Streaming Algorithms for High Throughput Massive Datasets

by

Gregory A. Vorsanger

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

January, 2017

Abstract

The field of streaming algorithms has enjoyed a deal of focus from the theoretical computer science community over the last 20 years. Many great algorithms and mathematical results have been developed in this time, allowing for a broad class of functions to be computed and problems to be solved in the streaming model. In the same amount of time, the amount of data being generated by practical computer systems is simply staggering.

In this thesis, we focus on solving problems in the streaming model that have a unified goal of being relevant to practical problems outside of the theory community. In terms of a common technical thread throughout this work, the theme here is an attention to runtime and the ability to handle large datasets that not only challenge in terms of memory available, but also in the throughput of the data and the speed at which the data must be processed.

We provide these solutions in the form of both theoretical algorithm and practical systems, and demonstrate that using practice to drive theory, and vice versa, can generate powerful new approaches for difficult problems in the streaming model.

Primary Reader: Vladimir Braverman

ABSTRACT

Secondary Reader: Benjamin Van Durme

Third Reader: Joshua Edmison

Acknowledgments

I owe a tremendous amount of gratitude and thanks to all three of my thesis readers who have all been incredible mentors to me.

Dr. Joshua Edmison has been a guiding light through this process and a great friend. His unique insight into solving problems using both science and engineering know-how is an inspiration and has had a great impact on how I approached my graduate research. At every step of the way he has shown great patience and support of my academic studies, and always reminded me to keep focused on my goal of graduating. With humor, creative thinking, and wisdom, Josh has been an excellent person to work for. Thank you Josh.

Dr. Benjamin Van Durme has been a great teacher and role model to me. Ben is a model of academic efficiency, managing a small army of grad students and producing incredible research, and I hope to someday achieve that level of managerial and scientific excellence. Despite how many people he manages, and how busy he is with his many projects, Ben always responds right away whenever I have a problem, always takes the time to explain even small concepts if they are unfamiliar, and always supports and listens to my ideas, no matter how small or how they might turn out. As a co-advisor, Ben has given me incredible

constructive feedback about how to become a better scientist as well as given me many great opportunities to develop. From the start, Ben has understood my goals both as a student and post graduation, and has provided me with thoughtful advice and help at every turn. Thank you Ben.

Dr. Vladimir "Vova" Braverman and I met almost by accident. After a helpful conversation with Dr. Rao Kosaraju on the process of finding a Ph.D. program, I was trying to determine if I would even be able to find a topic to study at JHU, let alone a professor to study with. He suggested looking at Dr. Braverman's website where I saw said he was studying streaming algorithms. I hadn't even taken Algorithms 1. Despite my lack of knowledge in algorithms, Dr. Braverman believed in me and took me as a student. Vova is kind, patient, and generous with his time. He has been a fantastic teacher to me and gave me an environment to grow and learn that many others would not have. Vova always encouraged me to chase my own academic interests and has provided me with tremendous support and freedom to do so. Vova has taught me the importance of taking learning one step at a time, and that with dedication and patience you can become an expert. I would never have been able to learn the theory that became this thesis without your help and guidance. Thank you Vova.

My parents, Gary and Debbie, who have not only shown me unwavering support and love, but have also taught me so much about the importance of hard work and to be unafraid to follow an unbeaten path. They've helped me immeasurably to figure out my unconventional path through grad school; I cannot thank them enough. Thank you Mom and Dad.

ACKNOWLEDGMENTS

Stefanie Berges, who was with me the entire way with love, support, companionship. Stef would also give me the occasional reminder that, while graduate school is tough, I should stop complaining, pull it together, and get back to work. I'd be lost without you. Thank you Stef.

I would like to thank all of my coworkers at Raytheon BBN Technologies who in many different ways supported me throughout my graduate studies, be it with flexible scheduling, inspiring technical conversations, or just a beer after work. I'd also like to thank BBN management for sponsoring my education and being supportive of my schedule. Thank you BBN.

Finally, to all of my family and friends not mentioned by name, thank you.

Dedication

For Mom, Dad, and Stef. I surely wouldn't have gotten here without you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The motivation for this thesis is surprisingly simple; storing information has become very affordable, and time is expensive as ever. The amount of data being generated today is simply staggering. Twitter generates 500 Million tweets per day,[1] data center volume is constantly increasing,[2] and the number of sensors in the world is increasing sharply.[3] However, the increase in processing power of computers may not be able to keep up in the near future,[4] and there are still only 24 hours in a day. With this comes a question, what do we do with all of the data?

In the last 20 years, the field of streaming algorithms has looked to address the theoretical limitations of processing massive datasets. Many celebrated algorithms have not only pushed forward the field of theoretical computer science, but also have been seen wide spread use in applications such as Networks, Compressed Sensing, Machine Learning, and Security.[5–8]

Because many problems are formally hard in the general case, a compromise is necessary.

By softening the assumptions and constraints from the general case, powerful practical software can be created. In the case of streaming algorithms and the problem of providing statistics for large data sets, the trade off is that perfect accuracy is not possible and approximate answers are returned.

In a sentence, the goal of this thesis is for theory to drive practice, and practice to drive theory. In this thesis we show that using theoretical methods to design algorithms for practical problems is a valuable problem solving methodology for large datasets. By focusing in on specific cases as opposed to broad and general ones, this method can provide novel and useful approaches to computer science problems that may be hard to solve in the general setting. The goal of all of the algorithms presented in this thesis is the same, to provide improved methods of calculation that can be used to solve practical problems.

In this research we make multiple different types of practical assumptions, ranging from the limited availability of finite precision arithmetic (Chapter 3) to the inability to read every element in a stream (Chapter 2). These assumptions, when combined with a design goal of processing data quickly, enable the creation of algorithms that provide better performance when throughput is high and speed is critical.

1.1 Overview of Main Results

In this thesis we consider and provide results for the following problems:

- For the problem of SubSampling in the Streaming model we provide results for a class of streams we describe as *dense* streams.[9] We provide improved bounds for frequency

moments with $k = 2$. Additionally, as best we are aware, we provide the first bounds for $k < 2$, and heavy hitters on sampled streams.

- For the problem of weighted sampling, we provide an algorithm for sampling without replacement that is less reliant on floating point precision that previous methods.[10]
- For the problem of measuring the independence of two streams, we provide a generalization as well as an improved bound from $O\left(\left(\frac{\log(nm)}{\epsilon}\right)^{1024}\right)$ to $O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2\left(\frac{nm}{\epsilon}\right)\right)$.[11]
- We introduce a clustering style problem in the streaming model called Fuzzy Heavy Hitters and introduce the first algorithm and implementation.

While the topics and applications for these results cover a variety of problems in the same model, the solutions share a common thread. All of these results are focused on providing analytics for large datasets in a more efficient way than was previously possible, with a focus on being useful in software, not just theory. This goal is achieved by carefully choosing assumptions and providing fast per update runtime with the aim of supporting high throughput data streams.

We provide more detail on the motivation for each of these problems in Section 1.5 and for each problem we present the work as its own chapter.

1.2 The Streaming Model

This research is focused in the streaming model, a theoretical framework for very large datasets. Algorithms built in this model are called streaming algorithms, and have three

core facets.[12] First, they use a *sublinear* amount of memory. As the dataset grows larger and larger, the amount of memory used to compute grows much more slowly, allowing a small amount of memory to enable computation for the largest datasets. Second, the algorithm uses a small constant number of passes over the data set (ideally one pass). For truly large datasets, looking through the data more than a handful of times is impractical, and in many cases, such as network traffic, the data may be unavailable after the initial processing window. Finally, the algorithm needs to use a sublinear (ideally constant) amount of time to process each new element.

While there are multiple different types of streams inside of the streaming model, such as the time-series, cash register, sliding window, and decay, this work focuses mainly on the cash-register model, which is defined as follows from Muthukrishnan:[12]

Definition 1.2.1. *Given an underlying* signal *A, a one dimensional function* $A : [1 \cdots N] \rightarrow R$. *The algorithm is then given an input* stream *S of length* m *with elements* $a_1, a_2, ...a_m$*, where element* a_i *is a pair* (j, I_i) *with* $j \in \{1, ..., n\}$ *and* $I_i \geq 0$. *Each* a_i *can be viewed as a increment of* I_i *to A[j].*

Within the constraints of the streaming model, there has been a broad focus on providing algorithms for many different statistics. Many recent and exciting results in the theory community have focused on driving down the memory cost of algorithms, tightening wider approximations, or reducing the number of passes. While some of the work present in this document certainly aligns with the same goals, the focus of this thesis is how to improve efficiency for practical data sets.

While the streaming model is a general model of computation, considerable attention has been paid to the problems of Heavy Hitters and Frequency Moments. Aside from the more intuitive use of finding frequent elements, F_0 measures the number of unique elements in a stream, F_2 can measure the output size of a self join of a database, and higher frequency moments can measure the skew of a dataset to determine which algorithms to use for partitioning.[13] Frequency moments can also be used to measure L_k distance between two streams.

Definition 1.2.2. *Frequency Moments*

*A k-th **frequency moment** of a stream D is defined by* $F_k(D) = \sum_{i \in [n]} f_i^k$.

Definition 1.2.3. *Heavy Hitters*

Let D *be a stream and* α *be a parameter. The index* $i \in [n]$ *is a* α-heavy element *if* $f_i^k \geq \alpha F_k$.

These problems are very much related, as retrieving the heavy hitters for a dataset can in turn give an estimate of the frequency moment. This is especially true for larger frequency moments, such as when $k > 3$.

The problem of frequency moments was first studied in the streaming model by.[14] In this work they provided an optimal algorithm for F_2, but larger frequency moments, $k > 2$, was left as an open problem. Algorithms for finding F_1 and F_2 heavy hitters have been well studied.[15–17] Large frequency moments has also had multiple algorithms steadily improving bounds.[18–20] Further, the field continues to evolve with more pushes towards generalized

sketches such as the work of[21,22] that can solve multiple problems using one primitive sketch solving for heavy hitters.

1.3 Memory and Compromise

At its core, the streaming model is a model of compromise. Consider one of the fundamental problems in the streaming model, heavy hitters. Informally, heavy hitters represents a question: ”If I have a lot of labeled elements, such as numbered golf balls, which number do I have the most of?”

Answering this question on a small scale with a *linear* amount of memory compared to the number of elements is trivial using a heap or hash table. However, answering it using a *sublinear* amount of memory presents a more difficult challenge. In fact, it’s not possible for a deterministic algorithm to use a small amount of memory, as shown in a simple information theoretic argument from Muthukrishnan,[12] Theorem 9.

This is a simple example, but is indicative of a larger theme; there are necessary trade-offs to break boundaries of existing bounds. This means it is necessary to prioritize different qualities of algorithms over others, such as speed over accuracy, or limiting the generality of the problem. As mentioned in the previous section, this work focuses on leveraging these compromises to provide improved algorithms which can make a difference in practical settings when these compromises make sense.

1.4 Techniques

The algorithms in this thesis use a handful of techniques at their core. We now briefly define them and explain why they are used.

Sampling

When the time comes to implement these algorithms, asymptotic constant time per update can leave much to be desired. The logical next step then, is to try and figure out how to make our streaming algorithms more time efficient. While many techniques are used in this thesis, the two main pillars are sampling and hashing.

Sampling allows an algorithm to process less of the underlying dataset by only selecting a small portion. As further discussed in Chapters 2 and 3, sampling is a technique that has been well studied in the streaming model including heavy hitters and quantiles,[12] as well as others.[2324252627282930] The practical motivations for sampling expand beyond reducing the total work load, as some data provides such a high throughput that it's simply not possible to process every element if the update time is too slow, such as packets on network routers.[31]

Hashing

Hashing is another flexible tool used in many different ways throughout this thesis. We rely on two different kinds of hashing, Universal Hashing and Locality Sensitive Hashing.

Universal Hashing has the advantage of improving worst case retrieval time by reducing collisions. It does so using a family of hash functions. We use the definition for Universal hashing from.[32] Hashing is used extensively throughout streaming literature for building

sketches, including famous algorithms such as the F_2 algorithm,[14] Count Sketch,[15] Count Min Sketch,[16] my work with Braverman, Katzman, and Seidell in,[20] the sketches referenced in Chapter 2 and the sketch in Chapter 4.

Definition 1.4.1. Universal Hashing

Let H be a collection of hash functions that maps from keys from universe S into m bins. For all $h \in H$, $s_a = s_b$, $H(s_a) = H(s_b)$, i.e. all instances of key s_i end up in the same bin m_j .H is universal *if, given $s_c \neq s_d$, the probability that $Pr[H(s_c) = H(s_d)] = \frac{1}{m}$.*

Locality Sensitive Hashing is a process of solving the approximate nearest neighbor problem.[33] This method was later applied by Charikar[34] as a method for approximating the relative angle of high dimensional vectors by mapping them into a lower dimensional space.

Definition 1.4.2. Locality Sensitive Hashing

For Locality Sensitive Hash Function L, with $M = 2^b$ bins, L maps a high dimensional vector $v \in R^d$ into a lower dimensionality R^b. For u, $Pr[L(u) = L(v)] = 1 - \theta(u, v)$, where $\theta(u, v)$ is the angle between u and v.

1.5 Topics

The chapters are divided into two main groups, theory and implementation. Each topic is focused on the main theme of answering theoretical questions that relate to practical problems, or use theory to drive designing a practical system, as in Chapter 5. The topics range from solving fundamental problems in the streaming algorithm community, to trying to apply those concepts to practical computer science settings.

1.5.1 Theory

Chapter 2 Sampling and Subsampling:

A common, if not fundamental, problem in data analysis is to ask whether or not looking at a subset of a larger group of data is sufficient for calculation. This chapter investigates the ability to sample relatively small amounts of data from a stream and approximately calculate statistics on the original stream. Generally, sketch algorithms are able to store such small amounts of data by preserving important characteristics of the data set. With this, the difficulty of accurate sketching increases dramatically when there is no guarantee that critical elements will even be observed by the algorithm.

McGregor et al.[35] provide worst case theoretical bounds that show space costs for sampling that are inversely correlated with the sampling rate. Indeed, while the lower bound of McGregor et al. cannot be improved in the general case, we show it is possible to improve the space bound for stream D of domain n, when the average positive frequency $\mu = F_1/F_0$ is sufficiently large. We consider the following range of parameters: $\mu \geq \log(n)$ and sample rate $p \geq C_k \mu^{-1} \log(n)$, where C_k is a constant. On these streams we improve the bound from $\tilde{O}(\frac{1}{p} n^{1-2/k})$ to $\tilde{O}(n^{1-2/k})$ thus giving polynomial improvement in space for sufficiently large μ and p^{-1}.

Chapter 3 Cascade Sampling:

Continuing with another sampling result, we present Cascade Sampling, an efficient method for performing weighted sampling without replacement. Weighted sampling without

replacement has proved to be a very important tool in designing new algorithms. Cascade Sampling is not a revolution for weighted sampling, but instead represents a theoretical result aimed at removing an assumption about decimal precision, in turn providing greater applicability for practice, especially for systems where floating point operations can be limited or not implemented, or a high degree of precision is desired.

Efraimidis and Spirakis (IPL 2006) presented an algorithm for weighted sampling without replacement from data streams. Their algorithm works under the assumption of precise computations over the interval $[0, 1]$. Cohen and Kaplan (VLDB 2008) used similar methods for their bottom-k sketches.

Efraimidis and Spirakis ask as an open question whether using finite precision arithmetic impacts the accuracy of their algorithm. In this paper we show a method to avoid this problem by providing a precise reduction from k-sampling without replacement to k-sampling with replacement. We call the resulting method Cascade Sampling.

Chapter 4 Measuring Independence of Datastreams:

Another seemingly simple problem that is very difficult in the streaming model is the problem of determining if two streams of data are independent. Our algorithm provides a solid improvement over the previous bound $O(log(n)^{1024})$ to $O(log(n)^{14})$, but unfortunately this improved algorithm still does not quite push the usefulness of the technique to a practical level, as this approach only begins to do better than the traditional algorithm at approximately 1 trillion points, which is still larger than most current large data sets where this statistic needs to be calculated.

Phrasing the problem more formally, we aim to obtain small-space approximations of entrywise functions performed on a matrix that is generated by the outer product of two vectors given as a stream. In other works, streams typically define matrices in a standard way via a sequence of updates, as in the work of Woodruff[36] and others. We describe the matrix formed by the outer product, as well as other matrices that are not updated directly by new streaming elements, as implicit matrices. As such, we consider the general problem of computing over such implicit matrices with Hadamard functions, which are functions applied entrywise on a matrix. In this chapter, we apply this generalization to provide new techniques for identifying independence between two data streams. The previous state of the art algorithm of Braverman and Ostrovsky[37] gave a $(1 \pm \epsilon)$-approximation for the L_1 distance between the product and joint distributions, using space $O(\log^{1024}(nm)\epsilon^{-1024})$, where m is the length of the stream and n denotes the size of the universe from which stream elements are drawn. Our general techniques include the L_1 distance as a special case, and we give an improved space bound of $O(\log^{12}(n)\log^2(\frac{nm}{\epsilon})\epsilon^{-7})$.

1.5.2 Implementation

Chapter 5 Fuzzy Heavy Hitters:

This chapter came out of an interest in better understanding twitter, but grew into a study on how to group high dimensional data outside of existing clustering techniques such as k-means, k-medians, or DBscan.[38]

In this chapter, we define the problem of finding Fuzzy Heavy Hitters; the detection of

the heaviest clusters of points in a data stream, and propose a new algorithm to solve this problem. This algorithm was implemented in C++ with efficiency as a core design tenet. Experimental results on Twitter and synthetic data sets demonstrate the capability of this new approach.

Appendix: A Challenges in Developing Streaming Algorithms:

To close the thesis, this appendix provides insight and suggestions for improving the development of streaming algorithms to be used by the practical computer science community. The aim of this appendix is to better bridge the gap between theoretical streaming algorithms and software. Topics include programming languages, data selection, and the importance of prototypes.

Chapter 2

Subsampling

2.1 Introduction

An exciting topic of current algorithms research is evaluating the ability to sample relatively small amounts of data from a stream and to be able to approximately calculate statistics on the stream as a whole. In a recent paper,[35] McGregor, Pavan, Tithrapura, and Woodruff provided worst case theoretical bounds that show space costs for sampling that are inversely correlated with the sampling rate. [1] This implies it is not possible to sample effectively on the stream without a cost tradeoff. However, experimental work has shown that sampling can be performed on the stream without sacrificing additional space for accuracy.[39] Let us define the following terms:

Definition 2.1.1. *Let m,n be positive integers. A* ***stream*** *D = D(n,m) is a sequence of size*

[1] We have recently learned from an anonymous reviewer that these lower bounds may not hold. We stress that our techniques are independent of this result, and thus they hold regardless of the correctness of this work.

m of integers $a_1, a_2, \ldots, a_m$*, where* $a_i \in \{1, \ldots, n\}$*. A* ***frequency vector*** *is a vector of dimensionality n with non-negative entries* $f_i, i \in [n]$ *defined as:*

$$f_i = |\{j : 1 \leq j \leq m, a_j = i\}|$$

Definition 2.1.2. *A k-th* ***frequency moment*** *of a stream D is defined by* $F_k(D) = \sum_{i \in [n]} f_i^k$*.* F_0 *is the number of distinct elements in the stream and* $F_\infty = max_{i \in [n]} f_i$*.*

Definition 2.1.3. *A* ***dense stream*** *is any stream D s.t* $F_1(D)/F_0(D) \geq \log(n)$

While the lower bound of McGregor et al. cannot be improved in the general case, we show it is possible to improve the space bound for a stream D of domain n and length m, when the the average positive frequency $\mu = F_1/F_0$ is sufficiently large. Specifically, we consider the following range of parameters: $\mu \geq \log(n)$ and $p \geq C_k \mu^{-1} \log(n)$, where C_k is a constant (defined in (2.6)).

As our main technical claim, we show in Theorem 2.3.1 that the frequency moment on the sampled stream, D_p, is a $1 + \epsilon$ approximation for the frequency moment on the entire stream with high probability. As a result, we show the problem of computing F_k on D is reducible to the problem of computing F_k on D_p and the reduction preserves the space bounds up to a constant factor. In particular, the space bounds are independent of the sample rate, p. We stress that for our range of parameters the problem of approximating F_k is as hard as the problem of approximating F_k on the set of all streams. In this case, the lower bound from[40] still applies. However, the lower bound from[35] does not apply, as this bound

is proven for streams with average frequency bounded by a constant. On these streams we improve the bound[2] from $\tilde{O}(\frac{1}{p}n^{1-2/k})$ to $\tilde{O}(n^{1-2/k})$ thus giving polynomial improvement in space for sufficiently large μ and p^{-1}. Additionally, we provide proof that the same result is applicable for finding heavy elements (heavy hitters) in the stream. Specifically, we show that heavy elements in the original stream are heavy elements in the sampled stream. Thus, techniques to analyze heavy elements are also unaffected by the sampling rate. We also describe several practical applications where streams have high average frequency.

2.1.1 Related Work

For many applications it is practical to consider sampling data instead of attempting to process the entire data set. This is especially true as data sets grow larger and larger. The concept of accurately calculating statistics using small portions of a stream is not new, and sampling algorithms in the streaming setting have been studied for a long time,[23],[24],[25],[26],[27,28],[29].[30]

Calculating frequency moments is one of the central problems for streaming algorithms, see, e.g.,,[14],[41],[42],[43][44,45],[46],[47],[48],[40],[49–53],[54],[55],[56],[57],[58],[59] and the references therein. Further, computing frequency moments and other functions using sampling has been an intriguing question for a long time.[60] For example, Bar-Yossef showed[3][25] that the complexity of sampling from streams differs from the complexity of sketching by a polynomial factor in the worst case. Specifically computing F_2 is possible using $\tilde{O}(1)$ bits, but $\tilde{\Omega}(n^{0.5})$ samples are still needed.

[2]The $\tilde{O}$ notation suppresses factors polynomial to $\frac{1}{\epsilon}$ and factors logarithmic in m and n.

[3]While Bar-Yossef showed his results in a slightly different model the lower bounds are applicable for the sampled streams as well. See also Theorem 3.1 from.[35]

In 2007 Bhattacharyya, Madeira, Muthukrishnan, and Ye[61] considered skipping certain portions of the stream and only examining every Nth item deterministically.

Following this, in[62] Problem 13, Matias asked about the effects of subsampling on the streaming data. His question addresses the issue of very fast streams, ones that cannot be analyzed effectively even if each element can be processed in O(1) time. In addition to asking questions regarding,[61] he also asked about how subsampling effects the accuracy of standard calculations, such as frequency moments.

Recent work provided by McGregor, Pavan, Tirthapura, and Woodruff[35] considered sampling streams and addressed several fundamental problems, including frequency moments, heavy hitters, entropy, and distinct elements. In particular, they provide a matching (up to a polylogarithmic factor) upper and lower bound for the problem of frequency moments for $k > 2$ with lower bound of $\tilde{\Omega}(\frac{1}{p}n^{1-2/k})$. However, if we can observe the entire stream, then we can apply the well known upper bound of $\tilde{O}(n^{1-2/k})$ from Indyk and Woodruff.[19] Thus, the bound of[35] shows that it is not possible to obtain approximations without increasing the space required by a factor of p^{-1}, in the worst case.

However, in 2009, Rusu and Dobra[39] experimentally showed that when 10% of the original stream is sampled then the second frequency moment is still preserved. This provides intuition that there may be certain inputs that allow for an improvement over the bound from.[35]

2.1.2 Relation to Existing Work on Lower Bounds

We now explain why the lower bound of $\tilde{\Omega}(\frac{1}{p}n^{1-2/k})$ does not apply to our analysis.

The lower bound in question only applies when $n = \Theta(m)$. Consider streams such that $F_0 = \Omega(n)$ and for all i either $f_i = 0$ or $f_i > n$. Clearly in this case $n = o(m)$ and thus the lower bound of[35] do not necessarily apply. Indeed, if we sample with probability $p = n^{-0.5}$ then, with high probability, all sampled frequencies will be in the range $[(1-\epsilon)n^{-0.5}f_i, (1+\epsilon)n^{-0.5}f_i]$ for constant ϵ and sufficiently large n. Thus, it is not hard to show that F_k on the entire stream can be approximated by computing the frequency moment on the sampled stream, $\tilde{F}_k$. In this paper we investigate the range of parameters for which sampled streams possess these properties.

Consider Theorem 4.33 from.[25] Let us consider the case when $k = 2$. To prove the lower bound of Bar-Yossef considers the following example. Either (1) the stream represents a frequency vector with all frequencies bounded by 1 or (2) the stream represents a frequency vector with all frequencies bounded by 1 and one frequency is $O(n^{1/2})$. Observe that in both cases the average non-zero frequency $\mu = O(1)$. Since we require $\mu = \Omega(C_k \log(n))$ the lower bound from[25] is not applicable directly to our range of parameters. Is it possible to increase μ by repeating the same element many times. However, the bound from[25] is for algorithms that are based solely on sampled data. In our model, we first sample and then we can apply an arbitrary algorithm, including the sketching algorithm for F_2 from.[14] In this case the lower bound on the number of samples from[25] becomes the lower bound on the

length of the sampled stream D_p which is $\tilde{F}_1$.

In the same way consider Section 3.3 from.[35] The authors explicitly state that their bound is for the case when $m = \Theta(n)$. (It is important to note that the implicit (and standard) assumption in[35] is that $F_0 = \Theta(n)$. Otherwise, better bounds are possible, even on the original stream. E.g., if $F_0 = O(1)$ then we can compute any F_k precisely). In the proof of Theorem 3.3 in[35] the construction requires each element to be included at most once in the stream (except for one special element).

We give polynomial improvements over previous methods for the case when the non-zero average frequency is polynomial. Consider the stream where the average non-zero frequency is n^3. For sampling rate $p = \frac{1}{n^2}$ the bound[35] is $\tilde{\Omega}(min(n, \frac{1}{p}n^{1-2/k})) = \tilde{\Omega}(n)$. Our improvement for such streams can be as large as $\tilde{\Omega}(n^{2/k})$. Consider streams with the average frequency n^ζ where $0 < \zeta < 1$. If the sampling rate is $p = C_k \log(n)\frac{1}{n^\zeta}$ then our improvement is of order $\tilde{\Omega}(\frac{1}{p})$.

2.1.3 Objective

Specifically we ask the following question:

Question 2.1.1. *For which range of parameters it is possible to overcome the lower bound of[35] and show that similar results can be achieved at a lower space cost?*

Our goal is to investigate the range of input where sub-sampling is effective despite the lower bounds of.[35] In particular, we show that if $F_1 \geq C_k \log(n) F_0$ then it is possible to sample the stream with rate $\Theta\left(\frac{C_k \log(n) F_0}{F_1}\right)$ without sacrificing precision. We will analyze

upper space bounds for frequency moments on sampled streams.

2.1.4 Results

We show in this paper that the space requirement bound in[35] can be improved on a sufficiently long stream, given input with specific characteristics such that the stream is a dense stream. We have found good examples of data which have the characteristics and review them in the discussion section. Specifically, we improve these results for stream D of domain n, when the average frequency of all elements in D is greater than $C_k \log(n)$.

To the best of our knowledge, this is the first theoretical bound that shows strict improvement for sampling (no time/space trade-off) and thus gives justification for practical observations such as.[39] Note that our results do not contradict the lower bounds of.[35] In,[35] the lower bound is given for the case when $F_1 = \Theta(F_0)$; this is not the case for the streams we analyze, and thus does not effect correctness of the upper bounds in this paper.

