CS 229r Project - Graph Sparsification and Dimensionality Reduction

Cameron and Christopher Musco

December 11, 2013

1 Introduction

We are studying the connection between graph sparsifiers and subspace embeddings, two closely related forms of matrix dimensionality reduction. The first is concerned specifically with matrices used to represent graphs, while the later is applicable to any matrix. While work on sparsification has impacted work in the general case, fewer techniques have flowed the opposite way. Hoping to reverse this trend, we originally set out to apply recent work on “oblivious subspace embeddings” [23] [5], the fastest techniques for sparse matrix embedding, to the vertex-edge incidence matrices of graphs. While we have abandoned this direction for now, based on our improved understanding of sparsification and general matrix compression, we are currently working towards a new algorithm for sparse subspace embeddings. Our approach expands on and, if successful, would improve work in [22].

In this report, we begin by presenting background research on sparsification and dimensionality reduction (Section 2). We then review current algorithms for sparsification and their application to general matrices (Section 3). We briefly introduce a current (unproven) approach that we are working on for sparse subspace embeddings (Section 4). Finally, over the course of this project we explored several related topics, including spectral sparsification in the semi-streaming model. We will present a brief literature review of that area in Appendix A. In the appendix we will also briefly explain our initial thoughts about oblivious subspace embeddings and their potential application to graph matrices.

2 Dimensionality Reduction for Graphs and Matrices

2.1 Spectral Sparsification

Consider an undirected, weighted graph $G = (V, E)$ with n vertices and m edges. We can write each edge $e \in E$ as the pair of vertices it connects: (v_i, v_j). Each $e = (v_i, v_j)$ is assigned a positive weight, $w_{i,j}$. If (v_i, v_j) is not an edge in G, $w_{i,j} = 0$. The weighted degree of a vertex v_i is the sum of edge weights for edges adjacent to that vertex: $deg(v_i) = \sum_{j=1}^{n} w_{i,j}$. G can be represented by the $n \times n$ Laplacian matrix L_G:

$$L_G(i,j) \overset{\text{def}}{=} \begin{cases} \deg(v_i) \text{ if } i = j \\ -w_{i,j} \text{ if } i \neq j \end{cases}$$

The Laplacian is symmetric diagonally dominant and positive semidefinite since its diagonal entries are ≥ 0. L_G is especially useful since its quadratic form measures the smoothness or “energy” of a function x over the vertices of G. If x varies little, the value of the quadratic form will be small. Larger discrepancies in the value of x between adjacent vertices will increase the measure. Discrepancies between strongly connected vertices (large $w_{i,j}$) are penalized more heavily.

$$x^\top L_G x = \sum_{(v_i,v_j)\in E} w_{i,j}(x(v_i) - x(v_j))^2 \tag{1}$$

Definition 1. *A spectral sparsifier H for G is a subgraph (possibly with reweighted edges) such that*

$$\forall x \in \mathbb{R}^n, (1-\epsilon)x^\top L_G x \leq x^\top L_H x \leq (1+\epsilon)x^\top L_G x$$

That is, H approximately preserves the Laplacian quadratic form, and hence the measure of smoothness of functions over G. Another common way to denote this guarantee is to use the symbol $\preceq$. $A \preceq B$ if $B - A$ is positive semidefinite. In other words, for all x, $x^\top(B-A)x \geq 0$ and thus $x^\top Bx \geq x^\top Ax$. So we can rewrite the inequality in Definition 1 as:

$$(1-\epsilon)L_G \preceq L_H \preceq (1+\epsilon)L_G$$

To be more succinct, we will also use the notation $L_H \approx_\epsilon L_G$ to denote this inequality.

Spielman and Teng first introduced spectral sparsification when working on fast solvers for systems of equations on Laplacian and symmetric diagonally dominant (SDD) matrices. Their work appears initially in [25], but has been recently cleaned up and reorganized for clarity in [28], [27], [26]. Since [25], spectral sparsifiers have found wide application in approximation algorithms for graph problems. Since a sparsifier H approximates G, we can often reduce G to H and then run a graph algorithm on H to find an approximate solution for G. As we will discuss, sparsifiers with only $O(n \log n/\epsilon^2)$ edges can be found in nearly linear time in the size of the original graph, which may have up to $m = O(n^2)$ edges. Thus, the speed up from approximation can be significant for algorithms whose runtime depends on the number of edges in G.

However, from the above definition of approximation, it is not always obvious that such an approach will work. We saw that L_H approximates some measure of smoothness over L_G, but how do we know an algorithm run on H will yield an approximate solution for G? Does H approximate G in the "right way"? This answer depends on the specific graph problem we are attempting to solve approximately, but often it boils down to one very useful fact: the Laplacian quadratic form of a spectral sparsifier H preserves every eigenvalue of L_G multiplicatively (even the smallest eigenvalues). This characterization of the approximation requirement lends spectral sparsification its name.

