf{T}}_j$ so that exactly one of the $\hat{\mathbf{T}}_j$ contains c (it does not matter which of the tables containing the column c has c assigned to it in the definition of $\hat{\mathbf{T}}_j$). More formally, we fix any partition $\{\hat{C}_j\}_{j \in [m]}$ of C, such that $C = \cup_j \hat{C}_j$ and $\hat{C}_j \subseteq C_j$ for all j.

For simplicity, given a block $\vec{i} \in \mathcal{B}$, which was defined as a row in $\mathbb{R}^{|Q|}$, we drop the vector notation and write $\vec{i} = i \in \mathcal{B}$. For a given $i = (i_1, \ldots, i_{|Q|}) \in \mathcal{B}$, let $s_{(i)}$ denote the *size* of the block, meaning the number of rows r of the join $\mathbf{J}$ such that i_j is in the j-th column of r for all $j \in Q$. For $i \in \mathcal{B}$, let $\mathbf{T}_j^{(i)}$ be the subset of rows r in $\mathbf{T}_j$ such that $r|_Q = i|_{C_j}$, and similarly define $\hat{\mathbf{T}}_j^{(i)}, \mathbf{J}^{(i)}$ to be the subset of rows r in $\hat{\mathbf{T}}_j$ (respectively $\mathbf{J}$) such that $r|_Q = i|_{\hat{C}_j}$ (respectively $r|_Q = i$). For a row r such that $r|_Q = i$ we say that r "belongs" to the block $i \in \mathcal{B}$. Let $s_{(i),j}$ denote the number of rows of $\mathbf{T}_j^{(i)}$, so that $s_{(i)} = \prod_{j=1}^m s_{(i),j}$.

As an example, considering the join $T_1(A, B) \bowtie T_2(B, C)$, we have one block for each distinct value of B that is present in both T_1 and T_2, and for a given block $B = b$, the size of the block can be computed as the number of rows in T_1, with $B = b$, multiplied by the number of rows in T_2, with $B = b$.

Using the above notion of blocks of a join, we can construct $\mathbf{J}$ as a stacking of matrices $\mathbf{J}^{(i)}$ for $i \in \mathcal{B}$. For the case of two table joins $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, we have $\mathbf{J}^{(i)} = \left(\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)}\right) \in \mathbb{R}^{s_{(i)} \times d}$. In other words, $\mathbf{J}^{(i)}$ is the subset of rows of $\mathbf{J}$ contained in block i. Observe that the entire join $\mathbf{J}$ is the result of stacking the matrices $\mathbf{J}^{(i)}$ on top of each other, for all $i \in \mathcal{B}$. In other words, if $\mathcal{B} = \{i_1, i_2, \ldots, i_{|\mathcal{B}|}\}$, the join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ is given by $\mathbf{J} = \left[(\mathbf{J}^{(i_1)})^T, (\mathbf{J}^{(i_2)})^T, \ldots, (\mathbf{J}^{(i_{|\mathcal{B}|})})^T\right]^T$.

Figure 11.1 illustrates an example of blocks in a two table join. In this example column f_2 is

Table 11.1: Table of Notation

T_i	$\triangleq$	a $n_i \times d_i$ sized table
$\mathbf{J}$	$\triangleq$	join of all tables, i.e., $\mathbf{J} = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$
C	$\triangleq$	set of columns of $\mathbf{J}$
C_i	$\triangleq$	set of columns of T_i
$\hat{C}_i$	$\triangleq$	Partition of C such that $\hat{C}_i \subseteq C_i$ for $i \in [m]$.
$T\|_U$	$\triangleq$	projection of T onto columns $U \cap U'$, where U' are the columns of T
$\hat{T}_i$	$\triangleq$	Result of padding $T_i\|_{\hat{C}_i}$ with zero-valued columns in $C \setminus \hat{C}_i$
$\mathcal{B}$	$\triangleq$	set of *blocks*, i.e., distinct rows of $\mathbf{J}\|_Q$
$s_{(i)}$	$\triangleq$	size of block i, i.e., number of rows r of $\mathbf{J}$ with $r\|_q = i \in \mathcal{B}$
$T_j^{(i)}$	$\triangleq$	subset of rows r in $\mathbf{T}_j$ such that $r\|_Q = i\|_{C_j}$
$\hat{T}_j^{(i)}$	$\triangleq$	subset of rows r in $\hat{T}_j$ such that $r\|_Q = i\|_{\hat{C}_j}$
$\mathbf{J}_j^{(i)}$	$\triangleq$	subset of rows r in $\mathbf{J}$ such that $r\|_Q = i$

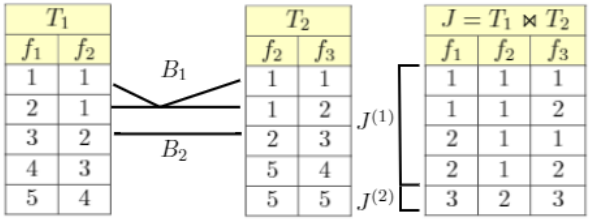


Figure 11.1: Example of two table join blocks

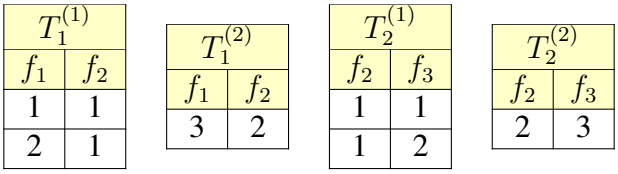


Figure 11.2: Examples of $T_i^{(j)}$

the column that we are joining the two tables on, and there are two values for f_2 that are present in both tables, namely the values$\{1, 2\}$. Thus $\mathcal{B} = \{B_1, B_2\}$, where $B_1 = 1$ and $B_2 = 2$. In other words, Block B_1 is the block for value 1, and its size is $s_1 = 4$, and similarly B_2 has size $s_2 = 4$. Figure 11.1 illustrates how the join $\mathcal{J}$ can be written as stacking together the block-matrices $\mathbf{J}^{(1)}$ and $\mathbf{J}^{(2)}$. Figure 11.2 shows the tables $T_i^{(j)}$ for different values of i and j in the same example.

Finally, for any subset $U \subseteq [N]$, let $\mathbf{J}_U$ denote the set of rows of $\mathbf{J}$ belonging to U. If L is a set of blocks of J, meaning $L \subseteq \mathcal{B}(\mathbf{J})$, then let $\mathbf{J}_L$ denote the set of rows of $\mathbf{J}$ belonging to some block $i \in L$ (recall that a row r "belongs" to a block $i \in L \subseteq \mathcal{B}$ if $r|_Q = i$).

11.2.1 Some Useful Sketches

We now introduce several sketches which will be useful for our main regression algorithms. We begin by defining the notion of the *statistical dimension* of a matrix.

Definition 11.2.1 (Statistical Dimension)**.** *For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, and a non-negative scalar λ, the λ-statistical dimension is defined to be $d_\lambda = \sum_i \frac{\lambda_i(\mathbf{A}^T\mathbf{A})}{\lambda_i(\mathbf{A}^T\mathbf{A})+\lambda}$, where $\lambda_i(\mathbf{A}^T\mathbf{A})$ is the i-th eigenvalue of $\mathbf{A}^T\mathbf{A}$.*

We also recall the definition of a subspace embedding from Chapter 2. For the purposes of this chapter only, it will be useful to change our terminology slightly, and refer to the embedded matrix itself $\mathbf{SA}$ as being an ϵ-subspace embedding for $\mathbf{A}$, as opposed to calling the sketching matrix $\mathbf{S}$ a subspace embedding for $\mathbf{A}$. Note that the latter is the terminology used in the rest of the thesis (as introduced in Section 2.4.1), although both conventions are common in the literature. Specifically, we will use the following terminology for subspace embeddings in this chapter.

Definition 11.2.2 (Subspace Embedding)**.** *For an $\epsilon \geq 0$, we say that $\tilde{\mathbf{A}} \in \mathbb{R}^{m\times d}$ is an ϵ-subspace embedding for $\mathbf{A} \in \mathbb{R}^{n\times d}$ if for all $x \in \mathbb{R}^d$ we have*

$$(1-\epsilon)\|\mathbf{A}x\|_2 \leq \|\tilde{\mathbf{A}}x\|_2 \leq (1+\epsilon)\|\mathbf{A}x\|_2.$$

Note that if $\tilde{\mathbf{A}} \in \mathbb{R}^{m\times d}$ is an ϵ-subspace embedding for $\mathbf{A} \in \mathbb{R}^{n\times d}$, in particular this implies that $\mathbf{\Sigma}_i(\mathbf{A}) = (1 \pm \epsilon)\mathbf{\Sigma}_i(\tilde{\mathbf{A}})$ for all $i \in [d]$.

Generalized Leverage Scores In this section, we introduce the notion of *generalized leverage scores*, which extend the Definition 2.4.1 from Chapter 2. First, recall that the leverage score of the i-th row $\mathbf{A}_{i,*}$ of $\mathbf{A} \in \mathbb{R}^{n\times d}$ is defined to be $\tau_i(\mathbf{A}) = \mathbf{A}_{i,*}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}_{i,*}^T$. Let $\tau(\mathbf{A}) \in \mathbb{R}^n$ be the vector such that $(\tau(\mathbf{A}))_i = \tau_i(\mathbf{A})$. Then $\tau(\mathbf{A})$ is the diagonal of $\mathbf{A}(\mathbf{A}^T\mathbf{A})^+\mathbf{A}^T$, which is a projection matrix. Thus $\tau_i(\mathbf{A}) \leq 1$ for all $i \in [n]$. It is also easy to see that $\sum_{i=1}^n \tau_i(\mathbf{A}) \leq d$ [CLM+15]. Given matrices $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $\mathbf{B} \in \mathbb{R}^{n_1\times d}$, the generalized leverage scores of $\mathbf{A}$ with respect to $\mathbf{B}$ are defined as

$$\tau_i^{\mathbf{B}}(\mathbf{A}) = \begin{cases} \mathbf{A}_{i,*}(\mathbf{B}^T\mathbf{B})^+\mathbf{A}_{i,*}^T & \text{if } \mathbf{A}_{i,*} \perp \ker(\mathbf{B}) \\ 1 & \text{otherwise} \end{cases}$$

We remark that in the case were $\mathbf{A}_{i,*}$ has a component in the kernel (null space) of B, denoted by $\ker(\mathbf{B})$, $\tau_i^{\mathbf{B}}(\mathbf{A})$ is defined to be ∞ in [CLM+15]. However, as stated in that paper, this definition was simply for notational convenience, and the results would equivalently hold setting $\tau_i^{\mathbf{B}}(\mathbf{A}) = 1$ in this case. Note that for a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ with SVD $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we have $(\mathbf{B}^T\mathbf{B})^+ = \mathbf{V}\mathbf{\Sigma}^{-2}\mathbf{V}^T$. Thus, for any $x \in \mathbb{R}^d$, we have $x^T(\mathbf{B}^T\mathbf{B})^+x = \|x^T\mathbf{V}\mathbf{\Sigma}^{-1}\|_2^2$, and in particular $\tau_i^{\mathbf{B}}(\mathbf{A}) = \|\mathbf{A}_{i,*}^T\mathbf{V}\mathbf{\Sigma}^{-1}\|_2^2$ if $\mathbf{A}_{i,*}\perp\ker(\mathbf{B})$, where $\mathbf{A}_{i,*}\perp\ker(\mathbf{B})$ means $\mathbf{A}_{i,*}$ is perpendicular to the kernel of $\mathbf{B}$.

Proposition 11.2.3. *If $\mathbf{B}' \in \mathbb{R}^{n_1'\times d}$ is an ϵ-subspace embedding for $\mathbf{B} \in \mathbb{R}^{n_1\times d}$, and $\mathbf{A} \in \mathbb{R}^{n\times d}$ is any matrix, then $\tau_i^{\mathbf{B}'}(\mathbf{A}) = (1 \pm O(\epsilon))\tau_i^{\mathbf{B}}(\mathbf{A})$*

Proof. If $\mathbf{B}'$ is an ϵ-subspace embedding for $\mathbf{B}$, then the spectrum of $(\mathbf{B}'(\mathbf{B}')^T)^+$ is a $(1\pm\epsilon)^{-2}$ approximation to the spectrum of $(\mathbf{B}\mathbf{B}^T)^+$, so $x^T(\mathbf{B}'(\mathbf{B}')^T)^+x = (1\pm\epsilon)^{-2}x^T(\mathbf{B}\mathbf{B}^T)^+x$ for all $x \in \mathbb{R}^d$, which completes the proof. □

Our algorithm will employ a mixture of several known oblivious subspace embeddings as tools to construct our overall database join SE. The first result we will need is an improved variant of *Tensor-Sketch*, which is an SE that can be applied quickly to tensor products of matrices.

Lemma 11.2.4 (Fast Tensor-Sketch, Theorem 3 of [AKK+20b])**.** *Fix any matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m$, where $\mathbf{A}_i \in \mathbb{R}^{n_i\times d_i}$, fix $\epsilon > 0$ and $\lambda \geq 0$. Let $n = n_1\cdots n_m$ and $d = d_1\cdots d_m$. Let $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_m$ have statistical dimension d_λ. Then there is an oblivious randomized sketching algorithm which produces a matrix $\mathbf{S} \in \mathbb{R}^{k\times n}$, where $k = O(d_\lambda m^4/\epsilon^2)$, such that with probability $1 - 1/\mathrm{poly}(n)$, we have that for all $x \in \mathbb{R}^d$*

$$\|\mathbf{S}\mathbf{A}x\|_2^2 + \lambda\|x\|_2^2 = (1\pm\epsilon)(\|\mathbf{A}x\|_2^2 + \lambda\|x\|_2^2).$$

Note for the case of $\lambda = 0$, this implies that $\mathbf{S}\mathbf{A}$ is an ϵ-subspace embedding for $\mathbf{A}$. Moreover, $\mathbf{S}\mathbf{A}$ can be computed in time $\tilde{O}(\sum_{i=1}^m \mathrm{nnz}(\mathbf{A}_i)/\epsilon^2 \cdot m^5 + kdm)$.[5]

For the special case of $\lambda = 0$ in Lemma 11.2.4, the statistical dimension is d, and Tensor-Sketch is just a standard SE.

Lemma 11.2.5 (OSNAP Transform [NN13])**.** *Given any $\mathbf{A} \in \mathbb{R}^{N\times d}$, there is a randomized oblivious sketching algorithm that produces a matrix $\mathbf{W} \in \mathbb{R}^{t\times N}$ with $t = \tilde{O}(d/\epsilon^2)$, such that*

[5]Theorem 3 of [AKK+20b] is written to be applied to the special case of the polynomial kernel, where $A_1 = A_2 = \cdots = A_m$. However, the algorithm itself does not use this fact, nor does it require the factors in the tensor product to be non-distinct.

$\mathbf{WA}$ *can be computed in time* $\tilde{O}(\mathrm{nnz}(\mathbf{A})/\epsilon^2)$*, and such that* $\mathbf{WA}$ *is an* ϵ*-subspace embedding for* $\mathbf{A}$ *with probability at least* $99/100$*. Moreover, each column of* $\mathbf{W}$ *has at most* $\tilde{O}(1)$ *non-zero entries.*

Lemma 11.2.6 (Count-Sketch [CW17])**.** *For any fixed matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$*, and any* $\epsilon > 0$*, there exists an algorithm which produces a matrix* $\mathbf{S} \in \mathbb{R}^{k\times n}$*, where* $k = O(d^2/\epsilon^2)$*, such that* $\mathbf{SA}$ *is an* ϵ*-subspace embedding for* $\mathbf{A}$ *with probability at least* $99/100$*. Moreover, each column of* $\mathbf{S}$ *contains exactly one non-zero entry, and therefore* $\mathbf{SA}$ *can be computed in* $O(\mathrm{nnz}(\mathbf{A}))$ *time.*

11.3 Subspace Embeddings for Two-Table Database Joins

In this section, we will describe our algorithms for fast computation of in-database subspace embeddings for joins of two tables $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, where $\mathbf{T}_i \in \mathbb{R}^{n_i\times d_i}$, and $\mathbf{J} \in \mathbb{R}^{N\times d}$. As a consequence of our subspace embeddings, we obtain an input sparsity time algorithm for machine precision in-database regression. Here, machine precision refers to a convergence rate of $\log(1/\epsilon)$ to the optimal solution. In what follows, we first present a high-level discussion of the algorithm and intuition.

Our subspace embedding algorithm can be run with two separate hyper-parameterizations, one of which we refer to as the *dense case* where the tables $\mathbf{T}_1, \mathbf{T}_2$ have many non-zero entries, and the other is referred to as the *sparse case*, where we exploit the sparsity of the tables $\mathbf{T}_1, \mathbf{T}_2$. In the former, we will obtain $\tilde{O}(\frac{1}{\epsilon^2}((n_1 + n_2)d + d^3))$ runtime for construction of our subspace embedding, and in the latter we will obtain $\tilde{O}(\frac{1}{\epsilon^2}(\mathrm{nnz}(\mathbf{T}_1) + \mathrm{nnz}(\mathbf{T}_2) + d^5))$ time. Thus, when the matrices $\mathbf{T}_1, \mathbf{T}_2$ are dense, we have $(n_1 + n_2)d = \Theta(\mathrm{nnz}(\mathbf{T}_1) + \mathrm{nnz}(\mathbf{T}_2))$, in which case the former algorithm has a strictly better runtime dependence on n_1, n_2, and d. However, for the many applications where $\mathbf{T}_1, \mathbf{T}_2$ are sparse, the latter algorithm will yield substantial improvements in runtime. By first reading off the sparsity of $\mathbf{T}_1, \mathbf{T}_2$ and choosing the hyperparameters which minimize the runtime, the final runtime of the algorithm is the minimum of the two aforementioned runtimes.

Our main subspace embedding is given in Algorithm 12. As noted in Section 11.2, we can describe the join $\mathbf{J}$ as the result of stacking several "blocks" $\mathbf{J}^{(i)}$, where the rows of $\mathbf{J}^{(i)}$ consist of all pairs of concatenations of a row of $\mathbf{T}_1^{(i)}$ and $\mathbf{T}_2^{(i)}$, where the $\mathbf{T}_j^{(i)}$'s partition $\mathbf{T}_j$. We deal separately with blocks i for which $\mathbf{J}^{(i)}$ contains a very large number of rows, and smaller blocks. Formally, we split the set of blocks $\mathcal{B}(\mathbf{J})$ into $\mathcal{B}_{\text{big}}$ and $\mathcal{B}_{\text{small}}$. For each block $\mathbf{J}^{(i)}$ from $\mathcal{B}_{\text{big}}$, we apply a fast tensor sketch transform to obtain a subspace embedding for that block. By

construction, there can only be a small number, namely at most $(n_1+n_2)/(\gamma d)$, big blocks, where $\gamma = 1$ in the dense case and $\gamma = d$ in the sparse case. Thus, the additive $O(d)$ term required for running tensor sketch on each block results in a runtime which is within our required bounds.

For the smaller blocks, however, we need a much more involved routine. This is because there can be many more than $(n_1 + n_2)/(\gamma d)$ small blocks, so we cannot afford to run a tensor-sketch on each. Instead, our algorithm computes a random sample of the rows of the blocks $\mathbf{J}^{(i)}$ from $\mathcal{B}_{\text{small}}$, denoted $\tilde{\mathbf{J}}_{\text{small}}$. Using the results of [CLM+15], it follows that sampling sufficiently many rows from the distribution induced by the generalized leverage scores $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ of $\mathbf{J}_{\text{small}}$ with respect to $\tilde{\mathbf{J}}_{\text{small}}$ yields a subspace embedding of $\mathbf{J}_{\text{small}}$. However, it is not possible to write down (let alone compute) all the values $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$, since there can be more rows i in $\mathbf{J}_{\text{small}}$ than our entire allowable running time.

To handle this issue, we first note that by Proposition 11.2.3 and the discussion prior to it, the value $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ is well-approximated by $\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\mathbf{\Sigma}^{-1}\|_2^2$, which in turn is well-approximated by $\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{G}\|_2^2$ if $\mathbf{G}$ is a Gaussian matrix with only a small $\Theta(\log N)$ number of columns. Thus, sampling from the generalized leverage scores $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ can be approximately reduced to the problem of sampling a row i from $\mathbf{J}_{\text{small}}\mathbf{Y}$ with probability proportional to $\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{Y}\|_2^2$, where $\mathbf{Y}$ is any matrix given as input. We then design a fast algorithm which accomplishes precisely this task: namely, for any join $\mathbf{J}'$ and input matrix $\mathbf{Y}$ with a small number of columns, it samples rows from $\mathbf{J}'\mathbf{Y}$ with probability proportional to the squared row norms of $\mathbf{J}'\mathbf{Y}$. Since $\mathbf{J}_{\text{small}} = \mathbf{J}'$ is itself a database join, this is the desired sampler. This procedure is given in Algorithms 13 (pre-processing step) and 14 (sampling step). We can apply this sampling primitive to efficiently sample from the generalized leverage scores in time substantially less than constructing $\mathbf{J}_{\text{small}}$, which ultimately allows for our final subspace embedding guarantee of Theorem 135.

Finally, to obtain our input sparsity runtime machine precision regression algorithm, we apply our subspace embedding with constant ϵ to precondition the join $\mathbf{J}$, after which the regression problem can be solved quickly via gradient descent. While a general gradient step is not always possible to compute efficiently with respect to the join $\mathbf{J}$, we demonstrate that when the products used in the gradient step arise from vectors in the column span of $\mathbf{J}$, the updates can be computed efficiently, which will yield our main regression result (Theorem 136).

11.3.1 Analysis of the Two-Table Algorithm

In this section, we formally prove the correctness of our algorithm for the fast computation of in-database subspace embeddings for the join of two tables $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$, where $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, and $\mathbf{J} \in \mathbb{R}^{N \times d}$. These algorithms yield input-sparsity runtime solvers for in-database machine precision regression. Our full algorithm is given formally below in Algorithm 12. It uses two crucial subroutines, Algorithms 13 and 14, which efficiently sample rows of a portion $\mathbf{J}_{\text{small}}$ of the design matrix $\mathbf{J}$.

Algorithm 12: Subspace embedding for join $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$.

1. In the *dense* case set $\gamma = 1$. In the *sparse* case, case $\gamma = d$.
2. Compute block sizes $s_{(i)}$, and let $\mathcal{B}_{\text{big}} = \{i \in \mathcal{B} \mid \max_j\{s_{(i),j}\} \geq d \cdot \gamma\}$. Set $\mathcal{B}_{\text{small}} = \mathcal{B} \setminus \mathcal{B}_{\text{big}}$, $n_{\text{small}} = |\mathcal{B}_{\text{small}}|$.
3. For each $i \in \mathcal{B}_{\text{big}}$, generate a Fast Tensor-Sketch matrix $\mathbf{S}^i \in \mathbb{R}^{t \times s_{(i)}}$ (Lemma 11.2.4) and compute $\mathbf{S}^i\mathbf{J}^{(i)}$.
4. Let $\tilde{\mathbf{J}}_{\text{big}}$ be the matrix from stacking the matrices $\{\mathbf{S}^i\mathbf{J}^{(i)}\}_{i \in \mathcal{B}_{\text{big}}}$. Generate a Count-Sketch matrix $\mathbf{S}'$ (Lemma 11.2.6) and compute $\mathbf{S}'\tilde{\mathbf{J}}_{\text{big}}$.
5. Let $\mathbf{J}_{\text{small}} = \mathbf{J}_{\mathcal{B}_{\text{small}}}$, and sample uniformly a subset U of $m = \Theta((n_1 + n_2)/\gamma)$ rows of $\mathbf{J}_{\mathcal{B}_{small}} \in \mathbb{R}^{n_{\text{small}} \times d}$ and form the matrix $\tilde{\mathbf{J}}_{\text{small}} = (\mathbf{J}_{\text{small}})_U \in \mathbb{R}^{m \times d}$.
6. Generate OSNAP transform $\mathbf{W}$ (Lemma 11.2.5) and compute $\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}$ and the SVD $\mathbf{W} \cdot \tilde{\mathbf{J}}_{\text{small}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.
7. Generate Gaussian matrix $\mathbf{G} \in \mathbb{R}^{d \times t}$ with entries drawn i.i.d. from $\mathcal{N}(0, 1/t^2)$, $t = \Theta(\log N)$, and Gaussian vector $g \sim \mathcal{N}(0, \mathbb{I}_d)$.
8. For all rows i of $\mathbf{J}_{\text{small}}$, set $\|(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)g\|_2^2 = \alpha_i$, and
$$\tilde{\tau}_i = \begin{cases} 1 & \text{if } \alpha_i > 0 \\ \|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{G}\|_2^2 & \text{otherwise} \end{cases}.$$
9. Using Algorithms 13 and 14, construct diagonal row sampling matrix $\mathbf{S} \in \mathbb{R}^{n_{\text{small}} \times n_{\text{small}}}$ such that $\mathbf{S}_{i,i} = \frac{1}{\sqrt{p_i}}$ with probability p_i, and $\mathbf{S}_{i,i} = 0$ otherwise, where $p_i \geq \min\left\{1, \frac{\log d}{\epsilon^2} \cdot \tilde{\tau}_i\right\}$
10. Return $\tilde{\mathbf{J}}$, where $\tilde{\mathbf{J}}$ is the result of stacking the matrices $\mathbf{S}'\tilde{\mathbf{J}}_{\text{big}}$ with $\mathbf{S}\mathbf{J}_{\text{small}}$.

We will begin by proving our main technical sampling result, which proceeds in a series of lemmas, and demonstrates that the construct of the diagonal sampling matrix $\mathbf{S}_{\text{small}}$ in Algorithm 12 can be carried out extremely quickly. We first prove a simple proposition.

Proposition 11.3.1. *Let $\mathbf{J}_{small} \in \mathbb{R}^{n_{small} \times d}$ be the matrix constructed as in Algorithm 12. Then we have $n_{small} \leq (n_1 + n_2)d \cdot \gamma$, where $\gamma = 1$ in the dense case and $\gamma = d$ in the sparse case.*

Proof. Recall that $\mathbf{J}_{\text{small}}$ consists of all blocks i of $\mathbf{J}$ with $\max\{s_{(i),1}, s_{(i),2}\} < d\cdot\gamma$, and thus $s_{(i)} \leq d^2\gamma^2$. The total number n_{small} of rows in $\mathbf{J}_{\text{small}}$ is then $\sum_{i\in\mathcal{B}_{\text{small}}} s_{(i),1}\cdot s_{(i),2} \leq \|s^1_{\text{small}}\|_2\|s^2_{\text{small}}\|_2$ by the Cauchy-Schwarz inequality, where s^j_{small} is the vector with coordinates given by the values $s_{(i),j}$ for $i \in \mathcal{B}_{\text{small}}$. Observe that these vectors admit the ℓ_1 bound of $\|s^j_{\text{small}}\|_1 \leq n_j$ since each table $\mathbf{T}_j$ has only n_j rows. Moreover, they admit the ℓ_∞ bound of $\|s^j_{\text{small}}\|_\infty \leq d\gamma$. With these two constraints, it is standard that the ℓ_2 norm is maximized by placing all of the ℓ_1 mass on coordinates with value given by the ℓ_∞ bound. It follows that $\|s^j_{\text{small}}\|_2$ is maximized by having $n_i/(d\gamma)$ coordinates equal to $d\gamma$, giving $\|s^j_{\text{small}}\|_2^2 \leq n_j d\gamma$ for $j \in [2]$, so $n_{\text{small}} \leq \|s^1_{\text{small}}\|_2\|s^2_{\text{small}}\|_2 \leq (n_1+n_2)d\gamma$ as required. □

We now demonstrate how we can quickly ℓ_2 sample rows from a join-vector or join-matrix product after input sparsity time pre-processing. This procedure is split into two algorithms, Algorithm 13 and 14. Algorithm 13 is an input sparsity time pre-processing step, which given $\mathbf{J} \in \mathbb{R}^{n\times d}$ and $\mathbf{Y} \in \mathbb{R}^{d\times t}$, constructs several binary tree data structures. Algorithm 14 then uses these date structures to sample a row of the product $\mathbf{JY}$ with probability proportional to its ℓ_2 norm, in time $O(\log N)$.

The following lemma shows we can compute $\mathbf{SJ}_{\text{small}}$ in input sparsity time using Algorithm 13 and 14. We defer the proof of the Lemma to Section 11.3.2, and first show how our main results follow given Lemma 11.3.2.

Lemma 11.3.2. *Set the value $\gamma = 1$ in the dense case, and $\gamma = d$ in the sparse case. Let $\mathbf{J}_{small} \in \mathbb{R}^{n_{small}\times d}$ and let $\tilde{\mathbf{J}}_{small} \in \mathbb{R}^{m\times d}$ be the subset of rows of $\mathbf{J}_{small}$ constructed in Algorithm 12, where $m = \Theta((n_1+n_2)/\gamma)$. Let $\mathbf{S}$ be the diagonal sampling matrix as constructed in Algorithm 12. Then with probability $1-1/d$, we have that $\mathbf{SJ}_{small}$ is an ϵ-subspace embedding for $\mathbf{J}_{small}$. Moreover, $\mathbf{S}$ has at most $\tilde{O}(d^2\gamma^2/\epsilon^2)$ non-zero entries, and $\mathbf{SJ}_{small}$ can be computed in time $\tilde{O}(\text{nnz}(\mathbf{T}_1)+\text{nnz}(\mathbf{T}_2)+d^3\gamma^2/\epsilon^2)$.*

We now prove our main theorem for in-database subspace embeddings, given Lemma 11.3.2.

Theorem 135 [In-Database Subspace Embedding] *Suppose $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{n\times d}$ is a join of two tables, where $\mathbf{T}_1 \in \mathbb{R}^{n_1\times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2\times d_2}$. Then Algorithm 12 outputs a sketching matrix $\mathbf{S}^* \in \mathbb{R}^{k\times n}$ with $k = \tilde{O}(d^2\gamma^2/\epsilon^2)$ (where γ is chosen as in Lemma 11.3.2) such that $\tilde{\mathbf{J}} = \mathbf{S}^*\mathbf{J}$ is an ϵ-subspace embedding for $\mathbf{J}$, meaning*

$$\|\mathbf{S}^*\mathbf{J}x\|_2^2 = (1\pm\epsilon)\|\mathbf{J}x\|_2^2$$

for all $x \in \mathbb{R}^d$ with probability at least $9/10$. The runtime to return $\mathbf{S}^*\mathbf{J}$ is the minimum of

$\tilde{O}((n_1+n_2)d/\epsilon^2 + d^3/\epsilon^2)$ *and* $\tilde{O}((\text{nnz}(\mathbf{T}_1)+\text{nnz}(\mathbf{T}_2))/\epsilon^2 + d^5/\epsilon^2)$.

Proof. Our algorithm partitions the rows of $\mathbf{J}$ into those from $\mathcal{B}_{\text{small}}$ and $\mathcal{B}_{\text{big}}$, and outputs the result of stacking sketches for $\mathbf{J}_{\text{small}}$ and $\mathbf{J}^{(i)}$ for each $i \in \mathcal{B}_{\text{big}}$. Thus it suffices to show that each sketch $\mathbf{S}^i\mathbf{J}^{(i)}$ is a subspace embedding for $\mathbf{J}^{(i)}$, and $\mathbf{S}\mathbf{J}_{\text{small}}$ is a subspace embedding for $\mathbf{J}_{\text{small}}$. The latter holds by Lemma 11.3.2, and the former follows directly from applying Lemma 11.2.4 and a union bound over the at most $O(n)$ such i. Since each such $\mathbf{J}^{(i)}$ can be written as $\mathbf{J}^i = \left(\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)}\right) \in \mathbb{R}^{s_{(i)} \times d}$, the Fast Tensor-sketch lemma can be applied to $\mathbf{J}^{(i)}$ in time $\tilde{O}((\text{nnz}(\mathbf{T}_1^{(i)})+\text{nnz}(\mathbf{T}_2^{(i)})+d^2)/\epsilon^2)$. Note that $|\mathcal{B}_{\text{big}}| \leq 2(n_1+n_2)/(d\gamma)$, since there can be at most this many values of $s^1_{(i)} + s^2_{(i)} > d\gamma$. Thus the total running time is $\tilde{O}(\sum_{i\in\mathcal{B}_{\text{big}}}(\text{nnz}(\mathbf{T}_1^{(i)}) + \text{nnz}(\mathbf{T}_2^{(i)}) + d^2)/\epsilon^2) = \tilde{O}((\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2))/\epsilon^2 + (n_1+n_2)d/(\gamma\epsilon^2))$. Finally, applying CountSketch will cost $\tilde{O}((n_1+n_2)/(d\gamma)\cdot d/\epsilon^2 \cdot d)$, which is $\tilde{O}((n_1+n_2)d/\epsilon^2)$ for the dense case and $\tilde{O}((n_1+n_2)/\epsilon^2)$ for the sparse case. The remaining runtime analysis follows from Lemma 11.3.2, setting γ to be either 1 or d.

□

We now demonstrate how our subspace embeddings can be easily applied to obtain fast algorithms for regression. To do this, we will first need the following proposition, which shows that $w^T\mathbf{J}^T\mathbf{J}$ can be computed in input sparsity time for any $w \in \mathbb{R}^d$.

Proposition 11.3.3. *Suppose* $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{n\times d}$ *is a join of two tables, where* $\mathbf{T}_1 \in \mathbb{R}^{n_1\times d_1}, \mathbf{T}_2 \in \mathbb{R}^{n_2\times d_2}$. *Let* $b \in \mathbb{R}^n$ *be any column of the join* $\mathbf{J}$. *Let* $U \subseteq [d]$ *be any subset, and let* $\mathbf{J}_U \in \mathbb{R}^{n\times|U|}$ *be the subset of the columns of* $\mathbf{J}$ *contained in* $\mathbf{U}$. *Let* $w \in \mathbb{R}^d$ *be any vector, and let* $x = \mathbf{J}w \in \mathbb{R}^n$. *Then given* w, *the vector* $x^T\mathbf{J}_U = w^T\mathbf{J}^T\mathbf{J}_U \in \mathbb{R}^{|U|}$ *can be computed in time* $O(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2))$.

Proof. Fix any $i \in \mathcal{B}(\mathbf{J})$, and similarly let $\mathbf{J}_U^{(i)}, \hat{\mathbf{T}}_{1,U}^{(i)}, \hat{\mathbf{T}}_{2,U}^{(i)}$ be $\mathbf{J}^{(i)}, \hat{\mathbf{T}}_1^{(i)}, \hat{\mathbf{T}}_2^{(i)}$ respectively, restricted to the columns of U. Note that we have $\mathbf{J}_U^{(i)} = (\hat{\mathbf{T}}_{1,U}^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),2}} \otimes \hat{\mathbf{T}}_{2,U}^{(i)}) \in \mathbb{R}^{s_i\times|U|}$. Let $x^{(i)} \in \mathbb{R}^{s_{(i)}}$ be x restricted to the rows inside of block $i \in \mathcal{B}(\mathbf{J})$. Since $x = \mathbf{J}w$ for some

$w \in \mathbb{R}^d$, we have $x^{(i)} = \mathbf{J}^{(i)}w = (\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)})w$. Then we have

$$\begin{aligned} x^T\mathbf{J}_U &= \sum_{i\in\mathcal{B}}(x^{(i)})^T\mathbf{J}_U^{(i)} \\ &= \sum_{i\in\mathcal{B}} w^T\left(\hat{\mathbf{T}}_1^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_2^{(i)}\right)^T \\ &\cdot \left(\hat{\mathbf{T}}_{1,U}^{(i)} \otimes \mathbf{1}^{s_{(i),2}} + \mathbf{1}^{s_{(i),1}} \otimes \hat{\mathbf{T}}_{2,U}^{(i)}\right) \\ &= \sum_{i\in\mathcal{B}} w^T\Big(s_{(i),2}(\hat{\mathbf{T}}_1^{(i)})^T\hat{\mathbf{T}}_{1,U}^{(i)} + ((\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}}) \\ &\otimes ((\mathbf{1}^{s_{(i),2}})^T\hat{\mathbf{T}}_{2,U}^{(i)}) + (\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_{1,U}^{(i)} \otimes ((\hat{\mathbf{T}}_2^{(i)})^T\mathbf{1}^{s_{(i),2}}) \\ &+ s_{(i),1}(\hat{\mathbf{T}}_2^{(i)})^T\hat{\mathbf{T}}_{2,U}^{(i)}\Big) \end{aligned} \tag{11.1}$$

where the last equality follows from the mixed product property of Kronecker products (see e.g., [VL00]). First note that the products $w^T s_{(i),2}(\hat{\mathbf{T}}_1^{(i)})^T\hat{\mathbf{T}}_{1,U}^{(i)}$ and $w^T s_{(i),1}(\hat{\mathbf{T}}_2^{(i)})^T\hat{\mathbf{T}}_{2,U}^{(i)}$ can be computed in $O(\text{nnz}(\mathbf{T}_1^{(i)}) + \text{nnz}(\mathbf{T}_2^{(i)}))$ time by computing the vector matrix product first. Thus, it suffices to show how to compute $w^T((\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_{1,U}^{(i)}) \otimes ((\hat{\mathbf{T}}_2^{(i)})^T\mathbf{1}^{s_{(i),2}})$ and $w^T((\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}}) \otimes ((\mathbf{1}^{s_{(i),2}})^T\hat{\mathbf{T}}_{2,U}^{(i)})$ quickly. By reshaping the Kronecker products [VL00], we have

$$w^T(\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_1^{(i)} \otimes ((\hat{\mathbf{T}}_{2,U}^{(i)})^T\mathbf{1}^{s_{(i),2}}) = ((\hat{\mathbf{T}}_{2,U}^{(i)})^T\mathbf{1}^{s_{(i),2}}w^T(\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}})^T$$

Now $w^T(\hat{\mathbf{T}}_1^{(i)})^T$ can be computed in $O(\text{nnz}(\mathbf{T}_1^{(i)}))$ time, at which point $w^T(\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}}$ can be computed in $O(s_{(i),1}) = O(\text{nnz}(\mathbf{T}_1^{(i)}))$ time. Next, we can compute $\mathbf{1}^{s_{(i),2}}(w^T(\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}})$ in $O(s_{(i),2}) = O(\text{nnz}(\mathbf{T}_2^{(i)}))$ time. Finally, $(\hat{\mathbf{T}}_{2,U}^{(i)})^T(\mathbf{1}^{s_{(i),2}}w^T \cdot (\hat{\mathbf{T}}_1^{(i)})^T\mathbf{1}^{s_{(i),1}})$ can be computed in $O(\text{nnz}(\mathbf{T}_2^{(i)}))$ time. A similar argument holds for computing $w^T(\mathbf{1}^{s_{(i),1}})^T\hat{\mathbf{T}}_{1,U}^{(i)} \otimes ((\hat{\mathbf{T}}_2^{(i)})^T\mathbf{1}^{s_{(i),2}})$, which completes the proof, noting that

$$\sum_{i\in\mathcal{B}} \text{nnz}(\mathbf{T}_1^{(i)}) + \text{nnz}(\mathbf{T}_2^{(i)}) = \text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2)$$

□

We now state our main theorem for machine precision regression. We remark again that the success probability can be boosted to $1-\delta$ by boosting the success probability of the corresponding subspace embedding to $1-\delta$, as described earlier.

Theorem 136 [In-Database Regression (Theorem 136)] *Suppose* $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{n\times d}$

is a join of two tables, where $T_1 \in \mathbb{R}^{n_1 \times d_1}, T_2 \in \mathbb{R}^{n_2 \times d_2}$. *Let* $U \subset [d]$ *be any subset, and let* $\mathbf{J}_U \in \mathbb{R}^{N \times |U|}$ *be* $\mathbf{J}$ *restricted to the columns in* U, *and let* $b \in \mathbb{R}^n$ *be any column of the join* $\mathbf{J}$. *Then there is an algorithm which returns* $\hat{x} \in \mathbb{R}^{|U|}$ *such that with probability* $9/10$ *we have*

$$\|\mathbf{J}_U \hat{x} - b\|_2 \leq (1+\epsilon) \min_{x \in \mathbb{R}^{|U|}} \|\mathbf{J}_U x - b\|_2$$

The runtime required to compute $\hat{x}$ *is the minimum of*

$$\tilde{O}\left(((n_1+n_2)d + d^3) + (\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2) + d^2)\log(1/\epsilon)\right)$$

and

$$\tilde{O}\left(d^5 + (\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2) + d^2)\log(1/\epsilon)\right)$$

Proof. The following argument follows a standard reduction from having a subspace embedding to obtaining high precision regression (see, e.g., Section 2.6 of [W+14]). We first compute a subspace embedding $\tilde{\mathbf{J}}_= \mathbf{S}^* \mathbf{J}$ for $\mathbf{J}$ via Theorem 135 with precision parameter $\epsilon_0 = \Theta(1)$, so that $\mathbf{S}^*\mathbf{J}$ has $k = \tilde{O}(d^2\gamma^2)$ rows. Note that in particular this implies that $\mathbf{S}^*\mathbf{J}_U$ is an ϵ_0-subspace embedding for $\mathbf{J}_U$. We then generate an OSNAP matrix $\mathbf{W} \in \mathbb{R}^{\tilde{\Theta}(d) \times k}$ via Lemma 11.2.5 with precision ϵ_0, and condition on the fact that $\mathbf{W}\mathbf{S}^*\mathbf{J}_U$ is an ϵ_0-subpsace embedding for $\mathbf{S}^*\mathbf{J}_U$, which holds with large constant probability, from which it follows that $\mathbf{W}\mathbf{S}^*\mathbf{J}_U$ is an $O(\epsilon_0)$-subspace embedding for $\mathbf{J}_U$. We then compute the QR factorization $\mathbf{W}\mathbf{S}^*\mathbf{J}_U = \mathbf{Q}\mathbf{R}^{-1}$, which can be done in $O(d^\omega)$ time via fast matrix multiplication [DDH07]. By standard arguments [W+14], the matrix $\mathbf{J}_U\mathbf{R}$ is now $O(1)$-well conditioned – namely, we have $\sigma_{\max}(\mathbf{J}_U\mathbf{R})/\sigma_{\min}(\mathbf{J}_U\mathbf{R}) = O(1)$. Given this, we can apply the gradient descent update $x_{t+1} \leftarrow x_t + \mathbf{R}^T\mathbf{J}_U^T(b - \mathbf{J}_U\mathbf{R}x_t)$, which can be computed in $O(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2) + d^2)$ time via Proposition 11.3.3 (note we compute $\mathbf{R}x_t$ in $O(d^2)$ time first, and then compute $(\mathbf{J}_U\mathbf{R}x_t)$. Here we use the fact that $b - \mathbf{J}_U\mathbf{R}x_t = \mathbf{J}w$ for some w is a vector in the column span of $\mathbf{J}$, and moreover we can determine the value of w from computing $\mathbf{R}x_t$ and noting that $b = \mathbf{J}e_{j^*}$, where $j^* \in [d]$ is the index of b in $\mathbf{J}$. Since $\mathbf{R}^T\mathbf{J}_U^T$ is now well -conditioned, gradient descent now converges in $O(\log 1/\epsilon)$ iterations given that we have a constant factor approximation x_0 [W+14]. Specifically, it suffices to have an $x_0 \in \mathbb{R}^d$ such that $\|\mathbf{J}_U x_0 - b\|_2 \leq (1+\epsilon_0)\min_{x \in \mathbb{R}^d} \|\mathbf{J}_U x - b\|_2$. But recall that such an x_0 can be obtained by simply solving $x_0 = \arg\min_x \|\mathbf{S}^*\mathbf{J}_U x - \mathbf{S}^* b\|_2$, using the fact that $\mathbf{S}^*$ is an ϵ_0-subspace embedding for the span of $\mathbf{J}$, which completes the proof of the theorem. □

Algorithm 13: Pre-processing step for fast ℓ_2 sampling from rows of $\mathbf{J} \cdot \mathbf{Y}$, where $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2$ and $\mathbf{Y} \in \mathbb{R}^{d \times r}$.

For each block $i \in \mathcal{B}$, compute $a_{(i)} = \hat{T}_1^{(i)}\mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$

For each block $i \in \mathcal{B}$, construct a binary tree $\tau_{(i)}(\mathbf{T}_1), \tau_{(i)}(\mathbf{T}_2)$ as follows:

Each node $v \in \tau_{(i)}(\mathbf{T}_j)$ is a vector $v \in \mathbb{R}^{3r}$

If v is not a leaf, then $v = v_{\text{lchild}} + v_{\text{rchild}}$ with $v_{\text{lchild}}, v_{\text{rchild}}$ the left and right children of v.

The leaves of $\tau_{(i)}(\mathbf{T}_j)$ are given by the set

$$\{v_l^{(i),j} = \left(v_{l,1}^{(i),j}, v_{l,2}^{(i),j}, \ldots, v_{l,r}^{(i),j}\right) \in \mathbb{R}^{3r} \mid l \in [s_{(i),j}]\}$$

where $v_{l,q}^{(i),1} = (1, 2(a_{(i)})_{l,q}, (a_{(i)})_{l,q}^2) \in \mathbb{R}^3$ and $v_{l,q}^{(i),2} = ((b_{(i)})_{l,q}^2, (b_{(i)})_{l,q}, 1) \in \mathbb{R}^3$.

Compute the values $\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$ for all $i \in \mathcal{B}$, and also compute the sum $\sum_i \langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$.

Algorithm 14: Sampling step for fast ℓ_2 sampling from rows of $\mathbf{J} \cdot \mathbf{Y}$.

Sample a block $i \in \mathcal{B}$ with probability

$$\frac{\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle}{\sum_j \langle \text{root}(\tau_j(\mathbf{T}_1)), \text{root}(\tau_j(\mathbf{T}_2))\rangle}$$

Sample $l_1 \in [s_{(i),1}]$ with probability

$$p_{l_1} = \frac{\left\langle v_{l_1}^{(i),1}, \text{root}(\tau_i(\mathbf{T}_2))\right\rangle}{\sum_l \left\langle v_l^{(i),1}, \text{root}(\tau_j(\mathbf{T}_2))\right\rangle}$$

Sample $l_2 \in [s_{(i),2}]$ with probability

$$p_{l_2 \mid l_1} = \frac{\left\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\right\rangle}{\sum_l \left\langle v_{l_1}^{(i),1}, v_l^{(i),2}\right\rangle}$$

Return: the row corresponding to (l_1, l_2) in block i.

11.3.2 Proof of Lemma 11.3.2

We start our proof by showing Algorithm 14 can sample one row with probability according to the ℓ_2 norm quickly after running the pre-processing Algorithm 13.

Lemma 11.3.4. *Let $\mathbf{J} = \mathbf{T}_1 \bowtie \mathbf{T}_2 \in \mathbb{R}^{N \times d}$ be any arbitrary join on two tables, with $\mathbf{T}_i \in \mathbb{R}^{n_i \times d_i}$, and fix any $\mathbf{Y} \in \mathbb{R}^{d \times t}$. Then Algorithms 13 and 14, after a $O((\mathrm{nnz}(\mathbf{T}_1) + \mathrm{nnz}(\mathbf{T}_2))(t + \log N))$-time pre-processing step (Algorithm 13), can produce samples $i^* \sim \mathcal{D}_{\mathbf{Y}}$ (Algorithm 14) from the distribution $\mathcal{D}_{\mathbf{Y}}$ over $[N]$ given by*

$$Pr_{i^* \sim \mathcal{D}_{\mathbf{Y}}}[i^* = j] = \frac{\|(\mathbf{J} \cdot \mathbf{Y})_{j,*}\|_2^2}{\|\mathbf{J} \cdot \mathbf{Y}\|_F^2}$$

such that each sample is produced in $O(\log N)$ time.