All of our results are applicable for the following range of parameters: $\mu \geq \log(n)$ and $p \geq C_k \mu^{-1} \log(n)$, where C_k is a constant defined in (2.6). Our contributions are:

- As our main technical claim, we show in Theorem 2.3.1 that the frequency moment on the sampled stream is a $1 + \epsilon$ approximation for the frequency moment on the entire stream with high probability.

- As a result, we show the problem of computing F_k on D is reducible to the problem of computing F_k on D_p and the reduction preserves the space bounds up to a constant factor. In particular, the space bounds are independent of the sample rate, p.

- We provide the bound of $\tilde{O}(n^{1-2/k})$ for $k > 2$. On our range of parameters we improve the bounds of[35] by a factor of $1/p$. In fact, our recent result[44] implies a bound of $O(n^{1-2/k})$ bits.

- We provide the bound of $\tilde{O}(1)$ for $1 \leq k < 2$ for F_k approximation. To the best of our knowledge this is the first theoretical bound for this range of k on sampled streams.

- We provide proof that our result is also applicable for finding heavy elements (heavy hitters) in a stream. See Section 4. To the best of our knowledge this is the first theoretical bound for heavy hitters in sampled streams.

- We give a concentration bound on the sum of k-th powers of binomial random variables using inequalities for Sterling numbers of the 2nd kind, Bell numbers, and the Hölder Inequality.

It is important to note that the space lower bound $\tilde{\Omega}(n^{1-2/k})$ holds for streams with arbitrary large μ. To see this, consider a stream D with the average non-zero frequency smaller than some parameter t. Replace stream D with stream D', where every element of D is repeated exactly t times. In this case the average non-zero frequency in D' is increased exactly by factor of t. Since the μ is always at least one, we conclude that $\mu(D') \geq t$. It is not hard to see that the lower bound from[40] will be applicable for such D'. Thus, our restrictions do not make the problem of approximating F_k easier.

2.1.5 Intuition

Given a stream D, of length m with domain n, we assume that $m = \theta(n)$. However, as datasets get large, it is often the case that the expected frequency of a given element increases significantly. If this is the case then we can sample the stream without losing much precision (at least for the F_k approximation). As a result, we can improve the space bounds for frequency moments on sampled streams.

Our main claim is that $\tilde{F}_k$ is approximately $p^{-k}F_k$ if the *expected* frequency μ is sufficiently large and $p \geq \mu^{-1}\log(n)$. Specifically, we prove that the value of the frequency moment will be preserved (up to a multiplicative error) with high probability. It is easy to see[4] that the sampled frequency $\tilde{f}_i$ is a random variable with binomial distribution. Thus, the frequency moment on the sampled stream is $\tilde{F}_k = \sum_{i=1}^{n} \tilde{f}_i^k$ where $\tilde{f}_i \sim B(f_i, p)$.[5] Note that $\tilde{f}_i$s are independent but not identically distributed since the numbers of trials are different.[6] To obtain our result, we use the relation between the the moments of $\tilde{f}_i$, the Stirling numbers of the second kind and the Bell numbers.

Intuitively, when sampling datasets with large average frequency, we can divide all elements into one of three categories: A_1, the category of all elements with frequency greater than the sampling rate multiplied by an $O(\log(n))$ factor, A_2, elements with frequency greater than the sampling rate but less than A_1, and A_3, elements with frequency smaller

[4] Similar observation has been made in[35]

[5] We denote $B(0, p)$ as the degenerate distribution concentrated at 0.

[6] A slightly different case is well studied, when $Y = \sum_{i=1}^{n} Y_i^k$ where $Y_i \sim B(n, p_i)$, i.e., the number of trials is the same, but success probabilities are different. See e.g.,[63] for more details.

than the sampling rate. With this, we can prove that the group of elements in A_1 dominates the frequency moment of a dense stream. In this paper, we prove that the contribution of the sampled frequencies from the second two groups is negligible, with high probability. This allows us to accurately estimate the frequency moment of the sampled stream using only elements in A_1. We also prove that the frequency of each element f_i in A_1 is preserved within $1 \pm \Theta(\epsilon)$, while sampling with rate $p \geq C_k \mu^{-1} \log(n)$, for sufficiently large constant C_k. Thus, $\tilde{F}_k$ is a $(1 \pm \epsilon)$-approximation of F_k, and we can accurately perform our computations on D_p instead of D.

2.2 Definitions and Facts

The average positive frequency is defined as

$$\mu = \mu(D) = F_1/F_0. \tag{2.1}$$

Note that $\mu \geq 1$. Let us prove the following simple fact. Fact $\mu^k F_0 \leq F_k$

Proof. By Hölder inequality $F_1 \leq F_0^{1-1/k} F_k^{1/k}$. Thus, $\mu^k F_0 = (F_1/F_0)^k F_0 \leq F_k$. □

Definition 2.2.1. *Given data stream $D = \{a_1, a_2, \ldots, a_m\}$ and a fixed real $p \in (0, 1)$, let D_p be a random sub-stream of D obtained as follows. Let $Z_1, \ldots, Z_m$ be independent random variables such that $Z_i = a_i$ with probability p and $Z_i = -1$ with probability $(1-p)$. Denote D' to be the sequence $Z_1, \ldots, Z_m$. Next let D_p be the subsequence of D' obtained*

by deleting all -1*s. Define*[7]

$$\tilde{f}_i = frequency\ of\ i\ in\ D_p. \tag{2.2}$$

$$\tilde{F}_k = \sum_{i=1}^{n} \tilde{f}_i^k. \tag{2.3}$$

$B(N, p)$ is the binomial distribution with N trials and success probability p, where N is a positive integer and $p \in [0, 1]$. For completeness, define $B(0, p)$ to be the degenerate distribution concentrated at 0.

2.3 Frequency Moments on Sampled Streams

Define:

$$\alpha_k = 64(k/\epsilon)^2, \tag{2.4}$$

$$\beta_k = (k+1)B_k, \tag{2.5}$$

where B_k is the k-th Bell number (see[64] for the definition).

$$C_k = \epsilon^{-1/k}(10\beta_k)^{1/k}\alpha_k. \tag{2.6}$$

Consider stream D such that:

$$\mu \geq C_k \log(n). \tag{2.7}$$

[7]Note that we make "two passes" on D to define D_p but our algorithms will only need one pass on D_p.

Let p be such that:

$$1 \geq p \geq \mu^{-1} C_k \log(n). \tag{2.8}$$

Let $k > 1$ and ϵ be arbitrary constants. We now divide elements by frequency. Define:

$$S_1 = \{i : f_i \geq \alpha_k p^{-1} \log(n)\}, \tag{2.9}$$

$$S_2 = \{i : p^{-1} \leq f_i < \alpha_k p^{-1} \log(n)\}, \tag{2.10}$$

$$S_3 = \{i : f_i < p^{-1}\}, \tag{2.11}$$

Denote random variables $X_j, j \in \{1, 2, 3\}$:

$$X_j = p^{-k} \sum_{i \in S_j} \tilde{f}_i^k. \tag{2.12}$$

Denote numbers $A_j, j \in \{1, 2, 3\}$:

$$A_j = \sum_{i \in S_j} f_i^k. \tag{2.13}$$

For completeness define $A_j = X_j = 0$ if $S_j = \emptyset$ for $j = 1, 2, 3$. It follows that $p^{-k}\tilde{F}_k = X_1 + X_2 + X_3$ and $F_k = A_1 + A_2 + A_3$. We will show that, with high probability: A_1 is very close to X_1, $A_3 + A_2$ is negligible in terms of F_k, and $X_2 + X_3$ is bounded by $c(A_3 + A_2)$ for some constant c. As a result, we will prove that $X_1 \approx p^{-k}\tilde{F}_k$ is a good approximation of

F_k. Define

$$\gamma = \epsilon/2k \tag{2.14}$$

Fact For any i the following is true. If

$$|p^{-1}\tilde{f}_i - f_i| \leq \gamma f_i \tag{2.15}$$

then

$$|p^{-k}\tilde{f}_i^k - f_i^k| \leq \epsilon f_i^k. \tag{2.16}$$

Proof. For any fixed value of $\tilde{f}_i$ put $x = p^{-1}\tilde{f}_i$ and $y = f_i^k$. The lemma follows from Facts 2.8.2 and 2.8.2. Thus, the lemma follows for the random variable $\tilde{f}_i$ as well. □

Lemma 2.3.1. $A_2 + A_3 \leq 0.1\beta_k^{-1}\epsilon F_k < \epsilon F_k$.

Proof. Recall that $i \in (S_2 \cup S_3)$ implies $f_i < \alpha_k p^{-1}\log(n)$. Thus,

$$A_2 + A_3 = \sum_{i \in S_2 \cup S_3} f_i^k \leq (\alpha_k p^{-1}\log(n))^k F_0. \tag{2.17}$$

Recall that $p \geq C_k \mu^{-1}\log(n)$. Thus,

$$A_2 + A_3 \leq F_0(\alpha_k C_k^{-1}\mu)^k. \tag{2.18}$$

Fact (2.6) yields

$$A_2 + A_3 \leq F_k(\alpha_k C_k^{-1})^k. \tag{2.19}$$

The first inequality of the lemma follows from the definition (2.6) of C_k. The second inequality follows since $\beta_k > 1$. □

Lemma 2.3.2. $P(X_2 \geq \epsilon F_k) \leq 0.1$

Proof. To bound X_2 we observe that $\tilde{f}_i \sim B(f_i, p)$. Also $i \in S_2$ implies that $1/p \leq f_i$. Thus, we can apply Lemma 2.8.1. In particular, the case (2.32) gives:

$$E(\tilde{f}_i^k) \leq \beta_k (f_i p)^k, \tag{2.20}$$

which in turn gives

$$E(X_2) = \frac{1}{p^k} \sum_{i \in S_2} E(\tilde{f}_i^k) \leq \beta_k \sum_{i \in S_2} f_i^k = \beta_k A_2. \tag{2.21}$$

Combining (2.21) with Lemma $(2.3.1)$ we obtain $E(X_2) \leq 0.1\epsilon F_k$. Note that X_2 is non-negative. Thus, the lemma follows from Markov inequality. □

Lemma 2.3.3. $P(X_3 \geq \epsilon F_k) \leq 0.1$

Proof. To bound X_3 we observe that $i \in S_3$ implies $1/p > f_i$. Thus we can apply Lemma 2.8.1. In particular (2.33) gives us:

$$E(X_3) = \frac{1}{p^k} \sum_{i \in S_3} E(\tilde{f}_i^k) \leq \frac{1}{p^k} \beta_k F_0. \tag{2.22}$$

Recall that $p \geq C_k \mu^{-1} \log(n)$ (see (2.8)). Thus, Fact 2.2 gives us:

$$E(X_3) \leq \frac{1}{p^k} \beta_k F_0 \leq \frac{C_k^{-k} \beta_k \mu^k F_0}{\log^k(n)} \leq 0.1 \epsilon F_k. \tag{2.23}$$

The lemma follows. □

Lemma 2.3.4. *If $i \in \{2, 3\}$ and $|X_i - A_i| > \epsilon F_k$ then $X_i > \epsilon F_k$.*

Proof. If $|X_i - A_i| > \epsilon F_k$ then either

$$X_i > A_i + \epsilon F_k \tag{2.24}$$

or

$$X_i < A_i - \epsilon F_k. \tag{2.25}$$

Note that $0 \leq A_i < \epsilon F_k$ (by the definition and Lemma 2.3.1) and $X_i \geq 0$ (by the definition). Thus (2.25) is not possible and (2.24) implies $X_i > \epsilon F_k$. □

Lemma 2.3.5. $P(|X_1 - A_1| > \epsilon F_k) \leq 0.1$.

Proof. Note that if $S_1 = \emptyset$ then $X_1 = A_1 = 0$ and thus the lemma is correct. Otherwise, let $i \in S_1$ be fixed. First, we will show that

$$P(|\tilde{f}_i - p f_i| > \gamma p f_i) \leq \frac{1}{10n}. \tag{2.26}$$

Indeed, $\tilde{f}_i = \sum_{j=1}^{f_i} Y_{i,j}$ where $Y_{i,j}$ are i.i.d. indicators with mean p. Thus $E(\tilde{f}_i) = p f_i$, and

by Chernoff bound (see e.g.,,[65] B.2) we have:

$$P(|\tilde{f}_i - pf_i|) > \gamma p f_i) \leq 2e^{(-\gamma^2 p f_i)/4}. \tag{2.27}$$

Direct computations and the definitions (2.4) and (2.14) imply that $\gamma^2\alpha_k = 16$. Since $i \in S_1$, it follows that $f_i \geq p^{-1}\alpha_k \log(n)$. Thus, $\gamma^2 p f_i \geq \gamma^2\alpha_k \log(n) = 16\log(n)$. Substituting this bound into (2.27) we obtain (for sufficiently large n):

$$P(|\tilde{f}_i - pf_i|) > \gamma p f_i) \leq 2e^{-4\log(n)} \leq \frac{1}{10n},$$

and thus (2.26) holds. Further, Fact 2.3 and (2.26) imply

$$P(|p^{-k}\tilde{f}_i^k - f_i^k| > \epsilon f_i^k) \leq \frac{1}{10n}. \tag{2.28}$$

If we apply (2.28) to every $i \in S_1$ and use the union bound and the fact that $|S_1| \leq n$ then the lemma follows immediately. Indeed,

$$P(|X_1 - A_1| > \epsilon F_k) \leq P(|X_1 - A_1| > \epsilon A_1) = \tag{2.29}$$

$$P(|\sum_{i\in S_1} p^{-k}\tilde{f}_i^k - \sum_{i\in S_1} f_i^k| > \epsilon(\sum_{i\in S_1} f_i^k)) \leq$$

$$P(\cup_{i\in S_1}(|p^{-k}\tilde{f}_i^k - f_i^k| > \epsilon f_i^k)) \leq \sum_{i\in S_1} P(|p^{-k}\tilde{f}_i^k - f_i^k| > \epsilon f_i^k) \leq 0.1.$$

□

Theorem 2.3.1. *Let D be a stream such that $\mu = \mu(D) \geq C_k \log(n)$ and let p be a number such that $1 \geq p \geq \mu^{-1} C_k \log(n)$. Let D_p be the sampled stream (see Definition 2.2.1). Let $k > 1$ and ϵ be constants. Then the following bound holds for sufficiently large n.*

$$P(|\tilde{F}_k - F_k| > 3\epsilon F_k) \leq 0.3.$$

Proof. Indeed,

$$P(|\tilde{F}_k - F_k| > 3\epsilon F_k) \leq \tag{2.30}$$

$$P(|X_1 - A_1| > \epsilon F_k) + P(|X_2 - A_2| > \epsilon F_k) + P(|X_3 - A_3| > \epsilon F_k).$$

Applying Lemma 2.3.4 we obtain:

$$P(|\tilde{F}_k - F_k| > 3\epsilon F_k) \leq \tag{2.31}$$

$$P(|X_1 - A_1| > \epsilon F_k) + P(X_2 > \epsilon F_k) + P(X_3 > \epsilon F_k).$$

The theorem follows from the union bound and Lemmas 2.3.5, 2.3.3, 2.3.2. □

Theorem 2.3.2. *Let D be a stream such that $\mu = \mu(D) \geq C_k \log(n)$ and let p be a number such that $1 \geq p \geq \mu^{-1} C_k \log(n)$. Let D_p be the sampled stream. Let $k > 1$ and ϵ be constants. Then it is possible to output the $(1 \pm \epsilon)$-approximation of F_k by making a single pass over D_p and computing $\tilde{F}_k$. Thus, the problem of computing F_k on D is reducible to the problem of computing F_k on D_p and the reduction preserves the space bounds. In*

particular, the space bounds are independent of p. Current best bounds for F_k include:

1. $\tilde{O}(n^{1-2/k})$ *memory bits for* $k > 2$.

2. $\tilde{O}(1)$ *memory bits for* $1 \leq k < 2$.

2.4 Finding Heavy Elements

Definition 2.4.1. *Let D be a stream and ρ be a parameter. The index $i \in [n]$ is a* ρ-heavy element *if $f_i^k \geq \rho F_k$.*

In this section, we show that a heavy element in the original stream remains a heavy element in the sampled stream, and therefore we can apply existing techniques for heavy hitters. The frequency of the found heavy element is $(1 \pm \epsilon)pf_i$, with high probability, by Chernoff bound.

Theorem 2.4.1. *Let D be a stream and i be a heavy element w.r.t. F_k on D. Let $k \geq 1$ and let $p \geq \mu^{-1} = F_0/F_1$. Then there exists a constant c_k such that with a constant probability, i is a c_k-heavy element w.r.t. F_k on D_p.*

Proof. By Chernoff bound, the frequency of i in D_p is at least $(1-\epsilon)pf_i$ with high probability. By Fact 2.4, the k-th frequency moment of D_p is bounded by $\alpha_k \mu^{-k} \sum_{i=1}^{n} v_i^k$. Thus, i is a heavy element. □

Fact Let $V \in (Z^+)^n$ be a vector with strictly positive integer entries v_i. Let $\mu = \frac{1}{n}\sum_{i=1}^{n} v_i$. Note that $\mu \geq 1$. Let $X_i \sim B(v_i, \mu^{-1})$ and $X = \sum_{i=1}^{n} X_i^k$. Then there exists a constant α_k that depends only on k such that $P(X > \alpha_k \mu^{-k} \sum_{i=1}^{n} v_i^k) < 0.1$.

Proof. By Lemma 2.8.1 (See Appendix A)

$$E(X_i^k) \leq \beta_k((\mu^{-1}v_i)^k + 1)$$

Thus,

$$E(X) < \beta_k(\mu^{-k}\sum_{i=1}^{n} v_i^k) + \beta_k n.$$

Also, by the Hölder inequality

$$\frac{\sum_{i=1}^{n} v_i}{n^{1-1/k}} \leq (\sum_{i=1}^{n} v_i^k)^{1/k}$$

Thus,

$$n^{1/k} = \mu^{-1}\frac{\sum_{i=1}^{n} v_i}{n^{1-1/k}} \leq \mu^{-1}(\sum_{i=1}^{n} v_i^k)^{1/k}$$

Finally, $n < (\mu^{-k}\sum_{i=1}^{n} v_i^k)$. We conclude the proof by putting $\alpha_k = 200\beta_k$ and applying Markov's inequality. □

2.5 Discussion

In addition to our theoretical findings, we have identified practical examples of datasets with high average frequency where using the methods from this paper would be advantageous.

First, social media websites, such as Facebook, seem to be good candidates for subsampling. These sites have datasets large enough to require streaming algorithms[66] and a low ratio of unique users to data generated. In 2010, Facebook[66] produced over 60 Terabytes of

data a day, and were storing up to 15 Petabytes of data total. In 2012, Facebook announced it had 1 billion users and over 1.13 trillion "Likes" on their website[67] and with the large amount of data generated per day, even with the naive assumption that each user generates the same amount of information, the stream will maintain a high average frequency of datapoints per user.

Another potential application lies in detecting specific kinds of DDOS attacks. The amount of data that is transferred during these attacks is tremendous, with recent examples sending up to 300 Gigabits of data per second.[68] Streaming algorithms have already been explored as a potential technique to stop DDOS attacks,[69] and subsampling can provide the ability to monitor a small fraction of the total network traffic, while still preserving statistics on the stream of information, thus aiding the process of packet source IP address monitoring during DDOS attacks. The act of subsampling provides the ability to gain information about DDOS traffic that was previously difficult to process due to the volume of packets received.[70] Many DDOS attacks have used a technique known as IP address spoofing, which allows an attacker to generate a random IP address for every packet sent.[71] However, the use of ingress filters on routers[71] creates interesting challenges for malicious parties attempting this category of attacks. As such, these attackers are forced to use randomized addresses within a single subnet mask,[71] which is much smaller range of potential IP addresses being used during an attack. Additionally, techniques such as IP density monitoring[72] also force attackers to use more limited IP ranges. This reduction in potential IP space, combined with the massive amount of packets being transferred may provide a suitable dataset with high

average frequency of packets per IP address or subnet, allowing our methods to be applied.

2.6 Open Questions

We believe there is more work to be done in analyzing sampled streams with specific input. We ask the following questions:

- Can we expand our techniques to other functions considered by McGregor et al. in[35]? We believe this is possible.
- Is it possible to expand our techniques to other functions, such as convex functions?
- Can we define an adaptive version of sampling in which we modify the sampling rate as a function of stream length?
- Can we apply these techniques to noisy data streams, that is, streams that have a chance randomly of deleting elements with a probability different than our sample rate? [8]

2.7 Conclusion

We believe that sampling can be a powerful tool in streaming analysis given datasets with the proper characteristics. Our improved bounds for these datasets allow space efficient subsampling while preserving frequency moments. We have shown practical examples of

[8] We thank an anonymous reviewer for this suggestion.

real world datasets that possess the necessary high average frequency and that would be good candidates for subsampling.

2.8 Appendix

2.8.1 Binomial Distribution and Stirling Numbers

Lemma 2.8.1. *Let $X \sim B(N,p)$. There exists a constant β_k that depends only on k and such that if $Np \geq 1$ then*

$$E(X^k) \leq \beta_k (Np)^k, \tag{2.32}$$

and if $Np < 1$ then

$$E(X^k) \leq \beta_k. \tag{2.33}$$

Proof. Let $S(k,l)$ be a Stirling number of the second kind and let B_k be the k-th Bell number (see[64] for the definition). Using (3.5) and (1.247) from,[63] we can write:

$$E(X^k) = \sum_{l=0}^{k} S(k,l) \frac{N! p^l}{(N-l)!}. \tag{2.34}$$

Recall that $B_k = \sum_{l=0}^{k} S(k,l)$. Thus,

$$E(X^k) \leq B_k \sum_{l=0}^{k} (Np)^l. \tag{2.35}$$

If $Np \geq 1$ then

$$E(X^k) \leq (k+1) B_k (Np)^k, \tag{2.36}$$

if $Np < 1$ then

$$E(X^k) \leq (k+1)B_k. \tag{2.37}$$

We conclude[9] our proof by defining $\beta_k = (k+1)B_k$. □

2.8.2 Useful Inequalities

We will use the following Bernoulli inequality : for $1 > x > 0$ and $k > 1$ (see,[74] inequality (1) in Section 3.1):

$$(1 - kx) \leq (1 - x)^k. \tag{2.38}$$

Also, for $x > 0$ and $k > 1$ and $0 < q < p$ (see,[74] inequality (8) in Section 3.1):

$$(1 + (x/q))^q \geq (1 + (x/p))^p. \tag{2.39}$$

In particular, putting $q = 0.5$ and $p = k$ we obtain (using the fact that $y \geq y^{0.5}$ for $y = (1 + 2x)$):

$$(1 + 2x) \geq (1 + (x/k))^k. \tag{2.40}$$

Fact Let x, y, ϵ be real numbers, let $k > 1$ and let $\gamma = \epsilon/2k$ as defined in (2.14). If

$$x - y \leq \gamma y \tag{2.41}$$

[9]The recent bound on Bell number is $B_k < \left(\frac{0.792k}{ln(k+1)}\right)^k$ due to[73]

then

$$x^k - y^k \leq \epsilon y^k. \tag{2.42}$$

Proof. Using (2.14) we have $x \leq (1 + \frac{\epsilon}{2k})y$ and thus

$$x^k \leq (1 + \frac{\epsilon}{2k})^k y^k. \tag{2.43}$$

Applying (2.40) we obtain

$$(1 + \frac{\epsilon}{2k})^k \leq (1 + \epsilon). \tag{2.44}$$

The fact follows. □

Fact Let x, y, ϵ be real numbers, let $k > 1$ and let $\gamma = \epsilon/2k$ as defined in (2.14). If

$$x - y \geq -\gamma y \tag{2.45}$$

then

$$x^k - y^k \geq -\epsilon y^k. \tag{2.46}$$

Proof. The proof is identical to the proof of Fact 2.8.2, but instead of (2.44) we use

$$(1 - \frac{\epsilon}{2k})^k \geq (1 - 0.5\epsilon) \geq (1 - \epsilon), \tag{2.47}$$

where we apply (2.38) for the first inequality. □

Chapter 3

Weighted Sampling Without Replacement

3.1 Introduction

Random sampling is a fundamental tool that has many applications in computer science (see e.g., Motwani and Raghavan,[75] Knuth,[76] Tille,[77] and Olken[78]). Random sampling methods are widely used is data stream processing because of their simplicity and efficiency[79–84][1]. In a stream, the size of the domain and the probability of sampling an element both change constantly; this makes the process of sampling non-trivial. We distinguish between sampling *with replacement*, where all samples are independent (and thus can be repeated), and sampling *without replacement*, where repetitions are prohibited. As best as we authors can

[1] In turn, this makes this method useful for fast moving streams, as it gives a strategy to skip elements when a stream is too fast every element.

determine, weighted sampling without replacement was first used by Rosen[852][2]

In particular, weighted sampling without replacement has proven to be a very important tool. In weighted sampling, each element is given a weight, where the probability of an element being selected is based on its weight. In their work Efraimidis and Spirakis[86] presented an algorithm for weighted sampling without replacement. Cohen and Kaplan[87] use similar methods for their bottom-k sketches. While their preliminary implementation yielded promising results, Efraimidis and Spirakis[86] state, as the main open problem of the paper, *"However, the question if, and to what extent, the finite precision arithmetic affects the algorithms remains an open problem."*

In this paper we continue this work and provide a new algorithm to avoid the issue of relying on finite precision arithmetic. With this result we show that precision loss is not required in order to sample without replacement. We accomplish this by providing a precise reduction from k-sampling without replacement to k-sampling with replacement, using a special case of k-sampling with replacement, unit sampling (where k=1). Additionally, we believe that in the future our method of expressing different random samples via reduction will provide a tool that allows further translation of other sampling methods into a more effective form for streams.

[2]The authors would like to thank Edith Cohen for help tracing the history of this problem

3.1.1 Related Work

Due to its fundamental nature, the problem of random sampling has received considerable attention in the last few decades.

In 2005, Vitter[30] presented uniform sampling using a reservoir (with and without replacement) over streams. Further, the question of reductions between sampling methods has been addressed before. For instance, Chaudhuri, Motwani and Narasayya[29] briefly discuss reductions for various sampling methods. Cohen and Kaplan[87] use a "mimicking process" in their papers, which is essentially a reduction from sampling without replacement to sampling with replacement.

Chaudhuri, Motwani and Narasayya[29] use the well-known method of "over-sampling", i.e. we sample the set independently until k distinct elements are obtained. Clearly, this schema does not introduce any precision loss, since unit sampling is used as a black-box.

Unfortunately, the amount of resources required to determine this information is a function of the weight distribution for the data set, and thus can be arbitrarily large.

In particular, consider the case when there is an element with weight that is overwhelmingly larger than the rest of the population. In this case, the number of repetitions found while sampling with replacement is significantly larger then k.

Probably the first effective non-streaming solution for the weighted sampling without replacement problem was the algorithm of Wong and Easton.[88] It is used by many other algorithms (see Olken[78] for the discussion). For data streams, Efraimidis and Spirakis[86]

proposed an algorithm that is based on the "exponent method". The algorithm requires precise computations of random keys $r^{1/w(p)}$, where $r \sim U[0,1]$. The sample generated is composed of the k elements with maximal keys. Cohen and Kaplan[87] used similar methods as a building block for their bottom-k sketches. The bottom-k sketch is an effective construction that has been extensively used for various applications including approximations of aggregative queries over data streams. As Cohen and Kaplan[89] show, these methods are very effective in practical applications and are superior to the sketches that are based on sampling with replacement.

While effective in practice, the algorithms of Efraimidis and Spirakis and Cohen and Kaplan introduce a loss of accuracy, since their techniques require additional floating point arithmetic operations.

