Dimensionality Reduction for Sparse and Structured Matrices

by

Christopher Paul Musco

B.S., Yale University (2012)

Submitted to the Dept. of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

© Massachusetts Institute of Technology 2015. All rights reserved.

Signature redacted

Author ............

Dept. of Electrical Engineering and Computer Science May 20, 2015

Signature redacted

Certified by ________________________________ Martin C. Rinard

Professor of Electrical Engineering and Computer Science Thesis Supervisor

Certified by ________________________________ Jonathan A. Kelner

Professor of Applied Mathematics Thesis Supervisor

Accepted by ________________________________

Professor Leslie A. Kolodziejski

Chairman of the Committee on Graduate Students
Dimensionality Reduction for
Sparse and Structured Matrices

by

Christopher Paul Musco

Submitted to the Dept. of Electrical Engineering and Computer Science
on May 20, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Dimensionality reduction has become a critical tool for quickly solving massive matrix problems. Especially in modern data analysis and machine learning applications, an overabundance of data features or examples can make it impossible to apply standard algorithms efficiently. To address this issue, it is often possible to distill data to a much smaller set of informative features or examples, which can be used to obtain provably accurate approximate solutions to a variety of problems.

In this thesis, we focus on the important case of dimensionality reduction for sparse and structured data. In contrast to popular structure-agnostic methods like Johnson-Lindenstrauss projection and PCA, we seek data compression techniques that take advantage of structure to generate smaller or more powerful compressions. Additionally, we aim for methods that can be applied extremely quickly – typically in linear or nearly linear time in the input size.

Specifically, we introduce new randomized algorithms for structured dimensionality reduction that are based on importance sampling and sparse-recovery techniques. Our work applies directly to accelerating linear regression and graph sparsification and we discuss connections and possible extensions to low-rank approximation, $k$-means clustering, and several other ubiquitous matrix problems.

Thesis Supervisor: Martin C. Rinard
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Jonathan A. Kelner
Title: Associate Professor of Applied Mathematics
Acknowledgments

I want to start by thanking both of my research advisors, Martin Rinard and Jonathan Kelner. Most students are lucky to have one good mentor – I somehow managed to find two great ones. Martin has been nothing but supportive from my first day at MIT. His energy, optimism, and ability to push at the heart of a question are inspiring and I can only hope these qualities rub off on me over time. I can say for sure that Martin has already made me a better writer, presenter, and question asker.

At the same time, Jon is an invaluable mentor within the Theoretical Computer Science group at MIT. Every time we sit down to discuss research, he patiently introduces me to a huge range of new concepts, techniques, and problems that can keep me busy for months. I could not ask for better academic guidance. Jon has also been incredibly kind in helping me settle on my path at graduate school, from finding good problems to good collaborators.

I have also been lucky to find many unofficial mentors who have helped me immensely over the past two years. Aaron Sidford and Yin Tat Lee, thanks for taking me under your wing early on and for being fun and supportive collaborators and friends. Stelios Sidiroglou-Douskos, thanks for the constant support and confidence in me. Michael Carbin, I really appreciate your patience and guidance when I first showed up at MIT. Thanks to everyone else who has taken the time to guide me into the world of research, including Richard Peng, Michael Kapralov, Ankur Moitra, Jelani Nelson, Piotr Indyk, Brendan Juba and many more. I’ve learned so much from each of you. Thanks also to my frequent collaborator, Michael Cohen, whose unmatched energy and enthusiasm certainly makes research is the theory group easier and more enjoyable.

I was also fortunate to have many great teachers and mentors before MIT. Thanks to everyone at the Wheeler School, including Mark Harris for helping me explore any and every curiosity throughout the years. George Milkowski, thanks for inspiring a lasting interest in code and computer science. From Yale I want to thank Dana Angluin, Daniel Spielman, and Vladimir Rokhlin for helping focus my interest on
research and graduate school. Dan and Vladimir, you have both had a lasting impact on my current research interests – thanks for a great jump start.

In the end, I am most grateful for my infinitely loving and supportive family, including every wonderful cousin, aunt, uncle, and grandparent. Mom and Dad, I love you so much – I cannot imagine better parents. Thanks for teaching me to explore as a kid. Jenna, Brett, and Cameron, I’m so happy how close we’ve been throughout the years. I’m very lucky to have you all. Cam, you’ve been by my side literally since day one, making everything in life more fun, interesting, and worthwhile. Grad school is just the next of many journeys we’ve taken together and I wouldn’t have it any other way. Finally, Nicole, I owe you more thanks than I can express. From unwavering support, sacrifice, and daily joy, I can’t imagine life without you anymore. I love you very much.
3 Uniform Sampling for Matrix Approximation

3.1 Introduction ............................................................................................................. 29
  3.1.1 Our Approach ..................................................................................................... 31
  3.1.2 Road Map ........................................................................................................... 34

3.2 Leverage Score Estimation via Uniform Sampling .................................................. 35

3.3 Coherence Reducing Reweighting ......................................................................... 37

3.4 Leverage Score Estimation via Undersampling ...................................................... 40
  3.4.1 Improved Bound for Uniform Sampling ............................................................. 43

3.5 Final Algorithms ..................................................................................................... 44
  3.5.1 Descriptions and Pseudocode ........................................................................... 45
  3.5.2 Runtime Analysis ............................................................................................... 46
  3.5.3 Achieving Input-Sparsity Time ........................................................................... 49
  3.5.4 General Sampling Framework ............................................................................ 50

4 Single Pass Spectral Sparsification in Dynamic Streams ........................................ 53

4.1 Introduction ............................................................................................................. 53
  4.1.1 The Dynamic Semi-Streaming Model ................................................................. 54
  4.1.2 Streaming Sparsification ..................................................................................... 55
  4.1.3 Our Contributions ............................................................................................... 57
  4.1.4 Road Map ........................................................................................................... 59

4.2 Additional Notation and Preliminaries ................................................................... 59
  4.2.1 Graph Notation ................................................................................................... 59
  4.2.2 Spectral Sparsification ......................................................................................... 60
  4.2.3 Sparse Recovery .................................................................................................. 60

4.3 Algorithm Overview ............................................................................................... 61
  4.3.1 Effective Resistances ......................................................................................... 61
  4.3.2 Sampling in the Streaming Model ...................................................................... 62
  4.3.3 A Chain of Coarse Sparsifiers .......................................................................... 64

4.4 Recursive Sparsifier Construction ......................................................................... 64

4.5 Streaming Row Sampling ....................................................................................... 68
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6 Sparsification of Weighted Graphs</td>
<td>73</td>
</tr>
<tr>
<td>4.7 Sparsification of Structured Matrices</td>
<td>74</td>
</tr>
<tr>
<td>4.7.1 Generalized Row Sampling</td>
<td>76</td>
</tr>
<tr>
<td>4.7.2 Generalized Recursive Sparsification</td>
<td>80</td>
</tr>
<tr>
<td>4.8 Pseudorandom Number Generation</td>
<td>82</td>
</tr>
<tr>
<td>A Properties of Leverage Scores</td>
<td>85</td>
</tr>
<tr>
<td>A.1 Spectral Approximation via Leverage Score Sampling</td>
<td>85</td>
</tr>
<tr>
<td>A.2 Rank 1 Updates</td>
<td>88</td>
</tr>
<tr>
<td>A.3 Lower Semi-continuity of Leverage Scores</td>
<td>89</td>
</tr>
<tr>
<td>B Sparse Recovery Sketch</td>
<td>91</td>
</tr>
<tr>
<td>C Recursive Sparsification</td>
<td>95</td>
</tr>
</tbody>
</table>
Chapter 1

Randomized Linear Algebra

Art is the elimination of the unnecessary. – Pablo Picasso

Numerical linear algebra is one of computer science’s oldest and richest fields. For decades it has been driven by applications in engineering, the sciences, and more recently, data analysis and machine learning. To a large extent, the field’s success is due to widespread modularity: most problems can be handled with a fairly small core of fundamental numerical routines. Matrix multiplication, matrix decompositions, system solvers, and eigenvalue/singular value algorithms have been especially fundamental. Well developed software libraries give practitioners off-the-shelf access to fast, stable algorithms for these problems [BDD+02, ABB+99, GJ+10].

However, several modern applications have outpaced the ability of classical algorithms. Especially in machine learning and data analysis, matrices often encode data sets with so many features and examples that we cannot easily fit them into working memory, let alone operate on the matrices efficiently. Even for fundamental tasks like linear regression and low-rank approximation, a new approach is needed [Mah11].

Enter randomized numerical linear algebra, a relatively new field that has risen from collaborations between researchers in numerical analysis, theoretical computer science, and machine learning. By augmenting core deterministic routines with simple applications of randomness, its techniques can accelerate algorithms for many classic matrix problems without sacrificing accuracy [AMT10].
1.1 Dimensionality Reduction

Dimensionality reduction is one of the central tools of randomized linear algebra. The general paradigm is to apply randomized techniques (e.g. random sampling, random projection, hashing, etc.) to quickly reduce the dimension of a matrix before operating on it [IBM14]. Ideally, the compressed matrix is sufficient for providing a near optimal solution to the original problem, at a fraction of the computational cost.

Intuitively, when the amount of data available exceeds the dimension of the desired solution, this approach should be able to achieve very accurate results. For example, as illustrated in Figure 1-1, for least squares linear regression with many more data points than dimensions, it is possible to operate over a significantly reduced point set while still obtaining a nearly optimal solution. For a data matrix $A \in \mathbb{R}^{n \times d}$ with $n$ data points, each in $d$ dimensions, subsampling points corresponds to reducing $A$'s height.

Alternatively, as illustrated in Figure 1-2, we could imagine reducing the dimension of points in a high dimensional clustering problem by projecting to a lower dimensional subspace. This corresponds to reducing the width of our data matrix $A \in \mathbb{R}^{n \times d}$. A common approach based on the seminal work of Johnson and Lindenstrauss [JL84] is to project data points to a randomly chosen low dimensional subspace.

![Image](image.png)

(a) Original Regression Problem  
(b) Subsampled Regression Problem

Figure 1-1: For a regression problem with many more data points than free dimensions, it is possible to reduce the size of our data set while still obtaining a nearly optimal solution.
Figure 1-2: Projecting data from high dimensional clustering problems to a lower dimensional space often maintains cluster integrity.

1.1.1 Advantages and Goals

Helpful surveys on randomized numerical linear algebra and dimensionality reduction are available in [HMT11], [Mah11], and [Woo14]. We refer the reader to these papers for a wide range of recent theoretical achievements. In addition, the area has seen significant practical success [AMT10, IBM14, SKT14]. As a paradigm, dimensionality reduction for linear algebra has several general advantages:

1. It simultaneously reduces runtime cost, memory requirements, and communication cost (in a multiprocessor or distributed environment).

2. Random sampling and reduction routines are often problem independent and can be applied to a wide variety of matrix computations.

3. Known techniques tend to be naturally robust and stable.

The second point is reminiscent of the modular approach taken in classical numerical linear algebra [HMT11]. Research has greatly benefitted from decoupling
algorithms for reducing the size of $A$ and algorithms for actual analyzing the matrix (e.g. regression, clustering, low rank approximation). This second step is usually handled by time-tested deterministic algorithms. Decoupling makes it easier to reuse theoretical tools, ensure robustness, and eventually to write and deploy efficient code for huge matrices. As is standard, we often refer to a size reduced $A$ as a “sketch” and will denote the matrix by $\tilde{A}$. The typical approach to proving the accuracy of randomized matrix algorithms is to demonstrate that a dimensionality reduction technique generates a sketch that is in some way similar to $A$. For example, when reducing $A$’s height, we might prove that $\|A^T A - \tilde{A}^T \tilde{A}\|$ is small. It is then shown that this level of similarity suffices for approximately solving a problem over $A$ with $\tilde{A}$. Accordingly, algorithms for dimensionality reduction balance two goals:

1. Produce a sketch $\tilde{A}$ that is as small as possible while still useful for approximating $A$.

2. Generate $\tilde{A}$ quickly and ideally with cheap memory access patterns (e.g. one or few passes over the original data in $A$).

Since our ultimate objective is to accelerate a matrix operation by evaluating over our sketch instead of $A$, a compact $\tilde{A}$ is essential. However, if generating the sketch takes as long as the original problem or requires as much fast memory to produce quickly, then of course the technique is useless. In many cases, it is known how to generate an optimally small sketch using slow deterministic methods – randomized techniques have gained popularity because they typically offer massive computational savings at relatively minor cost in the achievable size of $\tilde{A}$.

In addition to dependence on randomness, another beneficial trend in dimensionality reduction is the use of linear transformations, often referred to as “linear sketches” in the literature. For example, projection and sampling operations can be evaluated as $\tilde{A} = PA$ or $\tilde{A} = SA$ where $P$ is a projection matrix and $S$ is a (typically weighted) sampling matrix with a single non-zero entry per row. Linearity allows for the simple and natural application of dimensionality reduction operations in parallel, streaming, and distributed settings.
In certain cases, in particular for random projection methods, linearity also allows for the computation of $\tilde{A}$ in a single memory pass over our data matrix $A$. Alternatively, we can update a sketch like $PA$ in an online fashion as we observe changes in entries in a dynamic $A$. For sampling, the situation is a bit more complicated as it is typically necessary to carefully compute different sampling probabilities for each row or column in a matrix. Thus, while sampling itself can be performed in a single pass, computing $S$ may be more intensive in terms of data access patterns and it is not so clear how to update $SA$ in an online fashion.

1.1.2 Considerations for Structured Data

In addition to the two goals mentioned, this thesis considers a more complicated picture that arises in the case of sparse or structured data. Scientific, combinatorial, and machine learning applications of matrix algorithms often involve data that is structured in some way (e.g. banded, categorical, etc.) or is highly sparse (contains many 0 values). Such structure often allows for dramatic algorithmic acceleration.

As an example, suppose $A \in \mathbb{R}^{n \times d}$ is a sparse matrix with its number of non-zero entries equal to $\text{nnz}(A)$. If $\text{nnz}(A) \ll (n \times d)$, which is the sparsity of a fully dense matrix, not only can we store $A$ in $\ll (n \times d)$ space by only maintaining pointers to non-zero entries, but we can often operate on the matrix very quickly. For example, multiplying a vector by $A$ requires just $O(\text{nnz}(A))$ time, in comparison to $O(nd)$ time for a dense matrix. Numerical analysts often take advantage of this fact by designing matrix algorithms that can solve linear systems and find eigenvalues/vectors iteratively via a succession of matrix vector products. Such algorithms (gradient descent, conjugate gradient, power method, Arnoldi iteration, etc.) often run much faster than traditional dense methods for sufficiently sparse data.

Thus, one might hope to combine the computational benefits of sparsity and structure with the benefits of dimensionality reduction by focusing on finding a sketch $\tilde{A}$ that maintains existing structure in $A$. For example, as apparent in Figure 1-3, subsampling is often an effective dimensionality reduction strategy for maintaining sparsity. On the other hand, projection to a low dimensional subspace, as suggested
Due to potentially huge computational benefits when processing $\tilde{A}$, this thesis concentrates exclusively on compression techniques designed to preserve structure and sparsity. Structure in $A$ might also allow us to accelerate the dimensionality reduction process or achieve smaller final compressions. For example, again referring to Figure 1-3, $SA$ can be stored in much less space than $PA$.

All of our methods will take this possibility into consideration, taking advantage of structure both during the dimensionality reduction process and in the subsequent application of standard matrix algorithms. We will rely on sampling methods, which in some ways are more difficult to optimize and analyze in comparison to Johnson-Lindenstrauss random projection. As mentioned, even though $S$ can be applied easily, it is not always clear how to compute the sampling matrix quickly since usually each row or column of $A$ needs to be selected with a different probability.
1.2 Linear Regression and Spectral Approximation

We focus in particular on dimensionality reduction algorithms for the ubiquitous problem of linear regression:

For a matrix $A \in \mathbb{R}^{n \times d}$ and a vector $b \in \mathbb{R}^n$, find:

$$x^* = \arg \min_{x \in \mathbb{R}^d} \|Ax - b\|_2.$$

Typically $n$, the number of data points available, is much larger than their dimension, $d$. Thus, $A$ is a tall, skinny matrix and we will aim to reduce its height. Specifically, our goal is to find sketches $\tilde{A}$ and $\tilde{b}$ with $m \ll n$ rows such that, for $\tilde{x}^* = \arg \min_{x \in \mathbb{R}^d} \|\tilde{A}x - \tilde{b}\|_2$, $\|\tilde{A}\tilde{x}^* - \tilde{b}\|_2 \approx \|Ax^* - b\|_2$. In other words, solving the regression problem over our sketch gives a solution $\tilde{x}^*$ with error nearly as small as the optimal solution $x^*$.

While this problem is obviously interesting in its own right, we are especially interested because the sort of approximation required for solving linear regression, spectral approximation, is more widely applicable. Omitting details on how to deal with $b$, for an approximation factor $\lambda \geq 1$, we say that $\tilde{A}$ is a $\lambda$-spectral approximation for $A$ if, $\forall x$:

$$\frac{1}{\lambda} \|Ax\|_2^2 \leq \|\tilde{A}x\|_2^2 \leq \|Ax\|_2^2.$$

Not only does this requirement guarantee that $\tilde{A}$ is useful for approximating a regression problem over $A$, but it is the same requirement needed for a good preconditioner for iterative linear system solvers, for solving a variety of network problems when $A$ encodes the vertex-edge incidence matrix of a graph, and for a multitude of other applications in and beyond numerical linear algebra. Techniques for finding spectral approximations are also closely connected to dimensionality reduction for low-rank approximation, clustering, and other problems, so their study has fueled significant progress in these areas as well.
1.3 Our Contributions

This thesis presents two main results on spectral approximation for structured matrices. In Chapter 3 we present work published in [CLM+15] on computing spectral approximation sketches via importance sampling alone, without any dependence on random projection or costly deterministic methods. As mentioned, sampling can preserve sparsity and generic structure.

In Chapter 4 we address a stronger type of structure, focusing on spectral approximation for matrices like graph vertex-edge incidence matrices whose rows are selected from a limited set of potential elements. This work was published in [KLM+14], which focuses on an application to spectral sparsification of graphs in the dynamic streaming model.

1.3.1 Iterative Sampling Methods (Chapter 3)

For spectral approximation, and thus linear regression, it is well known that a small, manageable set of data rows can be randomly selected to approximate a tall, skinny data matrix. However, as mentioned, it is essential that importance sampling is employed – we want to select certain important points with higher probability than others. For theoretical performance guarantees, each row must be sampled with probability proportional to its statistical leverage score, a common measure of statistical influence [DMM06a, DMM06b, SS11]. Unfortunately, leverage scores are difficult to compute. A simple alternative is to sample rows uniformly at random. While this often works in practice [MT11], uniform sampling will eliminate critical row information for many natural instances [AMT10, KMT12].

We take a fresh look at uniform sampling by examining what information it does preserve. Specifically, we show that uniform sampling yields a matrix that, in some sense, well approximates a large fraction of the original. While this weak form of approximation is not enough for solving linear regression directly, it is enough to compute a better approximation.

This observation leads to simple iterative row sampling algorithms for matrix ap-
approximation that are the first to run in input-sparsity time, $O(\text{nnz}(A))$, and preserve row structure and sparsity at all intermediate steps. Our work provides an alternative to state-of-the-art Johnson-Lindenstrauss random projection algorithms that run as fast, but do not preserve any matrix structure [CW13, MM13, NN13]. We extend work in [MP12, LMP13] that takes several fundamental steps towards a pure sampling approach, but ultimately relies on random projection.

In addition to an improved understanding of uniform sampling, our main proof introduces a structural result of independent interest: we show that every matrix can be made to have low coherence by reweighting a small subset of its rows. This general idea has already seen applications, for example in practical algorithms for matrix completion [WZZ14].

1.3.2 Spectral Approximation via Sparse Recovery

(Chapter 4)

We next show how to obtain a spectral approximation for a matrix $A \in \mathbb{R}^{n \times d}$ from a very small linear sketch, when $A$'s rows are drawn from a polynomially sized dictionary of potential elements$^{1}$. This structural restriction is natural, for example, when $A$ is the vertex-edge incidence matrix of a graph with $d$ nodes and each of its rows must be one of the $\binom{d}{2} = O(d^2)$ possible edge indicator vectors. Our sketch is of size $O(d \text{polylog}(d))$, a significant improvement over the provably required $\Omega(d^2)$ space without this structural assumption [CW09].

This project was motivated by the mentioned application to finding a spectral approximation for the vertex-edge incidence matrix of a graph. When the approximation consists of a reweighted sample of the matrix's original rows, it is typically referred to as a spectral sparsifier [ST11]. Specifically, we close a popular open question by giving the first nearly space optimal, single pass algorithm for computing spectral sparsifiers of graphs in the challenging dynamic streaming model, which allows for a data stream containing both edge insertions and deletions to the graph.

$^{1}$ Polynomial in the width of the matrix, $d$.  

19
While $O(d \text{polylog}(d))$ space algorithms are known for computing cut sparsifiers [BK96] in dynamic streams [AGM12b, GKP12] and spectral sparsifiers in insertion-only streams [KL13], prior to our work, the best known single pass algorithm for maintaining spectral sparsifiers in dynamic streams required sketches of size $\Omega(d^{5/3})$ [AGM13].

To achieve our result, we show that, using a coarse sparsifier of the graph and a linear sketch of its incidence matrix, it is possible to sample edges by leverage score, obtaining a spectral sparsifier of arbitrary precision. Sampling from the sketch requires a novel application of $\ell_2/\ell_2$ sparse recovery, a natural extension of the $\ell_0$ methods introduced by Ahn, Guha, and McGregor for cut sparsifiers [AGM12a, AGM12b]. Work in [MP12] on iterative row sampling for matrix approximation (the problem discussed in Section 1.3.1) gives a recursive approach for obtaining the required coarse sparsifiers.

### 1.3.3 Open Questions

In addition to presenting our recent theoretical results, we also hope to introduce a number of open questions and future research directions that are not discussed in the published papers related to this thesis. Whenever possible, these questions will appear under Open Question headings, so they are easily searchable.
Chapter 2

Background and Preliminaries

Before presenting the main results of this thesis, we provide additional background on dimensionality reduction for linear regression and spectral approximation, which was introduced in Section 1.2. Additionally, we present required linear algebraic notation, concepts, and tools that will be used throughout the thesis.