Proof. We begin by arguing the correctness of Algorithms 13 and 14. Let j^* be any row of $\mathbf{J} \cdot \mathbf{Y}$. Note that the row j^* corresponds to a unique block $i \in \mathcal{B} = \mathcal{B}(\mathbf{J})$, and two rows $l_1 \in [s_{(i),1}], l_2 \in [s_{(i),2}]$, such that $\mathbf{J}_{j^*,*} = (\hat{T}_1)_{l_1',*} + (\hat{T}_2)_{l_2',*}$, where $l_1' \in [n_1], l_2' \in [n_2]$ are the indices which correspond to l_1, l_2. For $l \in [s_{(i),j}]$, let $v_l^j, v_{l,q}^j$ be defined as in Algorithm 13. We first observe that if j^* corresponds to the block $i \in \mathcal{B}$, then $\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2} \rangle = \|(\mathbf{JY})_{j^*,*}\|_2^2$, since

$$\begin{aligned}
\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2} \rangle &= \sum_{q=1}^{r} v_{l_1,q}^{(i),1} \cdot v_{l_2,q}^{(i),2} \\
&= \sum_{q=1}^{r} (a_{(i)})_{l_1,q}^2 + 2(a_{(i)})_{l_1,q}(b_{(i)})_{l_2,q} + (b_{(i)})_{l_2,q}^2 \\
&= \sum_{q=1}^{r} \left((a_{(i)})_{l_1,q} + (b_{(i)})_{l_2,q} \right)^2 \\
&= \|(\mathbf{JY})_{j^*,*}\|_2^2
\end{aligned} \tag{11.2}$$

where for each block $i \in \mathcal{B}$, we compute $a_{(i)} = \hat{T}_1^{(i)} \mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$ as defined in Algorithm 13. Thus it suffices to sample a row j^*, indexed by the tuple (i, l_1, l_2) where $i \in \mathcal{B}, l_1 \in [s_{(i),1}], l_2 \in [s_{(i),2}]$, such that the probability we sample j^* is given by $p_{j^*} = \langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2} \rangle / (\sum_{i',l_1',l_2'} \langle v_{l_1'}^{(i'),1}, v_{l_2'}^{(i'),2} \rangle)$. We argue that Algorithm 14 does precisely this. First note that for any $i \in \mathcal{B}$, we have

$$\langle v_{l_1}^{(i),1}, \mathrm{root}(\tau_i(\mathbf{T}_2)) \rangle = \sum_{l_1 \in [s_{(i),1}], l_2 \in [s_{(i),2}]} \langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2} \rangle$$

Thus we first partition the set of all rows j^* by sampling a block i with probability

$$\frac{\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle}{\sum_j \langle \text{root}(\tau_j(\mathbf{T}_1)), \text{root}(\tau_j(\mathbf{T}_2))\rangle}$$

which is exactly the distribution over blocks induced by the ℓ_2 mass of the blocks. Conditioned on sampling $i \in \mathcal{B}$, it suffices now to sample l_1, l_2 from that block. To do this, we first sample l_1 with probability $\langle v_{l_1}^{(i),1}, \text{root}(\tau_i(\mathbf{T}_2))\rangle / (\sum_l \langle v_l^{(i),1}, \text{root}(\tau_j(\mathbf{T}_2))\rangle)$, which is precisely the distribution over indices $l_1 \in [s_{(i),1}]$ induced by the contribution of l_1 to the total ℓ_2 mass of block i. Similarly, once conditioned on l_1, we sample l_2 with probability $\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2}\rangle / (\sum_l \langle v_{l_1}^{(i),1}, v_l^{(i),2}\rangle)$, which is the distribution over indices $l_2 \in [s_{(i),2}]$ induced by the contribution of the row (l_1, l_2), taken over all l_2 with l_1 fixed. Taken together, the resulting sample $j^* \cong (i, l_1, l_2)$ is drawn from precisely the desired distribution.

Finally, we bound the runtime of this procedure. First note that computing $a_{(i)} = \hat{T}_1^{(i)}\mathbf{Y}$ and $b_{(i)} = \hat{T}_2^{(i)}$ for all blocks i can be done in $O(t(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2)))$ time, since each row of the tables $\mathbf{T}_1, \mathbf{T}_2$ is in exactly one of the blocks, and each row is multiplied by exactly t columns of $\mathbf{Y}$. Once the $a_{(i)}, b_{(i)}$ are computed, each tree $\tau_{(i)}(\mathbf{T}_j)$ can be computed bottom up in time $O(\log N)$, giving a total time of $O(\log N(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2)))$ for all trees. Given this, the values $\langle \text{root}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$ can be computed in less than the above runtime. Thus the total pre-processing time is bounded by $O((t + \log N)(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2)))$ as needed. For the sampling time, it then suffices to show that we can carry out Lines 2 and 3 in $O(\log N)$ time. But these samples can be samples from the root down, by first computing $\langle \text{root}_{\text{lchild}}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$ and $\langle \text{root}_{\text{rchild}}(\tau_i(\mathbf{T}_1)), \text{root}(\tau_i(\mathbf{T}_2))\rangle$, sampling one of the left or right children with probability proportional to its size, and recursing into that subtree. Similarly, l_2 can be sampled by first computing $\langle v_{l_1}^{(i),1}, \text{root}_{\text{lchild}}(\tau_i(\mathbf{T}_2))\rangle$ and $\langle v_{l_1}^{(i),1}, \text{root}_{\text{rchild}}(\tau_i(\mathbf{T}_2))\rangle$ sampling one of the left or right children with probability proportional to its size, and recursing into that subtree. This completes the proof of the $O(\log N)$ runtime for sampling after pre-processing has been completed. □

Then we show how we can construct $\mathbf{S}$ by invoking Algorithm 13 and 14.

Lemma 11.3.5. *Let $\mathbf{J}_{small} \in \mathbb{R}^{n_{small} \times d}$ be the matrix constructed as in Algorithm 12 in the dense case. Then the diagonal sampling matrix $\mathbf{S}$, as defined within lines 5 through 9 of Algorithm 12, can be constructed in time $\tilde{O}(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2) + d^2\gamma^2/\epsilon^2)$.*

Proof. We first show how we can quickly construct the matrix $\tilde{\mathbf{J}}_{\text{small}}$, which consists of $m = \Theta((n_1 + n_2)/\gamma)$ uniform samples from the rows of $\mathbf{J}_{\text{small}}$. First, to sample the rows uniformly,

since we already know the sizes of each block $s_{(i)}$, we can first sample a block $i \in \mathcal{B}_{\text{small}}$ with probability proportional to its size, which can be done in $O(\log(|\mathcal{B}_{\text{small}}|)) = O(\log N)$ time after the $s_{(i)}$'s are computed. Next, we can sample a row uniformly from T_i^j for each $j \in [2]$, and output the join of the two chosen rows, the result of which is a truly uniform row from $\mathbf{J}_{\text{small}}$. Since we need m samples, and each sample has d columns, the overall runtime is $\tilde{O}((n_1 + n_2)d/\gamma)$ to construct $\tilde{\mathbf{J}}_{\text{small}}$, which is $O(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2))$ in the sparse case.

Once we have $\tilde{\mathbf{J}}_{\text{small}}$, we compute in line 6 of Algorithm 12 the sketch $\mathbf{W} \cdot \tilde{\mathbf{J}}_{\text{small}}$, where $\mathbf{W} \in \mathbb{R}^{t \times d}$ is the OSNAP Transformation of Lemma 11.2.5 with $\epsilon = 1/100$, where $t = \tilde{O}(d)$, which we can compute in $\tilde{O}(md) = \tilde{O}((n_1 + n_2)d/\gamma)$ time by Lemma 11.2.5. Given this sketch $\mathbf{W} \cdot \tilde{\mathbf{J}}_{\text{small}} \in \mathbb{R}^{t \times d}$, the SVD of the sketch can be computed in time $O(d^\omega)$ [DDH07], where $\omega < 2.373$ is the exponent of fast matrix multiplication. Since $\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}$ is a $1/100$ subspace embedding for $\tilde{\mathbf{J}}$ with probability $99/100$ by Lemma 11.2.5, by Proposition 11.2.3 we have $\tau^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{A}) = (1 \pm 1/100)\tau^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{A})$ for any matrix $\mathbf{A}$. Next, we can compute $\mathbf{V}\mathbf{\Sigma}\mathbf{G}$ in the same $O(d^\omega)$ runtime, where $\mathbf{G} \in \mathbb{R}^{d \times t}$ is a Gaussian matrix with $t = \Theta(\log N)$ and with entries drawn independently from $\mathcal{N}(0, 1/t^2)$. By standard arguments for Johnson Lindenstrauss random projections (see e.g., Lemma 4.5 of [LMP13]), we have that $\|(x^T\mathbf{G})\|_2^2 = (1 \pm 1/100)\|x\|_2^2$ for any fixed vector $x \in \mathbb{R}^d$ with probability at least $1 - n^{-c}$ for any constant $c \geq 1$ (depending on t).

We now claim that $\tilde{\tau}_i$ as defined in Algorithm 12 satisfies $C^{-1}\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ for some fixed constant $C \geq 1$. As noted above, $\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) = (1 \pm 1/100)\tau^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{A})$, so it suffices to show $C^{-1}\tau_i^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\mathbf{W}\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$. To see this, first note that if $(\mathbf{J}_{\text{small}})_{i,*}$ is contained within the row span of $\tilde{\mathbf{J}}_{\text{small}}$, then

$$\begin{aligned}
\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) &= \|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\mathbf{\Sigma}^{-1}\|_2^2 \\
&= (1 \pm 1/100)\|(\mathbf{J}_{\text{small}})_{i,*}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{G}\|_2^2 \\
&= (1 \pm 1/100)\tilde{\tau}_i
\end{aligned} \tag{11.3}$$

Thus it suffices to show that if $(\mathbf{J}_{\text{small}})_{i,*}$ has a component outside of the span of $\tilde{J}_{\text{small}}$, then we have $\|(\mathbf{J}_{\text{small}})_{i,*}(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T)g\|_2^2 > 0$. To see this, note that $\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right)$ is the projection onto the orthogonal space to the span of $\tilde{\mathbf{J}}_{\text{small}}$. Thus $\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g$ is a random non-zero vector in the orthogonal space of $\tilde{\mathbf{J}}_{\text{small}}$, thus $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g \neq 0$ almost surely if $(\mathbf{J}_{\text{small}})_{i,*}$ has a component outside of the span of $\tilde{\mathbf{J}}_{\text{small}}$, which completes the proof of the claim.

Finally, and most significantly, we show how to implement line 9 of Algorithm 12, which carries out the the construction of $\mathbf{S}$. Given that $1/C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$, we can apply Theorem 1 of [CLM+15], using that $n_{\text{small}} \leq (n_1 + n_2)d \cdot \gamma$ via Proposition 11.3.1, which

yields that $\sum_i \tilde{\tau}_i = O(n_{\text{small}} \frac{d}{m}) = O(d^2\gamma^2)$. Thus to construct the sampling matrix $\mathbf{S}$, it suffices to sample $\alpha = O(d^2\gamma^2 \log d/\epsilon^2)$ samples from the distribution over the rows i of $\mathbf{J}_{\text{small}}$ given by $q_i = \frac{\tilde{\tau}_i}{\sum_i \tilde{\tau}_i}$. We now describe how to accomplish this.

We first show how to sample the rows i with $(\mathbf{J}_{\text{small}})_{i,*}(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T)g = 0$. To do this, it suffices to sample rows from the distribution induced by the ℓ_2 norm of the rows of $(\mathbf{J}_{\text{small}})\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{G}$. To do this, we can simply apply Algorithms 13 and 14 to the product $\mathbf{J}_{\text{small}} \cdot (\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{G})$. First note that we can do this because $\mathbf{J}_{\text{small}}$ itself is a join of $(\mathbf{T}_1)_{\text{small}}$ and $(\mathbf{T}_2)_{\text{small}}$, which are just $\mathbf{T}_1, \mathbf{T}_2$ with all the rows contained in blocks $i \in \mathcal{B}_{\text{big}}$ removed. Since $\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{G} \in \mathbb{R}^{d\times t}$ for $t = \tilde{O}(1)$, by Lemma 11.3.4 after $\tilde{O}(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2))$, for any $s \geq 1$ we can sample s times independently from this induced ℓ_2 distribution over rows in time $\tilde{O}(s)$. Altogether, we obtain the required $\alpha = O(d^2\gamma^2 \log d/\epsilon^2)$ samples from the distribution over the rows i of $\mathbf{J}_{\text{small}}$ given by $q_i = \frac{\tilde{\tau}_i}{\sum_i \tilde{\tau}_i}$ in the case that $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g = 0$, with total runtime $\tilde{O}(\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2) + d^2\gamma^2/\epsilon^2)$.

Finally, for the case that $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g > 0$, we can apply the same Algorithms 13 and 14 to sample from the rows of $\mathbf{J}_{\text{small}} \cdot \left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g$. First observe, using the fact that $\sum_i \tilde{\tau}_i \leq O(d^2\gamma^2)$ by Theorem 1 of [CLM+15], it follows that there are at most $O(d^2\gamma^2)$ indices i such that $\mathbf{J}_{\text{small}} \cdot \left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g > 0$. Thus, when applying Algorithm 14 after the pre-processing step is completed, instead of sampling independently from the distribution induced by the norms of the rows, we can deterministically find all rows with $\mathbf{J}_{\text{small}} \cdot \left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g > 0$ in $\tilde{O}(d^2\gamma^2)$ time, by simply enumerating over all computation paths of Algorithm 14 that occur with non-zero probability. Since there are $O(d^2\gamma^2)$ such paths, and each one is carried out in $O(\log N)$ time by Lemma 11.3.4, the resulting runtime is the same as the case where $(\mathbf{J}_{\text{small}})_{i,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g = 0$.

Finally, we argue that we can compute exactly the probabilities p_j with which we sampled a row j of $\mathbf{J}_{\text{small}}$, for each i that was sampled, which will be needed to determine the scalings of the rows of $\mathbf{J}_{\text{small}}$ that are sampled. For all the rows sampled with $(\mathbf{J}_{\text{small}})_{j,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g > 0$, the corresponding value of p_j is by definition 1. Note that if a row was sampled in both of the above cases, then it should in fact have been sampled in the case that $(\mathbf{J}_{\text{small}})_{j,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g > 0$, so we set $p_j = 1$. For every row j sampled via Algorithm 14 when $(\mathbf{J}_{\text{small}})_{j,*}\left(\mathbb{I}_d - \mathbf{V}\mathbf{V}^T\right) g = 0$, such that j corresponds to the tuple (i, l_1, l_2) where $i \in \mathcal{B}_{\text{small}}$ and $l_1 \in [s_{(i),1}], l_2 \in [s_{(i),2}]$, we can compute the probability it was sampled exactly via

$$p_j = p_{(i,l_1,l_2)} = \frac{\langle v_{l_1}^{(i),1}, v_{l_2}^{(i),2} \rangle}{\sum_{c\in\mathcal{B}} \langle \text{root}(\tau_c(\mathbf{T}_1)), \text{root}(\tau_c(\mathbf{T}_2)) \rangle}$$

using the notation in Algorithm 14. Then setting $\mathbf{S}_{j,j} = \frac{1}{\sqrt{p_j}}$ for each sampled row j between

both processes yields the desired construction of $\mathbf{S}$. □

With the above lemma in hand, we can give the proof of Lemma 11.3.2.

Proof of lemma 11.3.2. As argued in the proof of Lemma 11.3.5, we have that $(1/C)\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}}) \leq \tilde{\tau}_i \leq C\tau_i^{\tilde{\mathbf{J}}_{\text{small}}}(\mathbf{J}_{\text{small}})$ for some fixed constant $C \geq 1$ and all $i \in \mathcal{B}_{\text{small}}$. The result then follows immediately from Theorem 4 of [CLM+15], using the fact that $n_{\text{small}}/m = \tilde{O}(d\gamma^2)$, where m is the number of rows subsampled in $\tilde{\mathbf{J}}_{\text{small}}$ from $\mathbf{J}_{\text{small}}$. □

11.4 General Join Queries

Below we introduce DB-Sketch as a class of algorithms, and show how any oblivious sketching algorithm that has the properties of DB-Sketch can be implemented efficiently for data coming from a join query. Since the required properties are very similar to the properties of linear sketches for Kronecker products, we will be able to implement them inside of a database. Given that the statistical dimension can be much smaller than the actual dimensions of the input data, our time complexity for ridge regression can be significantly smaller than that for ordinary least squares regression, which is important in the context of joins of many tables. We first begin with a primer on useful database concepts.

11.4.1 Database Background

We begin with some fundamental definitions relating to database joins.

Definition 11.4.1 (Join Hypergraph)**.** *Given a join $\mathbf{J} = \mathbf{T}_1 \bowtie \cdots \bowtie \mathbf{T}_m$, the hypergraph associated with the join is $H = (V, E)$ where V is the set of vertices and for every column c_i in J, there is a vertex v_i in V, and for every table $\mathbf{T}_i$ there is a hyper-edge e_i in E that has the vertices associated with the columns of $\mathbf{T}_i$.*

Definition 11.4.2 (Acyclic Join)**.** *We call a join query **acyclic** if one can repeatedly apply one of the two operations and convert the query to an empty query:*

1. *remove a column that is only in one table.*
2. *remove a table for which its columns are fully contained in another table.*

Definition 11.4.3 (Hypergraph Tree Decomposition)**.** *Let $H = (V, E)$ be a hypergraph and $T = (V', E')$ be a tree on a set of vertices, where each vertex $v' \in V'$ called the **bag** of v', denoted by $b(v')$, and corresponds to a subset of vertices of V. Then T is called a **hypergraph tree decomposition** of H if the following holds:*

1. *for each hyperedge $e \in E$, there exists $v' \in V'$ such that $e \subseteq b(v')$, and*
2. *for each vertex $v \in V$, the set of vertices in V' that have v in their bag is non-empty and they form a connected subtree of T.*

Definition 11.4.4. *Let $H = (V, E)$ be a join hypergraph and $T = (V', E')$ be its tree decomposition. For each $v' \in V'$, let $X^{v'} = (x_1^{v'}, x_2^{v'}, \ldots, x_m^{v'})$ be the optimal solution to the following linear program: $\texttt{min} \sum_{j=1}^{t} x_j$, subject to $\sum_{j:v_i \in e_j} x_j \geq 1, \forall v_i \in b(v')$ where $0 \leq x_j \leq 1$ for each $j \in [t]$. Then the **width of** v' is $\sum_i x_i^{v'}$, denoted by $w(v')$, and the **fractional width of** T is $\max_{v' \in V'} w(v')$.*

Definition 11.4.5 (fhtw)**.** *Given a join hypergraph $H = (V, E)$, the **fractional hypertree width of** H, denoted by fhtw, is the minimum fractional width of its hypergraph tree decomposition. Here the minimum is taken over all possible hypertree decompositions.*

Observation 137. The fractional hypertree width of an acyclic join is 1, and each bag in its hypergraph tree decomposition is a subset of the columns in some input table.

Definition 11.4.6 (FAQ)**.** *Let $J = T_1 \bowtie \cdots \bowtie T_m$ be a join of m input tables. For each table T_i, let $F_i : T_i \to S$ be a function mapping the rows of T_i to a set S. For every row $X \in J$, let X_i be the projection of X onto the columns of T_i. Then the following is a SumProd Functional Aggregation Query (FAQ):*

$$\bigoplus_{X \in J} \bigotimes_{i} F_i(X_i) \tag{11.4}$$

where $(S, \oplus, \otimes)$ is a commutative semiring.

Theorem 138 ([AKNR16])**.** *Inside-out is an algorithm which computes the result of a FAQ in time $O(Tmd^2 n^{fhtw} \log(n))$ where m is the number of tables, d is the number of columns in J, n is the maximum number of rows in any input table, T is the time to compute the operators $\oplus$ and $\otimes$ on a pair of operands, and $fhtw$ is the fractional hypertree width of the query.*

In [AKNN+18], given a join $J = T_1 \bowtie \cdots \bowtie T_m$, it is shown that the entries of $J^T J$ can be

expressed as a FAQ and computed using the inside-out algorithm.

11.4.2 The General Algorithm

In the following we assume the join query is acyclic; nevertheless, for cyclic queries it is possible to obtain the hypergraph tree decomposition of the join and create a table for each vertex in the tree decomposition by joining the input tables that are a subset of the vertex's bag in the hypergraph tree decomposition. One can then replace the cyclic join query with an acyclic query using the new tables.

In our algorithm, we use FAQ and inside-out algorithm as a subroutine. The definition of FAQ is given in Section 11.4.1. Let $\mathbf{J}' = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m$ be an acyclic join. Let ρ be a binary expression tree that shows in what order the algorithm inside-out [AKNR16] multiplies the factors for an arbitrary single-semiring FAQ; meaning, ρ has a leaf for each factor (each table) and $(m-1)$ internal nodes such that if two nodes have the same parent then their values are multiplied together during the execution of inside-out. We number the tables based on the order that a depth-first-search visits them in this expression tree. We let ρ denote the multiplication order of $\mathbf{J}'$.

Now that the ordering of the tables is fixed, we reformulate the Join table $\mathbf{J}'$ so that it can be expressed as a summation of tensor products. Assign each column c to one of the input tables that has c, and then let E_i denote the columns assigned to table $\mathbf{T}_i$, X_i be the projection of X onto E_i, and D_i be the domain of the tuples X_i (projection of $\mathbf{T}_i$ onto E_i).

Letting $N = |D_1||D_2|\dots|D_m|$, we can reformulate $\mathbf{J}'$ as $\mathbf{J} \in \mathbb{R}^{N\times d}$ to have a row for any possible tuple $X \in D_1 \times \cdots \times D_m$. If a tuple X is present in the join, we put its value in the row corresponding to it, and if it is not present we put 0 in that row. Note that $\mathbf{J}$ has all the rows in $\mathbf{J}'$ and also may have many zero rows; however, we do not need to present $\mathbf{J}$ explicitly, and the sparsity of $\mathbf{J}$ does not cause a problem. One key property of this formulation is that by knowing the values of a tuple x, the location of x in $\mathbf{J}$ is well-defined. Also note that since we have only added rows that are 0, any subspace embedding of $\mathbf{J}$ would be a subspace embedding of $\mathbf{J}'$, and for all vectors x, $\|\mathbf{J}x\|_2^2 = \|\mathbf{J}'x\|_2^2$.

Given a join query $\mathbf{J}$ with m tables and its multiplication order ρ, an oblivious sketching algorithm is an (m,ρ)-DB-Sketch if there exists a function $F : \mathbb{R}^{n\times d} \to R_F$ where R_F is the range of F and $F(\mathbf{A})$ represents the sketch of $\mathbf{A}$ in some form and has the following properties:

1. $F(\mathbf{A}_1 + \mathbf{A}_2) = F(\mathbf{A}_1) \oplus F(\mathbf{A}_2)$ where $\oplus$ is a commutative and associative operator.

2. For any V resulting from a Kronecker product of matrices $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_m \in \mathbb{R}^{n_1 \ldots n_m}$, $F(\mathbf{A}) = F_1(\mathbf{A}_1) \odot F_2(\mathbf{A}_2) \odot \cdots \odot F_m(\mathbf{A}_m)$ where $\odot$ is applied based on the ordering in ρ, and for all i, F_i has the same range as F, and $F_i(X_1 + X_2) = F_i(X_1) \oplus F_i(X_2)$. Furthermore, it should be possible to evaluate $F_i(v_i)$ in time $O(T_f \operatorname{nnz}(v_i))$.
3. For any values A, B, C in the range of F, $A \odot (B \oplus C) = (A \odot B) \oplus (A \odot C)$

Theorem 139. *Given a join query $\mathbf{J}' = \mathbf{T}_1 \bowtie \mathbf{T}_2 \bowtie \cdots \bowtie \mathbf{T}_m$ and a DB-Sketch algorithm, there exists an algorithm to evaluate $F(\mathbf{J}')$ in time $O(m(T_\odot + T_\oplus)T_{FAQ} + T_f mnd)$ where n is the size of the largest table and T_{FAQ} is the time complexity of running a single semiring FAQ, while $T_\odot$ and $T_\oplus$ are the time complexities of $\odot$ and $\otimes$, respectively.*

Corollary 11.4.7. *For any join query $\mathbf{J}$ with multiplication order ρ of depth $m - 1$ and any scalar λ, let d_λ be the λ-statistical dimension of $\mathbf{J}$. Then there exists an algorithm that produces $\mathbf{S} \in \mathbb{R}^{k \times n}$ where $k = O(d_\lambda m^4/\epsilon^2)$ in time $O((mkd)T_{FAQ} + m^6 nd/\epsilon^2)$ such that with probability $1 - \frac{1}{\operatorname{poly}(n)}$ simultaneously for all $x \in \mathbb{R}^d$*

$$\|\mathbf{S}\mathbf{J}x\|_2^2 + \lambda\|x\|_2^2 = (1 \pm \epsilon)(\|\mathbf{J}x\|_2^2 + \lambda\|x\|_2^2).$$

Proof. The proof follows by showing that the algorithm in Lemma 11.2.4 is a DB-Sketch. We demonstrate this by introducing the functions F_i and the operators $\oplus$ and $\odot$. The function $F_i(v_i)$ is an OSNAP transform (Lemma 11.2.5) of v_i, $\mathbf{A} \odot \mathbf{B}$ is $\mathbf{S}(\mathbf{A} \otimes \mathbf{B})$ where $\mathbf{S}$ is the Tensor Subsampled Randomized Hadamard Transform as defined in [AKK+20b], and $\mathbf{A} \oplus \mathbf{B}$ is a summation of tensors $\mathbf{A}$ and $\mathbf{B}$. Then it is easy to see that all the properties hold since Kronecker product distributes over summation.

Since F_i needs to be calculated for all rows in each table $\mathbf{T}_i$, which takes $O(T_f mnd)$ time, the operator $\oplus$ takes $O(kd)$ time to apply since the size of the sketch is $k \times d$. The operator $\otimes$ takes at most $O(kd)$ time to apply using the Fast Fourier Transform (FFT) [AKK+20b]. Therefore, the total time complexity can be bounded by $O((mkd)T_{\text{FAQ}} + m^6 nd/\epsilon^2)$.

Lastly, we remark that the algorithm in [AKK+20b] requires that the $\odot$ operator be applied to the input tensors in a binary fashion; however, it is shown in a separate version [AK19] of the paper [AKK+20b] that the sketching construction and results of [AKK+20b] continue to hold when the tensor sketch is applied linearly. See Lemma 6 and 7 in [AK19] and Lemma 10 in [AKK+20b]. □

Corollary 11.4.7 gives an algorithm for all join queries when the corresponding hypertree

decomposition is a path, or has a vertex for which all other vertices are connected to it. Although the time complexity for obtaining an ϵ-subspace embedding ($\lambda = 0$) is not better compared to the exact algorithm for ordinary least squares regression, for ridge regression it is possible to create sketches with many fewer rows and still obtain a reasonable approximation.

In the following we explain the algorithm for DB-Sketch using the FAQ formulation and inside-out algorithm [AKNR16]. Let $\mathbf{J}_i$ denote the matrix resulting from keeping the column i of $\mathbf{J}$ and replacing all other columns with 0. Then we have $\mathbf{J} = \sum_i \mathbf{J}_i$. Using $\mathbf{J}_i$ we can define the proposed algorithm as finding $F(\mathbf{J}_i)$ for all tables $\mathbf{T}_i$ and then calculating $\oplus_i F(\mathbf{J}_i)$. Therefore, all we need to do is to calculate $F(\mathbf{J}_i)$ for all values of i. In the following we introduce an algorithm for the calculation of $F(\mathbf{J}_i)$ and then $F(\mathbf{J})$ is just the summation of the results for the different tables.

Let $e(X_k)$ be the D_k-dimensional unit vector that is 1 in the row corresponding to X_k, and let $v(X_k)$ be the $D_k \times d$ dimensional matrix that agrees with X_k in the row corresponding to X_k, and is 0 everywhere else.

Lemma 11.4.8. *For all tables,* $\mathbf{J}_i = \sum_{X \in \mathbf{J}'} e(X_1) \otimes \cdots \otimes e(X_{i-1}) \otimes v(X_i) \otimes e(X_{i+1}) \otimes \cdots \otimes e(X_m)$*, where* $\otimes$ *is the Kronecker product.*

Proof. For each tuple X, the term inside the summation has $N = |D_1||D_2| \dots |D_m|$ rows and only 1 non-zero row because all of the tensors have only one non-zero row. The non-zero row is the row corresponding to X, and its value is the value of the same row in $\mathbf{J}_i$; therefore, $\mathbf{J}_i$ can be obtained by summing over all the tuples of $\mathbf{J}$. □

Lemma 11.4.9. *Let* F *be a DB-Sketch. Then* $F(\mathbf{J}_i)$ *can be computed in time* $O((T_\odot + T_\oplus)T_{FAQ} + T_f ndm)$

Proof. We define a FAQ for $F(\mathbf{J}_i)$ and then show how to calculate the result. Let $g_j(X_j) = F_j(e(X_j))$ for all $j \neq i$ and $g_i(X_i) = F_i(v(X_i))$. Note that the number of non-zeros entries in $e(X_i)$ is at most d. Therefore, it is possible to find all values of $F_i(v(X_i))$ in time $O(T_f nd)$. The claim is it is possible to use the inside-out [AKNR16] algorithm for the following query and find $\mathbf{SJ}_i$:

$$\bigoplus_{X \in \mathbf{J}'} \bigodot_j g_j(X_j)$$

The mentioned query would be a FAQ if $\odot$ were commutative and associative. However, since

we defined the ordering of the tables based on the multiplication order ρ, the inside-out algorithm multiplies the factors exactly in the same order needed for the DB-sketch algorithm. Therefore, we do not need the commutative and associative property of the $\odot$ operator to run inside-out.

Now we need to show that the query truly calculates $F(\mathbf{J}_i)$. Based on Lemma 11.4.8 and the properties of F we have:

$$\begin{aligned}
F(\mathbf{J}_i) =& F\Big(\sum_{X\in\mathbf{J}'} e(X_1)\otimes\cdots\otimes e(X_{i-1})\otimes v(X_i)\otimes e(X_{i+1})\otimes\cdots\otimes e(X_m)\Big)\\
=& \bigoplus_{X\in\mathbf{J}'} F(e(X_1)\otimes\cdots\otimes e(X_{i-1})\otimes v(X_i)\otimes e(X_{i+1})\otimes\cdots\otimes e(X_m))\\
=& \bigoplus_{X\in\mathbf{J}'} \Big(F_1(e(X_1))\odot\cdots\odot F_{i-1}(e(X_{i-1}))\odot F_i(v(X_i))\odot F_{i+1}(e(X_{i+1}))\odot\cdots\odot F_m(e(X_m))\Big)\\
=& \bigoplus_{X\in\mathbf{J}'} \bigodot_j g_j(X_j)
\end{aligned}$$

as needed. □

Proof of Theorem 139. Finding the values of $F_i(X_i)$ for all i and all tuples of X_i takes $O(T_f ndm)$ time because F_i can be calculated in time $O(T_f \operatorname{nnz}(\mathbf{T}_i))$, and the total number of non-zero entries can be bounded by $O(ndm)$. The calculation of $F(\mathbf{J}_i)$ can be done in time $O(m(T_\odot + T_\oplus)T_{\text{FAQ}}+T_f ndm)$ using m rounds of the inside-out algorithm where $T_{FAQ} = O(md^2 n^{\text{fhtw}}\log(n))$ [AKNR16]. After this step, we need to aggregate the results using the $\oplus$ operator to obtain the final result which takes $O(mT_\oplus)$ time. □

11.5 Evaluation

We study the performance of our sketching method on several real datasets, both for two-table joins and general joins.[6] We first introduce the datasets we use in the experiments. We consider two datasets: LastFM [CBK11] and MovieLens [HK15]. Both of them contain several relational tables. We will compare our algorithm with the FAQ-based algorithm on the joins of some relations.

The LastFM dataset has three relations: **Userfriends** (the friend relations between users), **Userartists** (the artists listened by each user) and **Usertaggedartiststimestamps** (the tag as-

[6]Code available at https://github.com/AnonymousFireman/ICML_code

signments of artists provided by each particular user along with the timestamps).

The MovieLens dataset also has three relations: **Ratings** (the ratings of movies given by the users and the timestamps), **Users** (gender, age, occupation, and zip code information of each user), **Movies** (release year and the category of each movie).

11.5.1 Two-Table Joins

In the experiments for two-table joins, we solve the regression problem $\min_x \|\mathbf{J}_U x - b\|_2^2$, where $\mathbf{J} = T_1 \bowtie T_2 \in \mathbb{R}^{N \times d}$ is a join of two tables, $U \subset [d]$ and b is one of the columns of $\mathbf{J}$. In our experiments, suppose column p is the column we want to predict. We will set $U = [d] \setminus \{p\}$ and b to be the p-th column of $\mathbf{J}$.

To solve the regression problem, the FAQ-based algorithm computes the covariance matrix $\mathbf{J}^T\mathbf{J}$ by running the FAQ algorithm for every two columns, and then solves the normal equations $\mathbf{J}^T\mathbf{J}x = \mathbf{J}^T b$. Our algorithm will compute a subspace embedding $\tilde{\mathbf{J}}$, and then solve the regression problem $\min_x \|\tilde{\mathbf{J}}_U x - \tilde{b}\|_2^2$, i.e., solve $\tilde{\mathbf{J}}^T\tilde{\mathbf{J}}x = \tilde{\mathbf{J}}^T\tilde{b}$.

We compare our algorithm to the FAQ-based algorithm on the LastFM and MovieLens datasets. The FAQ-based algorithm employs the FAQ-based algorithm to calculate each entry in $\mathbf{J}^T\mathbf{J}$.

For the LastFM dataset, we consider the join of **Userartists** and **Usertaggedartiststimestamps**: $\mathbf{J}_1 = \mathbf{UA} \bowtie_{\mathbf{UA}.\text{user}=\mathbf{UTA}.\text{user}} \mathbf{UTA}$. Our regression task is to predict how often a user listens to an artist based on the tags. For the MovieLens dataset, we consider the join of **Ratings** and **Movies**: $\mathbf{J}_2 = \mathbf{R} \bowtie_{\mathbf{R}.\text{movie}=\mathbf{M}.\text{movie}} \mathbf{M}$. Our regression task is to predict the rating that a user gives to a movie.

In our experiments, we do the dataset preparation mentioned in [SOC16], to normalize the values in each column to range $[0, 1]$. For each column, let $v_{\max}$ and $v_{\min}$ denote the maximum value and minimum value in this column. We normalize each value v to $(v - v_{\min})/(v_{\max} - v_{\min})$.

11.5.2 General Joins

For general joins, we consider the ridge regression problem. Specifically, our goal is to find a vector x that minimizes $\|\mathbf{J}x - b\|_2^2 + \lambda\|x\|_2^2$, where $\mathbf{J} = \mathbf{T}_1 \bowtie \cdots \bowtie \mathbf{T}_m \in \mathbb{R}^{N \times d}$ is an arbitrary join, b is one of the columns of $\mathbf{J}$ and $\lambda > 0$ is the regularization parameter. The

Table 11.2: Experimental Results for Two-Table Joins

	n_1	n_2	d	T_{bf}	T_{ours}	**Err**
$\mathbf{J}_1$	92834	186479	6	.034	.011	0.70%
$\mathbf{J}_2$	1000209	3883	23	.820	.088	0.66%

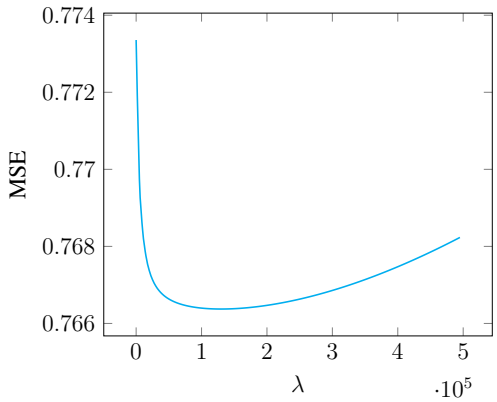


Figure 11.3: MSE vs. λ for our algorithm

optimal solution to the ridge regression problem can be found by solving the normal equations $(\mathbf{J}^T\mathbf{J} + \lambda \mathbb{I}_d)x = \mathbf{J}^T b$.

The FAQ-based algorithm is the same as in the experiment for two-table joins. It directly runs the FAQ algorithm a total of $d(d+1)/2$ times to compute every entry of $\mathbf{J}^T\mathbf{J}$.

We run our algorithm, discussed in Section 11.4, and the FAQ-based algorithm on the MovieLens-25m dataset, which is the largest of the MovieLens [HK15] datasets. We consider the join of **Ratings**, **Users** and **Movies**: $\mathbf{J}_3 = \mathbf{R} \bowtie_{\mathbf{R}.\text{user}=\mathbf{U}.\text{user}} \mathbf{U} \bowtie_{\mathbf{U}.\text{movie}=\mathbf{M}.\text{movie}} \mathbf{M}$. Our regression task is to predict the rating that a user gives to a movie.

11.5.3 Results

‘

We run the FAQ-based algorithm and our algorithm on those joins and compare their running times. To measure accuracy, we compute the relative mean-squared error, given by:

$$\textbf{Err} = \frac{\|\mathbf{J}_U x_{\text{ours}} - b\|_2^2 - \|\mathbf{J}_U x_{\text{bf}} - b\|_2^2}{\|\mathbf{J}_U x_{\text{bf}} - b\|_2^2}$$

in the experiments for two-table joins, where x_{bf} is the solution given by the FAQ-based al-

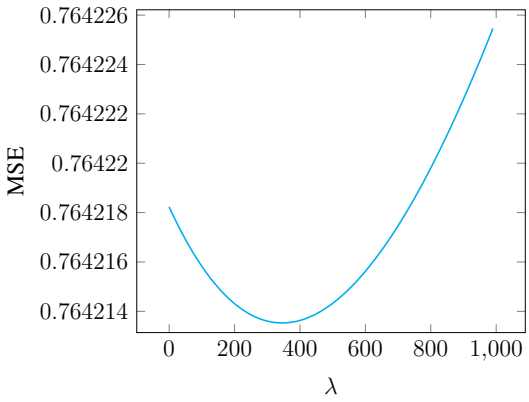


Figure 11.4: MSE vs. λ for the FAQ-based algorithm

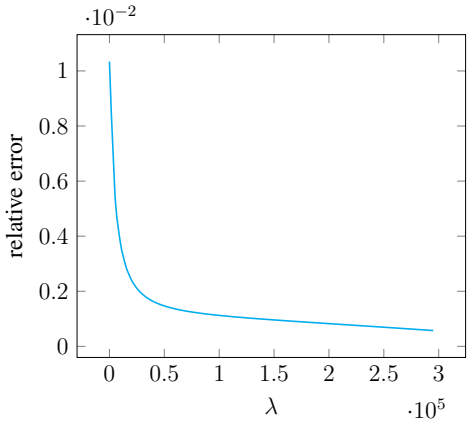


Figure 11.5: Relative Error vs. λ

gorithm, and x_{ours} is the solution given by our algorithm. All results (runtime, accuracy) are averaged over 5 runs of each algorithm.

In our implementation, we adjust the target dimension in our sketching algorithm for each experiment, as in practice it appears unnecessary to parameterize according to the worst-case theoretical bounds when the number of features is small, as in our experiments. Additionally, for two-table joins, we replace the Fast Tensor-Sketch with Tensor-Sketch ([AKK+20b, Pag13]) for the same reason. The implementation is written in MATLAB and run on an Intel Core i7-7500U CPU with 8GB of memory.

We let T_{bf} be the running time of the FAQ-based algorithm and T_{ours} be the running time of our approach, measured in seconds. Tables 11.2 shows the results of our experiments for two-table joins. From that we can see our approach can give a solution with relative error less than 1%, and its running time is significantly less than that of the FAQ-based algorithm.

For general joins, due to the size of the dataset, we implement our algorithm in Taichi [HLA+19, HAL+20] and run it on an Nvidia GTX1080Ti GPU. We split the dataset into a training set and a validation set, run the regression on the training set and measure the MSE (mean squared error) on the validation set. We fix the target dimension and try different values of λ to see which value achieves the best MSE.

Our algorithm runs in 0.303s while FAQ-based algorithm runs in 0.988s. The relative error of MSE (namely, $\frac{\text{MSE}_{\text{ours}}-\text{MSE}_{\text{bf}}}{\text{MSE}_{\text{bf}}}$, both measured under the optimal λ) is only 0.28%.

We plot the MSE vs. λ curve for the FAQ-based algorithm and our algorithm in Figure 11.3 and 11.4. We observe that the optimal choice of λ is much larger in the sketched problem than in the original problem. This is because the statistical dimension d_λ decreases as λ increases. Since we fix the target dimension, ϵ thus decreases. So a larger λ can give a better approximate solution, yielding a better MSE even if it is not the best choice in the unsketched problem.

We also plot the relative error of the objective function in Figure 11.5. For ridge regression it becomes

$$\frac{(\|\mathbf{J}x_{\text{ours}} - b\|_2^2 + \lambda\|x_{\text{ours}}\|_2^2) - (\|\mathbf{J}x_{\text{bf}} - b\|_2^2 + \lambda\|x_{\text{bf}}\|_2^2)}{\|\mathbf{J}x_{\text{bf}} - b\|_2^2 + \lambda\|x_{\text{bf}}\|_2^2}.$$

We can see that the relative error decreases as λ increases in accordance with our theoretical analysis. sub

Table 11.3: General Algorithm on Two-Table Joins

	k	running time	relative error
$\mathbf{J}_1$	40	0.086	42.3%
	80	0.10	9.79%
	120	0.14	3.67%
	160	0.16	0.87%
	200	0.19	1.05%
$\mathbf{J}_2$	400	1.25	5.79%
	800	2.02	2.32%
	1200	2.93	1.85%
	1600	3.74	1.19%
	2000	4.50	0.96%

11.5.4 Comparing the Algorithms for Two-Table Joins

In this chapter, we have presented two algorithms for regression on database joins: one specifically for two-table joins, and one for more general join queries. A natural question is whether it is always advantageous to use the two-table join algorithm when the input is, in fact, a join of two tables. Thus, we now compared the two algorithms in the context of two-table joins, both theoretically and empirically.

Theoretical Comparison

By Theorem 135, the total running time to obtain a subspace embedding is the minimum of $\tilde{O}((n_1+n_2)d/\epsilon^2+d^3/\epsilon^2)$ and $\tilde{O}((\text{nnz}(\mathbf{T}_1)+\text{nnz}(\mathbf{T}_2))/\epsilon^2+(n_1+n_2)/\epsilon^2+d^5/\epsilon^2)$ using Algorithm 12.

Now we consider the algorithm stated in Corollary 11.4.7. When running on the join of two tables, the algorithm is equivalent to applying the Fast Tensor-Sketch to each block and summing them up. Thus, the running time is $\tilde{O}(\sum_{i\in\mathcal{B}}(\text{nnz}(\mathbf{T}_1^{(i)})/\epsilon^2 + \text{nnz}(\mathbf{T}_2^{(i)})/\epsilon^2 + d^2/\epsilon^2)) = \tilde{O}((\text{nnz}(\mathbf{T}_1) + \text{nnz}(\mathbf{T}_2))/\epsilon^2 + (n_1 + n_2)d^2/\epsilon^2)$.

The running time of the algorithm in Corollary 11.4.7 is greater than the running time of Algorithm 12. In the extreme case the number of blocks can be really large, and even if each block has only a few rows we still need to pay an extra $\tilde{O}(d^2/\epsilon^2)$ time for it. This is the reason why we split the blocks into two sets ($\mathcal{B}_{\text{big}}$ and $\mathcal{B}_{\text{small}}$) and use a different approach when designing the algorithm for two-table joins.

Experimental Comparison

In our experiments we replace the Fast Tensor-Sketch with Tensor-Sketch. The theoretical analysis is similar since we still need to pay an extra $O(kd \log k)$ time to sketch a block for target dimension k.

We run the algorithm for the general case on joins $\mathbf{J}_1$ and $\mathbf{J}_2$ for different target dimensions. As shown in Table 11.3, in order to achieve the same relative error as Algorithm 12, we need to set a large target dimension and the running time would be significantly greater than it is in Table 11.2, even compared with FAQ-based algorithm. This experimental result agrees with our theoretical analysis.

Part III

Database Query Evaluation

Chapter 12

Approximate Counting and Uniform Sampling from Database Queries

Part III of this thesis concerns the problems of *approximate counting* of and *uniform sampling* from a database query. Informally, given as input a relational database $D = \{\mathbf{R}_1, \ldots, \mathbf{R}_m\}$, where each $\mathbf{R}_i$ is a relation, and a query Q, one can define the set $Q(D)$ of "answers" to the query Q over the databse D. One would then like to estimate the size $|Q(D)|$, or uniformly sample from the set $Q(D)$. These two tasks are of fundamental importance for query optimization [RGG03, PS13b], data mining, and statistical inference [AD20]. Drawing an example from Chapter 11, the query $Q(D)$ could be a database join on the relations $\mathbf{R}_1, \ldots, \mathbf{R}_m$. Importantly, and as seen in Chapter 11, the size of the query $Q(\mathcal{D})$ can grow exponentially in the size of the database D. Consequentially, counting and uniform sampling are inherently non-trivial.

We consider the class of *conjunctive queries* (CQ), which are one of the most common class of queries used in database systems, and the best studied in the literature. Since the relation $Q(D)$ is often a high-dimensional object, expressed by the lower dimensional representation (Q, D), one may hope to apply sketching techniques on Q, D so as to compress $Q(D)$ into, for instance, a uniform sample of answers. However, in general, even checking if $Q(\mathcal{D})$ is empty (known as the *decision problem*) is NP-Hard [CM77b], thus polynomial time algorithms for approximating $|Q(D)|$, or generating sub-sampled sketches, are unlikely to exist.

A seminal result of Grohe, Schwentick, and Segoufin [GSS01] characterized the classes of conjunctive queries for which the decision problem is in P. In this chapter, we describe a line of work set out in the papers [ACJR19, ACJR21] which characterizes the classes of conjunctive queries for which polynomial time algorithms exist for approximate counting and sampling.

Roughly, our main result is the first fully polynomial time randomized approximation scheme (FPRAS) for counting answers to CQs with bounded *tree-width* (definition in Section 12.3.2), as well as a polynomial time uniform sampler for such answers. By a lower bound of [GSS01], the decision problem, and therefore approximate counting, is hard for classes with unbounded treewidth, demonstrating that our result is the best possible.

The main ingredient in our result is the resolution of a fundamental counting problem from automata theory. Specifically, the answers to a class of CQs can be modeled by the accepting inputs to families of finite automata. In [ACJR19], we demonstrate the first FPRAS for counting the number of words of length n accepted by a *non-deterministic finite automton* (NFA). In [ACJR21], we extend this algorithm, by demonstrating the first FPRAS and polynomial time sampler for the set of trees of size n accepted by a *tree automaton*, which significantly generalize NFAs, and can express the answers to CQs with bounded tree-width. Previously, the best known algorithm for either task was a quasi-polynomial time randomized approximation scheme (QPRAS) of Gore, Jerrum, Kannan, Sweedyk, and Mahaney [GJK+97].

In contrast to the QPRAS of [GJK+97], the algorithm presented in this chapter is based on sketching techniques. Specifically, if $\mathcal{L}_n$ is the language of accepting words (or trees) of an NFA $\mathcal{N}$ (or tree automata $\mathcal{T}$), one can write down a natural dynamic program (DP) to generate the entire set $\mathcal{L}_n$. The intermediate states of this DP consist of sets of substrings (or subtrees) which can be composed together to ultimately form $\mathcal{L}_n$. Since $|\mathcal{L}_n|$ is exponential in the input $(n, |\mathcal{N}|)$ to the problem, the intermediate states are exponentially large. Our approach is to compress these states, replacing each with a sketch of the corresponding set consisting of i.i.d. uniform samples. The main challenge is how to generate these sketches bottom up through the DP – the majority of this chapter consists of addressing this challenge.

This chapter is based on joint works with Marcelo Arenas, Luis Alberto Croquevielle, and Cristian Riveros [ACJR19, ACJR21]. The first paper [ACJR19] appeared in PODS 2019, where it won the Best Paper Award, received a ACM SIGMOD Research Highlight Award (2020), an invitation the the Journal of the ACM, and an invitation as a highlighted plenary talk at STOC 2021. The second paper [ACJR21] appeared in STOC 2021.