3.1.2 Results

In this paper we show that the tradeoff between precision and performance is not a necessary property of sampling without replacement from data streams and construct a precise streaming reduction from k-sampling without replacement to k-sampling with replacement. This result provides a practical improvement to the algorithms of Efraimidis and Spirakis in cases where high accuracy is required.

Our method is yields a surprisingly simple algorithm, given the importance of sampling without replacement and the existence of many previous methods. We call this algorithm Cascade Sampling. In particular, when used with the algorithm from[29] Cascade Sampling

requires $O(k)$ memory, constant time per element and the same precision as in.[29]

3.1.3 Intuition

Let Λ be *any* algorithm that maintains a unit weighted sample from stream D. Similarly to the over-sampling method, we maintain instances of Λ. Namely, we maintain k instances $\Lambda_1, \ldots, \Lambda_k$. However, we introduce the idea of *stream modification*. That is, instead of applying Λ independently and symmetrically on D, we apply Λ_i on the modified stream D_i that does not contain samples of Λ_j for $j < i$. In particular, Λ_i may process its input elements in an order *different from the order of their arrival in* D. This simple but novel idea is sufficient to solve the problem. In particular, we can claim that the input of Λ_i is a random set that precisely matches the definition of weighted sampling without replacement. Since we use Λ as a black box with only a constant number of auxiliary variables, specifically pointers, the resulting schema is a precise reduction.

3.2 Definitions

An important building block of our algorithm is the concept of a unit sample, that is, the ability to sample a single element from a set.

Definition 3.2.1. *Let S be a finite set of elements and let w be a non negative function $w : S \to R$. A random element X_S with values from S is a* **unit weighted random sample** *if, for any $a \in S$, $P(X_S = a) = \frac{w(a)}{w(S)}$. Here $w(S) = \sum_{a \in S} w(a)$.*

For an algorithm instantiating weighted unit sampling we provide Black-Box WR2

from.[29] Black-Box WR2 is a unit sample when $r = 1$.

Algorithm 1 Black-Box WR2: Algorithm for Weighted Unit Sampling

1. $W \leftarrow 0$.
2. Initialize reservoir with length $r = 1$, λ_0.
3. For each tuple t in stream:
   (a) Get next tuple t with weight $w(t)$
   (b) $W \leftarrow W + w(t)$
   (c) Set $\lambda_0 = t$ with prob. $\frac{w(t)}{W}$
4. Return λ_0

Definition 3.2.2. *A* **data stream** *is an ordered, set of elements, $p_1, p_2, \ldots, p_n$, that can be observed only once. An algorithm A is a streaming sampling algorithm if A outputs a sample using a single pass over the data set.*

Definition 3.2.3. *A set $X = \{X_1, \ldots, X_k\}$ is called a* **k-sample with replacement** *from S if $X_1, \ldots, X_k$ are independent random unit samples from S.*

Another fundamental sampling method is *weighted sampling without replacement*.

Definition 3.2.4. *Let S be a finite set such that $|S| \geq k$. An ordered set $X = \{X_1, \ldots, X_k\}$ is called a k-***sample without replacement** *from $S, |S| \geq k$ if X_1 is a weighted unit sample from S and for any $j > 1$, X_j is a weighted unit sample from $S \setminus \{X_1, \ldots, X_{j-1}\}$.*

Definition 3.2.5. *We say that there exists an a* reduction *from a k-sampling to a unit sampling if for any unit sampling algorithm Λ there exists a k-sampling algorithm $\Upsilon = \Upsilon(\Lambda)$ that uses Λ as a black-box. We say that the reduction is* precise *if for any Λ that requires memory m and time t:*

1. *$\Upsilon(\Lambda)$ requires $O(km)$ memory and $O(kt)$ time.*

2. *$\Upsilon(\Lambda)$ only uses comparisons (in addition to using A as a black box).*

In other words, $\Upsilon(\Lambda)$ does not introduce any precision loss.

There exists a (trivial) precise reduction from weighted sampling with replacement to unit sampling. In this paper we give the first precise streaming reduction for weighted sampling without replacement to unit sampling.

3.3 Cascade Sampling

Let S be a finite set such that $|S| \geq k$ and let $a \notin S$. Denote $T = S \cup \{a\}$, and let $w : T \mapsto R^+$ be a function. Let $\{X_1, \ldots, X_k\}$ be a k-sample without replacement from S with respect to w. Define an ordered sequence $\{Y_1, \ldots, Y_k\}$[3] as follows:

$$Y_1 = \begin{cases} a, & \text{w.p. } \frac{w(a)}{w(T)}; \\ X_1, & \text{otherwise.} \end{cases} \tag{3.1}$$

[3]Here the additional randomness is independent.

For $i \geq 1$ define[4]:

$$L_i = \{X_1, \ldots, X_i, a\} \setminus \{Y_1, \ldots, Y_i\}. \tag{3.2}$$

We will show that $|L_i| = 1$; assuming that, let Z_i be the single element from L_i, i.e., $L_i = \{Z_i\}$. Put $U_i = T \setminus \{Y_1, \ldots, Y_i\}$. Define

$$Y_{i+1} = \begin{cases} Z_i, & \text{w.p. } \frac{w(Z_i)}{w(U_i)}; \\ X_{i+1}, & \text{otherwise.} \end{cases} \tag{3.3}$$

Lemma 3.3.1. *For all $i = 1, \ldots, k$ the ordered set $\{Y_1, \ldots, Y_i\}$ is an i-sample without replacement from T with respect to w.*

Proof. We prove the lemma by induction on i. For $i = 1$ the statement follows from direct computation and definitions. Assuming that the lemma is correct for i we need to prove that

$$Y_{i+1} \in T \setminus \{Y_1, \ldots, Y_i\}, \tag{3.4}$$

and for any $b \in U_i$:

$$P(Y_{i+1} = b) = \frac{w(b)}{w(U_i)}. \tag{3.5}$$

To show (3.4) observe that $\{Y_1, \ldots, Y_i\} \subseteq \{X_1, \ldots, X_i, a\}$ and $Y_{i+1} \in \{X_{i+1}, Z_i\}$. By definition $X_{i+1} \notin \{X_1, \ldots, X_i, a\}$ and $Z_i \notin \{Y_1, \ldots, Y_i\}$.

To show (3.5) fix $\{X_1, \ldots, X_i\}$ and $\{Y_1, \ldots, Y_i\}$; it follows that Z_i is fixed as well. Denote $V_i = U_i \setminus \{Z_{i-1}\}$ and $H_i = S \setminus \{X_1, \ldots, X_i\}$; it follows that $H_i = V_i$. For any

[4]Here $\setminus$ denotes the set difference, i.e. $A \setminus B = \{x : x \in A, x \notin B\}$.

fixed $b \in V_i$ we have

$$P(Y_{i+1} = b) = P(X_{i+1} = b)\frac{w(V_i)}{w(U_i)} = \frac{w(b)}{w(H_i)}\frac{w(V_i)}{w(U_i)} = \frac{w(b)}{w(U_i)}.$$

The case $b = Z_{i-1}$ is similar. □

3.4 Precise Reduction and Resulting Algorithm

Let Λ be an algorithm that maintains a unit weighted sample from D. The algorithm from[29] is an example of Λ but our reduction works with *any* algorithm for unit weighted sampling. We construct an algorithm $\Upsilon = \Upsilon(\Lambda)$ such that Υ maintains a k-sample without replacement. Specifically, we maintain k instances of Λ: $\Lambda_1, \ldots, \Lambda_k$ such that the input of Λ_i is a random substream of D that is selected in a special way. We denote the input stream for Λ_i as D_i. Let X_i be the sample produced by Λ_i. The critical observation is that our algorithm maintains the following invariant: at any moment $D_i = D \setminus \{X_1, \ldots, X_{i-1}\}$. Thus, by definition, the weighted sample from D_i is the i-th weighted sample from D when the samples are without replacement.

Theorem 3.4.1. *Algorithm $\Upsilon = \Upsilon(\Lambda)$ maintains a weighted k-sample without replacement from D. If Λ requires space $O(g)$ and time per element $O(h)$, then Υ requires $O(kg)$ space and $O(kh)$ time respectfully. Thus, there exists a precise reduction from k-sampling without replacement to a unit sampling.*

Proof. Follows from the description of the algorithm (See Algorithm 2) and Lemma 3.3.1.

□

Algorithm 2 Cascade Sampling

Input: Data Stream $D = \{p_1, \ldots, p_n\}$,

Λ *is an algorithm that maintains a unit weighted sample from* D,

$\Lambda_1, \ldots, \Lambda_k$ *are independent instances of* Λ

Output: Weighted k*-Sample Without Replacement* $\{Y_1, \ldots, Y_k\}$

1. For $j = 1, 2, \ldots, n$
   (a) $new = p_j$
   (b) For $i = 1, \ldots, \min\{j, k\}$
      i. If $(i < j)$ then set $previous = Y_i$ (where Y_i the current output of Λ_i).
      ii. Feed Λ_i with new
      iii. If Y_i changes its value to new, then set $new = previous$.
2. Output $\{Y_1, \ldots, Y_k\}$

Algorithm 2 provides a solution to the weighted k-Sampling without replacement problem. To better demonstrate the algorithm, we show an example of updating a single unit sample inside of loop *(b)* in Figure 1. In this example, unit sample λ_1 has currently sampled element a and unit sample λ_2 has currently sampled element b, where a and b are elements that appeared previously in the stream.

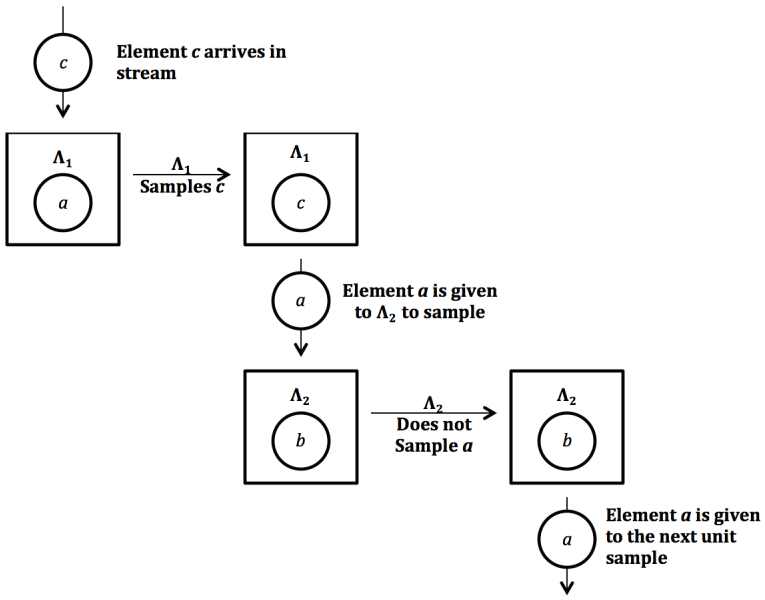

Figure 3.1: Updating a Unit Sample

3.4.1 Discussion

There are several directions in which our algorithm can be improved. In particular, run time dependent on the number of samples is one issue for practical datasets with large k. We believe this can be improved by combining several sampling steps into a single step which will be useful for the cases when the element will not be sampled into any of the substreams. This will often be the cases with elements with small weights. Specifically, we ask if it is possible to reduce the total running time from $O(nk)$ to $O(n \log k)$.

Another interesting direction is applying this algorithm to weighted random sampling with a bounded number of replacements as shown in.[90] Finally, this method may also be interesting when applied to the Sliding Window Model[45] and Streams with Deletions.[82]

Chapter 4

Measuring Hadamard Distance

4.1 Introduction

Measuring Independence is a fundamental statistical problem that is well studied in computer science. Traditional non-parametric methods of testing independence over empirical data usually require space complexity that is polynomial in either the support size or input size. With large datasets, these space requirements may be impractical, and designing small-space algorithms becomes desirable [1].

Measuring independence is a classic problem in the field of statistics (see Lehmann[91]) as well as an important problem in databases. Further, the process of reading in a two-column database table can be viewed as a stream of pairs. Thus, the streaming model is a natural choice when approximating pairwise independence as memory is limited. Indeed,

[1] While the sketch based methods shown in this chapter have fast per-element update time, they may in fact have slower update time than maintaining a traditional matrix data structure, but the size of the matrix required by such a data structure is simply too large for massive datasets.

identifying correlations between database columns by measuring the level of independence between columns is of importance to the database and data warehouse community (see, e.g.,[92] and,[93] respectively).

In this paper we provide new techniques for measuring independence between two data streams and present new tools to expand existing techniques. The topic of independence was first studied in the streaming model by Indyk and McGregor[94] where the authors gave an optimal algorithm for approximating the L_2 distance between the product and joint distributions of two random variables which generate a stream. In their work, they provided a sketch that is pairwise independent, but not 4-wise independent, so analysis similar to that of Alon, Matias, and Szegedy[14] cannot be applied directly. This work was continued by Braverman and Ostrovsky,[37] where the authors considered comparing among a stream of k-tuples and provided the first $(1 \pm \epsilon)$-approximation for the L_1 distance between the product and joint distributions. Their algorithm is currently the best known space bound, and uses $O(\frac{1}{\epsilon^{1024}} \log^{1024}(nm))$ space for $k = 2$, where m is the length of the stream and n denotes the size of the universe from which stream elements are drawn. We present new methods, in the form of a general tool, that enable us to improve this bound to $O(\frac{1}{\epsilon^7} \log^{12}(n) \log^2(\frac{nm}{\epsilon}))$. In previous works, a central challenge has been maintaining an approximation of the matrix that is generated by the outer product of the two streaming vectors. As such, we consider computing functions on such an implicit matrix. While, matrices have been studied previously in the streaming model (e.g.,[36]), note that we cannot use standard linear sketching techniques, as the entries of the matrix are given implicitly and thus these methods do not

apply directly.

Generalizing this specific motivating example, we consider the problem of obtaining a $(1 \pm \epsilon)$-approximation of the L_1 norm of the matrix $g[A]$, where $g[A]$ is the matrix A with a function g applied to it entrywise. Such mappings g are called *Hadamard* functions (see[95,96]). Note that we sometimes abuse notation and apply the function g to scalar values instead of matrices (e.g., $g(a_{ij})$ where a_{ij} is the $(i,j)^{th}$ entry in matrix A). We require the scalar form of function g to be even, subadditive, non-negative, and zero at the origin. We show that, given a blackbox $r(n)$-approximation of $\|g[A]\|_1 = \sum_i \sum_j g(a_{ij})$ (where a_{ij} is the $(i,j)^{th}$ entry in matrix A) and a blackbox $(1 \pm \epsilon)$-approximation of the aggregate of g applied entrywise to a vector obtained by summing over all rows, we are able to improve the $r(n)$-approximation to a $(1 \pm \epsilon)$-approximation (where $r(n)$ is a sufficiently large monotonically increasing function of n). Hence, we give a reduction for any such function g. Our reduction can be applied as long as such blackbox algorithms exist.

An interesting special case of our result is when the matrix is defined by the L_1 distance between the joint and product distributions, which corresponds to measuring independence in data streams. Since such blackbox algorithms are known for L_1, not only does our framework generalize the problem of measuring independence according to the L_1 distance, but our algorithmic techniques also yield improved space bounds over the previous state of the art result.[37] Moreover, our framework would immediately translate improved space bounds for the blackbox algorithms to improved space bounds for the application of measuring independence. Note that, for L_p where $0 < p < 1$, such blackbox algorithms are not known.

If such algorithms for the L_p distance were to be designed, our reductions work and can be applied. While there are a variety of ways to compute distances between distributions, the L_p distance is of particular significance as evidenced in.[97]

Motivating Problem

We begin by presenting our motivating problem, which concerns (approximately) measuring the distance between the product and joint distributions of two random variables. That is, we attempt to quantify how close two random variables X and Y over a universe $[n] = \{1, \ldots, n\}$ are to being independent. There are many ways to measure the distance between distributions, but we focus on the L_1 distance. Recall that two random variables X and Y are independent if, for every i and j, we have $\Pr[X = i \wedge Y = j] = \Pr[X = i] \Pr[Y = j]$. In our model, we have a data stream D which is presented as a sequence of m pairs $d_1 = (i_1, j_1), d_2 = (i_2, j_2), \ldots, d_m = (i_m, j_m)$. Each pair $d_k = (i_k, j_k)$ consists of two integers taken from the universe $[n]$.

Intuitively, we imagine that the two random variables X and Y over the universe $[n]$ generate these pairs, and in particular, the frequencies of each pair (i, j) define an empirical joint distribution, which is the fraction of pairs that equal (i, j). At the same time, the stream also defines the empirical marginal distributions $\Pr[X = i], \Pr[Y = j]$, namely the fraction of pairs of the form $(i, \cdot)$ and $(\cdot, j)$, respectively. We note that, even if the pairs are actually generated from two independent sources, it may not be the case that the empirical distributions reflect this fact, although for sufficiently long streams the joint distribution

should approach the product of the marginal distributions for each i and j. This fundamental problem has received considerable attention within the streaming community, including the works of.[37,94]

Problem 1. *Let X and Y be two random variables defined by the stream of m pairs $d_1 = (i_1, j_1), \ldots, d_m = (i_m, j_m)$, where each $i_k, j_k \in [n]$ for all k. Define the frequencies $f_i = |\{k : d_k = (i, \cdot)\}|$ and $f_j = |\{k : d_k = (\cdot, j)\}|$ (i.e., the frequency with which i appears in the first coordinate and j appears in the second coordinate, respectively). Moreover, let $f_{ij} = |\{k : d_k = (i, j)\}|$ be the frequency with which the pair (i, j) appears in the stream. This naturally defines the joint distribution $Pr[X = i \wedge Y = j] = \frac{f_{ij}}{m}$ and the product of the marginal distributions $Pr[X = i]Pr[Y = j] = \frac{f_i f_j}{m^2}$. The L_1 distance between the product and joint distributions is given by:*

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left|\frac{f_{ij}}{m} - \frac{f_i f_j}{m^2}\right|.$$

If X and Y are independent, we should expect this sum to be close to 0, assuming the stream is sufficiently long. As a generalization to this problem, we can view the n^2 values which appear in the summation as being implicitly represented via an $n \times n$ matrix, where the $(i, j)^{th}$ entry is given by $\left|\frac{f_{ij}}{m} - \frac{f_i f_j}{m^2}\right|$. For the motivating problem, this matrix is given implicitly as it is not given up front and changes over time according to the data stream (each new pair in the stream may change a particular entry in the matrix). However, one can imagine settings in which these entries are defined through other means. In practice, we may

still be interested in computing approximate statistics over such implicitly defined matrices.

Contributions and Techniques

Our main contributions in this paper make progress on two important problems:

1. For any subadditive, even Hadamard function g where g is non-negative and $g(0) = 0$, given an implicitly defined $n \times n$ matrix A with entries a_{ij}, let $g[A]$ be the matrix where the $(i,j)^{th}$ entry is $g(a_{ij})$. We are the first to provide a general reduction framework for approximating $\|g[A]\|_1 = \sum_{i=1} \sum_{j=1}^{n} g(a_{ij})$ to within a $(1 \pm \epsilon)$-factor with constant success probability. More formally, suppose we have two blackbox algorithms with the following guarantees. One blackbox algorithm operates over the implicit matrix A and provides a very good ($\approx 1 \pm \epsilon$) approximation to $\|g[JA]\|_1 = \sum_{j=1}^{n} g(\sum_{i=1}^{n} a_{ij})$ except with inverse polylogarithmic probability, where $J = (1, \ldots, 1)$ is the row vector of dimension n with every entry equal to 1. The second blackbox algorithm operates over the implicit matrix A and solves the problem we wish to solve (i.e., approximating $\|g[A]\|_1$) with constant success probability, although it does so with a multiplicative approximation ratio of $r(n)$ (which may be worse than $(1 \pm \epsilon)$ in general). We show how to use these two blackbox algorithms and construct an algorithm that achieves a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$. If S_1, S_2 denote the space used by the first and second blackbox algorithms, respectively, then our algorithm uses space $O\left(\frac{r^4(n) log^8(n)}{\epsilon^5} \cdot (\log^3(n) + S_1 + \log(n) \cdot S_2)\right)$. We state this formally in Theorem 4.2.1.

2. Given the contribution above, it follows that setting $g(x) = |x|$ solves Problem 1, namely the problem of measuring how close two random variables are to being independent, as long as such blackbox algorithms exist. In particular, the work of Indyk[97] provides us with the first blackbox algorithm, and the work of[94] provides us with the second blackbox algorithm for this choice of g. Combining these results, we improve over the previous state of the art result of Braverman and Ostrovsky[37] and give improved bounds for measuring independence of random variables in the streaming model by reducing the space usage from $O\left((\frac{\log(nm)}{\epsilon})^{1024}\right)$ to $O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2(\frac{nm}{\epsilon})\right)$ (see Table 5.3).

Previous Work	L_1 approximation	Memory
IM08[94]	$\log(n)$	$O\left(\frac{1}{\epsilon^2}\log\left(\frac{nm}{\epsilon}\right)\log\left(\frac{m}{\epsilon}\right)\right)$
BO10[137]	$(1\pm\epsilon)$	$O\left(\left(\frac{\log(nm)}{\epsilon}\right)^{1024}\right)$
Our Result	$(1\pm\epsilon)$	$O\left(\frac{1}{\epsilon^7}\log^{12}(n)\log^2\left(\frac{nm}{\epsilon}\right)\right)$.

Table 4.1: Comparing Approximation Ratios and Space Complexity

Examples of such Hadamard functions which are subadditive, even, non-negative, and zero at the origin include $g(x) = |x|^p$, for any $0 < p \leq 1$. Note that our reduction in the first item can only be applied to solve the problem of approximating $\|g[A]\|_1$ if such blackbox

[1]The paper of[37] provides a general bound for the L_1 distance for k-tuples, but we provide analysis for pairs of elements, $k = 2$, in this paper. The bound in the table is for $k = 2$.

algorithms exist, but for some functions g this may not be the case. As a direct example of the tools we present, we give a reduction for computing the L_p distance for $0 < p < 1$ between the joint and product distributions in the streaming model (as this function is even and subadditive). However, to the best of our knowledge, such blackbox algorithms do not exist for computing the L_p distance. Thus, as a corollary to our main result, the construction of such blackbox algorithms that are space efficient would immediately yield an algorithm that measures independence according to the L_p distance that is also space efficient.

Our techniques leverage concepts provided in[37,94] and manipulates them to allow them to be combined with the Recursive Sketches data structure[98] to gain a large improvement compared to existing bounds. Note that we cannot use standard linear sketching techniques because the entries of the matrix are given implicitly. Moreover, the sketch of Indyk and McGregor[94] is pairwise independent, but not 4-wise independent. Therefore, we cannot apply the sketches of[14,94] directly. We first present an algorithm, independent of the streaming model, for finding heavy rows of a matrix norm given an arbitrary even subadditive Hadamard function g. In order to do this, we first prove a key theorem regarding such Hadamard functions g which states that the quantity $\|g[JA]\|_1 = \sum_{j=1}^{n} g(\sum_{i=1}^{n} a_{ij})$ is a $(1 \pm \epsilon)$-approximation to the heavy row of the matrix $g[A]$ (if it exists). With this in mind, we show how to use the blackbox algorithm that yields an $r(n)$-approximation to $\|g[A]\|_1$ in order to identify when heavy rows exist in the matrix, and then use the other blackbox algorithm to obtain a $(1 \pm \epsilon)$-approximation of $\|g[JA]\|_1$ (which is in turn a $(1 \pm \epsilon)$-approximation to the heavy row, as just mentioned). These ideas form the foundation

of our algorithm for approximating heavy rows. We then apply the Recursive Sum algorithm from[98] on top of our heavy rows algorithm to obtain our main result.

4.1.1 Related Work

In their seminal 1996 paper Alon, Matias and Szegedy[14] provided an optimal space approximation for L_2. A key technical requirement of the sketch is the assumption of 4-wise independent random variables. This technique is the building block for measuring the independence of data streams using L_2 distances as well.

The problems of efficiently testing pairwise, or k-wise, independence were considered by Alon, Andoni, Kaufman, Matulef, Rubinfeld and Xie;[99] Alon, Goldreich and Mansour;[100] Batu, Fortnow, Fischer, Kumar, Rubinfeld and White;[101] Batu, Kumar and Rubinfeld;[102] Batu, Fortnow, Rubinfield, Smith and White[103] and.[104] They addressed the problem of minimizing the number of samples needed to obtain a sufficient approximation, when the joint distribution is accessible through a sampling procedure.

In their 2008 work, Indyk and McGregor[94] provided exciting results for identifying the correlation of two streams, providing an optimal bound for determining the L_2 distance between the product and joint distributions of two random variables.

In addition to the L_2 result, Indyk and McGregor presented a $\log(n)$-approximation for the L_1 distance. This bound was improved to a $(1 \pm \epsilon)$-approximation in the work of Braverman and Ostrovsky[37] in which they provided a bound of $O(\frac{1}{\epsilon^{1024}} \log^{1024}(nm))$ for pairs of elements. Further, they gave bounds for the comparison of multiple streaming

vectors and determining k-wise relationships for L_1 distance. Additionally, in[105] Braverman et al. expanded the work of[94] to k dimensions for L_2. While our paper only addresses computation on matrices resulting from pairwise comparison ($k = 2$), we believe the techniques presented here can be expanded to tensors, (i.e., when stream elements are k-tuples), similarly to.[105] Recently, McGregor and Vu[106] studied a related problem regarding Bayesian networks in the streaming model.

Statistical Distance, L_1, is one of the most fundamental metrics for measuring the similarity of two distributions. It has been the metric of choice in many of the above testing papers, as well as others such as Rubinfeld and Servedio;[107] Sahai and Vadhan.[108] As such, a main focus of this work is improving bounds for this measure in the streaming model.

4.2 Problem Definition and Notation

In this paper we focus on the problem of approximating even, subadditive, non-negative Hadamard functions which are zero at the origin on implicitly defined matrices (e.g., the streaming model implicitly defines matrices for us in the context of measuring independence). The main problem we study in this paper is the following:

Problem 2. *Let g be any even, subadditive, non-negative Hadamard function such that $g(0) = 0$. Given any implicit matrix A, for any $\epsilon > 0$, $\delta > 0$, output a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$ except with probability δ.*

We now provide our main theorem, which solves Problem 2.

Theorem 4.2.1. *Let g be any even, subadditive, non-negative Hadamard function g where $g(0) = 0$, and fix $\epsilon > 0$. Moreover, let A be an arbitrary matrix, and J be the all 1's row vector $J = (1, \ldots, 1)$ of dimension n. Suppose there are two blackbox algorithms with the following properties:*

1. *Blackbox Algorithm 1, for all $\epsilon' > 0$, returns a $(1 \pm \epsilon')$-approximation of $\|g[JA]\|_1$, except with probability δ_1.*

2. *Blackbox Algorithm 2 returns an $r(n)$-approximation of $\|g[A]\|_1$, except with probability δ_2 (where $r(n)$ is a sufficiently large monotonically increasing function of n).*

Then, there exists an algorithm that returns a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$, except with constant probability. If Blackbox Algorithm 1 uses space $SPACE1(n, \delta_1, \epsilon')$, and Blackbox Algorithm 2 uses space $SPACE2(n, \delta_2)$, the resulting algorithm has space complexity

$$O\left(\frac{r^4(n)}{\epsilon^5}(\log^{10}(n) + \log^8(n)SPACE1(n, \delta_1, \epsilon') + \log^9(n)SPACE2(n, \delta_2))\right),$$

where $\epsilon' = \frac{\epsilon}{2}$, δ_1 is a small constant, and δ_2 is inverse polylogarithmic.