2.1 Dimensionality Reduction for Linear Regression

We focus on dimensionality reduction for linear regression, i.e. solving overdetermined systems, which requires a spectral approximation \( \tilde{A} \) to the data matrix \( A \in \mathbb{R}^{n \times d} \). Our final goal is to obtain a \( (1 + \epsilon) \) approximate solution for our regression problem. In other words, referring to the notation of Section 1.2, we want to find an \( x^* \) satisfying:

\[
\|A\tilde{x}^* - b\|_2 \leq (1 + \epsilon)\|Ax^* - b\|_2
\]

where \( x^* = \arg \min_{x \in \mathbb{R}^d} \|Ax - b\|_2 \).

One possibility is to obtain a \( (1 \pm \epsilon) \) spectral approximation to \( A \), which typically requires with \( \tilde{O}(d/\epsilon^2) \) rows for fast randomized methods, a potentially significant reduction from our original \( n \) rows. Solving the regression problem on \( \tilde{A} \) will immediately give a \( (1 + \epsilon) \) approximate solution. In other words, for an appropriately chose \( \tilde{b} \), we can set \( \tilde{x}^* = \arg \min_{x \in \mathbb{R}^d} \|\tilde{A}x - \tilde{b}\|_2 \).
Alternatively, to improve stability and eliminate the potentially costly $1/e^2$ dependence in the size of $\tilde{A}$, randomized schemes can be combined with known iterative regression algorithms [AMT10, CRT11, CW13, MSM14, RT08]. These methods converge after $\log(1/e)$ steps, each requiring a system solve in $\tilde{A}$, but only need a constant factor spectral approximation, which requires just $O(d)$ rows.

2.1.1 Projection Methods

As mentioned, randomized techniques for matrix dimensionality reduction divide roughly into two categories – random projection methods [PTRVO0, MRT06, Sar06, CW09, CW13, MM13, NN13] and sampling methods [DFK+04, FKV04, DKM06a, DMM06b, DMM06c, DMM08, DM10, LMP13].

Random projection methods set $\tilde{A} = \Pi A$ for some randomly generated matrix $\Pi$, which typically contains random Gaussian entries, random $\pm 1$'s, or performs a random hashing operation. If $\Pi$ is chosen such that $\Pi A$ is a spectral approximation to $A$ with high probability then it is known as a subspace embedding. Descending from the Johnson-Lindenstrauss Lemma [JL84] and related results, subspace embeddings are impressive for their simplicity and speed. $\Pi$ does not depend on our input $A$ at all. Furthermore, recent progress has significantly sped up the process of multiplying $\Pi A$, leading to the first input-sparsity time algorithms for linear regression (or nearly input-sparsity time if iterative methods are employed) [CW13, MM13, NN13].

2.1.2 Sampling Methods

However, all random projection methods necessarily recombine $A$’s rows to form $\tilde{A}$. Sampling methods, on the other hand, seek to approximate a large matrix by judiciously selecting (and reweighting) few rows. Sampling itself is even simpler and faster than random projection – the challenge becomes efficiently computing the correct measure of “importance” for rows in $A$. More important rows or columns need to be selected with higher probability.

For spectral matrix approximation, $O(d \log d/e^2)$ rows can be sampled with prob-
ability proportional to their *statistical leverage scores*. It took some time to converge on the correct sampling probabilities, with leverage scores suggested in [DMM06a] but only analyzed to work for $O(d^2)$ samples. An analysis by Spielman and Srivastava in [SS11] obtains $O(d \log d)$ dependence using a concentration result from [RV07]. This concentration result and its predecessor, [Rud99], are precursors to modern *matrix Chernoff bounds* [AW02, Oli09, Oli10, Tro12], which are typically used to prove the correctness of leverage score sampling today.

[SS11] specifically focuses on spectral approximations for the vertex-edge incidence matrix of a graph. This is more commonly referred to as *spectral sparsification* – a primitive that has been very important in literature on graph algorithms. Each row in a graph’s (potentially tall) vertex-edge incident matrix corresponds to an edge and the row’s leverage score is exactly the edge’s *weighted effective resistance*, which is used as the sampling probability in [SS11].

While leverage scores for the vertex-edge incidence matrix of a graph can be computed quickly [ST14, KMP11, KOSZ13], in general, computing leverage scores requires evaluating $(A^T A)^{-1}$, which is as difficult as solving regression in the first place. Fortunately, it is possible to sample by even rough approximations to these scores. So, for general matrices, approximation algorithms for computing leverage scores are essential.

### 2.2 Technical Preliminaries

A more technical treatment of leverage score sampling, in addition to required notation and linear algebraic tools, follows.

#### 2.2.1 Asymptotic Complexity Notation

As noted, we used $\text{nnz}(A)$ throughout to denote the number of nonzero entries in a matrix $A$. We often refer to $O(\text{nnz}(A))$ as *input-sparsity time*. We will sometimes use the notation $\tilde{O}(x)$ to denote $O(x \text{polylog}(x))$, where $\text{polylog}(x) = \log^c(x)$ for some fixed constant $c$. We will refer to $\tilde{O}(x)$ as *nearly $x$ time*. E.g. nearly linear time,
nearly input-sparsity time, etc.

2.2.2 Singular Value Decomposition and the Pseudoinverse

Any $A \in \mathbb{R}^{n \times d}$ with rank $r$ has a reduced singular value decomposition, $A = U \Sigma V^T$, where $U \in \mathbb{R}^{n \times r}$ and $V \in \mathbb{R}^{d \times r}$ have orthonormal columns and $\Sigma \in \mathbb{R}^{r \times r}$ is diagonal and contains the nonzero singular values of $A$. $A^T A = \Sigma \Sigma^T U V^T = V \Sigma \Sigma^T V^T$. Let $(A^T A)^+$ denote the Moore-Penrose pseudoinverse of $A^T A$. $(A^T A)^+ = V (\Sigma^2)^{-1} V^T$.

2.2.3 Spectral Approximation

For any $\lambda \geq 1$, we say that $\tilde{A} \in \mathbb{R}^{n' \times d}$ is a $\lambda$-spectral approximation of $A \in \mathbb{R}^{n \times d}$ if, $\forall x \in \mathbb{R}^d$

$$\frac{1}{\lambda} \|Ax\|^2 \leq \|\tilde{A}x\|^2 \leq \|Ax\|^2,$$

or equivalently

$$\frac{1}{\lambda} x^T A^T A x \leq x^T \tilde{A}^T \tilde{A} x \leq x^T A^T A x.$$

Letting $\sigma_i$ denote the $i$th singular value of a matrix, $\lambda$-spectral approximation implies:

$$\forall i, \frac{1}{\lambda} \sigma_i(A) \leq \sigma_i(\tilde{A}) \leq \sigma_i(A).$$

So, a spectral approximation preserves the magnitude of matrix-vector multiplication with $A$, the value of $A^T A$'s quadratic form, and consequently, each singular value of $A$. For conciseness, we sometimes use the Loewner ordering, writing $\frac{1}{\lambda} A^T A \preceq \tilde{A}^T \tilde{A} \preceq A^T A$ where $C \preceq D$ indicates that $D - C$ is positive semidefinite. Even more succinctly, $\tilde{A}^T \tilde{A} \approx_\lambda A^T A$ denotes the same condition.
2.2.4 Leverage Scores

The leverage score of the $i^{th}$ row $a_i^T$ of $A$ is:

$$\tau_i(A) \overset{\text{def}}{=} a_i^T(A^T A)^+ a_i.$$ (2.2)

We also define the related cross leverage score as $\tau_{ij}(A) \overset{\text{def}}{=} a_i^T(A^T A)^+ a_j$. Let $\tau(A)$ be a vector containing $A$'s $n$ leverage scores. $\tau(A)$ is the diagonal of $A(A^T A)^+ A^T$, which is a projection matrix. Therefore, $\tau_i(A) = 1_i^T A (A^T A)^+ A^T 1_i \leq 1$. In addition to this individual bound, because $A(A^T A)^+ A^T$ is a projection matrix, the sum of $A$'s leverage scores is equal to the matrix's rank,

$$\sum_{i=1}^n \tau_i(A) = \text{tr}(A(A^T A)^+ A^T) = \text{rank}(A(A^T A)^+ A^T) = \text{rank}(A) \leq d.$$ (2.3)

A row's leverage score measures how important it is in composing the row space of $A$. If a row has a component orthogonal to all other rows, its leverage score is 1. Removing it would decrease the rank of $A$, completely changing its row space. If all rows are the same, each has leverage score $d/n$. The coherence of $A$ is $\|\tau(A)\|_\infty$. If $A$ has low coherence, no particular row is especially important. If $A$ has high coherence, it contains at least one row whose removal would significantly affect the composition of $A$'s row space. A characterization that helps with this intuition follows:

**Lemma 1.** For all $A \in \mathbb{R}^{n \times d}$ and $i \in [n]$ we have that

$$\tau_i(A) = \min_{A^T x = a_i} \|x\|_2^2.$$

Let $x_i$ denote the optimal $x$ for $a_i$. The $j^{th}$ entry of $x_i$ is given by $x_i^{(j)} = \tau_{ij}(A)$.

**Proof.** For the solution $x_i$ to have minimal norm, we must have $x_i \perp \text{ker}(A^T)$. Thus, $x_i \in \text{im}(A)$ and we can write $x_i = Ac$ for some $c \in \mathbb{R}^d$. Using the constraints of the optimization problem we have that $A^T x_i = A^T Ac = a_i$. Thus $c = (A^T A)^+ a_i$, so
\[ x_i = A(A^T A)^+ a_i. \] This gives \[ x_i^{(j)} = a_j^T (A^T A)^+ a_i = \tau_{ij}(A). \] Furthermore:

\[
\|x_i\|_2^2 = a_i^T (A^T A)^+ A^T A (A^T A)^+ a_i \\
= a_i^T (A^T A)^+ a_i = \tau_i(A).
\]

We often approximate the leverage scores of \( A \) by computing them with respect to some other matrix \( B \in \mathbb{R}^{n' \times d} \). We define the generalized leverage score:

\[
\tau_i^B(A) \equiv \begin{cases} 
    a_i^T (B^T B)^+ a_i & \text{if } a_i \perp \ker(B), \\
    \infty & \text{otherwise.}
\end{cases}
\] (2.4)

If \( a_i \) has a component in \( \ker(B) \), we set its generalized leverage score to \( \infty \), since it might be the only row in \( A \) pointing in this direction. Thus, when sampling rows, we cannot remove it. We could set the generalized leverage score to 1, but using \( \infty \) simplifies notation in some of our proofs. If \( B \) is a spectral approximation for \( A \), then every generalized leverage score is a good multiplicative approximation to its corresponding true leverage score:

**Lemma 2** (Leverage Score Approximation via Spectral Approximation - Lemma 4.3 in [LMP13]). If \( B \) is a \( \lambda \)-spectral approximation of \( A \), so \( \frac{1}{\lambda} A^T A \preceq B^T B \preceq A^T A \), then \( \tau_i(A) \leq \tau_i^B(A) \leq \lambda \cdot \tau_i(A) \).

**Proof.** This follows simply from the definition of leverage scores and generalized leverage scores and the fact that \( \frac{1}{\lambda} A^T A \preceq B^T B \preceq A^T A \) implies \( \lambda (A^T A)^+ \succeq (B^T B)^+ \succeq A^T A^+ \). \( \square \)

### 2.2.5 Leverage Score Sampling

Sampling rows from \( A \) according to their exact leverage scores gives a spectral approximation for \( A \) with high probability. Sampling by leverage score overestimates also suffices. Formally:
Lemma 3 (Spectral Approximation via Leverage Score Sampling). Given an error parameter $0 < \epsilon < 1$, let $u$ be a vector of leverage score overestimates, i.e., $\tau_i(A) \leq u_i$ for all $i$. Let $\alpha$ be a sampling rate parameter and let $c$ be a fixed positive constant. For each row, we define a sampling probability $p_i = \min \{ 1, \alpha \cdot u_i c \log d \}$. Furthermore, let $\text{Sample}(u, \alpha)$ denote a function which returns a random diagonal matrix $S$ with independently chosen entries. $S_{ii} = \frac{1}{\sqrt{p_i}}$ with probability $p_i$ and 0 otherwise.

If we set $\alpha = \epsilon^{-2}$, $S = \text{Sample}(u, \epsilon^{-2})$ has at most $\sum \min \{ 1, \alpha \cdot u_i c \log d \} \leq \alpha c \log d \|u\|_1$ non-zero entries and $\frac{1}{\sqrt{1+\epsilon}}SA$ is a $\frac{1}{1-\epsilon}$-spectral approximation for $A$ with probability at least $1 - d^{-c/3}$.

For completeness, we prove Lemma 3 in Appendix A using a matrix concentration result of [Tro12].
Chapter 3

Uniform Sampling for Matrix Approximation

This chapter presents work published in [CLM+15], and is thus a collaborative effort between Michael B. Cohen, Yin Tat Lee, Cameron Musco, Richard Peng, Aaron Sidford, and myself.

3.1 Introduction

Fast randomized algorithms for solving massive regression problems rely on the fundamental building block of spectral approximation. However, as discussed, all of the fastest algorithms for this dimensionality reduction guarantee rely on variations of Johnson-Lindenstrauss random projection [CW13, MM13, NN13, LMP13].

By re-examining uniform sampling, a heuristic that works for low coherence data, we give spectral approximation algorithms that avoid random projection entirely. Our methods are the first to match state-of-the-art runtimes (i.e. $O(\text{nnz } A)$ time) while preserving row structure and sparsity in all matrix operations.

As discussed in Section 2.1.2, it is known that for a data matrix $A \in \mathbb{R}^{n \times d}$, a spectral approximation can be obtained by sampling $O(d \log d)$ rows, each with probability proportional to its statistical leverage score [DMM06a, DMM06b, SS11].

Unfortunately, leverage scores are difficult to calculate – finding them involves
computing the pseudoinverse $(A^T A)^+$, which is as costly as solving our regression problem in the first place! In practice, data is often assumed to have low coherence [MT11], in which case simply selecting rows uniformly at random works [AMT10, KMT12]. However, uniform sampling could be disastrous – if $A$ contains a row with some component orthogonal to all other rows, removing it will reduce the rank of $A$ and thus we cannot possibly preserve all vector products ($\|\tilde{A}x\|_2$ will start sending some vectors to 0). Any uniform sampling scheme is likely to drop any such single row.\footnote{On the other hand, when leverage score sampling, such a row would have the highest possible leverage score of 1.}

Possible fixes include randomly “mixing” data points to avoid degeneracies [AMT10]. However, this approach sacrifices sparsity and structure in our data matrix, increasing storage and runtime costs. Is there a more elegant fix? First note that sampling $A$ by approximate leverage scores is fine, but we may need to select more than the optimal $O(d \log d)$ rows. With that in mind, consider the following straightforward algorithm for iterative sampling (inspired by [LMP13]):

\textbf{Step 1} Reduce $A$ significantly by sampling uniformly.

\textbf{Step 2} Approximate $(A^T A)^+$ using the smaller matrix and estimate leverage scores for $A$.

\textbf{Step 3} Resample rows from $A$ using these estimates, obtaining an accurate but potentially large spectral approximation $\tilde{A}$.

\textbf{Step 4} Repeat from Step 1 to reduce $\tilde{A}$ further and obtain a smaller approximation.

While intuitive, this scheme was not previously known to work! Our main technical result in this chapter is proving that it does. This process (and related schemes) will quickly converge on a small spectral approximation to $A$ – i.e. with $O(d \log d)$ rows.

A few results come close to an analysis of such a routine – in particular, two iterative sampling schemes are analyzed in [LMP13] and [MP12] (an earlier version of
the paper). However, the first ultimately requires Johnson-Lindenstrauss projections that mix rows, something we want to avoid. The second almost maintains sparsity and row structure (except for possibly including rows of the identity in $\tilde{A}$), but its convergence rate depends on the condition number of $A$.

More importantly, both of these results are similar in that they rely on the primitive that a (possibly poor) spectral approximation to $A$ is sufficient for approximately computing leverage scores, which are in turn good enough for obtaining an even better spectral approximation. As mentioned, uniform sampling will not in general give a spectral approximation – it does not preserve information about all singular values. Our key contribution is a better understanding of what information uniform sampling does preserve. It turns out that, although weaker than a spectral approximation, the matrix obtained from uniform sampling can nonetheless give leverage score estimates that are good enough to obtain increasingly better approximations to $A$.

### 3.1.1 Our Approach

Suppose we compute a set of leverage score estimates, $\{\tilde{r}_i\}$, using $(\tilde{A}^T\tilde{A})^+$ in place of $(A^TA)^+$ for some already obtained matrix approximation $\tilde{A}$. As long as our leverage score approximations are upper bounds on the true scores ($\tilde{r}_i \geq r_i$) we can use them for sampling and still obtain a spectral approximation to $A$ (see Lemma 3). The number of samples we take will be

$$c \cdot \log d \cdot \sum_{i=1}^{n} \tilde{r}_i,$$

where $c$ is some fixed constant. When sampling by exact leverage scores, $\sum_{i=1}^{n} r_i \leq d$ (2.3) so we take $c \cdot d \log d$ rows.

Thus, to prove that our proposed iterative algorithm works, we need to show that, if we uniformly sample a relatively small number of rows from $A$ (Step 1) and estimate leverage scores using these rows (Step 2), then the sum of our estimates will be small. Then, when we sample by these estimated leverage scores in Step 3, we can sufficiently reduce the size of $A$. Note that we will not aim to reduce $A$ to
$O(d \log d)$ height in one shot – we just need our leverage estimates to sum to say, $n/(2c \log d)$, which allows us to cut the large matrix in half at each step.

In prior work, the sum of overestimates was bounded by estimating each leverage score to within a multiplicative factor. This requires a spectral approximation, which is why previous iterative sampling schemes could only boost poor spectral approximations to better spectral approximations. Of course, a “for each” statement is not required, and we will not get one through uniform sampling. Thus, our core result avoids this technique. Specifically, we show,

**Theorem 4 (Leverage Score Approximation via Uniform Sampling).** For any $m$, we can select $O(m)$ rows uniformly at random from $A$ to obtain $\tilde{A}$. Then, letting $\{\tilde{\tau}_i\}$ be a set of leverage score estimates for $A$ computed using $\tilde{A}^2$, both of the following hold:

$$\forall i, \tilde{\tau}_i \geq \tau_i,$$

$$\mathbb{E} \left[ \sum_{i=1}^{n} \tilde{\tau}_i \right] \leq \frac{nd}{m}.$$ 

The validity of our proposed iterative sampling scheme immediately follows from Theorem 4. For example, if we set $m = O(d \log d)$ with a high enough constant, $c \log d \sum \tilde{\tau}_i \leq \frac{n}{2}$, allowing us to cut our matrix in half. Alternatively, if we uniformly sample $m = O(n)$ rows (say $n/2$) then $c \log d \sum \tilde{\tau}_i \leq O(d \log d)$, so we can cut our matrix down to $O(d \log d)$ rows. There is a convenient tradeoff – the more rows uniformly sampled in Step 1, the more we can cut $A$ down by in Step 3. This tradeoff leads to natural recursive and iterative algorithms for row sampling.

**Open Question 1 (Uniform Sampling for Low Rank Approximation).** While our current work focuses on spectral approximation and linear regression, we would like to know if similar iterative sampling algorithms can be adapted to approximate singular value decomposition/low rank approximation.

---

2 We describe exactly how each $\tilde{\tau}_i$ is computed when we prove Theorem 4 in Section 3.2.
In [CEM+15] we show how construct a sketch via a version of rank-restricted leverage score sampling from which its possible to extract a $(1 + \epsilon)$ error low rank approximation (with respect to Frobenius norm). The required sampling probabilities need to be computed from a low rank approximation to $A$, leading to the same "chicken and egg" problem encountered for standard leverage score computation.

However, as for standard leverage scores and spectral approximation, the procedure from [CEM+15] is robust to over sampling and it is thus possible to use approximate sampling estimates to the rank-restricted leverage scores? The goal would be an approximate singular value decomposition that does not rely on random projection methods.

We give a proof of Theorem 4 using a simple expectation argument that bounds $E \tilde{\tau}_i$ for all $i$. We also prove alternative versions of Theorem 4 with slightly different guarantees (Theorems 9 and 10) using a technique that we believe is of independent interest. It is well known that, if $A$ has low coherence – that is, has a low maximum leverage score – then uniform sampling from the matrix is actually sufficient for obtaining a full spectral approximation. The uniform rate will upper bound the leverage score rate for every row. With this in mind, we show a powerful fact: while many matrices do not have low coherence, for any $A$, we can decrease the weight on a small subset of rows to make the matrix have low coherence. Specifically,

**Lemma 5** (Coherence Reducing Reweighting). For any $n \times d$ matrix $A$ and any coherence upper bound $\alpha > 0$ there exists a diagonal reweighting matrix $W \in \mathbb{R}^{n \times n}$ with all entries in $[0, 1]$ and just $(d/\alpha)$ entries not equal to 1, such that:

$$\forall i, \tau_i(WA) \leq \alpha.$$

Intuitively, this lemma shows that uniform sampling gives a matrix that spectrally approximates a large sub-matrix of the original data. It follows from our more general Theorem 6, which describes exactly how leverage scores of $A$ can be manipulated through row reweighting.

Note that we never actually find $W$ explicitly – simply its existence implies our
uniform sampling theorems! As explained, since \( \mathbf{WA} \) has low coherence, uniform sampling would give a spectral approximation and thus a multiplicatively good approximation to each leverage score. Thus, the sum of estimated leverage scores for \( \mathbf{WA} \) will be low, i.e. \( < O(d) \). It can be shown that, for any row that is not reweighted, the leverage score in \( \mathbf{A} \) computed using a uniformly sampled \( \hat{\mathbf{A}} \), is never greater than the corresponding leverage score in \( \mathbf{WA} \) computed using a uniformly sampled \( \hat{\mathbf{WA}} \). Thus, the sum of approximate leverage scores for rows in \( \mathbf{A} \) that are not reweighted is small by comparison to their corresponding leverage scores in \( \mathbf{WA} \). How about the rows that are reweighted in \( \mathbf{WA} \)? Lemma 5 claims there are not too many of these – we can trivially bound their leverage score estimates by 1 and even then the total sum of estimated leverage scores will be small.

This argument gives the result we need: even if a uniformly sampled \( \hat{\mathbf{A}} \) cannot be used to obtain good per row leverage score upper bounds, it is sufficient for ensuring that the sum of all leverage score estimates is not too high. Varying \( \alpha \) and setting \( m = \alpha \log d/n \) leads to a range of iterative schemes, as described under Theorem 4.