Highlighted Contributions

The main contributions of this chapter as as follows:

- We give the first FPRAS and polynomial time uniform sampler the set of answers to CQs with bounded *hypertree-width* (Section 12.3).

- Given a integer n and a non-deterministic finite automata (NFA) $\mathcal{N}$, we give the first FPRAS for approximating the number of strings of length n accepted by $\mathcal{N}$. Furthermore, we give a the first FPRAS for the more general task of approximating the number of *trees* of size n accepted by a tree automata $\mathcal{T}$ (Sections 12.5 and 12.6).

- We demonstrate how our results can be applied to obtain an FPRAS for many open problems, such as counting solutions to constraint satisfaction problems (CSP) with bounded hypertree-width, counting the number of error threads in programs with nested call subroutines, and counting valid assignments to structured DNNF circuits (Section 12.7).

12.1 Background

Let $\mathbf{C}$ and $\mathbf{V}$ be two disjoint sets of *constants* and *variables*, respectively. Then a conjunctive query (CQ) is an expression of the form:

$$Q(\bar{x}) \quad \leftarrow \quad R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n), \tag{12.1}$$

where for every $i \in [n]$, R_i is a k_i-ary relation symbol ($k_i \geq 1$), $\bar{u}_i \in (\mathbf{C} \cup \mathbf{V})^{k_i}$ is a k_i-ary tuple of variables and constants, and $\bar{x} = (x_1, \ldots, x_m)$ is a tuple of variables such that each variable x_i in $\bar{x}$ occurs in at least one $\bar{u}_i$. A database D is an instantiation $D = \{R_1^D, R_2^D, \ldots, R_n^D\}$ of the relations, so that each $R_i^D \subset \mathbf{C}^{k_i}$ is a finite subset of k_i-ary tuples $\bar{a}$ of constants (known as *facts*). A homomorphism from Q to D is a function $h : \mathbf{V} \to \mathbf{C}$ such that for every $i \in [n]$, we have $h(\bar{u}_i) \in R_i^D$, where $h(\bar{u}_i)$ is applied coordinate-wise and acts as the identity on $\mathbf{C}$. Given such a homomorphism h, the tuple $h(\bar{x})$ is called an *answer* to Q over D, and the set of all such answers is denoted $Q(D)$.

Conjunctive queries are the most common class of queries used in database systems. They correspond to *select-project-join* queries in relational algebra and *select-from-where* queries in SQL, and are closely related to *constraint satisfaction problems* (CSPs). Therefore, the computational complexity of tasks related to the evaluation of conjunctive queries is a fundamental object of study.

Given as input a database instance D and a conjunctive query $Q(\bar{x})$, the *query evaluation*

problem is defined as the problem of computing $Q(D)$.The corresponding *query decision problem* is to verify whether or not $Q(D)$ is empty. It is well known that even the query decision problem is NP-complete for conjunctive queries [CM77b]. Thus, a major focus of investigation in the area has been to find tractable special cases [Yan81, CR97, GLS98, GSS01, GLS02, FG06, GGLS16].

In addition to evaluation, two fundamental problems for conjunctive queries are counting the number of answers to a query and uniformly sampling such answers. The counting problem for CQ is of fundamental importance for query optimization [RGG03, PS13b]. Specifically, the optimization process of a relational query engine requires, as input, an estimate of the number of answers to a query (without evaluating the query). Furthermore, uniform sampling is used to efficiently generate representative subsets of the data, instead of computing the entire query, which are often sufficient for data mining and statistical tasks [AD20]. Starting with the work of Chaudhuri, Motwani and Narasayya [CMN99b], the study of random sampling from queries has attracted significant attention from the database community [ZCL+18, CY20].

Beginning with the work in [Yan81], a fruitful line of research for finding tractable cases for CQs has been to study the *degree of acyclicity* of a CQ. In particular, the *treewidth* $\mathrm{tw}(Q)$ of Q [CR97, GSS01], and more generally the *hypertree width* $\mathrm{hw}(Q)$ of Q [GLS02], are two primary measurements of the degree of acyclicity. It is known that the query decision problem can be solved in polynomial time for every class $\mathcal{C}$ of CQs with bounded treewidth [CR97, GSS01] or bounded hypertree width [GLS02].[1] A seminal result of Grohe, Schwentick, and Segoufin [GSS01] demonstrates that for every class $\mathcal{G}$ of graphs, the evaluation of all conjunctive queries whose underlying graph is in $\mathcal{G}$ is tractable if, and only if, $\mathcal{G}$ has bounded treewidth. Hence, the property of bounded treewidth provides a characterization of tractability of the query decision problem.

Unfortunately, uniform generation and exact counting are more challenging than query evaluation for CQs. Specifically, given as input a conjunctive query Q and database D, computing $|Q(D)|$ is #P-complete even when $\mathrm{tw}(Q) = 1$ [PS13b] (that is, for so called *acyclic* CQs [Yan81]). Moreover, even approximate counting is intractable for queries with unbounded treewidth, since any multiplicative approximation clearly solves the decision problem. On the other hand, these facts do not preclude the existence of efficient approximation algorithms for classes of CQs with bounded treewidth, as the associated query decision problem is in P. Despite this possibility, to date no efficient approximation algorithms for these classes are known.

[1] $\mathcal{C}$ has bounded treewidth (hypertree width) if $\mathrm{tw}(Q) \leq k$ ($\mathrm{hw}(Q) \leq k$) for every $Q \in \mathcal{C}$, for a fixed constant k.

In this paper, we fill this gap by demonstrating the existence of a fully polynomial-time randomized approximation scheme (FPRAS) and a fully polynomial-time almost uniform sampler (FPAUS) for every class of CQs with bounded hypertree width. Since $\mathrm{hw}(Q) \leq \mathrm{tw}(Q)$ for every CQ Q [GLS02], our result also includes every class of CQs with bounded treewidth, as well as classes of CQs with bounded hypertree width but unbounded treewidth [GLS02]. Specifically, we show the following.

Theorem 140 (Theorem 144 informal)**.** *Let $\mathcal{C}$ be a class of CQs with bounded hypertree width. Then there exists a fully polynomial-time randomized approximation scheme (FPRAS) that, given $Q \in \mathcal{C}$ and a database D, estimates $|Q(D)|$ to multiplicative error $(1 \pm \epsilon)$. Moreover, there is a fully polynomial-time almost uniform sampler (FPAUS) that generates samples from $Q(D)$.*

Our algorithm of Theorem 140 in fact holds for a larger class of queries, including *unions* of conjunctive queries with bounded hypertree width (Proposition 12.3.3). Note that, as defined in [JVV86a], an FPAUS samples from a distribution with variational distance δ from uniform (see Section 12.2.2 for a formal definition).

An interesting question is whether there exists a larger class of queries $\mathcal{C}$ that admits an FPRAS. Since the decision problem for $\mathcal{C}$ is in BPP whenever $\mathcal{C}$ admits an FPRAS, as a corollary of Theorem 140 and the characterization of [GSS01], we obtain the following answer to this question (see Section 12.3.3 for a precise statement of this result, and for the necessary terminology for this statement):

Corollary 12.1.1 (Corollary 12.3.1 informal)**.** *Let $\mathcal{G}$ be a class of (undirected) graphs and $\mathcal{C}$ be the class of all CQs whose underlying graph is in $\mathcal{G}$. Then assuming $\mathrm{W}[1] \neq \mathrm{FPT}$ and $\mathrm{BPP} = \mathrm{P}$, the following are equivalent: (1) the problems of computing $|Q(D)|$ and of sampling from $Q(D)$, given as input $Q \in \mathcal{C}$ and a database D, admit an FPRAS and an FPAUS, respectively; and (2) $\mathcal{G}$ has bounded treewidth.*

Corollary 12.1.1 shows that the results of [GSS01] can be extended to the approximate counting problem for CQs. Perhaps surprisingly, this demonstrates that the classes of CQs for which the decision problem is tractable, in the sense studied in [GSS01], are precisely *the same* as the classes which admit an FPRAS. Besides, this gives a positive answer to the line of research started in [CMN99b], by providing a characterization of the class of queries that admit an almost uniform sampler.

12.1.1 An FPRAS for tree automata

The key to our results is the resolution of a fundamental counting problem from automata theory; namely, the counting problem for *tree automata*. Specifically, we first demonstrate that the solution space $Q(D)$ of a conjunctive query with bounded hypertree-width can be efficiently expressed as the language accepted by a tree automaton $\mathcal{T}$. We then demonstrate the first FPRAS for the problem of counting the number of trees accepted by a tree automaton $\mathcal{T}$.

Tree automata are the natural extension of *non-deterministic finite automata* (NFA) from words to trees. This extension is a widely studied topic, since they have a remarkable capacity to model problems, while retaining many of the desirable computational properties of NFAs [Sei90, Tho97]. Beginning with the strong decidability result established by Rabin [Rab69], many important problems have been shown to be decidable via tree automata. Moreover, the fact that tree automata are equivalent to monadic second order-logic [TW68] is a basic component of the proof of Courcelle's theorem [Cou90]. Further applications of tree automata, among others, include model checking [EJ91, Var95], program analysis [AEM04, AM04, AM09], databases [Nev02, Sch07], and knowledge representation [Ter99, CDGL99, BCM$^+$03] (see also [Tho97, CDG$^+$07] for a survey).

The counting problem of tree automata. Similarly to how a non-deterministic finite automaton N accepts a language $\mathcal{L}(N)$ of words, a tree automaton $\mathcal{T}$ accepts a language $\mathcal{L}(\mathcal{T})$ of labeled trees. Given $\mathcal{T}$ and an integer n, define the n-slice of $\mathcal{L}(\mathcal{T})$ as:

$$\mathcal{L}_n(\mathcal{T}) = \{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}$$

where $|t|$ is the number of vertices in t. We consider the counting problem #TA, which is the main counting problem studied in this paper regarding tree automata:

Problem:	#TA
Input:	A tree automaton $\mathcal{T}$ and a string 0^n
Output:	$\lvert\mathcal{L}_n(\mathcal{T})\rvert$

While exactly computing the size of the n-slice for deterministic finite automata and deterministic tree automata is tractable [Mai94, BGS00, KSM95], this is not the case for their *non-deterministic* counterparts. In fact, given as input an NFA $\mathcal{A}$ and a number n in unary, the problem of computing $|\mathcal{L}_n(\mathcal{A})|$ is #P-hard [ÀJ93], which implies #P-hardness for tree automata. Naturally, this does not rule out the possibility of efficient approximation algorithms. This obser-

vation was first exploited by Kannan, Sweedyk, and Mahaney, who gave a *quasi polynomial-time approximation scheme* (QPRAS) for NFAs [KSM95], which was later extended by the aforementioned authors, along with Gore and Jerrum [GJK+97], to the case of tree automata.[2] Specifically, the algorithm of [GJK+97] runs in time $\epsilon^{-2}(nm)^{O(\log(n))}$, where $m = |\mathcal{T}|$ is the size of the description of $\mathcal{T}$, and ϵ is the error parameter. Improving the complexity of this algorithm to polynomial time has been an open problem.

The algorithms of [GJK+97] and [KSM95] are based on a recursive form of Karp-Luby sampling [KLM89], which is a type of rejection sampling. This approach has the drawback that the probability a sample is chosen is exponentially small in the depth of the recursion. Recently, using a different sampling scheme, it was shown that an FPRAS and an FPAUS exist for NFAs [ACJR19]. However, the techniques in [ACJR19] break down fundamentally (discussed in the following) when applied to tree automata. The main technical contribution of this work is to address the failing points of [ACJR19] for tree automata, and design an FPRAS for this case.

Theorem 141. *Given a tree automaton $\mathcal{T}$ and $n \geq 1$, there is an algorithm which runs in time $\mathrm{poly}(|\mathcal{T}|, n, \epsilon^{-1}, \log(\delta^{-1}))$ and with probability $1 - \delta$, outputs an estimate $\widetilde{N}$ with:*

$$(1 - \epsilon)|\mathcal{L}_n(\mathcal{T})| \leq \widetilde{N} \leq (1 + \epsilon)|\mathcal{L}_n(\mathcal{T})|$$

Conditioned on the success of this event, there is a sampling algorithm where each call runs in time $\mathrm{poly}(|\mathcal{T}|, n, \log(\delta^{-1}))$, and either outputs a uniformly random tree $t \in \mathcal{L}_n(\mathcal{T})$, or $\perp$. Moreover, it outputs $\perp$ with probability at most $1/2$.

Note that conditioned on the success of the above FPRAS (run once), every subsequent call to the sampler generates a *truly* uniform sample (or $\perp$). Observe that this notion of sampling is stronger than the standard notion of FPAUS (see Section 12.2.2). We note that the existence of an FPAUS is in fact a corollary of the existence of an FPRAS for the above [JVV86a].

Succinct NFAs. A key step in the proof of Theorem 141 is a reduction to counting and sampling from a *succinct* NFA $\mathcal{N}$, which is an NFA with succinctly encoded alphabet and transitions. Formally, a succinct NFA $\mathcal{N}$ is a 5-tuple $(S, \Sigma, \Delta, s_{\text{init}}, s_{\text{final}})$, where S is a set of states, Σ is an alphabet, $s_{\text{init}}, s_{\text{final}} \in S$ are the initial and final states, and $\Delta \subseteq S \times 2^{\Sigma} \times S$ is the transition relation, where each transition is labeled by a set $A \subseteq \Sigma$. We assume that Σ is succinctly encoded via some representation (e.g. a DNF formula), and likewise for each set A such that $e = (s, A, s')$ is a transition in Δ. Therefore, the size of the alphabet Σ and the size of each such set A can be

[2] The QPRAS of [GJK+97] holds more generally for *context free grammars* (CFG).

exponentially large in the representation of $\mathcal{N}$. A word $w = w_1 w_2 \dots w_n \in \Sigma^*$ is *accepted* by $\mathcal{N}$ if there is a sequence $s_{\text{init}} = s_0, s_1, \dots, s_n = s_{\text{final}}$ of states such that there exists a transition $(s_{i-1}, A, s_i) \in \Delta$ with $w_i \in A$ for each $i = 1, 2, \dots, n$. Note that the special case where each transition $(s, A, s') \in \Delta$ satisfies $|A| = 1$ is precisely the standard definition of an NFA. To solve the aforementioned problems for succinct NFA, we must assume that the encodings of the label sets satisfy some basic conditions. Specifically, we require that for each transition (s, A, s'), we are given an oracle which can **(1)** test membership in A, **(2)** produce an estimate of the size of $|A|$, and **(3)** generate almost-uniform samples from A.

Theorem 142. *Let $\mathcal{N} = (S, \Sigma, \Delta, s_{init}, s_{final})$ be a succinct NFA and $n \geq 1$. Suppose that the sets A in each transition $(s, A, s') \in \Delta$ satisfy the properties **(1)**, **(2)** and **(3)** described above. Then there is an FPRAS and an FPAUS for $\mathcal{L}_n(\mathcal{N})$.*

While standard (non-succinct) NFAs are known to admit an FPRAS by the results of [ACJR19], Theorem 142 is a strong generalization of the main result of [ACJR19], and requires many non-trivial additional insights and techniques.

12.1.2 Additional applications of the FPRAS

We demonstrate that the FPRAS of Theorem 141 results in the first polynomial-time randomized approximation algorithms for many previously open problems in the fields of constraint satisfaction problems, verification of correctness of programs with nested calls to subroutines, and knowledge compilation. We give a brief overview of these results in what follows. Further details are deferred to the full version.

Constraint satisfaction problems. Constraint satisfaction problems (CSPs) offer a general and natural setting to represent a large number of problems where solutions must satisfy some constraints, and which can be found in different areas [Var00, CKS01, RVBW06, HN04, BHvMW09, RN16]. The most basic task associated to a CSP is the problem of verifying whether it has a solution, which corresponds to an assignment of values to the variables of the CSP that satisfies all the constraints of the problem. Tightly related with this task is the problem of counting the number of solutions to a CSP. In this work, we consider this counting problem in the usual setting where a projection operator for CSPs is allowed, so that it is possible to indicate the output variables of the problem. We denote this setting as ECSP.

As counting the number of solutions of an ECSP is #P-complete and cannot admit an FPRAS

(unless NP $=$ RP), we focus on two well known notions of acyclicity that ensure that solutions can be found in polynomial time [GLS00, GLS02]. More precisely, we define #AECSP as the problem of counting, given an acyclic ECSP $\mathcal{E}$, the number of solutions to $\mathcal{E}$. Moreover, given a fixed $k \geq 0$, we define $\#k$-HW-ECSP as the problem of counting, given an ECSP $\mathcal{E}$ whose hypertree-width is at most k, the number of solutions for $\mathcal{E}$. Although both problems are known to be #P-complete [PS13b], we obtain as a consequence of Theorem 141 that both #AECSP and $\#k$-HW-ECSP admit FPRAS.

Software verification. Nested words have been proposed as a model for the formal verification of correctness of structured programs that can contain nested calls to subroutines [AEM04, AM04, AM09]. In particular, the execution of a program is viewed as a linear sequence of states, where a matching relation is used to specify the correspondence between each point during the execution at which a procedure is called with the point when we return from that procedure call. This idea gives rise to the notion of nested word, which is defined as a regular word accompanied by a matching relation. Moreover, properties of programs to be formally verified are specified by using nested word automata (NWA). The emptiness problem for nested word automata ask whether, given a NWA $\mathcal{N}$, there exists a nested word accepted by $\mathcal{N}$. This is a fundamental problem when looking for faulty executions of a program with nested calls to subroutines; if $\mathcal{N}$ is used to encode the complement of a property we expect to be satisfied by a program, then a nested word accepted by $\mathcal{N}$ encodes a bug of this program. In this sense, the following is also a very relevant problem for understanding how faulty a program is. Define #NWA as the problem of counting, given a nested word automaton $\mathcal{N}$ and a string 0^n, the number of nested words of length n accepted by $\mathcal{N}$. As expected, #NWA is a #P-complete problem. Interestingly, from Theorem 141 and the results in [AM09] showing how nested word automata can be represented by using tree automata over binary trees, it is possible to prove that #NWA admits an FPRAS.

Knowledge compilation. Model counting is the problem of counting the number of satisfying assignments given a propositional formula. Although this problem is #P-complete [Val79], there have been several approaches to tackle it [GSS09]. One of them comes from the field of *knowledge compilation*, a subarea in artificial intelligence [DM02]. Roughly speaking, this approach consists in dividing the reasoning process in two phases. The first phase is to compile the formula into a target language (e.g. Horn formulae, BDDs, circuits) that has good algorithmic properties. The second phase is to use the new representation to solve the problem efficiently. The main goal then is to find a target language that is expressive enough to encode a rich set of propositional formulae and, at the same time, that allows for efficient algorithms to solve the counting problem.

A target language for knowledge compilation that has attracted a lot of attention is the class of DNNF circuits [Dar01a]. DNNF has good algorithmic properties in terms of satisfiability and logical operations. Furthermore, DNNF can be seen as a generalization of DNF formulae, ordered binary decision diagrams (OBDDs) [Bry92] and free binary decision diagrams (FBDDs) [DM02], in the sense that every expression in these formalisms can be transformed into a DNNF circuit in polynomial time. Moreover, DNNF is exponentially more succinct than DNF, OBDD and FBDD [DM02], and hence it is a more appealing language for knowledge compilation. Regarding model counting, DNNF circuits can easily encode #P-complete problems (e.g. #DNF) and, therefore, researchers have look into subclasses of DNNF where counting can be done more efficiently. One such a class that has recently received a lot of attention is the class of structured DNNF [PD08], which has been used for efficient enumeration [ABJM17, ABMN19], and has proved to be appropriate to compile propositional CNF formulae with bounded width (e.g. CV-width) [OD14]. Unfortunately, the problem of computing the number of propositional variable assignments that satisfy a structured DNNF circuit is a #P-complete problem, as these circuits include the class of DNF formulae. However, and in line with the idea that structured DNNF circuits allow for more efficient counting algorithms, we can prove that the counting problem of structured DNNF circuits admits a fully-polynomial time randomized approximation scheme as a consequence of Theorem 141.

12.1.3 Outline of the Chapter

In Section 12.2, we introduce the relevant automata theory background needed for this chapter. Section 12.3 formalizes the connection between conjunctive queries and tree automata. Section 12.4 gives a general overview of the main FPRAS for tree automata, which is formally described and proven in Sections 12.5 and 12.6. In Section 12.7, we describe the various additional applications and corollaries of our main result. Finally, in Section 12.8, we discuss open problems and future work.

12.2 Primer on Automata Theory

In this section, we introduce several fundamental concepts related to finite automata, which will be central to our results.

12.2.1 Intervals, strings, trees and tree automata

Basic notation. Given $m \leq n$ with $n, m \in \mathbb{N}$, we use notation $[m, n]$ for the set $\{m, m+1, \ldots, n\}$, and notation $[n]$ for the set $[1, n]$. Moreover, given $u, \epsilon \in \mathbb{R}$ with $\epsilon \geq 0$, let $(u \pm \epsilon)$ denote the real interval $[u-\epsilon, u+\epsilon]$. In general, we consider real intervals of the form $(1 \pm \epsilon)$, and we use $x(1 \pm \epsilon)$ to denote the range $[x - x\epsilon, x + x\epsilon]$, and $x = (1 \pm \epsilon)y$ to denote the containment $x \in [y - \epsilon y, y + \epsilon y]$.

Strings and Sequences. Given a finite alphabet Σ, a finite string over Σ is a sequence $w = w_1 \ldots w_n$ such that $n \geq 0$ and $w_i \in \Sigma$ for every $i \in [n]$. Notice that if $n = 0$, then w is the empty word, which is denoted by λ. We write $|w| = n$ for the length of w. As usual, we denote by Σ^* all strings over Σ. For two sets $A, B \subseteq \Sigma^*$ we denote by $A \cdot B = \{u \cdot v \mid u \in A, v \in B\}$, where $u \cdot v$ is the concatenation of two strings u and w, and by A^i the concatenation of A with itself i times, that is, $A^0 = \{\lambda\}$ and $A^{i+1} = A \cdot A^i$ for every $i \in \mathbb{N}$.

Ordered Trees. Fix $k \in \mathbb{N}$ with $k \geq 1$. A finite ordered k-tree (or just a k-tree) is a prefix-closed non-empty finite subset $t \subseteq [k]^*$, namely, if $w \cdot i \in t$ with $w \in [k]^*$ and $i \in [k]$, then $w \in t$ and $w \cdot j \in t$ for every $j \in [i]$. For a k-tree t, $\lambda \in t$ is the called the root of t and every maximal element in t (under prefix order) is called a leaf. We denote by $\mathrm{leaves}(t)$ the set of all leaves of t. For every $u, v \in t$, we say that u is a child of v, or that v is the parent of u, if $u = v \cdot i$ for some $i \in [k]$. We say that v has n children if $v \cdot 1, \ldots, v \cdot n \in t$ with $n = \max_{v \cdot i \in t}\{i\}$. We denote by $v = \mathrm{parent}(u)$ when v is the parent of u (if u is the root, then $\mathrm{parent}(u)$ is undefined). Furthermore, we say that v is an ancestor of u, or u is a descendant of v, if v is a prefix of u. The size of t, i.e. the number of nodes, is denoted by $|t|$.

Let Σ be a finite alphabet and t be a k-tree. Slightly abusing notation, we also use t to denote a k-tree labeled over Σ. That is, we also consider t as a function such that for every $u \in t$, it holds that $t(u) \in \Sigma$ is the label assigned to node u. For $a \in \Sigma$, we denote just by a the tree consisting of one node labeled with a. For labeled k-trees t and t', and a leaf $\ell \in t$, we define $t[\ell \to t']$ the labeled k-tree resulting from "hanging" t' on the node ℓ in t. Formally, we have that $t[\ell \to t'] = t \cup (\{\ell\} \cdot t')$, $t[\ell \to t'](u) = t(u)$ whenever $u \in (t \setminus \{\ell\})$ and $t[\ell \to t'](\ell \cdot u) = t'(u)$ whenever $u \in t'$. Note that the leaf ℓ takes in $t[\ell \to t']$ the label on t' instead of its initial label on t. When t consists of just one node with label a and with two children, we write $a(t_1, t_2)$ for the tree defined as $t[1 \to t_1][2 \to t_2]$, namely, the tree consisting of a root a with t_1 and t_2 hanging to the left and right, respectively. In particular, $t = a(b, c)$ is the tree with three nodes such that $t(\lambda) = a$, $t(1) = b$, and $t(2) = c$. Finally, we denote by $\mathrm{Trees}_k[\Sigma]$ the set of all k-trees labeled over Σ (or just k-trees over Σ).

Tree Automata. A (top-down) tree automaton $\mathcal{T}$ over $\mathrm{Trees}_k[\Sigma]$ is a tuple $(S, \Sigma, \Delta, s_{\text{init}})$ where S is a finite set of states, Σ is the finite alphabet, $\Delta \subseteq S \times \Sigma \times (\cup_{i=0}^{k} S^i)$ is the transition relation, and $s_{\text{init}} \in S$ is the initial state. We will usually use s, q, and r to denote states in S. A run ρ of $\mathcal{T}$ over a k-tree t is a function $\rho : t \to S$ that assigns states to nodes of t such that for every $u \in t$, if $u \cdot 1, \ldots, u \cdot n$ are the children of u in t, then $(\rho(u), t(u), \rho(u \cdot 1)\rho(u \cdot 2) \ldots \rho(u \cdot n)) \in \Delta$. In particular, if u is a leaf, then it holds that $(\rho(u), t(u), \lambda) \in \Delta$. We say that $\mathcal{T}$ accepts t if there exists a run of $\mathcal{T}$ over t with $\rho(\lambda) = s_{\text{init}}$, and we define $\mathcal{L}(\mathcal{T}) \subseteq \mathrm{Trees}_k[\Sigma]$ as the set of all k-trees over Σ accepted by $\mathcal{T}$. We write $\mathcal{L}_n(\mathcal{T})$ to denote the *n-slice* of $\mathcal{L}(\mathcal{T})$, namely $\mathcal{L}_n(\mathcal{T})$ is the set $\{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}$ of all k-trees of size n in $\mathcal{L}(\mathcal{T})$.

Give a state $s \in S$, we will usually parameterize $\mathcal{T}$ by the initial state s, specifically, we write $\mathcal{T}[s] = (S, \Sigma, \Delta, s)$ for the modification of $\mathcal{T}$ where s is the new initial state. Furthermore, let $\tau = (s, a, w) \in \Delta$ be any transition. We denote by $\mathcal{T}[\tau] = (S, \Sigma, \Delta \cup \{(s^\star, a, w)\}, s^\star)$ where $s^\star$ is a fresh state not in Q. In other words, $\mathcal{T}[\tau]$ is the extension $\mathcal{T}$ that recognizes trees where runs are forced to start with transition τ.

A binary labeled tree t is a labeled 2-tree such that every node has two children or is a leaf. Notice that 2-trees are different from binary trees, as in the former a node can have a single child, while in the latter this is not allowed. For every non-leaf $u \in t$, we denote by $u \cdot 1$ and $u \cdot 2$ the left and right child of u, respectively. Similar than for k-trees, we denote by $\mathrm{Trees}_{\mathrm{B}}[\Sigma]$ the set of all binary trees. We say that a tree automaton $\mathcal{T} = (S, \Sigma, \Delta, s_{\text{init}})$ is over $\mathrm{Trees}_{\mathrm{B}}[\Sigma]$ if $\Delta \subseteq S \times \Sigma \times (\{\lambda\} \cup S^2)$.

12.2.2 Approximate Counting, Almost Uniform Sampling, and Parsimonious Reductions

Definition of FPRAS. Given an input alphabet Σ, a *randomized approximation scheme* (RAS) for a function $f : \Sigma^* \to \mathbb{R}$ is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0, 1) \to \mathbb{R}$ such that for every $w \in \Sigma^*$ and $\epsilon \in (0, 1)$:

$$\mathbf{Pr}\left[|\mathcal{A}(w, \epsilon) - f(w)| \leq \epsilon \cdot f(w)\right] \quad \geq \quad \frac{3}{4}.$$

A randomized algorithm $\mathcal{A} : \Sigma^* \times (0, 1) \to \mathbb{R}$ is a *fully polynomial-time randomized approximation scheme* (FPRAS) [JVV86a] for f, if it is a randomized approximation scheme for f and, for every $w \in \Sigma^*$ and $\epsilon \in (0, 1)$, $\mathcal{A}(w, \epsilon)$ runs in polynomial time over $|w|$ and ϵ^{-1}. Thus, if $\mathcal{A}$ is an FPRAS for f, then $\mathcal{A}(w, \epsilon)$ approximates the value $f(w)$ with a relative error of $(1 \pm \epsilon)$, and

it can be computed in polynomial time in the size w and ϵ^{-1}.

Definition of FPAUS. In addition to polynomial time approximation algorithms, we also consider polynomial time (almost) uniform samplers. Given an alphabet Σ and a finite universe Ω, let $g : \Sigma^* \to 2^\Omega$. We say that g admits a *fully polynomial-time almost uniform sampler* (FPAUS) [JVV86a] if there is a randomized algorithm $\mathcal{A} : \Sigma^* \times (0,1) \to \Omega \cup \{\perp\}$ such that for every $w \in \Sigma^*$ with $g(w) \neq \emptyset$, and $\delta \in (0,1)$, $\mathcal{A}(w,\delta)$ outputs a value $x^* \in g(w) \cup \{\perp\}$ with

$$\mathbf{Pr}\left[x^* = x\right] = (1 \pm \delta)\frac{1}{|g(w)|} \quad \text{for all } x \in g(w)$$

and, moreover, $\mathcal{A}(w,\delta)$ runs in polynomial time over $|w|$ and $\log \frac{1}{\delta}$. If $g(w) = \emptyset$, a FPAUS must output a symbol $\perp$ with probability 1. The symbol $\perp$ can be thought of as a "failure" symbol, where the algorithm produces no output. Notice that whenever $g(w)$ admits a deterministic polynomial time membership testing algorithm (i.e. to test if $x \in g(w)$), it is easy to ensure that a sampler only outputs either a element $x \in g(w)$ or $\perp$. Also notice that the conditions imply that if $g(w) \neq \emptyset$, we have $\mathbf{Pr}\left[x^* = \perp\right] \leq \delta$. Given a set $S = g(w)$, when the function g and the input w is clear from context, we will say that the set S admits an FPAUS to denote the fact that g admits an FPAUS.

For an example of an FPAUS, w could be the encoding of a non-deterministic finite automata $\mathcal{N}$ and a number $n \in \mathbb{N}$ given in unary, and $g(w)$ could be the set of strings of length n accepted by $\mathcal{N}$. A poly-time almost uniform sampler must then generate a string from $\mathcal{L}_n(\mathcal{N})$ from a distribution which is pointwise a $(1 \pm \delta)$ approximation of the uniform distribution over $\mathcal{L}_n(\mathcal{N})$, output $\perp$ with probability at most δ, and run in time $\mathrm{poly}(|\mathcal{N}|, n, \log \frac{1}{\delta})$. Notice that an FPAUS must run in time $\mathrm{poly}(\log \frac{1}{\delta})$, whereas an FPRAS may run in time $\mathrm{poly}(\frac{1}{\epsilon})$.

Parsimonious Reduction. Finally, given functions $f, g : \Sigma^* \to \mathbb{N}$, a polynomial-time parsimonious reduction from f to g is a polynomial-time computable function $h : \Sigma^* \to \Sigma^*$ such that, for every $w \in \Sigma^*$, it holds that $f(w) = g(h(w))$. If such a function h exists, then we use notation $f \leq_{\text{PAR}} g$. Notice that if $f \leq_{\text{PAR}} g$ and g admits an FPRAS, then f admits an FPRAS.

12.2.3 The counting problems for tree automata

The following is the main counting problem studied in this paper regarding tree automata:

Problem:	#TA
Input:	A tree automaton $\mathcal{T}$ over $\mathrm{Trees}_k[\Sigma]$ and a string 0^n
Output:	$\lvert\mathcal{L}_n(\mathcal{T})\rvert$

By the results in [CDG+07] about encoding k-trees as binary trees using an extension operator @, it is possible to conclude the following:

Lemma 12.2.1. *Let Σ be a finite alphabet and $@ \notin \Sigma$. Then there exists a polynomial-time algorithm that, given a tree automata $\mathcal{T}$ over $\mathrm{Trees}_k[\Sigma]$, produces a tree automaton $\mathcal{T}'$ over $\mathrm{Trees}_{\mathrm{B}}[\Sigma \cup \{@\}]$ such that, for every $n \geq 1$:*

$$\Big|\{t \mid t \in \mathcal{L}(\mathcal{T}) \text{ and } |t| = n\}\Big| \quad = \quad \Big|\{t' \mid t' \in \mathcal{L}(\mathcal{T}') \text{ and } |t'| = 2n-1\}\Big|$$

Therefore, we also consider in this paper the following problem:

Problem:	#BTA
Input:	A tree automaton $\mathcal{T}$ over $\mathrm{Trees}_{\mathrm{B}}[\Sigma]$ and a string 0^n
Output:	$\lvert\mathcal{L}_n(\mathcal{T})\rvert$

As we know from Lemma 12.2.1 that there exists a polynomial-time parsimonious reduction from #TA to #BTA, we can show that #TA admits an FPRAS by proving that #BTA admits an FPRAS.

12.3 From Conjunctive Queries to Tree Automata

In this section, we provide the formal link between Conjunctive Queries (CQ) and tree automata, and formalize our results for the former. In particular, we show that it is possible to reduce #ACQ to #TA, where #ACQ is the problem of counting the number of solutions to an acyclic CQ. Hence, the existence of an FPRAS for #ACQ is inferred from the existence of an FPRAS for #TA. In fact, we will prove the more general statement that one can reduce the problem of counting solutions to CQ's with *bounded-hypertree width*, and unions of such queries, to the problem of #TA.

We start by formally introducing conjunctive queries. We first fix two disjoint (countably) infinite sets $\mathbf{C}$ and $\mathbf{V}$ of constants and variables, respectively. Then a conjunctive query (CQ) is

an expression of the form:

$$Q(\bar{x}) \quad \leftarrow \quad R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n), \tag{12.2}$$

where for every $i \in \{1, \ldots, n\}$, R_i is a k_i-ary relation symbol ($k_i \geq 1$) and $\bar{u}_i$ is a k_i-ary tuple of variables and constants (that is, elements from $\mathbf{V}$ and $\mathbf{C}$), and $\bar{x} = (x_1, \ldots, x_m)$ is a tuple of variables such that each variable x_i in $\bar{x}$ occurs in some $\bar{u}_i$. The symbol Q is used as the name of the query, and var(R_i) is used to denote the set of variables in relation symbol R_i. Moreover, var(Q) denotes the set of all variables appearing in the query (i.e., left- and right-hand sides).

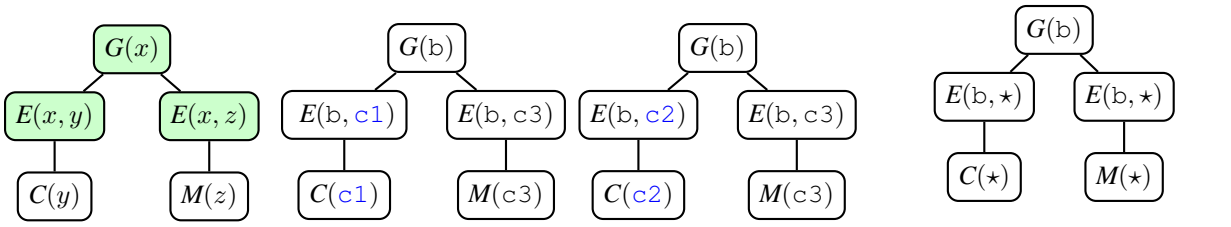
Intuitively, the right-hand side $R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$ of Q is used to specify a pattern over a database, while the tuple $\bar{x}$ is used to store the answer to the query when such a pattern is found. More precisely, a database D is a set of facts of the form $T(\bar{a})$ where $\bar{a}$ is a tuple of constants (elements from $\mathcal{C}$), which indicates that $\bar{a}$ is in the table T in D. Then a homomorphism from Q to D is a function h from the set of variables occurring in Q to the constants in D such that for every $i \in \{1, \ldots, n\}$, it holds that $R_i(h(\bar{u}_i))$ is a fact in D, where $h(\bar{u}_i)$ is obtained by applying h to each component of $\bar{u}_i$ leaving the constants unchanged. Moreover, given such a homomorphism h, the tuple of constants $h(\bar{x})$ is said to be an answer to Q over the database D, and $Q(D)$ is defined as the set of answers of Q over D.

12.3.1 High-level overview of the reduction to #TA

We now give a high-level overview of our reduction to #TA from a simple class of acyclic conjunctive queries. This overview will be sufficient to provide intuition for why tree automata are the correct tool for representing the number of solutions to such conjunctive queries. Consider a CQ:

$$Q_1(x) \leftarrow G(x), E(x, y), E(x, z), C(y), M(z)$$

This query is said to be acyclic, because it can be encoded by a *join tree*, that is, by a tree t where each node is labeled by the relations occurring in the query, and which satisfies the following connectedness property: each variable in the query induces a connected subtree of t [Yan81]. In particular, a join tree for $Q_1(x)$ is depicted in Figure 12.1a, where the connected subtree induced by variable x is marked in green. An acyclic conjunctive query Q can be efficiently evaluated by using a join tree t encoding it [Yan81]; in fact, a tree *witnessing* the fact that $\bar{a} \in Q(D)$ can be constructed in polynomial time. For example, if $D_1 = \{G(\mathtt{a}), G(\mathtt{b}), E(\mathtt{a}, \mathtt{c1}), E(\mathtt{b}, \mathtt{c1}), E(\mathtt{b}, \mathtt{c2}), E(\mathtt{b}, \mathtt{c3}), C(\mathtt{c1}), C(\mathtt{c2}), M(\mathtt{c3})\}$, then $\mathtt{b}$ is an answer to Q_1 over D_1. In fact, two



(a) A join tree.

(b) Two witness trees for the answer b.

(c) An anonymous tree for b.

Figure 12.1: Join, witness and anonymous trees for a CQ.

witness trees for this answer are shown in Figure 12.1b. Notice that the assignments to variable y that distinguish these two trees are marked in blue.

In this work, we consider the following problem:

Problem:	#ACQ
Input:	An acyclic CQ Q and a database D
Output:	$\lvert Q(D)\rvert$

One might think that #ACQ can also be solved in polynomial time given that the number of witness trees can be counted in polynomial time. However, there is no one-to-one correspondence between the answers to an acyclic CQ and their witness trees; as shown in Figure 12.1b, two trees may witness the same answer. In fact, #ACQ is #P-complete [PS13b].

However, we first observe that in a witness tree t, if only output variables are given actual values and non-output variables are assigned an anonymous symbol $\star$, then there *will* be a one-to-one correspondence between answers to a query and witnesses. Let's us denote such structures as *anonymous* trees, an example of which is given in Figure 12.1c. But how can we specify when an anonymous tree is valid? For example, if t' is the anonymous tree obtained by replacing b by a in Figure 12.1c, then t' is not a valid anonymous tree, because a is not an answer to Q_1 over D_1. We demonstrate that tree automata provide the right level of abstraction to specify the validity of such anonymous trees, so that #ACQ can be reduced to a counting problem over tree automata.

12.3.2 A more general notion of acyclicity

In the high level overview above, we considered a CQ to be cyclic if it could be encoded by a join tree. This corresponds to the notion of the tree-width of a CQ. However, our results apply to a more general notion of acyclicity, known as the *hypertree width* of a CQ. We now formalize

this more general notion of acyclicity. Let Q be a CQ of the form $Q(\bar{x}) \leftarrow R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$. A hypertree for Q is a triple $\langle T, \chi, \xi \rangle$ such that $T = (N, E)$ is a rooted tree, and χ and ξ are node-labeling functions such that for every $p \in N$, it holds that $\chi(p) \subseteq \text{var}(Q)$ and $\xi(p) \subseteq \{R_1, \ldots, R_n\}$. Moreover, $\langle T, \chi, \xi \rangle$ is said to be a hypertree decomposition for Q [GLS02] if the following conditions hold:

- for each atom $i \in \{1, \ldots, n\}$, there exists $p \in N$ s.t. $\text{var}(R_i) \subseteq \chi(p)$;
- for each variable $x \in \text{var}(Q)$, the set $\{p \in N \mid x \in \chi(p)\}$ induces a (connected) subtree of T;
- for each $p \in N$, it holds that

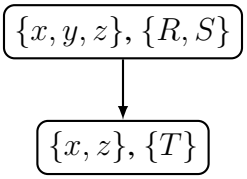
$$\chi(p) \quad \subseteq \quad \bigcup_{R \in \xi(p)} \text{var}(R)$$

- for each $p \in N$, it holds that

$$\left(\bigcup_{R \in \xi(p)} \text{var}(R) \right) \cap \left(\bigcup_{p' \,:\, p' \text{ is a descendant of } p \text{ in } T} \chi(p') \right) \quad \subseteq \quad \chi(p)$$

The width of the hypertree decomposition $\langle T, \chi, \xi \rangle$ is defined as the maximum value of $|\xi(p)|$ over all vertices $p \in N$. Finally, the hypertree width $\text{hw}(Q)$ of CQ Q is defined as the minimum width over all its hypertree decompositions [GLS02].

Example 143. Consider the CQ $Q(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$. It is easy to see that Q is a non-acyclic query (it cannot be represented by a join tree as defined in Section 12.3.1), but we can still study its degree of acyclicity using the idea of hypertree width. In particular, the following is a hypertree decomposition for Q, where the values of $\chi(p)$ and $\xi(p)$ are shown on the left- and right-hand sides of the rectangle for node p:



Notice that the width of this hypertree decomposition is 2, as $|\xi(p)| = 2$ for the root. And in fact, no hypertree decomposition of width 1 can be constructed for Q, so that $\text{hw}(Q) = 2$ (otherwise, Q would be acyclic). In some way, we are forced to bundle two of the atoms (R and S) together and in the process increase the width, in order to create a *join tree*-like structure. □

It was shown in [GLS02] that a CQ Q is acyclic if and only if $\text{hw}(Q) = 1$. Thus, the notion of hypertree width generalizes the notion of acyclicity given before. We are interested in classes of queries with bounded hypertree width, for which it has been shown that the evaluation problem can be solved efficiently [GLS02]. More precisely, for every $k \geq 1$ define the following counting problem.

Problem:	$\#k$-HW
Input:	A CQ Q such that $\text{hw}(Q) \leq k$ and a database D
Output:	$\lvert Q(D) \rvert$

It is important to notice that #ACQ $=$ $\#1$-HW. However, we will keep both languages for historical reasons, as acyclic conjunctive queries were defined two decades earlier, and are widely used in databases. Both #ACQ and $\#k$-HW, for a fixed $k \geq 1$, are known to be #P-complete [PS13b]. On the positive side, based on the relationship with tree automata that we show below, we can conclude that these problems admit FPRAS and a FPAUS, as formalized in Section 12.2.2.

Theorem 144. #ACQ *admits an FPRAS and a FPAUS, and for every constant* $k \geq 1$, $\#k$-HW *admits an FPRAS and a FPAUS.*

Proof. Fix $k \geq 1$. We provide a polynomial-time parsimonious reduction from $\#k$-HW to #TA. In Section 12.5, we will show that #TA admits an FPRAS (see Corollary 12.5.5), which proves that $\#k$-HW admits an FPRAS as well. Moreover, the reduction will be performed in such a way that given a tree accepted by the constructed tree automata $\mathcal{T}$, one can uniquely construct a corresponding $x \in Q(D)$ in polynomial time. As a result, an FPAUS for tree automata implies an FPAUS for CQ's with bounded hypertree width.

Let D be a database and $Q(\bar{x})$ a CQ over D such that its atoms are of the form $R(\bar{t})$ and $\text{hw}(Q) \leq k$. We have from [GLS02] that there exists a polynomial-time algorithm that, given Q, produces a hypertree decomposition $\langle T, \chi, \xi \rangle$ for Q of width k, where $T = (N, E)$. Moreover, $\text{atoms}(Q)$ is used to denote the set of atoms occurring in the right-hand side of Q, and for every $R \in \text{atoms}(Q)$, notation $\bar{t}_R$ is used to indicate the tuple of variables in atom R. Whenever we have atoms indexed like R_i, we shall refer to $\bar{t}_{R_i}$ as $\bar{t}_i$ for the sake of clarity. Also, for every tuple $\bar{x}$ of variables, we use $\text{var}(\bar{x})$ to denote its set of variables, i.e., $\text{var}((x_1, \ldots, x_r)) = \{x_1, \ldots, x_r\}$. Finally, we can assume that $\langle T, \chi, \xi \rangle$ is a complete hypertree decomposition in the sense that for every $R \in \text{atoms}(Q)$, there exists $p \in N$ such that $\text{var}(\bar{t}_R) \subseteq \chi(p)$ and $R \in \xi(p)$ [GLS02]. Finally, let $n = |N|$.