Note that we can reduce the constant failure probability to inverse polynomial failure probability via standard techniques, at the cost of increasing our space bound by a logarithmic factor. Observe that Problem 2 is a general case of Problem 4.1 where $g(x) = |x|$ (i.e., L_1

distance). In the streaming model, we receive matrix A implicitly, but we conceptualize the problem as if the matrix were given explicitly and then resolve this issue by assuming we have blackbox algorithms that operate over the implicit matrix.

We define our stream such that each element in the stream d_k is a pair of values (i, j):

Definition 4.2.1 (Stream)**.** *Let m, n be positive integers. A **stream** $D = D(m, n)$ is a sequence of length m, $d_1, d_2, \ldots, d_m$, where each entry is a pair of values in $\{1, \ldots, n\}$.*

Let $g : \mathbb{R} \to \mathbb{R}$ be a non-negative, subadditive, and even function where $g(0) = 0$. Frequently, we will need to discuss a matrix where g has been applied to every entry. We use the notations from[95] which are in turn based on notations from.[96]

Definition 4.2.2 (Hadamard Function)**.** *Given Matrix A of dimensions $n \times n$ a* **Hadamard function** *g takes as input a matrix A and is applied entrywise to every entry of the matrix. The output is matrix $g[A]$. Further, we note that the L_1 norm of $g[A]$ is equivalent to the value we aim to approximate, $\|g[A]\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} g(a_{ij})$.*

We frequently use hash functions in our analysis, we now specify some notation. We sometimes express a hash function H as a vector of values $\{h_1, h_2, ..., h_n\}$. Multiplication of hash functions, denoted $H' = HAD(H^a, H^b)$, is performed entrywise such that $\{h'_1 = h_1^a h_1^b, ..., h'_n = h_n^a h_n^b\}$.

We now define two additional matrices. All matrices in our definitions are of size $n \times n$, and all vectors are of size $1 \times n$. We denote by $[n]$ the set $\{1, \ldots, n\}$.

Definition 4.2.3 (Sampling Identity Matrix)**.** *Given a hash function $H : [n] \to \{0, 1\}$, let*

$h_i = H(i)$. *The* **Sampling Identity Matrix** I_H *with entries* b_{ij} *is defined as:*

$$I_H = \begin{cases} b_{ii} = h_i \\ \\ b_{ij} = 0 \textit{ for } i \neq j. \end{cases}$$

That is, the diagonal of I_H are the values of H. When we multiply matrix I_H by A, each row of $I_H A$ is either the zero vector (corresponding to $h_i = 0$) or the original row i in matrix A (corresponding to $h_i = 1$). We use the term "sampling" due to the fact that the hash functions we use throughout this paper are random, and hence which rows remain untouched is random. The same observations apply to columns when considering the matrix AI_H.

Definition 4.2.4 (Row Aggregation Vector)**.** *A* **Row Aggregation Vector** J *is a* $1 \times n$ *vector with all entries equal to* 1.

Thus, JA yields a vector V where each value v_j is $\sum_{i=1}^{n} a_{ij}$.

Black Box Algorithm 1. $(1 \pm \epsilon')$*-Approximation of* g *on a Row Aggregate Vector*

Input: Matrix A*, and hash function* H.

Output: $(1 \pm \epsilon')$*-approximation of* $\|g[JI_H A]\|_1$ *with probability* $(1 - \delta_1)$.

The space Black Box Algorithm 1 (BA1) uses is referred to as $SPACE1(n, \delta_1, \epsilon')$ in our analysis.

Black Box Algorithm 2. $r(n)$*-Approximation of* $\|g[I_H A]\|_1$

Input: Matrix A *and hash function* H.

Output: An $r(n)$*-approximation of* $\|g[I_H A]\|_1$ *with probability* $(1 - \delta_2)$.

The space Black Box Algorithm 2 ($BA2$) uses is referred to as $SPACE2(n, \delta_2)$ in our analysis.

Definition 4.2.5 (Complement Hash Function)**.** *For hash function $H : [n] \to \{0, 1\}$ define the* **Complement Hash Function** $\bar{H}$ *as $\bar{H}(i) = 1$ if and only if $H(i) = 0$.*

Definition 4.2.6 (Threshold Functions)**.** *We define two threshold functions, which we denote by $\rho(n, \epsilon) = \frac{r^4(n)}{\epsilon}$ and $\tau(n, \epsilon) = \frac{2r^2(n)}{O(\epsilon)}$.*

Definition 4.2.7 (Weight of a Row)**.** *The* **weight** *of row i in matrix A is given by $u_{A,i} = \sum_{j=1}^{n} a_{ij}$.*

Definition 4.2.8 (α-Heavy Rows)**.** *Row i is α-***heavy** *for $0 < \alpha < 1$ if $u_{A,i} > \alpha \|A\|_1$.*

Definition 4.2.9 (Key Row)**.** *We say row i is a* **Key Row** *if: $u_{A,i} > \rho(n, \epsilon)(\|A\|_1 - u_{A,i})$.*

While Definition 4.2.8 and Definition 4.2.9 are similar, we define them for convenience, as our algorithm works by first finding key rows and then building on top of this to find α-heavy rows. We note that, as long as $\rho(n, \epsilon) \geq 1$, a matrix can have at most one key row (since any matrix can have at most $\frac{1}{\alpha}$ α-heavy rows, and a key row is α-heavy for $\alpha = \frac{\rho(n,\epsilon)}{1+\rho(n,\epsilon)}$).

4.3 Subadditive Approximations

In this section we show that a $(1 \pm \epsilon)$-approximation of even, subadditive, non-negative Hadamard functions which are zero at the origin are preserved under row or column aggregations in the presence of sufficiently heavy rows or columns.

Theorem 4.3.1. *Let B be an $n \times n$ matrix and let $\epsilon \in (0,1)$ be a parameter. Recall that J is a row vector with all entries equal to 1. Let g be any even, subadditive, non-negative Hadamard function which satisfies $g(0) = 0$. Denote $u_i = \sum_{j=1}^{n} g(b_{ij})$, and thus $\|g[B]\|_1 = \sum_{i=1}^{n} u_i$. If there is a row h such that $u_h \geq (1 - \frac{\epsilon}{2})\|g[B]\|_1$ then $|u_h - \|g[JB]\|_1| \leq \epsilon \|g[JB]\|_1$.*

Proof. Denote $V = JB$. Without loss of generality assume u_1 is the row such that $u_1 \geq (1 - \frac{\epsilon}{2})\|g[B]\|_1$. By subadditivity of g: $\|g[V]\|_1 \leq \|g[B]\|_1 \leq \frac{1}{1-\frac{\epsilon}{2}} u_1 \leq (1+\epsilon)u_1$. Further, we have $b_{1j} = (\sum_{i=1}^{n} b_{ij} - \sum_{i=2}^{n} b_{ij})$. Since g is even and subadditive, and such functions are non-negative, we have $g(b_{1j}) \leq g\left(\sum_{i=1}^{n} b_{ij}\right) + \sum_{i=2}^{n} g(b_{ij})$. Rearranging and summing over j, we get: $\sum_{j=1}^{n} g\left(\sum_{i=1}^{n} b_{ij}\right) \geq \sum_{j=1}^{n} \left(g(b_{1,j}) - \sum_{i=2}^{n} g(b_{ij})\right)$.

Therefore:

$$\|g[V]\|_1 = \sum_{j=1}^{n} g\left(\sum_{i=1}^{n} b_{ij}\right) \geq \sum_{j=1}^{n} \left(g(b_{1,j}) - \left(\sum_{i=2}^{n} g(b_{ij})\right)\right) = u_1 - (\|g[B]\|_1 - u_1).$$

Finally:

$$\|g[V]\|_1 \geq u_1 - (\|g[B]\|_1 - u_1) = 2u_1 - \|g[B]\|_1 \geq u_1(2 - \frac{1}{1-\frac{\epsilon}{2}}) = u_1 \frac{1-\epsilon}{1-\frac{\epsilon}{2}} \geq u_1(1-\epsilon).$$

□

4.4 Algorithm for Finding Key Rows

Definition 4.4.1 (Algorithm for Finding Key Rows)**.**

Input: Matrix A and Sampling Identity Matrix I_H generated from hash function H.

Output: Pair (a, b), where the following holds for a, b, and the matrix $W = I_H A$:

1. *The pair is either $(a, b) = (-1, 0)$ or $(a, b) = (i, \tilde{u}_{W,i})$. Here, $\tilde{u}_{W,i}$ is a $(1 \pm \epsilon)$-approximation to $u_{W,i}$ and the index i is the correct corresponding row.*

2. *If there is a key row i_0 for the matrix W, then $a = i_0$.*

Before describing the algorithm and proving its correctness, we prove the following useful lemma in Appendix 4.6.1.

Lemma 4.4.1. *Let $U = (u_1, \ldots, u_n)$ be a vector with non-negative entries of dimension n and let H' be a pairwise independent hash function where $H' : [n] \to \{0, 1\}$ and $P[H'(i) = 1] = P[H'(i) = 0] = \frac{1}{2}$. Denote by $\bar{H}'$ the hash function defined by $\bar{H}'(i) = 1 - H'(i)$. Let $X = \sum_i H'(i)u_i$ and $Y = \sum_i \bar{H}'(i)u_i$. If there is no $\frac{1}{16}$-heavy element with respect to U, then:*

$$\Pr\left[\left(X \leq \frac{1}{4} \cdot \|U\|_1\right) \cup \left(Y \leq \frac{1}{4} \cdot \|U\|_1\right)\right] \leq \frac{1}{4}.$$

Theorem 4.4.1. *If there exist two black box algorithms as specified in Black Box Algorithms 1 and 2, then there exists an algorithm that satisfies the requirements in Definition 4.4.1 with high probability.*

Algorithm 3 Algorithm Find-Key-Row

The algorithm takes as input matrix A and hash function $H : [n] \to \{0, 1\}$

for ℓ = 1 to $N = O(\log n)$ **do**
Generate a pairwise independent, uniform hash function $H_\ell : [n] \to \{0, 1\}$
Let $T_1 = HAD(H, H_\ell)$, $T_0 = HAD(H, \bar{H}_\ell)$
Let $y_1 = BA2(A, T_1)$, $y_0 = BA2(A, T_0)$ ($BA2$ is run with constant failure probability δ_2)
if $y_0 \geq \tau(n, \epsilon) \cdot y_1$ **then**
$b_\ell = 0$
else if $y_1 \geq \tau(n, \epsilon) \cdot y_0$ **then**
$b_\ell = 1$
else
$b_\ell = 2$
if $|\{\ell : b_\ell = 2\}| \geq \frac{2}{3}n$ **then**
Return $(-1, 0)$
else
if there is a row i such that i satisfies $|\{\ell : H_\ell(i) = b_\ell\}| \geq \frac{3}{4} \cdot N$ **then**
Return $(i, BA1(A, H))$ ($BA1$ is run with $\epsilon' = \frac{\epsilon}{2}$ and δ_1 is set to be inverse polylogarithmic)
else
Return $(-1, 0)$

Proof. We will prove that Algorithm 3 fits the description of Definition 4.4.1. Using standard methods such as in,[60] we have a loop that runs in parallel $O(\log(n))$ times so that we can find the index of a heavy element and return it, if there is one. To prove this theorem, we consider the following three exhaustive and disjoint cases regarding the matrix $g[I_H A]$ (recall that $H : [n] \to \{0, 1\}$):

1. The matrix has a key row (note that a matrix always has at most one key row).

2. The matrix has no α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$.

3. The matrix has an α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$, but there is no key row.

We prove that the algorithm is correct in each case in Lemmas 4.7.1, 4.7.2, and 4.7.3,

respectively. Due to page length constraints, these proofs can be found in Appendix 4.7. □

With the proof of these three cases, we are done proving that Algorithm 3 performs correctly. We now analyze the space bound for Algorithm 3.

Lemma 4.4.2. *Algorithm 3 uses* $O\left(SPACE1(n, \delta_1, \frac{\epsilon}{2}) + \log(n)(\log^2(n) + SPACE2(n, \delta_2))\right)$ *bits of memory, where* δ_1 *is inverse polylogarithmic and* δ_2 *is a constant.*

Proof. Note that, in order for our algorithm to succeed, we run $BA1$ with an error parameter of $\epsilon' = \frac{\epsilon}{2}$ and a failure probability parameter δ_1 which is inverse polylogarithmic. Moreover, we run $BA2$ with a constant failure probability. We also require a number of random bits bounded by $O(\log^2(n))$ for generating each hash function H_ℓ, as well as the space required to run $BA2$ in each iteration of the loop. Since there are $O(\log n)$ parallel iterations, this gives the lemma. □

4.4.1 Algorithm for Finding All α-Heavy Rows

Algorithm 3 only guarantees that we return key rows. Given a matrix A, we now show that this algorithm can be used as a subroutine to find all α-heavy rows i with respect to the matrix $g[A]$ with high probability, along with a $(1 \pm \epsilon)$-approximation to the row weights $u_{g[A],i}$ for all i. In order to do this, we apply an additional hash function $H : [n] \to [\tau]$ which essentially maps rows of the matrix to some number of buckets τ (i.e., each bucket corresponds to a set of sampled rows based on H), and then run Algorithm 3 for each bucket. The intuition for why the algorithm works is that any α-heavy row i in the original matrix

A is likely to be a key row for the matrix in the corresponding bucket to which row i is mapped. Note that, eventually, we find α-heavy rows for $\alpha = \frac{\epsilon^2}{\log^3 n}$. The algorithm sets $\tau = O\left(\frac{\rho(n,\epsilon)\log(n)}{\alpha^2}\right)$ and is given below.

Algorithm 4 Algorithm Find-Heavy-Rows

This algorithm takes as input a matrix A and a value $0 < \alpha < 1$.

Generate a pairwise independent hash function $H : [n] \to [\tau]$, where $\tau = O\left(\frac{\rho(n,\epsilon)\log(n)}{\alpha^2}\right)$

for $k = 1$ to τ **do**

Let $H_k : [n] \to \{0, 1\}$ be the function defined by $H_k(i) = 1 \iff H(i) = k$

Let $C_k =$ Find-Key-Row(A, H_k)

Return $\{C_k : C_k \neq (-1, 0)\}$

Theorem 4.4.2. *Algorithm 4 outputs a set of pairs $Q = \{(i_1, a_1), \ldots, (i_t, a_t)\}$ for $t \leq \tau$ which satisfy the following properties, except with probability $\frac{1}{\log n}$:*

1. $\forall j \in [t] : (1 - \epsilon)u_{g[A],i_j} \leq a_j \leq (1 \pm \epsilon)u_{g[A],i_j}$.
2. $\forall i \in [n]$*: If row i is α-heavy with respect to the matrix $g[A]$, then $\exists j \in [t]$ such that $i_j = i$ (for any $0 < \alpha < 1$).*

Proof. First, the number of pairs output by Algorithm 4 is at most the number of buckets, which equals τ. Now, the first property is true due to the fact that Algorithm 3 has a high success probability. In particular, as long as the failure probability is at most $\frac{1}{\tau \cdot \log^c(n)}$ for some constant c (which we ensure), then by union bound the probability that there exists a pair $(i_j, a_j) \in Q$ such that a_j is not a $(1 \pm \epsilon)$-approximation to $u_{g[A],i_j}$ is at most inverse polylogarithmic.

Now, to ensure the second item, we need to argue that every α-heavy row gets mapped to its own bucket with high probability, since if there is a collision the algorithm cannot find all α-heavy rows. Moreover, we must argue that for each α-heavy row i with respect to the matrix $g[A]$, if i is mapped to bucket k by H, then row i is actually a key row in the corresponding sampled matrix $g[A_k]$ (for ease of notation, we write A_k to denote the matrix $H_k A_k$). More formally, suppose row i is α-heavy. Then the algorithm must guarantee with high probability that, if $H(i) = k$, then row i is a key row in the matrix $g[A_k]$. If we prove these two properties, then the theorem holds (since Algorithm 3 outputs a key row with high probability, if there is one).

Observe that there must be at most $\frac{1}{\alpha}$ rows which are α-heavy. In particular, let R be the set of α heavy rows, and assume towards a contradiction that $|R| > \frac{1}{\alpha}$. Then we have:

$$\|g[A]\|_1 \geq \sum_{i \in R} u_{g[A],i} \geq \sum_{i \in R} \alpha \|g[A]\|_1 = \alpha \cdot \|g[A]\|_1 \cdot |R| > \|g[A]\|_1,$$

which is a contradiction. Hence, we seek to upper bound the probability of a collision when throwing $\frac{1}{\alpha}$ balls into τ bins. By a Birthday paradox argument, this happens with probability at most $\frac{1}{2 \cdot \tau \cdot \alpha^2}$, which can be upper bounded as follows:

$$\frac{1}{2\tau\alpha^2} \leq \frac{\alpha^2}{2\alpha^2 \rho(n,\epsilon)\log(n)} = \frac{1}{2\rho(n,\epsilon)\log(n)} = \frac{\epsilon}{2r^4(n)\log(n)},$$

which is inverse polylogarithmically small.

Now, we argue that every α-heavy row i for the matrix $g[A]$ is mapped to a sampled

matrix such that i is a key row in the sampled matrix with high probability. In particular, suppose $H(i) = k$, implying that row i is mapped to bucket k. For $\ell \neq i$, let X_ℓ be the indicator random variable which is 1 if and only if row ℓ is mapped to the same bucket as i, namely $H(\ell) = k$ (i.e., $X_\ell = 1$ means the sampled matrix $g[A_k]$ contains row i and row ℓ). If row i is not a key row for the matrix $g[A_k]$, this means that $u_{g[A_k],i} \leq \rho(n,\epsilon)(\|g[A_k]\|_1 - u_{g[A_k],i})$. Observe that, if row i is mapped to bucket k, then we have $u_{g[A_k],i} = u_{g[A],i}$. Hence, the the probability that row i is not a key row for the sampled matrix $g[A_k]$ (assuming row i is mapped to bucket k) can be expressed as $\Pr[u_{g[A],i} \leq \rho(n,\epsilon)(\|g[A_k]\|_1 - u_{g[A],i}) | H(i) = k]$. By pairwise independence of H, and by Markov's inequality, we can write:

$$\begin{aligned}
&\Pr\left[u_{g[A],i} \leq \rho(n,\epsilon)(\|g[A_k]\|_1 - u_{g[A],i}) \;\middle|\; H(i) = k\right] \\
&\qquad = \Pr\left[u_{g[A],i} \leq \rho(n,\epsilon)\sum_{\ell \neq i} u_{g[A],\ell} X_\ell \;\middle|\; H(i) = k\right] \\
&\qquad = \Pr\left[u_{g[A],i} \leq \rho(n,\epsilon)\sum_{\ell \neq i} u_{g[A],\ell} X_\ell\right] \\
&\qquad = \Pr\left[\sum_{\ell \neq i} u_{g[A],\ell} X_\ell \geq \frac{u_{g[A],i}}{\rho(n,\epsilon)}\right] \leq \frac{\rho(n,\epsilon)\mathbb{E}\left[\sum_{\ell \neq i} u_{g[A],\ell} X_\ell\right]}{u_{g[A],i}} \\
&\qquad = \frac{\rho(n,\epsilon)\sum_{\ell \neq i} u_{g[A],\ell}}{\tau \cdot u_{g[A],i}} \leq \frac{\rho(n,\epsilon)\|g[A]\|_1}{\alpha\tau\|g[A]\|_1} = \frac{\alpha^2\rho(n,\epsilon)}{4\alpha \cdot \rho(n,\epsilon)\log(n)} \leq \frac{\alpha}{4\log(n)}.
\end{aligned}$$

Here, we choose $\tau = \frac{4\rho(n,\epsilon)\log(n)}{\alpha^2}$, and get that the probability that a particular α-heavy row i is not a key row in its corresponding sampled matrix is at most $\frac{\alpha}{4\log(n)}$. Since there are at most $\frac{1}{\alpha}$ rows which are α-heavy, by union bound the probability that there exists an α-heavy row that is not a key row in its sampled matrix is at most $\frac{1}{4\log(n)}$.

Thus, in all, the probability that at least one bad event happens (i.e., there exists a pair (i_j, a_j) such that a_j is not a good approximation to $u_{g[A],i_j}$, there is a collision between α-heavy rows, or an α-heavy row is not a key row in its corresponding sampled matrix) is at most $\frac{1}{\log(n)}$. This gives the theorem. □

4.4.2 Sum from α-Heavy Rows

We now have an algorithm that is able to find all α-heavy rows for $\alpha = \frac{\epsilon^2}{\log^3 n}$, except with probability $\frac{1}{\log n}$. In the language of,[98] by Theorem 4.4.2, our α-heavy rows algorithm outputs an (α, ϵ)-cover with respect to the vector $(u_{g[A],1}, u_{g[A],2}, \ldots, u_{g[A],n})$ except with probability $\frac{1}{\log n}$, where $\epsilon > 0$ and $\alpha > 0$. Hence, we can apply the Recursive Sum algorithm from[98] (see Appendix 4.7.1 for the formal definition of an (α, ϵ)-cover, along with the Recursive Sum algorithm) to get a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$. Note that the Recursive Sum algorithm needs $\alpha = \frac{\epsilon^2}{\log^3 n}$ and a failure probability of at most $\frac{1}{\log n}$, which we provide. Hence, we get the following theorem.

Theorem 4.4.3. *The Recursive Sum Algorithm, using Algorithm 4 as a subroutine, returns a $(1 \pm \epsilon)$-approximation of $\|g[A]\|_1$.*

4.4.3 Space Bounds

Lemma 4.4.3. *Recursive Sum, using Algorithm 4 as a subroutine as described in Section 4.4.2, uses the following amount of memory, where $\epsilon' = \frac{\epsilon}{2}$, δ_1 is inverse polylogarithmic,*

and δ_2 is a small constant:

$$O\left(\frac{r^4(n)}{\epsilon^5}(\log^{10}(n)+\log^8(n)SPACE1(n,\delta_1,\epsilon')+\log^9(n)SPACE2(n,\delta_2))\right).$$

Proof. The final algorithm uses the space bound from Lemma 4.4.2, multiplied by $\tau = O\left(\frac{\rho(n,\epsilon)\log(n)}{\alpha^2}\right)$, where $\alpha = \frac{\epsilon^2}{\phi^3}$, $\phi = O(\log n)$, and $\rho(n,\epsilon) = \frac{r^4(n)}{\epsilon}$. This gives $\tau = \frac{1}{\epsilon^5}r^4(n)\log^7(n)$ to account for the splitting required to find α-heavy rows in Section 4.4.1. Finally, a multiplicative cost of $\log(n)$ is needed for Recursive Sum, giving the final bound.

□

4.5 Applications

We now apply our algorithm to the problem of determining the L_1 distance between joint and product distributions as described in Problem 4.1.

Space Bounds for Determining L_1 Independence

Given an $n \times n$ matrix A with entries $a_{ij} = \frac{f_{ij}}{m} - \frac{f_i f_j}{m}$, we have provided a method to approximate the value $\|g[A]\|_1$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} g\left(\frac{f_{ij}}{m} - \frac{f_i f_j}{m}\right).$$

Let g be the L_1 distance, namely $g(x) = |x|$ (hence, the $(i,j)^{th}$ entry in $g[A]$ is given by $|\frac{f_{ij}}{m} - \frac{f_i f_j}{m}|$). We now state explicitly which blackbox algorithms we use:

- Let Black Box Algorithm 1 ($BA1$) be the $(1 \pm \epsilon)$-approximation of L_1 for vectors from.[97] The space of this algorithm is upper bounded by the number of random bits required and uses $O(\log(\frac{nm}{\delta\epsilon}) \log(\frac{m}{\delta\epsilon}) \log(\frac{1}{\delta})\epsilon^{-2})$ bits of memory.

- Let Black Box Algorithm 2 ($BA2$) be the $r(n)$-approximation, using the L_1 sketch of the distance between joint and product distributions from.[94] This algorithm does not have a precise polylogarithmic bound provided, but we compute that it is upper bounded by the random bits required to generate the Cauchy random variables similarly to $BA1$. This algorithm requires $O(\log(\frac{nm}{\delta\epsilon}) \log(\frac{m}{\delta\epsilon}) \log(\frac{1}{\delta})\epsilon^{-2})$ bits of memory.

These two algorithms match the definitions given in Section 4.2, thus we are able to give a bound of $O(\frac{1}{\epsilon^7} \log^{14}(n) \log^2(\frac{nm}{\epsilon}))$ on the space our algorithm requires. We can improve this slightly as follows.

Corollary 1. *Due to the nature of the truncated Cauchy distribution (see[94]), we can further improve our space bound to* $O\left(\frac{1}{\epsilon^7} \log^{12}(n) \log^2(\frac{nm}{\epsilon})\right)$.

Proof. Due to the constant lower bound on the approximation of L_1, instead of $\frac{1}{r^2(n)} \leq \|g[W]\|_1 \leq r^2(n)$, we get $C \leq \|g[W]\|_1 \leq log^2(n)$ for some constant C. As the space cost from dividing the matrix into submatrices as shown in Section 4.4.1 directly depends on these bounds, we only pay an $O(r^2(n))$ multiplicative factor instead of an $O(r^4(n))$ multiplicative factor and achieve a bound of $O\left(\frac{1}{\epsilon^7} \log^{12}(n) \log^2(\frac{nm}{\epsilon})\right)$. □

4.6 Appendix

4.6.1 Proof of of Lemma 1

Proof. Note that we always have the equality $X+Y = \sum_i H'(i)u_i + \bar{H}'(i)u_i = \sum_i H'(i)u_i + (1 - H'(i))u_i = \|U\|_1$, and moreover $\mathbb{E}[X] = \sum_i u_i \mathbb{E}[H'(i)] = \frac{1}{2} \cdot \|U\|_1$. Also, observe that

$$\begin{aligned} Var[X] &= \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \\ &= \sum_i \mathbb{E}[(H'(i))^2]u_i^2 + \sum_{i \neq j} \mathbb{E}[H'(i)H'(j)]u_i u_j - \frac{1}{4} \cdot \|U\|_1^2 \\ &= \frac{1}{2}\sum_i u_i^2 + \frac{1}{4}\sum_{i \neq j} u_i u_j - \frac{1}{4}\left(\sum_i u_i^2 + \sum_{i \neq j} u_i u_j\right) = \frac{1}{4}\sum_i u_i^2. \end{aligned}$$

Using the fact that there is no $\frac{1}{16}$-heavy element with respect to U, which implies that $u_i \leq \frac{1}{16} \cdot \|U\|_1$ for all i, we have:

$$Var[X] = \frac{1}{4}\sum_i u_i^2 \leq \frac{\|U\|_1}{64}\sum_i u_i = \frac{\|U\|_1^2}{64}.$$

Now we can apply Chebyshev's inequality to obtain:

$$\begin{aligned} \Pr\left[\left(X \leq \frac{1}{4} \cdot \|U\|_1\right) \cup \left(Y \leq \frac{1}{4} \cdot \|U\|_1\right)\right] &= \Pr\left[|X - \mathbb{E}[X]| \geq \frac{\|U\|_1}{4}\right] \\ &\leq \frac{16 \cdot Var[X]}{\|U\|_1^2} \leq \frac{16 \cdot \|U\|_1^2}{64 \cdot \|U\|_1^2} = \frac{1}{4}. \end{aligned}$$

□

4.7 Proof of Correctness of Algorithm 1

Throughout the lemmas, we imagine that the hash function $H : [n] \to \{0, 1\}$ is fixed, and hence the matrix $g[I_H A]$ is fixed. All randomness is taken over the pairwise independent hash functions H_ℓ that are generated in parallel, along with both blackbox algorithms.