Lemma 1. *Let $\lambda_i^{(G)}$ be the i^{th} largest eigenvalue of L_G and let $\lambda_i^{(H)}$ be the i^{th} largest eigenvalue of L_H. If H is an ϵ spectral sparsifier for G, then*

$$(1-\epsilon)\lambda_i^{(G)} \leq \lambda_i^{(H)} \leq (1+\epsilon)\lambda_i^{(G)} \tag{2}$$

Proof. Following the approach suggested in [24], we can use the Courant-Fischer Max-Min theorem to characterize the eigenvalues of L_G:

$$\lambda_i^{(G)} = \max_{S:dim(S)=i} \min_{x \in S} \frac{x^\top L_G x}{x^\top x} \tag{3}$$

Let $S_k^{(G)}$ be the k-dimensional subspace spanned by the first k eigenvectors of L_G, $v_1^{(G)}, \ldots, v_k^{(G)}$. Define $S_k^{(H)}$ similarly for L_H. Then, by Equation 3:

$$\lambda_i^{(G)} \leq \min_{x \in S_k^{(H)}} \frac{x^\top L_G x}{x^\top x} \leq \min_{x \in S_k^{(H)}} \left(\frac{1}{1-\epsilon}\right) \frac{x^\top L_H x}{x^\top x} = \left(\frac{1}{1-\epsilon}\right) \lambda_i^{(H)} \tag{4}$$

The second to last inequality follows from Definition 1 of spectral approximation. Similarly:

$$\lambda_i^{(H)} \leq \min_{x \in S_k^{(G)}} \frac{x^\top L_H x}{x^\top x} \leq \min_{x \in S_k^{(G)}} (1+\epsilon) \frac{x^\top L_G x}{x^\top x} = (1+\epsilon)\, \lambda_i^{(G)} \tag{5}$$

Combining (4) and (5) gives the desired eigenvalue bound. □

2.1.1 Cut Sparsification

It is worth noting that spectral sparsifiers build on previously studied "cut sparsifiers", which ask for a subgraph H that preserves all cuts in G up to a multiplicative factor. The spectral sparsification guarantee is strictly stronger than the cut sparsification guarantee, which is equivalent to preserving $x^\top L_G x$ for all cut characteristic vectors - i.e. $x \in \{0,1\}^n$. Cut sparsifiers were first introduced by Benczur and Karger in work on faster minimum cut algorithms [4].

2.2 Subspace Embeddings

Definition 2. *An ϵ-subspace embedding for the column space of an $m \times n$ matrix A is an $m' \times m$ matrix Π with $m' << m$ such that, for any $x \in \mathbb{R}^n$, $(1-\epsilon)\|Ax\|_2 \leq \|\Pi Ax\|_2 \leq (1+\epsilon)\|Ax\|_2$.*

We will also write this guarantee as $\|\Pi Ax\|_2 \approx_\epsilon \|Ax\|_2$. Subspace embeddings are used to reduce a matrix's size while preserving its operation on vectors. Much as solving graph problems on sparsifiers can give approximate solutions to these problems on the original graph, solving problems such as linear regression, low rank approximation, and leverage score computation on the reduced matrix $\tilde{A} \overset{\text{def}}{=} \Pi A$ gives fast approximate solutions to these problems on the original matrix A. To maximize acceleration of linear algebraic problems, the goal is to find a Π that is both fast to apply to A and has few rows (i.e. it gives a small $\tilde{A}$).

Recent work on various subspace embedding techniques largely stems from closely related work on Johnson-Lindenstrauss transforms, which reduce the dimension of a finite set of vectors and preserve their norms. Subspace embeddings are a continuous analog, attempting to simultaneously compress and preserve the norms of every vector in a certain low dimensional subspace. Specifically, consider the n dimensional subspace spanned by the columns of A. This subspace lives in the much large $\mathbb{R}^m$. Every vector in the subspace can be written as Ax for some $x \in \mathbb{R}^n$. Thus, if $\|\Pi Ax\|_2 \approx_\epsilon \|Ax\|_2$, Π can be used to compress every vector in our subspace from $\mathbb{R}^m \to \mathbb{R}^n$ while preserving its ℓ_2 norm.

As mentioned, subspace embeddings are also closely related to graph sparsifiers. For any graph G with n vertices and m edges, let W_G be an $m \times m$ matrix with the edge weights of G on its diagonal. Let B_G be the $m \times n$ "vertex-edge incidence matrix" of G. Each row of B_G corresponds to an edge $e = (v_i, v_j)$. The row has a 1 at position i, a (-1) at position j, and zeros everywhere else. The 'direction' of an edge and thus the signs at positions i and j can be chosen arbitrarily. The Laplacian matrix can be written $L_G = (W_G^{1/2} B_G)^\top (W_G^{1/2} B_G)$, and thus:

$$x^\top L_G x = x^\top (W_G^{1/2} B_G)^\top (W_G^{1/2} B_G) x = \|W_G^{1/2} B_G x\|_2^2$$

So, if we have an ϵ-spectral sparsifier H with $x^\top L_H x = (1 \pm \epsilon) x^\top L_G x$, then:

$$\|W_H^{1/2} B_H x\|_2^2 = (1 \pm \epsilon) \|W_G^{1/2} B_G x\|_2^2 \tag{6}$$

Let $m' < m$ be the number of edges in H. We can write $W_H^{1/2} B_H = S W_G^{1/2} B_G$, where S is an $m' \times m$ sampling matrix with a single nonzero in each row. Each entry in S selects and reweights an edge of G to be placed in H. By Equation 6, S is a subspace embedding for $W_G^{1/2} B_G$, with error $\sqrt{1 \pm \epsilon}$. Therefore, finding a spectral sparsifier gives a subspace embedding. More broadly, the subspace embedding and spectral sparsification guarantees are exactly the same modulo an important point we will discuss below. Like spectral sparsifiers, a subspace embedding preserves every eigenvalue of $A^\top A$ multiplicatively.