In what follows, we define a tree automaton $\mathcal{T} = (S, \Sigma, \Delta, S_0)$ such that

$$|Q(D)| \;=\; |\{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}|.$$

Notice that for the sake of presentation, we are assuming that $\mathcal{T}$ has a set S_0 of initial states, instead of a single initial state. Such an automaton can be translated in polynomial time into a tree automaton with a single initial state. Given a tuple of variables $\bar{x} = (x_1, \ldots, x_r)$ and a tuple of constants $\bar{a} = (a_1, \ldots, a_r)$, we use notation $\bar{x} \mapsto \bar{a}$ to indicate that variable x_i is assigned value a_i for every $i \in [r]$. Notice that $\bar{x}$ can contain repeated variables, and if this is the case then each occurrence of a repeated variable is assigned the same value. For example, $(x, y, x, y) \to (a, b, a, b)$ is an assignment, while $(x, y, x, y) \to (a, b, a, c)$ is not an assignment if $b \neq c$. Besides, notice that $\emptyset \mapsto \emptyset$ is an assignment. Moreover, two such assignments $\bar{x} \mapsto \bar{a}$ and $\bar{y} \mapsto \bar{b}$ are said to be consistent if for every variable z that occurs both in $\bar{x}$ and $\bar{y}$, it holds that the same value is assigned to z in $\bar{x} \mapsto \bar{a}$ and in $\bar{y} \mapsto \bar{b}$. Then for every $p \in N$ such that:

$$\chi(p) \;=\; \{y_1, \ldots, y_r\} \tag{12.3}$$

$$\xi(p) \;=\; \{R_1, \ldots, R_s\}, \tag{12.4}$$

and assuming that $\chi(p) \cap \text{var}(\bar{x}) = \{z_1, \ldots, z_o\}$, $\bar{y} = (y_1, \ldots, y_r)$ and $\bar{z} = (z_1, \ldots, z_o)$, we define

$$\begin{aligned} S(p) \;=\; \Big\{ \Big[p,\, \bar{y} \mapsto \bar{a},\, \bar{z} \mapsto \bar{b},\, \bar{t}_1 \mapsto \bar{c}_1,\, \ldots,\, \bar{t}_s \mapsto \bar{c}_s\Big] \;\Big|\; & \\ & \bar{R}_i(\bar{c}_i) \text{ is a fact in } D \text{ for every } i \in [s], \\ & \bar{y} \mapsto \bar{a} \text{ is consistent with } \bar{z} \mapsto \bar{b}, \\ & \bar{y} \mapsto \bar{a} \text{ is consistent with } \bar{t}_i \mapsto \bar{c}_i \text{ for every } i \in [s], \\ & \text{and } \bar{t}_i \mapsto \bar{c}_i \text{ is consistent with } \bar{t}_j \mapsto \bar{c}_j \text{ for every } i, j \in [s]\}, \end{aligned}$$

and

$$\Sigma(p) \;=\; \Big\{\Big[p,\, \bar{z} \mapsto \bar{b}\Big] \mid \exists \bar{a} \exists \bar{c}_1 \cdots \exists \bar{c}_s : \Big[p,\, \bar{y} \mapsto \bar{a},\, \bar{z} \mapsto \bar{b},\, \bar{t}_1 \mapsto \bar{c}_1,\, \ldots,\, \bar{t}_s \mapsto \bar{c}_s\Big] \in S(p)\Big\}$$

With this terminology, we define $S_0 = S(p_0)$, where p_0 is the root of the hypertree decomposition $\langle T, \chi, \xi \rangle$, and we define:

$$\begin{aligned} S \;&=\; \bigcup_{p \in N} S(p) \\ \Sigma \;&=\; \bigcup_{p \in N} \Sigma(p) \end{aligned}$$

Finally, the transition relation Δ is defined as follows. Assume again that $p \in N$ satisfies (12.3) and (12.4). If p has children $p_1, \ldots, p_\ell$ in T, where $\ell \geq 1$ and for every $i \in [\ell]$:

$$\begin{aligned} \chi(p_i) &= \{u_{i,1}, \ldots, u_{i,r_i}\} \\ \xi(p_i) &= \{R_{i,1}, \ldots, R_{i,s_i}\}, \end{aligned}$$

with $s_i \leq k$. Then assuming that $\chi(p_i) \cap \mathrm{var}(\bar{x}) = \{w_{i,1}, \ldots, w_{i,o_i}\}$, $\bar{u}_i = (u_{i,1}, \ldots, u_{i,r_i})$ and $\bar{w}_i = (w_{i,1}, \ldots, w_{i,o_i})$ for each $i \in [\ell]$, the following tuple is included in Δ

$$\Big(\Big[p,\, \bar{y} \mapsto \bar{a},\, \bar{z} \to \bar{b},\, \bar{t}_1 \mapsto \bar{c}_1,\, \ldots,\, \bar{t}_s \mapsto \bar{c}_s\Big],\, \Big[p,\, \bar{z} \mapsto \bar{b}\Big], \\ \Big[p_1,\, \bar{u}_1 \mapsto \bar{d}_1,\, \bar{w}_1 \mapsto \bar{e}_1,\, \bar{t}_{1,1} \mapsto \bar{f}_{1,1},\, \ldots,\, \bar{t}_{1,s_1} \mapsto \bar{f}_{1,s_1}\Big] \cdots \\ \Big[p_\ell,\, \bar{u}_\ell \mapsto \bar{d}_\ell,\, \bar{w}_\ell \mapsto \bar{e}_\ell,\, \bar{t}_{\ell,1} \mapsto \bar{f}_{\ell,1},\, \ldots,\, \bar{t}_{\ell,s_\ell} \mapsto \bar{f}_{\ell,s_\ell}\Big]\Big)$$

whenever the following conditions are satisfied: (a) $\Big[p,\, \bar{y} \mapsto \bar{a},\, \bar{z} \mapsto \bar{b},\, \bar{t}_1 \mapsto \bar{c}_1, \ldots, \bar{t}_s \mapsto \bar{c}_s\Big] \in S(p)$; (b) $\Big[p_i,\, \bar{u}_i \mapsto \bar{d}_i,\, \bar{w}_i \mapsto \bar{e}_i,\, \bar{t}_{i,1} \mapsto \bar{f}_{i,1},\, \ldots,\, \bar{t}_{i,s_i} \mapsto \bar{f}_{i,s_i}\Big] \in S(p_i)$ for each $i \in [\ell]$; (c) $\bar{t}_i \mapsto \bar{c}_i$ is consistent with $\bar{t}_{j_1,j_2} \mapsto \bar{f}_{j_1,j_2}$ for every $i \in [s]$, $j_1 \in [\ell]$ and $j_2 \in [s_{j_1}]$; and (d) $\bar{t}_{j_1,j_2} \mapsto \bar{f}_{j_1,j_2}$ is consistent with $\bar{t}_{j_3,j_3} \mapsto \bar{f}_{j_3,j_4}$ for every $j_1 \in [\ell]$, $j_2 \in [s_{j_1}]$, $j_3 \in [\ell]$, $j_4 \in [s_{j_3}]$. On the other hand, if p has no children in T, then the following tuple is included in Δ

$$\Big(\Big[p,\, \bar{y} \mapsto \bar{a},\, \bar{z} \mapsto \bar{b},\, \bar{t}_1 \mapsto \bar{c}_1,\, \ldots,\, \bar{t}_s \mapsto \bar{c}_s\Big],\, \Big[p,\, \bar{z} \mapsto \bar{b}\Big],\, \lambda\Big)$$

whenever $\Big[p,\, \bar{y} \mapsto \bar{a},\, \bar{z} \mapsto \bar{b},\, \bar{t}_1 \mapsto \bar{c}_1,\, \ldots,\, \bar{t}_s \mapsto \bar{c}_s\Big] \in S(p)$.

It is straightforward to see that there exists a polynomial-time algorithm that generates S, S_0, Σ and Δ from Q, D and the hypertree decomposition $\langle T, \chi, \xi\rangle$ for Q. In particular, we have that $|S(p)|$ is $O(\|D\|^k)$, where $|S(p)|$ is the number of elements in $S(p)$ and $\|D\|$ is the size of the database D, by definition of $S(p)$ and the fact that $\chi(p) \subseteq \bigcup_{R \in \xi(p)} \mathrm{var}(\bar{t}_R)$. Notice that this implies that each $S(p)$ is of polynomial size given that k is fixed and each tuple in S is of polynomial size in $\|D\|$. Moreover, observe that as $n = |N|$, we can construct the (unary) input 0^n for the problem #TA in polynomial time in the size of Q, given that the hypertree decomposition $\langle T, \chi, \xi\rangle$ is of polynomial size in the size of Q.

Finally, we need to prove that $|Q(D)| = |\{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}|$. To see this, for every $\bar{a} \in Q(D)$, define a labeled tree $t_{\bar{a}}$ as follows. Tree $t_{\bar{a}}$ has the same structure as T, but every node $p \in N$ is assigned the following label in Σ. Assume that $\chi(p) \cap \mathrm{var}(\bar{x}) = \{z_1, \ldots, z_r\}$ and $\bar{z} = (z_1, \ldots, z_r)$. Moreover, assume that z_i receives the value a_i in $\bar{a}$ for every $i \in [r]$. Then

the label of p in $t_{\bar{a}}$ is $[p, \bar{z} \mapsto \bar{a}]$, where $\bar{a} = (a_1, \ldots, a_r)$. By definition of $\mathcal{T}$, we have that $\mathcal{L}(\mathcal{T}) = \{t_{\bar{a}} \mid \bar{a} \in Q(D)\}$. Therefore, given that $t_{\bar{a}} \neq t_{\bar{a}'}$ for every $\bar{a}, \bar{a}' \in Q(D)$ such that $\bar{a} \neq \bar{a}'$, we conclude that $|Q(D)| = |\{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}|$, as every tree accepted by $\mathcal{T}$ has n nodes. Moreover, given any $t_{\bar{a}} \in \mathcal{L}(\mathcal{T})$, one can read off the labels of the nodes in the tree $t_{\bar{a}}$ and uniquely reconstruct the corresponding $\bar{a} \in Q(D)$ in polynomial time, which verifies the second claim that a sample from $\mathcal{L}(\mathcal{T})$ yields in polynomial time a unique sample from $Q(D)$.

□

12.3.3 A characterization of classes of conjunctive queries admitting FPRAS

In this section, we describe how Theorem 144 and the results in [GSS01] can be combined to obtain a characterization of the classes of conjunctive queries which admit an FPRAS. We consider the slightly different notion of conjunctive query used in [GSS01], which explicitly includes equality atoms and does not contain constants. More precisely, in this section a conjunctive query (CQ) is an expression of the form:

$$Q(\bar{x}) \leftarrow \alpha_1(\bar{y}_1), \ldots, \alpha_n(\bar{y}_n), \tag{‡}$$

where for every $i \in \{1, \ldots, n\}$, $\alpha_i(\bar{y}_i)$ is either $R_i(\bar{y}_i)$ with R_i a k_i-ary relation symbol ($k_i \geq 1$) and $\bar{y}_i$ a k_i-ary tuple of variables from $\mathbf{V}$, or $\alpha_i(\bar{y}_i)$ is $y_{i,1} = y_{i,2}$ with $y_{i,1}$, $y_{i,2}$ variables from $\mathbf{V}$. Moreover, $\bar{x} = (x_1, \ldots, x_m)$ is a tuple of variables such that each variable x_i in $\bar{x}$ occurs in some $\bar{y}_i$.

As before, the symbol Q is used as the name of the query, and $\text{var}(Q)$ denotes the set of all variables appearing in Q.

Given a CQ $Q(\bar{x})$ of the form (‡), define a graph G_Q representing Q as follows. The set of vertices of G_Q is $\text{var}(Q)$, and there exists an edge between two distinct variables x and y if, and only if, there exists $i \in \{1, \ldots, n\}$ such that both x and y occur in $\alpha_i(\bar{y}_i)$. We consider here the standard notion of tree-width of a graph G and of a conjunctive query Q [GSS01, FG06], which are denoted by $\text{tw}(G)$ and $\text{tw}(Q)$, respectively. Notice that although $\text{tw}(Q)$ is not defined by using G_Q, it holds that $\text{tw}(Q) = \text{tw}(G_Q)$ [FG06]. Then a class $\mathcal{G}$ of graphs is said to have bounded treewidth if there exists a constant k such that $\text{tw}(G) \leq k$ for every $G \in \mathcal{G}$. Moreover, define $\text{CQ}(\mathcal{G})$ as the class of all conjunctive queries Q of the form (‡) whose representing graph G_Q is in $\mathcal{G}$.

Let $\mathcal{G}$ be a class of graphs, and assume that there exists a constant k such that $\text{tw}(G) \leq k$ for

every $G \in \mathcal{G}$. Then for every $Q \in \text{CQ}(\mathcal{G})$, we have that $\text{hw}(Q) \leq \text{tw}(Q) \leq k$. Moreover, given $Q \in \text{CQ}(\mathcal{G})$, if Q' is a query obtained from Q by replacing each equality atom $x = y$ by $\boldsymbol{EQ}(x, y)$, where $\boldsymbol{EQ}$ is a fresh predicate, then we have that $Q(D) = Q'(D')$ with $D' = D \cup \{\boldsymbol{EQ}(a, a) \mid a \text{ is an element of } D\}$, and we also have that $\text{tw}(Q') = \text{tw}(Q)$ and $\text{hw}(Q') = \text{hw}(Q)$. Therefore, by considering Theorem 144, we conclude that the problems of computing $|Q(D)|$ and of sampling from $Q(D)$, given as input $Q \in \text{CQ}(\mathcal{G})$ and a database D, admit an FPRAS and an FPAUS.

Given a class $\mathcal{G}$ of graphs, the query decision problem for $\text{CQ}(\mathcal{G})$ is the problem of verifying, given a CQ $Q \in \text{CQ}(\mathcal{G})$ and a database D, whether $Q(D) \neq \emptyset$. By the results of [GSS01], assuming that $\text{W}[1] \neq \text{FPT}$, for every class $\mathcal{G}$ of (undirected) graphs, the query decision problem for $\text{CQ}(\mathcal{G})$ is tractable if, and only if, $\mathcal{G}$ has bounded treewidth. Since the existence of an FPRAS or an FPAUS for the set $Q(D)$ of answers of a conjunctive query results in a BPP algorithm for the query decision problem, it follows that if $\text{BPP} = \text{P}$, it is not possible to obtain an FPRAS or an FPAUS for any class of CQs of the form $\text{CQ}(\mathcal{G})$ for a class of graphs $\mathcal{G}$ with unbounded treewidth. As a corollary of this and the discussion in the previous paragraph, we obtain the following characterization of classes of conjunctive queries admitting FPRAS and FPAUS:

Corollary 12.3.1. *Let $\mathcal{G}$ be a class of (undirected) graphs. Then assuming $\text{W}[1] \neq \text{FPT}$ and $\text{BPP} = \text{P}$, the following are equivalent:*

1. *The problems of computing $|Q(D)|$ and of sampling from $Q(D)$, given as input $Q \in \text{CQ}(\mathcal{G})$ and a database D, admit an FPRAS and an FPAUS, respectively.*
2. *$\mathcal{G}$ has bounded treewidth.*

It should be mentioned that a refinement of the result of [GSS01] is given in [Gro07], which can be applied over any recursively enumerable class of conjunctive queries of fixed arity. We do not know whether the results of this paper can be extended to this case, which in particular means proving there exists an FPRAS for each class of CQs whose cores [HN92] have bounded treewidth. We believe this to be an interesting problem for future work.

12.3.4 Union of conjunctive queries

An important and well-studied extension of the class of conjunctive queries is obtained by adding the union operator. A union of conjunctive queries (UCQ) is an expression of the form:

$$Q(\bar{x}) \quad \leftarrow \quad Q_1(\bar{x}) \vee \cdots \vee Q_m(\bar{x}), \tag{12.5}$$

where $Q_i(\bar{x})$ is a conjunctive query of the form (12.2) for each $i \in \{1, \ldots, m\}$, and the same tuple $\bar{x}$ of output variables is used in the CQs $Q_1(\bar{x})$, ..., $Q_m(\bar{x})$. As for the case of CQs, the symbol Q is used as the name of the query. A tuple $\bar{a}$ is said to be an answer of UCQ Q in (12.5) over a database D if and only if $\bar{a}$ is an answer to Q_i over D for some $i \in \{1, \ldots, m\}$. Thus, we have that:

$$Q(D) \quad = \quad \bigcup_{i=1}^{m} Q_i(D)$$

As expected, the problem of verifying, given a UCQ Q, a database D and a tuple of constants $\bar{a}$, whether $\bar{a}$ is an answer to Q over D is an NP-complete problem [CM77b]. Also as expected, the evaluation problem for union of acyclic conjunctive queries can be solved in polynomial time, given that the evaluation problem for acyclic CQs can be solved in polynomial time. Concerning to our investigation, we are interested in the following problem associated to the evaluation problem for union of acyclic conjunctive queries:

Problem:	#UACQ
Input:	A union of acyclic CQ Q and a database D
Output:	$\lvert Q(D) \rvert$

As expected from the result for conjunctive queries, #UACQ is #P-complete [PS13b]. However, #UACQ remains #P-hard even if we focus on the case of UCQs without existentially quantified variables, that is, UCQs of the form (12.5) where $\bar{x}$ consists of all the variables occurring in CQ $Q_i(\bar{x})$ for each $i \in \{1, \ldots, m\}$. Notice that this is in sharp contrast with the case of CQs, where #ACQ can be solved in polynomial time if we focus on the case of CQs without existentially quantified variables [PS13b]. However, by using Theorem 144, we are able to provide a positive result about the possibility of efficiently approximating #UACQ.

Proposition 12.3.2. #UACQ *admits an FPRAS and an FPAUS.*

As a final fundamental problem, we consider the problem of counting the number of solutions of a union of conjunctive queries of bounded hypertree width.

Problem:	$\#k$-UHW
Input:	A database D and a union of CQ $Q(\bar{x}) \leftarrow Q_1(\bar{x}) \vee \cdots \vee Q_m(\bar{x})$ such that $\text{hw}(Q_i) \leq k$ for all $i \in \{1, \ldots, m\}$
Output:	$\lvert Q(D) \rvert$

By using the same ideas as in the proof of Proposition 12.3.2, we obtain from Theorem 144 that:

Proposition 12.3.3. *For every $k \geq 1$, it holds that $\#k$-UHW admits an FPRAS and an FPAUS.*

12.3.5 Some related work on conjunctive queries

Several works have looked into the counting problem for CQs (and the related problems we listed above, like CSPs). In order to clarify the discussion, we will give a rough characterization of the research in this area. This will better illustrate how our results relate to previous work. So as a first idea, when counting solutions to CQs, an important source of difficulty is the presence of existentially quantified variables. Consider the query we used in Section 12.3.1:

$$Q_1(x) \quad \leftarrow \quad G(x), E(x, y), E(x, z), C(y), M(z).$$

Notice that there are three variables x, y and z in the right-hand side, while only x is present in the left-hand side. Thus, x is an output variable, while y and z are existentially quantified variables. An alternative notation for CQs makes the quantification even more explicit:

$$Q_1(x) \quad \leftarrow \quad \exists y \exists z \, (G(x) \wedge E(x, y) \wedge E(x, z) \wedge C(y) \wedge M(z)).$$

As we mention later in Section 12.3, when variables are existentially quantified, there is no one-to-one correspondence between the answers to a CQ and their witness trees. This introduces a level of ambiguity (i.e. potentially several witness trees for each answer) into the counting problem, which makes it more difficult, even though it does not make the evaluation problem any harder. In fact, it is proved in Theorem 4 in [PS13b] that the counting problem is #P-complete for acyclic CQs over graphs (i.e. bounded arity), even if queries are allowed a single existentially quantified variable (and an arbitrary number of output variables). In contrast, it is known (e.g. [DJ04]) that for each class of CQs with bounded treewidth and without existentially quantified variables, the counting problem can be solved exactly in polynomial time.

It was open what happens with the counting problem when CQs are considered with all their features, that is, when output and existentially quantified variables are combined. In particular, it was open whether the counting problem admits an approximation in that case. Our paper aims to study precisely that case, in contrast with previous work that does not consider such output variables combined with existentially quantified variables [Bv20, DJ04].

As a second idea, approaches to make the counting or evaluation problem for CQs tractable usually revolve around imposing some structural constraint on the query, in order to restrict its degree of cyclicity. Most well-known is the result in [Yan81], which proves that the evaluation problem is tractable for acyclic queries. In generalizations of this result (e.g. [GSS01]), the acyclicity is usually measured as the width of some query decomposition. Specific to the counting problem, this type of notion is used in [DM15] to characterize tractable cases. Notice, however, that they rely not only on the width of different query decompositions, but also on a measure of how free variables are spread in the query, which they call *quantified star size*. In contrast, we rely only on the structural width of the hypertree decomposition.

12.4 Technical Overview of the Tree Automata FPRAS

Given the results of Section 12.3, we will now focus on the problem of designing an FPRAS for #TA, which has as input a tree automata $\mathcal{T}$ and an integer $n \geq 1$ (given in unary), and asks to output $|\mathcal{L}_n(\mathcal{T})|$. In this section, we give an overview of the main components of our algorithm, their relation to prior techniques, and the technical challenges involved in designing such an FPRAS.

Binary tree automata

To capture the essence of the problem, in the following discussion we consider a simplified version of tree automata. Specifically, we restrict the discussion to *unlabeled* binary ordered trees, which are sufficient to present the main ideas of the algorithm. A binary ordered tree t (or just tree) is a rooted binary tree where the children of each node are ordered; namely, one can distinguish between the left and right child of each non-leaf node. For a non-leaf node u of t, we write $u1$ and $u2$ to denote the left and right children of u, respectively, and we denote the root of any tree t by λ, which representes the empty string. We will write $u \in t$ to denote that u is a node of t, and $|t|$ to denote the number of nodes of t. For example, Figure 12.2 depicts a binary

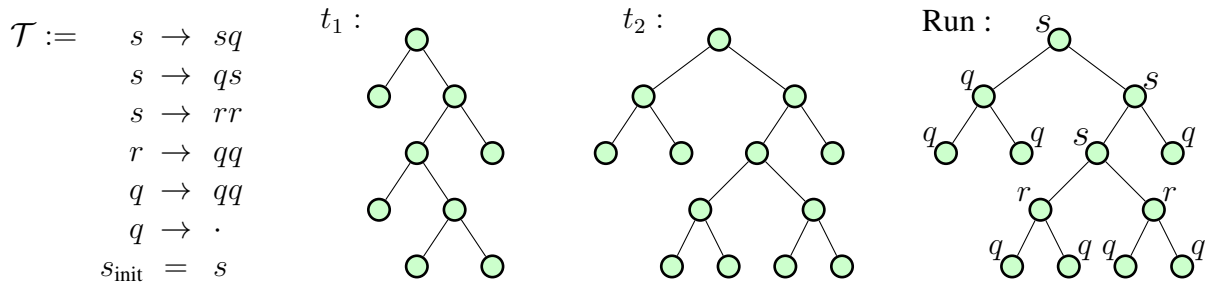


Figure 12.2: A tree automata $\mathcal{T}$, binary ordered trees t_1 and t_2, and a run of $\mathcal{T}$ over t_2.

ordered tree t_1 with $|t_1| = 9$, and another tree t_2 with $|t_2| = 13$, where the children are ordered from left to right.

A tree automaton $\mathcal{T}$ over a binary ordered tree is defined as a tuple $(S, \Delta, s_{\text{init}})$ where S is a finite set of states, $\Delta \subseteq (S \times S \times S) \cup S$ is the transition relation, and $s_{\text{init}} \in S$ is the initial state. A *run* ρ of $\mathcal{T}$ over a tree t is a function $\rho : t \to S$ mapping nodes to states that respects the transition relation. Namely, for every node u of t we have $\rho(u) \in \Delta$ whenever u is a leaf, and $(\rho(u), \rho(u1), \rho(u2)) \in \Delta$, otherwise. We say that $\mathcal{T}$ accepts t if there exists a run ρ of $\mathcal{T}$ over t such that $\rho(\lambda) = s_{\text{init}}$, and such a run ρ is called an accepting run of $\mathcal{T}$ over t. The set of all trees accepted by $\mathcal{T}$ is denoted by $\mathcal{L}(\mathcal{T})$, and the n-slice of $\mathcal{L}(\mathcal{T})$, denoted by $\mathcal{L}_n(\mathcal{T})$, is the set of trees $t \in \mathcal{L}(\mathcal{T})$ with size n (that is, $|t| = n$). For the sake of presentation, in the following we write $s \to qr$ to represent the transition $(s, q, r) \in \Delta$ and $s \to \cdot$ to represent $s \in \Delta$. Note that transitions of the form $s \to \cdot$ correspond to leaves that have no children, and can be thought as "final states" of a run.

Figure 12.2 gives an example of a tree automaton $\mathcal{T}$ with states $\{s, r, q\}$. The right-hand side of Figure 12.2 shows an example of an accepting run of $\mathcal{T}$ over t_2. One can easily check from the transitions of $\mathcal{T}$ in this example that a tree t is in $\mathcal{L}(\mathcal{T})$ if, and only if, there exists a node $v \in t$ such that both children of v are internal (non-leaf) nodes. For example, t_2 satisfies this property and $t_2 \in \mathcal{L}(\mathcal{T})$. On the other hand, all nodes $v \in t_1$ have at least one child that is a leaf, and thus there is no accepting run of $\mathcal{T}$ over t_1, so $t_1 \notin \mathcal{L}(\mathcal{T})$.

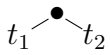
Unrolling the automaton

Fix $n \geq 1$ and a tree automaton $\mathcal{T} = (S, \Delta, s_{init})$ as defined above. Our first step will be to *unroll* the automaton, so that each state is restricted to only producing trees of a fixed size. Specifically, we construct an automaton $\overline{\mathcal{T}} = (\overline{S}, \overline{\Delta}, s^n_{\text{init}})$, where each state $s \in S$ is duplicated n times into

$s^1, s^2, \ldots, s^n \in \overline{S}$, and where s^i is only allowed to derive trees of size i. To enforce this, each transition $s \to rq$ in Δ is replaced with $s^i \to r^j q^k \in \overline{\Delta}$ for all $j, k > 0$ such that $i = j + k + 1$, and each transition $s \to \cdot$ in Δ is replaced with $s^1 \to \cdot \in \overline{\Delta}$. Now for every $s \in S$, let $T(s^i)$ be set of trees that can be derived beginning from the state s^i (all of which have size i). When $i > 1$, we can define $T(s^i)$ via the relation

$$T(s^i) = \bigcup_{(s^i \to r^j q^k) \in \overline{\Delta}} \Big(T(r^j) \otimes T(q^k) \Big) \tag{12.6}$$

where $T(r^j) \otimes T(q^k)$ is a shorthand to denote the set of all trees that can be created by taking every $t_1 \in T(r^j)$ and $t_2 \in T(q^k)$ and forming the tree:



This fact allows us to define each set $T(s^i)$ recursively as a union of "products" of other such sets. Our goal is then to estimate $|T(s^n_{\text{init}})|$ and sample from $T(s^n_{\text{init}})$.

It should be mentioned that for so-called "bottom-up deterministic" automata $\mathcal{T}$ [CDG+07], the sets $T(r^j) \otimes T(q^k)$ in the union in Equation (12.6) are disjoint, so

$$|T(s^i)| \quad = \sum_{(s^i \to r^j q^k) \in \overline{\Delta}} |T(r^j)| \cdot |T(q^k)|$$

and one can then compute the values $|T(s^i)|$ exactly via dynamic programming. Thus, the core challenge is the *ambiguity* of the problem: namely, the fact that trees $t \in T(s^n_{\text{init}})$ may admit exponentially many runs in the automaton. For example, the tree automaton $\mathcal{T}$ from Figure 12.2 can accept t_2 by two different runs. In what follows, we will focus on the problem of uniform sampling from such a set $T(s^i)$, since given a uniform sampler the problem of size estimation is routine.

A QPRAS via Karp-Luby sampling

To handle the problem of sampling with ambiguous derivations Gore et al. [GJK+97] used a technique known as *Karp-Luby* sampling. This technique is a form of rejection sampling, where given sets $T_1, \ldots, T_k$ and $T = \cup_i T_i$, one can sample from T via the following procedure:

(1) Sample a set T_i with probability proportional to $|T_i|$.

(2) Sample an element t uniformly from T_i.

(3) Accept t with probability $1/m(t)$, where $m(t)$ is the total number of sets T_j which contain t.

The QPRAS of [GJK+97] applied this procedure recursively, using approximations $\widetilde{N}(T_i)$ in the place of $|T_i|$, where the union $T = \cup_i T_i$ in question is just the union in Equation (12.6), and each T_i is a product of smaller sets $T_i = T_{i,1} \otimes T_{i,2}$ which are themselves unions of sets at a lower depth. So to carry out **(2)**, one must recursively sample from $T_{i,1}$ and $T_{i,2}$. The overall probability of acceptance in **(3)** is now exponentially small in the sampling depth. Using a classic depth reduction technique [VSBR83], they can reduce the depth to $\log(n)$, but since $m(t)$ can still be as large as $\Omega(n|\mathcal{T}|)$ at each step, the resulting acceptance probability is quasi-polynomially small.

A partition based approach

The difficult with Karp-Luby sampling is that it relies on a rejection step to compensate for the fact that some elements can be sampled in multiple ways. Instead, our approach will be to *partition* the sets in question, so that no element can be sampled in more than one way. Simply put, to sample from T, we will first partition T into disjoint subsets $T'_1, \ldots, T'_\ell$. Next, we sample a set T'_i with probability (approximately) proportional to $|T'_i|$, and lastly we set $T \leftarrow T'_i$ and now recursively sample from the new T. The recursion ends when the current set T has just one element. Clearly no rejection procedure is needed now for the sample to be approximately uniform. To implement this template, however, there are two main implementation issues which we must address. Firstly, how to partition the set T, and secondly, how to efficiently estimate the size of each part T'_i. In the remainder, we will consider these two issues in detail.

Our High-Level Sampling Template

Input: Arbitrary set T.

1. If $|T| = 1$, return T. Otherwise, find some partition
$$T = \cup_{i=1}^{\ell} T'_i$$
2. Call subroutine to obtain estimates $\widetilde{N}(T'_i) \approx |T'_i|$
3. Set $T \leftarrow T'_i$ with probability $\widetilde{N}(T'_i)/\sum_j \widetilde{N}(T'_j)$, and recursively sample from T.

For the rest of this section, fix some state s^i. It will suffice to show how to generate a uniform

sample from the set $T(s^i)$. To implement the above template, we will rely on having inductively pre-computed estimates of $|T(r^j)|$ for every $r \in S$ and $j < i$. Specifically, our algorithm proceeds in rounds, where on the j-th round we compute an approximation $\widetilde{N}(r^j) \approx |T(r^j)|$ for each state $r \in S$. In addition to these estimates, a key component of our algorithm is that, on the i-th round, we also store *sketches* $\widetilde{T}(r^j)$ of each set $T(r^j)$ for $j < i$, which consist of polynomially many uniform samples from $T(r^j)$. One can uses these sketches $\widetilde{T}(r^j)$ to aid in the generation of uniform samples for the larger sets $T(s^i)$ on the i-th round. For instance, given a set of trees $T = \cup_{j=1}^{k} T_j$ for some sets $T_1, \ldots, T_k$ where we have estimates $\widetilde{N}(T_j) \approx |T_j|$ and sketches $\widetilde{T}_j \subseteq T_j$, one could estimate $|T|$ by the value

$$\sum_{j=1}^{k} \widetilde{N}(T_j) \left(\frac{\left|\widetilde{T}_j \setminus \cup_{j'<j} T_{j'}\right|}{\left|\widetilde{T}_j\right|} \right) \tag{12.7}$$

Here, the term in parenthesis in 12.7 estimates the fraction of the set $\widetilde{T}_j$ which is not already contained in the earlier sets $T_{j'}$.

The partition scheme for NFA

The above insight of sketching the intermediate subproblems $T(r^j)$ of the dynamic program and applying 12.7 was first made in our paper [ACJR19], where we developed an FPRAS for non-deterministic finite automata (NFA). Given an NFA $\mathcal{N}$ with states S, $\Sigma = \{0, 1\}$, and any state $s \in S$ of $\mathcal{N}$, one can similarly define the intermediate subproblem $W(s^i)$[3] as the set of *words* of length i that can be derived starting at the state s. The FPRAS of [ACJR19] similarly pre-computes sketches for these sets in a bottom-up fashion. To sample a string $w = w_1 \cdots w_i \in W(s^i)$, they sampled the symbols in w bit by bit, effectively "growing" a prefix of w. First, $W(s^i)$ is partitioned into $W(s^i, 0) \cup W(s^i, 1)$, where $W(s^i, b) \subseteq W(s^i)$ is the subset of strings $x = x_1 \cdots x_i \in W(s^i)$ with first bit x_1 equal to b. If for any prefix w', we define $R_{w'} \subseteq S$ to be the set of states r such that there is a path of transitions from s to r labeled by w', then observe that $W(s^i, b) = \{b\} \cdot \cup_{r \in R_b} W(r^{i-1})$, where $\cdot$ is the concatenation operation for sets of words. Thus $|W(s^i, b)|$ can be estimated directly by Equation 12.7 in polynomial time. After the first bit $w_1 = b$ is sampled, they move on to sample the second bit w_2 conditioned on the prefix $w_1 = b$. By partitioning the strings again into those with prefix equal to either $b0$ or $b1$, each of which is described compactly as $\{bb'\} \cdot \cup_{r \in R_{bb'}} W(r^{i-2})$ for $b' \in \{0, 1\}$, one can use Equation 12.7 again to sample w_2 from the correct distribution, and so on.

[3] We use W to denote sets of words, and T for sets of trees.

The key "victory" in the above approach is that for NFAs, one can compactly condition on a prefix w' of a word $w \in W(s^i)$ as a union $\cup_{r \in R_{w'}} W(r^{i-|w'|})$ taken over some easy to compute subset of states $R_{w'} \subseteq S$. In other words, to condition on a partial derivation of a word, one need only remember a subset of states. This is possible because, for NFAs, the overall configuration of the automata at any given time is specified only by a single current state of the automata. However, this fact breaks down fundamentally for tree automata. Namely, at any intermediate point in the derivation of a tree, the configuration of a tree automata is described not by a single state, but rather by the combination of states $(r_{t_1}^{j_1}, \ldots, r_{t_k}^{j_k})$ assigned to the (possibly many) leaves of the partially derived tree. So the number of possible configurations is exponential in the number of leaves of the partial tree. Consequentially, the number of sets in the union of Equation 12.7 is exponentially large.[4] Handling this lack of a compact representation is the main challenge for tree automata, and will require a substantially different approach to sampling.

The partition scheme for tree languages

Similarly at a high level to the word case, our approach to sampling will be to "grow" a tree t from the root down. However, unlike in the word case, there is no longer any obvious method to partition the ways to grow a tree (for words, one just partitions by the next bit in the prefix). Our solution to this first challenge is to partition based on the *sizes* of the subtrees of all the leaves of t. Namely, at each step we expand one of the leaves ℓ of t, and choose what the final sizes of the left and right subtrees of ℓ will be. By irrevocably conditioning on the final sizes of the left and right subtrees of a leaf ℓ, we partition the set of possibles trees which t can grow into based on the sizes that we choose. Importantly, we do **not** condition on the states which will be assigned to any of the vertices in t, since doing so would no longer result in a partition of $T(s^i)$.

More formally, we grow a *partial tree* τ, which is an ordered tree with the additional property that some of its leaves are labeled with positive integers, and these leaves are referred to as *holes*. For an example, see the leftmost tree in Figure 12.3. A partial tree τ is called complete if it has no holes. For a hole H of τ, we denote its integral label by $\tau(H) \geq 1$, and call $\tau(H)$ the *final size* of H, since $\tau(H)$ will indeed be the final size of the subtree rooted at H once τ is complete. Intuitively, to complete τ we must replace each hole H of τ with a subtree of size exactly $\tau(H)$. Because no states are involved in this definition, a partial tree τ is by itself totally independent of the automata.

[4] By being slightly clever about the order in which one derives the tree, one can reduce the number of "active" leafs to $O(\log n)$, which would result in a quasi-polynomial $|S|^{O(\log n)}$ time algorithm following the approach of [ACJR19], which in fact is a slight improvement on the $(|S|n)^{O(\log n)}$ obtained from [GJK+97].

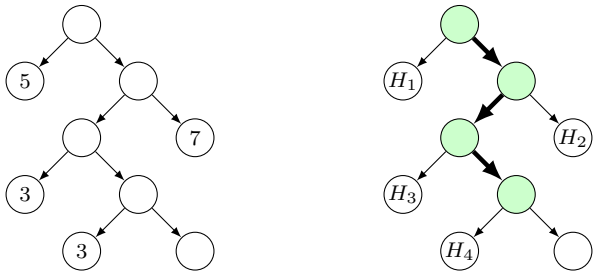


Figure 12.3: Two examples of partial trees. The left-hand side tree shows the label of each hole written inside the node. The right-hand side tree illustrates the main path, where non-white (green) nodes and thick arcs are used to highlight the vertices and edges on the main path.

We can now define the set $T(s^i, \tau) \subseteq T(s^i)$ of *completions* of τ as the set of trees $t \in T(s^i)$ such that τ is a subtree of t sharing the same root, and such that for every hole $H \in \tau$ the subtree rooted at the corresponding node $H \in t$ has size $\tau(H)$. Equivalently, t can be obtained from τ by replacing each hole $H \in \tau$ with a subtree t_H of size $\tau(H)$. If $\textcircled{i}$ is the partial tree consisting of a single hole with final size i, then we have $T(s^i, \textcircled{i}) = T(s^i)$. So at each step in the construction of τ, beginning with $\tau = \textcircled{i}$, we will attempt to sample a tree t uniformly from $T(s^i, \tau)$. To do so, we can pick any hole $H \in \tau$, and expand it by adding left and right children and fixing the final sizes of the subtrees rooted at those children. There are $\tau(H)$ ways of doing this: namely, we can fix the final size of the left and right subtrees to be j and $\tau(H) - j - 1$ respectively, for each $j \in \{0, 1, \ldots, \tau(H) - 1\}$. So let τ_j be the partial tree resulting from fixing these final sizes to be j and $\tau(H) - j - 1$, and notice that $T(s^i, \tau_0), \ldots, T(s^i, \tau_{\tau(H)-1})$ partitions the set $T(s^i, \tau)$. Thus it will now suffice to efficiently estimate the sizes $|T(s^i, \tau_j)|$ of each piece in the partition.

Estimating the number of completions via the main path

The remaining challenge can now be rephrased in following way: given any partial tree τ, design a subroutine to estimate the number of completions $|T(s^i, \tau)|$. The key tool in our approach to this is a reduction which allows us to represent the set $T(s^i, \tau)$ as the language generated by a succinct NFA, whose transitions are labeled by large sets which are succinctly encoded (see earlier definition Theorem 142). In our reduction, on round $i \leq n$, the alphabet Σ of the succinct NFA will be the set of all ordered trees of size at most i. Note that this results in Σ and the label sets A being exponentially large in n, preventing one from applying the algorithm of [ACJR19].

Our first observation is that by always choosing the hole H at the lowest depth to expand in the partitioning scheme, the resulting holes $H_1, \ldots, H_k \in \tau$ will be *nested* within each other.

Namely, for each $i > 1$, H_i will be contained in the subtree rooted at the sibling of H_{i-1}. Using this fact, we can define a distinguished path P between the parent of H_1 and the parent of H_k. Observe that each hole H_j must be a child of some node in P. We call P the *main path* of τ (see Figure 12.3). For simplicity, assume that each vertex $v \in P$ has exactly one child that is a hole of τ,[5] and label the vertices of the path $P = \{v_1, v_2, \ldots, v_k\}$, so that H_j is the child of v_j. Notice by the above nestedness property, the holes $H_j, H_{j+1}, \ldots, H_k$ are all contained in the subtree rooted at v_j.

Observe that any completed tree $t \in T(s^i, \tau)$ can be uniquely represented by the trees $(t_1, \ldots, t_k)$, such that t is obtained from τ by replacing each hole $H_i \in \tau$ by the tree t_i. Thinking of each tree t_i as a symbol in the alphabet Σ of all ordered trees, we can thus specify the tree t by a *word* $t_1 t_2 \cdots t_k \in \Sigma^*$. So our goal is to show that the set of words $T(s^i, \tau) = \{t_1 \cdots t_k \in \Sigma^* \mid t_1 \cdots t_k \in T(s^i, \tau)\}$ is the language accepted by an succinct NFA $\mathcal{N}$ over the alphabet of ordered trees Σ with polynomially many states and set-labeled transitions.

Now NFAs can only express labeled paths (i.e., words) and not trees. However, the key observation is that if we restrict ourselves to the main path P, then the sequences of states from $\mathcal{T}$ which can occur along P can indeed be expressed by an NFA. Informally, for every vertex $v_j \in P$ with (wlog) left child H_j, and for every transition $s \to rs'$ in the tree automata $\mathcal{T}$ which could occur at v_j, we create a unique transition $s \to s'$ in the succinct NFA. Here, the two states s, s' are assigned to the vertices v_j, v_{j+1} on the main path P, and the state r is placed inside of the hole H_j. Now the set of trees t_j which could be placed in H_j by this transition *only* depends on the state r. Specifically, this set of trees is exactly $T(r^{\tau(H_j)})$. Thus, if we label this transition $s \to s'$ in the succinct NFA by the set $T(r^{\tau(H_j)})$, the language accepted by the NFA will be precisely $T(s^i, \tau)$. The full details can be found in the full version.

The crucial fact about this construction is that the transition labels of the succinct NFA are all sets of the form $T(r^j)$ for some $r \in S$ and $j < i$. Since $j < i$, our algorithm has already pre-computed the sketches $\widetilde{T}(r^j)$ and estimates $\widetilde{N}(T^j)$ of the label sets $T(r^j)$ at this point. We will use these sketches and estimate to satisfy the "oracle" assumptions of Theorem 142.

An FPRAS for Succinct NFAs

Now that we have constructed the succinct NFA $\mathcal{N}$ which recognizes the language $T(s^i, \tau)$ as its k-slice, we must devise a subroutine to approximate the size of the k-slice of $\mathcal{N}$. Let S', Δ' be the

[5] Extra care should be taken when this is not the case.

states and transitions of $\mathcal{N}$. In order to estimate $|\mathcal{L}_k(\mathcal{N})|$, we mimic the inductive, dynamic programming approach of our "outside" algorithm.[6] Namely, we define partial states of a dynamic program on $\mathcal{N}$, by setting $W(x^\ell)$ to be the set of words of length ℓ accepted by $\mathcal{N}$ starting from the state $x \in S'$. We then similarly divide the computation of our algorithm into rounds, where on round ℓ of the subroutine, we inductively pre-compute new NFA sketches $\widetilde{W}(x^\ell)$ of $W(x^\ell)$ and estimates $\widetilde{N}(x^\ell)$ of $|W(x^\ell)|$ for each state $x \in S'$. Given these estimates and sketches, our procedure for obtaining the size estimates $\widetilde{N}(x^\ell)$ is straightforward. Thus, similar to the outside algorithm, the central challenge is to design a polynomial time algorithm to sample from the set $W(x^\ell)$, allowing us to construct the sketch $\widetilde{W}(x^\ell)$.

For a string $u \in \Sigma^*$, define $W(x^\ell, u)$ to be the set of strings $w \in W(x^\ell)$ with prefix equal to u. Recall the approach of [ACJR19] to this problem for standard NFAs began by partitioning $W(x^\ell)$ into $\bigcup_{\alpha\in\Sigma} W(x^\ell, \alpha)$ and estimating the size $|W(x^\ell, \alpha)|$ for each $\alpha \in \Sigma$. Then one chooses α with probability (approximately)

$$\mathbf{Pr}\left[\alpha\right] = \frac{|W(x^\ell, \alpha)|}{\sum_{\beta\in\Sigma} |W(x^\ell, \beta)|}$$

and recurses into the set $W(x^\ell, \alpha)$. Clearly we can no longer follow this strategy, as $|\Sigma|$ is of exponential size with respect to $\mathcal{N}$. Specifically, we cannot estimate $|W(x^\ell, \alpha)|$ for each $\alpha \in \Sigma$. Instead, our approach is to approximate the behavior of the "idealistic" algorithm which *does* estimate all these sizes, by sampling from Σ without explicitly estimating the sampling probabilities $\mathbf{Pr}\left[\alpha\right]$. Namely, for a prefix u we must sample a string $v \sim W(x^\ell, u)$, by first sampling the next symbol $\alpha \sim \Sigma$ from a distribution $\widetilde{\mathcal{D}}(u)$ which is close to the true distribution $\mathcal{D}(u)$ over Σ given by $\mathbf{Pr}\left[\alpha\right] = |W(x^\ell, u \cdot \alpha)|/|W(x^\ell, u)|$ for each $\alpha \in \Sigma$.

To do this, first note that we can write

$$W\left(x^\ell, u\right) = \{u\} \cdot \bigcup_{y\in R(x,u)} W\left(y^{\ell-|u|}\right)$$

where $R(x, u) \subseteq S'$ is the set of states y such that there is a a path of transitions from x to y labeled by sets $A_1 \ldots A_{|u|}$ with $u_j \in A_j$ for each $j \in \{1, \ldots, |u|\}$. Thus the set of possible symbols α that we can append to u is captured by the sets of labels of the transitions out of some state $y \in R(x, u)$. Now consider the set of transitions $\{(y, A, z) \in \Delta' \mid y \in R(x, u)\}$, namely, all transitions out of some state in $R(x, u)$. Furthermore, suppose for the moment that we were given an oracle which generates uniform samples from each label set A of a transition

[6] We think of this subroutine to estimate $|T(s^i, \tau)|$ as being the "inner loop" of the FPRAS.

(y, A, z), and also provided estimates $\widetilde{N}(A)$ of the size of that set $|A|$. Given such an oracle, we design a multi-step rejection procedure to sample a symbol α approximately from $\mathcal{D}(u)$, based on drawing samples from the external oracle and then rejecting them based on intersection ratios of our pre-computed internal NFA sketches $\widetilde{W}(y^{\ell-|u|})$.

Since α is generated by a transition out of $R(x, u)$, we first sample such a transitions with probability proportional to the number of remaining suffixes which could be derived by taking that transition. More specifically, the number of suffixes that can be produced by following a transition (y, A, z) is given by $|A| \cdot |W(z^{\ell-|u|-1})|$, which can be approximated by $\widetilde{N}(A) \cdot \widetilde{N}(z^{\ell-|u|-1})$ using the oracle and our internal estimates. Then if Z is the sum of the estimates $\widetilde{N}(A) \cdot \widetilde{N}(z^{\ell-|u|-1})$ taken over all transitions $\{(y, A, z) \in \Delta' \mid y \in R(x, u)\}$, we choose a transition (y, A, z) with probability $\widetilde{N}(A) \cdot \widetilde{N}(z^{\ell-|u|-1})/Z$ and then call the oracle to obtain a sample $\alpha \sim A$. The sample α now defines a piece $W(x^\ell, u \cdot \alpha)$ of the partition of $W(x^\ell, u)$ which the idealistic algorithm would have estimated and potentially chosen. However, at this point α is not drawn approximately from the correct distribution $\mathcal{D}(u)$, since the sample from the oracle does not taken into account any information about the other transitions which could also produce α. To remedy this, we show that it suffices to accept the symbol α with probability:

$$\frac{\left|\widetilde{W}(z^{\ell-|u|-1}) \setminus \bigcup_{\zeta \in \mathbf{B}(\alpha) : \zeta \prec z} W(\zeta^{\ell-|u|-1})\right|}{\left|\widetilde{W}(z^{\ell-|u|-1})\right|} \tag{†}$$

where $\prec$ is an ordering over S' and $\mathcal{B}(\alpha)$ is the set of all states that can be reached from $R(x, u)$ by reading α, namely, all states ζ such that there exists a transition $(\eta, B, \zeta) \in \Delta'$ with $\eta \in R(x, u)$ and $\alpha \in B$. Otherwise, we reject α. Intuitively, probability (†) is small when the sets of suffixes which could be derived following transitions $\mathcal{B}(\alpha)$ that could also produce α intersect heavily. If this is the case, we have "overcounted" the contribution of the set $W(x^\ell, u \cdot \alpha)$ in the partition, and so the purpose of the probability (†) is to compensate for this fact. We show that this procedure results in samples α drawn from a distribution $\widetilde{\mathcal{D}}(u)$ which is close in statistical distance to the exact distribution $\mathcal{D}(u)$. Furthermore, one can bound the acceptance probability by (†) $\geq 1/\mathrm{poly}(n)$ in expectation over the choice of α, so after repeating the oracle call polynomially many times, we will accept a sample α. Once α is accepted, we condition on it and move to the next symbol, avoiding any recursive rejection sampling.

We now return to the assumption of having a oracle to sample from and approximate the size of the label sets A. By construction, A is a set of trees $T(s^j)$ for which we have pre-computed sketches and estimates $\widetilde{T}(s^j), \widetilde{N}(s^j)$ from the external algorithm. To simulate this oracle, we reuse the samples within the sketches $\widetilde{T}(s^j)$ for each call to the succinct NFA sub-routine, pre-

tending that they are being generated fresh and on the fly. However, since the same sketches must be reused on each call to the subroutine, we lose independence between the samples generated within subsequent calls. Ultimately, though, all that matters is that the estimate of $|T(s^i, \tau)|$ produced by the subroutine is correct. So to handle this, we show that one can condition on a deterministic property of the sketches $\{\widetilde{T}(s^j)\}_{s \in S, j<i}$, so that *every* possible run of the succinct NFA subroutine will yield a good approximation, allowing us to ignore these dependencies.

Lastly, we handle the propagation of error resulting from the statistical distance between $\widetilde{\mathcal{D}}(u)$ and $\mathcal{D}(u)$. This statistical error feeds into the error for the estimates $\widetilde{N}(x^{\ell+1})$ on the next step, both of which feed back into the statistical error when sampling from $W(x^{\ell+1})$, doubling the error at each step. We handle this by introducing an approximate rejection sampling step, inspired by an exact rejection sampling technique due to [JVV86b] (the exact version was also used in [ACJR19]). This approximately corrects the distribution of each sample w, causing the error to increases linearly in the rounds instead of geometrically, which will be acceptable for our purposes.

12.5 An FPRAS and Uniform Sampler for Tree Automata

In this section, we provide an FPRAS for #BTA. Thus, we obtain as well that #TA admits an FPRAS, since by Lemma 12.2.1 there exists a polynomial-time parsimonious reduction from #TA to #BTA.

Fix a tree automaton $\mathcal{T} = (S, \Sigma, \Delta, s_{\text{init}})$ over binary trees and let $n \geq 1$ be a natural number given in unary. We assume that every state in S is mentioned in Δ, and that every symbol in Σ is mentioned in Δ (if that is not the case, then the elements that are not mentioned in Δ can just be removed from the tree automaton). Let m be the size of the tree automaton $\mathcal{T}$, defined as $m = \|\Delta\|$, where $\|\Delta\|$ is the size of the transition relation Δ (represented as a string over an appropriate alphabet). In the following, fix an error parameter $\epsilon > 0$. Since our algorithm will run in time $\text{poly}(n, m, 1/\epsilon)$, we can assume $\epsilon < \frac{1}{(4nm)^{18}}$ without loss of generality. Note that if we are only interested in uniform sampling, we can just fix $\epsilon = 1/\text{poly}(nm)$. Finally, recall that $\mathcal{L}_n(\mathcal{T}) = \{t \mid t \in \mathcal{L}(\mathcal{T}) \text{ and } |t| = n\}$.