To ease the notation, we define

$$W = I_H A, W_1 = I_{T_1} A, \text{ and } W_0 = I_{T_2} A$$

(recall the notation from Algorithm 3 that $T_1 = HAD(H, H_\ell)$ and $T_0 = HAD(H, \bar{H}_\ell)$). Finally, for each row i in the matrix $g[W]$, we define the shorthand notation $u_i = u_{g[W],i}$.

Lemma 4.7.1. *If the matrix $g[I_H A]$ has a key row, Algorithm 3 correctly returns the index of the row and a $(1 \pm \epsilon)$-approximation of the weight of the key row except with inverse polylogarithmic probability.*

Proof. Suppose the matrix $g[I_H A]$ has a key row, and let i_0 be the index of this row. We prove that we return a good approximation of $u_{g[W],i_0}$ with high probability. In particular, we first argue that, for a fixed iteration ℓ of the loop, we have the property that b_ℓ equals $H_\ell(i_0)$, and moreover this holds with certainty. We assume without loss of generality that $H_\ell(i_0) = 1$ (the case when $H_\ell(i_0) = 0$ is symmetric). In particular, this implies that the key row i_0 appears in the matrix $g[W_1]$.

By definition of $BA2$, the following holds for $y_1 = BA2(A, T_1)$ and $y_0 = BA2(A, T_0)$, except with probability $2\delta_2$ (where δ_2 is the failure probability of $BA2$):

$$y_1 \geq \frac{\|g[W_1]\|_1}{r(n)} \text{ and } y_0 \leq \|g[W_0]\|_1 r(n).$$

We have the following set of inequalities:

$$\|g[W_1]\|_1 \geq u_{i_0} > \rho(n,\epsilon)(\|g[W]\|_1 - u_{i_0}) \geq \rho(n,\epsilon)\|g[W_0]\|_1,$$

where the first inequality follows since g is non-negative and the key row i_0 appears in the matrix $g[W_1]$ (and hence the L_1-norm of $g[W_1]$ is at least u_{i_0} since it includes the row i_0), the second inequality follows by definition of i_0 being a key row for the matrix W, and the last inequality follows since the entries in row i_0 of the matrix W_0 are all zero (as $H_\ell(i_0) = 1$) and the remaining rows of W_0 are sampled from W, along with the facts that g is non-negative and $g(0) = 0$.

Substituting for $\rho(n,\epsilon)$, and using the fact that y_1 and y_0 are good approximations for $\|g[W_1]\|_1$ and $\|g[W_0]\|_1$ (respectively), except with probability $2\delta_2$, we get:

$$y_1 \geq \frac{\|g[W_1]\|_1}{r(n)} > \frac{r^3(n)}{\epsilon} \cdot \|g[W_0]\|_1 \geq \frac{r^2(n)}{\epsilon} \cdot y_0 \geq \tau(n,\epsilon) \cdot y_0,$$

and thus in this iteration of the loop we have $b_\ell = 1$ except with probability $2\delta_2$ (in the case that $H_\ell(i_0) = 0$, it is easy to verify by a similar argument that $y_0 \geq \tau(n,\epsilon) \cdot y_1$, and hence

we have $b_\ell = 0$). Hence, for the row i_0, we have the property that $b_\ell = H_\ell(i_0)$ for a fixed ℓ, except with probability $2\delta_2$. By the Chernoff bound, as long as δ_2 is a sufficiently small constant, we have $b_\ell = H_\ell(i_0)$ for at least a $\frac{3}{4}$-fraction of iterations ℓ, except with inverse polynomial probability. The only issue to consider is the case that there exists another row $i \neq i_0$ with the same property, namely $b_\ell = H_\ell(i)$ for a large fraction of iterations ℓ. However, if $b_\ell = H_\ell(i)$, it must be that at least one of y_1, y_0 is a bad approximation or $H_\ell(i) = H_\ell(i_0)$, which happens with probability at most $2\delta_2 + \frac{1}{2}$. Therefore, by the Chernoff bound, the probability that this happens for at least a $\frac{3}{4}$-fraction of iterations ℓ is at most $\frac{1}{2^{O(\log n)}}$, which is inverse polynomially small. By applying the union bound, the probability that there exists such a row is at most $\frac{n-1}{2^{O(\log n)}}$, which is at most an inverse polynomial. Hence, in this case, the algorithm returns $(i_0, BA1(A, H))$ except with inverse polynomial probability.

We now argue that $\tilde{u}_{g[W],i_o} = BA1(A, H)$ is a $(1 \pm \epsilon)$-approximation of $u_{g[W],i_0}$, except with inverse polylogarithmic probability. By definition of $BA1$, which we run with an error parameter of $\epsilon' = \frac{\epsilon}{2}$, it returns a $\left(1 \pm \frac{\epsilon}{2}\right)$-approximation of $\|g[JW]\|_1$ except with inverse polylogarithmic probability, where $W = I_H A$. Moreover, since i_0 is a key row, we have:

$$u_{i_0} > \rho(n,\epsilon)(\|g[W]\|_1 - u_{i_0}) \Rightarrow u_{i_0} > \frac{\rho(n,\epsilon)\|g[W]\|_1}{1+\rho(n,\epsilon)} \geq \left(1 - \frac{\epsilon}{8}\right)\|g[W]\|_1,$$

where the last inequality follows as long as $r^4(n) \geq 8 - \epsilon$. This implies that i_0 is $\left(1 - \frac{\epsilon}{8}\right)$-heavy with respect to the matrix $g[W]$, and hence we can apply Theorem 4.3.1 to get

that:

$$(1 \pm \epsilon)u_{i_0} \geq \frac{\left(1+\frac{\epsilon}{2}\right)}{\left(1-\frac{\epsilon}{4}\right)}u_{i_0} \geq \left(1+\frac{\epsilon}{2}\right)\|g[JW]\|_1 \geq \tilde{u}_{g[W],i_0}$$
$$\geq \left(1-\frac{\epsilon}{2}\right)\|g[JW]\|_1 \geq \frac{\left(1-\frac{\epsilon}{2}\right)}{\left(1+\frac{\epsilon}{4}\right)}u_{i_0} \geq (1-\epsilon)u_{i_0},$$

where the first inequality holds for any $0 < \epsilon \leq 1$, the second inequality holds by Theorem 4.3.1, the third inequality holds since $\tilde{u}_{g[W],i_0}$ is a $\left(1 \pm \frac{\epsilon}{2}\right)$-approximation of $\|g[JW]\|_1$, and the rest hold for similar reasons. Hence, our algorithm returns a good approximation as long as $BA1$ succeeds. Noting that this happens except with inverse polylogarithmic probability gives the lemma. □

Lemma 4.7.2. *If the input matrix has no α-heavy row, where $\alpha = 1 - \frac{\epsilon}{8}$, then with high probability Algorithm 3 correctly returns $(-1, 0)$.*

Proof. In this case, we have no α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$, which implies that $u_i \leq \alpha\|g[W]\|_1 = \left(1 - \frac{\epsilon}{8}\right)\|g[W]\|_1$ for each row i in the matrix $g[W]$. In this case, we show the probability that Algorithm 3 returns a false positive is small. That is, with high probability, in each iteration ℓ of the loop the algorithm sets $b_\ell = 2$, and hence it returns $(-1, 0)$. We split this case into three additional disjoint and exhaustive subcases, defined as follows:

1. For each row i, we have $u_i \leq \frac{1}{16}\|g[W]\|_1$.

2. There exists a row i with $u_i > \frac{1}{16}\|g[W]\|_1$ and $\forall j \neq i$ we have $u_j \leq \frac{\epsilon}{128}u_i$.

3. There exist two distinct rows i, j where $u_i > \frac{1}{16}\|g[W]\|_1$ and $u_j > \frac{\epsilon}{128}u_i$.

We define $X = \sum_i h_i^\ell u_i$ and $Y = \sum_i \bar{h}_i^\ell u_i$, where $h_i^\ell = H_\ell(i)$ and $\bar{h}_i^\ell = \bar{H}_\ell(i)$. Hence, we have $X = \|g[W_1]\|_1$ and $Y = \|g[W_0]\|_1$, and moreover $X + Y = \|g[W]\|_1$ (recall that $g[W_1] = g[I_{T_1}A]$ and $g[W_0] = g[I_{T_0}A]$).

In the first subcase, where there is no $\frac{1}{16}$-heavy row, we can apply Lemma 4.4.1 to the vector $(u_1, \ldots, u_n)$ to get that:

$$\Pr\left[\left(X \leq \frac{\|g[W]\|_1}{4}\right) \cup \left(Y \leq \frac{\|g[W]\|_1}{4}\right)\right] \leq \frac{1}{4}.$$

By definition of $BA2$, the following holds for $y_1 = BA2(A, T_1)$ and $y_0 = BA2(A, T_0)$ except with probability $2\delta_2$, where δ_2 is the success probability of $BA2$:

$$\frac{\|g[W_1]\|_1}{r(n)} \leq y_1 \leq r(n)\|g[W_1]\|_1\,, \quad \frac{\|g[W_0]\|_1}{r(n)} \leq y_0 \leq r(n)\|g[W_0]\|_1.$$

Hence, except with probability $\frac{1}{4} + 2\delta_2$, we have the following constraints on y_0 and y_1:

$$y_0 \leq r(n)Y \leq r(n) \cdot \frac{3}{4} \cdot \|g[W]\|_1 \leq 3r(n)X \leq 3y_1 r^2(n) \leq \tau(n,\epsilon) \cdot y_1, \text{ and}$$
$$y_1 \leq r(n)X \leq r(n) \cdot \frac{3}{4} \cdot \|g[W]\|_1 \leq 3r(n)Y \leq 3y_0 r^2(n) \leq \tau(n,\epsilon) \cdot y_0,$$

in which case we set $b_\ell = 2$. If δ_2 is some small constant, say $\delta_2 \leq \frac{1}{32}$, then for a fixed iteration ℓ, we set $b_\ell = 2$ except with probability $\frac{5}{16}$. Now, applying the Chernoff bound, we can show that the probability of having more than a $\frac{2}{5}$-fraction of iterations ℓ with $b_\ell \neq 2$ is at most an inverse polynomial. Hence, in this subcase the algorithm outputs $(-1, 0)$, except

with inverse polynomial probability.

In the second subcase, we have $u_i > \frac{1}{16}\|g[W]\|_1$ and, for all $j \neq i$, $u_j \leq \frac{\epsilon}{128}u_i$. Then, since u_i is not $\left(1 - \frac{\epsilon}{8}\right)$-heavy with respect to $g[W]$, we have:

$$u_j \leq \frac{\epsilon}{128} \cdot u_i \leq \frac{1}{16}(\|g[W]\|_1 - u_i).$$

Hence, we can apply Lemma 4.4.1 to the vector $U = (u_1, \ldots, u_{i-1}, 0, u_{i+1}, \ldots, u_n)$ (since $\|U\|_1 = \|g[W]\|_1 - u_i$, and moreover each entry in U is at most $\frac{1}{16}\|U\|_1$). Letting $X' = \sum_{j \neq i} h_j^\ell u_j$ and $Y' = \sum_{j \neq i} \bar{h}_j^\ell u_j$, we get that:

$$\Pr\left[\left(X' \leq \frac{1}{4} \cdot \|U\|_1\right) \cup \left(Y' \leq \frac{1}{4} \cdot \|U\|_1\right)\right] \leq \frac{1}{4}.$$

This implies that $X \geq X' > \frac{1}{4}(\|g[W]\|_1 - u_i) \geq \frac{\epsilon}{32}\|g[W]\|_1$ and $Y \geq Y' > \frac{1}{4}(\|g[W]\|_1 - u_i) \geq \frac{\epsilon}{32}\|g[W]\|_1$. Moreover, except with probability $2\delta_2$, y_1 and y_0 are good approximations to $\|g[W_1]\|_1$ and $\|g[W_0]\|_1$, respectively. Thus, except with probability $\frac{1}{4} + 2\delta_2$, we have:

$$\begin{aligned}
y_0 \leq r(n)Y \leq r(n)\left(1 - \frac{\epsilon}{32}\right)\|g[W]\|_1 &\leq r(n)\left(1 - \frac{\epsilon}{32}\right) \cdot \frac{32}{\epsilon} \cdot X \\
&\leq \frac{32r^2(n)}{\epsilon} \cdot y_1 \leq \tau(n,\epsilon) \cdot y_1, \text{ and} \\
y_1 \leq r(n)X \leq r(n)\left(1 - \frac{\epsilon}{32}\right)\|g[W]\|_1 &\leq r(n)\left(1 - \frac{\epsilon}{32}\right) \cdot \frac{32}{\epsilon} \cdot Y \\
&\leq \frac{32r^2(n)}{\epsilon} \cdot y_0 \leq \tau(n,\epsilon) \cdot y_0.
\end{aligned}$$

This implies that, except with probability $\frac{1}{4} + 2\delta_2$, the algorithm sets $b_\ell = 2$ for each iteration

ℓ. Applying the Chernoff bound again, we see that the probability of having more than a $\frac{2}{5}$-fraction of iterations ℓ with $b_\ell \neq 2$ is at most an inverse polynomial. Thus, in this subcase, the algorithm outputs $(-1, 0)$ except with inverse polynomial probability.

We now consider the last subcase, where $u_i > \frac{1}{16}\|g[W]\|_1$ and there exists $j \neq i$ such that $u_j > \frac{\epsilon}{128} u_i$. Note that the probability that i and j get mapped to different matrices is given by $\Pr[H_\ell(i) \neq H_\ell(j)] = \frac{1}{2}$. Assume without loss of generality that $H_\ell(j) = 1$ (the case that $H_\ell(j) = 0$ is symmetric). In the event that i and j get mapped to difference matrices and y_1, y_0 are good approximations to $\|g[W_1]\|_1, \|g[W_0]\|_1$ respectively, which happens with probability at least $\frac{1}{2} - 2\delta_2$, we have:

$$\begin{aligned}
y_1 &\geq \frac{X}{r(n)} \geq \frac{u_j}{r(n)} \geq \frac{\epsilon}{128 r(n)} \cdot u_i \geq \frac{\epsilon}{128 r(n)} \cdot \frac{1}{16} \cdot \|g[W]\|_1 \\
&\geq \frac{\epsilon}{2048 r(n)} \cdot Y \geq \frac{\epsilon}{2048 r^2(n)} \cdot y_0 \Longrightarrow y_0 \leq \frac{2048 r^2(n)}{\epsilon} \cdot y_1 \leq \tau(n, \epsilon) \cdot y_1, \text{ and} \\
y_0 &\geq \frac{Y}{r(n)} \geq \frac{u_i}{r(n)} \geq \frac{\epsilon}{128 r(n)} \cdot u_i \geq \frac{\epsilon}{128 r(n)} \cdot \frac{1}{16} \cdot \|g[W]\|_1 \\
&\geq \frac{\epsilon}{2048 r(n)} \cdot X \geq \frac{\epsilon}{2048 r^2(n)} \cdot y_1 \Longrightarrow y_1 \leq \frac{2048 r^2(n)}{\epsilon} \cdot y_0 \leq \tau(n, \epsilon) \cdot y_0.
\end{aligned}$$

Thus, except with probability at least $\frac{1}{2} - 2\delta_2$, the algorithm sets $b_\ell = 2$ for each iteration ℓ. We apply the Chernoff bound again to get that $b_\ell = 2$ for at least a $\frac{2}{5}$-fraction of iterations, except with inverse polynomial probability. Hence, the algorithm outputs $(-1, 0)$ except with inverse polynomial probability.

□

Lemma 4.7.3. *If the matrix $g[I_H A]$ does not have a key row but has an α-heavy row i_0,*

where $\alpha = 1 - \frac{\epsilon}{8}$, then Algorithm 3 either returns $(-1, 0)$ or returns a $(1 \pm \epsilon)$-approximation of $u_{I_H A, i_0}$ and the corresponding row i_0 with high probability.

Proof. We know there is an α-heavy row, but not a key row. Note that there cannot be more than one α-heavy row for $\alpha = 1 - \frac{\epsilon}{8}$. If the algorithm returns $(-1, 0)$, then the lemma holds (note the algorithm is allowed to return $(-1, 0)$ since there is no key row). If the algorithm returns a pair of the form $(i, BA1(A, H))$, we know from Theorem 4.3.1 that the approximation of the weight of the α-heavy row is a $(1 \pm \epsilon)$-approximation of $\|g[W]\|_1$ as long as $BA1$ succeeds, which happens except with inverse polylogarithmic probability (the argument that the approximation is good follows similarly as in Lemma 4.7.1). We need only argue that we return the correct index, i_0. Again, the argument follows similarly as in Lemma 4.7.1. In particular, if $H_\ell(i) = b_\ell$ for a fixed iteration ℓ, then at least one of y_0, y_1 is a bad approximation or $H_\ell(i_0) = H_\ell(i)$, which happens with probability at most $2\delta_2 + \frac{1}{2}$ (where δ_2 is the failure probability of $BA2$). We then apply the Chernoff bound, similarly as before. □

With Lemmas 4.7.1, 4.7.2, and 4.7.3, we are done proving that Algorithm 3 fits the description of Definition 4.4.1, except with inverse polylogarithmic probability.

4.7.1 Recursive Sketches

Definition of a Cover:

Definition 4.7.1. *A non-empty set $Q \in Pairs_t$, i.e., $Q = \{(i_1, w_1), \ldots, (i_t, w_t)\}$ for some $t \in [n]$, is an (α, ϵ)-cover with respect to the vector $V \in [M]^n$ if the following is true:*

1. $\forall j \in [t]\ (1-\epsilon)v_{i_j} \leq w_j \leq (1 \pm \epsilon)v_{i_j}$.

2. $\forall i \in [n]$ *if* v_i *is* α*-heavy then* $\exists j \in [t]$ *such that* $i_j = i$.

Definition 4.7.2. *Let* $\mathcal{D}$ *be a probability distribution on* $Pairs$*. Let* $V \in [m]^n$ *be a fixed vector. We say that* $\mathcal{D}$ *is* δ*-good with respect to* V *if for a random element* Q *of Pairs with distribution D the following is true:*

$$P(Q \text{ is an } (\alpha, \epsilon)\text{-cover of } V) \geq 1 - \delta.$$

Using notation from,[98] for a vector $V = (v_1, \ldots, v_n)$, we let $|V|$ denote the L_1 norm of V, $|V| = \sum_{i=1}^{n} v_i$. Consider Algorithm 6 from:[98]

Algorithm 5 Recursive Sum (D, ϵ)

1. Generate $\phi = O(\log(n))$ pairwise independent zero-one vectors $H_1, \ldots, H_\phi$. Denote by D_j the stream $D_{H_1 H_2 \ldots H_\phi}$

2. Compute, in parallel, $Q_j = HH(D_j, \frac{\epsilon^2}{\phi^3}, \epsilon, \frac{1}{\phi})$

3. If $F_0(V_\phi) > 10^{10}$ then output 0 and stop. Otherwise, compute precisely $Y_\phi = |V_\phi|$

4. For each $j = \phi - 1, \ldots, 0$, compute

$$Y_j = 2Y_{j+1} - \sum_{i \in Ind(Q_j)} (1 - 2h_i^j) w_{Q_j}(i)$$

5. Output Y_0

Theorem 4.1 from:[98]

Theorem 4.7.1. *Algorithm 5 computes a* $(1 \pm \epsilon)$*-approximation of* $|V|$ *and errs with probability at most* 0.3*. The algorithm uses* $O(\log(n)\mu(n, \frac{1}{\epsilon^2 \log^3(n)}, \epsilon, \frac{1}{\log(n)}))$ *bits of memory, where* μ *is the space required by the above algorithm* HH*.*

Chapter 5

Fuzzy Heavy Hitters

5.1 Introduction

Streaming approaches have been developed for several classic clustering problems, including k-median,[109–112] k-means[113,114] and facility location.[115] These algorithms are typically applied to multidimensional elements such as vectors in $\mathbb{R}^d$. Separately, there are a variety of approaches for discovering the most frequent discrete elements in a stream, referred to as finding *heavy hitters*.[116]

We combine these two tasks into a novel problem definition, that of finding *Fuzzy Heavy Hitters*: we assume each point in the data stream is drawn from one of very many (k) latent clusters, and our goal is to find the k' heaviest clusters (that is, the clusters containing the most points from the stream). As we constrain ourselves to the streaming data setting, we cannot afford the memory to store nor the time to organize all the incoming data points.

Furthermore, we assume there is insufficient memory and time to track all k clusters, and that we are only allowed a single pass over the data.

We tackle this problem by creating a new algorithm which we call Local Fuzzy Counting (LFCount, or LFC). This algorithm is built by modifying an existing Heavy Hitter algorithm, the Space Saving[117] frequent elements algorithm, to fit the Fuzzy Heavy Hitters problem. This concept represents a general framework for solving problems with these underlying groupings.

One such example of latent clusters is the concept of finding matching topics across multiple documents. Topic detection in data streams, such as in[118] is just one application of the general FHH approach (see Section 5.6 for a discussion on topic detection and other applications for FHH).

We apply the Local Fuzzy Counting algorithm to the problem of topic detection in data streams, as shown in the work of Petrovic et. al.[118] Our implementation provides improved space and time complexity over previous results with a minimal accuracy trade-off.

5.2 Problem Definition and Assumptions

We begin by defining the data stream of vectors.

Definition 5.2.1. *Datastream*

Let n be a positive integer. A stream $D = D(n)$ *on $\mathbb{R}^d$ is a sequence of length n, denoted $a_1, a_2, \ldots, a_n$, in which each entry a_i is a vector in $\mathbb{R}^d$.*

Definition 5.2.2. *α-Heavy Hitter*

Given a stream of points $D = p_1, p_2, \ldots, p_n$ from a metric space (X, d), suppose that there exists a function $\tau : [n] \to [k]$ mapping each point in the stream to one of k classes. For each $i \in \{1, \ldots, k\}$, let $C_i = \{p_j : \tau(p_j) = i, 1 \leq j \leq n\}$ denote the set of points from the stream assigned to the i-th class, and let $f_i = |C_i|$ be the frequency of that class. We say the i-th class is α-heavy *if $f_i \geq \alpha n$.*

Definition 5.2.3. *Heavy Hitter Algorithm (HHA)*

A **Heavy Hitter Algorithm** *finds the* α-heavy *classes of a stream D.*

We now formally define the Fuzzy Heavy Hitter (FHH) problem.

Definition 5.2.4. (k, r, α)*-Fuzzy Heavy Hitter*

Consider a promise problem wherein we are guaranteed that the input classes and the function τ satisfy the following size property:

- *For all $i \in [k]$ we have* $\operatorname{diam} C_i \approx r$

where $\operatorname{diam} C_i = \max_{p,q \in C_i} d(p, q)$ *is the diameter of the i-th class. The* (k, r, α)-Fuzzy Heavy Hitter problem *is a promise problem wherein we are guaranteed that the above condition holds, and our task is to find a representative point from each of the α-heavy classes. That is, letting $H = \{i \in [n] \colon |C_i| \geq \alpha n\}$ denote the set of indices of heavy classes, we return a representative point $p \in C_i$ for each $i \in H$.*

Observation: using the same notation as in Definition 5.2.4, $|H| \leq 1/\alpha$.

We solve a slightly modified version of this problem:

Definition 5.2.5. (k, r, α, ϵ)*-Approximate Fuzzy Heavy Hitter*

The approximate Fuzzy Heavy Hitter Problem is formulated the same as the Fuzzy Heavy Hitter Problem in Definition 5.2.4, with the only difference being that while in the original problem we return exactly the exact heaviest items, we instead guarantee with probability greater than $2/3$ *we return* k *elements from* $H = \{i \in [n] \colon |C_i| \geq \alpha\epsilon n\}$ *where* $0 < \epsilon < 1$ *is a small constant and guarantee with probability greater than* $2/3$ *that we do not return any elements not in* H.

This approximate formulation represents a more reasonable problem to solve given the randomized tools that are useful to the field. The notion of a known diameter in both problems is a natural one; the algorithm needs some notion of a distance threshold between elements to group them as similar, so the clusters need to be no more than the given size.

In terms of the separability of classes, we assume that elements belonging to the heaviest clusters are ”nice” as named by Ackerman and Dasgupta,[119] and based on definitions from Balcan, Blum and Vempala.[120]

Definition 5.2.6. *Nice Clusters*

A Nice Cluster C_i *is defined where* $\forall j \notin C_i$, $\forall i \in C_i$, $d(i_1, i_2) < r < d(i, j)$

This assumption allows us to guarantee that two heavy clusters do not partially overlap, removing the ability to falsely report two clusters as one ID. We do not need to make a separability assumption about the non heavy clusters, as if they were to sufficiently overlap the overlapping section would be a heavy cluster by definition, and subject to the niceness assumption.

This separability assumption allows us to quickly obtain correctness for the LFCount algorithm in the theoretical setting as a direct result of SpaceSaving. In our implementation, we account for the potential overlap of classes with a variance reduction strategy.

In many practical datasets heavy items tend to be distinct, as shown in Table 5.1. When heavy elements are not distinct, or have less predictable shapes, other clustering methods such as density based clustering may be preferable.[38]

While there is requirement for the relative size of classes, there is no such assumption made about the separability between classes, we leverage the natural separability of classes in practical datasets to more easily distinguish between different labels.

For example, consider the task of finding the popular topics in a collection of text documents given the bag-of-words representation of the data.[121–123] In this setting, each document's representation is sparse and we expect the representations of two documents discussing different topics to be separated by a larger distance than the representations of two documents discussing the same topic. As another example, consider data drawn from a Gaussian mixture model: if the means of the latent classes are suitably far from one another and the variances are suitably small, then the data generated from this model will satisfy the two assumptions of Definition 5.2.4 with high probability.

While we focus on text for our experimentation, the FHH problem is in no way restricted to text, as we discuss further in Section 5.6.1.

In settings where separability is not a characteristic of the dataset, the problem is still

solvable [1]. In the case where classes overlap almost entirely, the classes are so similar that they can be safely treated as one and reporting on one is acceptable due Definition 5.2.5. Similarly, if there is minimal overlap under-counting is not an issue as it is unlikely for an element to be more than ϵ smaller than it's original value and thus will still be a member of the correct output set H. In this second case it is possible that the label for a cluster is incorrect, but this can be remedied by sampling centroids as opposed to choosing the first to arrive or updating the "centroid" as time passes; we leave this improvement for future work.

Throughout this chapter we will speak interchangeably about finding the approximate top-k' heaviest clusters and finding k $\alpha_{k'}$-approximately heavy clusters. We note that given some value α, by the observation above, there are at most $1/\alpha$ heavy clusters. In the opposite direction, if we want k' clusters, it is necessary that all of those clusters be suitably large such that they can be found in reasonably small space. Thus, we assume there is some value $\alpha_{k'}$, that is a constant lower bound to the size of the k'-th element. This is necessary since if the size of the k'-th cluster is $O(1/n)$, no small space algorithm can hope to find that cluster.

5.3 Comparison with Other Clustering Problems

At first glance, the FHH problem may seem similar to clustering with outliers,[124,125] but the problems are distinct. In clustering with outliers the goal is to remove a set number of points until the optimal clustering in minimized. This is still very much reliant on the distances between points, and can cause issues. Imagine an example where there are two large groups

[1] Additionally, future FHH algorithms may not need this assumption, this assumption is directly tied to the construction of LFCount

of elements near each other, and then a single smaller groups of points located very far away from the first two clusters. In clustering with outliers, the centroids would need to move towards the far away cluster, causing less total points being close to the centroid. On the other hand, FHH aims to place cluster IDs over regions containing the most points within a set distance of the centroid. This means that once a point is outside of the a clusters radius, the distance to those other points does not impact the result.