2.2.1 Row Selection vs. Recombination

The key difference between sparsifiers and subspace embeddings is that Definition 2 places no constraints on Π. The rows of $\tilde{A} = \Pi A$ are arbitrary linear combinations of the rows of A. However, a spectral sparsifier H must be a subgraph of G. $W_H^{1/2} B_H$ must contain only reweighted single rows of $W_G^{1/2} B_G$. Of course it is possible to consider a "spectral sparsifier matrix" that is not a subgraph, but still a subspace embedding of $W_G^{1/2} B_G$. Furthermore, in the general matrix context, it is possible to consider subspace embeddings where Π is restricted to be a row sampling matrix. Both of these formulations may be useful in application and we will consider them later in this report.

2.3 Outer Product Sparsification

We will present one final perspective on sparsification and subspace embedding that appears in the literature and may be helpful in gaining intuition. For a general $m \times n$ matrix A, lets call the $n \times n$ matrix $C = A^\top A$ the 'covariance matrix' of A (a true covariance matrix would have the columns centered to mean zero, so we are abusing vocabulary a bit). L_G is simply the covariance matrix of $W^{1/2}B$ and, like all covariance matrices, is symmetric and positive semidefinite since $x^\top Cx = \|Ax\|_2^2 \geq 0$ for all x. Preserving $\|Ax\|_2^2$ is equivalent to preserving the quadratic form on C, $x^\top Cx$. We can write C as a sum of rank one outer products:

$$C = \sum_{i=1}^{m} a_i^\top a_i$$

where a_i is the i^{th} row of A. If A is a vertex edge incidence matrix, a_i is simply the characteristic vector of the i^{th} edge of the graph (having ± 1 at the indices corresponding to the edge's endpoints). Thus, finding a spectral sparsifier, or more generally a subspace embedding via row selection, is equivalent to finding a sparse vector s such that:

$$C \approx_\epsilon \sum_{i=1}^{m} s_i a_i^\top a_i$$

Alternatively, we can put our row vectors in 'isotropic' position. Consider the singular value decomposition, $A = U\Sigma V^\top$. $C \overset{\text{def}}{=} A^\top A = V\Sigma^2 V^\top$. Let $C^{-1/2} \overset{\text{def}}{=} V\Sigma^{-1}$. Then, we see that $(AC^{-1/2})^\top(AC^{-1/2}) = I$. Our goal becomes to find a sparse s such that:

$$I \approx_\epsilon \sum_{i=1}^{m} s_i (a_i C^{-1/2})^\top (a_i C^{-1/2})$$

Thus, both graph sparsification and subspace embedding through row selection boil down to sparsifying a sum of rank one outer products that sum to the identity. Standard subspace embeddings that combine rows arbitrarily do not fit as cleanly into this framework.

2.4 Limits for Dimensionality Reduction

Before moving on to actual algorithms, lets quickly point out a lower bound on the number of rows in a subspace embedding matrix Π. Such a bound would also limit the minimum number of edges required for constructing a spectral sparsifier (since row selection is strictly more difficult than general subspace embedding) . If $\tilde{A} \approx_\epsilon A$, then $\tilde{A}$ and A must have the same null space, and hence the same rank. If there is a vector x in the null space of $\tilde{A}$ but not in the null space of A, then $\|\tilde{A}x\|_2^2 = 0$ while $\|\tilde{A}\|_2^2 > 0$. We would not have a $(1 \pm \epsilon)$ multiplicative preservation of norm. We conclude that $\tilde{A}$ (and therefore Π) must have at least $rank(A)$ rows, or n rows if A is full rank.

In fact, a Π achieving this lower bound can be found using the singular value decomposition, $A = U\Sigma V^\top$. Consider the "reduced SVD" where U is an $m \times rank(A)$ matrix, Σ is $rank(A) \times rank(A)$, and V is $rank(A) \times n$. U has orthonormal columns, so $U^\top U = I$. Thus:

$$\left\|U^\top Ax\right\|_2^2 = x^\top A^\top U^\top UAx = x^\top A^\top Ax = \|Ax\|_2^2 \tag{7}$$

So $\Pi = U^\top$ is actually a perfect subspace embedding and has just $rank(A)$ rows. $\tilde{A} = U^\top A = \Sigma V^\top$.

This lower bound is intuitive in the context of graph sparsification. It states that we cannot find $W_H^{1/2} B_H$ with fewer than $rank(W_G^{1/2} B_G)$ rows. If G is connected, the rank of $W_G^{1/2} B_G$ is $n-1$. Its null space, which is the same as the null space of L_G, is spanned by the all ones vector, which is a completely smooth function over G (see Equation 1). Thus, we cannot have a spectral sparsifier with fewer than $n-1$ edges. Well, that many edges is necessary even just to ensure that H is connected (a spanning tree requires $n-1$ edges). Recall that spectral sparsification is strictly stronger than cut sparsification. The graph better remain connected if we are to preserve cut values multiplicatively (if not, the cut between two sets of nodes that are disconnected in H but connected in G falls to 0). Technically, if k is the number of connected components in G, we can have a sparsifier with $n-k$ edges. However, we usually only consider the connected case since spectral sparsifiers of different connected components can be found independently from one another.

3 Algorithms for Spectral Sparsification

Known algorithmic results for graph sparsification are quite strong. Furthermore, most of the algorithms we discuss can actually be applied to row selection schemes that give good subspace embeddings for general matrices. It is known that spectral sparsifiers with $O(n/\epsilon^2)$ edges exist and can be found in polynomial time [3]. These sparsifiers are sometimes known as 'constant degree sparsifiers' since they have constant average degree. Of greater practical interest, sparsifiers with $O(n \log n/\epsilon^2)$ edges can be found in $\tilde{O}(m \log^2 n/\epsilon^2)$ time, where the tilde hides $\log \log n$ factors [18]. We briefly review algorithmic results for constant degree and $O(n \log n)$ sparsifiers before looking more in depth into edge sampling, the most popular technique for computing sparsifiers, and the one we are currently focused on.