Remark 145. We can assume that $m, n = \omega(1)$, since if $n = O(1)$, then the number of unlabeled trees is constant, so the number of labeled trees is a polynomial in m, and we can check whether each such a tree is in $\mathcal{L}(\mathcal{T})$ to compute $|\mathcal{L}_n(\mathcal{T})|$ in polynomial time. If $m = O(1)$, then we can

transform $\mathcal{T}$ into a constant sized deterministic bottom-up tree automaton,[7] and then $|\mathcal{L}_n(\mathcal{T})|$ can be computed in polynomial time by dynamic programming. Thus, for the remainder we can now assume that $n \geq 2$ and $m \geq 3$. □

Unfolding of the tree automaton $\mathcal{T}$. We begin by making a number of copies of the states in $\mathcal{T}$ in order to "unfold" $\mathcal{T}$ into n *levels*. For this, let the new set of states be $\overline{S} = \{s^i \mid i \in [n], s \in S\}$. Intuitively, from s^i we only want to accept trees of size i. This will allow us to define a natural partition scheme for the sampling procedure. To enforce this constraint, we build a new tree automaton $\overline{\mathcal{T}} = (\overline{S}, \Sigma, \overline{\Delta}, s^n_{\text{init}})$ such that for every transition $(s, a, q \cdot r) \in \Delta$ and $i \in [2, n]$, we add the transition $(s^i, a, q^j \cdot r^{i-j-1})$ to $\overline{\Delta}$ for every $j \in [1, i-2]$. Also, for every transition $(s, a, \lambda) \in \Delta$ we add (s^1, a, λ) to $\overline{\Delta}$. We say that i is the *level* of s^i. Note that one can construct the set $\overline{S}$ and the automaton $\overline{\mathcal{T}}$ in polynomial time in the size of $\mathcal{T}$ [CDG+07].

Given the definition of $\overline{\mathcal{T}}$, one can easily check that $\mathcal{L}(\overline{\mathcal{T}}[s^i]) = \{t \in \mathcal{L}(\mathcal{T}[s]) \mid |t| = i\}$ for every $s^i \in \overline{S}$. In particular, we have that $|\mathcal{L}(\overline{\mathcal{T}})| = |\{t \in \mathcal{L}(\mathcal{T}) \mid |t| = n\}|$ and, thus, the goal becomes to estimate $|\mathcal{L}(\overline{\mathcal{T}})| = |\mathcal{L}(\overline{\mathcal{T}}[s^n_{\text{init}}])|$. For clarity of notation, we write $T(s^i)$ for $\mathcal{L}(\overline{\mathcal{T}}[s^i])$ and $N(s^i)$ for $|T(s^i)|$. Note that the goal becomes to estimate $N(s^n_{\text{init}})$.

Remark 146 (Proviso on the sizes of trees)**.** Every binary tree has an odd number of nodes. Thus, we will have that $T(s^{2i}) = \emptyset$ and $N(s^{2i}) = 0$ for each $i \geq 1$. However, to make the notation simpler, we do not limit ourselves to the trees of odd sizes. On the contrary, the algorithms provided in this article are able to compute $N(s^{2i}) = 0$, and also to realize that no sample has to be produced from $T(s^{2i})$. □

Two basic properties, and the estimation of $N(s^i)$. Our algorithm simultaneously computes estimates $\widetilde{N}(s^i)$ for the set sizes $N(s^i)$, as well as *sketches* $\widetilde{T}(s^i)$ of $T(s^i)$ which consist of polynomially many uniform samples from $T(s^i)$. Specifically, at each level i and for every $s \in S$, our algorithm will store an estimate which satisfies $\widetilde{N}(s^i) = (1 \pm i\epsilon)N(s^i)$. At step i, for each $j < i$, our algorithm will also store i distinct independent uniformly sampled subsets sets $\widetilde{T}_1(s^j), \widetilde{T}_2(s^j), \ldots, \widetilde{T}_i(s^j)$ of $T(s^j)$ which satisfy certain deterministic criteria that will result in the correctness of our sampling algorithm on states s^i (see Lemma 12.5.3). Using these estimates $\widetilde{N}(s^j)$ and sketches $\widetilde{T}_i(s^j)$ for $j < i$ as input, we will construct a procedure that allows us to obtain fresh, independent samples from the sets $T(s^i)$ for all $s \in S$. Formally, the properties we need to inductively condition on are as follows:

[7] A tree automaton is bottom-up if it assigns states to a labeled tree t starting from the leaves, and moving toward the root [CDG+07]. In particular, if t is a binary tree, then the transition function is of the form $\Delta : S \times S \times \Sigma \to S$, that is, a state is assigned to a node depending on the states of its two children and its label.

Property 1: For a fixed $i \in [n]$, we have $\widetilde{N}(s^i) = (1 \pm i\epsilon)N(s^i)$ for all $s \in S$.

Property 2: For a fixed $i \in [n]$, we have an oracle which returns uniform, independent samples $t \sim T(s^j)$ for every $j \leq i$ and $s \in S$, and runs in $\mathrm{poly}(n, m, 1/\epsilon, \gamma)$ time, for some fixed parameter γ which we will later choose. The oracle is allowed to fail with probability at most $3/4$, in which case it outputs no sample.

We remark that the parameter γ will later be set to $\log(1/\delta) + n$, where δ is the failure probability. Fix an arbitrary $i \in [n]$, and suppose we have computed $\widetilde{N}(q^j)$ and $\widetilde{T}_k(q^j)$ for all $q \in S$, $j < i$ and $k \in [i]$. Fix now a state s. We first show how to compute the estimate $\widetilde{N}(s^i)$.

Proposition 12.5.1. *Fix $\delta \in (0,1)$. If Property 1 and 2 hold for all levels $j < i$, then with probability $1 - \delta$ and time $\mathrm{poly}(n, m, 1/\epsilon, \log(1/\delta))$ we can compute a value $\widetilde{N}(s^i)$ such that $\widetilde{N}(s^i) = (1 \pm i\epsilon)N(s^i)$. In other words, Property 1 holds for level i.*

Proof. If $i = 1$, we can compute $N(s^i)$ exactly in time $O(m)$, and we make $\widetilde{N}(s^i) = N(s^i)$. Thus, assume that $i \geq 2$. For each transition $\tau = (s^i, a, q^j \cdot r^{i-j-1}) \in \overline{\Delta}$, recall the definition of the extension $\mathcal{T}[\tau]$ (see Section 12.2), which recognizes trees where runs are forced to start with transition τ. We now define $N(\tau) = |\mathcal{L}(\mathcal{T}[\tau])|$, and observe that $N(\tau) = |T(q^j) \times T(r^{i-j-1})| = N(q^j) \cdot N(r^{i-j-1})$. Thus, we obtain an estimate $\widetilde{N}(\tau)$ of $N(\tau)$ via:

$$\begin{aligned}
\widetilde{N}(\tau) &= \widetilde{N}(q^j) \cdot \widetilde{N}(r^{i-j-1}) \\
&= (1 \pm j\epsilon)(1 \pm (i-j-1)\epsilon) \cdot N(q^j) \cdot N(r^{i-j-1}) \\
&= (1 \pm (j\epsilon + (i-j-1)\epsilon) + j(i-j-1)\epsilon^2) \cdot N(\tau) \\
&= (1 \pm ((i-1)\epsilon) + j(i-j-1)\epsilon^2) \cdot N(\tau) \\
&= \left(1 \pm \left(i - 1 + \frac{1}{n}\right)\epsilon\right) \cdot N(\tau)
\end{aligned}$$

Where in the last equation, we used our assumption that $\epsilon < 1/(4nm)^{18} < 1/n^3$ and then applied the fact that $j(i-j-1)\epsilon^2 \leq n^2\epsilon^2 \leq \epsilon/n$. Also, notice that we are using the fact that Property 1 holds for all sizes $j < i$. Now let $\tau_1, \tau_2, \ldots, \tau_\ell \in \overline{\Delta}$ be all the transitions of the form $\tau_j = (s^i, a_j, q_j \cdot r_j)$ with $q_j, r_j \in \overline{S}$ and $a_j \in \Sigma$. Observe that $N(s^i) = |\bigcup_{j=1}^{\ell} \mathcal{L}(\mathcal{T}[\tau_j])|$. Now for each $j \in [\ell]$, let p_j be the probability that a uniform sample $t \sim \mathcal{L}(\mathcal{T}[\tau_j])$ is not contained in $\mathcal{L}(\mathcal{T}[\tau_{j'}])$ for all $j' < j$. Then $N(s^i) = \sum_{j=1}^{\ell} N(\tau_j)p_j$, so in order to estimate $N(s^i)$ it suffices to estimate the values p_j. Since Property 2 holds for all levels less than i, by making calls to oracles $t_q \sim \mathcal{L}(\mathcal{T}[q_j])$ and $t_r \sim \mathcal{L}(\mathcal{T}[r_j])$ we can obtain an i.i.d. sample $a_j(t_q, t_r)$ from $\mathcal{L}(\mathcal{T}[\tau_j])$ (recall the notation for trees introduced in Section 12.2). By repeating this process, we can obtain

i.i.d. samples $t_1, t_2, \ldots, t_h \sim \mathcal{L}(\mathcal{T}[\tau_j])$ uniformly at random, where $h = O(\log(4m/\delta)m^2/\epsilon^2)$. Now let $\tilde{p}_j$ be the fraction of the samples t_k such that $t_k \notin \mathcal{L}(\mathcal{T}[\tau_{j'}])$ for each $j' < j$. Note that checking if $t_k \notin \mathcal{L}(\mathcal{T}[\tau_{j'}])$ can be done in $\mathrm{poly}(n, m)$ time via a membership query for tree automata. Thus if we let

$$X_k = \begin{cases} 1 & \text{if } t_k \notin \mathcal{L}(\mathcal{T}[\tau_{j'}]) \text{ for each } j' < j \\ 0 & \text{otherwise.} \end{cases}$$

then we have $\tilde{p}_j = h^{-1} \sum_{k=1}^{h} X_k$. Then setting $p_j = \mathbf{E}[X_k]$, by Hoeffding's inequality we have $|\tilde{p}_j - p_j| \leq \frac{\epsilon}{4m}$ with probability at least $1 - \delta/(2m)$,

so we can union bound over all $j \in [\ell]$ and obtain $|\tilde{p}_j - p_j| \leq \frac{\epsilon}{4m}$ for all $j \in [\ell]$ with probability at least $1 - \delta$. Putting all together, we can derive an estimate $\widetilde{N}(s^i)$ for $N(s^i)$ by using the estimates $\widetilde{N}(\tau_j)$ and $\tilde{p}_j$ of $N(\tau_j)$ and p_j, respectively, as follows:

$$\begin{aligned}
\widetilde{N}(s^i) &= \sum_{j=1}^{\ell} \widetilde{N}(\tau_j)\tilde{p}_j \\
&= \left(1 \pm \left(i - 1 + \frac{1}{n}\right)\epsilon\right) \sum_{j=1}^{\ell} N(\tau_j)\tilde{p}_j \\
&= \left(1 \pm \left(i - 1 + \frac{1}{n}\right)\epsilon\right) \left(\sum_{j=1}^{\ell} N(\tau_j)p_j \pm \frac{\epsilon}{4m} \sum_{j=1}^{\ell} N(\tau_j)\right) \\
&= \left(1 \pm \left(i - 1 + \frac{1}{n}\right)\epsilon\right) \left(N(s^i)\left(1 \pm \frac{\epsilon}{4}\right)\right) \\
&= (1 \pm i\epsilon)\, N(s^i).
\end{aligned}$$

Where we use that $\sum_{j=1}^{\ell} N(\tau_j) \leq \sum_{j=1}^{\ell} N(s^i) = \ell N(s^i) \leq mN(s^i)$ in the second to last step, and the fact that $n \geq i \geq 2$ in the last step. For runtime, notice that the key result $\mathbf{Pr}[|\tilde{p}_j - p_j| \leq \epsilon/(4m)] \geq 1 - \delta/(2m)$ is conditioned on the event that we were able to obtain h samples t_k using the oracle. Recall that the sampling oracle can fail with probability at most $3/4$. Then, the required number of calls h' to the poly-time sampling oracle is at most $4h/3$ in expectation. For our purposes, $h' = O(h)$ will also be enough, as we now show. For $j \in [\ell]$ call G_j the event that we obtain h samples from $\mathcal{L}(\mathcal{T}[\tau_j])$ and H_j the event that $|\tilde{p}_j - p_j| \leq \epsilon/(4m)$.

Then, as we showed above:

$$\begin{aligned}\mathbf{Pr}\left[\widetilde{N}(s^i) = (1 \pm i\epsilon)N(s^i)\right] &\geq \mathbf{Pr}\left[\bigcap_{j=1}^{\ell}(H_j \cap G_j)\right] \\ &= 1 - \mathbf{Pr}\left[\bigcup_{j=1}^{\ell}(\overline{H_j} \cup \overline{G_j})\right] \\ &\geq 1 - m\,\mathbf{Pr}\left[\overline{H_{j_0}} \cup \overline{G_{j_0}}\right]\end{aligned}$$

where the last inequality is due to a union bound obtained considering $j_0 = \operatorname{argmax}_{j\in[\ell]} \mathbf{Pr}\left[\overline{H_j} \cup \overline{G_j}\right]$. Recall that we want $\mathbf{Pr}\left[\widetilde{N}(s^i) = (1 \pm i\epsilon)N(s^i)\right] \geq 1 - \delta$, hence it suffices to show

$$1 - m\,\mathbf{Pr}\left[\overline{H_{j_0}} \cup \overline{G_{j_0}}\right] \geq 1 - \delta \iff \frac{\delta}{m} \geq \mathbf{Pr}\left[\overline{H_{j_0}} \cup \overline{G_{j_0}}\right] \iff \mathbf{Pr}\left[H_{j_0} \cap G_{j_0}\right] \geq 1 - \frac{\delta}{m} \tag{12.8}$$

By Hoeffding's inequality, as we showed before, $\mathbf{Pr}\left[H_{j_0} \mid G_{j_0}\right] \geq 1 - \delta/(2m)$. Suppose that we also have that $\mathbf{Pr}\left[G_{j_0}\right] \geq 1 - \delta/(2m)$. Then,

$$\mathbf{Pr}\left[H_{j_0} \cap G_{j_0}\right] = \mathbf{Pr}\left[H_{j_0} \mid G_{j_0}\right] \cdot \mathbf{Pr}\left[G_{j_0}\right] \geq \left(1 - \frac{\delta}{2m}\right)^2 \geq 1 - 2 \cdot \frac{\delta}{2m} = 1 - \frac{\delta}{m}$$

as required by equation (12.8). Thus, it suffices to show $\mathbf{Pr}\left[G_{j_0}\right] \geq 1 - \delta/(2m)$. Letting X_i be the random variable that indicates whether the i-th call to the sampling procedure was successful, then the total number of samples obtained is $X = \sum_{i=1}^{h'} X_i$, where $\mathbf{E}\left[X\right] \geq h'/4$, so by a Chernoff bound we have

$$\mathbf{Pr}\left[G_{j_0}\right] = 1 - \mathbf{Pr}\left[\overline{G_{j_0}}\right] = 1 - \mathbf{Pr}\left[X < h\right] \geq 1 - \exp\left(-\frac{h'}{8}\left(1 - \frac{4h}{h'}\right)^2\right) \geq 1 - \exp\left(-\frac{h'}{8}\left(1 - \frac{4h}{h'}\right)\right)$$

assuming that $4h < h'$. Hence,

$$1 - \exp\left(-\frac{h'}{8}\left(1 - \frac{4h}{h'}\right)\right) \geq 1 - \frac{\delta}{2m} \iff \frac{\delta}{2m} \geq \exp\left(-\frac{h'}{8}\left(1 - \frac{4h}{h'}\right)\right) \iff h' \geq 4h + 8\ln\left(\frac{2m}{\delta}\right).$$

so by definition of h, it is sufficient to set $h' = 5h$, which completes the proof. □

The notion of a partial tree. We need to demonstrate how to obtain uniform samples from $T(s^i)$ to build the sets $\widetilde{T}_j(s^i)$. To do this, we will provide an algorithm that recursively samples a tree $t \in T(s^i)$ from the top down. But before showing this procedure, we need to introduce the

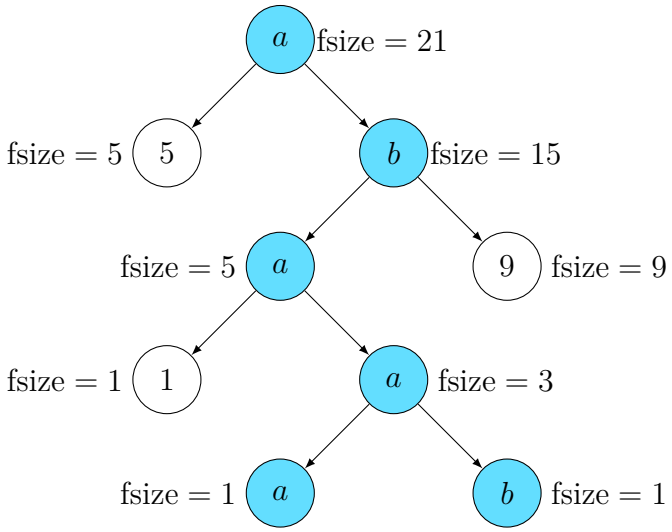


Figure 12.4: An example of a partial tree. White nodes corresponds to holes, which are labeled by integers. Nodes that are not holes are labeled with symbols from $\Sigma = \{a, b\}$.

notion of a *partial tree*. In the following, recall that Σ is a finite alphabet and assume, without loss of generality, that $\Sigma \cap [n] = \emptyset$.

Definition 12.5.2. *A partial tree is a binary labeled tree t over $\Sigma \cup [n]$. A node u labeled by $t(u) \in [n]$ is called a hole of t, and we assume that holes can appear only at the leaves of t. The full size of t, denoted by* $\mathrm{fsize}(t)$*, is defined as* $|\{u \mid t(u) \in \Sigma\}| + \sum_{u\,:\,t(u)\in[n]} t(u)$*. Moreover, a partial tree t is said to be complete if t contains no holes.*

Intuitively, in a partial tree t, a hole u represents a placeholder where a subtree of size $t(u)$ is going to be hanged. That is, partial tree t is representing all trees over Σ that have the same trunk as t and, for each hole u, the subtree rooted at u is of size $t(u)$. Notice that all trees represented by t will have the same size $|\{u \mid t(u) \in \Sigma\}| + \sum_{u\,:\,t(u)\in[n]} t(u)$ and, therefore, we define the full size of t as this quantity. Finally, observe that if a partial tree t is complete, then t contains no holes and, hence, no extension is needed. For an example of a partial tree, see Figure 12.4.

For every partial tree t and node $x \in t$, write t_x to denote the partial subtree of t rooted at x. For each hole $u \in t$ with size $t(u) = i$, we say that t' is an *immediate extension* of t over u if $t' = t[u \to a(j, i-j-1)]$ for some $a \in \Sigma$ and $j \in [i-2]$. That is, t is extended by replacing the label of u with a and hanging from u two new holes whose sizes sum to $i-1$ (note that the resulting partial subtree t'_u has full size i). In case that $i = 1$, then it must hold that $t' = t[u \to a]$ for some $a \in \Sigma$. We define the set of all immediate extensions of t over u as $\mathrm{ext}(t, u)$. Note that $|\mathrm{ext}(t, u)| = (i-2)|\Sigma|$. Finally, given two partial trees t and t', we write $t \hookrightarrow_u t'$ if t' is

an immediate extension of t over u, and $t \hookrightarrow t'$ if t' is an immediate extension of t over some hole $u \in t$. We then define the reflexive and transitive closure $\hookrightarrow^*$ of $\hookrightarrow$, and say that t' is an *extension* of t if $t \hookrightarrow^* t'$. In other words, $t \hookrightarrow^* t'$ if either $t' = t$ or t' can be obtained from t via a non-empty sequence of immediate extensions $t \hookrightarrow t_1 \hookrightarrow t_2 \hookrightarrow \cdots \hookrightarrow t'$. We say that t' is a *completion* of t when $t \hookrightarrow^* t'$ and t' is complete.

Obtaining uniform samples from $T(s^i)$. Given a partial tree t with $\mathrm{fsize}(t) = i$, consider now the set $T(s^i, t)$ of all completions t' of t derivable with s^i as the state in the root node, namely, $T(s^i, t) = \{t' \in T(s^i) \mid t' \text{ is a completion of } t\}$. Further, define $N(s^i, t) = |T(s^i, t)|$. To obtain a uniform sample from $T(s^i)$, we start with a partial tree $t = i$ (i.e. t is a partial tree with one node, which is a hole of size i). At each step, we choose the hole $u \in t$ with the smallest size $t(u)$, and consider an immediate extension of t over u. Note that the set $T(s^i, t)$ can be partitioned by the sets $\{T(s^i, t')\}_{t \hookrightarrow_u t'}$ of such immediate extensions. The fact that $T(s^i, t') \cap T(s^i, t'') = \emptyset$, whenever $t \hookrightarrow_u t'$, $t \hookrightarrow_u t''$ and $t' \neq t''$, follows immediately from the fact that t' and t'' have different labels from Σ in the place of u or unequal sizes of the left and right subtrees of u. We then will sample each partition $T(s^i, t')$ with probability approximately proportional to its size $N(s^i, t')$, set $t' \leftarrow t$, and continue like that recursively. Formally, the procedure to sample a tree in $T(s^i)$ is shown in Algorithm 15.

Algorithm 15: SAMPLE$\Big(s^i,\ \{\widetilde{T}_i(r^j)\}_{r\in S, j<i},\ \{\widetilde{N}(r^j)\}_{r\in S, j\le i},\ \epsilon,\ \delta\Big)$

Initialize a partial tree $t = i$, and set $\varphi = 1$
while *t is not complete* **do**
  Let u be the hole of t with the minimum size $t(u)$. If more than one node reaches this minimum value, choose the first such a node according to a prespecified order on the holes of t.
  Let $\mathrm{ext}(t, u) = \{t_1, \ldots, t_\ell\}$ be the set of immediate extensions of t over u.
  For each $k \in [\ell]$, call ESTIMATEPARTITION$(t_k,\ s^i,\ \{\widetilde{T}_i(r^j)\}_{r\in S, j<i},\ \{\widetilde{N}(r^j)\}_{r\in S, j\le i},\ \epsilon,\ \delta)$ to obtain an estimate $\widetilde{N}(s^i, t_k)$ of $N(s^i, t_k)$. `// Recall that` $\mathcal{T} = (S, \Sigma, \Delta, s_{\mathtt{init}})$
  Sample partition $k \in [\ell]$ with probability $\frac{\widetilde{N}(s^i, t_k)}{\sum_{k'=1}^{\ell} \widetilde{N}(s^i, t_{k'})}$.
  Set $\varphi \leftarrow \varphi \cdot \frac{\widetilde{N}(s^i, t_k)}{\sum_{k'=1}^{\ell} \widetilde{N}(s^i, t_{k'})}$.
  Set $t \leftarrow t_k$.
return *t with probability $\frac{1}{2\varphi\widetilde{N}(s^i)}$, otherwise output* **FAIL**.

Notice that SAMPLE$\Big(s^i,\ \{\widetilde{T}_i(r^j)\}_{r\in S, j<i},\ \{\widetilde{N}(r^j)\}_{r\in S, j\le i},\ \epsilon,\ \delta\Big)$ uses the precomputed values

$\widetilde{T}_i(r^j)$ for every $r \in S$ and $j \in [i-1]$, and the precomputed values $\widetilde{N}(r^j)$ for every $r \in S$ and $j \in [i]$. This procedure first selects a hole u with the minimum size $t(u)$, and then calls a procedure ESTIMATEPARTITION to obtain an estimate $\widetilde{N}(s^i, t_k)$ of $N(s^i, t_k)$ for every immediate extensions t_k of t over u. Thus, to prove our main theorem about the procedure SAMPLE, we first need the following lemma about the correctness of the partition size estimates. The proof of Lemma 12.5.3 is the main focus of Section 12.6.

Lemma 12.5.3. *Let $\delta \in (0, 1/2)$, and fix independent and uniform samples sets $\widetilde{T}_i(s^j)$ of $T(s^j)$ each of size $O(\log^2(\delta^{-1})(nm)^{13}/\epsilon^5)$, for every $s \in S$ and $j < i$. Suppose further that we have values $\widetilde{N}(s^j) = (1 \pm j\epsilon)N(s^j)$ for every $s \in S$ and $j \leq i$ Then with probability $1 - \delta^{nm}$, the following holds: for every state $s \in S$ and for every partial tree t with $\mathrm{fsize}(t) = i$, the procedure*

$$\textsc{EstimatePartition}\left(t,\ s^i, \left\{\widetilde{T}_i(r^j)\right\}_{r\in S, j<i}, \left\{\widetilde{N}(r^j)\right\}_{r\in S, j\leq i}, \epsilon,\ \delta\right)$$

runs in $\mathrm{poly}(n, m, \epsilon^{-1}, \log\delta^{-1})$-time and returns a value $\widetilde{N}(s^i, t)$ such that

$$\widetilde{N}(s^i, t) \ = \ \left(1 \pm (4nm)^{17}\epsilon\right) N\left(s^i, t\right)$$

Notice that the guarantee of Lemma 12.5.3 is that ESTIMATEPARTITIONalways runs in polynomial time. Conditioned on the success of Lemma 12.5.3, lines $1-9$ of the algorithm SAMPLEalways run in polynomial time (i.e, with probability 1). Thus, conditioned on Lemma 12.5.3, only on line 10 of SAMPLEis it possible for the algorithm to output **FAIL**. In the following Lemma, we demonstrate that, given the success of Lemma 12.5.3, the SAMPLE algorithm produces *truly* uniform samples, and moreover the probability of outputting **FAIL** is at most a constant. Notice that this implies that, after running the inner loop of SAMPLE a total of $O(\log\delta^{-1})$ times, we will obtain a sample with probability at least $1 - \delta$.

Lemma 12.5.4. *Given $\delta \in (0, 1/2)$, $\{\widetilde{T}_i(r^j)\}_{r\in S, j<i}$ and $\{\widetilde{N}(r^j)\}_{r\in S, j\leq i}$, suppose that the procedure*

$$\textsc{EstimatePartition}\left(t,\ s^i,\ \left\{\widetilde{T}_i(r^j)\right\}_{r\in S, j<i}, \left\{\widetilde{N}(r^j)\right\}_{r\in S, j\leq i}, \epsilon,\ \delta\right)$$

produces an estimate $\widetilde{N}(s^i, t)$ with $\widetilde{N}(s^i, t) = (1 \pm (4nm)^{17}\epsilon)N(s^i, t)$ for every partial tree t of size i and state s^i.

Further suppose that Property 1 holds for all $j \leq i$ (see page 621), and $n \geq 2$. Then

conditioned on not outputting **FAIL**, *each call to the procedure*

$$\textsc{Sample}\left(s^i,\ \left\{\widetilde{T}_i(r^j)\right\}_{r\in S, j<i},\ \left\{\widetilde{N}(r^j)\right\}_{r\in S, j\le i},\ \epsilon,\ \delta\right)$$

produces an independent, uniform sample $t \sim T(s^i)$. *Moreover, the probability that a given call outputs* **FAIL** *is at most* $3/4$, *and the number of times* EstimatePartition *is called in each iteration of the loop is at most* nm.

Proof. Fix a tree $t \in T(s^i)$. Then there is a unique sequence of partial trees $i = t_0, t_1, t_2, \ldots, t_i = t$ such that $T(s^i) = T(s^i, t_0) \supseteq T(s^i, t_1) \supseteq T(s^i, t_2) \supseteq \cdots \supseteq T(s^i, t_i) = \{t\}$, which gives a sequence of nested partitions which could have been considered in the call $\textsc{Sample}(s^i,\ \{\widetilde{T}_i(r^j)\}_{r\in S, j<i},\ \{\widetilde{N}(r^j)\}_{r\in S, j\le i},\ \epsilon,\ \delta)$. For $j \in [i]$, let p_j be the true ratio of $\frac{N(s^i,t_j)}{N(s^i,t_{j-1})}$, which is the probability that we should have chosen partition $T(s^i, t_j)$ conditioned on being in partition $T(s^i, t_{j-1})$. Note that $\prod_{j=1}^{i} p_j = \frac{1}{|T(s^i)|} = \frac{1}{N(s^i)}$. Now assuming EstimatePartition always returns an estimate with at most $(1 \pm (4nm)^{17}\epsilon)$-relative error, it follows that conditioned on being in partition $T(s^i, t_{j-1})$, we chose the partition $T(s^i, t_j)$ with probability $\tilde{p}_j = (1 \pm (4nm)^{17}\epsilon)p_j$. Thus the probability that we choose t at the end of the loop in step 2 of the Sample procedure is:

$$\varphi = \prod_{j=1}^{i} \tilde{p}_j = (1 \pm (4nm)^{17}\epsilon)^i \prod_{j=1}^{i} p_j = (1 \pm 1/n)\prod_{j=1}^{i} p_j = (1 \pm 1/n)\frac{1}{N(s^i)} = (1 \pm 1/n)(1 \pm i\epsilon)\frac{1}{\widetilde{N}(s^i)} = (1 \pm 1/n)(1 \pm 1/(4n))\frac{1}{\widetilde{N}(s^i)} = (1 \pm 2/n)\frac{1}{\widetilde{N}(s^i)}.$$

Notice that we use the fact that $\epsilon < (4nm)^{-18}$ and that Property 1 holds for all $j \le i$. The probability that we do not output **FAIL** can be bounded by

$$\frac{1}{2\varphi\widetilde{N}(s^i)} = \frac{1}{2\widetilde{N}(s^i)}\prod_{j=1}^{i}\frac{1}{\tilde{p}_j} \ge \frac{1}{2\widetilde{N}(s^i)}\frac{\widetilde{N}(s^i)}{(1+2/n)} \ge \frac{1}{2(1+2/n)} \ge 1/4$$

since $n \ge 2$, which completes the proof that the probability that the call $\textsc{Sample}(s^i,\ \{\widetilde{T}_i(r^j)\}_{r\in S, j<i},\ \{\widetilde{N}(r^j)\}_{r\in S, j\le i},\ \epsilon,\ \delta)$ outputs **FAIL** is at most $3/4$. For the uniformity claim, note that we accept t at the end with probability $\varphi \cdot \frac{1}{2\varphi\widetilde{N}(s^i)} = \frac{1}{2\widetilde{N}(s^i)}$, which is indeed uniform conditioned on not outputting **FAIL**, as it does not depend on t. Finally, notice that EstimatePartition is called at most $(t(u) - 2) \cdot |\Sigma| \le nm$ times in each iteration of the loop. □

An FPRAS for #BTA **and** #TA**.** We show in Algorithm 16 a fully polynomial-time approximation schema for #BTA, which puts together the different components mentioned in this section.

The correctness of this algorithm is shown in the following theorem.

Algorithm 16: FPRASBTA$(\mathcal{T}, 0^n, \epsilon, \delta)$

Set $m \leftarrow |\mathcal{T}|$
if $n < 2$ *or* $m < 3$ **then**
  Edge case, $|\mathcal{L}_n(\mathcal{T})|$ can be exactly computed (Remark 145)
Construct the tree automaton $\overline{\mathcal{T}}$
Set $\epsilon \leftarrow \min\{\epsilon, 1/(4mn)^{18} - 1\}$
Set $\gamma = \log(1/\delta) + 2n$
Set $\alpha \leftarrow O(\log^2(1/\delta)(nm)^{13}/\epsilon^5)$, *total* $\leftarrow O(\alpha)$
For each $s \in S$, compute $N(s^1)$ exactly and set $\widetilde{N}(s^1) \leftarrow N(s^1)$
For each $s \in S$, create set $\widetilde{T}_2(s^1)$ with α uniform, independent samples from $T(s^1)$
for $i = 2, \ldots, n$ **do**
  For each $s \in S$, compute $\widetilde{N}(s^i)$ such that
  $\mathbf{Pr}\left[\widetilde{N}(s^i) = (1 \pm i\epsilon)N(s^i)\right] \geq 1 - \exp(-\gamma n^{20})$
  if $i < n$ **then**
    for *each* $s \in S$ *and* $j = 1, \ldots, i$ **do**
      Set $\widetilde{T}_{i+1}(s^j) \leftarrow \emptyset$, *counter* $\leftarrow 1$
      while $|\widetilde{T}_{i+1}(s^j)| < \alpha$ *and counter* $\leq$ *total* **do**
        Call the procedure
        $\textsc{Sample}\Big(s^j, \{\widetilde{T}_i(r^k)\}_{r \in S, k<j}, \{\widetilde{N}(r^k)\}_{r \in S, k \leq j}, \epsilon, 2^{-2n}\delta\Big)$
        If this procedure returns a tree t, then set $\widetilde{T}_{i+1}(s^j) \leftarrow \widetilde{T}_{i+1}(s^j) \cup \{t\}$
        Set *counter* $\leftarrow$ *counter* $+ 1$
      if $|\widetilde{T}_{i+1}(s^j)| < \alpha$ **then**
        return FAIL
return $\widetilde{N}(s^n_{init})$.

Theorem 147. *Let* $\epsilon, \delta \in (0, 1/2)$, $n \geq 1$, $\mathcal{T} = (S, \Sigma, \Delta, s_{init})$ *be a tree automaton, and* $m = \|\Delta\|$ *be the size of* $\mathcal{T}$. *Then the call* FPRASBTA$(\mathcal{T}, 0^n, \epsilon, \delta)$[8] *returns, with probability at least*

[8]Here we write 0^n as the unary representation of n. Since our algorithms are polynomial in n, the algorithm is polynomial in the size of the input.

$1-\delta$, *a value* $\widetilde{N}$ *such that* $\widetilde{N} = (1 \pm \epsilon)|\mathcal{L}_n(\mathcal{T})|$. *Moreover, the runtime of the algorithm* FPRASBTA *is* $\mathrm{poly}(n, m, 1/\epsilon, \log(1/\delta))$.

Proof. Set $\alpha = O(\log^2(1/\delta)(nm)^{13}/\epsilon^5)$. For every $j \in [n]$, let $\mathcal{E}_j^1$ denote the event that Property 1 holds for level j, and similarly define $\mathcal{E}_j^2$ for Property 2. Set $\gamma = \log(1/\delta)$. We prove inductively that

$$\mathbf{Pr}\left[\bigwedge_{j\le i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] \;\ge\; 1 - 2^{-\gamma+2i}$$

for each $i \in [n]$. Since $N(s^1)$ is computed exactly in step 8 of FPRASBTA$(\mathcal{T}, 0^n, \epsilon, \delta)$ and the size of each tree in $T(s^1)$ is 1, the base case $i = 1$ trivially holds. Now at an arbitrary step $i \ge 2$, suppose $\mathcal{E}_j^1 \wedge \mathcal{E}_j^2$ holds for all $j < i$. By considering $\exp(-\gamma n^{20})$ as the value for the parameter δ in Proposition 12.5.1, it follows that $\mathcal{E}_i^1$ holds with probability at least $1 - \exp(-\gamma n^{20})$, and the runtime to obtain Property 1 is $\mathrm{poly}(n, m, \frac{1}{\epsilon}, \gamma)$. Thus,

$$\mathbf{Pr}\left[\mathcal{E}_i^1 \;\middle|\; \bigwedge_{j<i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] \;\ge\; 1 - \exp(-\gamma n^{20}).$$

We now must show that $\mathcal{E}_i^2$ holds – namely, that we can obtain uniform samples from all sets $T(s^i)$. By Lemma 12.5.3, if we can obtain fresh uniform sample sets $\widetilde{T}_i(s^j)$ of $T(s^j)$ for each $s \in S$ and $j < i$, each of size α, then with probability at least $1-2^{-\gamma nm}$, we have that for every partial tree t' of size i (that is, $\mathrm{fsize}(t') = i$) and state $s \in S$, the procedure ESTIMATEPARTITION$(t',\ s^i,\ \{\widetilde{T}_i(s^j)\}_{s\in S, j<i},\ \{\widetilde{N}(s^j)\}_{s\in S, j\le i},\ \epsilon,\ \delta)$ produces an estimate $\widetilde{N}(s^i, t')$ such that $\widetilde{N}(s^i, t') = (1 \pm (4nm)^{17}\epsilon)N(s^i, t')$. Since we have to call ESTIMATEPARTITION at most inm times (see Lemma 12.5.4), after a union bound we get that the conditions of Theorem 12.5.4 are satisfied with probability at least $1 - 2^{-\gamma}$, and it follows that we can sample uniformly from the set $T(s^i)$ for each $s \in S$ in polynomial time.

It remains to show that we can obtain these fresh sample sets $\widetilde{T}_i(s^j)$ of $T(s^j)$ for each $s \in S$ and $j < i$ in order to condition on the above. But the event $\mathcal{E}_{i-1}^2$ states precisely that can indeed obtain such samples in $\mathrm{poly}(n, m, \frac{1}{\epsilon}, \gamma)$ time per sample. Thus the conditions of the above paragraph are satisfied, so we have

$$\mathbf{Pr}\left[\mathcal{E}_i^2 \;\middle|\; \mathcal{E}_i^1 \wedge \bigwedge_{j<i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] \;\ge\; 1 - 2^{-\gamma}.$$

Therefore, we conclude that

$$\mathbf{Pr}\left[\mathcal{E}_i^1 \wedge \mathcal{E}_i^2 \;\middle|\; \bigwedge_{j<i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] \;\geq\; 1 - 2^{-\gamma} - \exp(-\gamma n^{20}) + 2^{-\gamma}\exp(-\gamma n^{20}) \;\geq\; 1 - 2^{-\gamma+1}$$

Hence, by induction hypothesis:

$$\begin{aligned}
\mathbf{Pr}\left[\bigwedge_{j\leq i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] &= \mathbf{Pr}\left[\mathcal{E}_i^1 \wedge \mathcal{E}_i^2 \;\middle|\; \bigwedge_{j<i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] \cdot \mathbf{Pr}\left[\bigwedge_{j<i}\left(\mathcal{E}_j^1 \wedge \mathcal{E}_j^2\right)\right] \\
&\geq (1 - 2^{-\gamma+1})(1 - 2^{-\gamma+2(i-1)}) \\
&= 1 - 2^{-\gamma+1} - 2^{-\gamma+2i-2} + 2^{-2\gamma+2i-1} \\
&\geq 1 - 2^{-\gamma+1} - 2^{-\gamma+2i-1} \\
&\geq 1 - 2^{-\gamma+2i-1} - 2^{-\gamma+2i-1} \\
&= 1 - 2^{-\gamma+2i}
\end{aligned}$$

which completes the inductive proof. Redefining $\gamma = \log(1/\delta) + 2n$ (see Line 6 of Algorithm 16) and considering $2^{-2n}\delta$ when using Lemma 12.5.3 (see Line 16 of Algorithm 16), we obtain that the success probability of the overall algorithm is $1 - \delta$ as needed.

For runtime, note that by Lemma 12.5.4, the expected number of trials to obtain α samples $\widetilde{T}_i(s^j)$ for each $s^j \in \overline{S}$ and $i \in [n]$ is $O(\alpha)$, and thus is $O(\alpha)$ with probability $1-2^{-\alpha} > 1-2^{-mn\gamma}$ by Chernoff bounds. That is, with $O(\alpha)$ trials, we have probability at least $1-2^{-mn\gamma}$ of not failing in step 20 of Algorithm 16. Since we go through that step at most $O(n^2 m)$ times during the whole run of the algorithm, that means that the overall probability of returning **FAIL** can be bounded by $1 - 2^{\gamma} = 1 - \delta$, which is a loose bound but enough for our purposes. Moreover, by Lemma 12.5.4, the runtime of each sampling trial in step 16 of Algorithm 16 is polynomial in n, m, $1/\epsilon$ and $\log(1/(2^{-2n}\delta)) = \gamma$. It follows that the entire algorithm runs in $\mathrm{poly}(n, m, 1/\epsilon, \log(1/\delta))$ time, which completes the proof. □

We now provide our main theorem for uniformly sampling from tree automata. The notion of sampling we get is in fact stronger than the definition of a FPAUS as defined in Section 12.2. Specif

Theorem 148. *Let $\delta \in (0, 1/2)$, $n \geq 1$, $\mathcal{T} = (S, \Sigma, \Delta, s_{init})$ be a tree automaton, and $m = \|\Delta\|$ be the size of $\mathcal{T}$. Then there is a sampling algorithm $\mathcal{A}$ and a pre-processing step with the following property. The preprocessing step runs in $\mathrm{poly}(n, m, \log \delta^{-1})$ time, and with probability*

$1-\delta$ *over the randomness used in this pre-processing step,*[9] *each subsequent call to the algorithm* $\mathcal{A}$ *runs in time* $\mathrm{poly}(n, m, \log \delta^{-1})$ *time, and returns either a uniform sample* $t \sim \mathcal{L}_n(\mathcal{T})$ *or* **FAIL**. *Moreover, if* $\mathcal{L}_n(\mathcal{T}) \neq \emptyset$, *the probability that the sampler returns* **FAIL** *is at most* $1/2$.

Additionally, this implies that there is an FPAUS for $\mathcal{L}_n(\mathcal{T})$ *as defined in Section 12.2.*

Proof. The preprocessing step here is just the computation of the estimates $\widetilde{N}(s^i)$ and sketches $\widetilde{T}(s^i)$ for all $i \leq n$, which are obtained by a single to the FPRAS of Theorem 147 using a fixed $\epsilon = (nm)^{-C}$ for a sufficiently large constant C. The sampling algorithm is then just a call to

$$\textsc{Sample}\left(s^n_{\text{init}}, \left\{\widetilde{T}_i(r^j)\right\}_{r\in S, j<n}, \left\{\widetilde{N}(r^j)\right\}_{r\in S, j\leq n}, \epsilon, \delta\right)$$

Then the first result follows from Lemmas 12.5.4 and 12.5.3, as well as Theorem 147. In particular, if we condition on the success of Theorem 147, which hold with probability $1-\delta$, then by the definition of Property 2, and the fact that Property 2 holds for the size n conditioned on Theorem 147, this is sufficient to guarantee that the samples produced by our sampling procedure are uniform. Furthermore, by Lemma 12.5.4, the probability that a call to SAMPLE outputs **FAIL** is at most $3/4$, so repeating the algorithm three times, the probability that a sample is not output is at most $(3/4)^3 < 1/2$. Finally, by Lemma 12.5.3 the runtime of each call to ESTIMATEPARTITION is at most $\mathrm{poly}(n, m, \log \delta^{-1})$, and by Lemma 12.5.4, the procedure ESTIMATEPARTITION is called at most nm times per call to SAMPLE, which completes the proof of the runtime.

We now verify that the above implies that $\mathcal{L}_n(\mathcal{T})$ admits an FPAUS. First, if $\mathcal{L}_n(\mathcal{T}) \neq \emptyset$, note that the probability δ of failure of the pre-processing algorithm induces an additive δ difference in total variational distance from the uniform sampler. Moreover, by testing deterministically whether the tree t obtained by the algorithm is contained in $\mathcal{L}_n(\mathcal{T})$, we can ensure that, conditioned on not outputting **FAIL**, the output of the algorithm is supported on $\mathcal{L}_n(\mathcal{T})$. We can then run the algorithm with $\delta_0 = \delta|\mathcal{L}_n(\mathcal{T})|^{-1} = \delta \exp(-\mathrm{poly}(n, m))$, which does not affect the stated polynomial runtime. This results in

$$\begin{aligned}\mathcal{D}(t) &= \frac{1}{|\mathcal{L}_n(\mathcal{T})|} \pm \delta_0 \\ &= \frac{1}{|\mathcal{L}_n(\mathcal{T})|} \pm \delta|\mathcal{L}_n(\mathcal{T})|^{-1} \\ &= \frac{(1\pm\delta)}{|\mathcal{L}_n(\mathcal{T})|}\end{aligned} \tag{12.9}$$

[9]Note that we cannot detect if the event within the preprocessing step that we condition on here fails, which occurs with probability δ.

for every $t \in \mathcal{L}_n(\mathcal{T})$, as desired. To deal with the $1/2$ probability that the output of our sampler is **FAIL**, we can run the sampler for a total of $\Theta(\log(\delta^{-1}|\mathcal{L}_n(\mathcal{T})|)) = \mathrm{poly}(n, m, \log \delta^{-1})$ trials, and return the first sample obtained from an instance that did not return **FAIL**. If all trails output **FAIL**, then we can also output $\perp$ as per the specification of an FPAUS. By doing so, this causes another additive $\delta|\mathcal{L}_n(\mathcal{T})|^{-1}$ error in the sampler, which is dealt with in the same way as shown in Equation (12.9) above. Finally, if $\mathcal{L}_n(\mathcal{T}) = \emptyset$, the algorithm must always output $\perp$, since given any potential output $t \neq \perp$, we can always test if $t \in \mathcal{L}_n(\mathcal{T})$ in polynomial time, which completes the proof that the algorithm yields an FPAUS.

□

We conclude this section by pointing out that from Theorem 147 and the existence of a polynomial-time parsimonious reduction from #TA to #BTA, and the fact that given a binary tree $t' \in \mathcal{L}_n(\mathcal{T}')$ after the reduction from a tree automata $\mathcal{T}$ to a binary tree automata $\mathcal{T}'$, the corresponding original tree $t \in \mathcal{L}_n(\mathcal{T})$ can be reconstructed in polynomial time, we obtain the following corollary:

Corollary 12.5.5. *Both* #BTA *and* #TA *admit an FPRAUS and an FPAUS.*

12.6 Estimating Partition Sizes via Succinct NFAs

The goal of this section is to prove Lemma 12.5.3, namely, to show how to implement the procedure EstimatePartition. To this end, we first show how EstimatePartition can be implemented by reducing it to the problem of counting words accepted by a succinct NFA, which we introduced in Section 12.4 and formally define here. Next, we demonstrate an FPRAS for counting words accepted by a succinct NFA, which will complete the proof of Lemma 12.5.3.

Succinct NFAs. Let Γ be a finite set of labels. A *succinct NFA* over Γ is a 5-tuple $\mathcal{N} = (S, \Gamma, \Delta, s_{\text{init}}, s_{\text{final}})$ where S is the set of states and each transition is labeled by a subset of Γ, namely $\Delta \subseteq S \times 2^{\Gamma} \times S$. Thus each transition is of the form (s, A, s'), where $A \subseteq \Gamma$. For each transition $(s, A, s') \in \Delta$, the set $A \subseteq \Gamma$ is given in some representation (e.g. a tree automaton, a DNF formula, or an explicit list of elements), and we write $\|A\|$ to denote the size of the representation. Note that while the whole set A is a valid representation of itself, generally the number of elements of A, denoted by $|A|$, will be exponential in the size of the representation $\|A\|$. We define the size of the succinct NFA $\mathcal{N}$ as $|\mathcal{N}| = |S| + |\Delta| + \sum_{(s,A,s') \in \Delta} \|A\|$. For notational simplicity, we will sometimes write $r = |\mathcal{N}|$.