A more closely related problem is that of Approximate Facility Location (AFL).[126] In AFL, a set of facilities are placed so that they may optimally service as many clients. In an applied setting, this might be placing hospitals such that driving distance is minimized for as many people as possible. This problem can be formulated with many different constraints, such as limiting the maximum number of clients per facility (capacitated), and usually also includes a method where the number of factories may be increased at some cost.

FHH can be thought of as a special case of AFL where a constant number of facilities is chosen in advance, the facilities are uncapacitated, and have a maximum radius of service r. Further discussion of related works can be found in Section 5.6. At the time of writing, we are not aware of any theoretical results that solve this specific variation of the problem or provide a lower bound, which is an easier formulation than the general case due to the constant number of facilities and the limited search radius.

5.4 Our Algorithm

A Fast Check for Membership

In most heavy hitter algorithms, the data being processed is discrete. This property allows for quick comparison by checking if two items are equivalent by direct comparison or comparing the results of universal hashing. However, for the FHH problem, the dataset is comprised of high dimensional vectors. In this setting we must match elements which are not identical, causing new challenges to arise. Intuitively, if we somehow knew the correct class label of each observation in our stream (and could determine this class label quickly), classical heavy hitter techniques would apply directly to the task at hand using universal hashing.

In the case of a stream of vector observations, the notion of distance between matching items becomes an issue. In our case, this means this universal hashing cannot be applied directly and we must find some other way to determine class membership for these vectors. Such methods also introduce additional challenges in applying heavy hitters algorithms directly.

For the problem of heavy hitters, the clusters in Definition 5.2.4 define a different method for determining class membership and we now describe a method for checking membership with an arbitrary distance function. If a distance function returns a point is within r from a centroid, it belongs to that centroid, and if a point is farther than r from a centroid, it does not belong.

Note, that due to the approximate nature of some of the algorithms we use to solve this problem, we are not able to guarantee that for a given two items within distance $2r$ that they will both be assigned to the same cluster. While this distance computation for determining membership may be costly for high-dimensional data, the sparsity observed in many practical applications allows us to determine cluster membership in close to constant time, allowing us to use counter-based heavy hitter algorithms[117] to solve the problem.

This ability to determine whether or not two elements are in the same cluster is a weaker capability than that assumed for standard heavy hitter algorithms, where by assumption each element of the stream is simply one of m possible types (i.e., the true class labels are known explicitly). This requires us to modify existing heavy hitter algorithms, and requires more computation per update than in the discrete case.

A Heavy Hitter Approach

A straightforward approach based on an adaptation of existing counter-based heavy hitter algorithms provides a baseline solution to the AFHH problem. This method requires $O(n/\alpha_{k'})$ time and only one pass over the data. Using the comparison properties discussed in the previous section, we can apply any number of algorithms for finding heavy clusters. In particular, we use the Space-Saving frequent elements algorithm.[117] Space-Saving is a counter-based technique for approximately finding the most frequent items in a datastream. The algorithm works by maintaining a set of counters that each correspond to a held heavy element and incrementing the counter when the element is seen. If an element does not match any of the held elements, the element with the lowest count is evicted and the new

element takes its place; the counter is not reset. With Space-Saving, if the k'-th heaviest element has weight $f_{k'} = \alpha_{k'} n$ then we can find the approximate top k' classes using $O(1/\alpha_{k'})$ space, in the sense that we do not overestimate the heaviness of any element by more than an additive error of $n\alpha_{k'}$. Moreover, under the assumptions of our promise problem and the niceness of heavy clusters, we are guaranteed not to miss any of the top k' elements.

The time bound of our adapted algorithm is worse than that of the standard Space-Saving algorithm in the integer setting as a direct result of the fact that we cannot use a traditional hash table on our vector observations. As a result, the arrival of each new point requires a a linear sweep over the buckets to search for a collision. This results in a $O(n/\alpha_{k'})$ runtime using a single pass over the data rather than the linear runtime required by the Space-Saving algorithm.

In practice, there tends to be a good deal of turnover in the SpaceSaving buckets with lower counts. Intuitively, this is due to the fact that while the data structure guarantees it will return elements that are α heavy, each bucket has no guarantee it will actually return a heavy element. Elements can be confirmed as heavy with a second pass by checking the fuzzy count of the heavy element, but can also be improved by maintaining a larger number of total buckets than is required to reduce the amount of error in the count.

For large practical datasets, α'_k can be a very large constant with multiplicative impact on runtime. Consider a data set with 1 billion points. If the goal is to find $\alpha'_k = 0.001$ heavy hitters, we would need to perform a linear sweep over one million buckets per update.

Unfortunately, this degree of scaling is not sufficient for real world datasets, and we need an improved approach.

LOCALITY SENSITIVE HASHING

While universal hashing is not applicable to FHH, locality sensitive hashing (LSH)[33] is. LSH gives an approximate solution to the nearest neighbor problem. This method was later applied by Charikar[34] as a method for approximating the relative angle of high dimensional vectors by mapping them into a lower dimensional space. Charikar's method approximately preserves cosine similarity, and we use this approach in our algorithm.

Definition 5.4.1. Locality Sensitive Hashing

For Locality Sensitive Hash Function H, with $M = 2^b$ bins, L maps a high dimensional vector $v \in R^d$ into a lower dimensionality R^b. For $v' \neq v$, $Pr[H(v') = H(v)] = 1-\theta(v', v)$, where $\theta(v', v)$ is the angle between v' and v.

To achieve this, LSH first generates a set of random hyperplanes $\vec{u} \in U, |U| = b$. Then, for each element hashed, LSH returns a *bit signature* by computing the boolean value $\forall \vec{u} \in U, \vec{v} \cdot \vec{u} > 0$ and returning a concatenation of the resulting values. The number of bits in the bit signature is equal to $2^{|U|}$. With this, we can imagine LSH dividing high dimensional space into different regions dependent on the number of these hyperplanes. We call these regions *bins*.

Other work has used LSH for it's ability to preserve cosine distance between elements and find nearest neighbors by computing the hamming distance of signatures, such as

Ravichandran et. al[127] and Jansen and Van Durme.[128] In this thesis, we use LSH only for the process of creating these regions, similarly to previous work like the Streaming First Story Detection work of Petrovic et. al.[118]

LSH is a powerful tool for approximating the nearest neighbors of an element, and can be used to give approximate labels to an item such that they could be computed by a sketch based heavy hitters algorithm. However, LSH also introduces an additional layer of approximation, which can have a negative impact on accuracy.

LSH is very much an enabling technology for solving FHH and presents many questions about using LSH to fill the role of universal hashing for many established techniques. However, LSH is not a cure-all solution for the problem, and below we provide some context on some of the design decisions we did not make when applying LSH, and why.

At one extreme, if a very large amount of LSH bins was used to try and divide space very finely, then each vector would receive its own unique integer representation. While this may work for finding the Heavy Hitters of high dimensional vectors, it does not solve the problem for Fuzzy Heavy Hitters, as the resulting integer for each vector would still need to be compared using a distance function to determine proximity to other elements.

Looking at SpaceSaving, a question is why would we not use LSH as a direct substitute for universal hashing to create a hash table to allow $O(1)$ lookup inside of our SpaceSaving data structure to check if it contains the new element. The answer is that when using LSH with a large number bits, such that there are a sufficient number of unique bins such that a hash table built with linked lists would not have an impractical amount of collision, the

distance r that lands inside a single bin is very small and similar items are not grouped. Conversely, if we use too few bits, we end up grouping many elements of varying distance together and are unable to provide an accurate analysis. This tension between LSH not providing enough resolution versus dividing clusters is one of the core challenges of our solution to the FHH problem.

As discussed in earlier chapters, sketch based algorithms are a common form of streaming algorithm. While many rely on hashing, the construction is more complex than the hash table based approach in SpaceSaving. Again, the question would be whether or not we could extend a sketch based algorithm such as Count Min Sketch[129] to FHH by replacing an integer hash function with LSH. While we do not claim that such a method is not possible, as many of these algorithms already handle item collisions as part of their design, we believe the amount of LSH bits required to obtain accurate division of the data set would likely end up dividing the data too far for accurate recovery of element counts and identifiers. Further, counter based approaches provide similar guarantees without requiring as many LSH bits. However, if such a technique were to exist, it certainly may provide improved performance, and we leave that approach as an open problem.

The fundamental tradeoff with using LSH is using an amount of bits such that you are able to effectively divide your dataset without using too many bits such that the dataset is over-divided.

A HYBRID APPROACH

We present Local Fuzzy Counting, or LFCount, an algorithm that uses both LSH and heavy hitters to quickly and accurately solve the FHH problem. The underlying data structure creates many instances of Space-Saving and LSH to assign elements to Space-Saving "buckets".

In the above Heavy Hitters based algorithm, and in early testing, we found that by maintaining only a single instance of Space-Saving had issues with both accuracy and runtime efficiency. As the number of buckets grew large, the linear sweep becomes very costly, and due to the noise in our experimental data sets many elements would be from small disjoint classes, causing a high turnover in keys.

To improve upon the initial design we use a streaming implementation of LSH. With LSH we create a hash mapping from vectors to b bins, with one bin for each unique LSH signature. With this, we maintain many parallel instances of Space-Saving each with c counters, one for each LSH bin. This reduces the search time for each Space-Saving as well as allowing us to maintain a stricter comparison inside of each LSH bin, as LSH provides that all elements are approximate nearest neighbors for a radius $r' > r$, where r is the radius of the clusters we aim to find.

The LSH method we use in our experiments functions in a streaming way[128] and gives fast updates. This gives us an update time of $O(c)$, where $c < n/\alpha_{k'}$ by approximately a factor of b to give similar results.

Algorithm 6 Online LFCount Algorithm

```
for all Documents d in corpus do
    add d to LSH
    S ← the SpaceSaving data structure in the LSH bin where d lands
    Insert d into S
    for all For s in S do
        c = distance(d, s)
        if c > thresh then
            s+=1
        else
            Evict s with smallest count, replace with d, keep old counter

    After online component, perform Variance Reduction (See Algorithm 7)
```

PERFORMANCE AND PARAMETER SELECTION

For an LFCount instance with 2^b bins each with c counters, the algorithm has a worst case runtime of $O(c * m)$, and assuming an even distribution of unique elements across the bins, will find all elements at least $\alpha = \frac{m}{c2^b}$ heavy on the data stream, and over-count by at most $\alpha = \frac{m}{c2^b}$, by the correctness of SpaceSaving.

With this, we observe that the runtime of the algorithm is linearly dependent on the number of buckets per SpaceSaving. The number of buckets per SpaceSaving is relative to the total number of buckets $c2^b$, which dictates the accuracy of SpaceSaving. A natural next step then would be to increase the number of LSH bits, which in turn would decrease the number of counters per bucket and improve speed; however, this can negatively impact accuracy.

As was discussed in the previous section, there is an upper limit to the amount of LSH bits that can be used before accuracy is compromised. At one extreme, we could imagine using a very long LSH signature. In this case, every unique element would have its own

Algorithm 7 LFCount Variance Reduction Method

```
Collect Heavy Hitters from each LSH bin, place all in array A
With var_t as the number of variance reduction iterations to perform:
for i < var_t do
    Create LSH'_i with new random hyperplanes
    Create HashTable B, mapping from int to array
    for all Heavy Documents a_i in A do
        j = LSH'_i(h_d)
        for all Heavy Documents in vector B[j] do
            c = distance(d, h_i^d)
            if c > thresh * 2 then
                s+ = 1
            else
                append a_i to B[j]
    for all B[j] in B do
        Create new Array A'
        Append all arrays B[j] to A'
        Delete B
        Set A = A'
        i++
```

SpaceSaving instance inside of a unique LSH bin and LFCount would not return the correct answer. On the other extreme, if LSH is not used, the accuracy is maximized, but a full linear search over all buckets is required.

Of course, in between these extremes is a lot of room for calibration and as such finding the right parameterization yields the best results. The ideal calibration is one where LSH does not split up too many heavy clusters between bins, while minimizing the number of buckets per LSH bin as to maximize the speed of the algorithm. This best configuration is data set dependent, and there is unfortunately no "silver bullet" solution for LFCount.

VARIANCE REDUCTION

One of the issues with this design is that clusters can be divided across LSH boundaries, so a variance reduction step is needed. In some cases, this division will simply remove a small portion of the "heaviness" from the element, in others, it's possible to divide a very heavy element into two separate heavy elements. Even with $k' << k$, we find experimentally that a direct pairwise comparison of the $b * c$ buckets held by SpaceSaving, requiring $O(k')$, returned by the algorithm required too much time to be effective at scale. Instead, we use an improved method.

Given the number of LSH bits used for the initial pass, we select a smaller number of bits (e.g. from 16 to 12 or 8 to 4), and give the LSH instance a new random seed. Then, create a hash table, B mapping from integer LSH values to arrays. For each item in the set of heavy elements, hash the item and then do a pairwise comparison with all members of the corresponding array given by the hash table.

Experimentally, a single pass of this method was found to be insufficient for certain values of r. To solve this issue, we iteratively feed the vector of heavy elements to the variance reduction step, each time recombining clusters previously divided by LSH. With this, the odds of dividing a cluster fall dramatically, but we still instead have a runtime of $O(k'^2/2^b)$, and allows parallel updating of the variance reduction step, allowing up to 2^{2b} parallel processes. Thus, we are able to recombine split elements when the separate parts are both heavy, providing more accurate counts and rankings of the heaviest elements. For larger k' in practice, it is usually the correct choice to choose a larger LSH value and perform

variance reduction multiple times.

This variance reduction method does introduce a challenge. Consider the following example, it is possible for cluster A and B to contain center a_c and b_c respectively, as well as elements a_e a_b which are both $thresh$ away from their centers. Then, if a_c and b_c are within a distance threshold, they would merge into a single cluster C. In this scenario, it is possible for C to have elements separated by a distance of $3r$ be grouped into the same cluster. This is solved by the recombination step using a higher threshold for equality than the main phase of the algorithm, as to avoid generating false positives by over-combining.

Finally, a drawback of this variance reduction approach is that it does not function on a per element basis. This computation is inefficient, and should only be performed periodically or at the end of the stream.

5.5 Experiments

IMPLEMENTATION AND TESTING

We implemented our LFCount algorithm using C++ and running on machines with 12/16 2.1Ghz Cores with tasks limited to 100GB of memory (the algorithm uses dramatically less memory). We evaluated our algorithm on two datasets. The first dataset is a 2-dimensional synthetic dataset generated under the assumptions of our promise problem: finding points distributed along an Archimedes Spiral. We also performed experiments on one billion tweets taken from the 1% sample of tweets provided by the Twitter API,[130] with dates ranging from 2013 to 2015.

5.5.1 Synthetic Data

In terms of choosing a distance function for our experiments, Ertoz, et al.[131] observe that choosing a distance function that captures an appropriate notion of similarity can be difficult. Determining the best distance function for a given dataset is not the focus of the current work. For this experiment we use L_1 distance, as this allows us to to best compare with $k - medians$.

This dataset is generated under the requirement that all centroids are at least distance $2r$ apart and such that all of the points for the centroid lie inside the radius. The points for the centroid are generated randomly inside of the circle, and are Gaussian-distributed away from the centroid. This is a stricter assumption than being "nice" for heavy elements, but helps model the data nicely and makes it easier to compute for k-medians. Distances are measured using L_1 distance.

The Archimedes spiral gives us an even distribution of clusters through space, as well as consistent inter-cluster distance, the constantly increasing spiral creates a lower bound for the minimum distance between two clusters.

In terms of accuracy, because we use synthetic data where all *centers* are $2r$ apart, with all elements inside of the radius r, we can quickly and accurately . Due to the correctness of Space-Saving, when LSH does not introduce error we properly find all of the heavy elements in the stream that are at least $\frac{m}{b}$ heavy, where b is the number of buckets. Thus, we use this data to evaluate the performance of our approach in finding dense regions in a streaming

manner.

COMPARISON WITH k-MEDIANS

We now compare LFCount against k-medians in the setting when the *centroids* are $2r$-separated and the points are Gaussian-distributed inside of the centroids. While the centroids are mutually $2r$-separated, we note that arbitrary points in adjoining clusters may not be. In this scenario, it is possible that the returned clusters are not well-aligned over the data and there is not a guarantee that our method will return the true top k. In this experiment 10,000 points are Zipfian distributed across 13 clusters. Our goal is to return the top k' = 4 clusters. Thus, we compare with k-medians in two cases, k = 4 and k = 13. We show both examples, as where k = 13 would be ideal to partition the data given that there are 13 true clusters, k = 4 compares directly with the result we aim to return from LFCount.

The different colored points represent the clustering assigned using k-medians. We show the lowest-cost k-medians solution out of three (using three random initializations). The solid black circles encompass the three true heaviest clusters; the dotted green circles are centered on the centroids returned by LFCount.

We evaluate the algorithms in four settings: No noise, 1,000 additional points of noise (10%), 2,500 additional points of noise (25%), and 5,000 additional points of noise (50%). The noise is generated by adding points at uniform random at a range at most 1.25 times the further most point in generated for a cluster. We show these results in Figures 5.1,5.2,5.3,5.4,5.5,5.6,5.7,5.8

What we see in these experiments is that in all cases with k = 13 , $k$medians struggles

with the data and divides up the true clusters across multiple centroids, failing to obtain information about the heavy clusters. With k = 4, k-medians tends to group two clusters together and then distinctly identify the other two, additionally,the location of the centroids placed by k-medians is not close to heavy region. When noise is added these issues intensify, and in 5.8, we see the points of a heavy cluster is even divided over two k-medians centroids. On the contrary, even with noise LFCount successfully finds and returns the heaviest regions with some error as noise increases.

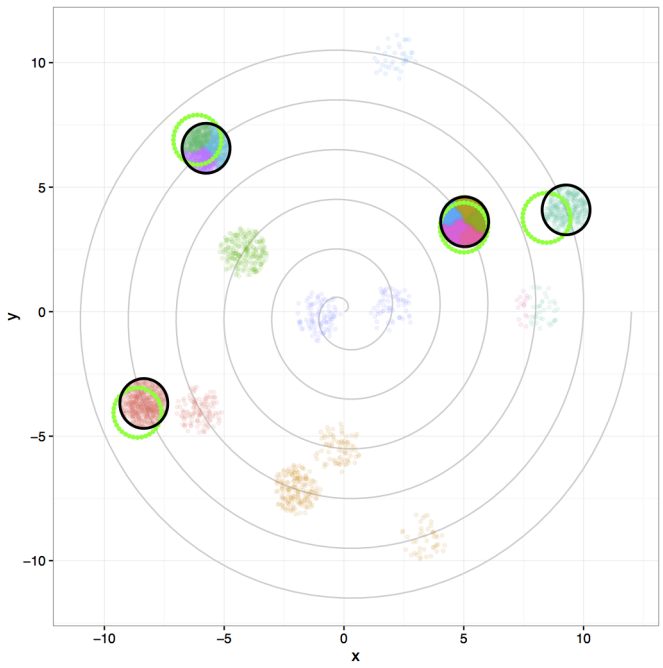

Figure 5.1: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. k = 13. No noise.

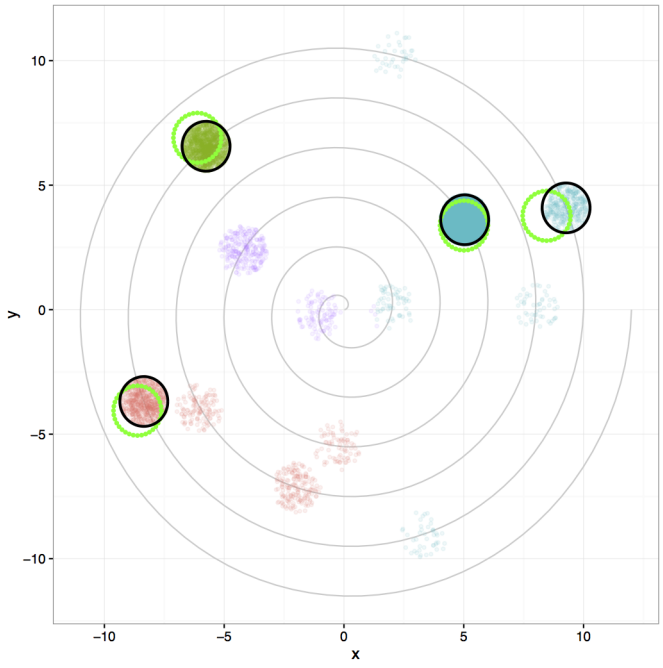

Figure 5.2: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. k = 4. No noise.

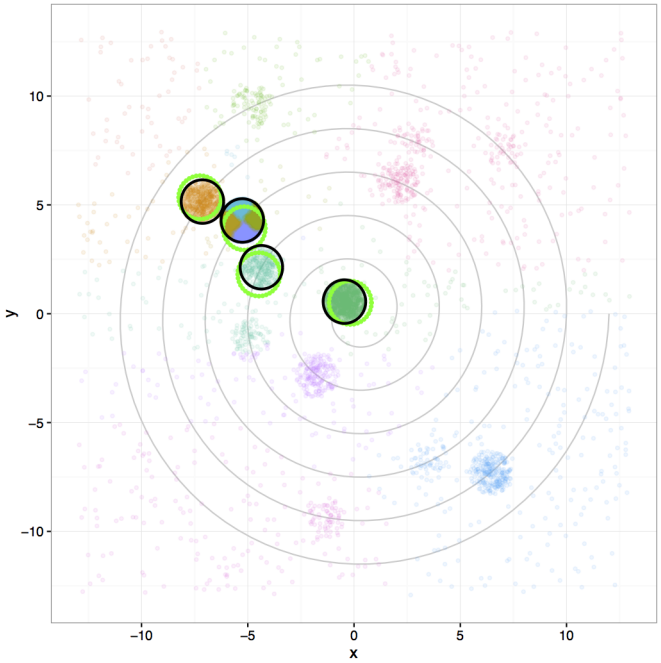

Figure 5.3: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. $k = 13$. 10% noise.

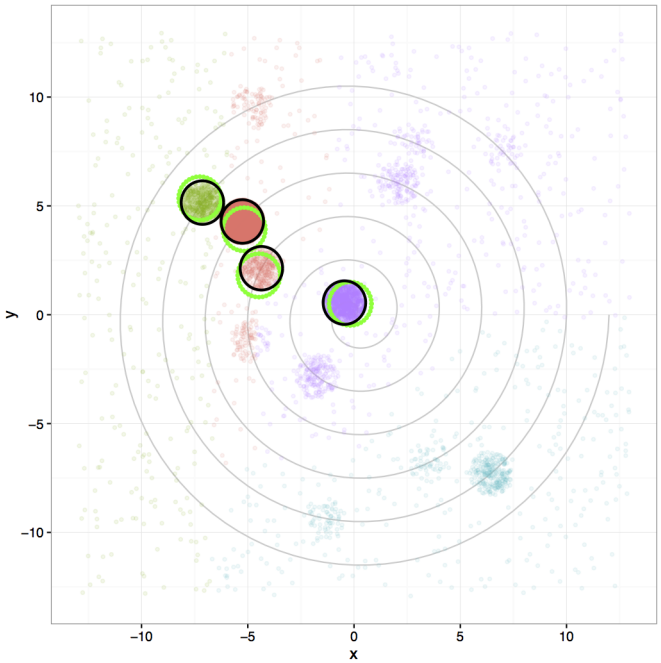

Figure 5.4: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. $k = 4$. 10% noise.

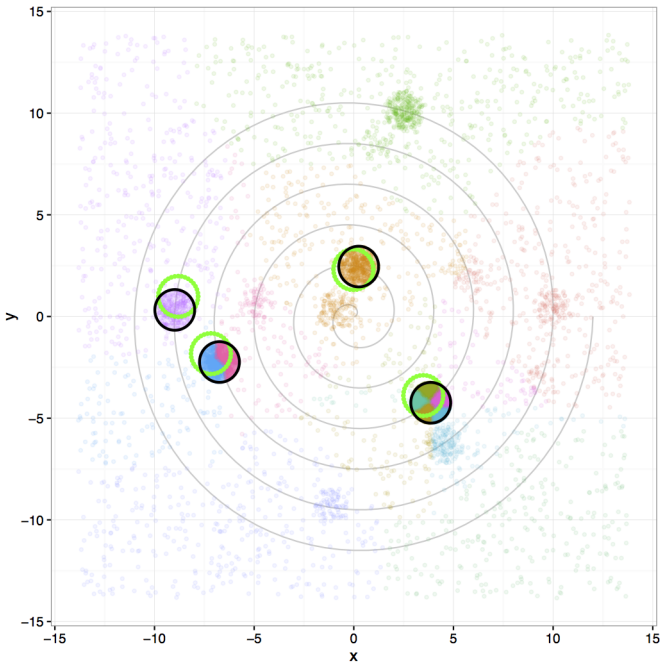

Figure 5.5: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. $k = 13$. 25% noise.

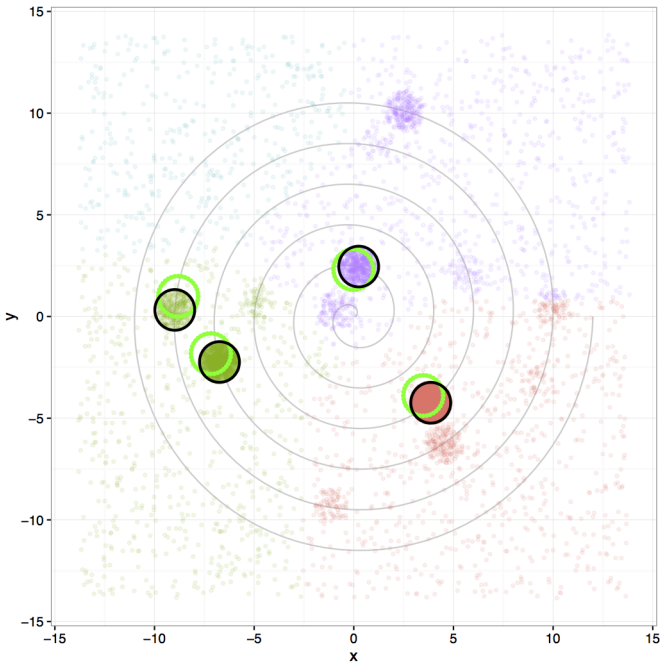

Figure 5.6: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. $k = 4$. 25% noise.

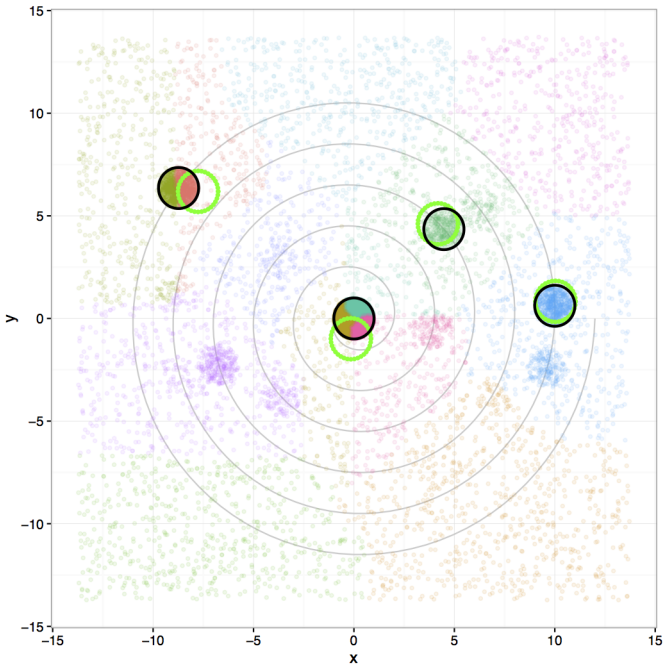

Figure 5.7: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. $k = 13$. 50% noise.