3.1 Constant Degree Sparsifiers

The existence of $O(n/\epsilon^2)$ sparsifiers was first shown by Batson, Spielman, and Srivastava in [3], which also gives a polynomial time algorithm for finding them (the existence proof is by construction). The authors named their sparsifiers 'Twice Ramanujan Sparsifiers' based on close connections to Ramanujan expanders. A Ramanujan expander is a d-regular graph that optimally approximates the complete graph amongst all possible d-regular graphs. For any given d, the Ramanujan graph achieves a multiplicative spectral approximation factor of $\frac{d+1+2\sqrt{d}}{d+1-2\sqrt{d}}$, which is known as the Ramanujan bound. The sparsification algorithm in [3] achieves this same error for *any* graph G by finding a subgraph with $d(n-1)$ edges (giving an average degree of $2d$, which is about twice as dense as a Ramanujan expander). Although the Ramanujan bound is only known to holds for d-regular graphs, it is conjectured that it also holds for graphs with average degree d [29]. If that is the case, then the graphs presented in [3] are essentially optimal (within a factor or 2).

Although improvements have followed [3], the current fastest algorithm for finding $O(n/\epsilon^2)$ sparsifiers requires $\tilde{O}(n^4 \log n/\epsilon^2 \max\{\log^2 n, 1/\epsilon^2\})$ time [30]. This is not fast enough to justify the use of such sparsifiers in accelerating graph algorithms, especially since nearly as good $O(n \log n/\epsilon^2)$ sparsifiers can be found in nearly linear time. Nevertheless, the existence of constant degree sparsifiers is interesting. As explained above, in order to preserve even connectivity, a sparsified graph must contain $(n-1)$ edges. It is surprising that only a constant factor increase in the number of edges is required for preserving the full spectrum of the Laplacian. Furthermore, the techniques introduced in [3] are directly applicable to general matrices, and can thus be used for finding subspace embeddings through row selection. This means that it is possible to match the number of rows achieved by the optimal embedding $\Pi = U^\top$ up to a factor of ϵ^{-2}, even when our Π is only allowed to select and reweight rows (as opposed to linearly combining them).

Finally, recall the perspective on sparsification presented in Section 2.3. Our goal is to sparsifier a sum of outer products, each of which is a rank 1 positive semidefinite matrix. De Carli Silva, Harvey, and Sato are actually able to extend the approach in [3] to the sparsification of sums of general PSD matrices [7] (as opposed to just rank-1 PSDs). For any set of $n \times n$ matrices $A_0, A_1, \ldots, A_m \succeq 0$ they show how to find s with sparsity $O(n/\epsilon^2)$ satisfying

$$(1-\epsilon)\sum_{i=1}^{m} A_i \preceq \sum_{i=1}^{m} s_i A_i \preceq (1+\epsilon)\sum_{i=1}^{m} A_i$$

Their algorithm requires $O(mn^3/\epsilon^2)$ time. These strong generalizations lead to an interesting open question:

Open Question. *If current state of the art algorithms do not use graph structure or, in the case of [7], even the fact that our component PSD matrices are outer products of single rows, can constant degree sparsifiers and subspace embeddings be computed more quickly? Is it possible to leverage these structural facts to improve runtime and then possibly use constant degree compressions for fast approximation algorithms?*

3.2 $O(n \log n)$ Sparsifiers

In the mean time, there has been significant work on computing spectral sparsifiers with $O((n \log n)/\epsilon^2)$ edges in time nearly linear in m (the original number of edges in G). As mentioned in Section 2.1, such sparsifiers were originally motivated by applications to solving symmetric diagonally dominant linear systems in nearly linear time [25]. The goal was to reduce a general SDD system to a Laplacian system and to then use sparsification to quickly approximate the solution to that system. If $(u-1)$ is the maximum edge weight in a graph G and δ is the maximum failure probability of the algorithm, [25] shows how to compute

$O(n \log^{31}(n/\delta) \log(u)/\epsilon^2)$ sparsifiers in $O(m \log^{15}(n) \log(u) \log(1/\delta))$ time. Their algorithm decomposes G into multiple high conductance components with few connections between them. It then uses uniformly random edge sampling to sparsify each component separately.

In [24], Spielman and Srivastava show that $O((n \log n)/\epsilon^2)$ spectral sparsifiers can be found by sampling edges according to their "effective resistances". We will explain this method in more detail below. Effective resistances can be computed in nearly linear time in m using Spielman and Teng's fast SDD solver, which itself employs Spielman and Teng's original sparsification scheme. So, while Spielman and Srivastava's work does not improve on the runtime of this scheme (it uses it as a subroutine), it does achieve better sparsifiers (removing the large exponent on $\log n$ and the dependence on $1/\delta$ and u).

Finally, an improved version of the effective resistance method is used by Koutis, Levine, and Peng in [18] to find sparsifiers with $O((n \log n)/\epsilon^2)$ edges in $\tilde{O}((m \log^2 n) nic/\epsilon^2)$ time. Improvements to SDD system solvers, which no longer rely on full $O(n\text{polylog}(n))$ sparsifiers like Spielman and Teng's original solver, have contributed to runtime improvements for effective resistance sampling algorithms ([19], [20], [17], [21]).