Given a succinct NFA $\mathcal{N}$ as defined above and elements $w_1, \ldots, w_n \in \Gamma$, we say that $\mathcal{N}$ accepts the word $w_1 w_2 \ldots w_n$ if there exist states $s_0, s_1, \ldots, s_n \in S$ and sets $A_1, \ldots, A_n \subseteq \Gamma$ such that:

- $s_0 = s_{\text{init}}$ and $s_n = s_{\text{final}}$
- $w_i \in A_i$ for all $i = 1 \ldots n$
- $(s_{i-1}, A_i, s_i) \in \Delta$ for all $i = 1 \ldots n$

We denote by $\mathcal{L}_k(\mathcal{N})$ the set of all words of length k accepted by $\mathcal{N}$. We consider the following general counting problem:

Problem:	#SuccinctNFA
Input:	$k \geq 1$ given in unary and a succinct NFA $\mathcal{N}$
Output:	$\lvert\mathcal{L}_k(\mathcal{N})\rvert$

Reduction to Unrolled Succinct NFAs Our algorithm for approximating $|\mathcal{L}_k(\mathcal{N})|$ first involves unrolling k times the NFA $\mathcal{N} = (S, \Gamma, \Delta, s_{\text{init}}, s_{\text{final}})$, to generate an unrolled NFA $\mathcal{N}^k_{unroll}$. Specifically, for every state $p \in S$ create $k-1$ copies $p^1, p^2, \ldots, p^{k-1}$ of p, and include them as states of the unrolled NFA $\mathcal{N}^k_{unroll}$. Moreover, for every transition (p, A, q) in Δ, create the edge $(p^\alpha, A, q^{\alpha+1})$ in $\mathcal{N}^k_{unroll}$, for every $\alpha \in \{1, \ldots, k-2\}$. Finally, if (s_{init}, A, q) is a transition in Δ, then $(s_{\text{init}}, A, q^1)$ is a transition in $\mathcal{N}^k_{unroll}$, while if (p, A, s_{final}) is a transition in Δ, then $(p^{k-1}, A, s_{\text{final}})$ is a transition in $\mathcal{N}^k_{unroll}$. In this way, we keep s_{init} and s_{final} as the initial and final states of $\mathcal{N}^k_{unroll}$, respectively. Since k is given in unary, it is easy to see that $\mathcal{N}^k_{unroll}$ can be constructed in polynomial time from $\mathcal{N}$. Thus, for the remainder of the section, we will assume that the input succinct NFA $\mathcal{N}$ has been unrolled according to the value k. Thus, we consider the following problem.

Problem:	#UnrolledSuccinctNFA
Input:	$k \geq 1$ given in unary and an unrolled succinct NFA $\mathcal{N}^k_{unroll}$
Output:	$\lvert\mathcal{L}_k(\mathcal{N}^k_{unroll})\rvert$

Clearly in the general case, without any assumptions on our representation $\|A\|$ of $|A|$, it will be impossible to obtain polynomial in $|\mathcal{N}|$ time algorithms for the problem above. In order to obtain polynomial time algorithms, we require the following four properties of the label sets A to be satisfied. The properties state that the sizes $|A|$ are at most singly exponential in $|\mathcal{N}|$, we can efficiently test whether an element $a \in \Gamma$ is a member of A, we can obtain approximations of $|A|$, and that we can generate almost uniform samples from A.

Definition 12.6.1 (Required properties for a succinct NFA)**.** *Fix $\epsilon_0 > 0$. Then for every label set A present in Δ, we have:*

1. ***Size bound:*** *There is a polynomial $g(x)$ such that $|A| \leq 2^{g(|\mathcal{N}|)}$.*
2. ***Membership:*** *There is an algorithm that given any $a \in \Gamma$, verifies in time $T = \text{poly}(|\mathcal{N}|)$ whether $a \in A$.*
3. ***Size approximations:*** *We have an estimate $\widetilde{N}(A) = (1 \pm \epsilon_0)|A|$.*
4. ***Almost uniform samples:*** *We have an oracle which returns independent samples $a \sim A$ from a distribution $\mathcal{D}$ over A, such that for every $a \in A$:*

$$\mathbf{D}(a) \quad = \quad (1 \pm \epsilon_0)\frac{1}{|A|}$$

□

The reason for the first condition is that our algorithms will be polynomial in $\log(N)$, where N is an upper bound on the size of $|\mathcal{L}_k(\mathcal{N})|$. We remark that for the purpose of our main algorithm, we actually have truly uniform samples from each set A that is a label in a transition. However, our results may be applicable in other settings where this is not the case. In fact, along with the first two conditions from Definition 12.6.1, a sufficient condition for our algorithm to work is that the representations of each set A allows for an FPRAS and a polynomial time almost uniform sampler.

The Main Path of a Partial Tree Next we show that if we can approximate the number of words of a given length accepted by a succinct NFA, then we can implement the procedure EstimatePartition. But first we need to introduce the notion of main path of a partial tree. Let t be a partial tree constructed via the partitioning procedure of Algorithm 15 (see page 625). Given that we always choose the hole with the minimal size in Line 3, one can order the holes of t as $u_1, u_2, \ldots, u_k$, such that for each i, $\text{parent}(u_i)$ is an ancestor of u_{i+1}, namely, $u_{i+1}, \ldots, u_k$ are contained in the subtree rooted at the parent of u_i. Note that by definition of the loop of Algorithm 15, it could be the case that two holes u and v share the same parent (e.g. the last step produced a subtree of the form $a(i, j)$). If this is the case, we order u and v arbitrarily. Then we define the *main path* π of t considering two cases. If no two holes share the same parent, then π is the path $\text{parent}(u_1), \text{parent}(u_2), \ldots, \text{parent}(u_k)$ (from the most shallow node u_1 to the deepest node u_k). On the other hand, if two holes share the same parent, then by definitions of Algorithm 15 and sequence u_1, …, u_k, these two nodes must be u_{k-1} and u_k. In this case, we define

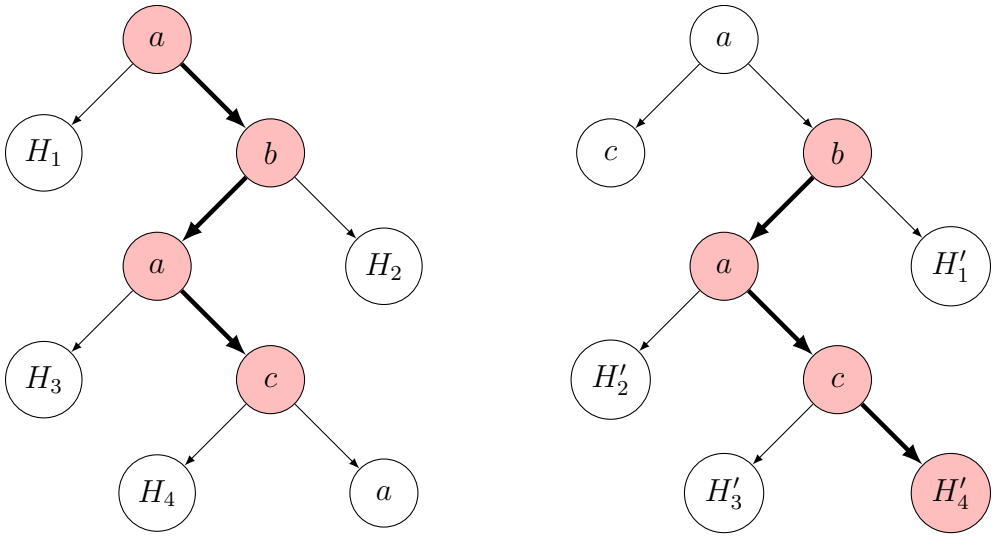


Figure 12.5: Two examples of a partial tree. The holes are indicated by the letters H and H', while the nodes that are not holes have labels $a, b, c \in \Sigma$. Non-white nodes and thick arcs are used to highlight the main paths.

π as the path $\mathrm{parent}(u_1), \mathrm{parent}(u_2), \ldots, \mathrm{parent}(u_{k-1}), u_k$, (again, from the most shallow node u_1 to the deepest node u_k).[10] Observe that by definition of Algorithm 15, every hole is a child of some node in π. We illustrate the notion of main path in Figure 12.5. For the partial tree in the left-hand side, we have that the main path is $\mathrm{parent}(H_1), \mathrm{parent}(H_2), \mathrm{parent}(H_3), \mathrm{parent}(H_4)$ as no two holes share the same parent. On the other hand, the main path for the partial tree in the right-hand side is $\mathrm{parent}(H'_1), \mathrm{parent}(H'_2), \mathrm{parent}(H'_3), H'_4$, as in this case holes H'_3 and H'_4 share the same parent.

Lemma 12.6.2. *There exists a polynomial-time algorithm that, given a tree automaton $\mathcal{T}$, a partial tree t with k holes constructed via the partitioning procedure of Algorithm 15 with $i = \mathrm{fsize}(t)$ and state s of $\mathcal{T}$, returns a succinct NFA $\mathcal{N}$ such that*

$$|T(s^i, t)| \quad = \quad |\mathcal{L}_k(\mathcal{N})|.$$

Moreover, $|\mathcal{N}| \leq 3(im)^4$, where m is the size of $\mathcal{T}$.

Proof. Let $u_1, \ldots, u_k$ be the holes of t. Assume first that no two holes of t share the same parent, so that $\pi = p_1, p_2, \ldots, p_k$ is the main path of t with $p_i = \mathrm{parent}(u_i)$. Counting the number of

[10] Strictly speaking, $\mathrm{parent}(u_1), \ldots, \mathrm{parent}(u_k)$ is a sequence and is not necessarily a path in the tree, because there could be missing nodes between the elements of the sequence. However, for the purpose of the proof the missing nodes do not play any role and will be omitted.

elements of $T(s^i, t)$ is the same as counting all sequences of trees $t_1, \ldots, t_k$ over Σ such that there exists a run ρ of $\overline{\mathcal{T}}$ over the tree $t[u_1 \to t_1] \cdots [u_k \to t_k]$ with $\rho(\lambda) = s^i$. In other words, we hang t_1 on u_1, ..., t_k on u_k to form a tree that is accepted by $\overline{\mathcal{T}}$ when s^i is the initial state. Then the plan of the reduction is to produce a succinct NFA $\mathcal{N}$ such that all words accepted by $\mathcal{N}$ are of the form $t_1 \cdots t_k$ with $t[u_1 \to t_1] \cdots [u_k \to t_k] \in T(s^i, t)$.

For the construction of $\mathcal{N}$ it will be useful to consider the following extension of $\overline{\mathcal{T}}$ over partial trees. Recall the definition of $\overline{\mathcal{T}} = (\overline{S}, \Sigma, \overline{\Delta}, s_{\text{init}})$ from Section 12.5, but assuming here that the unfolding is done for i levels. Then define $\overline{\mathcal{T}}^* = (\overline{S}, \Sigma \cup [i], \overline{\Delta}^*, s^i_{\text{init}})$ such that $\overline{\Delta}^* = \overline{\Delta} \cup \{(s^j, j, \lambda) \mid s^j \in \overline{S}\}$, namely we add to $\overline{\mathcal{T}}$ special transitions over holes when the level j of s^j coincides with the value of the hole. Intuitively, if we have a run ρ of $\overline{\mathcal{T}}^*$ over t with $\rho(\lambda) = s^i$ and $T(\rho(u_\ell)) \neq \emptyset$ for every $\ell \in [k]$, then t can be completed with trees $t_1 \in T(\rho(u_1)), \ldots, t_k \in T(\rho(u_k))$ such that $t[u_1 \to t_1] \cdots [u_k \to t_k] \in T(s^i, t)$.

Let $i_1, \ldots, i_k$ be the sizes $t(u_1), \ldots, t(u_k)$ on the holes $u_1, \ldots, u_k$, respectively. Furthermore, let $j_1, \ldots, j_k$ be the final sizes of the subtrees of t hanging from nodes $p_1, \ldots, p_k$, respectively. That is, if t_1 is the subtree hanging from p_1 in t, then $j_1 = \text{fsize}(t_1)$, and so on. Note that by the definition of the main path π, we have that $j_1 > j_2 > \ldots > j_k$ (since each p_i is the parent of p_{i+1}). We now have the ingredients to define the succinct NFA $\mathcal{N} = (S_\mathcal{N}, \Gamma, \Delta_\mathcal{N}, s_0, s_e)$. The set $S_\mathcal{N}$ of states will be a subset of the states of $\overline{S}$, plus two additional states s_0 and s_e, formally, $S_\mathcal{N} = \bigcup_{\ell=1}^{k} \{q^{j_\ell} \in \overline{S} \mid q \in S\} \cup \{s_0, s_e\}$. The set $\Delta_\mathcal{N}$ of transitions is defined as follows: for every states $q_1, q_2, r \in S$ and $\ell \in [k-1]$, we add a transition $(q_1^{j_\ell}, T(r^{i_\ell}), q_2^{j_{\ell+1}}) \in \Delta_\mathcal{N}$ if there exists a run ρ of $\overline{\mathcal{T}}^*$ over t such that $\rho(p_\ell) = q_1^{j_\ell}$, $\rho(p_{\ell+1}) = q_2^{j_\ell+1}$, and $\rho(u_\ell) = r^{i_\ell}$. Moreover, assuming that $\&$ is a fresh symbol, we add transition $(u_0, \{\&\}, q^{j_1})$ to $\Delta_\mathcal{N}$ if there exists a run ρ of $\overline{\mathcal{T}}^*$ over t such that $\rho(p_1) = q^{j_1}$, and $\rho(\lambda) = s^i$. Finally, we add transition $(q^{j_k}, T(r^{i_k}), u_e)$ to $\Delta_\mathcal{N}$.

Note that all transitions in the succinct NFA are directed from level $j_{\ell-1}$ to level j_ℓ with $j_{\ell-1} > j_\ell$, for some $\ell \in \{1, 2, \ldots, k-1\}$, which implies that $\mathcal{N}$ is unrolled. Here, the level j_ℓ is defined as the set of states $\{q^{j_\ell} \in \overline{S} \mid q \in S\}$. Furthermore, note that transitions are labeled by sets $T(r^{i_\ell})$ where $i_\ell < i$, which are represented by tree automaton $\overline{\mathcal{T}}[r^{i_\ell}]$ for $i_\ell < i$. Thus, the conditions required by Definition 12.6.1 are satisfied since for each transition label $T(r^{i_\ell})$, it holds that $|T(r^{i_\ell})|$ is at most exponential in the size of $\overline{\mathcal{T}}[r^{i_\ell}]$, and by Algorithm 15, we have already precomputed values such that we can check membership, approximate its size, and obtain an almost uniform sample from $T(r^{i_\ell})$. Finally, the existence of the run ρ for the definition of each transition in $\Delta_\mathcal{N}$ can be checked in polynomial time in the size of t [CDG+07] and, thus, $\mathcal{N}$ can be constructed from t and $\mathcal{T}$ in polynomial time.

It's only left to show that $|\mathcal{L}_k(\mathcal{N})| = |T(s^i, t)|$. For this, note that every word accepted by $\mathcal{N}$ is of length $k+1$ and of the form $\&t_1t_2\cdots t_k$. Then consider the function that maps words $\&t_1t_2\ldots t_k$ to the tree $t[u_1 \to t_1]\cdots[u_k \to t_k]$. One can show that each such a tree is in $T(s^i, t)$, and then the function goes from $\mathcal{L}_k(\mathcal{N})$ to $T(s^i, t)$. Furthermore, the function is a bijection. Clearly, if we take two different words, we will produce different trees in $T(s^i, t)$, and then the function is injective. To show that the function is surjective, from a tree $t' \in T(s^i, t)$ and a run ρ of $\overline{\mathcal{T}}$ over t', we can build the word $\&t_1t_2\ldots t_k$ where each t_i is the subtree hanging from the node u_i in t'. Also, this word is realized by the following sequence of transitions in $\mathcal{N}$:

$$(s_0, \{\&\}, \rho(p_1)), (\rho(p_1), T(\rho(u_1)), \rho(p_2)), \ldots, (\rho(p_{k-1}), T(\rho(u_{k-1})), \rho(p_k)), (\rho(p_k), T(\rho(u_k)), s_e).$$

Thus, the function is surjective. Hence, from the existence of a bijection from $\mathcal{L}_k(\mathcal{N})$ to $T(s^i, t)$, we conclude that $|\mathcal{L}_k(\mathcal{N})| = |T(s^i, t)|$.

Recall that the size of succinct NFA $\mathcal{N}$ is defined as $|\mathcal{N}| = |S_\mathcal{N}| + |\Delta_\mathcal{N}| + \sum_{(s,A,s')\in\Delta_\mathcal{N}} \|A\|$. Thus, given that $|S_\mathcal{N}| = im$, each set label $A = T(\rho(u_i))$ is represented by the tree automaton $\overline{\mathcal{T}}[\rho(u_i)]$ and the size of $\overline{\mathcal{T}}[\rho(u_i)]$ is bounded by $(im)^2$, we conclude that $|\Delta_\mathcal{N}| \leq (im)^3$ and $\sum_{(s,A,s')\in\Delta_\mathcal{N}} \|A\| \leq (im)^4$. Putting everything together, we conclude that $|\mathcal{N}| \leq 3(im)^4$, which was to be shown.

To finish with the proof, we need to consider the sequence $u_1, \ldots, u_k$ of holes of t, and assume that two holes of t share the same parent, so that $\pi = p_1, p_2, \ldots, p_k$ is the main path of t, with $p_i = \text{parent}(u_i)$ for each $i \in [k-1]$ and $p_k = u_k$. The proof for this case can be done in a completely analogous way. □

In the following theorem, we show how to estimate $|\mathcal{L}_k(\mathcal{N})|$ for a given unrolled succinct NFA $\mathcal{N}$ and integer $k \geq 1$ given in unary (recall the definition of unrolled succinct NFA from the beginning of this section).

Theorem 149. *Let $\mathcal{N}$ be an unrolled succinct NFA, $k \geq 1$ $\epsilon \in (100|\mathcal{N}|^4\epsilon_0, 1)$, where ϵ_0 is as in Definition 12.6.1. Moreover, fix $\delta \in (0, 1/2)$ and assume that N satisfies that $|\mathcal{L}_k(\mathcal{N})| \leq N$. Then there exists an algorithm that with probability at least $1-\delta$ outputs a value $\widetilde{N}$ such that $\widetilde{N} = (1 \pm \epsilon)|\mathcal{L}_k(\mathcal{N})|$. The algorithm runs in time*

$$O\left(T \cdot \frac{\log(N/\epsilon)\log^2(1/\delta)|\mathcal{N}|^{18}}{\epsilon^4}\right),$$

where T is as in Definition 12.6.1, and makes at most

$$O\left(\frac{\log^2(1/\delta)\log(N/\epsilon)|\mathcal{N}|^{18}}{\epsilon^4}\right)$$

queries to the sampling oracle. Furthermore, there is an almost uniform sampler which returns elements of $\mathcal{L}_k(\mathcal{N})$ such that

$$\mathbf{Pr}\left[\textit{outputs } \pi\right] \quad = \quad (1 \pm \epsilon)\frac{1}{|\mathcal{L}_k(\mathcal{N})|}$$

for every $\pi \in \mathcal{L}_k(\mathcal{N})$, and has the same runtime and oracle complexity as above.

As a corollary based on the reduction described earlier, we obtain the following.

Corollary 12.6.3. *Let $\mathcal{N}$ be a succinct NFA, $k \geq 1$, $u, v \in V$, $\epsilon \in (100((k+1)|G|)^4\epsilon_0, 1)$, where ϵ_0 is as in Definition 12.6.1. Moreover, fix $\delta \in (0, 1/2)$ and assume that N satisfies that $|\mathcal{L}_k(\mathcal{N})| \leq N$. Then there exists an algorithm that, with probability at least $1-\delta$ outputs a value $\widetilde{N}$ such that $\widetilde{N} = (1 \pm \epsilon)|\mathcal{L}_k(\mathcal{N})| \leq N$. The algorithm runs in time*

$$O\left(T \cdot \frac{\log(N/\epsilon)\log^2(1/\delta)(k|\mathcal{N}|)^{18}}{\epsilon^4}\right),$$

where T is as in Definition 12.6.1, and makes at most

$$O\left(\frac{\log^2(1/\delta)\log(N/\epsilon)(k|\mathcal{N}|)^{18}}{\epsilon^4}\right)$$

queries to the sampling oracle. Furthermore, there is an almost uniform sampler which returns elements of $\mathcal{L}_k(\mathcal{N})$ such that

$$\mathbf{Pr}\left[\textit{outputs } \pi\right] \quad = \quad (1 \pm \epsilon)\frac{1}{|\mathcal{L}_k(\mathcal{N})|}$$

for every $\pi \in \mathcal{L}_k(\mathcal{N})$, and has the same runtime and oracle complexity as above.

Proof. The result follows from Theorem 149, as well as the reduction described earlier from arbitrary succinct NFAs to unrolled succinct NFAs. Notice that in this reduction the size of $\mathcal{N}$ increases by a factor of $O(k)$, which completes the proof. □

Using Theorem 149, we can prove Lemma 12.5.3.

Proof of Lemma 12.5.3. Recall that we assume given a tree automaton $\mathcal{T} = (S, \Sigma, \Delta, s_{\text{init}})$ over binary trees, a natural number $n \geq 1$ given in unary, the relative error $\epsilon \in (0, 1)$ and a value $\delta \in (0, 1/2)$. Moreover, we assume that $m \geq 3$ is the size of $\mathcal{T}$, which we define as $m = \|\Delta\|$.

Let $s \in S$ and t be a partial tree such that $\text{fsize}(t) = i$. By using Lemma 12.6.2, we can construct in polynomial-time a succinct NFA $\mathcal{N}$ such that $N(s^i, t) = |T(s^i, t)| = |\mathcal{L}_k(\mathcal{N})|$. Moreover, by the reduction of Lemma 12.6.2, each label set A of a transition in $\mathcal{N}$ is of the form $A = T(s^j)$ for some state s and some $j < i$ in the graph. Thus, if we assume $\epsilon_0 = \epsilon 4nm$, then this gives us $\widetilde{N}(s^j) = (1 \pm \epsilon_0)N(s^j)$ as $\mathcal{N}$ is required to satisfy the properties of Definition 12.6.1. Moreover, define $\epsilon_1 = \epsilon_0(4nm)^{16}$, where ϵ_1 is the precision parameter from Theorem 149. Then we have that $\epsilon_1 > 300(nm)^{16}\epsilon_0 \geq 100|\mathcal{N}|^4\epsilon_0$ as required by Theorem 149, where here we used the fact that $|\mathcal{N}| \leq 3(nm)^4$ in the reduction of Lemma 12.6.2. Moreover, $\epsilon_1 = (4nm)^{17}\epsilon < 1$, as also required by Theorem 149, since we assume that $\epsilon < 1/(4nm)^{18}$. Finally, we also set $\delta_0 = \delta^{(nm)^3}$ to be the failure probability as in Theorem 149. Thus, by Theorem 149, using at most $O(\log^2(1/\delta_0)\log(N/\epsilon_1)(nm)^{4\cdot 18}/\epsilon_1^4) = O(\log^2(1/\delta)\log(N/\epsilon)(nm)^{10}/\epsilon^4)$ samples, we obtain a $(1 \pm \epsilon_1)$-estimate of the size of the number of labeled paths with probability $1 - \delta_0 = 1 - \delta^{(nm)^3}$. By Lemma 12.6.2, we therefore obtain the same estimate of the partition t, for a given partial tree t.

We now bound the number of ordered, rooted, labeled trees of size n. By Cayley's formula, we can bound the number of unlabeled, unordered, undirected trees by n^{n-2}. The number of rooted, unordered, undirected, unlabeled trees can then be bounded by n^{n-1}. For each tree, every vertex has $|\Sigma| \leq m$ choices of a labeling, thus there are $m^n n^{n-1} < (nm)^{nm}$ labeled unordered, undirected rooted trees (recall that $m \geq 3$). Finally, for each such a tree, we can bound the number of ways to transform it into an ordered and directed tree by $(2n)!$, which gives a bound of $(2n)! \cdot (nm)^{nm} \leq (nm)^{(nm)^2}$ (recall again that $m \geq 3$). Note that this also implies that $N \leq (nm)^{(nm)^2}$, which gives a total sample complexity bound of $O(\log^2(1/\delta)(nm)^{13}/\epsilon^5)$. Observe that the number of partial trees t such that $\text{fsize}(t) = i$ is bounded by $n(nm)^{(nm)^2}$. In particular, this bound is obtained by considering that $m \geq 3$ and the fact that the number of labels for partial trees is at most $|\Sigma| + n \leq m + n$. Now by a union bound and considering that $\delta < 1/2$, with probability

$$1 - (nm)(nm)^{(nm)^2}\delta^{(nm)^3} \geq 1 - \delta^{(nm)^3 - ((nm)^2+1)\log(nm)} \geq 1 - \delta^{nm},$$

$\text{EstimatePartition}(t, s^i, \{\widetilde{T}_i(s^j)\}_{s\in S, j<i}, \{\widetilde{N}(s^j)\}_{s\in S, j\leq i}, \epsilon, \delta)$ returns a $(1 \pm \epsilon_1) = (1 \pm (4nm)^{17}\epsilon)$ estimate for all trees t such that $\text{fsize}(t) = i$ and for all states s^i such that $s \in S$. Finally, note that the runtime is $\text{poly}(n, m, 1/\epsilon, \log(1/\delta))$ since it is bounded by a polynomial

in the sample complexity, which is polynomial in n, m, $1/\epsilon$ and $\log(1/\delta)$ by Lemma 12.6.6 and Theorem 149. □

12.6.1 Approximate Counting of Accepted Words in Succinct NFAs

The goal of this section is to prove Theorem 149. In what follows, fix a succinct NFA $\mathcal{N} = (S, \Gamma, \Delta, s_{\text{init}}, s_{\text{final}})$ over a finite set of labels Γ and recall that the label sets of Δ have to satisfy the conditions of Definition 12.6.1. Besides, assume that $\mathcal{N}$ is unrolled, and recall the definition of unrolled succinct NFA from the beginning of this section. Without loss of generality, assume that S only contains states which lie on a path from s_{init} to s_{final}. Furthermore, let $s_0, \ldots, s_n$ be a topological order of the states in S such that $s_0 = s_{\text{init}}$ and $s_n = s_{\text{final}}$, where $|S| = n + 1$. In other words, every path from s_0 to s_n can be written in the form $s_0, s_{i_1}, s_{i_2}, \ldots, s_n$, where $1 \leq i_1 \leq i_2 \leq \cdots \leq n$. Finally, for brevity, we write $r = |\mathcal{N}|$.

For the sake of presentation, for every state s_i, let $W(s_i) = \mathcal{L}(\mathcal{N}_{s_i})$ and $N(s_i) = |\mathcal{L}(\mathcal{N}_{s_i})|$, where $\mathcal{N}_{s_i}$ is an exact copy of $\mathcal{N}$ only with the final state changed to s_i. Then our goal is to estimate $N(s_n)$. Similar than for the previous section, we will simultaneously compute estimates $\widetilde{N}(s_i)$ of the set sizes $N(s_i)$, as well as multi-set sketches $\widetilde{W}(s_i)$ filled with i.i.d. nearly-uniform samples from $W(s_i)$. We do this iteratively for $i = 0, 1, 2, \ldots, n$. For the remainder of the section, fix $\epsilon, \epsilon_0, \delta$ as in Theorem 149, and assume that $\mathcal{N}$ satisfies the conditions of Definition 12.6.1, from which we know that $|W(s_i)| \leq N$ for each node s_i, for some $N \leq 2^{\text{poly}(r)}$. Set $\gamma = \log(1/\delta)$. Finally, since the FPRAS must run in time $\text{poly}(r, 1/\epsilon)$, we can assume $\epsilon < 1/(300r)$ without loss of generality.

We first observe that membership in $W(s_i)$ is polynomial-time testable given polynomial-time membership tests for each label A.

Proposition 12.6.4. *Suppose that given a sequence $a_1 \ldots a_t \in \Gamma^t$, we can test in time T whether $a_i \in A$ for each transition label A (T is the membership time in Definition 12.6.1). Then given any state s_j, we can test whether $a_1 \ldots a_t \in W(s_j)$ in time $O(|\Delta|T)$*

Proof. We can first remove all transitions not contained in a run of length exactly t from s to s_j in time $O(|\Delta|)$ by a BFS. Then for each transition $e = (s', A, s'')$ remaining which is on the i-th step from s to s_j, with $i \leq t$, we keep e if and only if $a_i \in A$. Note that i is unique for e as $\mathcal{N}$ is unrolled. It is now straightforward to check that s_j is reachable from s with the remaining transitions if and only if $a_1 \ldots a_t \in W(s_j)$. It is easy to check that the time needed by the entire procedure is $O(|\Delta|T)$. □

Now, analogous to the prior section, we define the following properties for each state s_i. Recall that we use $r = |\mathcal{N}|$ to denote the size of $\mathcal{N}$ for brevity.

Property 3: We say that s_i satisfies Property 3 if $\widetilde{N}(s_i) = (1 \pm i\epsilon/r)N(s_i)$.

Property 4: We say that s_i satisfies Property 4 if for every subset $L \subseteq \{0, 1, 2, \ldots, i-1\}$, we have that

$$\left| \frac{\left|\widetilde{W}(s_i) \setminus \left(\bigcup_{j \in L} W(u_j)\right)\right|}{|\widetilde{W}(s_i)|} - \frac{\left|W(s_i) \setminus \left(\bigcup_{j \in L} W(u_j)\right)\right|}{|W(s_i)|} \right| \leq \frac{\epsilon}{r}$$

Moreover, we have that the subsets $\widetilde{W}(s_i)$ are of size $|\widetilde{W}(s_i)| = O(\frac{r^3\gamma}{\epsilon^2})$.

Property 5: We say that s_i satisfies Property 5 if we have a polynomial-time algorithm which returns independent samples from $W(s_i)$, such that for all $w \in W(s_i)$:

$$\mathbf{Pr}\left[\text{ outputs } w \mid \neg\textbf{FAIL}\right] = \left(1 \pm \frac{\epsilon}{3r^2}\right)\frac{1}{N(s_i)}$$

The algorithm is allowed to fail with probability at most $1/4$, in which case it returns **FAIL** (and returns no element). Finally, each run of the algorithm is allowed to use at most $O(\frac{\log(N/\epsilon)\gamma r^{11}}{\epsilon^2})$ oracle calls to the sampling oracle of Definition 12.6.1.

Lemma 12.6.5. *Suppose that Properties 3, 4 and 5 hold for all s_j with $j < i$. Then with probability at least $1 - 2^{-\gamma r}$ we can return an estimate $\widetilde{N}(s_i) = (1 \pm i\epsilon/r)N(s_i)$. In other words, under these assumptions it follows that Property 3 holds for s_i. Moreover, the total number of calls to the sampling oracle of Definition 12.6.1 is $O(\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4})$, and the total runtime can be bounded by $O(T\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4})$, where T is the membership test time in Definition 12.6.1.*

Proof. First note that for $i = 0$, Property 3 trivially holds since $W(s_0) = \emptyset$. Otherwise, let $i \geq 1$ and $(v_1, A_1, s_i), \ldots, (v_k, A_k, s_i) \in \Delta$ be the set of all transitions going into s_i (recall that we sorted the $\{s_i\}_{i \in [0,n]}$ by a topological ordering, so $v_1, \ldots, v_k \in \{s_0, \ldots, s_{i-1}\}$). Observe $W(s_i) = \bigcup_{j=1}^{k} (W(v_j) \cdot A_j)$. Fix a transition (v_j, A_j, s_i) and assume that $\epsilon_0 = \epsilon/(100r^4)$ in Definition 12.6.1, so that we are given estimates $\widetilde{N}(A_j) = (1 \pm \epsilon_0)|A_j|$ since $\mathcal{N}$ satisfies the conditions in this definition. Then the number of words reaching s_i through (v_j, A_j, s_i) is given

by $N(v_j)|A_j|$, and it can be estimated as follows assuming that $v_j = s_k$ with $k < i$:

$$\begin{aligned}\widetilde{N}(v_j) \cdot \widetilde{N}(A_j) &= (1 \pm k\epsilon/r)(1 \pm \epsilon_0)N(v_j)|A_j| \\ &= (1 \pm (i-1)\epsilon/r)(1 \pm \epsilon/(100r^4))N(v_j)|A_j| \\ &= (1 \pm (i - 1 + 1/r^2)(\epsilon/r))N(v_j)|A_j|.\end{aligned}$$

Notice that in this deduction we use the fact that $i - 1 \leq r$. Let p_j denote be the probability that a uniformly drawn $s \sim (W(v_j) \cdot A_j)$ is not contained in $\bigcup_{j'<j} (W(v_{j'}) \cdot A_{j'})$. Then we can write $W(s_i) = \sum_{j=1}^{k} N(v_j)|A_j|p_j$. We now estimate p_j via $\tilde{p}_j$. By Property 5 and the assumptions from Definition 12.6.1, we can obtain nearly uniform samples $w \sim W(v_j)$ and $a \in A_j$ in polynomial time, such that the probability of sampling a given w and a are $(1 \pm \epsilon/(3r^2))N(v_j)^{-1}$ and $(1 \pm \epsilon_0)|A_j|^{-1}$, respectively. Moreover, $w \cdot a$ is a sample from $W(v_j) \cdot A_j$, such that for any $w' \cdot a' \in W(v_j) \cdot A_j$:

$$\mathbf{Pr}\left[w \cdot a = w' \cdot a'\right] = \left(1 \pm \frac{\epsilon}{3r^2}\right) \frac{1 \pm \epsilon_0}{N(v_j)|A_j|} \tag{12.10}$$

Note that the relative error $(1 \pm \epsilon/(3r^2))(1 \pm \epsilon_0)$ can be bounded in the range $(1 \pm 2\epsilon/(5r^2))$ using the fact that $\epsilon_0 = \epsilon/(100r^4)$. We repeat this sampling process $d = O(\gamma r^5/\epsilon^2)$ times, obtaining samples $w_1a_1, \ldots, w_da_d \sim W(v_j) \cdot A_j$ and set $\tilde{p}_j$ to be the fraction of these samples not contained in $\bigcup_{j'<j} (W(v_{j'}) \cdot A_{j'})$. Then by Hoeffding's inequality, with probability $1 - 2^{-\gamma r}$ we have that $\tilde{p}_j = p_j \pm 2\epsilon/(3r^2)$ (here we use (12.10), which tell us that the expectation of $\tilde{p}_j$ is at most $2\epsilon/(5r^2)$ far from the correct expectation p_j). We then set:

$$\begin{aligned}\widetilde{N}(s_i) &= \sum_{j=1}^{k} \widetilde{N}(v_j) \cdot \widetilde{N}(A_j) \cdot \tilde{p}_j \\ &= (1 \pm (i - 1 + 1/r^2)(\epsilon/r)) \sum_{j=1}^{k} N(v_j) \cdot |A_j| \cdot \left(p_j \pm \frac{2\epsilon}{3r^2}\right) \\ &= (1 \pm (i - 1 + 1/r^2)(\epsilon/r)) \left[\sum_{j=1}^{k} N(v_j)|A_j|p_j \pm \frac{2\epsilon}{3r^2} \sum_{j=1}^{k} N(v_j)|A_j|\right] \\ &= (1 \pm (i - 1 + 1/r^2)(\epsilon/r)) \left[N(s_i) \pm \frac{2\epsilon}{3r^2} \sum_{j=1}^{k} N(s_i)\right] \\ &= (1 \pm (i - 1 + 1/r^2)(\epsilon/r)) \left[N(s_i) \pm 2\epsilon/(3r)N(s_i)\right] \\ &= (1 \pm i\epsilon/r)N(s_i)\end{aligned}$$

as desired. Note that we need only compute $\tilde{p}_j$ for at most r values of v_j, thus the total number

of samples required is $O(\gamma r^6/\epsilon^2)$. By Property 5, each sample required $O(\frac{\log(N/\epsilon)\gamma r^{11}}{\epsilon^2})$ oracle calls, thus the total oracle complexity is $O(\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4})$ as needed. By Proposition 12.6.4, each membership test required while computing the probabilities $\tilde{p}_j$ required $O(Tr)$ time, thus the total runtime can be bounded by $O(T\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4})$, which was to be shown.

□

Algorithm 17: SAMPLEFROMSTATE$(s_i, \widetilde{N}(s_i))$

if $\widetilde{N}(s_i) = 0$ **then**
  return $\perp$ `// ⊥ indicates that W(s_i) is empty`
Initialize $w \leftarrow \lambda,\ q \leftarrow 1$. `// λ is the empty string`
for $\beta = 1, 2, \ldots, d(s_0, s_i)$ **do**
  while $|w| < \beta$ **do**
    Let $\mathcal{F}(s_i, w) = \{(x_1, A_1, y_1), \ldots, (x_k, A_k, y_k)\}$, set $Z_j = \widetilde{N}(x_j)\widetilde{N}(A_j)$ for each $j \in [k]$, and $Z = \sum_{j=1}^{k} Z_j$.
    Order the Z_i's so that $Z_1 \geq Z_2 \geq \cdots \geq Z_k$.
    Sample $j \sim [k]$ with probability $\frac{Z_j}{Z}$.
    Obtain an almost uniform sample $a \sim A_j$. `// via Definition 12.6.1`
    Let $\mathbf{B}(a) = \{j' \in [k] \mid a \in A_{j'}\}$ and accept a with probability:

$$q_{a,j} \quad = \quad \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a)\,:\, j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|}$$

    if a *is accepted* **then**
      $\rho \leftarrow 0$, $M \leftarrow \Theta(\frac{\log(N/\epsilon)\gamma r^{10}}{\epsilon^2})$.
      `// ρ approximates the probability that a trial fails to accept some a`
      for $h = 1, 2, \ldots, M$ **do**
        Sample $j \sim [k]$ with probability $\frac{Z_j}{Z}$, then sample $a_h \sim A_j$.
        With probability $1 - q_{a_h,j}$ increment $\rho \leftarrow \rho + 1$.
      $\rho \leftarrow \frac{\rho}{M}$, and update: $q \leftarrow q \cdot \left(\frac{\sum_{j' \in B(a)} \frac{\widetilde{N}(x_{j'})}{Z} q_{a,j'}}{1-\rho}\right)$
      $w \leftarrow aw$
`// q approximates the probability w was sampled so far`
With probability $\frac{1}{2q\widetilde{N}(s_i)}$ **return** w, otherwise **return FAIL**.

We now describe our sampling procedure. To do so, we will first develop some notation. We extend our previous notation and use $\mathcal{N}_{s,s'}$ to denote an exact copy of $\mathcal{N}$ but with s as the initial state and s' as the final state. Let s_i be a vertex and $w \in \Gamma^*$ a sequence of symbols. If w contains at least one symbol, then let $\mathcal{F}(s_i, w) = \{(s_a, A, s_b) \in \Delta \mid w \in \mathcal{L}_k(\mathcal{N}_{s_b,s_i})$, namely, $\mathcal{F}(s_i, w)$ is the set of all transitions (s_a, A, s_b) incident to a state s_b from which we can reach s_i by a path labeled by w. Otherwise, we have that $w = \lambda$, where λ is the empty string, and $\mathcal{F}(s_i, \lambda)$ is defined as $\{(s_a, A, s_b) \in \Delta \mid s_b = s_i\}$. Moreover, let $|w|$ be the length of w and $d : S \times S \to \mathbb{N}$ be the distance metric between states of $\mathcal{N}$ when considered as a graph, i.e. $d(s, s')$ is the number of transitions that we need to make to get from s to s'. Without loss of generality (by unrolling the succinct NFA if needed), we can make sure that d is well defined.

We now present our main sampling algorithm of this section: Algorithm 17. For ease of presentation, Algorithm 17 is written as a Las Vegas randomized algorithm, which could potentially have unbounded runtime. However, by simply terminating the execution of the algorithm after a fixed polynomial runtime and outputting an arbitrary string of bits, the desired correctness properties of the sampler will hold. The analysis of Algorithm 17, along with the finite-time termination procedure, is carried out in the proof of Lemma 12.6.6 below.

Lemma 12.6.6. *Suppose Property 3 holds for all levels $j \leq i$, and Property 4 holds for all levels $j < i$. If $W(s_i) = \emptyset$, then* SAMPLEFROMSTATE$(s_i, \widetilde{N}(s_i))$ *return $\perp$ with probability 1. Otherwise, conditioned on not outputting* **FAIL**, SAMPLEFROMSTATE$(s_i, \widetilde{N}(s_i))$ *returns $w \sim W(s_i)$ from a distribution $\mathbf{D}$ over $W(s_i)$ such that*

$$\mathbf{D}(w) \quad = \quad \left(1 \pm \frac{\epsilon}{3r^2}\right) \frac{1}{|W(s_i)|}$$

for all $w \in W(s_i)$. Moreover, the algorithm uses at most $O(\frac{\log(N/\epsilon)\gamma r^{11}}{\epsilon^2})$ calls to the uniform sampling oracle of Definition 12.6.1, runs in time $O(T\frac{\log(N/\epsilon)\gamma^2 r^{15}}{\epsilon^4})$, and outputs **FAIL** *with probability at most $3/4$.*

Proof. Assume first $W(s_i) = \emptyset$, so that $N(s_i) = 0$. Then given that $\widetilde{N}(s_i) = (1 \pm \epsilon)N(s_i)$ by Property 3, we conclude that $\widetilde{N}(s_i) = 0$ and the algorithm returns $\perp$ in line 2. Notice that if $W(s_i) \neq \emptyset$, then $N(s_i) > 0$ and, therefore, $\widetilde{N}(s_i) > 0$ by Property 3. Thus, if $W(s_i) \neq \emptyset$, then the algorithm does not return $\perp$.

Assume that $W(s_i) \neq \emptyset$, and notice that this implies $r \geq 4$. Consider an element w sampled so far at any intermediate state of the execution of SAMPLEFROMSTATE$(s_i, \widetilde{N}(s_i))$. Let $W(s_i, w) = \{t \in W(s_i) \mid t = w' \cdot w\}$. In other words, $W(s_i, w) \subseteq W(s_i)$ is the subset of words

with suffix equal to w. We now want to sample the next symbol $a \in \Gamma$ conditioned on having sampled the suffix w of a path so far. In other words, we want to sample a with probability proportional to the number of words in $W(s_i, w)$ which have the suffix aw, meaning we want to choose a with probability:

$$\frac{|W(s_i, aw)|}{|W(s_i, w)|}.$$

However, we do not know these sizes exactly, so we must approximately sample from this distribution. Let us consider the probability that our algorithm samples a on this step (given w). For the algorithm to sample a, it must first choose to sample a transition (x_j, A_j, y_j) from the set $\mathcal{F}(s_i, w) = \{(x_1, A_1, y_1), (x_2, A_2, y_2), \ldots, (x_k, A_k, y_k)\}$ such that $a \in A_i$, which occurs with probability Z_j/Z with $Z = \sum_{j'=1}^{k} Z_{j'}$ and $Z_{j'} = \widetilde{N}(x_{j'})\widetilde{N}(A_{j'})$. Then, on the call to the oracle on line 9, it must obtain $a \sim A_j$ as the almost uniform sample, which occurs with probability $(1 \pm \epsilon_0)|A_j|^{-1}$ by Definition 12.6.1. Finally, it must choose to keep a on line 10, which occurs with probability $\frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|}$, where $\mathbf{B}(a) = \{j' \in [k] \mid a \in A_{j'}\}$. Thus, altogether, the probability that we choose $a \in \Gamma$ on this step is

$$\begin{aligned}
&\sum_{j \in \mathbf{B}(a)} \frac{\widetilde{N}(x_j)\widetilde{N}(A_j)}{Z} \cdot \frac{1 \pm \epsilon_0}{|A_j|} \cdot \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|} \\
&\quad = (1 \pm 3\epsilon_0) \sum_{j \in \mathbf{B}(a)} \frac{\widetilde{N}(x_j)}{Z} \cdot \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|} \qquad (12.11) \\
&\quad = (1 \pm 3\epsilon_0) \sum_{j \in \mathbf{B}(a)} \frac{(1 \pm \epsilon)|W(x_j)|}{Z} \cdot \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|} \\
&\quad = (1 \pm 2\epsilon) \sum_{j \in \mathbf{B}(a)} \frac{|W(x_j)|}{Z} \cdot \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|} \\
&\quad = (1 \pm 2\epsilon) \sum_{j \in \mathbf{B}(a)} \frac{|W(x_j)|}{Z} \left[\frac{\left|W(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right|}{\left|W(x_j)\right|} \pm \frac{\epsilon}{r}\right] \\
&\quad = (1 \pm 2\epsilon) \frac{1}{Z} \sum_{j \in \mathbf{B}(a)} \left[\left|W(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right| \pm |W(x_j)|\frac{\epsilon}{r}\right] \\
&\quad = (1 \pm 2\epsilon) \frac{1}{Z} \left(\sum_{j \in \mathbf{B}(a)} \left|W(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(a) \,:\, j' < j} W(x_{j'})\right)\right| \pm \sum_{j \in \mathbf{B}(a)} |W(x_j)|\frac{\epsilon}{r}\right) \\
&\quad = (1 \pm 2\epsilon) \frac{1}{Z} \left(|W(s_i, aw)| \pm |W(s_i, aw)|\epsilon\right) \qquad (12.12) \\
&\quad = (1 \pm 4\epsilon) \frac{1}{Z} |W(s_i, aw)|
\end{aligned}$$

Where equation (12.11) uses the fact that $\widetilde{N}(A_j) = (1 \pm \epsilon_0)|A_j|$, and equation (12.12) uses the fact that $W(s_i, aw) = (\bigcup_{j \in \mathbf{B}(a)} W(x_j)) \cdot \{aw\}$. The above demonstrates that on a **single** trial of the inner while loop in lines 5 to 17, conditioned on having chosen the sample w so far, for each $a \in W(s_i, w)$ we choose a with probability $(1 \pm 4\epsilon)\frac{|W(s_i,aw)|}{Z}$. However, we do not break out of the while loop on line 17 and move to the next step in the outer for loop in line 4 until we have chosen an $a \in W(s_i, w)$ to append to w. If on a given trial of the loop in line 5, the algorithm does not choose some element to append to w, we say that it outputs no sample. Call the event that we output some sample $\mathcal{E}_i$, and let $\mathcal{E}_i(a)$ denote the event that we specifically output $a \in \Gamma$. Then

$$\begin{aligned}
\mathbf{Pr}\left[\mathcal{E}_i\right] &= \sum_{a \in W(s_i,w)} \mathbf{Pr}\left[\mathcal{E}_i(a)\right] \\
&= (1 \pm 4\epsilon) \sum_{a \in W(s_i,w)} \frac{|W(s_i, aw)|}{Z} \\
&= (1 \pm 4\epsilon)\frac{|W(s_i, w)|}{Z}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbf{Pr}\left[\mathcal{E}_i(a) \mid \mathcal{E}_i\right] &= \frac{\mathbf{Pr}\left[\mathcal{E}_i(a)\right]}{\mathbf{Pr}\left[\mathcal{E}_i\right]} \\
&= \frac{(1 \pm 4\epsilon)}{(1 \pm 4\epsilon)}\left(\frac{Z}{|W(s_i, w)|}\right)\frac{|W(s_i, aw)|}{Z} \\
&= (1 \pm 10\epsilon)\frac{|W(s_i, aw)|}{|W(s_i, w)|}.
\end{aligned}$$

Thus, conditioned on outputting a sample at this step, we choose $a \in \Gamma$ with probability

$$(1 \pm 10\epsilon)\frac{|W(s_i, aw)|}{|W(s_i, w)|} \tag{12.13}$$

Observe the above is within $(1 \pm 10\epsilon)$ of the correct sampling probability.

Estimating the probability that we sample a given $w \in W(s_i)$. We now analyze the quantity q in the algorithm, and argue that at the point where line 18 is executed, q is a good approximation of the probability that our algorithm sample w at this point. Now let ρ^*_β be the probability that, within step $\beta \in \{1, 2, \ldots, d(s_0, s_i)\}$ of the outer for loop on line 4, a given run of the inner while loop between lines 5 to 17 fails to append a new sample a to w. Let ρ_β be the value that *we* assign to the variable ρ at the end of the for loop in line 13 (note that this loop is executed at most once

within step β of the outer loop 4). The variable ρ_β will be our estimate of ρ^*_β.