### 3.1.2 Road Map

The remained of Chapter 3 is organized as follows:

**Section 3.2** Prove that uniform sampling is sufficient for leverage score estimation (Theorem 4).

**Section 3.3** Show the existence of small, coherence-reducing reweightings (Theorem 6, which generalizes Lemma 5).

**Section 3.4** Use Theorem 6 to prove alternative versions of Theorem 4 (Theorems 9 and 10).

**Section 3.5** Describe simple and efficient iterative algorithms for spectral matrix approximation that run in input-sparsity or nearly input-sparsity time.
3.2 Leverage Score Estimation via Uniform Sampling

In this section, we prove Theorem 4 using a simple expectation argument. We restate a more complete version of the theorem below:

**Theorem 4 (Full Statement).** Given any $A \in \mathbb{R}^{n \times d}$. Let $S$ denote a uniformly random sample of $m$ rows from $A$ and let $S \in \mathbb{R}^{m \times n}$ be its diagonal indicator matrix (i.e. $S_{ii} = 1$ for $i \in S$, $S_{ii} = 0$ otherwise). Define

$$
\tilde{\tau}_i \overset{\text{def}}{=} \begin{cases} 
\tau_i^{SA}(A) & \text{if } i \in S, \\
\frac{1}{1 + \tau_i^{SA}(A)} & \text{otherwise.}
\end{cases}
$$

Then, $\tilde{\tau}_i \geq \tau_i(A)$ for all $i$ and

$$
\mathbb{E} \left[ \sum_{i=1}^{n} \tilde{\tau}_i \right] \leq \frac{nd}{m}.
$$

**Proof.** First we show that our estimates are valid leverage score upper bounds, i.e. $\tilde{\tau}_i \geq \tau_i(A)$. Let $S(i)$ be the diagonal indicator matrix for $S \cup \{i\}$. We claim that, for all $i$,

$$
\tilde{\tau}_i = \tau_i^{S(i)A}(A). \tag{3.1}
$$

This is proved case-by-case:

1. When $i \in S$, $S = S(i)$ so it holds trivially.

2. When $i \notin S$ and $a_i \not\in \ker(SA)$, then by definition, $\tau_i^{SA}(A) = \infty$ and $\tilde{\tau}_i = \frac{1}{1 + \infty} = 1 = \tau_i^{S(i)A}(A)$.

3. When $i \notin S$ and $a_i \perp \ker(SA)$ then by the Sherman-Morrison formula for
pseudoinverses [Mey73, Thm 3],

\[
\tau_i^{S(i)A}(A) = a_i^T \left( (A^T S^2 A + a_i a_i^T)^+ \right) a_i \\
= a_i^T \left( (A^T S^2 A)^+ \frac{(A^T S^2 A)^+ a_i (A^T S^2 A)^+}{1 + a_i^T (A^T S^2 A)^+ a_i} \right) a_i \\
= \tau_i^{SA}(A) - \frac{\tau_i^{SA}(A)^2}{1 + \tau_i^{SA}(A)} = \frac{1}{1 + \frac{1}{\tau_i^{SA}(A)}} = \tilde{\tau}_i. 
\]

(Sherman-Morrison formula)

By (3.1) and the fact that \( A^T S(i)^2 A \preceq A^T A \) (see Lemma 2), we have \( \tilde{\tau}_i = \tau_i^{S(i)A}(A) \geq \tau_i(A) \), so our estimates are upper bounds as desired. It remains to upper bound the expected sum of \( \tilde{\tau}_i \). We can break down the sum as

\[
\sum_{i=1}^{n} \tilde{\tau}_i = \sum_{i \in S} \tilde{\tau}_i + \sum_{i \notin S} \tilde{\tau}_i.
\]

The first term is simply the sum of \(SA\)'s leverage scores, so it is equal to \(\text{rank}(SA) \leq d\) by (2.3). To bound the second term, consider a random process that first selects \(S\), then selects a random row \(i \notin S\) and returns \(\tilde{\tau}_i\). There are always exactly \(n - m\) rows \(\not\in S\), so the value returned by this random process is, in expectation, exactly equal to \(\frac{1}{n-m} \cdot E \sum_{i \notin S} \tilde{\tau}_i\).

This random process is also equivalent to randomly selecting a set \(S'\) of \(m + 1\) rows, then randomly choosing a row \(i \in S'A\) and returning its leverage score! In expectation it is therefore equal to the average leverage score in \(S'A\). \(S'A\) has \(m + 1\) rows and its leverage scores sum to its rank, so we can bound its average leverage score by \(\frac{d}{m+1}\). Overall we have:

\[
E \left[ \sum_{i=1}^{n} \tilde{\tau}_i \right] \leq d + (n - m) \cdot \frac{d}{m + 1} \leq \frac{d(n + 1)}{m + 1} \leq \frac{nd}{m}.
\]
3.3 Coherence Reducing Reweighting

In this section, we prove that we can reweight a small number of rows in any matrix $A$ to make it have low coherence. This structural result may be of independent interest. It is also fundamental in proving Theorem 9, a slightly stronger and more general version of Theorem 4 that we will prove in Section 3.4.

Actually, in Theorem 6 we prove a more general statement, studying how to select a diagonal row reweighting matrix $W$ to arbitrarily control the leverage scores of $WA$. One simple conjecture would be that, given a vector $u$, there always exists a $W$ such that $\tau_i(WA) = u_i$. This conjecture is unfortunately not true – if $A$ is the identity matrix, then $\tau_i(WA) = 0$ if $W_{ii} = 0$ and $\tau_i(WA) = 1$ otherwise. Instead, we show the following:

**Theorem 6 (Leverage Score Bounding Row Reweighting).** For any $A \in \mathbb{R}^{n \times d}$ and any vector $u \in \mathbb{R}^n$ with $u_i > 0$ for all $i$, there exists a diagonal matrix $W \in \mathbb{R}^{n \times n}$ with $0 \preceq W \preceq I$ such that:

$$\forall i, \tau_i(WA) \leq u_i, \quad (3.2)$$

and

$$\sum_{i:W_{ii} \neq 1} u_i \leq d. \quad (3.3)$$

Note that (3.2) is easy to satisfy – it holds if we set $W = 0$. Hence, the main result is the second claim. Not only does a $W$ exist that gives the desired leverage score bounds, but the set of rows we need to reweight in $A$ has low total weight in terms of $u$.

For any incoherence parameter $\alpha$, if we set $u_i = \alpha$ for all $i$, then this theorem shows the existence of a reweighting that reduces coherence to $\alpha$. Such a reweighting has $\sum_{i:W_{ii} \neq 1} \alpha \leq d$ and therefore $|\{i: W_{ii} \neq 1\}| \leq \frac{d}{\alpha}$. So, we see that Lemma 5 follows as a special case of Theorem 6.

In order to prove Theorem 6, we first give two technical lemmas which are proved
Lemma 7 (Leverage Score Changes Under Rank 1 Updates). Given any $A \in \mathbb{R}^{n \times d}$, $\gamma \in (0, 1)$, and $i \in [n]$, let $W$ be a diagonal matrix such that $W_{ii} = \sqrt{1 - \gamma}$ and $W_{jj} = 1$ for all $j \neq i$. Then,

$$
\tau_i(WA) = \frac{(1 - \gamma) \tau_i(A)}{1 - \gamma \tau_i(A)} \leq \tau_i(A),
$$

and for all $j \neq i$,

$$
\tau_j(WA) = \tau_j(A) + \frac{\gamma \tau_{ij}(A)}{1 - \gamma \tau_i(A)} \geq \tau_j(A).
$$

Next we claim that leverage scores are lower semi-continuous in the weighting of the rows. This allows us to reason about weights that arise as the limit of Algorithm 1 for computing them.

Lemma 8 (Leverage Scores are Lower Semi-continuous). $\tau(WA)$ is lower semi-continuous in the diagonal matrix $W$, i.e. for any sequence $W^{(k)} \to \bar{W}$ with $W_{ii}^{(k)} \geq 0$ for all $k$ and $i$, we have

$$
\tau_i(\bar{W}A) \leq \liminf_{k \to \infty} \tau_i(W^{(k)}A).
$$

With Lemmas 7 and 8 in place, we are ready to prove the main reweighting theorem.

Proof of Theorem 6. We prove the existence of the required $W$ by considering the limit of the following algorithm for computing a reweighting matrix.
**Algorithm 1 COMPUTE REWEIGHTING (a.k.a the whack-a-mole algorithm)**

Initialize $W = I$

while true do

for $i = 1$ to $n$ do

if $\tau_i(WA) \geq u_i$ then

if $\tau_i(WA) < 1$ then

Decrease $W_{ii}$ so that $\tau_i(WA) = u_i$.

else

Set $W_{ii} = 0$

end if

end if

end for

end while

return $W$

For all $k \geq 0$, let $W^{(k)}$ be the value of $W$ after the $k^{th}$ update to the weight. We show that $\overline{W} = \lim_{k \to \infty} W^{(k)}$ meets the conditions of Theorem 6. First note that Algorithm 1 is well defined and that all entries of $W^{(k)}$ are non-negative for all $k \geq 0$. To see this, suppose we need to decrease $W_{ii}^{(k)}$ so that $\tau_i(W^{(k+1)}A) = u_i$. Note that the condition $\tau_i(W^{(k)}A) < 1$ gives

$$\lim_{\gamma \to 1} \frac{(1 - \gamma) \tau_i(W^{(k)}A)}{1 - \gamma \tau_i(W^{(k)}A)} = 0.$$ 

Therefore, Lemma 7 shows that we can make $\tau_i(W^{(k+1)}A)$ arbitrary small by setting $\gamma$ close enough to 1. Since the leverage score for row $i$ is continuous, this implies that $W^{(k+1)}$ exists as desired.

Since, the entries of $W^{(k)}$ are non-negative and decrease monotonically by construction, clearly $\overline{W}$ exists. Furthermore, since setting $W_{ii} = 0$ makes $\tau_i(WA) = 0$, we see that, by construction,

$$\lim_{k \to \infty} \inf \tau_i(W^{(k)}A) \leq u_i \text{ for all } i \in [n].$$
Therefore, by Lemma 8 we have that \(\tau_i(\overline{WA}) \leq u_i\).

It only remains to show that \(\sum_{i:W_{ii} \neq 1} u_i \leq d\). Let \(k\) be the first iteration such that \(W_i^{(k)} \neq 1\) for any \(i\) such that \(\overline{W}_{ii} \neq 1\). Let \(S \subseteq [n]\) be the set of rows such that \(W_i^{(k)} = 0\) and let \(T = \{i : \overline{W}_{ii} \neq 1\} - S\). Since decreasing the weight of one row increases the leverage scores of all other rows, we have

\[
\sum_{i \in T \cup S} u_i \leq \sum_{i \in T} \tau_i (W^{(k)}A) + \sum_{i \in S} 1 \\
\leq \text{rank} (W^{(k)}A) + |S|.
\]

When we set \(W_{ii} = 0\), it must be the case that \(\tau_i(WA) = 1\). In this case, removing the \(i^{th}\) row decreases the rank of \(WA\) by 1 and hence \(\text{rank}(W^{(k)}A) \leq d - |S|\). Therefore,

\[
\sum_{i:W_{ii} \neq 1} u_i = \sum_{i \in T \cup S} u_i \leq d.
\]

\[
\square
\]

### 3.4 Leverage Score Estimation via Undersampling

Theorem 4 alone is enough to prove that a variety of iterative methods for spectral matrix approximation work. However, in this section we prove Theorem 9, a slight strengthening and generalization that improves runtime bounds, proves correctness for some alternative sampling schemes, and gives some more intuition for why uniform sampling allows us to obtain leverage score estimates with low total sum.

Theorem 9 relies on Theorem 6, which intuitively shows that a large fraction of our matrix \(A\) has low coherence. Sampling rows uniformly will give a spectral approximation for this portion of our matrix. Then, since few rows are reweighted in \(WA\), even loose upper bounds on the leverage scores for those rows will allow us to bound the total sum of estimated leverage scores when we sample uniformly.

Formally, we show an upper bound on the sum of estimated leverage scores obtained from undersampling \(A\) according to any set of leverage score upper bounds.
Uniform sampling can be viewed as undersampling $A$ when all we know is that each leverage score is upper bounded by 1. That is, in the uniform case, we set $u = 1$.

The bound in Theorem 9 holds with high probability, rather than in expectation like the bound in Theorem 4. This gain comes at a cost of requiring our sampling rate to be higher by a factor of $\log d$. At the end of this section we show how the $\log d$ factor can be removed at least in the case of uniform sampling, giving a high probability statement that matches the bound of Theorem 4.

**Theorem 9 (Leverage Score Approximation via Undersampling).** Let $u$ be a vector of leverage score overestimates, i.e., $\tau_i(A) \leq u_i$ for all $i$. For some undersampling parameter $\alpha \in (0, 1]$, let $S' = \sqrt{\frac{1}{4}} \cdot \text{Sample}(u, 9\alpha)$. Let $u^{(new)}_i = \min\{\tau_i^{SA}(A), u_i\}$. Then, with high probability, $u^{(new)}$ is a vector of leverage score overestimates, i.e. $\tau_i(A) \leq u^{(new)}_i$, and

$$\sum_{i=1}^{n} u^{(new)}_i \leq \frac{3d}{\alpha}.$$ 

Furthermore, $S'$ has $O(\alpha \|u\|_1 \log d)$ nonzeros.

**Proof.** Let $S = \sqrt{\frac{3}{4}} \cdot \text{Sample}(u, 9)$. Since $u$ is a set of leverage score overestimates, Lemma 3 (with $\epsilon = 1/3$) shows that, with high probability,

$$A^T S^2 A \preceq A^T A.$$ 

In Sample, when we include a row, we reweight it by $1/\sqrt{p_i}$. For $S' = \sqrt{\alpha} \sqrt{\frac{3}{4}} \cdot \text{Sample}(u, 9\alpha)$, we sample at a rate lower by a factor of $\alpha$ as compared with $S$, so we weight rows by a factor of $\frac{1}{\sqrt{\alpha}}$ higher. The $\sqrt{\alpha}$ multiplied by $S'$ makes up for this difference. Thus, $S'$ is equivalent to $S$ with some rows removed. Therefore:

$$A^T S'^2 A \preceq A^T S^2 A \preceq A^T A.$$ 

So, for all $i$, $\tau_i(A) \leq \tau_i^{S' A}(A)$. By assumption $\tau_i(A) \leq u_i$, so this proves that $\tau_i(A) \leq u^{(new)}_i$.

By Theorem 6, there is a diagonal matrix $W$ such that $\tau_i(WA) \leq \alpha u_i$ for all $i$.
and $\sum_{i:W_i \neq 1} \alpha u_i \leq d$. For this $W$, using the fact that $u_i^{(new)} = \min\{\tau_i^{S^i A}(A), u_i\}$, we have

$$
\sum_{i=1}^{n} u_i^{(new)} \leq \sum_{i:W_i \neq 1} u_i + \sum_{i:W_{ii}=1} \tau_i^{S^i A}(A) \\
\leq \frac{d}{\alpha} + \sum_{i:W_{ii}=1} \tau_i^{S^i A}(A) \\
= \frac{d}{\alpha} + \sum_{i:W_{ii}=1} \tau_i^{S^i A}(WA). 
$$

(3.4)

Using $W \preceq I$, we have

$$
\tau_i^{S^i A}(WA) \leq \tau_i^{S^i WA}(WA). 
$$

(3.5)

Now, note that $S' = \sqrt{\alpha} \sqrt{\frac{3}{4}} \cdot \text{Sample}(u, 9\alpha) = \sqrt{\alpha} \sqrt{\frac{3}{4}} \cdot \text{Sample}(\alpha u, 9)$. Since $\alpha u$ is an overestimate of leverage scores for $WA$, Lemma 3 (again with $\epsilon = 1/3$) shows that $\alpha \cdot \frac{1}{2} A^T W^2 A \preceq A^T WS^2 W A$. Hence (3.5) along with Lemma 2 shows that

$$
\tau_i^{S^i A}(WA) \leq \frac{2}{\alpha} \tau_i(WA).
$$

Combining with (3.4), we have

$$
\sum_{i=1}^{n} u_i^{(new)} \leq \frac{d}{\alpha} + \frac{2}{\alpha} \sum_{i:W_{ii}=1} \tau_i(WA) \\
\leq \frac{d}{\alpha} + \frac{2d}{\alpha} \leq \frac{3d}{\alpha}.
$$

Choosing an undersampling rate $\alpha$ is equivalent to choosing a desired sampling rate and setting $\alpha$ accordingly. From this perspective, it is clear that the above theorem gives an extremely simple way to iteratively improve leverage scores. Start with $u^{(1)}$ with $\|u^{(1)}\|_1 = s_1$. Undersample at rate $\frac{6d}{s_1}$ to obtain a sample of size $O(d \log d)$, which gives new leverage score estimates $u^{(2)}$ with $\|u^{(2)}\|_1 = \frac{3d}{6d/s_1} = \frac{s_1}{2}$. Repeat this process, cutting the sum of leverage score estimates in half with each iteration. Recall that we restrict $\alpha < 1$, so once the sum of leverage score estimates
converges on $O(d)$, this halving process halts – as expected, we cannot keep cutting
the sum further.

### 3.4.1 Improved Bound for Uniform Sampling

The algorithm just described corresponds to Algorithm 3 in Section 3.5 and differs
somewhat from approaches discussed earlier (e.g. our proposed iterative algorithm
from Section 3.1). It always maintains a sample of just $O(d \log d)$ rows that is im-
proved iteratively.

Consider instead sampling few rows from $A$ with the goal of estimating leverage
scores well enough to obtain a spectral approximation with $n/2$ rows. In the uni-
form sampling case, when $u = 1$, if we set $\alpha = \frac{d \log d}{6n}$ for example, then sampling
$O(\alpha \|u\|_1 \log d) = O(d \log^2 d)$ rows uniformly will give us leverage score estimates
summing to $\frac{n}{2 \log d}$. This is good enough to cut our original matrix in half. However,
we see that we have lost a $\log d$ factor to Theorem 4, which let us cut down to expected
size $\frac{n}{2}$ by sampling just $O(d \log d)$ rows uniformly.

At least when $u = 1$, this $\log d$ factor can be eliminated. In Theorem 9, we set
$S' = \sqrt{\alpha} \sqrt{\frac{3}{4}} \cdot \text{Sample}(u, 9\alpha)$, meaning that rows selected for $S'$ are included with
weight $\sqrt{\alpha} \sqrt{\frac{3}{4}} \cdot \frac{1}{\sqrt{n}} = \sqrt{\frac{3\alpha}{4 \min(1, 9\alpha \log d)}}$. Instead of reweighting rows, consider simply
setting all non-zero values in $S'$ to be 1. We know that our leverage score estimates
will still be overestimates as we still have $S' \preceq I$ and so $A^T S'^2 A \preceq A^T A$. Formally,
consider two cases:

1. $(1 \leq 9\alpha \log d)$. In this case, $S'A$ is simply $A$ itself, so we know our leverage
score estimates are exact and thus their sum is $\leq d$. We can use them to obtain
a spectral approximation with $O(d \log d)$ rows.

2. $(1 > 9\alpha \log d)$. In this case, we reweight rows by $\sqrt{\alpha} \sqrt{\frac{3}{4}} \cdot \frac{1}{\sqrt{n}} = \sqrt{\frac{3\alpha}{4 \cdot 49\alpha \log d}} = \sqrt{\frac{3}{4 \cdot 9\alpha \log d}}$. Thus, increasing weights in $S'$ to 1 will reduce leverage score esti-
mates by a factor of $\frac{3}{4d \log d}$. So overall we have:

$$\sum_{i=1}^{n} u_i^{(new)} \leq \sum_{i: W_{ii} \neq 1} u_i + \sum_{i: W_{ii} = 1} r_{i}^{*A}(A)$$

$$\leq |\{i : W_{ii} \neq 1\}| + \frac{3}{4 \log d} \cdot \frac{d}{\alpha}.$$ 

Recall from Lemma 3 that sampling by $u^{(new)}$ actually gives a matrix with $\sum_{i} \min\{1, u_i^{(new)} \cdot e^{-2c \log d}\}$ rows. Thus, we obtain a $\frac{1+d}{1-e}$-spectral approximation to $A$ with the following number of rows:

$$|\{i : W_{ii} \neq 1\}| + e^{-2c \log d} \cdot \frac{3}{4 \log d} \cdot \frac{d}{\alpha} + \frac{d}{\alpha} + \frac{e^{-2d}}{6}.$$ 

Setting $\alpha = \frac{m}{n \cdot 9 \log d}$ for some $m \leq n$ so that Sample$(1, 9\alpha)$ samples rows at rate $m/n$ yields the following theorem:

**Theorem 10.** Given $A \in \mathbb{R}^{n \times d}$, suppose we sample rows uniformly and independently at rate $\frac{m}{n}$, without reweighting, to obtain $SA$. Computing $\tilde{r}_i = \min\{1, r_i^{SA}(A)\}$ for each row and resampling from $A$ by these estimates will, with high probability, return a $\frac{1+\epsilon}{1-\epsilon}$-spectral approximation to $A$ with at most $O(\frac{nd \log d e^{-2}}{m})$ rows.

Choosing $m = O(d \log d)$ allows us to find a spectral approximation of size $\frac{d}{2}$, as long as $O(d \log d) < n$. This matches the bound of Theorem 4, but holds with high probability.

### 3.5 Final Algorithms

As discussed in the introduction, Theorems 4, 9, and 10 immediately yield new, extremely simple algorithms for spectral matrix approximation. For clarity, we initially present versions running in *nearly* input-sparsity time. However, we later explain how our first algorithm can be modified with standard techniques to remove log factors, achieving input-sparsity time and thus matching state-of-the-art results [CW13, MM13, NN13]. Our algorithms rely solely on row sampling, which preserves
matrix sparsity and structure, possibly improving space usage and runtime for intermediate system solves.

3.5.1 Descriptions and Pseudocode

The first algorithm we present, REPEATED HALVING, is a simple recursive procedure. We uniformly sample $\frac{n}{2}$ rows from $A$ to obtain $A'$. By Theorems 4 and 10, estimating leverage scores of $A$ with respect to this sample allows us to immediately find a spectral approximation to $A$ with $O(d \log d)$ rows. Of course, $A'$ is still large, so computing these estimates would be slow. Thus, we recursively find a spectral approximation of $A'$ and use this to compute the estimated leverage scores.