3.3 Edge Sampling

We are focusing on edge sampling, which is the most popular technique for $O((n \log n)/\epsilon^2)$ sparsification. The general approach is to assign a probability to each edge in G and to then sample $O((n \log n)/\epsilon^2)$ edges independently according to this probability distribution. The edges are appropriately reweighted and combined to form a sparsifier H. The goal is to choose a distribution over edges that ensures H is an ϵ spectral sparsifier with high probability. We are especially interested in edge sampling techniques because they are fairly simple to analyze and they extend naturally to row sampling schemes for subspace embedding.

Informally, a sparsifier with $O(n \log n)$ edges is essentially optimal for any sampling approach. Consider sparsifying a complete graph with unit edge weights. If you sample each edge with equal probability (since the graph is completely symmetric) then even to have one edge incident to every vertex with high probability, you need $O(n \log n)$ samples. This is an instance of the coupon collector's problem. Recall that at minimum a graph must remain connected if it is to achieve the spectral sparsification guarantee.

3.3.1 Sampling for Cut Sparsification

Edge sampling was first introduced by David Karger as a method for computing cut sparsifiers [15]. Each edge in G is included in the sparsifier H with probability $\Theta(\ln n/(\epsilon^2 c))$, where c is the minimum cut of G. Included edges are reweighted by the inverse of the probability of inclusion so that, in a sense, the expected value of H is G. By using a crude, easy to compute under estimate for c, Karger's first sparsifiers could be used to find fast approximations of minimum cuts, maximum flows, and other graph properties.

In [4], Benczur and Karger removed the dependence on the minimum cut value of G, showing that cut sparsifiers with $O(n \log n/\epsilon^2)$ edges can be computed in $O(m \log^3 n)$ time. In this paper, edges are sampled *nonuniformly*, with probability inversely proportional to the edge's "strong connectivity", c_e. Upper bounded by an edge's standard connectivity, c_e is is the maximum value of k such that a k-connected subgraph of G contains e. Benczur and Karger show how to quickly find good enough approximations to c_e for sampling. [12] finally resolves a conjecture that edges can be sampled simply by regular connectivity (the minimum cut between their endpoints), to obtain $O(n \log n/\epsilon^2)$ cut sparsifiers.

3.3.2 Sampling for Spectral Sparsification

Sampling algorithms for spectral sparsifiers require further refinements to the nonuniform sampling probabilities. $O(n \log n/\epsilon^2)$ spectral sparsifiers can be constructed by sampling edges of G by "effective resistance" [24]. Treating the edges of G as resistors with resistance inversely proportional to their weight, the effective resistance between v_i and v_j is the voltage drop between v_i and v_j if 1 unit of current is forced to flow between them. The effective resistance, R_e, of an edge can be computed by $R_e = B_e L^+ B_e^\top$, where B_e is the row of the edge-vertex incidence matrix B corresponding to edge e. L^+ is the pseudoinverse of the graph Laplacian. Sampling probabilities are selected to be proportional to $w_e R_e$. Spielman and Srivastava use a matrix concentration result of Rudelson and Vershynin to prove that effective resistance sampling is sufficient for spectral sparsification (see Lemma 5 in [24]). Furthermore, using a fast SDD solver, L^+ can be found in nearly linear time. The bottleneck for computing effective resistances then becomes the multiplication of $B_e L^+ B_e^\top$, which is accelerated via an application of the Johnson-Lindenstrauss lemma.

3.3.3 Sampling for Subspace Embeddings

Effective resistance sampling extends naturally to row sampling for general matrices. For an $m \times n$ matrix A with rows a_i, define the statistical leverage score of row i as $\tau_i = a_i(A^\top A)^+ a_i^\top$, which is the i^{th} diagonal entry of $AC^+A^\top$. Leverage scores have appeared in the statistics literature for a while. If A is a data matrix (each row is a high dimensional data point), τ_i measures the importance of row i in determining a line of best fit. For a weighted vertex edge incidence matrix:

$$\tau_e = (W^{1/2}B)_e L^+ (W^{1/2}B)_e^T = w_e R_e$$

Theorem 3 (Leverage Score Sampling)**.** *If $q = O(n \log n/\epsilon^2)$ rows of A are sampled (independently with replacement) with probability $p_i \propto \tau_i$, and reweighted by a factor of $\frac{1}{p_i q}$ where q is the number of samples taken, then with high probability $\tilde{A}^\top \tilde{A} \approx_\epsilon A^\top A$.*

In other words, leverage score sampling gives an ϵ subspace embedding for A with $O(n \log n/\epsilon^2)$ rows. The proof follows immediately from Spielman and Srivistava's matrix concentration result in [24], which does not depend on graph structure. As far as we could tell, this paper is the first to realize that leverage scores (effective resistances) can be used directly for row sampling. However, other lines of research were quite close to that realization at the time. For example, [9] and [10] give strict requirements on the probabilities that would be required for finding subspace embeddings via sampling. It is easy to verify that leverage scores satisfy the properties proposed.