Note that each trial of the inner while loop is independent, so ρ^*_β only depends on the β from the outer loop, and the value of w sampled so far. Let $\mathbf{D}'_\beta(aw)$ be the exactly probability that entry a is chosen on step β of the outer loop of our algorithm, conditioned on having chosen w so far. Being in step β of the outer loop then implies that $|aw| = \beta$. Now fix any $w = w_1 w_2 \dots w_{d(s_0,s_i)} \in W(s_i)$. Let $\mathbf{D}'(w)$ be the exact probability that w is sampled at this point right before the execution of line 18. By definition we have

$$\mathbf{D}'(w) = \mathbf{D}'_0(w_{d(s_0,s_i)}) \cdot \left(\prod_{j=1}^{d(s_0,s_i)-1} \mathbf{D}'_j \left(w_{d(s_0,s_i)-j} \dots w_{d(s_0,s_i)} \right) \right)$$

so via (12.13) we obtain:

$$\mathbf{D}'(w) \;=\; \frac{1}{|W(s_i)|} \prod_{j=1}^{d(s_0,s_i)} (1 \pm 10\epsilon) \;=\; \frac{(1 \pm 10\epsilon)^r}{|W(s_i)|} \;=\; \frac{(1 \pm 20r\epsilon)}{|W(s_i)|} \tag{12.14}$$

Claim 12.6.7. *If q is the value the variable q takes at the point where line 18 is executed, given that $w = w_1 \dots w_{d(s_0,s_i)}$ is the value of e at this point, then*

$$\mathbf{D}'(w) \;=\; \left(1 \pm \frac{\epsilon}{50r^2}\right) q$$

with probability at least $1 - r(\epsilon/N)^{2\gamma}$.

Proof. To see this, consider step $\beta \in \{1, 2, \dots, d(s_0, s_i)\}$ of the for outer loop in line 4. We first claim that $\rho^*_\beta \leq 1 - \frac{1}{r}$. To see this, note that the probability that Z_1 is chosen is at least $\frac{1}{k} \geq \frac{1}{r}$, since we ordered $Z_1 \geq Z_2 \geq \dots \geq Z_k$, and if Z_1 is chosen the sample $a \sim A_1$ is never rejected, which completes the claim. Now each iteration of the for loop in line 13 defines a random variable Z which indicates if a random trial of the inner loop in line 5 would result in a failure. Here, if $Z = 1$ (a trials fails), then we increment $\rho = \rho + 1$, otherwise we do not. Thus $\mathbf{E}[Z] = \rho^*_\beta$, and by Hoeffding's inequality, after repeating $M = \Theta(\frac{\log(N/\epsilon)\gamma r^{10}}{\epsilon^2})$ times, it follows that with probability $(1 - (\epsilon/N)^{2\gamma})$ that we have $\rho_\beta = \rho^*_\beta \pm \epsilon/(400r^5)$ and, therefore, $1 - \rho_\beta = (1 \pm \epsilon/(400r^4))(1 - \rho^*_\beta)$ since $1/r \leq 1 - \rho^*_\beta$. Thus, it holds that

$$\frac{1}{1 - \rho^*_\beta} \;=\; \left(1 \pm \frac{\epsilon}{200r^4}\right) \frac{1}{1 - \rho_\beta} \tag{12.15}$$

Let $\tau = d(s_0, s_i) - \beta + 1$, so that on step β of the for outer loop in line 4 we are consid-

ering the probability that we sample a $w_\tau \in \Gamma$ given that we have already sampled $w_{-\tau} = w_{\tau+1} \dots w_{d(s_0,s_i)}$. Now as shown above, the probability that w_τ is accepted on one trial of the while loop is precisely:

$$q^*(w_\tau) = \sum_{j \in \mathbf{B}(w_\tau)} \frac{\widetilde{N}(x_j)\widetilde{N}(A_j)}{Z} \cdot \frac{\epsilon'}{|A_j|} \cdot \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(w_\tau) : j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|}.$$

Notice that we are not trying to bound $q^*(w_\tau)$ in this expression, we are computing the exact value of $q^*(w_\tau)$, but based on an unknown value ϵ'. However, we know by Definition 12.6.1 that $1 - \epsilon_0 \leq \epsilon' \leq 1 + \epsilon_0$. Thus, although we do not know the exact value of $q^*(w_\tau)$, we do know that $1 - 3\epsilon_0 \leq \widetilde{N}(A_j) \cdot \epsilon'/|A_j| \leq 1 + 3\epsilon_0$ by the assumptions of Definition 12.6.1. Thus, we can estimate $q^*(w_\tau)$ by

$$\hat{q}(w_\tau) = \sum_{j \in B(w_\tau)} \frac{\widetilde{N}(x_j)}{Z} \frac{\left|\widetilde{W}(x_j) \setminus \left(\bigcup_{j' \in \mathbf{B}(w_\tau) : j' < j} W(x_{j'})\right)\right|}{\left|\widetilde{W}(x_j)\right|}$$

so that $q^*(w_\tau) = (1 \pm 3\epsilon_0)\hat{q}(w_\tau)$. The probability that w_τ is accepted overall before moving to the next step of the loop is $\sum_{j=1}^{\infty} q^*(w_\tau)(\rho^*_\beta)^{j-1} = q^*(w_\tau)(\frac{1}{1-\rho^*_\beta})$, for which by equation (12.15) we have a $(1 \pm \epsilon/(200r^4))(1 \pm 3\epsilon_0) = (1 \pm \epsilon/(100r^3))$ estimate of via the value $\hat{q}(w_\tau)/(1 - \rho_\beta)$ (recall that $r \geq 4$). Note that this is precisely the value which we scale the variable q by after an iteration of the inner loop that appends a new sample a to w in line 17 of the algorithm. It follows that at the end of the main loop, we have:

$$\mathbf{D}'(w) \;=\; \left(1 \pm \frac{\epsilon}{100r^3}\right)^{d(s_0,s_i)} \cdot q \;=\; \left(1 \pm \frac{\epsilon}{50r^2}\right) \cdot q$$

as needed. Notice that this equality holds under the condition that for every $\beta = 1, \dots, d(s_0, s_i)$, it holds that $\rho_\beta = \rho^*_\beta \pm \epsilon/(400r^5)$, which occurs with probability $1 - (\epsilon/N)^{2\gamma}$ for each β. By a union bound, we obtain the desired success probability of at least $1 - r(\epsilon/N)^{2\gamma}$. □

Thus, by rejecting with probability $\frac{1}{2q\widetilde{N}(s_i)}$, it follows from Claim 12.6.7 that the true probability $\mathbf{D}^\star(w)$ that we output a given $w \in W(s_i)$ is

$$\mathbf{D}^\star(w) \;=\; \frac{\mathbf{D}'(w)}{2q\widetilde{N}(s_i)} \;=\; \left(1 \pm \frac{\epsilon}{50r^2}\right)\frac{1}{2\widetilde{N}(s_i)} \tag{12.16}$$

Note that for the above fact to be true, we need that $\frac{1}{2q} \leq \widetilde{N}(s_i)$, else the above rejection proba-

bility could be larger than 1. But again by Claim 12.6.7 we have that

$$\begin{aligned}\frac{1}{2q\widetilde{N}(s_i)} &\leq \left(1+\frac{\epsilon}{50r^2}\right)\frac{1}{2\widetilde{N}(s_i)\mathbf{D}'(w)} \\ &\leq (1+2\epsilon)\frac{1}{2|W(s_i)|\mathbf{D}'(w)} \\ &\leq (1+122r\epsilon)\frac{1}{2} \\ &\leq 3/4\end{aligned}$$

where the second to last inequality holds applying (12.14), and the last inequality holds give that $\epsilon < 1/(300r)$. Therefore, the rejection probability is always a valid probability. Similarly:

$$\begin{aligned}\frac{1}{2q\widetilde{N}(s_i)} &\geq \left(1-\frac{\epsilon}{50r^2}\right)\frac{1}{2\widetilde{N}(s_i)\mathbf{D}'(w)} \\ &\geq (1-2\epsilon)\frac{1}{2|W(s_i)|\mathbf{D}'(w)} \\ &\geq (1-42r\epsilon)\frac{1}{2} \\ &\geq 1/4\end{aligned}$$

Thus, by the above, we can bound the probability that we output **FAIL** on this last step by $3/4$ as required. Now, we are ready to analyize the true output distribution $\mathcal{D}$ over $W(s_i)$, which is given by the distribution $D^\star$ conditioned on not outputting **FAIL**. Now for any $w \in W(s_i)$, we

can apply equation (12.16) to compute $\mathcal{D}(w)$ via:

$$\begin{aligned}
\mathbf{D}(w) &= \mathbf{Pr}\left[\text{output } w \in W(s_i) \mid \neg\textbf{FAIL}\right] \\
&= \frac{\mathbf{Pr}\left[\text{output } w \in W(s_i) \wedge \neg\textbf{FAIL}\right]}{\mathbf{Pr}\left[\text{not output } \textbf{FAIL}\right]} \\
&= \frac{\mathbf{Pr}\left[\text{output } w \in W(s_i)\right]}{\mathbf{Pr}\left[\neg\textbf{FAIL}\right]} \\
&= \frac{\mathbf{D}^\star(w)}{\sum_{w \in W(s_i)} D^\star(w)} \\
&= \frac{(1 \pm \epsilon/(50r^2))}{2\widetilde{N}(s_i)(\sum_{w \in W(s_i)} \frac{1 \mp \epsilon/(50r^2)}{2\widetilde{N}(s_i)})} \\
&= \frac{(1 \pm \epsilon/(50r^2))}{\sum_{w \in W(s_i)}(1 \mp \epsilon/(50r^2))} \\
&= \frac{(1 \pm \epsilon/(50r^2))}{|W(s_i)|(1 \mp \epsilon/(50r^2))} \\
&= \left(1 \pm \frac{3\epsilon}{50r^2}\right)\frac{1}{|W(s_i)|} \\
&= \left(1 \pm \frac{\epsilon}{10r^2}\right)\frac{1}{|W(s_i)|}
\end{aligned}$$

which is the desired result.

Oracle complexity and runtime For the complexity of the sample procedure, note that each iteration to sample a $a \in \Gamma$ has failure probability at most $\frac{1}{r}$ independently, thus with probability $1 - (\epsilon/(rN))^2 2^{-10r\gamma}$ it requires at most $10r^3 \log(Nr/\epsilon)\gamma$ iterations. Thus with probability $1 - (\epsilon/(N))^2 2^{-10r\gamma}$, the total number of iterations required to produce a single sample (or output **FAIL** at the end) is $10r^4 \log(Nr/\epsilon)\gamma$. Note that each iteration that fails to accept an $a \in \Gamma$ produces one call to the unit oracle. Once an a is accepted, we run an experiment M times, which produces $M = O(\frac{\log(N/\epsilon)\gamma r^{10}}{\epsilon^2})$ oracle calls. Since this occurs at most r times, the total number of oracle calls is $O(\frac{\log(N/\epsilon)\gamma r^{11}}{\epsilon^2})$. Note that the runtime is dominant by the cost of the ρ estimation procedure, wherein the probability $q_{a_h,j}$ is computed at each step of line 13. Note that to compute $q_{a_h,j}$, we must test for each sample in $s \in \widetilde{W}(x_j)$ if s is contained in the union of at most r sets, which requires at most Tr runtime by the assumptions of Definition 12.6.1. Note that each set has size at most $O(\frac{\gamma r^3}{\epsilon^2})$. Thus the total runtime can be bounded by $O(T\frac{\log(N/\epsilon)\gamma^2 r^{15}}{\epsilon^4})$.

In summary, with probability $1 - (\epsilon/(N))^2 2^{-10r\gamma}$, the total number of samples (unit oracle calls) required is $O(\frac{\log(N/\epsilon)\gamma r^{11}}{\epsilon^2})$ (and the runtime is as stated above). Now if the sample complexity becomes too large we can safely output anything we would like (specifically, we can output

FAIL, or even an arbitrary sequence of bits). The probability that this occurs, or that any of our $O(r)$ estimate of the inner failure probabilities ρ fails to be within our desired bounds, is at most $(\epsilon/(N))^2 2^{-10r\gamma} + r(\epsilon/N)^2 2^{-\gamma} \leq (\epsilon/N)2^{-\gamma}$. Call the event that the sample complexity becomes too large $\mathcal{Q}$, and let $\mathcal{P}$ be the event that any of our $O(r)$ estimate of ρ fail to be within our desired bounds. We have just proven that

$$\mathbf{Pr}\left[\text{we output } w \in W(s_i) \mid \neg\mathcal{P}\right] = (1 \pm \epsilon/(10r^2))\frac{1}{|W(s_i)|}.$$

Now since $\mathbf{Pr}\left[\mathcal{P} \cup \mathcal{Q}\right] \leq (\epsilon/N)2^{-\gamma}$, we have

$$\begin{aligned}\mathbf{Pr}\left[\text{we output } w \in W(s_i) \mid \neg\mathcal{Q}\right] &= \mathbf{Pr}\left[\text{ we output } w \in W(s_i) \mid \neg\mathcal{Q}, \neg\mathcal{P}\right] \pm (\epsilon/N)2^{-\gamma} \\ &= \mathbf{Pr}\left[\text{ we output } w \in W(s_i) \mid \neg\mathcal{P}\right] \pm 3(\epsilon/N)2^{-\gamma},\end{aligned} \tag{12.17}$$

so it follows that for each $w \in W(s_i)$, we have

$$\begin{aligned}\mathbf{Pr}\left[\text{we output } w \in W(s_i) \mid \neg\mathcal{Q}\right] &= (1 \pm \epsilon/(10r^2))\frac{1}{|W(s_i)|} \pm 3(\epsilon/N)2^{-\gamma} \\ &= (1 \pm \epsilon/(10r^2))\frac{1}{u_i} \pm \epsilon\frac{3}{|W(s_i)|}2^{-\gamma} \\ &= (1 \pm \epsilon/(3r^2))\frac{1}{|W(s_i)|},\end{aligned} \tag{12.18}$$

which shows that our sampler is still correct even if we output random bits whenever $\mathcal{Q}$ fails to hold, which is the desired result taking $\gamma = \Omega(\log(r/\epsilon))$.

□

We can use the above sampling regime to now show that having properties $3, 4, 5$ for s_j with $j < i$ will imply them for s_i.

Lemma 12.6.8. *Fix any $\gamma > 0$. Suppose Properties $3, 4, 5$ hold for all s_j with $j < i$. Then with probability $1 - 2^{-10\gamma}$, properties $3, 4$ and 5 hold for s_i. Moreover, the total number of oracle calls is at most $O(\gamma^2 \log(N/\epsilon)r^{17}/\epsilon^4)$, and the total runtime is $O(T\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4})$.*

Proof. We obtain property 3 with probability $1-2^{-\gamma r}$ by Lemma 12.6.5, which uses $O(\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4})$ sampling oracle calls. By Lemma 12.6.6, conditioned on property 4 holding for all levels $j < i$ and property 3 holding for all $j \leq i$, we now have a procedure which can sample each $w \sim W(s_i)$ with probability in the range $(1 \pm \epsilon/(3r^2))\frac{1}{|W(s_i)|}$, and such that the sampler satisfies the other

conditions of Property 5. Thus property 5 for level i now holds deterministically conditioned on property 3 holding for i and all $j < i$.

Now for property 4, we can take $s' = \Theta(\gamma r^3/\epsilon^2)$ samples to build $\widetilde{W}(s_i)$. By Lemma 12.6.6, each run of the algorithm requires $O(\gamma r^{11}\log(N/\epsilon)/\epsilon^2)$ oracle calls, and fails to return a sample with probability at most $3/4$. Applying Hoeffding's inequality on the required number of trials of the sampling algorithm to obtain s' independent samples, this requires $O(\gamma^2 r^{15}\log(N/\epsilon)/\epsilon^4)$ oracle calls with probability $1 - 2^{-100\gamma}$. Given this, we have that each sample in $\widetilde{W}(s_i)$ is a $(1 \pm \epsilon/(3r^2))$-relative error almost uniform sample. Applying Hoeffding's inequality again, it follows that for a fixed set $L \subset \{s_0, \ldots, s_{i-1}\}$, we have

$$\left| \frac{|\widetilde{W}(s_i) \setminus \left(\cup_{s_j \in L} W(s_j)\right)|}{|\widetilde{W}(s_i)|} - \frac{|W(s_i) \setminus \left(\cup_{s_j \in L} W(s_j)\right)|}{|W(s_i)|} \right| \leq \frac{\epsilon}{3r^2} + \frac{\epsilon}{2r} \leq \frac{\epsilon}{r}$$

with probability $1 - 2^{-100\gamma r}$, and since there are only at most 2^r such subsets L, by a union bound this holds for all such subsets with probability $1 - 2^{-100\gamma r - r}$. Thus the overall probability of success is $1 - 2^{-100\gamma r - r} - 2^{-\gamma r} > 1 - 2^{-10\gamma}$. Note that the runtime is dominated by the time required to obtain Property 3 via Lemma 12.6.5, which is $O(T^{\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4}})$. □

We are now ready the prove the main theorem.

Proof of Theorem 149. By Lemma 12.6.8, conditioned on having Properties $3, 4$, and 5 for a level i, we get it for $i+1$ with probability $1 - 2^{-10\gamma}$ with at most $O(\gamma^2 \log(N/\epsilon) r^{17}/\epsilon^4)$ oracle calls. It follows inductively that with probability $1 - r2^{-10\gamma}$, we have Property 3 and 5 for all levels with at most $O(\gamma^2 \log(N/\epsilon) r^{18}/\epsilon^4)$ oracle calls, which completes the proof after recalling that $\gamma := \log(1/\delta)$. The runtime for each level is $O(T^{\frac{\log(N/\epsilon)\gamma^2 r^{17}}{\epsilon^4}})$ by Lemma 12.6.8, thus the total runtime is $O(T^{\frac{\log(N/\epsilon)\gamma^2 r^{18}}{\epsilon^4}})$. □

12.7 Applications of the FPRAS

12.7.1 Constraint satisfaction problems

Constraint satisfaction problems offer a general and natural setting to represent a large number of problems where solutions must satisfy some constraints, and which can be found in different areas such as artificial intelligence, satisfiability, programming languages, temporal reasoning,

scheduling, graph theory, and databases [Var00, CKS01, RVBW06, HN04, BHvMW09, RN16]. Formally, a constraint satisfaction problem (CSP) is a triple $\mathcal{P} = (V, D, C)$ such that $V = \{x_1, \ldots, x_m\}$ is a set of variables, D is a set of values and $C = \{C_1, \ldots, C_n\}$ is a set of constraints, where each constraint C_i is a pair $(\bar{t}_i, R_i)$ such that $\bar{t}_i$ is a tuple of variables from V of arity k, for some $k \geq 1$, and $R_i \subseteq D^k$. Moreover, an assignment $\nu : V \to D$ is said to be a solution for $\mathcal{P}$ if for every $i \in [n]$, it holds that $\nu(\bar{t}_i) \in R_i$ [RN16], where $\nu(\bar{t}_i)$ is obtained by replacing each variable x_j occurring in $\bar{t}_i$ by $\nu(x_j)$. The set of solutions for CSP $\mathcal{P}$ is denoted by $\text{sol}(\mathcal{P})$.

The two most basic tasks associated to a CSP are the evaluation and the satisfiability problems. In the evaluation problem, we are given a CSP $\mathcal{P}$ and an assignment ν, and the question to answer is whether $\nu \in \text{sol}(\mathcal{P})$. In the satisfiability problem, we are given a CSP $\mathcal{P}$, and the question to answer is whether $\text{sol}(\mathcal{P}) \neq \emptyset$. Clearly, these two problems have very different complexities, as in the former we only need to verify the simple condition that $\nu(t) \in R$ for every constraint (t, R) in $\mathcal{P}$, while in the latter we need to search in the space of all possible assignments for one that satisfies all the constraints. In fact, these two problems also look different in terms of our interest in the specific values for the variables of the CSP; in the former we are interested in the value of each one of them that is given in the assignment ν, while in the latter the variables of $\mathcal{P}$ are considered as existential quantifiers, as we are interested in knowing whether there exists a solution for $\mathcal{P}$ even if we do not know how to construct it. As a way to unify these two problems, and to indicate for which variables we are interested in their values, a projection operator has been used in the definition of CSPs [CJ06, Wil10]. Notice that the definition of this operator has also played an important role when classifying the complexity of CSPs in terms of algebraic properties of relations [CJ06]. Formally, an *existential* CSP (ECSP) is defined as a pair $\mathcal{E} = (U, \mathcal{P})$, where $\mathcal{P} = (V, D, C)$ is a CSP and $U \subseteq V$. Moreover, the set of solution for $\mathcal{E}$ is defined as

$$\text{sol}(\mathcal{E}) \quad = \quad \{\nu|_U \mid \nu \in \text{sol}(\mathcal{P})\},$$

where $\nu|_U$ is the restriction of function ν to the domain U. Notice that both the evaluation and the satisfiability problems for a CSP $\mathcal{P}$ can be reduced to the evaluation problem for an ECSP. In fact, the satisfiability problem for $\mathcal{P}$ corresponds to the problem of verifying whether the assignment with empty domain belongs to $\text{sol}(\mathcal{E})$, where $\mathcal{E}$ is the ECSP $(\emptyset, \mathcal{P})$. Moreover, the evaluation and satisfiability problems are polynomially interreducible for ECSPs, so ECSPs provide a uniform framework for these two problems allowing us to focus only on the evaluation problem.

Clearly the satisfiability problem for CSPs, as well as the evaluation problem for ECSPs,

is NP-complete; in particular, NP-hardness is a consequence that the satisfiability of 3-CNF propositional formulae can be easily encoded as a constraint satisfaction problem. Thus, a large body of research has been devoted to understanding the complexity of the evaluation problem for ECSPs, and finding tractable cases. In particular, two prominent approaches in this investigation have been based on the idea of viewing an ECSP as a homomorphism problem where the target structure is fixed [FV98, Bul17, Zhu17] or on the use of decomposition methods that require of some acyclicity conditions on an ECSP to be satisfied [GLS00, GLS02]. In this section, we focus on the latter class of methods, and show how the main results of this article can be used to deal with the fundamental problem of counting the number of solutions to an ECSP.

The evaluation problem for ECSPs is equivalent to the evaluation problem for CQs [KV00]. To see why this is the case, take an existential CSP $\mathcal{E} = (U, \mathcal{P})$, where $\mathcal{P} = (V, D, C)$ is a CSP with $U = \{y_1, \ldots, y_m\} \subseteq V$ and $C = \{C_1, \ldots, C_n\}$ where each constraint C_i is a pair $(\bar{t}_i, R_i)$ such that $\bar{t}_i$ is a tuple of variables from V of arity k, for some $k \geq 1$, and $R_i \subseteq D^k$. For each R_i, let $\bar{R}_i$ be a k-ary relational symbol. Then define the CQ:

$$Q_{\mathcal{E}}(\bar{y}) \quad \leftarrow \quad \bar{R}_1(\bar{t}_i), \ldots, \bar{R}_n(\bar{t}_n), \tag{12.19}$$

with $\bar{y} = (y_1, \ldots, y_m)$ and the database $D_{\mathcal{E}}$ such that $\bar{R}_i(\bar{a}) \in D_{\mathcal{E}}$ if, and only if, $\bar{a} \in R_i$ for each $i \leq n$. Then it is easy to see that for every assignment ν it holds that:

$$\nu \in \text{sol}(\mathcal{E}) \quad \text{ if and only if } \quad \nu(\bar{y}) \in Q_{\mathcal{E}}(D_{\mathcal{E}})$$

This tight connection can be used to extend the notions of acyclicity given in Section 12.3 to the case of ECSPs. More precisely, $\mathcal{E}$ is said to be acyclic if and only if $Q_{\mathcal{E}}$ is acyclic [Yan81, GLS98], and $\text{hw}(\mathcal{E})$ is defined as $\text{hw}(Q_{\mathcal{E}})$ [GLS02].

The notion of acyclic CSP coincides with the notion of α-acyclicity for hypergraphs [Fag83, BFMY83], and it has played an important role in finding tractable cases for ECSPs [GLS00]. In fact, if $\text{AECSP} = \{(\mathcal{E}, \nu) \mid \mathcal{E} \text{ is an acyclic ECSP and } \nu \in \text{sol}(\mathcal{E})\}$, then it holds that AECSP is LogCFL-complete under many-to-one logspace reductions [GLS98]. Recall that LogCFL consists of all decision problems that are logspace reducible to a context-free language, and it holds that $\text{NL} \subseteq \text{LogCFL} \subseteq \text{AC}^1$. Thus, we have that all problems in LogCFL can be solved in polynomial time and are highly parallelizable.

Concerning to our investigation, we are interested in the fundamental problem of counting the number of solution of a ECSP. In general, this problem is #P-complete and cannot admit an FPRAS (unless NP = RP, given that the evaluation problem for CSP is NP-complete). Thus, we

focus on the following fundamental problems where the degree of cyclicity of ECSP is bounded.

Problem:	#AECSP
Input:	An acyclic ECSP $\mathcal{E}$
Output:	$\lvert\text{sol}(\mathcal{E})\rvert$

Problem:	$\#k$-HW-ECSP
Input:	An ECSP $\mathcal{E}$ such that $\text{hw}(\mathcal{E}) \leq k$
Output:	$\lvert\text{sol}(\mathcal{E})\rvert$

From the characterization of the evaluation problem for ECSPs in terms of the evaluation problem of CQs and Theorem 144, we conclude that these problems admit FPRAS.

Theorem 150. #AECSP *and* $\#k$-HW-ECSP *admit an FPRAS for every* $k \geq 1$.

12.7.2 Nested words

Nested words have been proposed as a model for the formal verification of correctness of structured programs that can contain nested calls to subroutines [AEM04, AM04, AM09]. In particular, the execution of a program is viewed as a linear sequence of states, but where a matching relation is used to specify the correspondence between each point during the execution at which a procedure is called with the point when we return from that procedure call.

Formally, a binary relation μ on an interval $[n]$ is a matching if the following conditions hold: (a) if $\mu(i,j)$ holds then $i < j$; (b) if $\mu(i,j)$ and $\mu(i,j')$ hold then $j = j'$, and if $\mu(i,j)$ and $\mu(i',j)$ hold then $i = i'$; (c) if $\mu(i,j)$ and $\mu(i',j')$ hold, where $i \neq i'$ and $j \neq j'$, then either $[i,j] \cap [i',j'] = \emptyset$ or $[i,j] \subseteq [i',j']$ or $[i',j'] \subseteq [i,j]$. Moreover, given a finite alphabet Σ, a *nested word* of length n over Σ is a tuple $\bar{w} = (w, \mu)$, where $w \in \Sigma^*$ is a string of length n, and μ is a matching on $[n]$.

A position i in a nested word $\bar{w}$ is a call (resp., return) position if there exists j such that $\mu(i,j)$ (resp., $\mu(j,i)$) holds. If i is neither a call nor a return position in $\bar{w}$, then i is said to be an internal position in $\bar{w}$. Figure 12.6 shows a nested word (without the labeling with alphabet symbols). Solid lines are used to draw the linear edges that define a standard word, while nesting edges are drawn using dashed lines. Thus, the relation μ is $\{(2,4),(5,6),(1,7)\}$, the set of call positions is $\{1,2,5\}$, the set of return positions is $\{4,6,7\}$ and the set of internal positions is $\{3,8\}$.

Properties to be formally verified are specified by using nested word automata. Such automata have the same expressiveness as monadic second order logic over nested words [AM09], so they are expressive enough to allow the specification and automatic verification of a large variety of properties over programs with nested calls to subroutines. Formally, a (nondeterministic) *nested word automaton* (NWA) $\mathcal{N}$ is a tuple $(S, \Sigma, S_0, F, P, \Delta_C, \Delta_I, \Delta_R)$ consisting of a finite set of states S, an alphabet Σ, a set of initial states $S_0 \subseteq S$, a set of final states $F \subseteq S$, a finite set of hierarchical symbols P, a call-transition relation $\Delta_C \subseteq S \times \Sigma \times S \times P$, an internal-transition relation $\Delta_I \subseteq S \times \Sigma \times S$, and a return-transition relation $\Delta_R \subseteq S \times P \times \Sigma \times S$.

An NWA $\mathcal{N} = (S, \Sigma, S_0, F, P, \Delta_C, \Delta_I, \Delta_R)$ works as follows with input a nested word $\bar{w}$. $\mathcal{N}$ starts in an initial state in S_0 and reads $\bar{w}$ from left to right. The state is propagated along the linear edges of $\bar{w}$ as in case of a standard word automaton. However, at a call position in $\bar{w}$, the nested word automaton propagates a state along the linear edge together with a hierarchical symbol along the nesting edge of $\bar{w}$. At a return position in $\bar{w}$, the new state is determined based on the state propagated along the linear edge as well as the symbol along the incoming nesting edge. Formally, a run ρ of the automaton $\mathcal{N}$ over a nested word $\bar{w} = (a_1 \cdots a_n, \mu)$ is a sequence $s_0, s_1, \ldots, s_n$ of states along the linear edges, and a sequence p_i, for every call position i, of hierarchical symbols along the nesting edges, such that: (a) $s_0 \in S_0$; (b) for each call position i, it holds that $(s_{i-1}, a_i, s_i, p_i) \in \Delta_C$; (c) for each internal position i, it holds that $(s_{i-1}, a_i, s_i) \in \Delta_I$; and (d) for each return position i such that $\mu(j, i)$ holds, we have that $(s_{i-1}, p_j, a_i, s_i) \in \Delta_R$. Moreover, the run ρ is accepting if $s_n \in F$, and

$$\mathcal{L}(\mathcal{N}) \;=\; \{\bar{w} \mid \bar{w} \text{ is a nested word over } \Sigma^* \text{ and there exists an accepting run of } \mathcal{N} \text{ with input } \bar{w}\}.$$

The emptiness problem for nested word automata ask whether, given a NWA $\mathcal{N}$, there exists a nested word $\bar{w}$ accepted by $\mathcal{N}$. This is a fundamental problem when looking for faulty executions of a program with nested calls to subroutines; if $\mathcal{N}$ is used to encode the complement of a property we expect to be satisfied by a program, then a nested word $\bar{w} \in \mathcal{L}(\mathcal{N})$ encodes a bug of this program. In this sense, the following is also a very relevant problem for understanding how faulty a program is:

Problem:	#NWA
Input:	A nested word automaton $\mathcal{N}$ and a string 0^n
Output:	$\lvert\{\bar{w} \in \mathcal{L}(\mathcal{N}) \mid \lvert\bar{w}\rvert = n\}\rvert$

As there exists a trivial polynomial-time parsimonious reduction from #NFA to #NWA, we have that #NWA is #P-complete. Interestingly, from the existence of an FPRAS for #BTA (see Corol-

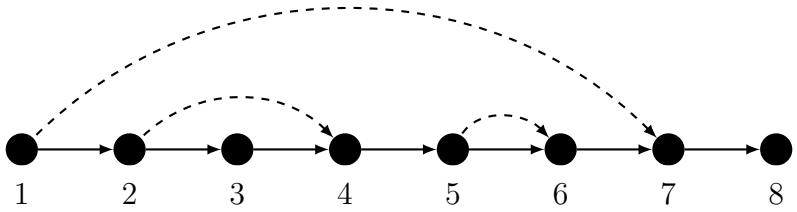


Figure 12.6: A nested word

lary 12.5.5) and the results in [AM09] showing how nested word automata can be represented by using tree automata over binary trees, it is possible to prove that:

Theorem 151. #NWA *admits an FPRAS.*

12.7.3 Knowledge compilation

Model counting is the problem of counting the number of satisfying assignments given a propositional formula. Although this problem is #P-complete [Val79], there have been several approaches to tackle it [GSS09]. One of them comes from the field of *knowledge compilation*, a subarea in artificial intelligence [DM02]. Roughly speaking, this approach consists in dividing the reasoning process in two phases. The first phase is to compile the formula into a target language (e.g. Horn formulae, BDDs, circuits) that has good algorithmic properties. The second phase is to use the new representation to solve the problem efficiently. The main goal then is to find a target language that is expressive enough to encode a rich set of propositional formulae and, at the same time, that allows for efficient algorithms to solve the counting problem.

One of the most used formalism in knowledge compilation are circuits in Negation Normal Form (NNF for short). An NNF circuit $C = (V, E, g_0, \mu)$ is a directed acyclic graph (V, E) where V are called gates, edges E are called wires, and $g_0 \in V$ is a distinguished gate called the output gate. The function μ assigns a type to each gate that can be $\wedge$ (AND), $\vee$ (OR), or a literal (i.e. a variable or the negation of a variable). We assume that all literals have in-degree 0 and we call them input gates. Without loss of generality, we assume that all $\wedge$-gate and $\vee$-gate have in-degree two (if not, we can convert any NNF circuit in poly-time to binary gates). For a gate g we define the set $\mathrm{Vars}(g)$ of all variables whose value can alter the value of g, formally, $v \in \mathrm{Vars}(g)$ if and only if there exists an input gate g' with variable v (i.e. $\mu(g) = v$ or $\mu(g) = \bar{v}$) and there is a path from g' to g in (V, E). A valuation for C is a mapping ν from the variables

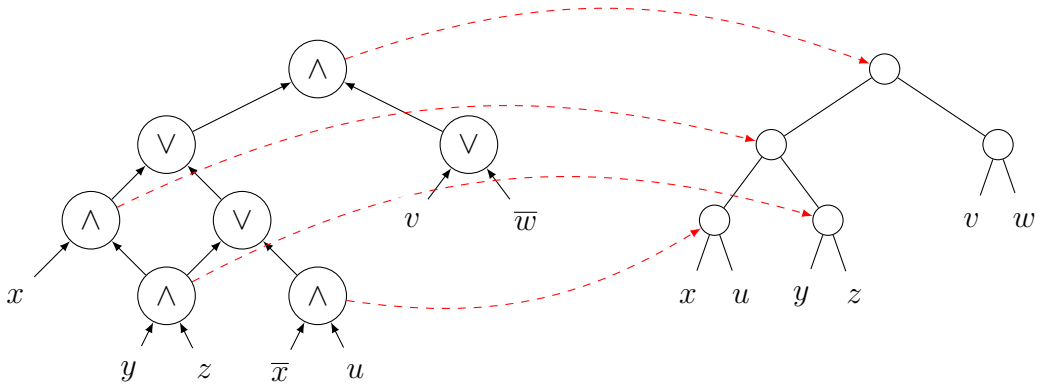


Figure 12.7: A structured DNNF circuit and its corresponding v-tree.

of C to $\{0, 1\}$. The valuation of C with ν, denoted by $\nu(C)$, is the value (i.e. 0 or 1) taken by g_0 when C is evaluated in a bottom up fashion.

A target language for knowledge compilation that has attracted a lot of attention is the class of DNNF circuits. An NNF circuit C is called decomposable [Dar01a] if and only if for every $\wedge$-gate g with incident gates g_1, g_2 it holds that $\mathrm{Vars}(g_1) \cap \mathrm{Vars}(g_2) = \emptyset$. In other words, if the incident gates of every $\wedge$-gate share no variables. For example, one can easily check that the NNF circuit of Figure 12.7 is decomposable. DNNF is the set of all NNF circuits that are decomposable. DNNF has good algorithmic properties in terms of satisfiability and logical operations. Furthermore, DNNF can be seen as a generalization of DNF formulae and, in particular, of binary decision diagrams (BDD), in the sense that every BDD can be transformed into a DNNF circuit in polynomial time. Nevertheless, DNNF is exponentially more succint than DNF or BDD, and then it is a more appealing language for knowledge compilation.

Regarding model counting, DNNF circuits can easily encode #P-complete problems (e.g. #DNF) and, therefore, researchers have look into subclasses of DNNF with efficient counting properties. Deterministic DNNF (d-DNNF for short) is a subclass of DNNF where the counting problem can be solved in polynomial time (see [Dar01b] for a definition of d-DNNF). Indeed, several problems can be compiled into d-DNNF circuits, finding applications in probabilistic reasoning [CDJ06], query evaluation [BLRS17], planning [BG06], among others.

However, as pointed out in [PD08] the compilation into d-DNNF circuits usually satisfies a structural property between variables, which naturally brings the class of structured DNNF circuits. A v-tree is a binary tree t whose leaves are in one-to-one correspondence with a set of variables. Similar than for circuits, for a node u in a v-tree, we denote by $\mathrm{Vars}(u)$ the set of all variables in the leaves of the subtree rooted at u. Then we say that a DNNF circuit C respects a

v-tree t if for every $\wedge$-gate g and the two incident gates g_1 and g_2 of g, there exists a node u in t such that $\mathrm{Vars}(g_1) \subseteq \mathrm{Vars}(u_1)$ and $\mathrm{Vars}(g_2) \subseteq \mathrm{Vars}(u_2)$, where u_1 and u_2 are the left and right child of u in t, respectively. We say that a DNNF circuit C is structured if and only if there exists a v-tree t such that C respects t. For example, in the right-hand side of Figure 12.7, we show a v-tree for variables $\{x, y, z, u, v, w\}$. The red dashed lines show how $\wedge$-gates have to be assigned to the nodes in the v-tree in order for the circuit to respect this v-tree. Structured DNNF is the class of all DNNF circuits that are structured. As it was already mentioned, the compilation into d-DNNF circuits usually produces circuits that are also structured [PD08]. Structured DNNF have been recently used for efficient enumeration [ABJM17, ABMN19] and in [OD14] it was shown that CNF formulae with bounded width (e.g. CV-width) can be efficiently compiled into structured DNNF circuits. Since structured DNNF circuits includes the class of DNF formulae, its underlying counting problem is #P-complete. We now prove a positive approximation result. Specifically, consider the problem:

Problem:	#StructuredDNNF
Input:	A DNNF circuit C and a v-tree t such that C respects t.
Output:	$\lvert\{\nu \mid \nu(C) = 1\}\rvert$

By using the existence of an FPRAS for #TA, we can show that #StructuredDNNF also admits an FPRAS.

Theorem 152. #StructuredDNNF *admits an FPRAS.*

Proof. The connection between structured DNNF and tree automata was already used in [ABJM17, ABMN19], so this connection is not new. Here, we show that there exists a parsimonious reduction from #StructuredDNNF into #TA, which proves the FPRAS for structured DNNF.

Let t be a v-tree and $C = (V, E, g_0, \mu)$ be a DNNF circuit such that C respects t. Let $V_\wedge$ be all gates in V that are $\wedge$-gates or input gates. Furthermore, let $f : V_\wedge \to t$ be the function that realizes that C respects t, namely, for every $\wedge$-gate g and gates g_1 and g_2 incident to g it holds that $\mathrm{Vars}(g_1) \subseteq \mathrm{Vars}(f(g) \cdot 1)$ and $\mathrm{Vars}(g_2) \subseteq \mathrm{Vars}(f(g) \cdot 2)$. We also assume that if $g \in V_\wedge$ is a literal, then $f(g)$ is the leaf in t that has the same variable as g in C. One can easily check that for two gates $g_1, g_2 \in V_\wedge$ in C, if there is a path from g_2 to g_1 in C, then $f(g_2)$ is a descendant of $f(g_1)$ in t. Without loss of generality, we assume that g_0 is a $\wedge$-gate in C and $f(g_0) = \epsilon$. Finally, for any $\wedge$-gate g we define the set $D(g)$ of all gates $g' \in V_\wedge$ such that there exists a path from g' to g in (V, E) passing only through $\vee$-gates. Intuitively, if $g' \in D(g)$ then g' is directly affecting the value of g in the sense that if g' is true, then at least one of the incident wires to g is true.

The idea for the parsimonious reduction is to construct a tree automaton $\mathcal{T}_C$ that will accept trees that encode valuations that makes C true. To encode a valuation, $\mathcal{T}_C$ will only accept binary trees having exactly the same tree-shape as t, but its leaves are labeled with 0 or 1 (internal node will have any symbol, e.g. $@$). Given that leaves of t are in one-to-one correspondence with the variables in C, then these encodings are in one-to-one correspondence with valuations in C. So, for a valuation ν let t_ν be the tree that has the same tree-shape as t and whose leaves encode ν. Furthermore, suppose that t_ν is an input tree for the tree automaton $\mathcal{T}_C$. For checking that $\nu(C) = 1$, the states of $\mathcal{T}_C$ will be either nodes $u \in t$ or pairs of the form (u, g) where $u \in t$, $g \in V_\wedge$, and $f(g)$ is a descendant of u in t. A node u in a state (e.g. in the pair (u, g)) will take care of checking that the tree-shape of t_ν is the same as t. On the other hand, the gate g in the pair (u, g) will be use to navigate C and find whether g is evaluated to 1 given the valuation encoded by t_ν. When $f(g)$ is a strict descendant of u (i.e. $f(g) \neq u$) we will continue down t_ν trying to find a node u' such that $f(g) = u'$. When a node u with $f(g) = u$ is found, then to evaluate g to 1 we need to find two gates $g_1, g_2 \in D(g)$ that are also evaluated to 1 with ν. Given that C respects t we know that $f(g_1)$ and $f(g_2)$ must be descendants of $f(g)$ in t_ν, and then $\mathcal{T}_C$ will recurse into the states $(u1, g_1)$ and $(u2, g_2)$ continuing into g_1 and g_2. If the non-deterministic decisions of $\mathcal{T}_C$ are taken correctly, $\mathcal{T}_C$ will reach the leaves u of t_ν that has the same variable as the gate g and it will check if the value $t_\nu(u)$ is correct with respect to g.

Let $\mathcal{T}_C = (Q, \Sigma, \Delta, s_{\text{init}})$ be the tree automaton constructed from C and t such that $\Sigma = \{@, 0, 1\}$, $Q = t \cup \{(u, g) \in t \times V_\wedge \mid f(g) \text{ is a descendant of } u \text{ in } t\}$, and $s_{\text{init}} = (\lambda, g_0)$ (recall that we assume that g_0 is a $\wedge$-gate and $f(g_0) = \lambda$). We define the transition relation Δ by case analysis:

- For $u \in t$ and u is not a leaf, then $(u, @, u1 \cdot u2) \in \Delta$.
- For $u \in t$ and u is a leaf, then $(u, a, \lambda) \in \Delta$ for every $a \in \{0, 1\}$.
- For $(u, g) \in Q$ such that $f(g) \neq u$, if $f(g)$ is a descendant of $u1$, then $((u, g), @, (u1, g) \cdot u2) \in \Delta$. Otherwise, if $f(g)$ is a descendant of $u2$, then $((u, g), @, u1 \cdot (u2, g)) \in \Delta$.
- For $(u, g) \in Q$ such that $f(g) = u$ and u is not a leaf in t, then $((u, g), @, (u1, g_1) \cdot (u2, g_2)) \in \Delta$ for every $g_1, g_2 \in D(g)$ with $f(g_1)$ is a descendant of $u1$ and $f(g_2)$ is a descendant of $u2$.
- For $(u, g) \in Q$ such that $f(g) = u$ and u is a leaf in t, then $((u, g), 1, \lambda) \in \Delta$ iff $\mu(g) = t(u)$, that is, if g is a positive literal and its variable coincide with the variable assign to u in t. Similarly, $((u, g), 0, \lambda) \in \Delta$ iff $\mu(g) = \neg t(u)$.

Finally, given a circuit C the reduction produces the tree automaton $\mathcal{T}_C$ and the value $0^{|t|}$. From the construction, it is straightforward to check that $\mathcal{T}_C$ will accept trees that have the same tree-shape as t and whose leaves encode a valuation of C. Furthermore, for a valuation ν and its tree t_ν one can check that $\nu(C) = 1$ if, and only if, $t_\nu \in \mathcal{L}(\mathcal{T}_C)$. Therefore, the reduction is parsimonious. Finally, the number of states and transitions of $\mathcal{T}_C$ is polynomial in the size of C and t, and thus the reduction can be computed in polynomial time. □

12.8 Open Problems and Future Work

In this chapter, we proved the existence of a fully polynomial-time randomized approximation scheme (FPRAS) for counting solutions to a large class of conjunctive queries, as well as the existence of a fully polynomial-time almost uniform sampler (FPAUS) for these solutions. In particular, our algorithm applies to all conjunctive queries with bounded hypertree width (as defined in Section 12.3.2).

In Section 12.3.3, and based on Theorem 144 and the results in [GSS01], we provide a characterization of the classes of conjunctive queries that admit an FPRAS, when such classes are of the form CQ($\mathcal{G}$) for a family $\mathcal{G}$ of graphs (see Corollary 12.3.1). It remains as an open problem how to extend this characterization to any class of conjunctive queries and, in particular, to identify the right restriction on conjunctive queries that is equivalent to the existence of an FPRAS. It should be mentioned that recently, in a follow-up work by Focke, Goldberg, Roth and Živnỳ [FGRŽ21], it was proved that our results can be extended to the case of conjunctive queries with bounded *fractional hypertree width*.

An important component of the line of work developed in our papers [ACJR19, ACJR21] is the study of approximate counting problems in automata theory. As demonstrating in this chapter, finite automata have strong expressive potential to model the solution space of many problems in computer science. Thus, resolving the counting problem for a class of automata can result in an FPRAS for many additional problems. So far, we know that the counting problem for non-deterministic finite automata (NFA), and now tree automata, admit polynomial time randomized approximations. However, there are still many classes of automata for which it is unknown if an FPRAS exists.

An additional natural case to consider is the class of *context free grammars* (CFG), which generate the class of context free languages (CFL). Context free grammars are more expressive than tree-automata, although they are closely related (for instance, the set of all derivation trees

of a CFG can be expressed by a tree automata). There are several barriers to generalizing the approach in this chapter to the case of context free grammars. Mainly, in order to carry out our high-level sampling template described in Section 12.4, we need to partition the current set T into a collection of disjoint subsets $T'_1, \ldots, T'_\ell$. We accomplish this for the case of tree automata by splitting based on the final sizes of the left and right subtrees. However, such a partition does not work for the case of CFG's, since a single word in a CFL can have multiple derivations whose subtrees have different sizes, thus this word would be contained in multiple such sets T'_i causing the scheme to fail to be a partition. Thus, the key barrier to extending our algorithmic framework to the case of CFG's is the design of an efficient partition scheme which avoids this issue.

Bibliography

[ABC09a] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. Functional monitoring without monotonicity. In *International Colloquium on Automata, Languages, and Programming*, pages 95–106. Springer, 2009.

[ABC09b] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. Functional monitoring without monotonicity. In *ICALP*, pages 95–106, 2009.

[ABFS02] Miklos Ajtai, Randal Chilton Burns, Ronald Fagin, and Larry Joseph Stockmeyer. System and method for differential compression of data from a plurality of binary sources, April 16 2002. US Patent 6,374,250.

[Abi95] Serge Abiteboul. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[ABIW09] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David Woodruff. Efficient sketches for earth-mover distance, with applications. 2009.

[ABJM17] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 111:1–111:15, 2017.

[ABMN19] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019*, pages 89–103, 2019.

[ABRS] Aditya Akella, Ashwin Bharambe, Mike Reiter, and Srinivasan Seshan. Detecting ddos attacks on isp networks.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 214–223, 2017.

[ACJR19] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, pages 59–73, New York, NY, USA, 2019. ACM.

[ACJR21] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. When is approximate counting for conjunctive queries tractable? *Proceedings of the Fifty-third Annual ACM Symposium on Theory of Computing (STOC))*, 2021.

[ACK+16] Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 311–319, 2016.

[ACL07] Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007.

[AD20] Serge Abiteboul and Gilles Dowek. *The Age of Algorithms*. Cambridge University Press, 2020.

[AD21] Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 172–180. SIAM, 2021.

[AEM04] Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Proceedings*, pages 467–481, 2004.

[AGM12a] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 459–467. Society for Industrial and Applied Mathematics, 2012.

[AGM12b] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 459–467, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics.

[AGM12c] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: spar-

sification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 5–14, 2012.

[AGMS99] Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 10–20. ACM, 1999.

[AGPR99] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 339–348. IEEE, 2005.

[AHLW16] Yuqing Ai, Wei Hu, Yi Li, and David P Woodruff. New characterizations in turnstile streams with applications. In *31st Conference on Computational Complexity (CCC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[AIK08a] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 343–352, 2008.

[AIK08b] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. pages 343–352, 2008.

[ÀJ93] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993.

[AJ06] José A Adell and Pedro Jodrá. Exact kolmogorov and total variation distances between some familiar discrete distributions. *Journal of Inequalities and Applications*, 2006(1):64307, 2006.

[AK19] Thomas D Ahle and Jakob BT Knudsen. Almost optimal tensor sketch. *arXiv preprint arXiv:1909.01821*, 2019.