Algorithm 2 REPEATED HALVING

input: $n \times d$ matrix $A$

output: spectral approximation $\tilde{A}$ consisting of $O(d \log d)$ rescaled rows of $A$

1: Uniformly sample $\frac{n}{2}$ rows of $A$ to form $A'$
2: If $A'$ has $> O(d \log d)$ rows, recursively compute a spectral approximation $\tilde{A}'$ of $A'$
3: Compute approximate generalized leverage scores of $A$ w.r.t. $\tilde{A}'$
4: Use these estimates to sample rows of $A$ to form $\tilde{A}$
5: return $\tilde{A}$

The second algorithm, REFINEMENT SAMPLING, makes critical use of Theorem 9, which shows that, given a set of leverage score upper bounds, we can undersample by these estimates and still significantly improve their quality with each iteration. We start with all of our leverage score upper bounds set to 1 so we have $\|\tilde{r}\|_1 = n$. In each iteration, we sample $O(d \log d)$ rows according to our upper bounds, meaning that we undersample at rate $\alpha = O\left(\frac{d}{\|\tilde{r}\|_1}\right)$. By Theorem 9, we cut $\|\tilde{r}\|_1$ by a constant fraction in each iteration. Thus, within $\log(n)$ rounds, $\|\tilde{r}\|_1$ will be $O(d)$ and we can simply use our estimates to directly obtain a spectral approximation to $A$ with $O(d \log d)$ rows.
Algorithm 3 Refinement Sampling

input: \( n \times d \) matrix \( A \)

output: spectral approximation \( \tilde{A} \) consisting of \( O(d \log d) \) rescaled rows of \( A \)

1: Initialize a vector of leverage score upper bounds, \( \tilde{\tau} \), to 1
2: while \( \|\tilde{\tau}\|_1 > O(d) \) do
3: Undersample \( O(d \log d) \) rows of \( A \) with probabilities proportional to \( \tilde{\tau} \) to form \( \tilde{A} \)
4: Compute a vector \( u \) of approximate generalized leverage scores of \( A \) w.r.t. \( \tilde{A} \)
5: Set \( \tilde{\tau}_i = \min(\tilde{\tau}_i, u_i) \)
6: end while
7: Use the final \( \tilde{\tau} \) to sample \( O(d \log d) \) rows from \( A \) to form \( \tilde{A} \)
8: return \( \tilde{A} \)

3.5.2 Runtime Analysis

In analyzing the runtimes of these algorithms, we assume \( n = O(\text{poly}(d)) \), which is a reasonable assumption for any practical regression problem. Furthermore, we use the fact that a \( d \times d \) system can be solved in time \( d^\omega \), where \( \omega \) is the matrix multiplication exponent. However, we emphasize that, depending on the structure and sparsity of \( A \), alternative system solving methods may yield faster results or runtimes with different trade-offs. For example, if the rows of \( A \) are sparse, solving a system in \( \tilde{A} \), where \( \tilde{A} \) consists of \( O(d \log d) \) rescaled rows from \( A \) may be accelerated by using iterative conjugate gradient, or other Krylov subspace methods (which can also avoid explicitly computing \( \tilde{A}^T \tilde{A} \)). It is best to think of \( d^\omega \) as the runtime of the fastest available system solver in your domain, and the quoted runtimes as general guidelines that will change somewhat depending on exactly how the above algorithms are implemented.

First, we give an important primitive showing that estimates of generalized leverage scores can be computed efficiently. Computing exact generalized leverage scores

\[\text{Algorithm 3 Refinement Sampling}\]

input: \( n \times d \) matrix \( A \)

output: spectral approximation \( \tilde{A} \) consisting of \( O(d \log d) \) rescaled rows of \( A \)

1: Initialize a vector of leverage score upper bounds, \( \tilde{\tau} \), to 1
2: while \( \|\tilde{\tau}\|_1 > O(d) \) do
3: Undersample \( O(d \log d) \) rows of \( A \) with probabilities proportional to \( \tilde{\tau} \) to form \( \tilde{A} \)
4: Compute a vector \( u \) of approximate generalized leverage scores of \( A \) w.r.t. \( \tilde{A} \)
5: Set \( \tilde{\tau}_i = \min(\tilde{\tau}_i, u_i) \)
6: end while
7: Use the final \( \tilde{\tau} \) to sample \( O(d \log d) \) rows from \( A \) to form \( \tilde{A} \)
8: return \( \tilde{A} \)

3.5.2 Runtime Analysis

In analyzing the runtimes of these algorithms, we assume \( n = O(\text{poly}(d)) \), which is a reasonable assumption for any practical regression problem. Furthermore, we use the fact that a \( d \times d \) system can be solved in time \( d^\omega \), where \( \omega \) is the matrix multiplication exponent. However, we emphasize that, depending on the structure and sparsity of \( A \), alternative system solving methods may yield faster results or runtimes with different trade-offs. For example, if the rows of \( A \) are sparse, solving a system in \( \tilde{A} \), where \( \tilde{A} \) consists of \( O(d \log d) \) rescaled rows from \( A \) may be accelerated by using iterative conjugate gradient, or other Krylov subspace methods (which can also avoid explicitly computing \( \tilde{A}^T \tilde{A} \)). It is best to think of \( d^\omega \) as the runtime of the fastest available system solver in your domain, and the quoted runtimes as general guidelines that will change somewhat depending on exactly how the above algorithms are implemented.

First, we give an important primitive showing that estimates of generalized leverage scores can be computed efficiently. Computing exact generalized leverage scores

\[3\text{ A simple method for handling even larger values of } n \text{ is outlined in [LMP13].}\]
is slow and we only need constant factor approximations, which will only increase our
sampling rates and hence number of rows sampled by a constant factor.

**Lemma 11.** Given $B$ containing $O(d \log d)$ rescaled rows of $A$, for any $\theta > 0$, it is possible to compute an estimate of $\tau^B(A)$, $\tilde{\tau}$, in $O(d^\omega \log d + \text{nnz}(A)\theta^{-1})$ time such that, w.h.p. in $d$, for all $i$, $\tilde{\tau}_i \geq \tau^B_i(A)$ and $\tilde{\tau}_i \leq d^\theta \tau^B_i(A)$.

By setting $\theta = O(\frac{1}{\log d})$, we can obtain constant factor approximations to generalized leverage scores in time $O(d^\omega \log d + \text{nnz}(A)\log d)$.

**Proof Sketch.**Lemma 11 follows from a standard randomized approximation technique [AT11, LMP13, SS11]. Presuming $a_i \perp \ker(B)$, the general idea is to write

$$\tau^B_i(A) = a_i^T(B^T B) a_i = \|B(B^T B)^+ a_i\|_2^2.$$  

If we instead compute $\|G(B^T B)^+ a_i\|_2^2$, where $G$ is a random Gaussian matrix with $O(\theta^{-1})$ rows, then by the Johnson-Lindenstrauss Lemma, with high probability, the approximation will be within a $d^\theta$ factor of $\|B(B^T B)^+ a_i\|_2^2$ for all $i$ (See Lemma 4.5 of [LMP13]).

The naive approach requires multiplying every row by $(B^T B)^+$, which has height $d$ and would thus incur cost $\text{nnz}(A)d$. On the other hand, computing $GB(B^T B)^+$ takes at most $O(\text{nnz}(A)\theta^{-1})$ time to compute $GB$, and $O(d^\omega \log d)$ time to compute $(B^T B)$ and invert it. It then takes less than time $O(d^\omega)$ to multiply these two matrices.

With $GB(B^T B)^+$ in hand, we just need to multiply by each row in $A$ to obtain our generalized leverage scores, which takes time $O(\text{nnz}(A)\theta^{-1})$. If we use an alternative system solver instead of explicitly computing $(B^T B)^+$, the JL reduction means we only need to solve $O(\theta^{-1})$ systems in $B$ to compute $GB(B^T B)^+$ (one for each row of $G$). Note that, for sparse or structured data, all of these system solves can still take advantage of any potential computational gains. We are simply applying $(B^T B)^+$ to rows that are no longer rows of our initial matrix,

A slight modification is needed to handle the case when $a_i \not\perp \ker(B)$ – we need to check whether the vector has a component in the null-space of $B$. There are a variety of ways to handle this detection. For example, we can choose a random gaussian vector $g$ and compute $g - B^T(BB^T)^+Bg$, which is the same as $g - (B^T B)^+ B^T B g$. This gives a random vector in the null space of $B$, so computing its dot product with
any row $a_i$ will tell us (with probability 1) whether $a_i$ is orthogonal to the null space or not.

With this primitive in place, we can analyze the runtime of our two algorithms. For simplicity, we just give runtimes for computing a constant factor spectral approximation to $A$. Such an approximation is sufficient for use as a preconditioner in iterative regression algorithms [AMT10, CW13, RT08]. Furthermore, it allows us to compute leverage scores of $A$ up to a constant factor, from which we can sample rows to directly obtain a $(1 + \epsilon)$ approximation with $O(d \log d \epsilon^{-2})$ rows. By Lemma 11 the runtime of this final refinement is just $O(\text{nnz}(A) \log d + d^{\omega} \log d)$.

**Lemma 12.** **Repeated Halving** (Algorithm 2) runs in time $O(\text{nnz}(A) \log d + d^{\omega} \log(n/d) \log d)$, outputting a matrix with $\tilde{A}$ with $O(d \log d)$ rows such that $\tilde{A}^\top \tilde{A} \approx_2 A^\top A$.

**Proof.** The proof is by induction – it suffices to show that the work done at the top level is $O(\text{nnz}(A) \log d + d^{\omega} \log d)$. At each of the $O(\log(n/d))$ levels of recursion, we cut our matrix in half uniformly so $\text{nnz}(A)$ will also be cut approximately in half with high probability.

By Theorem 10, sampling by $\tau_i^A'(A)$ allows us to obtain $\tilde{A}$ with $O(d \log d)$ rows. If we instead use $\tilde{A}'$, our estimated leverage scores increase by at most a constant factor (since $\tilde{A}'$ is a constant factor spectral approximation to $A'$). Furthermore, using Lemma 11 to approximate generalized leverage scores increases our estimates by another constant factor at most. Overall, $\tilde{A}$ will have $O(d \log d)$ rows as desired and our runtime at the top level is just the runtime of estimating leverage scores from Lemma 11 – $O(\text{nnz}(A) \log d + d^{\omega} \log d)$.

**Lemma 13.** **Refinement Sampling** (Algorithm 3) runs in $O(\text{nnz}(A) \log(n/d) \log d + d^{\omega} \log(n/d) \log d)$ time, outputting an $\tilde{A}$ with $O(d \log d)$ rows such that $\tilde{A}^\top \tilde{A} \approx_2 A^\top A$.

**Proof.** The row count of $\tilde{A}$ and the fact that it spectrally approximates $A$ follows from the termination condition and Lemma 3. By Lemma 11, each iteration runs in
\(O(\text{nnz}(A) \log d + d^\omega \log d)\) time. Thus it suffices to show that the algorithm terminates after \(O(\log(n/d))\) iterations. At each iteration, we undersample by a factor \(\alpha = \frac{c_d}{\|\tilde{r}\|_1}\) for some constant \(c\). So by Theorem 9, \(\|\tilde{r}\|_1\) decreases to \(\frac{3d}{\alpha} = \frac{3\|\tilde{r}\|_1}{c}\). Setting \(c = 6\), we cut \(\|\tilde{r}\|_1\) in half each time. Since we start with \(\|\tilde{r}\|_1 = n\) and stop when \(\|\tilde{r}\|_1 = O(d)\), we terminate in \(O(\log(n/d))\) iterations.

3.5.3 Achieving Input-Sparsity Time

Using techniques from [LMP13], it is possible to remove the \(\log d\) factor on the \(\text{nnz}(A)\) term to achieve true input-sparsity time with \textsc{Repeated Halving}. The general idea is that, instead of using Lemma 11 to estimate generalized leverage scores from up to a constant factor using \(A'\), we only estimate them up to a \(d^\theta\) factor for some constant \(0 < \theta < 1\). Using these rough estimates, we obtain \(\tilde{A}\) with \(O(d^{1+\theta} \log d)\) rows. Then, for the rows in \(\tilde{A}\), we can again compute generalized leverage scores with respect to \(A'\), now up to constant factors, and reduce down to just \(O(d \log d)\) rows. In total, each iteration will take time \(O(\theta^{-1} \text{nnz}(A) + d^\omega \log d + d^{2+\theta} \log^2 d)\), so obtaining a constant factor approximation to \(A\) takes time \(O(\theta^{-1} \text{nnz}(A) + d^\omega \log^2 d + d^{2+\theta} \log^3 d)\). Recall that we assume \(n = \text{poly}(d)\), so we have \(\log(n/d) = O(\log d)\) iterations.

In order to obtain a \((1 + \epsilon)\)-spectral approximation with only \(O(d \log d^{-\epsilon/2})\) rows, we first obtain a constant factor approximation, \(\tilde{A}\), with \(O(d \log d)\) rows. We then use leverage scores estimated with \(\tilde{A}\) to compute a \((1 + \epsilon/2)\) approximation to \(A\) with \(O(d^{1+\theta} \log d^{-\epsilon/2})\) rows. Finally, we again use leverage scores estimated with \(\tilde{A}\) and Lemma 11 with \(\theta = O(1/\log d)\) to compute a \((1 + \epsilon/2)\) approximation to this smaller matrix with only \(O(d \log d^{-\epsilon/2})\) rows. This takes total time \(O(\theta^{-1} \text{nnz}(A) + d^\omega \log d + d^{2+\theta} \log^2 d^{-\epsilon/2})\). The \(d^{2+\theta} \log^2 d^{-\epsilon/2}\) comes from applying Lemma 11 to refine our second approximation, which has \(O(d^{1+\theta} \log d^{-\epsilon/2})\) rows and thus at most \(O(d^{2+\theta} \log d^{-\epsilon/2})\) nonzero entries. Overall, the technique yields:

\begin{lemma}
Given any constant \(0 < \theta \leq 1\), and any error \(0 \leq \epsilon < 1^4\), w.h.p. in
\end{lemma}

4. If \(\epsilon < 1/\text{poly}(d)\), then \(O(d \log d^{-\epsilon/2}) > n\), so we can trivially return \(\tilde{A} = A\) in \(O(\text{nnz}(A))\) time.
we can compute a matrix $\tilde{A}$ with $O(d \log \epsilon^{-2})$ rows such that $\tilde{A}^T \tilde{A} \approx_{1+\epsilon} A^T A$ in $O(\text{nnz}(A) + d^{\omega} \log^2 d + d^{2+\theta} \epsilon^{-2})$ time.

As is standard, $\log d$ factors on the $d^{2+\theta}$ term are ‘hidden’ as we can just slightly increase the value of $\theta$ to subsume them. The full tradeoff parameterized by $\theta$ is:

$$O(\theta^{-1} \text{nnz}(A) + d^{\omega} \log^2 d + d^{2+\theta}(\log^2 d + \log^2 \epsilon^{-2})).$$

### 3.5.4 General Sampling Framework

It is worth mentioning that Algorithms 2 and 3 are two extremes on a spectrum of algorithms between halving and refinement sampling. Generically, the full space of algorithms can be summarized using the pseudocode in Algorithm 4. For notation, note that $A$ always refers to our original data matrix. $\hat{A}$ is the data matrix currently being processed in the recursive call to Algorithm 4.

**Algorithm 4** Generic Row Sampling Scheme

**input:** original $n \times d$ matrix $A$, current $\bar{n} \times d$ matrix $\hat{A}$.

**output:** approximation $\hat{A}$ consisting of $O(d \log d)$ rescaled rows of $A$.

1: Uniform sample $n_1$ rows of $\hat{A}$, $A_1$
2: Approximate $A_1$ with a row sample, $A_2$, recursively
3: Estimate generalized leverage scores using $A_2$ and use them to sample $n_3$ rows of either $\hat{A}$ or $A$ itself to obtain $A_3$
4: Approximate $A_3$ with a row sample, $A_4$, recursively
5: **return** $A_4$

Different choices for $n_1$ and $n_3$ lead to different algorithms. Note that the last recursion to approximate $A_4$ has error build up incurred from sampling to create $A_3$. As a result, this generic scheme has error buildup, but it can be removed by sampling w.r.t. $A$ instead of $\hat{A}$.

Note that if we choose $n_1 = O(d \log d)$, we can simply set $A_2 \leftarrow A_1$, and the first recursive call in Line 2 is not necessary. Also, Theorem 9 gives that, if we pick
$n_1$ sufficiently large, $n_3$ can be bounded by $O(d \log d)$. This would then remove the last recursive call to compute $A_4$. Such modifications lead to head and tail recursive algorithms, as well as a variety of intermediate forms:

1. Head recursive algorithm, $n_1 = n/2$, giving Algorithm 2 (REPEATED HALVING).

2. Tail recursive algorithm, $n_1 = d \log d$, $n_3 = \frac{n}{2}$, sampled w.r.t. $\hat{A}$. At each step error compounds so setting error to $\frac{1}{\log n}$ per step gives a constant factor approximation.

3. $n_1 = d \log d$, $n_3 = d \log d$, sampled w.r.t. $A$, giving Algorithm 3 (REFINEMENT SAMPLING).

4. For situations where iterations are expensive, e.g. MapReduce, a useful choice of parameters is likely $n_1 = n_3 = O(\sqrt{nd \log d})$. This allows one to compute $A_2$ and $A_4$ without recursion, while still giving speedups.
Chapter 4

Single Pass Spectral Sparsification in Dynamic Streams

This chapter presents work published in [KLM+ 14], and is thus a collaborative effort between Yin Tat Lee, Michael Kapralov, Cameron Musco, Aaron Sidford, and myself.

4.1 Introduction

In the previous chapter, we showed how to obtain a spectral approximation for a matrix $A \in \mathbb{R}^{n \times d}$ that contains just $O(d \log d)$ original rows from $A$. [SS11] was one of the first papers to use leverage score sampling, the basis of our results. Their paper focuses exclusively on sketching the highly structured vertex-edge incidence matrices of graphs. In this case, a weighted row sample from $A$ is the vertex-edge incidence matrix of a weighted subgraph of the graph represented by $A$.

This property is essential for the application of dimensionality reduction to graphs, where spectral approximations are often used to approximate random walk properties, cut properties, spectral clusterings, and other combinatorial properties of the graph underlying $A$. By returning a row sample (the vertex-edge incidence matrix of a subgraph), we can apply combinatorial algorithms to our sketch $\tilde{A}$, which would be impossible if the matrix were produced by a projection method that arbitrarily mixed rows in $A$. 

53
Furthermore, a graph vertex-edge incidence matrix with $O(d \log d)$ rows can be stored in $O(d \log d)$ space, in contrast to $O(d^2 \log d)$ for an arbitrary dense $\tilde{A}$, and it is possible to solve systems in such matrices in nearly linear time [ST14]. These additional benefits of structure preserving dimensionality reduction make the case very clear for sampling over random projection when compressing graphs.

However, as discussed in Section 1.1.1, random projection offers significant advantages of its own. Specifically, random projection sketches are much easier to apply in streaming, distributed, and dynamic settings since they are oblivious. All we have to do is multiply $A$ by a random matrix chosen independently from $A$. On the other hand, sampling requires carefully computed probabilities, which depend on global properties of our data matrix.

Fortunately, it turns out that, at least for graphs and matrices with similar structure, we don’t have to pay for this tradeoff between sampling and oblivious sketching! This chapter shows how to compute a graph compression obliviously in nearly optimal $\tilde{O}(d)$ space. From the sketch, we show how to extract a legitimate subgraph whose vertex-edge incidence matrix spectrally approximates that of our original graph. We introduced and motivate our technique by applying it to the challenging open problem of dynamic spectral graph sparsification.

### 4.1.1 The Dynamic Semi-Streaming Model

To efficiently process massive datasets, the streaming model of computation restricts algorithms to few passes over the input and space polylogarithmic in the input size. Streaming algorithms have been studied extensively in various application domains – see [Mut05] for an overview. However, the model has proven too restrictive for even the simplest graph algorithms, rendering it useless for analyzing social networks, web topologies, or interaction graphs. For example, simply testing $s$-$t$ connectivity in a graph with $d$ nodes requires $\Omega(d)$ space [HRR99].

The less restrictive semi-streaming model, in which the algorithm is allowed $\tilde{O}(d)$ space, is more suited for graph algorithms [FKM+05], and has received significant attention in recent years. In this model, a processor receives a stream of edges over a
fixed set of $d$ nodes. Ideally, the processor should only have to perform a single pass (or few passes) over the edge stream, and the processing time per edge, as well as the time required to output the final answer, should be small.

In the dynamic semi-streaming model, the graph stream may include both edge insertions and deletions [AGM12a]. This extension captures the fact that large graphs are unlikely to be static. Dynamic semi-streaming algorithms allow us to quickly process general updates in the form of edge insertions and deletions to maintain a small-space representation of the graph from which we can later compute a result. Sometimes the dynamic model is referred to as the insertion-deletion model, in contrast to the more restrictive insertion-only model.

Work on semi-streaming algorithms in both the dynamic and insertion-only settings is extensive. Researchers have tackled connectivity, bipartiteness, minimum spanning trees, maximal matchings, spanners, and hypergraph problems among many others [FKM+05, ELMS11, Elk11, AGM12a, AGM12b, CCE+15, AKLY15, Kon15, KK15, GMT15, BHNT15]. In [McG14], McGregor surveys much of this progress and provides a more complete list of citations.

### 4.1.2 Streaming Sparsification

There has also been a focus on computing general purpose graph compressions in the streaming setting. The goal is to find a subgraph of an input graph $G$ that has significantly fewer edges than $G$, but still maintains important properties of the graph. Hopefully, this sparsified graph can be used to approximately answer a variety of questions about $G$ with reduced space and time complexity.

First introduced by Benczúr and Karger [BK96], a cut sparsifier of a graph $G$ is a weighted subgraph with only $O(\frac{1}{\varepsilon^2}d \log d)$ edges that preserves the total edge weight over every cut in $G$ to within a $(1 \pm \varepsilon)$ multiplicative factor. Cut sparsifiers can be used to compute approximations for minimum cut, sparsest cut, maximum flow, and a variety of other problems over $G$. In [ST11], Spielman and Teng introduce the stronger spectral sparsifier, a weighted subgraph whose Laplacian spectrally approximates the Laplacian of $G$. In addition to maintaining the cut approximation of Benczúr and
Karger, spectral sparsifiers can be used to approximately solve linear systems over the Laplacian of $G$, and to approximate effective resistances, spectral clusterings, random walk properties, and a variety of other computations.