Without a formal proof, lets consider why leverage scores make natural sampling probabilities. Recall that a subspace embedding preserves all of the eigenvalues of $A^\top A$ (the singular values of A) multiplicatively, regardless of their size. Thus, after sampling, we would like our compression, $\tilde{A}$, to contain a fair number of rows in the direction of every eigenvector of $A^\top A$ (right singular vector of A). Using the singular value decomposition:

$$\begin{aligned}\tau_i &= a_i(A^\top A)^+ a_i^\top = a_i((U\Sigma V^\top)^\top U\Sigma V^\top)^+ a_i^\top && (8)\\ &= a_i(V\Sigma^2 V^\top)^+ a_i^\top = a_i V\Sigma^{-2} V^\top a_i^\top && (9)\end{aligned}$$

The entries in the diagonal matrix Σ^2 are the eigenvalues of $A^\top A$, or the squared singular values of A. Σ^{-2} is formed by inverting each non-zero diagonal entry and leaving any 0's at 0. If A has $k \leq n$ non-zero singular values, $\sigma_1, \ldots, \sigma_k$, and v_j is the j^{th} column of V:

$$a_i V\Sigma^{-2} V^\top a_i^\top = \sum_{j=1}^{k} \frac{1}{\sigma_j^2} \langle a_i, v_j \rangle^2 \tag{10}$$

The squared singular value σ_j^2, is equal to $\sum_{i=1}^{m} \langle a_i, v_j \rangle^2$, so we see that:

$$\tau_i = \sum_{j=1}^{k} \frac{\langle a_i, v_j \rangle^2}{\sum_{l=1}^{m} \langle a_l, v_j \rangle^2} \tag{11}$$

In other words, to compute the leverage score of a row, we first compute its relative contribution to every singular vector, v_j. These contributions are summed to get our sampling probability, with no preference given to more dominant singular directions.

4 A New Approach to Row Selection

Since [24], leverage score sampling has been successfully applied to general matrix compression [8]. In contrast to unrestricted subspace embeddings, row selection schemes are often desirable because they give an approximation, $\tilde{A}$, that preserves some of the structure of A. For example, when compressing the vertex-edge incidence matrix of a graph, row selection ensures that $\tilde{A}$ is also a vertex-edge incidence matrix. Sometimes this preservation is important for interpretation: we would like to know which data points in a set are most important. In other case, it is important algorithmically. For graphs, maintaining an edge-vertex incidence is essential because it allows us to use fast SDD solvers on our compressed matrix. We are most interested in this second justification for row sampling.

Unfortunately, leverage score sampling for general matrices has had limited algorithmic success because computing leverage scores is slow. Both the multiplication of $A^\top A = C$ and the inversion of C are major algorithmic stumbling blocks. Thus, work on sampling has largely focused on data analysis, our first justification for row sampling. It is possible to obtain approximate leverage scores through other subspace embedding techniques, but if that approach is used, we may as well use those techniques directly. For example, sparse subspace embeddings can be used to find leverage scores for a sparse matrix in $\tilde{O}(nnz(A) + n^\omega)$ time [23]. If we incur the n^ω cost during compression, there is no point in finding an $\tilde{A}$ with structure that allows for faster system solves (which could be used for approximate regression, for example).

Recent work by Li, Miller, and Peng takes an important step towards solving this problem [22]. They introduces an iterative approach to leverage score sampling that computes a row sampled $\tilde{A}$ in $\tilde{O}(nnz(A) + n^{\omega+\theta})$ time, where θ is some small constant. While this does not improve upon other approaches, the paper avoids first compressing A via another subspace embedding technique. However, the $n^{\omega+\theta}$ term does arise from intermediate system solves on dense matrices whose rows are linear combinations of rows in A.

Open Question. *Is it possible to modify the iterative approach in [22] such that all intermediate system solves are only on matrices whose rows are scaled versions of rows in A? In doing so, can we eliminate the $n^{\omega+\theta}$ runtime factor, replacing it with the time required to solve, for example, a Laplacian system (via an SDD solver) or a sparse system (via the conjugate gradient method).*

Solving this question would justify the use of row selection in an algorithmic context. Depending on the structure of A (i.e. if it's a graph or sparse), it could make most sense to compress via a row selection technique so that any system solves on $\tilde{A}$ are faster. We believe that this goal is achievable, possibly using ideas that draw on the partitioning/uniform sampling techniques used by Spielman and Teng for finding their original $O(n \log n)$ sparsifiers (see Section 3.2). Unfortunately, our work on this final question (which is in collaboration with Aaron Sidford and Yin-Tat Lee at MIT) has come too close to the project deadline to be included in this report. However, we're working towards a proof of the technique we have in mind and will definitely pass along updates if it is successful.

References

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 5–14, New York, NY, USA, 2012. ACM.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 8096 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2013.

[3] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 255–262, New York, NY, USA, 2009. ACM.

[4] András A Benczúr and David R Karger. Approximating st minimum cuts in õ (n 2) time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55. ACM, 1996.

[5] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. *CoRR*, abs/1207.6365, 2012.

[6] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 81–90. ACM, 2013.

[7] Marcel K. de Carli Silva, Nicholas J. A. Harvey, and Cristiane M. Sato. Sparse sums of positive semidefinite matrices. *CoRR*, abs/1107.0088, 2011.

[8] Petros Drineas. Leverage scores (presentation). http://www.stanford.edu/group/mmds/slides2012/s-drineas.pdf, 2012.

[9] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Sampling algorithms for l_2 regression and applications. In *SODA*, pages 1127–1136, 2006.

[10] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error cur matrix decompositions. *SIAM J. Matrix Analysis Applications*, 30(2):844–881, 2008.

[11] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

[12] Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 71–80. ACM, 2011.

[13] Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *arXiv preprint arXiv:1203.4900*, 2012.

[14] Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 393–398. ACM, 2012.