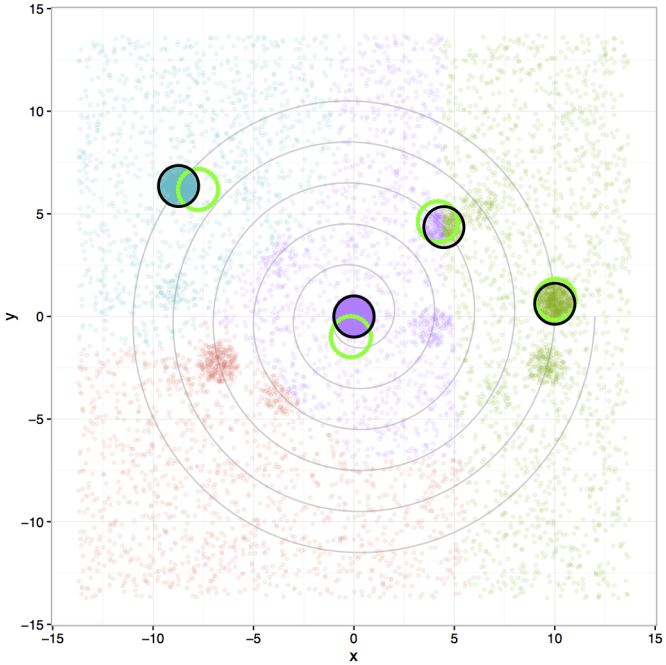

Figure 5.8: Comparison of FHH output with k-medians on Gaussian clusters arranged along an Archimedean spiral. $k = 4$. 50% noise.

5.5.2 Twitter

Our main goals in building a streaming algorithm are using a small amount of memory, taking a small number of passes, being accurate, and being fast enough to handle a high throughput data set. We are able to control the amount of memory as it is directly related to the number of buckets used by the data structure, and the algorithm is designed to take one pass. Thus, this section is chiefly concerned with testing the accuracy of finding the fuzzy heavy hitters and evaluating the speed of the algorithm. For these experiments we use cosine distance, as we felt is was best for handling the high dimensional sparse vector representation of tweets.

PARAMETERIZATION

LFCount has multiple parameters to consider, including number of parallel threads, number of buckets, and number of bins. The performance of the LFCount algorithm is dependent on the ratio of the number of unique keys to the size of both the LSH bins and the SpaceSaving buckets, and thus choosing a representative parametrization is key. We choose to use 1.3 million total buckets, divided across 16 bits of LSH, for 65536 bins with 20 buckets each. This allows for a quick updates, while having enough buckets per bin such that SpaceSaving accurately finds the heavy elements in that bin, as long as they are more than $\alpha = 0.05$ heavy on the substream. Assuming an even distribution of elements across bins, this lets us find items that occur at least $1,000$ times in the full dataset, and we overestimate the counts of heavy elements by at most $1,000$.

A major hurdle preventing us from obtaining ground truth at scale. While for LFCount we use LSH to allow us to only compare to 20 elements per update, in a naive approaches we must compare to every other element, which requires $O(n^2)$ time. This can be months of computation for larger data sets. To resolve this, for certain experiments we choose mid sized data sets of ten to fifty million tweets, as opposed to the full 1 billion tweet data set.

Tweets Found

In the Twitter data, the frequent clusters we found were commonly used expressions, automated messages, as well as popular events. A selection of some of the heaviest heavy labels returned is shown below.

These items are not the heaviest k' by count, but give a good representation of the elements shown in the top k'. The text is taken directly from the cluster ID returned by LFCount.

Tweet Text	Count
GET MORE FOLLOWERS MY BEST FRIENDS? I WILL FOLLOW YOU BACK IF YOU FOLLOW ME - http://t.co/t5SyGeh	157,000
Test	67,300
:(	67,000
:)	56,810
Lol	41,750
Good morning to all	37,560
Goodnight :)	33,469

Table 5.1: Selected Heavy Cluster IDs and Counts

What we see in the results for Twitter is that there is a lot of redundancy and non-event noise. Large amounts of tweets are greetings or automatically generated messages. The nature of this data suggests a potential additional use for the Fuzzy Heavy Hitters problem and LFCount, that we did not foresee: removing common but unimportant messages, which we call *s*top messages.

STOP MESSAGES

Using LFCount, we we can create a filtering concept similar to stop words[132] in text analysis. Where stop words are common terms that should be ignored, these ”stop messages” on Twitter represent the most commonly exchanged messages that we also wish to ignore.

This concept allows us to use LFCount as a tool to determine stop messages for a data set which can then be used as a filter to improve accuracy or reduce the size of the dataset for other algorithms [2], which we leave as future work.

HEAVY HITTERS AND FUZZY HEAVY HITTERS

Considering the close relationship between FHH and Heavy Hitters, we compare the effectiveness of the two for characterizing the Twitter dataset. With this, we are looking to resolve the question of whether or not a Heavy Hitters algorithm can provide an accurate approach to finding Fuzzy Heavy Hitter cluster IDs on Twitter. If the answer is yes, this implies that the fuzzy weight is primarily occupied by very similar near duplicates (or exact duplicates) on Twitter.

To evaluate this difference, Figure 5.9 shows an experiment evaluating the heavy hitters of a 50 million tweet dataset, and examining the $k' = 10,000$, $50,000$, and $500,000$ labels returned by a Heavy Hitter algorithm and compares the result to the LFCount algorithm.

In this case, the evaluation for Heavy Hitters operates as shown in Algorithm . Labels are obtained by using a memory unbounded heavy hitter algorithm using a hash table and counters. Then, for the k' heaviest hitters, we treat these as cluster ID labels, and perform a second pass over the dataset to determine the fuzzy count for these cluster IDs. For time efficiency, we divide the stream using LSH and then all items in the same bin are compared to see if they are within radius r. This gives us the final fuzzy weight of the top k' heavy hitters.

[2] For example, see section 5.6.1 for discussion of Topic Detection and Tracking algorithms.

Algorithm 8 Naive Heavy Hitter Algorithm

Initialize Hash Table H, with each entry being a counter.
for all Documents d in stream **do**
    H(d)++
Sort H by counter values
Let D' be top k' values in H
for all d' in D' **do**
    Compute LSHd'
    Create A, an array of pairs in the LSH bin where d lands
    Create counter $c_{d'} = 0$
    Append $(d', c_{d'})$ to A.
for all Documents d in stream **do**
    Compute LSH(d'
    for all $d' \in LSH(d)$ **do**
        if (dist(d,d') $>$ thresh) **then**
            $c_{d'}$++

A similar method is used for FHH as shown in Algorithm 9, where instead we take the cluster ID output of LFCount and use a second pass to get an accurate fuzzy count for the clusters returned, to remove any over-counting caused by SpaceSaving.

For these experiments, 1,000,000 total SpaceSaving buckets are used in all 3 experiments, and the $k' = 10{,}000$, 50,000, and 500,000 labels are returned. This constant number of total SpaceSaving buckets explains the similar runtime for the LFCount experiments.

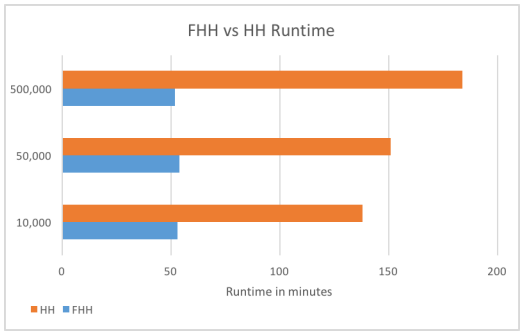

Figure 5.9: Comparison of LFCount and HH runtime

Algorithm 9 LFCount Second Pass for Accurate Fuzzy Counting

Given final LFCount Data Structure LFC
Sort LFC by counter values
Let TOP be top k' values in LFC
for all d' in TOP **do**
    Compute $LSH(d')$
    Create A, an array of pairs in the LSH bin where d lands
    Create counter $c_{d'} = 0$
    Append $(d', c_{d'})$ to A.
for all Documents d in stream **do**
    Compute $LSH(d')$
    for all $d' \in LSH(d)$ **do**
        if (dist(d,d') $>$ thresh) **then**
            $c_{d'}$++

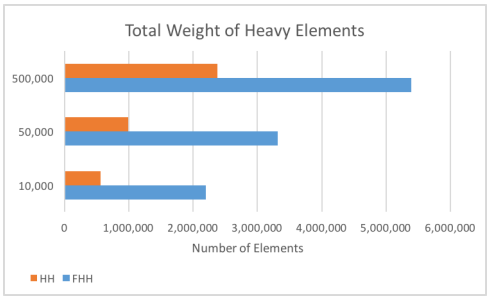

Figure 5.10: Comparison of LFCount and HH for heavy hitter weights

Effectiveness of LSH and Variance reduction

In the variance reduction step for LFCount, using a cosine distance of 0.6, there were approximately 5% collisions between clusters across LSH bins in the initial de-duplication round, and never for any subsequent iterative round. With this, we are reasonably confident that cluster splitting at 16 bit LSH and 1 billion tweets is not introducing additional error.

Accuracy of SpaceSaving Counts and Labels

In addition to understanding how LSH impacts accuracy, we also evaluate the accuracy of the labels returned by the SpaceSaving algorithm. SpaceSaving favors over-counting and

can dramatically over count elements that are not sufficiently heavy. Thus, for understanding the efficacy of the method on the Twitter dataset, we need to obtain more accurate counts for each label.

To accomplish this, we perform a second pass over the dataset using the labels returned by SpaceSaving, similarly to our approach in the previous experiment comparing LFCount and HH. Due to runtime concerns, we again use LSH to hash these labels into bins, and compare each element to the members of the corresponding bin. Thus, these counts are not precise, but are much more accurate and there is no potential for over-counting from SpaceSaving. Table 5.2 shows the results, and we observe that while FHH does over count, the labels returned represent the set of heavy clusters, and the heavy cluster IDs hold a meaningful proportion of the total fuzzy weight.

In the table, we measure at two different levels of buckets, 1.3 million and 2.6 million, and evaluate either the top $k' = 100,000$ or $k' = 500,000$ heaviest elements in those buckets. Our goal is to quantify the degree of over-counting, as well as understand the amount of fuzzy weight held by these top elements. The column "FHH Count" is the total fuzzy weight returned by LFCount, which is the over-counted approximation returned by SpaceSaving. The column "True Count (TC)" is the result of the second pass obtaining the actual counts for each cluster.

We compare the ratio of TC to FHH to determine the relative accuracy of the counts returned by the data structure. Finally, we evaluate the amount of fuzzy weight held by the top k' cluster IDs in the column "TC $\%$ of n". This gives a picture of how effective LFCount

is at giving a holistic picture of the dataset. The main result of this table is that with only 2.6 million counters, when FHH returns the 500,000 heaviest cluster IDs in those counters, those cluster IDs are representative of almost 15% of the dataset.

FHH Total Buckets	k'	k' vs n	FHH Count	True Count (TC)	TC vs FHH	TC % of n
1.3M	100,000	0.0001	212M	96M	0.45	9.6%
1.3M	500,000	0.0005	549M	116M	0.21	11.6%
2.6M	500,000	0.0005	370M	148M	0.4	14.8%

Table 5.2: SpaceSaving Accuracy in FHH

This data shows that with correct parameterization the count values provided by SpaceSaving do have a degree of over-counting, but the centroids returned do in fact hold a substantial amount of the elements in the Twitter example when compared to the amount of data stored. Notably, as k' approaches the number of FHH total buckets, the accuracy of counts decreases. This follows the properties of SpaceSaving, as the lower count buckets change labels more frequently and are more prone to error, depending on the number of heavy elements in the data set.

IMPLEMENTATION THROUGHPUT

The performance of our implementation is promising, with a throughput of over 1.5 billion tweets per day, as shown in Table 5.3. These results were obtained using 8 SpaceSaving threads, 2 LSH threads, and 3 additional thread used for queuing. Given the parallelized nature of the design, this method scales well to a larger number of threads, giving improved

throughput as needed.

Additionally, we implemented and tested code for online reporting of near duplicates [3] to evaluate the potential for LFCount to be used as a preprocessing filter for other algorithms.

LSH/SS buckets	Near Duplicate Counting	Variance Reduction	Time
16/20	No	No	10 hr
16/20	Yes	No	11.5 hr
16/20	Yes	8	12 hr
16/20	Yes	12	13.5 hr

Table 5.3: Runtimes of FHH with different features, k' = 10,000

5.6 Related Work

Approximation Algorithms for Clustering

Streaming clustering has been well-studied by the theoretical computer science community both in the case of general metric spaces[109, 109, 110, 133] and Euclidean geometries.[111–113, 134, 135] Further, approximations for k-means in both standard and[136] and streaming[137] settings are efficient in space and time. However, these methods require at least linear memory with relation to k.

Algorithms for Noisy Datasets

Other clustering techniques, such as DBSCAN,[38] relate closely to FHH. Where DBSCAN finds points and continues outwards depending on the number of near neighbors,

[3] See section 5.6.1 for discussion of Near Duplicate Detection

FHH focuses on finding the regions with the most points; which are the points with the most near neighbors.

Other methods, such as Skinny-dip,[138] also focus on clustering noisy datasets. We do not compare our FHH implementation directly with this algorithm, however, we differ in our focus on a single efficient pass over the dataset while using a small amount of memory relative to the data.

Finally, this method is similar to Clustering with Outliers,[124,125] which has also been studied in the streaming model.[139] As discussed in Section While clustering with outliers attempts to remove points to make a minimum cost clustering and carefully place the centroid, we solve the slightly easier problem of identifying clusters with many elements within a radius by placing approximate centers, without the need to place centers that optimizes distance between the points. While our algorithm may provide some approximation of a solution to the clustering with outliers problem, it is not the intention and we do not provide bounds. Future work may further consider the relationship between the two problems.

Heavy Hitters The Heavy Hitters problem is a popular problem in the field of Streaming algorithms.[18,140] Many approaches rely on *sketching* data, such as[15,16] but these methods do not work given the fuzzy nature of our data set as we describe below. Thus we relate closer to counting based algorithms, specifically.[117]

Clustering with Outliers The Fuzzy Heavy Hitters problem is distinct from the k-means with outliers problem.[141] Despite the similarities in the assumptions on the data, in the case of k-means with outliers the goal is to minimize a distance function where in our case we

are attempting to find dense regions of the data.

Relation to Location Problems This problem also has a relation to the facility location problem studied extensively by the Operations Research community.[142, 143] However, those formulations are based around significantly smaller dimensional data than our concern here. One such example is the result in the streaming mode provided by Czumaj, Lammersen, Monemizadeh and Sohler,[115] which provides an approximation algorithm for points in $\mathbb{R}^2$.

As discussed in Section 5.6, FHH and Approximate Facility Location are similar problems, and FHH can be viewed as a special case of Approximate Facility Location.

Social Media and Microblogs Social media websites like twitter have a tremendous amount of topics, but very few of them are of actual interest. These datasets are of primary interest and our goal is to provide a small space, one pass algorithm that can detect these stories. Other work such as[118, 144, 145] has explored topic detection in twitter.

Relation to Parallelization of SpaceSaving Recently, there has been a interest in parallelizing frequent element algorithms such as in[146] and.[147] It may be possible to improve the accuracy of our algorithm by using a parallelized Space Saving implementation and slightly different parameterization; however, our method obtains similar performance by running multiple instances of the Space Saving algorithms in parallel.

5.6.1 Related Problems and Other Datasets

We now briefly discuss related problems that apply to Twitter and other datasets that represent open problems and directions about how to apply FHH.

NEAR DUPLICATE DETECTION

One related problem to the FHH problem is Near Duplicate Detection, which is the process of finding similar items and grouping them. Approximate near duplicate detection has been used in the text community for handling large data sets such as websites[148].[149] An example of the usefulness of this method is to return a variety of responses to a search engine query, as opposed to many very similar results. LFCount is able to solve a slightly modified version of the problem where it can detect the near duplicates of *heavy elements*.

LFCount is able to report online in one pass whether or not a new item is a near duplicate of a heavy element among all *previously* seen items, but requires a second pass to be able to perform near duplicate detection for items that are globally heavy to the stream. LFCount is able to find near duplicates for globally heavy elements by modifying Algorithm 9 such that after every update, an item is returned as a near duplicate if any counter is incremented, and otherwise is not a near duplicate. LFCount may prove a useful application for approximately finding near duplicates, which allows for algorithms that want to process only unique elements.

TOPIC DETECTION AND TRACKING

Another area related to the FHH problem and the LFCount algorithm is Topic Detection and Tracking (TDT). This problem has been well studied and many algorithms have been directly compared thanks to a common NIST dataset of newspaper articles.[150] Different methods have also been applied to Twitter[151].[152] A major goal of topic detection is to find

information related to events and be able to report that a new event has occurred. For datasets where important events are heavy, FHH provides an efficient solution for finding these events. On Twitter, while these topics are not extremely heavy, we do find event data, as shown in Table 5.4.

In Table 5.4 we show topics found in output processing a 50 million Tweet sample. We this smaller volume as it represents a smaller timeframe for tweets to be distributed over (3 months versus 3 years), which improves performance for finding events. We seen in this table that we are able to find instances of sporting events, technology, and news. We observed that many of the notable events in the dataset were retweets. Retweets are the same message shared multiple times, which are not necessarily fuzzy in nature, but any embellishment or modification of the message is caught by FHH and considered the same message. An important step in testing on applying FHH to the TDT problem in future work would be evaluating the performance on finding topics where there are fewer retweets.

Tweet Text	Count
Giambi comes through with the walk off 3 run homer. Crazy 9th inning, but the bottom line is a good win for the Rockies. Tacos tomorrow.	1200
Key: RT @androidcentral: Minecraft Pocket Edition now available in the Android Market for the Xperia Play http://t.co/CQxzCVH #android	780
RT @ndtv: Delhi Police shouldnt have allowed themselves to be used as a tool in the hands of the govt, says Kiran Bedi	490
RT @freepsports: #Michigan beats #NotreDame, 35-31	425

Table 5.4: Topic and Event Heavy Cluster IDs and Counts (50 million Tweets)

An open problem is to use LFCount to make a preprocessing filter for TDT algorithms. The goal of this construction would be to remove the *stop messages* introduced in Section 5.5.2. These messages, which may not have value as topics, but are very frequent, can be removed by LFCount, in turn improving the performance of the TDT algorithm.

A variation of TDT, where the goal is to find the first instance of an event is Streaming First Story Detection,[118] is discussed in the next section.

FINDING HALOS

Streaming algorithms have recently been very successful at finding patterns in the distribution of matter and galaxies in the universe called halos.[153] This approach brings a

problem once reserved for super computers to a laptop while yielding accurate results. In this task, the goal is to return k halos, and the size of the particle stream, 10^{10}, is large enough where FHH may be more effective than other clustering methods. The FHH construction may help further, as the current methodology requires drawing a static grid over space and finding heavy cells, FHH may be able to group these data points more efficiently.

5.7 Comparison with First Story Detection

STREAMING FIRST STORY DETECTION

There are some important similarities with this work and the paper Streaming First Story Detection (SFSD)[118] by Petrovic, Osborne, and Lavrenko. The goal of First Story Detection is to determine if a story arriving is the first of its kind. Similarly to LFCount, they use Twitter data as their test dataset. To achieve this goal, the authors create an algorithm using LSH and holding a constant number of the most recent tweets. This combination allows the algorithm to quickly evaluate the newness of incoming tweets in a memory efficient, streaming way.

To compare the algorithms used to solve the respective problems; LFCount also uses a constant number of items per LSH bit signature. However, by using a different underlying data structure for comparison and storage, we are able to gain a major improvement in accuracy per bucket using the LFCount algorithm for solving the FHH problem. This in turn allows us to use fewer buckets, which directly improves speed and memory, compared to using the SFSD algorithm to try and solve the FHH problem. This gain lies in the efficiency

for all Documents d in corpus **do**
add d to LSH
$S \leftarrow$ contents of the LSH bucket where d lands
for all Documents d' in S **do**
$c =$ distance(d, d')
if $c < dis_{min}(d)$ **then**
$dis_{min}(d) \leftarrow c$
if $dis_{min}(d) > t$ **then**
compare d to a fixed number of most
recent documents as in traditional document
comparison and update dis_{min} as necessary
assign score dis_{min} to d
add d to inverted index.

Figure 5.11: SFSD Algorithm

for all Documents d in corpus **do**
add d to LSH
$S \leftarrow$ the SpaceSaver data structure in the LSH bin where d lands
Insert d into S
for all For s in S **do**
$c =$ distance(d, s)
if $c > thresh$ **then**
s+=1
else
Evict s with smallest count, replace with d, keep old counter
Variance Reduction (See Algorithm 7)

Figure 5.12: LFCount Algorithm

of our modified Space-Saving data structure where we preserve elements in memory based on their frequency, not by timestamp as is the case with SFSD.

A notable difference between the two algorithms is that LFCount is not capable of finding new stories at the point of origination (finding the first node in a thread) due to a smaller number of stored elements. However, Petrovic et. al focus on identifying tweets that are in "threads that grow the fastest", a task we are able to accomplish. By using an FHH based approach, we are able to find the heaviest threads from the Topic Detection perspective. Further, with modification, the algorithm can also account for the notion of only finding *recent* events by using some notion of decay or using sliding a window heavy hitter algorithm, and we leave this improvement for future work.

In terms of a performance and memory comparison, we maintain k copies of the Space Saver data structure with b buckets each. By comparison, SFSD stores S items per LSH value, where $S >> b$

Further, SFSD uses multiple hash tables, as finding a near neighbor is critical for each element. We don't have this requirement, as we simply need a majority of items per cluster to be detected, and use a single hash table at a time. We perform offline variance reduction to recombine any clusters which may have been divided by LSH in this way.

PERFORMANCE

We note that while the algorithm is similar to that of SFSD, due to the nature of their algorithm, similar accuracy requires many more buckets per bin, and thus has a large constant multiplier in time, and does not allow a direct time comparison.

5.8 Future Work

In introducing this problem, we also open up many experiments and questions that may be addressed in future work, both algorithmic and in implementation. We believe at this point the most important algorithmic step forward would be evaluating the applicability of sketch based algorithms for this problem, as such an algorithm could provide both accuracy and performance improvements. In terms of application improvements, a key focus moving forward will be determining a method to automatically parameterize this algorithm, as the current implementation is highly dependent on domain knowledge.

In the opposite direction of generalization, one approach for future work would be to try and apply specialized domain specific methods. In our experimentation we do not use any text preprocessing outside of tokenization, and do not use any text tools such as TFIDF, noun identification, or other methods commonly used for higher accuracy in analyzing text. Using these methods, or similar methods for other fields , may show how useful FHH is for a variety of problems.

An additional direction for future work is considering the importance of the age of data and how that impacts the importance of heavy hitters. Two approaches come to mind as reasonable approaches. The first would be a sliding window based approach where multiple copies of a data structure such as LFCount are started at various time intervals and run for a set amount of time. The second approach would be to slowly decay counter values over time to allow new heavy hitters to more rapidly overtake older elements. Developing techniques

may help adapt FHH to solve additional problems.

5.9 Chapter Conclusions

In sum, we introduce a new problem definition for finding groups of heavy elements in vector space. We provide the LFCount to solve this problem. We use Twitter as an example of how this problem and solution is applicable to practical datasets. Our future work might consider increased effort in processing and filtering the content for further dataset specific improvements on processing the Twitter dataset. Beyond Twitter, this problem definition and approach are relevant to other domains, such as other social media streams and other settings with sparse vectors such as data from many heterogeneous sensors.

Chapter 6

Conclusion

With all of this work taken together, I believe we present a strong argument; not only furthering the evidence for the usefulness of streaming algorithms for fast datasets, but for the importance of designing theoretical methods with the goal of solving practical problems, and the capability of the resulting software.

This thesis covers a wide variety of topics: sampling, heavy hitters, frequency moments, and matrix functions. The variety and complexity of these problems speaks to the usefulness of the approach. Most importantly, each solution is built to help improve practical software that processes fast datasets.

Data will not get smaller. The world will not slow down. The best path forward is one where theory can drive practice and practice can drive theory.

Bibliography

[1] I. LiveStats. Twitter usage statistics. [Online]. Available: http://www.internetlivestats.com/twitter-statistics/

[2] J. M. Hawkins. (2015, November) 2016: The year of the data center. [Online]. Available: http://www.datacenterknowledge.com/archives/2015/11/23/2016-year-data-center/

[3] I. Gartner. Gartner says 6.4 billion connected ”things” will be in use in 2016, up 30 percent from 2015gartner says 6.4 billion connected ”things” will be in use in 2016, up 30 percent from 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317

[4] T. SImonite. (2016, March) Intel puts the brakes on moore’s law. [Online]. Available: https://www.technologyreview.com/s/601102/intel-puts-the-brakes-on-moores-law/

[5] M. Yu, L. Jose, and R. Miao, “Software defined traffic measurement with opensketch,” in *Proceedings of the 10th USENIX Conference on Networked Systems Design and*

Implementation, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482631

[6] A. Gilbert and P. Indyk, "Sparse recovery using sparse matrices." Institute of Electrical and Electronics Engineers, 2010.

[7] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, "Hash kernels for structured data," *Journal of Machine Learning Research*, vol. 10, no. Nov, pp. 2615–2637, 2009.

[8] S. Schechter, C. Herley, and M. Mitzenmacher, "Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks," in *Proceedings of the 5th USENIX conference on Hot topics in security*. USENIX Association, 2010, pp. 1–8.

[9] V. Braverman and G. Vorsanger, "Sampling from dense streams without penalty," in *International Computing and Combinatorics Conference*. Springer, 2014, pp. 13–24.

[10] V. Braverman, R. Ostrovsky, and G. Vorsanger, "Weighted sampling without replacement from data streams," *Information Processing Letters*, vol. 115, no. 12, pp. 923–926, 2015.

[11] V. Braverman, A. Roytman, and G. Vorsanger, "Approximating subadditive hadamard functions on implicit matrices," *RANDOM 2016*, 2016.

BIBLIOGRAPHY

[12] S. Muthukrishnan, "Data streams: algorithms and applications," *Found. Trends Theor. Comput. Sci.*, vol. 1, no. 2, pp. 117–236, Aug. 2005. [Online]. Available: http://dx.doi.org/10.1561/0400000002

[13] D. P. Woodruff, "Frequency moments," in *Encyclopedia of Database Systems*, 2009, pp. 1169–1170.

[14] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, 1999.

[15] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ser. ICALP '02. London, UK, UK: Springer-Verlag, 2002, pp. 693–703. [Online]. Available: http://dl.acm.org/citation.cfm?id=646255.684566

[16] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jalgor.2003.12.001

[17] K. G. Larsen, J. Nelson, H. L. Nguyên, and M. Thorup, "Heavy hitters via cluster-preserving clustering," *57th Annual Foundations of Computer Science (FOCS)*, 2016.

[18] V. Braverman and R. Ostrovsky, "Approximating large frequency moments with pick-and-drop sampling," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2013, pp. 42–57.