It turns out that spectral sparsification is exactly equivalent to our earlier notion of spectral approximation for matrices—a spectral approximation of $G$'s vertex-edge incidence matrix (obtained via sampling) gives the vertex-edge incidence matrix of a spectral sparsifier. Again, a $(1 + \epsilon)$ error sparsifier requires $O(\frac{1}{\epsilon^2}d \log d)$ edges (i.e. rows in our matrix approximation).

The problem of computing graph sparsifiers in the semi-streaming model has received a lot of attention. Given just $\tilde{O}(d) = O(d \text{polylog}(d))$ space, the hope is to compute a sparsifier using barely more space than required to store the sparsifier, which will typically have $O(d \log d)$ edges. Ahn and Guha give the first single pass, insertion-only algorithm for cut sparsifiers [AG09]. Kelner and Levin give a single pass, insertion-only algorithm for spectral sparsifiers [KL13]. Both algorithms store a sparse graph: edges are added as they are streamed in and, when the graph grows too large, it is resparsified. The construction is very clean, but inherently does not extend to the dynamic model since, to handle edge deletions, we need more information than just a sparsifier itself. Edges eliminated to create an intermediate sparsifier may become critically important later if other edges are deleted, so we need to maintain information that allows recovery of such edges.

Ahn, Guha, and McGregor make a very important insight in [AGM12a], demonstrating the power of linear graph sketches in the dynamic model. They present the first dynamic algorithm for cut sparsifiers, which initially required $O(\frac{1}{\epsilon^2} d^{1+\gamma})$ space and $O(1/\gamma)$ passes over the graph stream. However, the result was later improved to a single pass and $O(\frac{1}{\epsilon^2} d \text{polylog}(d))$ space [AGM12b, GKP12]. Our algorithm extends the sketching and sampling approaches from these papers to the spectral problem.

In [AGM13], the authors show that linear graph sketches that capture connectivity information can be used to coarsely approximate spectral properties and they obtain spectral sparsifiers using $O(\frac{1}{\epsilon^2} d^{5/3} \text{polylog}(d))$ space in the dynamic setting. However, they also show that their coarse approximations are tight, so a new approach
is required to obtain spectral sparsifiers using just $O(\frac{1}{\epsilon^2}d \text{polylog}(d))$ space. They conjecture that a dynamic algorithm for doing so exists. The development of such an algorithm is also posed as an open question in [McG14]. A two-pass algorithm for constructing a spectral sparsifier in the dynamic streaming model using $O(\frac{1}{\epsilon^2}d^{1+o(1)})$ space is presented in [KW14]. The approach is very different from ours: it leverages a reduction from spanner constructions to spectral sparsification presented in [KP12]. It is not known if this approach extends to a space efficient single pass algorithm.

### 4.1.3 Our Contributions

Our main result is an algorithm for maintaining a small graph sketch from which we can recover a spectral sparsifier. For simplicity, we present the algorithm in the case of unweighted graphs. However, in Section 4.6, we show that it is easily extended to weighted graphs. This model matches what is standard for dynamic cut sparsifiers [AGM12b, GKP12].

**Theorem 15 (Main Result).** There exists an algorithm that, for any $\epsilon > 0$, processes a list of edge insertions and deletions for an unweighted graph $G$ in a single pass and maintains a set of linear sketches of this input in $O(\frac{1}{\epsilon^2}d \text{polylog}(d))$ space. From these sketches, it is possible to recover, with high probability, a weighted subgraph $H$ with $O(\frac{1}{\epsilon^2}d \log d)$ edges such that $H$ is a $(1\pm\epsilon)$ spectral sparsifier of $G$. The algorithm recovers $H$ in $O(\frac{1}{\epsilon^2}d^2 \text{polylog}(d))$ time.

It is well known that independently sampling edges from a graph $G$ according to their effective resistances (i.e. leverage scores) gives a $(1\pm\epsilon)$ spectral sparsifier of $G$ with $O(\frac{1}{\epsilon^2}d \log d)$ edges [SS11]. We can ‘refine’ any coarse sparsifier for $G$ by using it to approximate effective resistances and then resample edges according to these approximate resistances. We show how to perform this refinement in the streaming setting, extending graph sketching techniques initially used for cut sparsifiers ([AGM12b, GKP12]) and introducing a new sampling technique based on an $\ell_2$ heavy hitters algorithm. Our refinement procedure is combined with a clever recursive method for obtaining a coarse sparsifier introduced by Miller and Peng in their
recent paper on iterative row sampling for matrix approximation [MP12], which we
expand on in Chapter 3.

Our algorithm maintains an oblivious linear sketch of the streamed graph using
hashing and random projection techniques. Thus, it can easily handle edge deletions,
which are simply treated as negative edge insertions. Additionally, our sketches are
composable - sketches of subgraphs can simply be added to produce a sketch of
the full graph. Thus, our techniques are directly applicable in distributed settings
where separate processors hold different subgraphs or each processes different edge
substreams.

In addition to the algorithmic advantages, the application of linear sketching also
gives a nice information theoretic result on graph compression. A spectral sparsifier is
a powerful compression for a graph. It maintains, up to an $\epsilon$ factor, all spectral infor-
mation about the Laplacian using just $O(\frac{1}{\epsilon^2}d \log d)$ space. As mentioned, this beats
the standard $O(\frac{1}{\epsilon^2}d^2)$ space required for spectrally approximating general matrices
($O(\frac{1}{\epsilon^2}d)$ rows, each of size $d$). At first glance, it may seem that such a compression
requires careful analysis of the input graph to determine what information to keep
and what to discard. However, our non-adaptive linear sketches produce a compres-
sion that holds as much information as a spectral sparsifier - in fact, we show how to
extract a spectral sparsifier from it. Furthermore, the compression is only larger than
$O(\frac{1}{\epsilon^2}d \log d)$ by log factors. This ability to eliminate adaptivity without a significant
space cost mirrors results in sparse recovery for vectors. Our algorithm can be viewed
as a compressive sensing or sparse recovery type result for graphs.

Finally, it can be noted that our proofs rely very little on the fact that our
data stream represents a graph. We show that, with a few modifications, given a
stream of row updates for a general structured matrix $A$, it is possible to maintain a
$O(\frac{1}{\epsilon^2}d \polylog(d))$ sized sketch from which a spectral approximation to $A^T A$ can be
recovered. By structured, we mean any matrix whose rows are selected from some
fixed dictionary of size $\poly(d)$. Spectral graph sparsification is a special case of this
problem: set $A$ to be the vertex-edge incidence matrix of our graph. The dictionary
is the set of all possible $\binom{d}{2}$ edge rows that may appear in $A$ and $A^T A$ is the graph
Laplacian.

### 4.1.4 Road Map

**Section 4.2** Introduce additional notation and foundations for spectral sparsification and sparse recovery not covered in Chapter 2.

**Section 4.3** Give an overview of our main sparsification algorithm, providing intuition and motivation.

**Section 4.4** Present an algorithm of Miller and Peng ([MP12]) for building a chain of coarse sparsifiers and prove our main result, assuming a primitive for sampling edges by effective resistance in the dynamic streaming model.

**Section 4.5** Develop this sampling primitive, our main technical contribution.

**Section 4.6** Show how to extend the algorithm to weighted graphs.

**Section 4.7** Show how to extend the algorithm to general structured matrices.

**Section 4.8** Remove our assumption of fully independent hash functions, using a pseudorandom number generator to achieve a final small space algorithm.

### 4.2 Additional Notation and Preliminaries

For clarity, we will always use $B$ to refer to a matrix representation of a graph, whereas $A$ will be reserved for general matrices.

#### 4.2.1 Graph Notation

Let $B_n \in \mathbb{R}^{\binom{d}{2} \times d}$ be the vertex edge incidence matrix of the undirected, unweighted complete graph over $d$ vertices. $b_e$, the row corresponding to edge $e = (u, v)$ contains a 1 in column $u$, a $(-1)$ in column $v$, and 0's elsewhere.

We write the vertex edge incidence matrix of an unweighted, undirected graph $G(V, E)$ as $B = SB_d$ where $S$ is an $\binom{d}{2} \times \binom{d}{2}$ diagonal matrix with ones at positions
corresponding to edges contained in $G$ and zeros elsewhere.$^1$ The $d \times d$ Laplacian matrix of $G$ is given by $K = B^T B$.

### 4.2.2 Spectral Sparsification

For consistency with prior work on graphs, we will slightly redefine our definition of spectral approximation/spectral sparsification from Section 2.2.3. The requirement we use is exactly the same, and accordingly we can apply the same sampling procedures to generate sparsifiers. Only the approximation factor is defined differently.

For any matrix $B \in \mathbb{R}^{n \times d}$, $\tilde{K}$ is a $(1 \pm \epsilon)$ spectral sparsifier of $K = B^T B$ if, $\forall x \in \mathbb{R}^d$, $(1 - \epsilon)x^T K x \leq x^T \tilde{K} x \leq (1 + \epsilon)x^T K x$. Again, this condition can also be written as $(1 - \epsilon)K \preceq \tilde{K} \preceq (1 + \epsilon)K$ where $C \preceq D$ indicates that $D - C$ is positive semidefinite and, more succinctly, $\tilde{K} \approx_{\epsilon} K$ denotes the same condition in this chapter. We also use the slightly weaker notation $(1 - \epsilon)K \preceq_r \tilde{K} \preceq_r (1 + \epsilon)K$ to indicate that $(1 - \epsilon)x^T K x \leq x^T \tilde{K} x \leq (1 + \epsilon)x^T K x$ for all $x$ in the row span of $K$. If $\tilde{K}$ has the same row span as $K$ this notation is equivalent to the initial notion of spectral sparsification/approximation.

Note that we do not always require our approximation $\tilde{K}$ to be the graph Laplacian of a weighted subgraph, which is a standard assumption. For this reason, we avoid the standard $L_G$ notation for the Laplacian. For our purposes, $\tilde{K}$ is always a sparse symmetric diagonally dominant matrix with no more than $O(d \log d)$ non-zero entries. In fact, it will always be the Laplacian of a sparse subgraph, but possibly with weight added to its diagonal entries. Furthermore, the final approximation returned by our streaming algorithm will be a bonafide spectral graph sparsifier – i.e. the Laplacian matrix of a weighted subgraph of $G$.

### 4.2.3 Sparse Recovery

While we cannot sample by leverage score directly in the streaming model, we can use a sparse recovery primitive to sample edges from a set of linear sketches. We use an $^1$ Typically rows of $B$ that are all 0 are removed, but this formulation more convenient for us.
\( \ell_2 \) heavy hitters algorithm that, for any vector \( \mathbf{x} \), lets us recover from a small linear sketch \( \mathbf{F} \mathbf{x} \), the index \( i \) and the approximate value of \( x_i \) for all \( i \) such that \( x_i > \eta \| \mathbf{x} \|_2 \).

**Lemma 16 (\( \ell_2 \) Heavy Hitters).** For any \( \eta > 0 \), there is a decoding algorithm \( D \) and a distribution on matrices \( \mathbf{F} \) in \( \mathbb{R}^{O(\eta^{-2} \text{polylog}(N)) \times N} \) such that, for any \( \mathbf{x} \in \mathbb{R}^N \), given \( \mathbf{F} \mathbf{x} \), the algorithm \( D \) returns a vector \( \mathbf{w} \) such that \( \mathbf{w} \) has \( O(\eta^{-2} \text{polylog}(N)) \) non-zeros and satisfies

\[
\| \mathbf{x} - \mathbf{w} \|_\infty \leq \eta \| \mathbf{x} \|_2.
\]

with probability \( 1 - N^{-c} \) over the choice of \( \mathbf{F} \). The sketch \( \mathbf{F} \mathbf{x} \) can be maintained and decoded in \( O(\eta^{-2} \text{polylog}(N)) \) space.

This procedure allows us to distinguish from a sketch whether or not a specified entry in \( \mathbf{x} \) is equal to 0 or has value \( > 2\eta \| \mathbf{x} \|_2 \). We give a proof of Lemma 16 in Appendix B

### 4.3 Algorithm Overview

Before formally presenting a proof of our main result, Theorem 15, we give an informal overview of the algorithm to provide intuition.

#### 4.3.1 Effective Resistances

As explained in Section 2.2.5, spectral sparsifiers can be generated by sampling edges, i.e. rows of the vertex-edge incidence matrix. For a graph \( G \), each edge \( e \) is sampled independently with probability proportional to its leverage score, \( \tau_e \). After sampling, we reweight and combine any sampled edges. The result is a subgraph of \( G \) containing, with high probability, \( O(\frac{1}{d} \log d) \) edges and spectrally approximating \( G \).

If we view an unweighted \( G \) as an electrical circuit, with each edge representing a unit resistor, the leverage score of an edge \( e = (i, j) \) is equivalent to its effective resistance. This value can be computed by forcing 1 unit of current out of vertex \( i \) and 1 unit of current into vertex \( j \). The resulting voltage difference between the two vertices is the effective resistance of \( e \). Qualitatively, if the voltage drop is low,
there are many low resistance (i.e. short) paths between $i$ and $j$. Thus, maintaining a direct connection between these vertices is less critical in approximating $G$, so $e$ is less likely to be sampled. Effective resistance can be computed as:

$$\tau_e = b_e^T K^+ b_e$$

Note that $\tau_e$ can be computed for any pair of vertices, $(i, j)$, or in other words, for any possible edge in $G$. We can evaluate $b_e^T K^+ b_e$ even if $e$ is not present in the graph. Thus, we can reframe our sampling procedure. Instead of just sampling edges actually in $G$, imagine we run a sampling procedure for every possible $e$. When recombining edges to form a spectral sparsifier, we separately check whether each edge $e$ is in $G$ and only insert into the sparsifier if it is.

### 4.3.2 Sampling in the Streaming Model

With this procedure in mind, a sampling method that works in the streaming setting requires two components. First, we need to obtain a constant factor approximation to $\tau_e$ for any $e$. Known sampling algorithms, including our Lemma 3, are robust to this level of estimation. Second, we need to compress our edge insertions and deletions in such a way that, during post-processing of our sketch, we can determine whether or not a sampled edge $e$ actually exists in $G$.

The first requirement is achieved through the recursive procedure given in [MP12]. We will give the overview shortly but, for now, assume that we have access to a coarse sparsifier, $\tilde{K} \approx_{1/2} K$. Computing $b_e^T \tilde{K}^+ b_e$ gives a 2 factor multiplicative approximation of $\tau_e$ for each $e$. Furthermore, as long as $\tilde{K}$ has sparsity $O(d \log d)$, the computation can be done in small space using an iterative system solver (e.g. conjugate gradient) or a nearly linear time solver for symmetric diagonally dominant matrices (e.g. [KMP11]).

Solving part two (determining which edges are actually in $G$) is a bit more involved.
As a first step, consider writing

\[ \tau_e = b_e^T K^+K^+b_e = \|BK^+b_e\|_2^2 = \|SB_dK^+b_e\|_2^2. \]

Referring to Section 4.2, recall that \( B = SB_d \) is exactly the same as a standard vertex edge incidence matrix except that rows in \( B_d \) corresponding to nonexistent edges are zeroed out instead of removed. Denote \( x_e = SB_dK^+b_e \). Each nonzero entry in \( x_e \) contains the voltage difference across some edge (resistor) in \( G \) when one unit of current is forced from \( i \) to \( j \).

When \( e \) is not in \( G \), then the \( e \)th entry of \( x_e \), \( x_e(e) \) is 0. If \( e \) is in \( G \), \( x_e(e) = \tau_e \). Furthermore, \( \|x_e\|_2^2 = \tau_e \). Given a space allowance of \( \text{polylog}(d) \), the sparse recovery algorithm from Lemma 16 allows us to recover an entry if it accounts for at least an \( \Omega(1/\text{polylog}(d)) \) fraction of the total \( \ell_2 \) norm. Currently, \( x_e(e)/\|x_e\|_2 = \sqrt{\tau_e} \), which could be much smaller than \( O(1/\text{polylog}(d)) \). However, suppose we had a sketch of \( x_e \) with all but a \( \tau_e \) fraction of edges randomly sampled out. Then, we would expect \( \|x_e\|_2^2 \approx \tau_e^2 \) and thus, \( x_e(e)/\|x_e\|_2 = O(1) = \Omega(1/\text{polylog}(d)) \) and sparse recovery would successfully indicate whether or not \( e \in G \). What’s more, randomly zeroing out entries of \( x_e \) can serve as our main sampling routine for edge \( e \). This process will set \( x_e(e) = 0 \) with probability \( 1 - \tau_e \), exactly what we wanted to sample by in the first place!

But how do we go about sketching every appropriately sampled \( x_e \)? Consider subsampling our graph at geometrically decreasing rates, \( 1/2^s \) for \( s \in \{0, 1, \ldots, O(\log d)\} \). Maintain linear sketches \( \Pi_1B_1, \ldots, \Pi_{O(\log d)}B_{O(\log d)} \) of the vertex edge incidence matrix for every subsampled graph using the \( \ell_2 \) sparse recovery sketch distribution from Lemma 16. When asked to output a spectral sparsifier, for every possible edge \( e \), we compute using \( \tilde{K} \) a rate \( 1/2^s \) that approximates \( \tau_e \).

Since each sketch is linear, we can just multiply \( \Pi_{1/2^s}B_{1/2^s} \) on the right by \( \tilde{K}^+b_e \) to compute

\[ \Pi_{1/2^s}B_{1/2^s}\tilde{K}^+b_e \approx \Pi_{1/2^s}x_e^{1/2^s}. \]
where $x_e^{1/2^s}(e)$ is $x_e$ sampled at rate $1/2^s \approx \tau_e$. Then, as explained, we can use our sparse recovery routine to determine whether or not $e$ is present. If it is, we have obtained a sample for our spectral sparsifier!

### 4.3.3 A Chain of Coarse Sparsifiers

Our final requirement is access to some sparse $\tilde{K} \approx_{1/2} K$. This coarse sparsifier is obtained recursively by constructing a chain of matrices, $[K(0), K(1), \ldots, K(m), K]$ each weakly approximating the next. Specifically, imagine producing $K(m)$ by adding a fairly light identity matrix to $K$. As long as the identity’s weight is small compared to $K$’s spectrum, $K(m)$ approximates $K$. Add even more weight to the diagonal to form $K(m - 1)$. Again, as long as the increase is small, $K(m - 1)$ approximates $K(m)$. We continue down the chain until $K(0)$, which will actually have a heavy diagonal after all the incremental increases. Thus, $K(0)$ can be approximated by an appropriately scaled identity matrix, which is clearly sparse. Miller and Peng show that parameters can be chosen such that $m = O(\log d)$ [MP12].

Putting it all together, we maintain $O(\log d)$ sketches $[K(0), K(1), \ldots, K(m), K]$. We first use a weighted identity matrix as a coarse approximation for $K(0)$, which allows us to recover a good approximation to $K(0)$ from our sketch. This approximation will in turn be a coarse approximation for $K(1)$, so we can recover a good sparsifier of $K(1)$. Continuing up the chain, we eventually recover a good sparsifier for our final matrix, $K$.

### 4.4 Recursive Sparsifier Construction

In this section, we formalize the recursive procedure for obtaining a chain of coarse sparsifiers that was introduced by Miller and Peng – “Introduction and Removal of Artificial Bases” [MP12]. We prove Theorem 15 by combining this technique with the sampling algorithm developed in Section 4.5.

**Theorem 17** (Recursive Sparsification – [MP12], Section 4). Consider any PSD
matrix $\mathbf{K}$ with maximum eigenvalue bounded from above by $\lambda_u$ and minimum non-zero eigenvalue bounded from below by $\lambda_l$. Let $d = \lceil \log_2(\lambda_u/\lambda_l) \rceil$. For $\ell \in \{0, 1, 2, ..., m\}$, define

$$\gamma(\ell) = \frac{\lambda_u}{2^\ell}.$$  

So, $\gamma(0) = \lambda_u$ and $\gamma(m) \leq \lambda_l$. Then the PSD matrices, $[\mathbf{K}(0), \mathbf{K}(1), \ldots, \mathbf{K}(m)]$ with

$$\mathbf{K}(\ell) = \mathbf{K} + \gamma(\ell)\mathbf{I}_{d \times d},$$

satisfies the following relations:

1. $\mathbf{K} \preceq_r \mathbf{K}(m) \preceq_r 2\mathbf{K}$,
2. $\mathbf{K}(\ell) \preceq \mathbf{K}(\ell - 1) \preceq 2\mathbf{K}(\ell)$ for all $\ell \in \{1, \ldots, m\}$,
3. $\mathbf{K}(0) \preceq 2\gamma(0)\mathbf{I} \preceq 2\mathbf{K}(0)$.

When $\mathbf{K}$ is the Laplacian of an unweighted graph, its largest eigenvalue $\lambda_{\text{max}} < 2d$ and its smallest non-zero eigenvalue $\lambda_{\text{min}} > 8/d^2$. Thus the length of our chain, $m = \lceil \log_2 \lambda_u/\lambda_l \rceil$, is $O(\log d)$.

For completeness, we include a proof of Theorem 17 in Appendix C. Now, to prove our main result, we need to state the sampling primitive for streams that we develop in Section 4.5. This procedure maintains a linear sketch of a vertex edge incidence matrix $\mathbf{B}$, and using a coarse sparsifier of $\mathbf{K}(\ell) = \mathbf{B}^\top \mathbf{B} + \gamma(\ell)\mathbf{I}$, performs independent edge sampling as required by Lemma 3, to obtain a better sparsifier of $\mathbf{K}(\ell)$.

**Theorem 18.** Let $\mathbf{B} \in \mathbb{R}^{d \times n}$ be the vertex edge incidence matrix of an unweighted graph $G$, specified by an insertion-deletion graph stream. Let $\gamma = \text{poly}(d)$ be a fixed parameter and consider $\mathbf{K} = \mathbf{B}^\top \mathbf{B} + \gamma(\ell)\mathbf{I}$. For any $0 < \epsilon < 1$, there exists a sketching procedure $\text{MaintainSketches}(\mathbf{B}, \epsilon)$ that outputs an $O(d \text{ polylog}(d))$ sized sketch $\mathbf{I}_{d\mathbb{B}}$. There exists a corresponding recovery algorithm $\text{RefineSparsifier}$ running in $O(d \text{ polylog}(d))$ space, such that, if $\bar{\mathbf{K}}$ is a spectral approximation to $\mathbf{K}$ with $O(d \log d)$
non-zeros and \( cK \preceq r \tilde{K} \preceq r K \) for some constant \( 0 < c < 1 \) then:

\[ \text{RefineSparsifier}(\Pi B, \tilde{K}, \gamma, \epsilon, c) \text{ returns, with high probability, } \tilde{K}_\epsilon = \tilde{B}_\epsilon^T \tilde{B}_\epsilon + \gamma I, \]

where \((1 - \epsilon)K \preceq r \tilde{K}_\epsilon \preceq r (1 + \epsilon)K\), and \( \tilde{B}_\epsilon \) contains only \( O(\epsilon^{-2}c^{-1}d \log d) \) reweighted rows of \( B \) with high probability. \text{RefineSparsifier} runs in \( O(d^2 \text{polylog}(d)) \) time.