[15] David R. Karger. Random sampling in cut, flow, and network design problems. In *STOC*, pages 648–657, 1994.

[16] Jonathan A Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. In *28th International Symposium on Theoretical Aspects of Computer Science*, page 440, 2011.

[17] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *STOC*, pages 911–920, 2013.

[18] Ioannis Koutis, Alex Levin, and Richard Peng. Faster spectral sparsification and numerical algorithms for sdd matrices. *CoRR*, abs/1209.5821, 2012.

[19] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *FOCS*, pages 235–244. IEEE Computer Society, 2010.

[20] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *FOCS*, pages 590–598, 2011.

[21] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. *CoRR*, abs/1305.1922, 2013.

[22] Mu Li, Gary L. Miller, and Richard Peng. Iterative row sampling. *CoRR*, abs/1211.2713, 2012.

[23] Jelani Nelson and Huy L. Nguyen. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. *CoRR*, abs/1211.1002, 2012.

[24] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC '08, pages 563–568, New York, NY, USA, 2008. ACM.

[25] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[26] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.

[27] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[28] Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1):1–26, 2013.

[29] Nikhil Srivastava. Spectral sparsification and restricted invertibility, 2010.

[30] Anastasios Zouzias. A matrix hyperbolic cosine algorithm and applications. *CoRR*, abs/1103.2793, 2011.

A Other Research Directions

A.1 Oblivious Subspace Embeddings for Graph Compression

An Oblivious Subspace Embedding (OSE) is a random subspace embedding matrix Π that can be applied to any $m \times n$ matrix A. OSE's are attractive because they can be generated independently, without examining A. This makes its easy to process any updates to A since we can just multiply the update vector or matrix by Π and add the result to our current compressed representation, ΠA, without recomputing Π or reapplying it to the full matrix A. In addition, very sparse constructions for Π are known, which allow fast computation of the embedded matrix ΠA when A is sparse (on the order of the number of non-zero entries in A).

We initially decided to look into using OSEs as an alternative to spectral sparsifiers for graph compression. As discussed above, a spectral sparsification algorithm is an algorithm for producing a row sampling matrix S that is a subspace embedding of the weighted vertex edge incidence matrix $W_G^{1/2}B$ of a graph G. Using OSEs for row sampling is hopeless. Imagine a graph with two large cliques connected by a single edge. In order to spectrally approximate $W_G^{1/2}B_G$ with high probability, a row sampling OSE has to select this edge with high probability (Note: the probability is over randomly generated instances of the OSE matrix Π). However, since the embedding is generated obliviously, it must work for any reordering of the rows in $W_G^{1/2}B_G$. So, it must select every row with high probability and hence cannot meaningfully reduce the number of rows in $W_G^{1/2}B_G$.

Despite this limitation, OSEs with very few nonzero entries per column are known. These OSE's produce an embedded matrix whose rows are a linear combination of a limited number of rows of the original matrix. Briefly, letting $m' < m$ be the dimension Π reduces A to, and s be the number of nonzero entries per column of Π, constructions are known with the following parameters ([**?**], [6]):

$$m' = O(d^2/\epsilon^2), s = 1$$
$$m' = O(d^{1+\gamma}/\epsilon^2), s = 1/\epsilon^2, \; for\ any\ \gamma > 0$$
$$m' = O(d\text{polylog}(d)/\epsilon^2), s = O(\text{polylog}(d)/\epsilon^2)$$

The parameters are quite remarkable - we can obliviously include each row of our original matrix only a constant $(1/\epsilon^2)$ number of times in the rows of a reduced matrix with only $O(d^{1+\gamma}/\epsilon^2)$, and still spectrally preserve the original matrix. However, consider even a non-existant ideal OSE that gives $m' = n, s = 1$. Then if we have a dense graph with $m = n^{(1+\gamma')}$ edges, we still have an average of $n^{\gamma'}$ nonzeros per row in $\tilde{W}^{1/2}\tilde{B} = \Pi(W^{1/2}B)$ - since each of the $n^{(1+\gamma')}$ rows of $W^{1/2}B$ has two nonzeros and is included once in $\tilde{W}^{1/2}\tilde{B}$, at the position that the nonzero in its corresponding column of Π falls.

This is as many nonzeros as are in the Laplacian matrix L_G, and in fact, as many nonzeros as in $W^{1/2}B$ originally, so $\tilde{W}^{1/2}\tilde{B}$ is not a compression in terms of storage space - just dimension.

It seems that, anything we can do with $\tilde{W}^{1/2}\tilde{B}$, which only preserves the spectral properties of $W^{1/2}B$, and not graph structure, we can do faster with L_G. For example, we could apply Π and then invert the covariance matrix of $\tilde{W}^{1/2}\tilde{B}$ to estimate the effective resistances of the edges in G. This takes time roughly $\tilde{O}(nnz(W^{1/2}B) + n^\omega) = \tilde{O}(n^2 + n^\omega) = \tilde{O}(n^\omega)$. To be competitive with using nearly linear time solvers to invert L_G and compute effective resistances, we would need to get this down to $\tilde{O}(n^2)$. However, there is a difficult to avoid bottleneck in the matrix inversion of $\tilde{B}^\top \tilde{W} \tilde{B}$, which no longer has the SDD structure of L_G. (Note that computing the Laplacian given $W^{1/2}B$ only takes time $O(nnz(W^{1/2}B)) = O(n^2)$).