[19] P. Indyk and D. Woodruff, "Optimal approximations of the frequency moments of data streams," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, ser. STOC '05. New York, NY, USA: ACM, 2005, pp. 202–208. [Online]. Available: http://doi.acm.org/10.1145/1060590.1060621

[20] V. Braverman, J. Katzman, C. Seidell, and G. Vorsanger, "An optimal algorithm for large frequency moments using o(nˆ(1-2/k)) bits," in *APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, 2014, pp. 531–544. [Online]. Available: http://dx.doi.org/10.4230/LIPIcs.APPROX-RANDOM.2014.531

[21] V. Braverman and R. Ostrovsky, "Zero-one frequency laws," in *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, ser. STOC '10. New York, NY, USA: ACM, 2010, pp. 281–290. [Online]. Available: http://doi.acm.org/10.1145/1806689.1806729

[22] V. Braverman and S. R. Chestnut, "Universal Sketches for the Frequency Negative Moments and Other Decreasing Streaming Sums," in *APPROX/RANDOM 2015*, ser. Leibniz International Proceedings in Informatics (LIPIcs), N. Garg, K. Jansen, A. Rao, and J. D. P. Rolim, Eds., vol. 40. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 591–605. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2015/5325

[23] C. C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proceedings of the 32nd international conference on Very large data bases*,

ser. VLDB '06. VLDB Endowment, 2006, pp. 607–618. [Online]. Available: http://dl.acm.org/citation.cfm?id=1182635.1164180

[24] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '02. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002, pp. 633–634. [Online]. Available: http://dl.acm.org/citation.cfm?id=545381.545465

[25] Z. Bar-Yossef, "The complexity of massive data set computations," Ph.D. dissertation, Berkeley, CA, USA, 2002, aAI3183783.

[26] V. Braverman, R. Ostrovsky, and D. Vilenchik, "How hard is counting triangles in the streaming model?" in *Automata, Languages, and Programming*, ser. Lecture Notes in Computer Science, F. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, Eds. Springer Berlin Heidelberg, 2013, vol. 7965, pp. 244–254. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39206-1_21

[27] V. Braverman, R. Ostrovsky, and G. Vorsanger, "Weighted sampling without replacement from data streams," *submitted*, 2013.

[28] V. Braverman, R. Ostrovsky, and C. Zaniolo, "Optimal sampling from sliding windows," in *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ser. PODS

'09. New York, NY, USA: ACM, 2009, pp. 147–156. [Online]. Available: http://doi.acm.org/10.1145/1559795.1559818

[29] S. Chaudhuri, R. Motwani, and V. Narasayya, "On random sampling over joins," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 263–274. [Online]. Available: http://doi.acm.org/10.1145/304182.304206

[30] J. S. Vitter, *Random sampling with a reservoir*. v.11 n.1, pp.37–57: ACM Transactions on Mathematical Software (TOMS), 1985.

[31] B. Claise, "Cisco systems netflow services export version 9," vol. RFC 3954.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 6.

[33] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, no. 6, 1999, pp. 518–529.

[34] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ser. STOC '02. New York, NY, USA: ACM, 2002, pp. 380–388. [Online]. Available: http://doi.acm.org/10.1145/509907.509965

[35] A. McGregor, A. Pavan, S. Tirthapura, and D. Woodruff, "Space-efficient estimation of statistics over sub-sampled streams," in *Proceedings of the 31st symposium on*

Principles of Database Systems, ser. PODS '12. New York, NY, USA: ACM, 2012, pp. 273–282. [Online]. Available: http://doi.acm.org/10.1145/2213556.2213594

[36] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *CoRR*, vol. abs/1411.4357, 2014. [Online]. Available: http://arxiv.org/abs/1411.4357

[37] V. Braverman and R. Ostrovsky, "Measuring independence of datasets," in *Proceedings of the 42nd annual ACM Symposium on Theory of Computing*, 2010.

[38] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[39] F. Rusu and A. Dobra, "Sketching sampled data streams," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ser. ICDE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 381–392. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2009.31

[40] A. Chakrabarti, S. Khot, and X. Sun, "Near-optimal lower bounds on the multi-party communication complexity of set disjointness," in *IEEE Conference on Computational Complexity*, 2003, pp. 107–117.

[41] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, "An information statistics approach to data stream and communication complexity," in *Proceedings of the 43rd Symposium on Foundations of Computer Science*, ser. FOCS '02. Washington,

DC, USA: IEEE Computer Society, 2002, pp. 209–218. [Online]. Available: http://dl.acm.org/citation.cfm?id=645413.652164

[42] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, ser. RANDOM '02. London, UK, UK: Springer-Verlag, 2002, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=646978.711822

[43] P. Beame, T. S. Jayram, and A. Rudra, "Lower bounds for randomized read/write stream algorithms," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, ser. STOC '07. New York, NY, USA: ACM, 2007, pp. 689–698. [Online]. Available: http://doi.acm.org/10.1145/1250790.1250891

[44] V. Braverman, J. Katzman, C. Seidell, and G. Vorsanger, "Approximating large frequency moments with $o(n^{1-2/k})$ bits," *CoRR*, vol. abs/1401.1763, 2014.

[45] V. Braverman and R. Ostrovsky, "Smooth histograms for sliding windows," in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 283–293. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2007.63

[46] ——, "Approximating large frequency moments with pick-and-drop sampling," in *APPROX-RANDOM*, 2013, pp. 42–57.

[47] ——, "Generalizing the layering method of Indyk and Woodruff: Recursive sketches for frequency-based vectors on streams," in *APPROX-RANDOM*, 2013, pp. 58–70.

[48] A. Chakrabarti, G. Cormode, and A. McGregor, "Robust lower bounds for communication and stream computation," in *Proceedings of the 40th annual ACM symposium on Theory of computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 641–650. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374470

[49] D. Coppersmith and R. Kumar, "An improved data stream algorithm for frequency moments," in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '04. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2004, pp. 151–156. [Online]. Available: http://dl.acm.org/citation.cfm?id=982792.982815

[50] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan, "Comparing data streams using hamming norms (how to zero in)," in *Proceedings of the 28th international conference on Very Large Data Bases*, ser. VLDB '02. VLDB Endowment, 2002, pp. 335–345. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287369.1287399

[51] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, "An approximate l1-difference algorithm for massive data streams," in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, ser. FOCS '99. Washington,

DC, USA: IEEE Computer Society, 1999, pp. 501–. [Online]. Available: http://dl.acm.org/citation.cfm?id=795665.796530

[52] S. Ganguly, "Estimating frequency moments of data streams using random linear combinations," in *APPROX-RANDOM*, 2004, pp. 369–380.

[53] S. Ganguly and G. Cormode, "On estimating frequency moments of data streams," in *Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques*, ser. APPROX '07/RANDOM '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 479–493. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74208-1_35

[54] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee, "Estimating statistical aggregates on probabilistic data streams," in *PODS '07: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 2007, pp. 243–252.

[55] D. M. Kane, J. Nelson, and D. P. Woodruff, "On the exact space complexity of sketching and streaming small norms," in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, 2010.

[56] ——, "An optimal algorithm for the distinct elements problem," in *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of*

database systems, ser. PODS '10. New York, NY, USA: ACM, 2010, pp. 41–52. [Online]. Available: http://doi.acm.org/10.1145/1807085.1807094

[57] P. Li, "Compressed counting," in *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '09. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009, pp. 412–421. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496770.1496816

[58] J. Nelson and D. P. Woodruff, "Fast Manhattan sketches in data streams," in *PODS '10: Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems of data*. New York, NY, USA: ACM, 2010, pp. 99–110.

[59] D. Woodruff, "Optimal space lower bounds for all frequency moments," in *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2004, pp. 167–175.

[60] V. Braverman and R. Ostrovsky, "Zero-one frequency laws," in *Proceedings of the 42nd ACM symposium on Theory of computing*, ser. STOC '10. New York, NY, USA: ACM, 2010, pp. 281–290. [Online]. Available: http://doi.acm.org/10.1145/1806689.1806729

[61] S. Bhattacharyya, A. Madeira, S. Muthukrishnan, and T. Ye, "How to scalably and accurately skip past streams," in *Proceedings of the 2007 IEEE 23rd International*

Conference on Data Engineering Workshop, ser. ICDEW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 654–663. [Online]. Available: http://dx.doi.org/10.1109/ICDEW.2007.4401052

[62] "Open problems in data streams and related topics," in *IITK workshop on algorithms for data streams. http://www.cse.iitk.ac.in/users/sganguly/data-stream-probs.pdf*.

[63] N. Johnson, A. Kemp, and S. Kotz, *Univariate discrete distributions*. Wiley-Interscience, 2005.

[64] D. E. Knuth, *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[65] V. V. Vazirani, *Approximation algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[66] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 1013–1020. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807278

[67] H. Popkin, "Facebook hits 1 billion users," http://www.today.com/tech/facebook-hits-1-billion-users-6271714, October 2012.

[68] D. Lee, "Global internet slows after 'biggest ddos in history'," http://www.bbc.co.uk/news/technology-21954636, March 2013.

[69] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming algorithms for robust, real-time detection of ddos attacks," in *Proceedings of the 27th International Conference on Distributed Computing Systems*, ser. ICDCS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 4–. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2007.142

[70] Y. Xiang, Y. Lin, W. Lei, and S. Huang, "Detecting ddos attack based on network self-similarity," *IEE Proceedings-Communications*, vol. 151, no. 3, pp. 292–295, 2004.

[71] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 39–53, 2004.

[72] M. Goldstein, M. Reif, A. Stahl, and T. Breuel, "Server-side prediction of source ip addresses using density estimation," in *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, 2009, pp. 82–89.

[73] D. Berend and T. Tassa, "Improved bounds on bell numbers and on moments of sums of random variables," in *Probability and Mathematical Statistics*, vol. 30, no. 2, pp. 185–205.

[74] P. Bullen, D. Mitrinović, and P. Vasić, *Means and their inequalities*, ser. Mathematics

and its applications. East European series. D. Reidel, 1988. [Online]. Available: http://books.google.com/books?id=cjzvAAAAMAAJ

[75] R. Motwani and P. Raghavan, *Randomized algorithms*. New York, NY: Cambridge University Press, 1995.

[76] D. E. Knuth, *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[77] Y. Tille, *Sampling Algorithms*. Verlag: Springer, 2006.

[78] F. Olken and D. Rotem, "Random sampling from databases: a survey," *Statistics and Computing*, vol. 5, no. 1, pp. 25–42, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF00140664

[79] M. Szegedy, "The DLT priority sampling is essentially optimal," in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, ser. STOC '06. New York, NY, USA: ACM, 2006, pp. 150–158. [Online]. Available: http://doi.acm.org/10.1145/1132516.1132539

[80] T. Johnson, S. Muthukrishnan, and I. Rozenbaum, "Sampling algorithms in a stream operator," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066159

[81] P. B. Gibbons and Y. Matias, "New sampling-based summary statistics for improving approximate query answers," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '98. New York, NY, USA: ACM, 1998, pp. 331–342. [Online]. Available: http://doi.acm.org/10.1145/276304.276334

[82] G. Frahling, P. Indyk, and C. Sohler, "Sampling in dynamic data streams and applications," in *Proceedings of the twenty-first annual symposium on Computational geometry*, ser. SCG '05. New York, NY, USA: ACM, 2005, pp. 142–149. [Online]. Available: http://doi.acm.org/10.1145/1064092.1064116

[83] M. Kolonko and D. Wäsch, "Sequential reservoir sampling with a nonuniform distribution," *ACM Trans. Math. Softw.*, vol. 32, no. 2, pp. 257–273, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1141885.1141891

[84] K.-H. Li, "Reservoir-sampling algorithms of time complexity o(n(1 + log(n/n)))," *ACM Trans. Math. Softw.*, vol. 20, no. 4, pp. 481–493, Dec. 1994. [Online]. Available: http://doi.acm.org/10.1145/198429.198435

[85] B. Rosén, "Asymptotic theory for successive sampling with varying probabilities without replacement, I," *The Annals of Mathematical Statistics*, vol. 43, no. 2, pp. 373–397, 1972. [Online]. Available: http://www.jstor.org/stable/2239977

[86] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Inf. Process. Lett.*, vol. 97, no. 5, pp. 181–185, Mar. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.ipl.2005.11.003

[87] E. Cohen and H. Kaplan, "Tighter estimation using bottom k sketches," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 213–224, Aug. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1453856.1453884

[88] C.-K. Wong and M. C. Easton, "An efficient method for weighted sampling without replacement," *SIAM Journal on Computing*, vol. 9, no. 1, pp. 111–113, 1980.

[89] E. Cohen and H. Kaplan, "Summarizing data using bottom-k sketches," in *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. ACM, 2007, pp. 225–234.

[90] P. S. Efraimidis, "Weighted random sampling over data streams," *arXiv preprint arXiv:1012.0256*, 2010.

[91] E. L. Lehmann and J. P. Romano, *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[92] V. Poosala and Y. E. Ioannidis, "Selectivity estimation without the attribute value independence assumption," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[93] R. Kimball and J. Caserta, "The data warehouse etl toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data," 2004.

[94] P. Indyk and A. McGregor, "Declaring independence via the sketching of sketches,"

in *Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms*, 2008.

[95] D. Guillot, A. Khare, and B. Rajaratnam, "Complete characterization of hadamard powers preserving loewner positivity, monotonicity, and convexity," *Journal of Mathematical Analysis and Applications*, vol. 425, no. 1, pp. 489–507, 2015.

[96] R. A. Horn and C. R. Johnson, "Topics in matrix analysis," *Cambridge University Presss, Cambridge*, 1991.

[97] P. Indyk, "Stable distributions, pseudorandom generators, embeddings, and data stream computation," *J. ACM*, vol. 53, no. 3, pp. 307–323, 2006.

[98] V. Braverman and R. Ostrovsky, "Generalizing the layering method of Indyk and Woodruff: Recursive sketches for frequency-based vectors on streams," *Accepted to the 16th. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'2013).*, 2013.

[99] N. Alon, A. Andoni, T. Kaufman, K. Matulef, R. Rubinfeld, and N. Xie, "Testing k-wise and almost k-wise independence," in *Proceedings of the 39th annual ACM Symposium on Theory of Computing*, 2007.

[100] N. Alon, O. Goldreich, and Y. Mansour, "Almost k-wise independence versus k-wise independence," *Information Processing Letters*, vol. 88, no. 3, pp. 107–110, 2003.

[101] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White, "Testing

random variables for independence and identity," in *Proceedings of the 42nd annual IEEE Symposium on Foundations of Computer Science*, 2001.

[102] T. Batu, R. Kumar, and R. Rubinfeld, "Sublinear algorithms for testing monotone and unimodal distributions," in *Proceedings of the 36th annual ACM Symposium on Theory of Computing*, 2004.

[103] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White, "Testing that distributions are close," in *Proceedings of the 41st annual IEEE Symposium on Foundations of Computer Science*, 2000.

[104] ——, "Testing closeness of discrete distributions," *Journal of the ACM*, vol. 60, no. 1, p. 4, 2013.

[105] V. Braverman, K.-M. Chung, Z. Liu, M. Mitzenmacher, and R. Ostrovsky, "AMS Without 4-Wise Independence on Product Domains," in *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, 2010.

[106] A. McGregor and H. T. Vu, "Evaluating bayesian networks via data streams," in *Proceedings of the 21st International Computing and Combinatorics Conference*, 2015.

[107] R. Rubinfeld and R. A. Servedio, "Testing monotone high-dimensional distributions," in *Proceedings of the 37th annual ACM Symposium on Theory of Computing*, 2005.

[108] A. Sahai and S. Vadhan, "A complete problem for statistical zero knowledge," *Journal of the ACM*, vol. 50, no. 2, pp. 196–249, 2003.

[109] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in *Proceedings 41st FOCS*, 2000., pp. 359–366.

[110] M. Charikar, L. O'Callaghan, and R. Panigrahy, "Better streaming algorithms for clustering problems," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, ser. STOC '03. New York, NY, USA: ACM, 2003, pp. 30–39. [Online]. Available: http://doi.acm.org/10.1145/780542.780548

[111] S. Har-Peled and S. Mazumdar, "Coresets for k-means and k-median clustering and their applications," in *Proceedings36th STOC*, 2004, pp. 291–300.

[112] G. Frahling and C. Sohler, "Coresets in dynamic geometric data streams," in *Proceedings37th STOC*, 2005., pp. 209–217.

[113] K. Chen, "On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications," *SIAM Journal on Computing*, vol. 39, no. 3, pp. 923–947, 2009.

[114] D. Feldman, M. Monemizadeh, and C. Sohler, "A PTAS for k-means clustering based on weak coresets," in *Proceedings23rd Symposium on Computational Geometry*, 2007., pp. 11–18.

[115] A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler, "$(1+\varepsilon)$-approximation

for facility location in data streams," in *Proceedings24th SODA*, 2013., pp. 1710–1728.

[116] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1530–1541, Aug. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1454159.1454225

[117] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Database Theory-ICDT 2005*. Springer, 2005, pp. 398–412.

[118] S. Petrović, M. Osborne, and V. Lavrenko, "Streaming first story detection with application to twitter," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 181–189.

[119] M. Ackerman and S. Dasgupta, "Incremental clustering: The case for extra clusters," in *Advances in Neural Information Processing Systems*, 2014, pp. 307–315.

[120] M.-F. Balcan, A. Blum, and S. Vempala, "A discriminative framework for clustering via similarity functions," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM, 2008, pp. 671–680.

[121] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, Jan 2003.

[122] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, Apr 2004.

[123] D. M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[124] K. Chen, "A constant factor approximation algorithm for k-median clustering with outliers." in *SODA*, vol. 8. Citeseer, 2008, pp. 826–835.

[125] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan, "Algorithms for facility location problems with outliers," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 642–651.

[126] D. B. Shmoys, É. Tardos, and K. Aardal, "Approximation algorithms for facility location problems," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997, pp. 265–274.

[127] D. Ravichandran, P. Pantel, and E. Hovy, "Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005, pp. 622–629.

[128] A. Jansen and B. Van Durme, "Efficient spoken term discovery using randomized

algorithms," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 401–406.

[129] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jalgor.2003.12.001

[130] [Online]. Available: https://dev.twitter.com/docs

[131] L. Ertöz, M. Steinbach, and V. Kumar, "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data." in *SDM*. SIAM, 2003, pp. 47–58.

[132] H. Schütze, "Introduction to information retrieval," in *Proceedings of the international communication of association for computing machinery conference*, 2008.

[133] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan, "Maintaining variance and k-medians over data stream windows," in *Proceedings9th PODS*, 2003., pp. 234–243.

[134] P. Indyk, "Algorithms for dynamic geometric problems over data streams," in *Proceedings36th STOC*, 2004., pp. 373–380.

[135] S. Har-Peled and A. Kushal, "Smaller coresets for k-median and k-means clustering," in *Proceedings21st Symposium on Computational Geometry*, 2005., pp. 126–134.

[136] R. Ostrovsky, Y. Rabani, L. Schulman, and C. Swamy, "The effectiveness of Lloyd-type methods for the k-means problem," in *FOCS*, 2006.

BIBLIOGRAPHY

[137] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku, "Streaming k-means on well-clusterable data," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '11. SIAM, 2011, pp. 26–40. [Online]. Available: http://dl.acm.org/citation.cfm?id=2133036.2133039

[138] S. Maurus and C. Plant, "Skinny-dip: Clustering in a sea of noise," *KDD*, 2016.

[139] R. M. McCutchen and S. Khuller, "Streaming algorithms for k-center clustering with outliers and with anonymity," in *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2008, pp. 165–178.

[140] G. Cormode and M. Hadjieleftheriou, "Finding the frequent items in streams of data," *Communications of the ACM*, vol. 52, no. 10, pp. 97–105, 2009.

[141] M. Charikar, L. O'Callaghan, and R. Panigrahy, "Better streaming algorithms for clustering problems," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. ACM, 2003, pp. 30–39.

[142] A. Meyerson, "Online facility location," in *FOCS*, 2001.

[143] Z. Drezner and H. W. Hamacher, Eds., *Facility Location: Applications and Theory*. Springer Verlag, 2004.

[144] M. Osborne, S. Moran, R. McCreadie, A. V. Lunen, M. Sykora, E. Cano, N. Ireson, C. MacDonald, I. Ounis, Y. He, T. Jackson, F. Ciravegna, and A. O'Brien, "Real-time detection, tracking and monitoring of automatically discovered events in social media,"

in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, USA, 2014.

[145] W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K. Wang, "Topicsketch: Real-time bursty topic detection from twitter," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 2013, pp. 837–846.

[146] P. Roy, J. Teubner, and G. Alonso, "Efficient frequent item counting in multi-core hardware," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1451–1459.

[147] M. Cafaro, M. Pulimeno, and P. Tempesta, "A parallel space saving algorithm for frequent items and the riemann-hurwitz zeta distribution," *arXiv preprint arXiv:1401.0702*, 2014.

[148] M. Theobald, J. Siddharth, and A. Paepcke, "Spotsigs: robust and efficient near duplicate detection in large web collections," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2008, pp. 563–570.

[149] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 1157–1166, 1997.

[150] M. Connell, A. Feng, G. Kumaran, H. Raghavan, C. Shah, and J. Allan, "Umass at tdt 2004," in *Topic Detection and Tracking Workshop Report*, vol. 19, 2004.

[151] S. Phuvipadawat and T. Murata, "Breaking news detection and tracking in twitter," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, vol. 3. IEEE, 2010, pp. 120–123.

[152] M. Cataldi, L. Di Caro, and C. Schifanella, "Emerging topic detection on twitter based on temporal and social terms evaluation," in *Proceedings of the Tenth International Workshop on Multimedia Data Mining*. ACM, 2010, p. 4.

[153] Z. Liu, N. Ivkin, L. Yang, M. Neyrinck, G. Lemson, A. Szalay, V. Braverman, T. Budavari, R. Burns, and X. Wang, "Streaming algorithms for halo finders," in *e-Science (e-Science), 2015 IEEE 11th International Conference on*. IEEE, 2015, pp. 342–351.

[154] (2016). [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[155] (2016, 11). [Online]. Available: https://www.khronos.org/opencl/

[156] [Online]. Available: https://wiki.tiker.net/CudaVsOpenCL

[157] (2016). [Online]. Available: http://jcuda.org/

[158] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 101–114.

BIBLIOGRAPHY

[159] (2016, 11). [Online]. Available: http://www-03.ibm.com/software/products/en/ibm-streams

[160] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix A

Challenges in Developing Streaming Algorithms

APPENDIX A. CHALLENGES IN DEVELOPING STREAMING ALGORITHMS

In the course of both implementing and observing others implement streaming algorithms, the challenge is clear.

This section comes from observations of my professional experience working at Raytheon BBN Technologies as well as implementing algorithms in my academic research. In an engineering setting, when a novel and difficult problem arise there is a clear need for quickly evaluating the applicability of different techniques. The barrier of entry to experimentation and evaluation makes or breaks the likelihood of a method being used, regardless of whether or not it is objectively the best method to solve the problem in a vacuum.

Understandably, in my research I largely found that papers that successfully implement streaming algorithms do not discuss some of the systems engineering difficulties that they face, instead focusing on design choices. Selecting a programming language, finding test data, and rapid prototyping are all key steps in evaluating the usefulness of a new algorithm, and these steps require an almost disjoint skill set to that of theoretical algorithm development.

Thus, having seen both sides of this problem, I believe it is of critical importance that the streaming and data science communities work together to remove the very high barrier of entry to implementing and testing streaming algorithms, so that exciting new theoretical algorithms are not passed over in practical settings.

Programming Languages:

Programming languages present a serious road-bump for proof of concept algorithm implementation. Variance reduction is a critical step in many randomized algorithms, and

in streaming algorithms this usually takes the form of running the algorithm many times in parallel and then comparing the results to get a final answer (e.g.,[20,37,129] and most of the other sketches referenced in this thesis). Parallel programming adds a great deal of additional programming challenges and pitfalls, as running a polylogarthmic number of copies of an algorithm can require careful memory and CPU management.

Moreover, popular implementations of Python and Ruby use a Global Interpreter Lock, which prevents multiple threads from running simultaneously. Evaluating on such large datasets demands threads to run at the same time; and combined with the fact that interpreted languages such as Python are slower, this pushes development to compiled languages, where comparative development time can be longer as well.

Inside the space of compiled languages, further considerations such as leveraging GPU processing[154,155] also can give a major improvement performance and improve parallelization. However, different GPU libraries perform differently with different hardware (see this website dedicated to comparing NVIDIA's CUDA against the open source Open CL[156]). Additionally, different levels of software maturity, such as the difference between jcuda,[157] Java bindings for CUDA, and NVIDIA's C++ CUDA library, can also impact the gains obtained by using GPU processing.

Data and Ground Truth:

As shown in Chapter 2, new streaming algorithms can provide dramatically improved performance over previous methods on datasets with particular properties, however, as datasets grow large, establishing accurate ground truth becomes increasingly difficult. While

in some cases it may be possible to generate synthetic datasets with these characteristics, there are cases where analyzing practical data is preferable.

Even if given an unlimited amount of memory, many traditional algorithms require $O(n^2)$ computation for exact solutions. For larger datasets, this may take years, but for carefully chosen test datasets, weeks or months is an option. Spending a month to generate ground truth is expensive when prototyping a new algorithm. Further, this can leads to a variety of different datasets, with different sizes and qualities. However, if in the near future, standard datasets can be prepared, we would in turn see better cross paper comparison, allowing easier selection of the best algorithms, and improving the path to transitioning cutting-edge theory.

The need for Prototypes:

The purpose of this section is to highlight the difficulty in moving these algorithms from theory to practice. From experience, I have found the strongest argument that can be made in a software development is a working prototype. While many specific algorithms are available implemented and made publicly available, we have not observed a general push to provide a baseline set of tools for supporting streaming algorithm implementation. Some groups have made platforms for improving development, such as open-sketch for network monitoring,[5] or the Univmon project[158] but there is no such platform we are aware of for general software development. Without concrete implementations of these algorithms which can be tested and observed outside of the theory community, these algorithms may be overlooked.

Some big data development environments, such as IBM streams,[159] support creating software that supports lots of hardware in a parallelized way, with a focus on large development teams. This is a critical task in its own right; however, these kind of systems do not appear to do enough help move new complex algorithms from theory to a software prototype.

A Path Forward for the Data Science Community:

With these three challenges, we are faced with a very complex problem. We have algorithms based in complex mathematics, that require increased development time due to the use of compiled programming languages, and we need appropriate datasets for testing. Taken together, this lack of tools represents a major boundary for improving classic results with good asymptotic bounds, but may be able to perform better with modifications in software. The main observation of this thesis is the need to develop of a library of functionality to quickly implement cutting edge streaming algorithms to evaluate their effectiveness on practical datasets.

This library solves a few problems, one, it provides a strong guideline for a programming language to be used, allowing for future API and function development to be reused or re-purposed by future algorithm creators. Further, giving developers the building blocks to build algorithms can dramatically increase development time, as well as decreasing the knowledge barrier by not requiring the developer to be experts in commonly used primitives such as random generators or hash functions not available in standard libraries such as LSH in Scikit Learn for Python.[160] Finally, it provides easier cross comparison between algorithms when identical code primitives for randomization are used to evaluate practical

memory usage as well as execution speed.

Creating this set of tools greatly reduces the complexity many individual tasks on the road to use. However, more importantly, it helps bridge the very large and very important gap between theory and practice.

Vita

Gregory Alexander Vorsanger received his BS degree in Computer Engineering from Johns Hopkins University in 2011, where he graduated with General University Honors, as well as The Charles A. Conklin Award in recognition of his academic achievements. In 2012, he earned a Masters degree in Security Informatics, and began working part time at Raytheon BBN Technologies. In 2013, Greg enrolled in the Computer Science PhD program, where his work focused on Streaming Algorithms. Greg received an MSE in Computer Science in 2014, and is currently a Staff Scientist at Raytheon BBN Technologies.