Using this sampling procedure, we can initially set \( \tilde{K} = 2\gamma(0)I \) and use it obtain a sparsifier for \( K(0) \) from a linear sketch of \( B \). This sparsifier is then used on a second sketch of \( B \) to obtain a sparsifier for \( K(1) \), and so on. Working up the chain, we eventually obtain a sparsifier for our original \( K \). While sparsifier recovery proceeds in several levels, we construct all required sketches in a single pass over edge insertions and deletions. Recovery is performed in post-processing.

\textit{Proof of Theorem 15.} Let \( K \) be the Laplacian of our graph \( G \). Process all edge insertions and deletions, using \text{MaintainSketches} to produce a sketch, \((\Pi B)_\ell \) for each \( \ell \in \{0, 1, \ldots, \lceil \log_2 \lambda_u/\lambda_1 \rceil + 1 \}. \) We then use Theorem 18 to recover an \( \epsilon \) approximation, \( \tilde{K}(\ell) \), for any \( K(\ell) \) given an \( \epsilon \) approximation for \( K(\ell - 1) \). First, consider the base case, \( K(0) \). Let:

\[ \tilde{K}(0) = \text{RefineSparsifier}(\Pi B)_0, \gamma(0)I, \gamma(0), \epsilon, \frac{1}{2} \).

By Theorem 17, Relation 3:

\[ \frac{1}{2} K(0) \preceq \gamma(0)I \preceq K(0). \]

Thus, with high probability, \((1 - \epsilon)K(0) \preceq r \tilde{K}(0) \preceq r (1 + \epsilon)K(0)\) and \( \tilde{K}(0) \) contains \( O((1/2)^{-1} \cdot d \log d \cdot \epsilon^{-2}) = O(\epsilon^{-2}d \log d) \) entries.

Now, consider the inductive case. Suppose we have some \( \tilde{K}(\ell - 1) \) such that \((1 - \epsilon)K(\ell - 1) \preceq r \tilde{K}(\ell - 1) \preceq r (1 + \epsilon)K(\ell - 1) \). Let:

\[ \tilde{K}(\ell) = \text{RefineSparsifier}(\Pi B)_\ell, \frac{1}{2}(1 + \epsilon)K(\ell - 1), \gamma(\ell), \epsilon, \frac{1 - \epsilon}{2(1 + \epsilon)}. \]
By Theorem 17, Relation 2:

\[ \frac{1}{2} K(\ell) \preceq \frac{1}{2} K(\ell - 1) \preceq K(\ell). \]

Furthermore, by assumption we have the inequalities:

\[ \frac{1 - \epsilon}{1 + \epsilon} K(\ell - 1) \preceq \frac{1}{1 + \epsilon} \tilde{K}(\ell - 1) \preceq K(\ell - 1). \]

Thus:

\[ \frac{1 - \epsilon}{2(1 + \epsilon)} K(\ell) \preceq \frac{1}{2(1 + \epsilon)} \tilde{K}(\ell - 1) \preceq K(\ell). \]

So, with high probability \textbf{RefineSparsifier} returns \(\tilde{K}(\ell)\) such that \((1 - \epsilon)K(\ell) \preceq \tilde{K}(\ell) \preceq (1 + \epsilon)K(\ell)\) and \(\tilde{K}(\ell)\) contains just \(O\left(\frac{2(1+\epsilon)}{1-\epsilon}\right)^2 \epsilon^{-2d} \log d = O(\epsilon^{-2d} \log d)\) nonzero elements. It is important to note that there is no “compounding of error” in this process. Every \(\tilde{K}(\ell)\) is an \(\epsilon\) approximation for \(K(\ell)\). Error from using \(\tilde{K}(\ell - 1)\) instead of \(K(\ell - 1)\) is absorbed by a constant factor increase in the number of rows sampled from \(B\). The corresponding increase in sparsity for \(K(\ell)\) does not compound - in fact Theorem 18 is completely agnostic to the sparsity of the coarse approximation \(\tilde{K}\) used.

Finally, to obtain a bonafide graph sparsifier (a weighted subgraph of our streamed graph), for \(m = \lceil \log_2 \lambda_u / \lambda_f \rceil\) let

\[ \tilde{K} = \text{RefineSparsifier}((\Pi B)^m+1, \frac{1}{2(1 + \epsilon)} \tilde{K}(m), 0, \epsilon, \frac{1 - \epsilon}{2(1 + \epsilon)}). \]

As in the inductive case,

\[ \frac{1 - \epsilon}{2(1 + \epsilon)} K \preceq \frac{1}{2(1 + \epsilon)} \tilde{K}(m) \preceq K. \]

Thus, it follows that, with high probability, \(\tilde{K}\) has sparsity \(O(\epsilon^{-2d} \log d)\) and \((1 - \epsilon)K \preceq \tilde{K} \preceq (1 + \epsilon)K\). Since we set \(\gamma\) to 0 for this final step, \(\tilde{K}\) simply equals \(\tilde{B}^\top \tilde{B}\)
for some \( \tilde{B} \) that contains reweighted rows of \( B \). Any vector in the kernel of \( B \) is in the kernel of \( \tilde{B} \), and thus any vector in the kernel of \( K \) is in the kernel of \( \tilde{K} \). Thus, we can strengthen our approximation to:

\[
(1 - \epsilon)K \preceq \tilde{K} \preceq (1 + \epsilon)K.
\]

We conclude that \( \tilde{K} \) is the Laplacian of some graph \( H \) containing \( O(\epsilon^{-2}d\log d) \) reweighted edges and approximating \( G \) spectrally to precision \( \epsilon \). Finally, note that we require \( m + 1 = O(\log d) \) recovery steps, each running in \( O(d^2 \text{polylog}(d)) \) time. Thus, our total recovery time is \( O(d^2 \text{polylog}(d)) \).

4.5 Streaming Row Sampling

In this section, we develop the sparsifier refinement routine required for Theorem 15.

Proof of Theorem 18. Outside of the streaming model, given full access to \( B \) rather than just a sketch \( \Pi B \) it is easy to implement \texttt{RefineSparsifier} via leverage score sampling. Letting \( \oplus \) denote appending the rows of one matrix to another, we can define \( B_\gamma = B \oplus \sqrt{\gamma(\ell)} \cdot I \), so \( K = B^\top B + \gamma I = B_\gamma^\top B_\gamma \). Since \( \tau_i = b_i^\top K^+ b_i \) and \( \epsilon K \preceq \tilde{K} \preceq \epsilon K \), for any row of \( B_\gamma \) we have

\[
\tau_i \leq b_i^\top \tilde{K}^+ b_i \leq \frac{1}{\epsilon} \tau_i.
\]

Let \( \tilde{\tau}_i = b_i^\top \tilde{K}^+ b_i \) be the leverage score of \( b_i \) approximated using \( \tilde{K} \). Let \( \tilde{\tau} \) be the vector of approximate leverage scores, with the leverage scores of the \( n \) rows corresponding to \( \sqrt{\gamma(\ell)} I \) rounded up to 1. While not strictly necessary, including rows of the identity with probability 1 will simplify our analysis in the streaming setting. Using this \( \tilde{\tau} \) in Lemma 3, we can obtain \( \tilde{K}_\epsilon \approx \epsilon K \) with high probability. Since \( \| \tilde{\tau} \|_1 \leq \frac{1}{\epsilon} \| \tau \|_1 + d \leq \frac{1}{\epsilon} \cdot \text{rank}(B) + d \leq \frac{d}{\epsilon} + d \), we can write \( \tilde{K}_\epsilon = \tilde{B}_\epsilon^\top \tilde{B}_\epsilon + \gamma I \), where \( \tilde{B}_\epsilon \) contains \( O(\epsilon^{-2}\epsilon^{-1}d\log d) \) reweighted rows of \( B \) with high probability.

The challenge in the semi-streaming setting is actually sampling edges given only
a sketch of B. The general idea is explained in Section 4.3, with detailed pseudocode included below.

**Streaming Sparsifier Refinement**

MaintainSketches(B, \( \varepsilon \)):

1. For \( s \in \{1, \ldots, O(\log d)\} \) let \( h_s : \binom{d}{s} \rightarrow \{0, 1\} \) be a uniform hash function. Let \( B_s \) be \( B \) with all rows except those with \( \prod_{j \leq s} h_j(e) = 0 \) zeroed out. So \( B_s \) is \( B \) with rows sampled independently at rate \( \frac{1}{2s} \). \( B_0 \) is simply \( B \).

2. Maintain sketches \( \Pi_0 B_0, \Pi_1 B_1, \ldots, \Pi_{O(\log d)} B_{O(\log d)} \) where \( \{\Pi_0, \Pi_1, \ldots, \Pi_{O(\log d)}\} \) are drawn from the distribution from Lemma 16 with \( \eta = \frac{c_1}{c_2 \sqrt{\log d}} \).

3. Output all of these sketches stacked: \( \Pi B = \Pi_0 B_0 \oplus \ldots \oplus \Pi_{O(\log d)} B_{O(\log d)} \).

RefineSparsifier(\( \Pi B, \tilde{K}, \gamma, \varepsilon, c \)):

1. Compute \( \Pi_s B_s \tilde{K}^+ \) for each \( s \in \{0, 1, 2, \ldots, O(\log d)\} \).

2. For every edge \( e \) in the set of \( \binom{d}{2} \) possible edges:
   
   (a) Compute \( \tilde{\tau}_e = b^T_e \tilde{K}^+ b_e \) and \( p_e = c_2 \tilde{\tau}_e \log d \varepsilon^{-2} \), where \( c_2 \) is the oversampling constant from Lemma 3. Choose \( s \) such that \( \min\{1, p_e\} \leq \frac{1}{2s} \leq \min\{1, 2p_e\} \).
   
   (b) Compute \( \Pi_s x_e = \Pi_s B_s \tilde{K}^+ b_e \) and run the heavy hitters algorithm of Lemma 16. Determine whether or not \( x_e = 0 \) or \( x_e \geq \frac{c_1}{c_2 \sqrt{\log d}} \left\| B_s \tilde{K}^+ b_e \right\|_2 \) by checking whether the returned \( w_e > \tilde{\tau}_e / 2 \).
   
   (c) If it is determined that \( x_e(e) \neq 0 \) set \( W(e, e) = 2^s \).

3. Output \( \tilde{K}_e = B^T W B + \gamma I \).

We show that every required computation can be performed in the dynamic semi-
streaming model and then prove the correctness of the sampling procedure.

Implementation in the Semi-Streaming Model.

Assuming access to uniform hash functions, MaintainSketches requires total space \(O(d\ polylog\ d)\) and can be implemented in the dynamic streaming model. When an edge insertion comes in, use \(\{h_s\}\) to compute which \(B_s\)'s should contain the inserted edge, and update the corresponding sketches. For an edge deletion, simply update the sketches to add \(-b_e\) to each appropriate \(B_s\).

Unfortunately, storing \(O(\log\ d)\) uniform hash functions over \(\binom{d}{2}\) requires \(O(d^2\ \log\ d)\) space, and is thus impossible in the semi-streaming setting. If Section 4.8 we show how to cope with this issue by using a small-seed pseudorandom number generator.

Step 1 of RefineSparsifier can also be implemented in \(O(d\ polylog\ d)\) space. Since \(\tilde{K}\) has \(O(d\ \log\ d)\) non-zeros and \(\Pi_sB_s\) has \(O(\polylog\ d)\) rows, computing \(\Pi_sB_s\tilde{K}^+\) requires \(O(\polylog\ d)\) linear system solves in \(\tilde{K}\). We can use an iterative algorithm or a nearly linear time solver for symmetric diagonally dominant matrices to find solutions in \(O(d\ polylog\ d)\) space total.

For step 2(a), the \(s\) chosen to guarantee \(\min\{1, p_e\} \leq \frac{1}{2^s} \leq \min\{1, 2p_e\}\) could in theory be larger than the index of the last sketch \(\Pi_iB_i\) maintained. However, if we take \(O(\log\ d)\) samplings, our last will be empty with high probability. Accordingly, all samplings for higher values of \(s\) can be considered empty as well and we can just skip steps 2(b) and 2(c) for such values of \(s\). Thus, \(O(\log\ d)\) sampling levels are sufficient.

Finally, by our requirement that \(\tilde{K}\) is able to compute \(\frac{1}{c}\) factor leverage score approximations, with high probability, Step 2 samples at most \(O(d\ \log\ de^{-2})\) edges in total (in addition to selecting \(d\) identity edges). Thus, the procedure's output can be stored in small space.

Open Question 2. Is it possible to accelerate Step 2 of RefineSparsifier? Since we have to test against every possible graph edge \(b_e\), this step runs for \(O(d^2)\) iterations, dominating our sparsifier recovery time. Can we devise more efficient test vectors for detecting high effective resistance edges or otherwise devise an entirely different
recover procedure? It may be worth considering the design of fast sparse recovery algorithms for standard vector data. A nice goal would be recovery in $O(d \cdot \text{polylog}(d))$ time, which would match non-streaming algorithms up to polylogarithmic factors.

**Correctness**

To apply our sampling lemma, we need to show that, with high probability, the procedure `RefineSparsifier` independently samples each row of $B$ with probability $\hat{p}_e$ where $\min\{1, p_e\} \leq \hat{p}_e \leq \min\{1, 2p_e\}$. Since the algorithm samples the rows of $\sqrt{\gamma} \mathbf{I}$ with probability 1, and since $\tau_e \leq \bar{\tau}_e \leq \frac{1}{\epsilon} \tau_e$ for all $e$, by Lemma 3, with high probability, $\tilde{K}_e = B B^T + \gamma \mathbf{I} = \tilde{B}_e^T \tilde{B}_e + \gamma \mathbf{I}$ is a $(1 \pm \epsilon)$ spectral sparsifier for $K$. Furthermore, $\tilde{B}_e$ contains $O(r^2 c'd \log d)$ reweighted rows of $B$.

In `RefineSparsifier`, an edge is only included in $\tilde{K}_e$ if it is included in the $B_s(e)$ where

$$\min\{1, p_e\} \leq \frac{1}{2^{s(e)}} \leq \min\{1, 2p_e\}.$$  

The probability that $b_e$ is included in the sampled matrix $B_s(e)$ is simply $1/2^{s(e)}$, and sampling is done independently using uniform hash functions. So, we just need to show that, with high probability, any $b_e$ included in its respective $B_s(e)$ is recovered by Step 2(b).

Let $x_e = B \tilde{K}^+ b_e$ and $x_e^{s(e)} = B_{s(e)} \tilde{K}^+ b_e$. As explained in Section 4.3,

$$x_e^{s(e)}(e) = x_e(e) = 1_e B \tilde{K}^+ b_e = b_e^T \tilde{K}^+ b_e = \bar{\tau}_e.$$  

(4.1)

Furthermore, we can compute:

$$\|x_e\|^2 = b_e^T \tilde{K}^+ B^T B \tilde{K}^+ b_e$$

$$\leq b_e^T \tilde{K}^+ B^T B_{\gamma} \tilde{K}^+ b_e$$  

(Since $B^T B \preceq B_{\gamma}^T B_{\gamma}$)

$$\leq \frac{1}{c} \cdot b_e^T \tilde{K}^+ b_e$$  

(Since $c (B_{\gamma}^T B_{\gamma}) \preceq \tilde{K}$)

$$= \frac{1}{c} \bar{\tau}_e.$$  

(4.2)
Now, writing \( \tilde{\rho}_e = \frac{1}{2\epsilon(\sigma)} \), we expect \( \|x_e^{s(e)}(e)\|_2^2 \) to equal \( \tilde{\rho}_e \|x_e\|_2^2 = O(\tilde{\tau}_e^2 \log d \epsilon^{-2}) \). We want to argue that the norm falls close to this value with high probability. This follows from claiming that no entry in \( x_e \) is too large. For any edge \( e' \neq e \), define:

\[
\tilde{\tau}_{e',e} \overset{\text{def}}{=} x_e(e') = 1_e B \tilde{K}^+ b_e = b_e^T \tilde{K}^+ b_e.
\]

**Lemma 19.** \( \tilde{\tau}_{e',e} \leq \tilde{\tau}_e \)

**Proof.** Consider \( \tilde{v}_e = \tilde{K}^+ b_e \). Let \( e = (u_1, u_2) \) and \( e' = (u_1', u_2') \). If we have \(|\tilde{v}_e(u_1') - \tilde{v}_e(u_2')| \leq |\tilde{v}_e(u_1) - \tilde{v}_e(u_2)| \) then

\[
b_e^T \tilde{v}_e = b_e^T \tilde{K}^+ b_e \leq b_e^T \tilde{K}^+ b_e = b_e^T \tilde{v}_e,
\]

which implies \( \tilde{\tau}_{e',e} \leq \tilde{\tau}_e \) as desired.

Now, \( \tilde{K} \) is a weighted graph Laplacian added to a weighted identity matrix. Thus it is full rank and diagonally dominant. Since it has full rank, \( \tilde{K} \tilde{v}_e = \tilde{K} \tilde{K}^+ b_e = b_e \). Since \( \tilde{K} \) is diagonally dominant and since \( b_e \) is zero everywhere except at \( b_e(u_1) = 1 \) and \( b_e(u_2) = -1 \), it must be that \( \tilde{v}_e(u_1) \) is the maximum value of \( \tilde{v}_e \) and \( \tilde{v}_e(u_2) \) is the minimum value. So \( |\tilde{v}_e(u_1') - \tilde{v}_e(u_2')| \leq |\tilde{v}_e(u_1) - \tilde{v}_e(u_2)| \) and \( \tilde{\tau}_{e',e} \leq \tilde{\tau}_e \). \( \square \)

From Lemma 19, the vector \( \frac{1}{\tilde{\tau}_e} x_e \) has all entries (and thus all squared entries) in \([0, 1]\) so we can apply a Chernoff/Hoeffding bound to show concentration for \( \|\frac{1}{\tilde{\tau}_e} x_e^{s(e)}\|_2^2 \). Specifically, we use the standard multiplicative bound [Hoe63]:

\[
P(X > (1 + \delta) \mathbb{E} X) < e^{-2\delta^2 \mathbb{E} X}.
\] (4.3)

Since

\[
\mathbb{E} \|\frac{1}{\tilde{\tau}_e} x_e^{s(e)}\|_2^2 = \tilde{\rho}_e \cdot \frac{\tilde{\tau}_e}{c} \cdot \frac{1}{\tilde{\tau}_e^2} = \Theta(\log d \epsilon^{-2}),
\] (4.4)
we can set $\delta = \epsilon$ and conclude that
\[
P(\frac{1}{\bar{\tau}_e} |x_e^{(e)}|_2^2 > (1 + \epsilon) E \frac{1}{\bar{\tau}_e} |x_e^{(e)}|_2^2) = O(d^{-\Theta(1)}).
\]
Accordingly, $|x_e^{(e)}|_2^2 \leq c_3 \bar{\tau}_e^2 \log d e^{-2}$ with high probability for some constant $c_3$ and $\epsilon \leq 1$.

Now, if $x_e^{(e)}(e) = 0$, then our sparse recovery routine must return an estimated value for $x_e$ that is $\leq \eta |x_e^{(e)}|_2$. We set $\eta = \frac{\epsilon}{c_1 \sqrt{\log d}}$, so with high probability, the returned value is $< \frac{\epsilon}{c_1 \sqrt{\log d}} \sqrt{|x_e^{(e)}|_2^2} = \frac{\sqrt{\epsilon} \bar{\tau}_e}{c_1}$. On the other hand, if $x_e^{(e)}(e)$ is non-zero, it equals $\bar{\tau}_e$, so our sparse recovery sketch must return a value greater than $(1 - \frac{\sqrt{\epsilon}}{c_1}) \bar{\tau}_e$. Therefore, as long as we set $c_1$ high enough, we can distinguish between both cases by simply checking whether or not the return value is $> \bar{\tau}_e / 2$, as described for Step 2.

Thus, as long as $|x_e^{(e)}|_2^2$ concentrates as described, our procedure recovers $e$ if and only if $b_e$ is included in $B_{s(e)}$. As explained, this ensures that our process is exactly equivalent to independent sampling. Since concentration holds with probability $O(d^{-\Theta(1)})$, we can adjust constants and union bound over all $\binom{d}{2}$ possible edges to claim that our algorithm returns the desired $\tilde{K}_e$ with high probability.

### 4.6 Sparsification of Weighted Graphs

We can use a standard technique to extend our result to streams of weighted graphs in which an edge’s weight is specified at deletion, matching what is known for cut sparsifiers in the dynamic streaming model [AGM12b, GKP12]. Assume that all edge weights and the desired approximation factor $\epsilon$ are polynomial in $d$, then we can consider the binary representation of each edge’s weight out to $O(\log d)$ bits. For each bit of precision, we maintain a separate unweighted graph $G_0, G_1, ... G_{O(\log d)}$. We add each edge to the graphs corresponding to bits with value one in its binary representation. When an edge is deleted, its weight is specified, so we can delete it from these same graphs. Since $G = \sum_i 2^i \cdot G_i$, given a $(1 \pm \epsilon)$ sparsifier $\tilde{K}_i$ for each
we have:

\[(1 - \epsilon) \sum_i 2^i \cdot K_i \preceq \sum_i 2^i \cdot \tilde{K}_i \preceq (1 + \epsilon) \sum_i 2^i \cdot K_i \]

\[(1 - \epsilon) K \preceq \sum_i 2^i \cdot \tilde{K}_i \preceq (1 + \epsilon) K.\]

So \(\sum_i 2^i \cdot \tilde{K}_i\) is a spectral sparsifier for \(K\), the Laplacian of the weighted graph \(G\).