Of course, we cannot definitively say that OSEs cannot be productively applied to graph matrices. However, these sort of obstacles led us to abandon this line of work. One attractive feature of OSEs is that they are useful for streaming computation. If an edge of G is updated, the update vector can be multiplied by Π and added to the compressed representation of the graph in constant time, maintaining the compression. Inspired by this we spent some time thinking about streaming computation on graphs - specifically about computing spectral sparsifiers in the semi-streaming model.

A.2 Streaming Sparsification

In the streaming graph model, you are not given a single data structure representing the graph G. Instead, input comes in sequentially in the form of edge insertions of deletions. After seeing a stream of insertions and deletions, you are required to compute a function of G. Work is done on both single-pass algorithms - in which you only see one pass of the input stream, and on multi-pass streaming algorithms. We will only consider single-pass algorithms below.

Traditionally, streaming algorithms are required to use space sublinear in the input size. However, under this restriction, it is difficult to compute anything meaningful about G. So the authors of [11] introduced the 'semi-streaming' model. In this model you are allowed $O(n\text{polylog}(n))$ space. Meaning that, in principle, you have enough space to store a spectral sparsfier of G. An algorithm for maintaining a sparsifer while processing edge insertions and deletions would provide a remarkably general tool for semi-streaming graph computation, since many problems could be reduced to maintaining a spectral sparisfier H and computing the output function of H to approximate the function of G.

As discussed above, OSEs won't directly work in the semi-streaming model - an obliviously compressed representation of $W^{1/2}B$ will take $O(n^{1+\gamma'})$ space. However, related methods using oblivious sketch matrices provide powerful tools for streaming processing. A sketch matrix S, like an OSE is just a linear transformation that can be quickly applied to update vectors as they come in so SA can be maintained as edges are inserted and deleted. Here A is some representation of G. For example, A might be $W^{1/2}B$.

A.2.1 Streaming Cut Sparsifiers

A single-pass algorithm for maintaining cut sparsifiers in the dynamic semi-streaming graph model was first presented in by Ahn, Guha, and McGregor in [1]. 'Dynamic' means that the algorithm handles both edge insertions and deletions. This algorithm relies on the finding in [12] that cut sparsifiers can be found by sampling edges according to their connectivity. This allows the authors to combine previous work on estimating edge connectivity in the dynamic semi-streaming model with sparse recovery algorithms so that they can approximately sample from $W^{1/2}B$ using edge connectivities and output a sparsifier using a representation of the graph that takes only $O(n\text{polylog}(n))$ space to store, and that can be maintained with edge insertions and deletions.

Another single-pass dynamic algorithm for cut sparsifiers is given by Goel, Ashish, and Kapralov in [13]. This algorithm also relies on edge sampling using connectivity.

A.2.2 Streaming Spectral Sparsifiers

Currently, no single-pass dynamic algorithm for spectral sparsifiers is known, however there have been efforts in find such algorithms. In [2], Ahn, Guha, and McGregor prove that the connectivity of an edge approximates its effective resistance by a multiplicative factor of $O(n^{2/3})$ - so a spectral sparsifier can be found by sampling edges by connectivities and oversampling by a factor of $O(n^{2/3})$. Using this fact, they directly apply their results on dynamic cut sparsifiers to maintain a spectral sparsifier with $O(n^{5/3}\log n/\epsilon^2)$ edges in $O(n^{5/3}\text{polylog}(n))$ space. Unfortunately they also show that their connectivity-resistance bound is tight - so achieving dynamic spectral sparsifiers in $O(n\text{polylog}(n))$ requires a new approach.

In [16], Kelner and Levin show that it is possible to maintain a spectral sparsifier in the insertion-only model. The basic idea is to collect edges until you have $c * n\log n/\epsilon^2$ for some fixed c. Upon reaching this bound, you sparsify this graph, reducing down to $c' * n\log n/\epsilon^2$ for $c' < c$. You then continue collecting edges, until you again have too many and need to resparsify again. It is possible to show that using the $(i-1)^{th}$ sparsifier to estimate effective resistance when computing the i^{th} sparsifier gives a sufficient estimation. As edges are added to the graph, effective resistances only go down, so using the estimated effective resistances of the graph before the most recent set of insertions will only cause oversampling. However, the oversampling will not be so large as to require sampling more than $O(n\log n/\epsilon^2)$ edges. Unfortunately, this strategy does not generalize well to the dynamic setting.

Kelner and Levin's algorithm only maintains a current sparsifier of G plus some recently inserted edges. It maintains no other information. However, in the dynamic setting, additional information must be stored in some form, since after deletions it may be necessary to add edges to the graph that were previously excluded. For example, consider two connected subgraphs each of size $n/2$, and interconnected by $n^2/4$ edges. It is not possible to save all these connecting edges in a sparsifier. However, if all but one of the edges between the cliques are deleted, it is necessary to include the remaining edge. Since an adversary can choose deletions to insure that the remaining edge is not one that you saved, you must have stored additional information that lets you recover this edge with high probability.

In an interesting effort to pin down more precisely what is required to compute spectral sparsifiers in the dynamic streaming model, Kapralov and Panigrahy show that a spectral sparsifier can be constructed in nearly linear time using the union of spanners of $O(\text{polylog}(n))$ random subgraphs of G [14]. This finding implies that if there were an algorithm for computing a $\log n$-spanner of G in the dynamic semi-streaming model, one could use this algorithm black box to get a semi-streaming dynamic spectral sparsification algorithm. No such algorithm is known to exist, however this connection gives another direction to look for streaming algorithms for spectral sparsifiers.