[AKK+20a] Thomas D. Ahle, Michael Kapralov, Jakob B. T. Knudsen, Rasmus Pagh, Ameya Velingker, David P. Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *SODA*. Merger version of https://arxiv.org/pdf/1909.01410.pdf and https://arxiv.org/pdf/1909.01821.pdf, 2020.

[AKK+20b] Thomas D Ahle, Michael Kapralov, Jakob BT Knudsen, Rasmus Pagh, Ameya Velingker, David P Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 141–160. SIAM, 2020.

[AKNN+18] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, SIGMOD/PODS '18, pages 325–340, New York, NY, USA, 2018. ACM.

[AKNR16] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28. ACM, 2016.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 377–386. IEEE, 2010.

[AKO11] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 363–372. IEEE, 2011.

[AKPS19] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for ℓ_p-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1405–1424. SIAM, 2019.

[AM00] Srinivas M Aji and Robert J McEliece. The generalized distributive law. *IEEE transactions on Information Theory*, 46(2):325–343, 2000.

[AM04] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211, 2004.

[AM09] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[AMP+13] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Mad-

den, and Ion Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *ACM EuroSys*, pages 29–42, 2013.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[ANOY14] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. 2014.

[ANW14] Haim Avron, Huy Nguyen, and David Woodruff. Subspace embeddings for the polynomial kernel. In *Advances in neural information processing systems*, pages 2258–2266, 2014.

[AP09] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 235–244, 2009.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):1–37, 2009.

[AS14] Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. pages 555–564, 2014.

[AWR17] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. 2017.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. 1996.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. 1998.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[BBG18] Siddharth Barman, Arnab Bhattacharyya, and Suprovat Ghoshal. Testing sparsity over known and unknown bases. In *International Conference on Machine Learning*, pages 491–500, 2018.

[BCI+16] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. Bptree: an l2 heavy hitters algorithm using constant memory. *arXiv preprint arXiv:1603.00759*, 2016.

[BCI+17] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P. Woodruff. Bptree: An ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 361–376. ACM, 2017.

[BCJ20] Ainesh Bakshi, Nadiia Chepurko, and Rajesh Jayaram. Testing positive semi-definiteness via random submatrices. In *61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020.

[BCK+14] Joshua Brody, Amit Chakrabarti, Ranganath Kondapally, David P Woodruff, and Grigory Yaroslavtsev. Certifying equality with limited interaction. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[BCLL18] Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, and Yuanzhi Li. An homotopy method for ℓ_p regression provably beyond self-concordance and in input-sparsity time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1130–1137. ACM, 2018.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BCW20] Ainesh Bakshi, Nadiia Chepurko, and David P Woodruff. Robust and sample optimal algorithms for psd low-rank approximation. In *61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020.

[BCWY16] Vladimir Braverman, Stephen R Chestnut, David P Woodruff, and Lin F Yang. Streaming space complexity of nearly all functions of one variable on frequency vectors. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 261–276, 2016.

[BDI+20] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. 2020.

[BDM02] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM*

symposium on Discrete algorithms, pages 633–634. Society for Industrial and Applied Mathematics, 2002.

[BDN17] Jarosław Błasiok, Jian Ding, and Jelani Nelson. Continuous monitoring of lp norms in data streams. *arXiv preprint arXiv:1704.06710*, 2017.

[BDW16] Arnab Bhattacharyya, Palash Dey, and David P Woodruff. An optimal algorithm for l1-heavy hitters in insertion streams and related problems. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 385–400. ACM, 2016.

[BEO+13] Mark Braverman, Faith Ellen, Rotem Oshman, Toniann Pitassi, and Vinod Vaikuntanathan. A tight bound for set disjointness in the message-passing model. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 668–677. IEEE, 2013.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[BG06] Blai Bonet and Hector Geffner. Heuristics for planning with penalties and rewards using compiled knowledge. In *KR*, pages 452–462, 2006.

[BGS00] Alberto Bertoni, Massimiliano Goldwurm, and Massimo Santini. Random generation and approximate counting of ambiguously described combinatorial structures. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 567–580. Springer, 2000.

[BGW20] Mark Braverman, Sumegha Garg, and David Woodruff. The coin problem with applications to data streams. In *2020 IEEE 61th Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BI14] Arturs Bačkurs and Piotr Indyk. Better embeddings for planar earth-mover distance over sparse sets. 2014.

[BICS09] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. 2009.

[Big] https://cloud.google.com/bigquery-ml/docs/bigqueryml-intro.

[BIPW10] Khanh Do Ba, Piotr Indyk, Eric Price, and David P Woodruff. Lower bounds for

sparse recovery. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1190–1197. SIAM, 2010.

[BJS15] Srinadh Bhojanapalli, Prateek Jain, and Sujay Sanghavi. Tighter low-rank approximation via sampling the leveraged element. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 902–920. Society for Industrial and Applied Mathematics, 2015.

[BJWY20] Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 63–80, 2020.

[BKP+14] Karl Bringmann, Fabian Kuhn, Konstantinos Panagiotou, Ueli Peter, and Henning Thomas. Internal dla: Efficient simulation of a physical growth model. In *International Colloquium on Automata, Languages, and Programming*, pages 247–258. Springer, 2014.

[BKSV14] Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o (nˆ(1-2/k)) bits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[Bła18] Jarosław Błasiok. Optimal streaming and tracking distinct elements with high probability. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2432–2448. SIAM, 2018.

[BLRS17] Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Exact model counting of query expressions: Limitations of propositional methods. *ACM Trans. Database Syst.*, 42(1):1:1–1:46, 2017.

[BLS+16] Maria Florina Balcan, Yingyu Liang, Le Song, David Woodruff, and Bo Xie. Communication efficient distributed kernel principal component analysis. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 725–734. ACM, 2016.

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. *arXiv preprint arXiv:2002.02304*, 2020.

[BLWZ19] Maria-Florina Balcan, Yi Li, David P Woodruff, and Hongyang Zhang. Test-

ing matrix rank, optimally. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 727–746. SIAM, 2019.

[BMRT20] Florent Becker, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. The impact of locality in the broadcast congested clique model. *SIAM Journal on Discrete Mathematics*, 34(1):682–700, 2020.

[BO03] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *ACM SIGMOD*, pages 28–39, 2003.

[BO10] Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *arXiv preprint arXiv:1011.2571*, 2010.

[Bou85] Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[BOV15] Vladimir Braverman, Rafail Ostrovsky, and Gregory Vorsanger. Weighted sampling without replacement from data streams. *IPL*, 115(12):923–926, 2015.

[BOZ12] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[Bul17] Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330, 2017.

[Bv20] Andrei A. Bulatov and Stanislav Živný. Approximate counting CSP seen from the other side. *ACM Trans. Comput. Theory*, 12(2), May 2020.

[BVDPPH11] Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using lagrangian mass transport. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–12, 2011.

[BVKS19] Jess Banks, Jorge Garza Vargas, Archit Kulkarni, and Nikhil Srivastava. Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time. *arXiv preprint arXiv:1912.08805*, 2019.

[BVWY18] Vladimir Braverman, Emanuele Viola, David Woodruff, and Lin F Yang. Revisiting frequency moment estimation in random order streams. *arXiv preprint arXiv:1803.02270*, 2018.

[BW18] Ainesh Bakshi and David Woodruff. Sublinear time low-rank approximation of distance matrices. In *Advances in Neural Information Processing Systems*, pages 3782–3792, 2018.

[BWZ16] Christos Boutsidis, David P Woodruff, and Peilin Zhong. Optimal principal component analysis in distributed and streaming models. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 236–249. ACM, 2016.

[BY20] Omri Ben-Eliezer and Eylon Yogev. The adversarial robustness of sampling. pages 49–62, 2020.

[BYJK+02] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[BYJKS04] Ziv Bar-Yossef, Thathachar S Jayram, Ravi Kumar, and D Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

[CBK11] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[CBW99] R. Calinger, J.E. Brown, and T.R. West. *A Contextual History of Mathematics: To Euler*. A Contextual History of Mathematics: To Euler. Prentice Hall, 1999.

[CC13] Peter Clifford and Ioana Cosma. A simple sketching algorithm for entropy estimation over streaming data. In *Artificial Intelligence and Statistics*, pages 196–206, 2013.

[CCD11] Edith Cohen, Graham Cormode, and Nick G. Duffield. Structure-aware sampling: Flexible and accurate summarization. *PVLDB*, 4(11):819–830, 2011.

[CCD12] Edith Cohen, Graham Cormode, and Nick Duffield. Don't let the negatives bring you down: sampling from streams of signed updates. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):343–354, 2012.

[CCFC02a] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, London, UK,

UK, 2002. Springer-Verlag.

[CCFC02b] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, languages and programming*, pages 784–784, 2002.

[CCG+98] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. 1998.

[CCM10] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Transactions on Algorithms (TALG)*, 6(3):1–21, 2010.

[CCM16] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. *Theory of Computing*, 12(1):1–35, 2016.

[CD21] Xue Chen and Michał Dereziński. Query complexity of least absolute deviation regression via robust uniform convergence. *arXiv preprint arXiv:2102.02322*, 2021.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://tata.gforge.inria.fr/, 2007. release October, 12th 2007.

[CDGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *IJCAI*, volume 99, pages 84–89, 1999.

[CDJ06] Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling relational bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1-2):4–20, 2006.

[CDK+09] Edith Cohen, Nick Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Stream sampling for variance-optimal estimation of subset sums. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1255–1264. Society for Industrial and Applied Mathematics, 2009.

[CDK+14] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Algorithms and estimators for summarization of unaggregated data streams. *J. Comput. Syst. Sci.*, 80(7):1214–1244, 2014.

[CDMI+13] Kenneth L Clarkson, Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, Xiangrui Meng, and David P Woodruff. The fast cauchy transform and faster robust linear regression. In *Proceedings of the Twenty-Fourth Annual*

ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 466–477. Society for Industrial and Applied Mathematics, https://arxiv.org/pdf/1207.4684, 2013.

[CEM+15] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 163–172. ACM, https://arxiv.org/pdf/1410.6801, 2015.

[CFCHT20] Yi-Jun Chang, Martín Farach-Colton, Tsan-Sheng Hsu, and Meng-Tsung Tsai. Streaming complexity of spanning tree computation. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[CG05] Graham Cormode and Minos Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24. VLDB Endowment, 2005.

[CGP20] Edith Cohen, Ofir Geri, and Rasmus Pagh. Composable sketches for functions of frequencies: Beyond the worst case. In *International Conference on Machine Learning*, pages 2057–2067. PMLR, 2020.

[CGSS20] Edith Cohen, Ofir Geri, Tamas Sarlos, and Uri Stemmer. Differentially private weighted sampling. *arXiv preprint arXiv:2010.13048*, 2020.

[Cha02] Moses Charikar. Similarity estimation techniques from rounding algorithms. pages 380–388, 2002.

[Chu96] Fan RK Chung. Lectures on spectral graph theory. *Lecture Notes*, 1996.

[CIKM02] Graham Cormode, Piotr Indyk, Nick Koudas, and S Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *Proceedings 18th International Conference on Data Engineering*, pages 605–614. IEEE, 2002.

[CJ06] David A. Cohen and Peter Jeavons. The complexity of constraint languages. In *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

[CJ19] Graham Cormode and Hossein Jowhari. L p samplers and their applications: A survey. *ACM Computing Surveys (CSUR)*, 52(1):1–31, 2019.

[CJK+04] Graham Cormode, Theodore Johnson, Flip Korn, Shan Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. Holistic udafs at streaming speeds. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of*

data, pages 35–46. ACM, 2004.

[CJLW20a] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. An improved analysis of the quadtree for high dimensional emd. 2020.

[CJLW20b] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. An improved analysis of the quadtree for high-dimensional emd. 2020.

[CJMM16] Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. *arXiv preprint arXiv:1608.03118*, 2016.

[CK16] Amit Chakrabarti and Sagar Kale. Strong fooling sets for multi-player communication with applications to deterministic estimation of stream statistics. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 41–50, 2016.

[CK19] Zhaoyue Cheng and Nick Koudas. Nonlinear models over normalized data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1574–1577. IEEE, 2019.

[CKP+21] Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh Saxena, Zhao Song, and Huacheng Yu. Near-optimal two-pass streaming algorithm for sampling random walks over directed graphs. *arXiv preprint arXiv:2102.11251*, 2021.

[CKS01] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*, volume 7 of *SIAM monographs on discrete mathematics and applications*. SIAM, 2001.

[CKS03] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th IEEE Annual Conference on Computational Complexity, 2003. Proceedings.*, pages 107–117. IEEE, 2003.

[Cla05] Kenneth L Clarkson. Subgradient and sampling algorithms for ℓ_1 regression. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 257–266, 2005.

[CLM+15] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 181–190, 2015.

[CLP18] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed (δ+ 1)-

coloring algorithm? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 445–456, 2018.

[CLS19] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pages 938–942, 2019.

[CM77a] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, 1977.

[CM77b] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, STOC'77, pages 77–90, 1977.

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[CM21] David P. Woodruff Cameron Musco, Christopher Musco. Active sampling for linear regression beyond the ℓ_2 norm. Manuscript, 2021.

[CMN99a] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. *ACM SIGMOD Record*, 28(2):263–274, 1999.

[CMN99b] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On random sampling over joins. In *SIGMOD*, pages 263–274, 1999.

[CMR05] Graham Cormode, S Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st international conference on Very large data bases*, pages 25–36. VLDB Endowment, 2005.

[CMS76] John M Chambers, Colin L Mallows, and BW Stuck. A method for simulating stable random variables. *Journal of the american statistical association*, 71(354):340–344, 1976.

[CMY11] Graham Cormode, S Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms (TALG)*, 7(2):21, 2011.

[CMYZ10] Graham Cormode, S Muthukrishnan, Ke Yi, and Qin Zhang. Optimal sampling from distributed streams. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 77–86. ACM, 2010.

[CMYZ12] Graham Cormode, S Muthukrishnan, Ke Yi, and Qin Zhang. Continuous sampling from distributed streams. *Journal of the ACM (JACM)*, 59(2):10, 2012.

[Coh13] Edith Cohen. Tau cs 0368.3239, leveraging big data lecture notes, Fall 2013.

[Coh15] Edith Cohen. Stream sampling for frequency cap statistics. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 159–168. ACM, 2015.

[COP03] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, 2003.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[CPW20] Edith Cohen, Rasmus Pagh, and David Woodruff. Wor and p's: Sketches for ell_p -sampling without replacement. *Advances in Neural Information Processing Systems*, 33, 2020.

[CR97] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, pages 56–70, 1997.

[CR12] Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012.

[CRR14] Arkadev Chattopadhyay, Jaikumar Radhakrishnan, and Atri Rudra. Topology matters in communication. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 631–640. IEEE, 2014.

[CSWZ16] Jiecao Chen, He Sun, David Woodruff, and Qin Zhang. Communication-optimal distributed clustering. In *Advances in Neural Information Processing Systems*, pages 3727–3735, 2016.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[CT15] Yung-Yu Chung and Srikanta Tirthapura. Distinct random sampling from a distributed stream. In *IPDPS*, pages 532–541, 2015.

[CTW16] Yung-Yu Chung, Srikanta Tirthapura, and David P Woodruff. A simple message-optimal algorithm for random sampling from a distributed stream. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1356–1368, 2016.

[Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. 2013.

[CW79] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[CW09] Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 205–214, 2009.

[CW17] Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):54, 2017.

[CY20] Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *ICDT*, pages 7:1–7:18, 2020.

[CZ17] Jiecao Chen and Qin Zhang. Improved algorithms for distributed entropy monitoring. *Algorithmica*, 78(3):1041–1066, 2017.

[Dar01a] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.

[Dar01b] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.

[Dat10] Jon Dattorro. *Convex optimization & Euclidean distance geometry*. Lulu. com, 2010.

[DDH07] James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007.

[DDH+09] Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W Mahoney. Sampling algorithms and coresets for ℓ_p regression. *SIAM Journal on Computing*, 38(5):2060–2078, 2009.

[DDHK07] James Demmel, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg. Fast matrix multiplication is stable. *Numerische Mathematik*, 106(2):199–224, 2007.

[Dec96] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, 1996.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DJ04] Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1):315 – 323, 2004.

[DJS+19] Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David Woodruff. Optimal sketching for kronecker product regression and low rank approximation. In *Advances in Neural Information Processing Systems*, pages 4737–4748, 2019.

[DK19] Ilias Diakonikolas and Daniel M Kane. Recent advances in algorithmic high-dimensional robust statistics. *arXiv preprint arXiv:1911.05911*, 2019.

[DL09] Michel Marie Deza and Monique Laurent. *Geometry of cuts and metrics*, volume 15. Springer, 2009.

[DLT03a] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 325–336, 2003.

[DLT03b] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *SIGCOMM*, pages 325–336, 2003.

[DLT04] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Flow sampling under hard resource constraints. In *SIGMETRICS*, pages 85–96, 2004.

[DLT07] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54(6):32, 2007.

[DM02] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[DM15] Arnaud Durand and Stefan Mengel. Structural Tractability of Counting of Solutions to Conjunctive Queries. *Theory Comput. Syst.*, 57(4):1202–1249, 2015.

[DNPR10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC, page 715–724. ACM, 2010.

[DNSS92] David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB)*, pages 27–40, 1992.

[Don06] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[DSSW18] Huaian Diao, Zhao Song, Wen Sun, and David Woodruff. Sketching for kro-

necker product regression and p-splines. In *International Conference on Artificial Intelligence and Statistics*, pages 1299–1308. PMLR, 2018.

[Duf04] Nick Duffield. Sampling for passive internet measurement: A review. *Statistical Science*, pages 472–498, 2004.

[DV06] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

[EJ91] E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 368–377. IEEE, 1991.

[EK72] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[ELB+17] Tarek Elgamal, Shangyu Luo, Matthias Boehm, Alexandre V Evfimievski, Shirish Tatikonda, Berthold Reinwald, and Prithviraj Sen. Spoof: Sum-product optimization and operator fusion for large-scale machine learning. In *CIDR*, 2017.

[ES06] Pavlos S. Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *IPL*, 97(5):181 – 185, 2006.

[EV03] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

[EVF03] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 153–166. ACM, 2003.

[fac16] Facebook's top open data problems. https://research.fb.com/blog/2014/10/facebook-s-top-open-data-problems/, 2016. Accessed: 2021-05-04.

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

[FCCR18] Rémi Flamary, Marco Cuturi, Nicolas Courty, and Alain Rakotomamonjy. Wasserstein discriminant analysis. *Machine Learning*, 107(12):1923–1945, 2018.

[FCT15] Martín Farach-Colton and Meng-Tsung Tsai. Exact sublinear binomial sampling. *Algorithmica*, 73(4):637–651, 2015.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FGRŽ21] Jacob Focke, Leslie Ann Goldberg, Marc Roth, and Stanislav Živnỳ. Approximately counting answers to conjunctive queries with disequalities and negations. *arXiv preprint arXiv:2103.12468*, 2021.

[FIM+01] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J Strauss, and Rebecca N Wright. Secure multiparty computation of approximations. In *International Colloquium on Automata, Languages, and Programming*, pages 927–938. Springer, 2001.

[FIS08] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008.

[FKN21] Arnold Filtser, Michael Kapralov, and Navid Nouri. Graph spanners by sketching in dynamic streams and the simultaneous communication model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1894–1913. SIAM, 2021.

[FKV04] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[Fla85] Philippe Flajolet. Approximate counting: a detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[FSS13] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1434–1453. Society for Industrial and Applied Mathematics, 2013.

[FST88] Schkolnick Finkelstein, Mario Schkolnick, and Paolo Tiberio. Physical database design for relational databases. *ACM Transactions on Database Systems (TODS)*,

13(1):91–128, 1988.

[FV98] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[Gan09] Sumit Ganguly. Deterministically estimating data stream frequencies. In *International Conference on Combinatorial Optimization and Applications*, COCOA, pages 301–312. Springer, 2009.

[GCPB16] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. 2016.

[GGLS16] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: Questions and answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 57–74, 2016.

[GGMW20] Shafi Goldwasser, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff. Pseudo-deterministic streaming. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 79:1–79:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[GGR98] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[GGR16a] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer Berlin Heidelberg, 2016.

[GGR16b] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Data stream management: A brave new world. pages 1–9, 01 2016.

[GHR+12] Anna C Gilbert, Brett Hemenway, Atri Rudra, Martin J Strauss, and Mary Wootters. Recovering simple signals. In *2012 Information Theory and Applications Workshop*, pages 382–391. IEEE, 2012.

[GHS+12] Anna C Gilbert, Brett Hemenway, Martin J Strauss, David P. Woodruff, and Mary Wootters. Reusable low-error compressive sampling schemes through privacy. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 536–539. IEEE, 2012.

[Gib01] Phillip B. Gibbons. Distinct sampling for highly-accurate answers to distinct

values queries and event reports. In *VLDB*, pages 541–550, 2001.

[GJK+97] Vivek Gore, Mark Jerrum, Sampath Kannan, Z Sweedyk, and Steve Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997.

[GKM18] Parikshit Gopalan, Daniel M Kane, and Raghu Meka. Pseudorandomness via the discrete fourier transform. volume 47, pages 2451–2487. SIAM, 2018.

[GKMS01] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin Strauss. Quicksand: Quick summary and analysis of network data. Technical report, 2001.

[GKMS02] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 454–465. Elsevier, 2002.

[GL08] Rainer Gemulla and Wolfgang Lehner. Sampling time-based sliding windows in bounded space. In *ACM SIGMOD*, pages 379–392, 2008.

[GLH06] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 595–606, 2006.

[GLH07] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bernoulli samples over evolving multisets. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 93–102, 2007.

[GLH08] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bounded-size sample synopses of evolving datasets. *VLDB J.*, 17(2):173–202, 2008.

[GLS98] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98, November 8-11, 1998, Palo Alto, California, USA*, pages 706–715, 1998.

[GLS00] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

[GLS02] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[GM98a] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics

for improving approximate query answers. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 331–342, 1998.

[GM98b] Phillip B Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *ACM SIGMOD Record*, volume 27, pages 331–342. ACM, 1998.

[GM06] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[GMP] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms.

[GMT15] Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 241–247. ACM, 2015.

[GMUW09] Hector Garcia-Molina, Jeffrey D Ullman, and Jennifer Widom. Database systems: the complete book. 2009.

[GO16] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[Gol05] Oded Goldreich. Foundations of cryptography - A primer. *Foundations and Trends in Theoretical Computer Science*, 1(1), 2005.

[Gol10] Oded Goldreich. Introduction to testing graph properties. In *Property testing*, pages 105–141. Springer, 2010.

[Gol17] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[Goo89] I. J. Good. C332. surprise indexes and p-values. *Journal of Statistical Computation and Simulation*, 32(1–2):90–92, 1989.

[GP14] Mina Ghashami and Jeff M Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 707–717. SIAM, 2014.

[Gro07] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007.

[Gro09] Andre Gronemeier. Asymptotically optimal lower bounds on the nih-multi-party information. *arXiv preprint arXiv:0902.1609*, 2009.

[GSS01] Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, STOC'01, pages 657–666, 2001.

[GSS09] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of Satisfiability*, pages 633–654. 2009.

[GT01] Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *ACM SPAA*, pages 281–291, 2001.

[GT02] Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *ACM SPAA*, pages 63–72, 2002.

[GT11] Alex Gittens and Joel A Tropp. Tail bounds for all eigenvalues of a sum of random matrices. *arXiv preprint arXiv:1104.4513*, 2011.

[GVL13] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013.

[GW95] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[GW18] Sumit Ganguly and David P. Woodruff. High probability frequency moment sketches. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 58:1–58:15, 2018.

[GWWZ15] Dirk Van Gucht, Ryan Williams, David P. Woodruff, and Qin Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 199–212, 2015.

[Haa81] Uffe Haagerup. The best constants in the khintchine inequality. *Studia Mathematica*, 70(3):231–283, 1981.

[Haa16] Peter J. Haas. Data-stream sampling: Basic techniques and results. In *Data Stream Management - Processing High-Speed Data Streams*, pages 13–44. 2016.

[had] http://hadoop.apache.org/.

[HAL+20] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for

physical simulation. *ICLR*, 2020.

[Haz16] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[HKM+20a] Avinatan Hasidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *Advances in Neural Information Processing Systems*, 33, 2020.

[HKM+20b] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *arXiv preprint arXiv:2004.05975*, 2020.

[HLA+19] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

[HN92] Pavol Hell and Jaroslav Nesetril. The core of a graph. *Discret. Math.*, 109(1-3):117–126, 1992.

[HN04] Pavol Hell and Jaroslav Nesetril. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications*. Oxford University Press, 2004.

[HNG+07] Ling Huang, XuanLong Nguyen, Minos Garofalakis, Joseph M Hellerstein, Michael I Jordan, Anthony D Joseph, and Nina Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 134–142. IEEE, 2007.

[HNO08a] Nicholas JA Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 489–498. IEEE, 2008.

[HNO08b] Nicholas JA Harvey, Jelani Nelson, and Krzysztof Onak. Streaming algorithms for estimating entropy. In *2008 IEEE Information Theory Workshop*, pages 227–231. IEEE, 2008.

[HNSS96] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Arun N Swami. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, 52(3):550–569, 1996.

[Hoe94] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[HPIS13] Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. 2013.

[HS92] Peter J Haas and Arun N Swami. *Sequential sampling procedures for query size estimation*, volume 21. ACM, 1992.

[HW13] Moritz Hardt and David P Woodruff. How robust are linear sketches to adaptive inputs? In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 121–130. ACM, 2013.

[HYZ12] Zengfeng Huang, Ke Yi, and Qin Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 295–306. ACM, 2012.

[Ind04] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 373–380. ACM, 2004.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[Ind07] Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). 2007.

[IT03] Piotr Indyk and Nitin Thaper. Fast color image retrieval via embeddings. In *Workshop on Statistical and Computational Theories of Vision (at ICCV)*, 2003.

[IVWW19] Piotr Indyk, Ali Vakilian, Tal Wagner, and David Woodruff. Sample-optimal low-rank approximation of distance matrices. *arXiv preprint arXiv:1906.00339*, 2019.

[IW03] Piotr Indyk and David Woodruff. Tight lower bounds for the distinct elements problem. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 283–288. IEEE, 2003.

[IW05] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM, 2005.

[Jay09] TS Jayram. Hellinger strikes back: A note on the multi-party information com-

plexity of and. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 562–573. Springer, 2009.

[Jin18] Ce Jin. Simulating random walks on graphs in the streaming model. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[JKS08] Thathachar S Jayram, Ravi Kumar, and D Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

[JL84] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. 1984.

[JOW+02] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *ACM SIGARCH Computer Architecture News*, 30(5):96–107, 2002.

[JS89] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM journal on computing*, 18(6):1149–1178, 1989.

[JST11] Hossein Jowhari, Mert Sağlam, and Gábor 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*, PODS '11, pages 49–58, New York, NY, USA, 2011. ACM.

[JSTW19] Rajesh Jayaram, Gokarna Sharma, Srikanta Tirthapura, and David P Woodruff. Weighted reservoir sampling from distributed streams. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 218–235, 2019.

[JSWY21] Rajesh Jayaram, Alireza Samadian, David P. Woodruff, and Peng Ye. In-database regression in input sparsity time. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, Proceedings of Machine Learning Research. PMLR, 2021.

[JVV86a] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[JVV86b] Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical computer*

science, 43:169–188, 1986.

[JW09] Thathachar S Jayram and David P Woodruff. The data stream space complexity of cascaded norms. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 765–774. IEEE, 2009.

[JW13] Thathachar S Jayram and David P Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)*, 9(3):26, 2013.

[JW18a] Rajesh Jayaram and David P Woodruff. Data streams with bounded deletions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 341–354, 2018.

[JW18b] Rajesh Jayaram and David P Woodruff. Perfect lp sampling in a data stream. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 544–555. IEEE, 2018.

[JW19] Rajesh Jayaram and David P Woodruff. Towards optimal moment estimation in streaming and distributed models. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[JWZ21] Rajesh Jayaram, David P. Woodruff, and Samson Zhou. Truly perfect samplers for data streams and sliding windows. 2021.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[KCR06] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *ACM SIGMOD*, pages 289–300, 2006.

[KIDP16] Ramakrishnan Kannan, Mariya Ishteva, Barry Drake, and Haesun Park. Bounded matrix low rank approximation. In *Non-negative Matrix Factorization Techniques*, pages 89–118. Springer, 2016.

[KJY+15] Arun Kumar, Mona Jalal, Boqun Yan, Jeffrey Naughton, and Jignesh M Patel. Demonstration of santoku: optimizing machine learning over normalized data. *Proceedings of the VLDB Endowment*, 8(12):1864–1867, 2015.

[KKP18] John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 556–567. IEEE, 2018.

[KLM89] Richard M Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of algorithms*, 10(3):429–448, 1989.

[KMM+20] Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1814–1833. SIAM, 2020.

[KMNS21] Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming. *arXiv preprint arXiv:2101.10836*, 2021.

[KN14] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):4, 2014.

[Kna09] Michael P Knapp. Sines and cosines of angles in arithmetic progression. *Mathematics Magazine*, 82(5):371–372, 2009.

[KNN+18] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: in-database learning thunderstruck. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, page 8. ACM, 2018.

[KNP15] Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. Learning generalized linear models over normalized data. In *ACM SIGMOD International Conference on Management of Data*, pages 1969–1984, 2015.

[KNP+17] Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. *arXiv preprint arXiv:1704.00633*, 2017.

[KNP19] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. 2019.

[KNPW11a] Daniel M Kane, Jelani Nelson, Ely Porat, and David P Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 745–754. ACM, 2011.

[KNPW11b] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 745–754,

New York, NY, USA, 2011. ACM.

[KNPZ16] Arun Kumar, Jeffrey Naughton, Jignesh M. Patel, and Xiaojin Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *International Conference on Management of Data*, pages 19–34, 2016.

[Knu97a] D. Knuth. *The art of computer programming: sorting and searching*, volume 3. 1997.

[Knu97b] D. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. 1997.

[Knu98] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.

[KNW10a] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.

[KNW10b] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52. ACM, 2010.

[Kon15] Christian Konrad. Maximum matching in turnstile streams. In *Algorithms-ESA 2015*, pages 840–852. Springer, 2015.

[Kop13] Swastik Kopparty. Lecture 7: eps-biased and almost k-wise independent spaces. http://sites.math.rutgers.edu/~sk1233/courses/topics-S13/lec7.pdf, 2013.

[KOSZ13] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920, 2013.

[KP20] John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1223–1236, 2020.

[KPW20] Akshay Kamath, Eric Price, and David P. Woodruff. A simple proof of a new set disjointness with applications to data streams, 2020.

[KS03] Robert Krauthgamer and Ori Sasson. Property testing of data dimensionality. In

Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, pages 18–27. Society for Industrial and Applied Mathematics, 2003.

[KSKW15a] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966, 2015.

[KSKW15b] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. 2015.

[KSM95] Sampath Kannan, Z Sweedyk, and Steve Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 551–557. Society for Industrial and Applied Mathematics, 1995.

[Kuh55] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955.

[KV00] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

[KV17] Ravindran Kannan and Santosh Vempala. Randomized algorithms in numerical linear algebra. *Acta Numerica*, 26:95–135, 2017.

[KVW14] Ravi Kannan, Santosh Vempala, and David Woodruff. Principal component analysis and higher correlations for distributed data. In *Conference on Learning Theory*, pages 1040–1057, 2014.

[KW08] J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

[LBKW14] Yingyu Liang, Maria-Florina F Balcan, Vandana Kanchanapally, and David Woodruff. Improved distributed principal component analysis. In *Advances in Neural Information Processing Systems*, pages 3113–3121, 2014.

[LCD05] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

[LCK19] Side Li, Lingjiao Chen, and Arun Kumar. Enabling and optimizing non-linear feature interactions in factorized linear algebra. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1571–1588, 2019.

[LHW17] Xingguo Li, Jarvis Haupt, and David Woodruff. Near optimal sketching of low-rank tensor regression. In *Advances in Neural Information Processing Systems*,

pages 3466–3476, 2017.

[Li09] Ping Li. Compressed counting. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 412–421. SIAM, 2009.

[Lib13] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[LMP13] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 127–136. IEEE, 2013.

[LMPL18] Thodoris Lykouris, Vahab Mirrokni, and Renato Paes Leme. Stochastic bandits robust to adversarial corruptions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 114–122, 2018.

[LN95] Richard J Lipton and Jeffrey F Naughton. Query size estimation by adaptive sampling. *Journal of Computer and System Sciences*, 51(1):18–25, 1995.

[LNNT16] Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 61–70. IEEE, 2016.

[LNS90] Richard J Lipton, Jeffrey F Naughton, and Donovan A Schneider. *Practical selectivity estimation through adaptive sampling*, volume 19. ACM, 1990.

[LNW14a] Yi Li, Huy L. Nguyen, and David Woodruff. Turnstile streaming algorithms might as well be linear sketches. 2014.

[LNW14b] Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 174–183. ACM, 2014.

[LNW19] Yi Li, Huy L Nguyen, and David P Woodruff. On approximating matrix norms in data streams. *SIAM Journal on Computing*, 48(6):1643–1697, 2019.

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*. https://arxiv.org/pdf/1905.04447.pdf, 2019.

[LW13] Yi Li and David P Woodruff. A tight lower bound for high frequency moment estimation with small error. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 623–638. Springer, 2013.

[LW16] Yi Li and David P Woodruff. Tight bounds for sketching the operator norm, schatten norms, and subspace embeddings. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[LWW14] Yi Li, Zhengyu Wang, and David P Woodruff. Improved testing of low rank matrices. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 691–700, 2014.

[LYFC19] Tam Le, Makoto Yamada, Kenji Fukumizu, and Marco Cuturi. Tree-sliced variants of wasserstein distances. 2019.

[LZ11] Ping Li and Cun-Hui Zhang. A new algorithm for compressed counting with applications in shannon entropy estimation in dynamic data. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 477–496, 2011.

[M+05] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[Mah11] Michael W Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[Mai94] Harry G Mairson. Generating words in a context-free language uniformly at random. *Information Processing Letters*, 49(2):95–99, 1994.

[McC90] Kevin S McCurley. The discrete logarithm problem. In *Proceedings of Symposia in Applied Mathematics*, volume 42, pages 49–74, 1990.

[McD89] Colin McDiarmid. *On the method of bounded differences*, page 148–188. London Mathematical Society Lecture Note Series. Cambridge University Press, 1989.

[MCS+06] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176. ACM, 2006.

[MFHH05] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 30(1):122–173, 2005.

[MG82] Jayadev Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.

[MHS19] Omar Montasser, Steve Hanneke, and Nathan Srebro. Vc classes are adversarially robustly learnable, but only improperly. In *Conference on Learning Theory*, pages

2512–2530. PMLR, 2019.

[MI10] Malik Magdon-Ismail. Row sampling for matrix algorithms via a non-commutative bernstein bound. *arXiv preprint arXiv:1008.0587*, 2010.

[MJ15] Jonas W Mueller and Tommi Jaakkola. Principal differences analysis: Interpretable characterization of differences between distributions. In *Advances in Neural Information Processing Systems*, pages 1702–1710, 2015.

[MM12] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. *PVLDB*, 5(12):1699, 2012.

[MM13] Xiangrui Meng and Michael W Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 91–100. ACM, https://arxiv.org/pdf/1210.3135, 2013.

[MNS11] Ilya Mironov, Moni Naor, and Gil Segev. Sketching in adversarial environments. *SIAM Journal on Computing*, 40(6):1845–1870, 2011.

[MNSW95] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 103–111. ACM, 1995.

[Moo01] David Moore, 2001.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[MP80] J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980.

[MP14] Gregory T Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 669–686. Society for Industrial and Applied Mathematics, 2014.

[MPTW16] Andrew McGregor, A Pavan, Srikanta Tirthapura, and David P Woodruff. Space-efficient estimation of statistics over sub-sampled streams. *Algorithmica*, 74(2):787–811, 2016.

[MRWZ20a] Sepideh Mahabadi, Ilya Razenshteyn, David P Woodruff, and Samson Zhou. Non-adaptive adaptive sampling on turnstile streams. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1251–1264,

2020.

[MRWZ20b] Sepideh Mahabadi, Ilya P. Razenshteyn, David P. Woodruff, and Samson Zhou. Non-adaptive adaptive sampling on turnstile streams. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1251–1264, 2020.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. pages 3111–3119, 2013.

[MSDO05] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. Finding (recently) frequent items in distributed data streams. In *IEEE ICDE*, pages 767–778, 2005.

[MSW20] Debmalya Mandal, Nisarg Shah, and David P Woodruff. Optimal communication-distortion tradeoff in voting. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 795–813, 2020.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[MV20] Andrew McGregor and Sofya Vorotnikova. Triangle and four cycle counting in the data stream model. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 445–456, 2020.

[MW10] Morteza Monemizadeh and David P Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. SIAM, 2010.

[MW17] Cameron Musco and David P Woodruff. Sublinear time low-rank approximation of positive semidefinite matrices. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 672–683. IEEE, 2017.

[Nag06] HN Nagaraja. Order statistics from independent exponential random variables and the sum of the top order statistics. *Advances in Distribution Theory, Order Statistics, and Inference*, pages 173–185, 2006.

[Nev02] Frank Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NN13] Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra al-

gorithms via sparser subspace embeddings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 117–126. IEEE, https://arxiv.org/pdf/1211.1002, 2013.

[Nol] John P Nolan. Stable distributions.

[NS19] Vasileios Nakos and Zhao Song. Stronger L2/L2 compressed sensing; without iterating. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

[NW70] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[NY15a] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, pages 565–584. Springer, 2015.

[NY15b] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Advances in Cryptology - CRYPTO - 35th Annual Cryptology Conference*, pages 565–584, 2015.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[OD14] Umut Oztok and Adnan Darwiche. Cv-width: A new complexity parameter for cnfs. In *ECAI*, pages 675–680, 2014.

[Olk93] Frank Olken. *Random sampling from databases*. PhD thesis, University of California, Berkeley, 1993.

[Pag13] Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013.

[PC19] Gabriel Peyré and Marco Cuturi. Computational optimal transport: With applications to data science. 11(5–6):355–607, 2019.

[PD08] Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 517–522, 2008.

[PDGQ05] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[PP13] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference*

on Knowledge discovery and data mining, pages 239–247, 2013.

[PPP21] Aditya Parulekar, Advait Parulekar, and Eric Price. L1 regression with lewis weights subsampling. *arXiv preprint arXiv:2105.09433*, 2021.

[PR03] Michal Parnas and Dana Ron. Testing metric properties. *Information and Computation*, 187(2):155–195, 2003.

[Pri18] Eric Price. Personal communication. November, 2018.

[PS13a] Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79(6):984–1001, 2013.

[PS13b] Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.*, 79(6):984–1001, 2013.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. pages 1532–1543, 2014.

[PV20] Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication. *arXiv preprint arXiv:2007.10254*, 2020.

[PVZ12] Jeff M Phillips, Elad Verbin, and Qin Zhang. Lower bounds for number-in-hand multiparty communication complexity, made easy. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 486–501. SIAM, 2012.

[Rab69] Michael O Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the american Mathematical Society*, 141:1–35, 1969.

[RD07] Florin Rusu and Alin Dobra. Pseudo-random number generation for sketch-based estimations. *ACM Transactions on Database Systems (TODS)*, 32(2):11, 2007.

[RD08] Florin Rusu and Alin Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems (TODS)*, 33(3):15, 2008.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

[Rel] https://www.relational.ai/.

[Ren13] Steffen Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pages 337–348. VLDB Endowment, 2013.

[RGG03] Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. *Database management systems*, volume 3. McGraw-Hill New York, 2003.

[RN16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[Roh19] Dhruv Rohatgi. Conditional hardness of earth mover distance. *arXiv preprint arXiv:1909.11068*, 2019.

[RTG00a] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[RTG00b] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The Earth Mover's Distance as a metric for image retrieval. 40(2):99–121, 2000.

[RV07] Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)*, 54(4):21–es, 2007.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 12(2):187–260, 1984.

[Sar06] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 143–152. IEEE, 2006.

[SBM+18] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. Darts: Deceiving autonomous cars with toxic signs. *arXiv preprint arXiv:1802.06430*, 2018.

[Sch35] Isaac J Schoenberg. Remarks to maurice frechet's article"sur la definition axiomatique d'une classe d'espace distances vectoriellement applicable sur l'espace de hilbert. *Annals of Mathematics*, pages 724–732, 1935.

[Sch07] Thomas Schwentick. Automata for xml?a survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[Sch18] Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 214–227, 2018.

[SD] Amit Shukla and Prasad Deshpande. Storage estimation for multidimensional aggregates in the presence of hierarchies.

[SDGP+15] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[Seg13] Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proceedings of the 16th International Conference on Database Theory*, pages 10–20, 2013.

[Seg14] Luc Segoufin. A glimpse on constant delay enumeration. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 13–27, 2014.

[Seg15] Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Rec.*, 44(1):10–17, May 2015.

[Sei90] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.

[She17] Jonah Sherman. Generalized preconditioning and undirected minimum cost flow. 2017.

[SL+91] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

[SOC16] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 3–18. ACM, 2016.

[spa] https://spark.apache.org/.

[SS+11] Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and trends in Machine Learning*, 4(2):107–194, 2011.

[SS17] Ohad Shamir and Liran Szlak. Online learning with local permutations and delayed feedback. In *International Conference on Machine Learning*, pages 3086–3094. PMLR, 2017.

[SSS95] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[ST04] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90,

2004.

[ST13] Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013.

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 684–697. SIAM, 2010.

[SW11] Christian Sohler and David P Woodruff. Subspace embeddings for the ℓ_1-norm with applications. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 755–764. ACM, 2011.

[SW19] Xiaofei Shi and David P Woodruff. Sublinear time numerical linear algebra for structured matrices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4918–4925, 2019.

[SWYZ19] Xiaoming Sun, David P Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. *arXiv preprint arXiv:1906.05736*, 2019.

[SWZ16] Zhao Song, David P. Woodruff, and Huan Zhang. Sublinear time orthogonal tensor decomposition. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NIPS) 2016, December 5-10, 2016, Barcelona, Spain*, pages 793–801, 2016.

[SWZ19] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *SODA 2019*. https://arxiv.org/pdf/1704.08246, 2019.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[Tao11] Terence Tao. Topics in random matrix theory. *Lecture Notes*, 2011.

[TBG+06] Dan Teodosiu, Nikolaj Bjorner, Yuri Gurevich, Mark Manasse, and Joe Porkka. Optimizing file replication over limited-bandwidth networks using remote differential compression. 2006.

[Ter99] Eugenia Ternovskaia. Automata theory for reasoning about actions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 153–159, 1999.

[Tho72] Robert C Thompson. Principal submatrices ix: Interlacing inequalities for singular values of submatrices. *Linear Algebra and its Applications*, 5(1):1–12, 1972.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.

[TLJ10] Marina Thottan, Guanglei Liu, and Chuanyi Ji. Anomaly detection approaches for communication networks. In *Algorithms for Next Generation Networks*, pages 239–261. Springer, 2010.

[Tro08] Joel A Tropp. Norms of random submatrices and sparse approximation. *Comptes Rendus Mathematique*, 346(23-24):1271–1274, 2008.

[Tro15] Joel Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

[Tur48] Alan M Turing. Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 1948.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1):57–81, 1968.

[TW11] Srikanta Tirthapura and David P Woodruff. Optimal random sampling from distributed streams revisited. In *International Symposium on Distributed Computing*, pages 283–297. Springer, 2011.

[TZ] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation.

[UZ11] Vladimir V Uchaikin and Vladimir M Zolotarev. *Chance and stability: stable distributions and their applications*. Walter de Gruyter, 2011.

[Val79] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Var95] Moshe Y Vardi. Alternating automata and program verification. In *Computer Science Today*, pages 471–485. Springer, 1995.

[Var00] Moshe Y. Vardi. Constraint satisfaction and database theory: a tutorial. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 76–85, 2000.

[VB96] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM re-*

view, 38(1):49–95, 1996.

[Ver10] Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

[Vit85a] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[Vit85b] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[VL92] Charles F Van Loan. *Computational frameworks for the fast Fourier transform*, volume 10 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[VL00] Charles F Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

[VLP93] Charles F Van Loan and N. Pitsianis. Approximation with Kronecker products. In *Linear algebra for large scale and real-time applications (Leuven, 1992)*, volume 232 of *NATO Adv. Sci. Inst. Ser. E Appl. Sci.*, pages 293–314. Kluwer Acad. Publ., Dordrecht, 1993.

[VNG47] John Von Neumann and Herman H Goldstine. Numerical inverting of matrices of high order. *Bulletin of the American Mathematical Society*, 53(11):1021–1099, 1947.

[VSBR83] LG Valiant, S Skyum, S Berkowitz, and C Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013.

[W+14] David P Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[Wai19] M.J. Wainwright. *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019.

[Wan19] Lan Wang. A new tuning-free approach to high-dimensional regression. ., 2019.

[WDL+09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the*

26th annual international conference on machine learning, pages 1113–1120, 2009.

[Wil10] Ross Willard. Testing expressibility is hard. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, pages 9–23, 2010.

[WKL09] Lan Wang, Bo Kai, and Runze Li. Local rank inference for varying coefficient models. *Journal of the American Statistical Association*, 104(488):1631–1645, 2009.

[WL09] Lan Wang and Runze Li. Weighted wilcoxon-type smoothly clipped absolute deviation method. *Biometrics*, 65(2):564–571, 2009.

[Woo04] David Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 167–175. Society for Industrial and Applied Mathematics, 2004.

[Woo07] David Paul Woodruff. *Efficient and private distance approximation in the communication and streaming models*. PhD thesis, Massachusetts Institute of Technology, 2007.

[Woo09] David P Woodruff. The average-case complexity of counting distinct elements. In *Proceedings of the 12th International Conference on Database Theory*, pages 284–295. ACM, 2009.

[Woo11] David P. Woodruff. Near-optimal private approximation protocols via a black box transformation. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 735–744, 2011.

[WP05] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast ip networks. In *Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on*, pages 172–177. IEEE, 2005.

[WPB+18] Lan Wang, Bo Peng, Jelena Bradic, Runze Li, and Yunan Wu. A tuning-free robust and efficient approach to high-dimensional regression. Technical report, School of Statistics, University of Minnesota, 2018.

[WSV12] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of semidefinite programming: theory, algorithms, and applications*, volume 27. Springer Science & Business Media, 2012.

[WW15] Omri Weinstein and David P Woodruff. The simultaneous communication of disjointness with applications to data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 1082–1093. Springer, 2015.

[WW19] Ruosong Wang and David P Woodruff. Tight bounds for ℓ_p oblivious subspace embeddings. In *SODA*, 2019.

[WZ12] David P Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 941–960. ACM, 2012.

[WZ13] David P. Woodruff and Qin Zhang. Subspace embeddings and l_{p}-regression using exponential random variables. *CoRR*, abs/1305.5580, 2013.

[WZ16] David P Woodruff and Peilin Zhong. Distributed low rank approximation of implicit functions of a matrix. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 847–858. IEEE, 2016.

[WZ17] David P. Woodruff and Qin Zhang. When distributed computation is communication expensive. *Distributed Computing*, 30(5):309–323, 2017.

[WZ18] David P Woodruff and Qin Zhang. Distributed statistical estimation of matrix products with applications. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 383–394, 2018.

[WZ20a] David P. Woodruff and Amir Zandieh. Near input sparsity time kernel embeddings via adaptive sampling. In *International Conference on Machine Learning (ICML)*, 2020.

[WZ20b] David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. *CoRR*, abs/2011.07471, 2020.

[XTB08] Bojian Xu, Srikanta Tirthapura, and Costas Busch. Sketching asynchronous data streams over sliding windows. *Distributed Computing*, 20(5):359–374, 2008.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94, 1981.

[YGL+] Keyu Yang, Yunjun Gao, Lei Liang, Bin Yao, Shiting Wen, and Gang Chen. Towards factorized svm with gaussian kernels over normalized data.

[YO14] Arman Yousefi and Rafail Ostrovsky. Improved approximation algorithms for earth-mover distance in data streams. *1404.6287*, 2014.

[YZ13] Ke Yi and Qin Zhang. Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica*, 65(1):206–223, 2013.

[ZCL+18] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539, 2018.

[Zhu17] Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342, 2017.

[ZMW+21] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. Kll±approximate quantile sketches over dynamic datasets. *Proceedings of the VLDB Endowment*, 14(7):1215–1227, 2021.