### 4.7 Sparsification of Structured Matrices

Next, we extend our algorithm to sparsify certain general quadratic forms in addition to graph Laplacians. There were only three places in our analysis where we used that \(B\) was not an arbitrary matrix. First, we needed that \(B = SB_d\), where \(B_d\) is the vertex-edge incidence matrix of the unweighted complete graph on \(d\) vertices. In other words, we assumed that we had some dictionary matrix \(B_d\) whose rows encompass every possible row that could arrive in the data stream. In addition to this dictionary assumption, we needed \(B\) to be sparse and to have a bounded condition number in order to achieve our small space results. These conditions allow our compression to avoid an \(\Omega(d^2)\) lower bound for approximately solving regression on general \(\mathbb{R}^{n \times d}\) matrices in the streaming model [CW09].

As such, to handle the general 'structured matrix' case, we assume that we have some dictionary \(A \in \mathbb{R}^{n \times d}\) containing \(n\) rows \(a_i \in \mathbb{R}^d\). We assume that \(n = O(\text{poly}(d))\). In the dynamic streaming model we receive insertions and deletions of rows from \(A\) resulting in a matrix \(A = SA\) where \(S \in \mathbb{R}^{n \times n}\) is a diagonal matrix such that \(S_{ii} \in \{0, 1\}\) for all \(i \in [n]\). From an \(O(d \text{polylog}(n))\) space compression of \(A\), our goal is to recover a diagonal matrix \(W\) with at most \(O(d \log d)\) nonzero entries such that \(A^T W^2 A \approx A^T S^2 A = A^T A\). Formally, we prove the following:

**Theorem 20 (Streaming Structured Matrix Sparsification).** Given a row dictionary \(A \in \mathbb{R}^{n \times d}\) containing all possible rows of the matrix \(A\), there exists an algorithm that, for any \(\epsilon > 0\), processes a stream of row insertions and deletions for \(A\) in a single
pass and maintains a set of linear sketches of this input in $O\left(\frac{1}{\epsilon^2}d\log(n, \kappa_u)\right)$ space where $\kappa_u$ is an upper bound on the condition number of $A^T A$. From these sketches, it is possible to recover, with high probability, a matrix $\tilde{A}^T \tilde{A}$ such that $\tilde{A}$ contains only $O(\epsilon^{-2}d \log d)$ reweighted rows of $A$ and $\tilde{A}^T \tilde{A}$ is a $(1 \pm \epsilon)$ spectral sparsifier of $A^T A$. The algorithm recovers $\tilde{A}$ in $\text{poly}(n, \epsilon, d, \log \kappa_u)$ time.

Note that, when $n, \kappa_u = O(\text{poly}(d))$, the sketch space is $O\left(\frac{1}{\epsilon^2}d \text{polylog}(d)\right)$. To prove Theorem 20, we need to introduce a more complicated sampling procedure than what was used for the graph case. In Lemma 19, for the correctness proof of RefineSparsifier in Section 4.5, we relied on the structure of our graph Laplacian and vertex-edge incidence matrix to show that $\tau_{e',e} \leq \tau_e$. This allowed us to show that the norm of a sampled $x_{e(e)}$ concentrates around its mean. Thus, we could recover edge $e$ with high probability if it was in fact included in the sampling $B_{s(e)}$. Unfortunately, when processing general matrices, $\tilde{\tau}_e$ is not necessarily the largest element $x_{e(e)}$ and the concentration argument fails.

We overcome this problem by modifying our algorithm to compute more sketches. Rather than computing a single $\Pi A_s$, for every sampling rate $1/2^s$, we compute $O(\log d)$ sketches of different samplings of $A$ at rate $1/2^s$. Each sampling is fully independent from all others, including those at the same and different rates. This differs from the graph case, where $B_{1/2^{s+1}}$ was always a subsampling of $B_{1/2^s}$ (for ease of exposition). Our modified set up lets us show that, with high probability, the norm of $x_{e,s(i)}$ is close to its expectation for at least a $(1 - \epsilon)$ fraction of the independent samplings for rate $s(i)$. We can recover row $i$ if it is present in one of the 'good' samplings.

Ultimately, we argue, in a similar manner to [KP12], that we can sample rows according to some distribution that is close to the distribution obtained by independently sampling rows according to leverage score. Using this primitive, we can proceed as in the previous sections to prove Theorem 20. In Section 4.7.1, we provide the row sampling subroutine and in Section 4.7.2, we show how to use this sampling routine to prove Theorem 20.
4.7.1 Generalized Row Sampling

Our leverage score sampling algorithm for the streaming model is as follows:

**Streaming Row Sampling Algorithm**

MaintainMatrixSketches$(A, \epsilon, \kappa_u, \gamma, c)$:

1. Let $S = O(\log \kappa_u)$, $T = O(\log n)$, and for all $s \in [S]$ and $t \in [T]$ let $F_s^{(t)} \in \mathbb{R}^{n \times n}$ be a diagonal matrix with $[F_s^{(t)}]_{ii} = 1$ independently with probability $\frac{1}{2^s}$ and is 0 otherwise.$^2$

2. For all $s \in [S]$ and $t \in [T]$ maintain sketch $\Pi_s^{(t)} F_s^{(t)} A$ where each $\Pi_s^{(t)}$ is drawn independently from the distribution in Lemma 16 with $\gamma^2 = \frac{1}{c}$ and $C = c_1 \epsilon^{-3} \log n \log d$.

3. Add rows of $\gamma I$, independently sampled at rate $\frac{1}{2^s}$, to each sketch.

RowSampleMatrix$(\Pi A, \tilde{K}, \epsilon, c)$:

1. For all $s \in [S]$ and $t \in [T]$ let $x_s^{(t)} = F_s^{(t)} A \tilde{K}^+$ and compute $\Pi_s^{(t)} x_s^{(t)}$.

2. For every $i \in [n]$:

   (a) Compute $\tilde{r}_i = a_i^T \tilde{K}^+ a_i$ and $p_i = c_2 \tilde{r}_i \log d \epsilon^{-2}$, where $c_2$ is the oversampling constant from Lemma 3. Choose $s_i$ such that $\min\{1, p_i\} \leq \frac{1}{2^s} \leq \min\{1, 2p_i\}$.

   (b) Pick $t_i \in [T]$ uniformly at random and use Lemma 16 to check if $x_s^{(t_i)}(i)^2 \geq C^{-1} \|x_s^{(t_i)}\|_2^2$.

   (c) If $i$ is recovered, add row $i$ to the set of sampled edges with weight $2^{s_i}$.

We claim that, with high probability, the set of edges returned by the above algorithm is a random variable that is stochastically dominated by the two random variables obtained by sampling edges independently at rates $p_i$ and $(1 - \epsilon)p_i$, respec-
tively. The following property of PSD matrices is used in our proof of correctness:

**Lemma 21.** For any symmetric PSD matrix $K \in \mathbb{R}^{d \times d}$ and indices $i, j \in [d]$ we have

$$|K_{ij}| \leq \frac{1}{2} (K_{ii} + K_{jj}).$$

**Proof.** Let $1_i$ be the vector with a 1 at position $i$ and 0s else where. For all $i, j \in [d]$ by the fact that $K$ is PSD we have that

$$(1_i - 1_j)^T K (1_i - 1_j) \geq 0 \quad \text{and} \quad (1_i + 1_j)^T K (1_i + 1_j) \geq 0.$$

Expanding, we have that:

$$-K_{ii} - K_{jj} \leq 2K_{ij} \leq K_{ii} + K_{jj},$$

yielding the result. \qed

We can now proceed to prove that our sampling procedure approximates sampling the rows of $A$ by their leverage scores.

**Lemma 22.** Consider an execution of $\text{RowSampleMatrix}(\Pi A, \tilde{K}, c, \epsilon)$ where

- $cA^T A \preceq \tilde{K} \preceq A^T A$ for $c \in (0, 1]$, and
- $\epsilon \in (0, 1].$

Let $D$ be a random variable for the indices returned by $\text{RowSampleMatrix}(\Pi A, \tilde{K}, c, \epsilon)$. Let $I \subseteq [n]$ denote the indices of the nonzero rows of $A$ and let $D_r$ and $D_q$ be random variables for the subset of $[n]$ obtained by including each $i \in I$ independently with probability $r_i = (1 - \epsilon) \frac{1}{2s}$ and $q_i = \frac{1}{2s}$.

With high probability, i.e. except for a $(1 - \frac{1}{n^{\Omega(1)}}$ fraction of the probability space, $D$ is stochastically dominated by $D_q$ and $D$ stochastically dominates $D_r$ with respect to set inclusion.
Proof. By definition, \( \mathcal{D}_r \) and \( \mathcal{D}_q \) are always subsets of \( \mathcal{I} \) and \( \mathcal{D} \) is a subset of \( \mathcal{I} \) with high probability (it is a subset as long as the algorithm of Lemma 16 succeeds). Thus it remains to show that, with high probability for each \( \mathcal{J} \subseteq \mathcal{I} \),

\[
\prod_{i \in \mathcal{J}} r_i = \mathbb{P}[\mathcal{J} \subseteq \mathcal{D}_r] \leq \mathbb{P}[\mathcal{J} \subseteq \mathcal{D}] \leq \mathbb{P}[\mathcal{J} \subseteq \mathcal{D}_q] = \prod_{i \in \mathcal{J}} q_i.
\]

Furthermore, by definition, with high probability, \texttt{RowSampleMatrix} outputs \( i \in \mathcal{I} \) if and only if \( x_{s_l}^{(t_j)}(i)^2 \geq C^{-1}\|x_{s_l}^{(t_j)}\|_2^2 \) and consequently

\[
\mathbb{P}[\mathcal{J} \subseteq \mathcal{D}] = \mathbb{P}\left[ \forall i \in \mathcal{J} : x_{s_l}^{(t_j)}(i)^2 \geq C^{-1}\|x_{s_l}^{(t_j)}\|_2^2 \right]. \tag{4.5}
\]

As shown in Equation (4.1), when proving our graph sampling Lemma, for all \( i \in \mathcal{J} \),

\[x_{s_l}^{(t_j)}(i) = [\mathbf{F}_{s_l}^{(t_j)}]_{ii} \cdot \bar{r}_i.\]

Consequently, by the definition of \( [\mathbf{F}_{s_l}^{(t_j)}]_{ii} \) we can rewrite (4.5) as:

\[
\mathbb{P}[\mathcal{J} \subseteq \mathcal{D}] = \mathbb{P}\left[ \forall i \in \mathcal{J} : \|x_{s_l}^{(t_j)}\|_2^2 \leq C \cdot \bar{r}_i^2 \text{ and } [\mathbf{F}_{s_l}^{(t_j)}]_{ii} = 1 \right]. \tag{4.6}
\]

From (4.6) and the independence of \( [\mathbf{F}_{s_l}^{(t_j)}]_{ii} \) we obtain the following trivial upper bound on \( \mathbb{P}[\mathcal{J} \subseteq \mathcal{D}] \),

\[
\mathbb{P}[\mathcal{J} \subseteq \mathcal{D}] \leq \mathbb{P}\left[ \forall i \in \mathcal{J} : [\mathbf{F}_{s_l}^{(t_j)}]_{ii} = 1 \right] = \prod_{i \in \mathcal{J}} \frac{1}{2^q_i} = \prod_{i \in \mathcal{J}} q_i
\]

and consequently \( \mathcal{D} \) is stochastically dominated by \( \mathcal{D}_q \) as desired.

As shown in Equation 4.2, when proving the graph sampling case, for all \( i \in \mathcal{I} \) and \( t \in [I] \)

\[p_i \bar{r}_i \leq \mathbb{E}\left[\|x_{s_l}^{(t)}\|_2^2\right] \leq c \frac{2}{c} p_i \bar{r}_i. \tag{4.7}
\]

Recalling that \( p_i = c_2 \bar{r}_i \log d c^{-2} \), combining (4.6) and (4.7) yields:

\[
\mathbb{P}[\mathcal{J} \subseteq \mathcal{D}] \geq \mathbb{P}\left[ \forall i \in \mathcal{J} : \|x_{s_l}^{(t_j)}\|_2^2 \leq c_3 \log m e^{-1} \mathbb{E}[\|x_{s_l}^{(t_j)}\|_2^2] \text{ and } [\mathbf{F}_{s_l}^{(t_j)}]_{ii} = 1 \right], \tag{4.8}
\]

78
where \( c_3 = c_1 c / 2c_2 \).

To bound the probability that \( \|x_i^{(t_i)}\|_2^2 \leq c_3 \log n \varepsilon^{-1} \cdot \mathbb{E}[\|x_i^{(t_i)}\|_2^2] \) we break the contribution to \( \|x_i^{(t_i)}\|_2^2 \) for each \( t \) into two parts. For all \( i \) we let \( \mathcal{K}_i = \{ j \in \mathcal{I} | s_j = s_i \} \), i.e. the set of all rows \( j \) which we attempt to recover at the same sampling rate as \( i \). For any \( t \in [T] \), we let \( A_i^{(t)} = \sum_{j \in \mathcal{K}_i} x_i^{(t)}(j)^2 \) and \( B_i^{(t)} = \sum_{j \in \mathcal{I} - \mathcal{K}_i} x_i^{(t)}(j)^2 \). Using this notation and (4.8) we obtain the following lower bound

\[
\mathbb{P}[J \subseteq D] \geq \mathbb{P}\left[ \forall i \in J : A_i^{(4i)} \leq \frac{c_3 \log d \varepsilon^{-1}}{2} \mathbb{E}[\|x_i^{(t_i)}\|_2^2], B_i^{(t_i)} \leq \frac{c_3 \log d \varepsilon^{-1}}{2} \mathbb{E}[\|x_i^{(t_i)}\|_2^2], \right.
\text{and} \left. [F_{s_i}^{(t_i)}]_{ii} = 1 \right].
\]

For all \( j \in \mathcal{K}_i \), the rows that we attempt to recover at the same rate as row \( i \), we know that \( \tilde{\tau}_j \leq 2\tilde{\tau}_i \). By Lemma 21 we know that for all \( i \in \mathcal{I} \) with \( s_i \geq 1 \) and \( j \in \mathcal{K}_i \)

\[
x_i^{(t_i)}(j)^2 = [F_{s_i}^{(t_i)}]_{jj} |a_i^T \tilde{K} a_j|^2 \leq \left( \frac{\tilde{\tau}_i + 2\tilde{\tau}_j}{2} \right)^2 \leq \left( \frac{\tilde{\tau}_i + 2\tilde{\tau}_i}{2} \right)^2 \leq 3c_2^{-1} \varepsilon^2 \log^{-1} d \mathbb{E}[\|x_i^{(t_i)}\|_2^2].
\]

(4.9)

Now recall that \( C = \Omega(\varepsilon^{-2} \log d) \). If \( \tilde{\tau}_i > 1/2 \) and therefore \( s_i = 0 \) then \( x_i^{(t_i)}(i)^2 = \tilde{\tau}_i^2 \) and setting constants high enough and considering (4.7), we see that row \( i \) is output with high probability. On the other hand if \( s_i \geq 1 \), then by (4.9) and Chernoff bound choosing a sufficiently large constant we can ensure that with high probability \( A_i^{(t)} \leq \frac{c_1 \log n \varepsilon^{-1}}{2} \mathbb{E}[\|x_i^{(t)}\|_2^2] \) for all \( i \) and \( t \).

Further, by (4.7) and a Markov bound, \( \mathbb{P}\left[ B_i^{(t_i)} > \frac{c_3 \log n \varepsilon^{-1}}{2} \mathbb{E}[\|x_i^{(t)}\|_2^2] \right] \leq \frac{c}{\mathbb{O}(\log n)} \).

Therefore, by Chernoff bound, with high probability for each \( i \in J \) with \( s_i \geq 1 \) for at least a \( 1 - \varepsilon \) fraction of the values of \( t \in T \) we have \( B_i^{(t_i)} \leq \frac{c_3 \log n \varepsilon^{-1}}{2} \mathbb{E}[\|x_i^{(t)}\|_2^2] \).

However, note that by construction all the \( B_i^{(t_i)} \) are mutually independent of the \( A_i^{(t_i)} \) and the values of \( [F_{s_i}^{(t)}]_{jj} \) for \( j \in \mathcal{K}_i \). So, \textit{RowSampleMatrix} is simply picking each row \( i \) with probability \( \frac{1}{2\tilde{\tau}_i} \) (failing with only a \( \frac{1}{n c(\varepsilon)} \) probability) or not being able to recover each edge independently with some probability at most \( \varepsilon \) – the probability that \( B_i^{(t_i)} \) is too large. Consequently, except for a negligible fraction of the probability
space we have that

$$\mathbb{P} \left[ J \subseteq D \right] \geq \prod_{i \in J} (1 - \epsilon) \cdot |F_{i, i}| = \prod_{i \in J} \frac{1 - \epsilon}{2^{s_i}} = \prod_{i \in J} r_i,$$

giving the desired result. \(\square\)

### 4.7.2 Generalized Recursive Sparsification

Next we show how to construct a spectral sparsifier in the streaming model for a general structured matrix using the row sampling subroutine, RowSampleMatrix. In the graph case, Theorem 15 shows that, if we can find a sparsifier to a graph \(G\) using a coarse sparsifier, then we can use the chain of spectrally similar graphs provided in Theorem 17 to find a final \((1 \pm \epsilon)\) sparsifier for our input graph.

The proof of Theorem 15 includes our third reliance on the fact that we are sparsifying graphs – we claim that the condition number of an unweighted graph is polynomial in \(d\). This fact does not hold in the general matrix case since the condition number can be exponentially large even for bounded integer matrices. Therefore, our result for general matrix depends on the condition number of \(A\).

**Theorem 23.** Given a row dictionary \(A \in \mathbb{R}^{m \times n}\). Let \(A = SA\) be the matrix specified by an insertion-deletion stream where \(S \in \mathbb{R}^{n \times n}\) is a diagonal matrix such that \(S_{ii} \in \{0, 1\}\) for all \(i \in [n]\). Let \(\kappa_u\) be a given upper bound on the possible condition number of any \(A\). Let \(\gamma\) be a fixed parameter and consider \(K = A^T A + \gamma I\). For any \(\epsilon > 0\), there exists a sketching procedure \(\text{MaintainMatrixSketches}(A, \epsilon, \gamma, \kappa_u, c)\) that outputs an \(O(d \text{polylog}(n, \kappa_u))\) sized sketch \(IIA\). There exists a corresponding recovery algorithm \(\text{RefineMatrixSparsifier}\) such that if \(cK \leq \bar{K} \leq K\) for some \(0 < c < 1\) then:

\(\text{RefineMatrixSparsifier}(IIA, \bar{K}, \epsilon, c)\) returns, with high probability, \(\tilde{K}_e = \tilde{A}_e^T \tilde{A}_e + \gamma I\), where \((1 - \epsilon)K \leq \tilde{K}_e \leq (1 + \epsilon)K\), and \(\tilde{A}_e\) contains only \(O(\epsilon^{-2}d \log d)\) reweighted rows of \(A\) with high probability.
Proof. As in the graph case, we can think of the identity $\gamma I$ as a set of rows that we sample with probability 1. Hence, we have $\hat{K}_e = \hat{A}_e^+\hat{A}_e + \gamma I$.

Lemma 22 shows that RowSampleMatrix($\Pi A, \hat{K}, c, \epsilon$) returns a random set of indices of $A$ such that the generated random variable is dominated by $D_q$ and is stochastically dominates $D_r$. Recall that $D_r$ and $D_q$ are random variables for the subset of $[n]$ obtained by including each $i \in I$ independently with probability $r_i = (1 - \epsilon)\frac{1}{2^s_i}$ and $q_i = \frac{1}{2^s_i}$.

Since $\frac{1}{2^s_i}$ is a constant factor approximation of leverages score, Lemma 3 shows that sampling and reweighing the rows according to $D_r$ gives a spectral sparsifier of $K$ with the guarantee required. Similarly, sampling according to $D_q$ gives a sparsifier. Since the indices returned by RowSampleMatrix($A, \hat{K}, c, \epsilon$) are sandwiched between two processes which each give spectral sparsifiers, sampling according to RowSampleMatrix gives the required spectral sparsifier [KP12].

Using RefineMatrixSparsifier, the arguments in Theorem 15 yield Theorem 20. Our sketch size needs to be based on $\log \kappa_u$ for two reasons - we must subsample the matrix at $O(\log \kappa_u)$ different rates as our leverage scores will be lower bounded by $\poly(\kappa_u)$. Furthermore the chain of recursive sparsifiers presented in Theorem 17 will have length $\log \kappa_u$. Recovery will run in time $\poly(n, d, \epsilon, \log \kappa_u)$. Space usage will depend on the sparsity of the rows in $A$ as we will need enough space to solve linear systems in $\hat{K}$. In the worst case, this could require $\tilde{O}(d^2)$ space. However, it may be possible to generate rows from $A$ on-the-fly based on a given index, in which case we can apply an system iterative solver, performing the required matrix vector products one row at a time and thus using at most $\tilde{O}(d)$ space. Alternatively, as in the case for graph vertex-edge incidence matrices, constant row sparsity in $A$ is sufficient for $\tilde{O}(d)$ space system solves.

Open Question 3. Is it possible to eliminate our space dependence on $\kappa_u$? Not only is the condition number difficult to bound for general matrices, but an approach that avoids $\kappa_u$ could extend to weighted graphs, avoiding the implicit dependence on
log(weight_{max}/weight_{min}) that arises from the technique applied in Section 4.6.

A starting point may be to consider whether or not we can at least eliminate the dependence that arises from the length of the recursive sparsifier chain in Theorem 17. Perhaps we could avoid [MP12]'s "Introduction and Removal of Artificial Bases" technique by using a recursive sparsification procedure more similar to those described in Chapter 3, which do not involve condition number dependencies.

4.8 Pseudorandom Number Generation

In the proof of our sketching algorithm, Theorem 18, we assume that the procedure MaintainSketches has access to $O(\log d)$ uniform random hash functions, $h_1, \ldots, h_{O(\log d)}$ mapping every edge to $\{0, 1\}$. These functions are used to subsample our vertex edge incidence matrix, $B$, at geometrically decreasing rates. Storing the functions as described would require $O(d^2 \log d)$ space - we need $O(\log d)$ random bits for each of the possible $\binom{d}{2}$ edges in $G$.

To achieve $O(d \text{polylog}(d))$ space, we need to compress the hash functions using a pseudorandom number generator (PRG). We will apply Nisan's popular PRG for small space computation. Our approach follows an argument in [AGM12b] (Section 3.4) that was originally introduced in [Ind06] (Section 3.3). First, we summarize the pseudorandom number generator from [Nis92]

**Theorem 24** (Corollary 1 in [Nis92]). *Any randomized algorithm running in space($S$) and using $R$ random bits may be converted to one that uses only $O(S \log R)$ random bits (and runs in space $O(S \log R)$).*

[Nis92] gives this conversion explicitly by describing a method for generating $R$ pseudorandom bits from $O(S \log R)$ truly random bits. For any algorithm running in space($S$), the pseudorandom bits are "good enough" in that the output distribution of the algorithm under pseudorandom bits is very close to the output distribution under truly random bits. In particular, the total variation distance between the distributions is at worst $2^{-O(S)}$ (see Lemma 3 in [Nis92]). It follows that using pseudorandom bits
increases the failure probability of any randomized algorithm by just $2^{-O(S)}$ in the worst case.

As described, our algorithm runs in $O(d^2 \log d)$ space and it is not immediately obvious how to use Theorem 24 to reduce this requirement. However, consider the following: suppose our algorithm is used on a sorted edge stream where all insertions and deletions for a single edge come in consecutively. In this case, at any given time, we only need to store one random bit for each hash function, which requires just $O(\log d)$ space. The random bits can be discarded after moving on to the next edge. Thus, the entire algorithm can run in $O(d \text{polylog}(d))$ space. Then, we can apply Theorem 24, using the pseudorandom generator to get all of our required random bits by expanding just $S \log R = O(d \text{polylog}(d)) \cdot O(\log(d^2 \log d)) = O(d \text{polylog}(d))$ truly random bits. Since our failure probability increases by at most $1/2^{O(d \text{polylog}(d))}$, we still only fail with probability inverse polynomial in $d$.

Now notice that, since our algorithm is sketch based, edge updates simply require an addition to or subtraction from a sketch matrix. These operations commute, so our output will not differ if we reorder the insertion/deletion stream. Thus, we can run our algorithm on a general edge stream, using the pseudorandom number generator to generate any of the required $O(d^2 \log d)$ bits as they are needed and operating in only $O(d \text{polylog}(d))$ space.

Each time an edge is streamed in, we need to generate $\log d$ random bits from the pseudorandom generator. This can be done in $\log(R) \cdot S = O(d \text{polylog}(d))$ time [Ind06], which dominates the runtime required to process each streaming update.

Finally, Section 4.7 uses a slightly different sampling scheme for general structured matrices. Instead of building a sequence of subsampled matrices, the row dictionary is sampled independently at each level. In total, the required number of random bits is $O(n \log^2 d)$, where $n$ is the number of rows in the dictionary $\mathcal{A}$. We require that $n = \text{poly}(d)$, in which case the arguments above apply unmodified for the general matrix case.

**Note 4.** Jelani Nelson pointed out that it is possible to derandomized Lemma 3, our main leverage score sampling routine, using low independence hash functions, which
eliminates the need for a PRG. We plan to include a proof of the derandomized lemma in the journal version of [KLM+14].

Alternatively, in currently unpublished work with Michael Forbes and Cameron Musco, we show that it is actually possible to apply a small space PRG that generates pseudorandom bits faster than Nisan's PRG at the cost of an acceptable sacrifice in seed size and failure probability.
Appendix A

Properties of Leverage Scores

A.1 Spectral Approximation via
Leverage Score Sampling

Here we prove Lemma 3, which states that it is possible to obtain a spectral approximation by sampling rows from $A$ independently with probabilities proportional to leverage score overestimates.

**Lemma 3** (Spectral Approximation via Leverage Score Sampling). Given an error parameter $0 < \epsilon < 1$, let $u$ be a vector of leverage score overestimates, i.e., $\tau_i(A) \leq u_i$ for all $i$. Let $\alpha$ be a sampling rate parameter and let $c$ be a fixed positive constant. For each row, we define a sampling probability $p_i = \min\{1, \alpha \cdot u_i \cdot c \log d\}$. Furthermore, we define a function $\text{Sample}(u, \alpha)$, which returns a random diagonal matrix $S$ with independently chosen entries. $S_{ii} = \frac{1}{\sqrt{p_i}}$ with probability $p_i$ and 0 otherwise.

Setting $\alpha = \epsilon^{-2}$, $S$ has at most $\sum_i \min\{1, \alpha \cdot u_i \cdot c \log d\} \leq \alpha c \log d \|u\|_1$ non-zero entries and $\frac{1}{\sqrt{1+\epsilon}} S A$ is a $\frac{1+\epsilon}{1-\epsilon}$-spectral approximation for $A$ with probability at least $1 - d^{-e/3}$.

We rely on the following matrix concentration result, which is a variant of Corollary 5.2 from [Tro12], given by Harvey in [Har12]:

85
Lemma 25. Let $Y_1\ldots Y_k$ be independent random positive semidefinite matrices of size $d \times d$. Let $Y = \sum_i Y_i$ and let $Z = \mathbb{E}[Y]$. If $Y_i \preceq R \cdot Z$ then

$$P \left[ \sum_i^k Y_i \preceq (1 - \epsilon)Z \right] \leq d \varepsilon^2,$$

and

$$P \left[ \sum_i^k Y_i \preceq (1 + \epsilon)Z \right] \leq d \varepsilon^2.$$

Proof of Lemma 3. For each row $a_i$ of $A$ choose $Y_i = \frac{a_i a_i^T}{p_i}$ with probability $p_i$, and 0 otherwise. So $(SA)^T(SA) = \sum_i Y_i$. Note that $Z = \mathbb{E}[\sum_i Y_i] = \sum_i \mathbb{E}[Y_i] = \sum_i a_i a_i^T = A^T A$, as desired. To apply Lemma 25, we will want to show that

$$\forall i, Y_i \preceq \frac{1}{c \log d \varepsilon^{-2}} \cdot A^T A. \quad (A.1)$$

First, consider when $p_i < 1$. The $p_i = 1$ case is slightly less direct, and we will deal with it shortly. $\alpha = \varepsilon^{-2} \geq 1$, so $p_i < 1$ implies that $\tau_i(A) \leq u_i \leq \frac{1}{c \log d}$. We have

$$Y_i \preceq \frac{a_i a_i^T}{u_i \cdot c \log d \varepsilon^{-2}} \preceq \frac{a_i a_i^T}{\tau_i(A) \cdot c \log d \varepsilon^{-2}} \preceq \frac{1}{c \log d \varepsilon^{-2}} \cdot A^T A, \quad (A.2)$$

since

$$\frac{a_i a_i^T}{\tau_i(A)} \preceq A^T A. \quad (A.3)$$

Equation (A.3) is proved by showing that, for all $x \in \mathbb{R}^d$, $x^T a_i a_i^T x \leq \tau_i(A) \cdot x^T A^T A x$.

We can assume without loss of generality that $x$ is in the column space of $A^T A$ since, letting $x'$ be the component of $x$ in the null space of $(A^T A)$, $0 = x^T (A^T A) x' = x'^T a_i a_i^T x'$. Now, with $x$ in the column space, for some $y$ we can write $x = (A^T A)^{+} y$ where $(A^T A)^{+} = V \Sigma^{-1} V^T$ (recall that $(A^T A)^{+} = V \Sigma^{-2} V^T$ if the SVD of $A$ is
given by \( A = U\Sigma V^T \). So now we consider

\[
y^T (A^T A)^{+/2} a_i a_i^T (A^T A)^{+/2} y.
\]

\((A^T A)^{+/2} a_i a_i^T (A^T A)^{+/2}\) is rank 1, so its maximum (only) eigenvalue is equal to its trace. By the cyclic property, \( \text{tr}((A^T A)^{+/2} a_i a_i^T (A^T A)^{+/2}) = \text{tr}(a_i^T (A^T A)^{+/2} a_i) = \tau_i(A) \). Furthermore, the matrix is positive semidefinite, so

\[
y^T (A^T A)^{+/2} a_i a_i^T (A^T A)^{+/2} y \leq \tau_i(A) \|y\|^2_2 = \tau_i(A) \cdot y^T (A^T A)^{+/2} (A^T A)(A^T A)^{+/2} y,
\]

which gives us (A.3) and thus (A.1) when \( p_i < 1 \).

Equation (A.1) does not hold directly when \( p_i = 1 \). In this case, \( Y_i = a_i a_i^T \) with probability 1. However, selecting \( Y_i \) is exactly the same as selecting and summing \( c \log d e^{-2} \) random variables \( Y_i^{(1)}, ..., Y_i^{(c \log d e^{-2})} \), each equal to \( \frac{1}{c \log d e^{-2}} \cdot a_i a_i^T \) with probability 1, and thus clearly satisfying

\[
Y^{(j)} \preceq \frac{1}{c \log d e^{-2}} \cdot A^T A.
\]  

(A.4)

We can symbolically replace \( Y_i \) in our Lemma 25 sums with these smaller random variables, which does not change \( Z = E[Y] \), but proves concentration. We conclude that:

\[
(1 - \epsilon) \cdot A^T A \preceq \sum_i Y_i \preceq (1 + \epsilon) \cdot A^T A
\]

with probability at least

\[
1 - d e^{-c \log d e^{-2}/3} \geq 1 - d^{1-c/3}.
\]

As noted, \( \tilde{A}^T \tilde{A} = A^T S^2 A = \sum_i Y_i \), so this gives us that \( \frac{1}{\sqrt{1+\epsilon}} S A \) is a \( \frac{1+\epsilon}{1-\epsilon} \) spectral approximation to \( A \) with high probability. Furthermore, by a standard Chernoff bound, \( S \) has \( \sum_i \min\{1, u_i \cdot a \log d\} \leq a \log d \|u\|_1 \) nonzero entries with
A.2 Rank 1 Updates

Here we prove Lemma 7, making critical use of the Sherman-Morrison formula for the Moore-Penrose pseudoinverse [Mey73, Thm 3].

Lemma 7 (Leverage Score Changes Under Rank 1 Updates). Given any $A \in \mathbb{R}^{n \times d}$, $\gamma \in (0, 1)$, and $i \in [n]$, let $W$ be a diagonal matrix such that $W_{ii} = \sqrt{1 - \gamma}$ and $W_{jj} = 1$ for all $j \neq i$. Then, we have

$$\tau_i(WA) = \frac{(1 - \gamma)\tau_i(A)}{1 - \gamma \tau_i(A)} \leq \tau_i(A),$$

and for all $j \neq i$,

$$\tau_j(WA) = \tau_j(A) + \frac{\gamma \tau_{ij}(A)^2}{1 - \gamma \tau_i(A)} \geq \tau_j(A).$$

Proof.

$$\tau_i(WA) = 1_iW(A^TWA)^+A^TW^1_i$$  \hspace{1cm} \text{(definition of leverage scores)}

$$= (1 - \gamma) a_i^T (A^T A - \gamma a_i a_i^T)^+ a_i$$  \hspace{1cm} \text{(definition of W)}

$$= (1 - \gamma) a_i^T \left( (A^T A)^+ + \frac{\gamma (A^T A)^+ a_i a_i^T (A^T A)^+}{1 - \gamma a_i^T (A^T A)^+ a_i} \right) a_i$$  \hspace{1cm} \text{(Sherman-Morrison formula)}

$$= (1 - \gamma) \left( \tau_i(A) + \frac{\gamma \tau_{ii}(A)^2}{1 - \gamma \tau_i(A)} \right)$$

$$= \frac{(1 - \gamma)\tau_i(A)}{1 - \gamma \tau_i(A)}$$

$$\leq \tau_i(A).$$
Similarly,
\[
\tau_j(WA) = a_j^\top (A^\top A - \gamma a_i a_i^\top)^+ a_j \\
= a_j^\top \left((A^\top A)^+ + \frac{\gamma (A^\top A)^+ a_i a_i^\top (A^\top A)^+}{1 - \gamma a_i^\top (A^\top A)^+ a_i}\right) a_j \\
= \tau_j(A) + \frac{\gamma \tau_{ij}(A)^2}{1 - \gamma \tau_i(A)} \\
\geq \tau_j(A).
\]

\[\square\]

### A.3 Lower Semi-continuity of Leverage Scores

Here we prove Lemma 8 by providing a fairly general inequality, Lemma 26, for relating leverage scores under one set of weights to leverage scores under another.

**Lemma 8** (Leverage Scores are Lower Semi-continuous). \(\tau(WA)\) is lower semi-continuous in the diagonal matrix \(W\), i.e. for any sequence \(W(k) \to W\) with \(W_{ii}^{(k)} \geq 0\) for all \(k\) and \(i\), we have

\[
\tau_i(WA) \leq \liminf_{k \to \infty} \tau_i(W^{(k)}A). \quad (A.5)
\]

**Lemma 26** (Comparing Leverage Scores). Let \(W, \overline{W} \in \mathbb{R}^{n \times n}\) be non-negative diagonal matrices and suppose that \(W_{ii} > 0\) and \(\overline{W}_{ii} > 0\) for some \(i \in [n]\). Then

\[
\tau_i(WA) \leq \frac{\overline{W}_{ii}^2}{\overline{W}_{ii}^2} \left(1 + \sqrt[\lambda_{\max}(A (A^\top A)^+ A^\top) \|W - \overline{W}\|_{\infty}}\right)^2 \tau_i(WA). \quad (A.6)
\]

**Proof.** Scaling the variables in Lemma 1 we have that there exists \(x \in \mathbb{R}^d\) such that

\[
A^\top Wx = a_i \quad \text{and} \quad \|x\|_2^2 = \frac{\tau_i(WA)}{\overline{W}_{ii}^2}. \quad (A.7)
\]

Note that \(A^\top (W - \overline{W}) x\) is in the image of \(A^\top \overline{W}\) as \(A^\top Wx = a_i\) and \(\overline{W}_{ii} \neq 0\).
Consequently,

\[ A^T \overline{W} y = A^T (W - \overline{W}) x^{(k)} \quad \text{for} \quad y \overset{\text{def}}{=} \overline{W} A \left( A^T \overline{W}^2 A \right)^+ A^T (W - \overline{W}) x. \]

Since \( A^T \overline{W} (x + y) = a_i \), Lemma 1 implies

\[ \tau_i (\overline{W} A) \leq \overline{W}^2_{ii} ||x + y||^2_2. \quad (A.8) \]

We can bound the contribution of \( y \) by

\[
\begin{align*}
||y||^2_2 & \leq \left\| \overline{W} A \left( A^T \overline{W}^2 A \right)^+ A^T (W - \overline{W}) x \right\|^2_2 \\
& \leq \lambda_{\max} \left( A \left( A^T \overline{W}^2 A \right)^+ A^T \right) ||W - \overline{W}||^2_\infty ||x||^2_2. \quad (A.9)
\end{align*}
\]

Applying triangle inequality to (A.7), (A.8), and (A.9) yields the result. \( \square \)

**Proof of Lemma 8.** For any \( i \in [n] \) such that \( \overline{W}_{ii} = 0 \) (A.5) follows trivially from the fact that leverage scores are non-negative. For any \( i \in [n] \) such that \( \overline{W}_{ii} > 0 \), since \( W^{(k)} \rightarrow \overline{W} \) we know that, for all sufficiently large \( k \geq N \) for some fixed value \( N \), it is the case that \( W_{ii}^{(k)} > 0 \). Furthermore, this implies that as \( k \to \infty \) we have \( \overline{W}_{ii}^2 / (W_{ii}^{(k)})^2 \to 1 \) and \( ||W^{(k)} - \overline{W}||_\infty \to 0 \). Applying Lemma 26 with \( W = W^{(k)} \) and taking \( \lim \inf_{k \to \infty} \) on both sides of (A.6) gives the result. \( \square \)
Appendix B

Sparse Recovery Sketch

In this section we give a proof of the $\ell_2$ heavy hitters algorithm given in Lemma 16. It is known that $\ell_2$ heavy hitters is equivalent to the $\ell_2/\ell_2$ sparse recovery problem [GI10]. Some sparse recovery algorithms are in fact based on algorithms for solving heavy hitters problem. However, we were not able to find a suitable reference for an $\ell_2$ heavy hitters algorithm so we show the reduction here - namely, how to find $\ell_2$ heavy hitters using a sparse recovery algorithm.

We follow the terminology of [GLPS12]. An approximate sparse recovery system consists of parameters $k, N$, an $m \times N$ measurement matrix $\Phi$, and a decoding algorithm $D$. For any vector $x \in \mathbb{R}^N$ the decoding algorithm $D$ can be used to recover an approximation $\hat{x}$ to $x$ from the linear sketch $\Phi x$. In this paper we will use a sparse recovery algorithm that achieves the $\ell_2/\ell_2$ sparse recovery guarantee:

$$||\hat{x} - x||_2 \leq C \cdot ||x - x_k||_2$$

where $x_k$ is the best $k$-term approximation to $x$ and $C > 1$. Our main sparse recovery primitive is the following result of [GLPS12]:

**Theorem 27** (Theorem 1 in [GLPS12]). For each $k \geq 1$ and $\epsilon > 0$, there is an algorithm and a distribution $\Phi$ over matrices in $\mathbb{R}^{O(k \log(N/k)/\epsilon) \times N}$ satisfying that for any $x \in \mathbb{R}^N$, given $\Phi x$, the algorithm returns $\hat{x}$ such that $\hat{x}$ has $O(k \log^{O(1)} N/\epsilon)$
non-zeros and

\[ ||\tilde{x} - x||_2^2 \leq (1 + \epsilon)||x - x_k||_2^2 \]

with probability at least 3/4. The decoding algorithm runs in time \( O(k \log^{O(1)} N/\epsilon) \).

Using this primitive, we can prove Lemma B.

**Lemma 16 (\( \ell_2 \) Heavy Hitters).** For any \( \eta > 0 \), there is a decoding algorithm \( D \) and a distribution on matrices \( \Phi \) in \( \mathbb{R}^{O(\eta^{-2} \text{polylog}(N)) \times N} \) such that, for any \( x \in \mathbb{R}^N \), given \( \Phi x \), the algorithm \( D \) returns a vector \( w \) such that \( w \) has \( O(\eta^{-2} \text{polylog}(N)) \) non-zeros and satisfies

\[ ||x - w||_{\infty} \leq \eta ||x||_2. \]

with probability \( 1 - N^{-c} \) over the choice of \( \Phi \). The sketch \( \Phi x \) can be maintained and decoded in \( O(\eta^{-2} \text{polylog}(N)) \) space.

**Proof.** Let \( h : [N] \rightarrow [16/\eta^2] \) be a random hash function (pairwise independence suffices), and for \( j = 1, \ldots, 16/\eta^2 \) let \( y^j = x_i \) if \( h(i) = j \) and 0 o.w. For a vector \( u \in \mathbb{R}^N \) we write \( u_{-i} \) to denote \( u \) with the \( i \)-th component zeroed out.

By Markov’s inequality we have

\[ \mathbb{P}[||y^h(i)||^2 > \eta^2 ||x||^2/2] < 1/8. \]

Note that since we are only using Markov’s inequality, it is sufficient to have \( h \) be pairwise independent. Such a function \( h \) can be represented in small space. Now invoke the result of Theorem 27 on \( y^h(i) \) with \( k = 1, \epsilon = 1 \), and let \( w^h(i) \) be the output. We have

\[ ||y^h(i) - w^h(i)||_2 \leq 2 ||y^h(i) - y^h_k||_2^2 \leq 2 ||y^h_i||_2^2. \]

Hence, we have

\[ (y^h_i - w^h_i)^2 \leq \eta^2 ||x||^2. \]

This shows that applying sketches from Theorem 27 to vectors \( y^j \), for \( j = \ldots, 16/\eta^2 \)
1, \ldots, 16 / \eta^2 \text{ and outputting the vector } \mathbf{w} \text{ with } w_i = w_i^{h(i)} \text{ allows us to recover all } i \in [N] \text{ with } \eta \| \mathbf{x} \|_2 \text{ additive error with probability at least } 3/4 - 1/8.

Performing } O(\log N) \text{ repetitions and taking the median value of } w_i \text{ yields the result. Note that our scheme uses } O(\eta^{-2} \text{polylog}(N)) \text{ space and decoding time, and is linear in } \mathbf{x}, \text{ as desired.} \quad \square
Appendix C

Recursive Sparsification

For completeness, we give a short proof of Theorem 17:

**Theorem 17 (Recursive Sparsification – [MP12], Section 4).** Consider a PSD matrix $K$ with maximum eigenvalue bounded $\leq \lambda_u$ and minimum non-zero eigenvalue bounded $\geq \lambda_l$. Let $m = \lceil \log_2 (\lambda_u/\lambda_l) \rceil$. For $\ell \in \{0, 1, \ldots, m\}$, define:

$$\gamma(\ell) = \lambda_u/2^\ell$$

So $\gamma(m) \leq \lambda_l$ and $\gamma(0) = \lambda_u$. Then the chain of matrices, $[K(0), K(1), \ldots, K(m)]$ defined as:

$$K(\ell) = K + \gamma(\ell)I_{d \times d}$$

satisfies the following relations:

1. $K \preceq_r K(m) \preceq_r 2K$
2. $K(\ell) \preceq K(\ell - 1) \preceq 2K(\ell)$ for all $\ell \in \{1, \ldots, m\}$
3. $K(0) \preceq 2\gamma(0)I \preceq 2K(0)$

When $K$ is the Laplacian of an unweighted graph, $\lambda_{\text{max}} < 2d$ and $\lambda_{\text{min}} > 8/d^2$ (where here $\lambda_{\text{min}}$ is the smallest nonzero eigenvalue). Thus the length of our chain, $m = \lceil \log_2 (\lambda_u/\lambda_l) \rceil$, is $O(\log d)$. 

95
Proof. Relation 1 follows trivially from the fact that $\gamma(m) \leq \lambda_i$ is smaller than the smallest nonzero eigenvalue of $K$. For any $x \perp \ker(K)$:

$$x^T K(m) x = x^T K x + x^T (\gamma(m) I) x \leq x^T K x + x^T (\lambda_{min} I) x \leq 2x^T K x$$

The other direction follows from $\gamma(m) I \geq 0$. Using the same argument, relation 3 follows from the fact that $\gamma(0) \geq \lambda_{max}(K)$. For relation 2:

$$2K(\ell) = 2K + 2\gamma(\ell) I = 2K + \gamma(\ell - 1) I \geq K(\ell - 1)$$

Again, the other direction just follows from $\gamma(\ell) I \geq 0$.

Finally, we need to prove the required eigenvalue bounds. For an unweighted graph, $\lambda_{max} < d$ follows from fact that $d$ is the maximum eigenvalue of the Laplacian of the complete graph on $d$ vertices. $\lambda_{min} > 8/d^2$ by Lemma 6.1 of [ST14]. Note that this argument extends to weighted graphs when the ratio between the heaviest and lightest edge is bounded by a polynomial in $d$.  

$\square$
Bibliography


