2014 IEEE Annual Symposium on Foundations of Computer Science

Single Pass Spectral Sparsification in Dynamic Streams

Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, Aaron Sidford
Massachusetts Institute of Technology, EECS and Mathematics
Cambridge, MA 02139, USA
Email: {kapralov,yintat,cpmusco,cnmusco,sidford}@mit.edu

***Abstract*—We present the first single pass algorithm for computing spectral sparsifiers of graphs in the dynamic semi-streaming model. Given a single pass over a stream containing insertions and deletions of edges to a graph G, our algorithm maintains a randomized linear sketch of the incidence matrix into dimension $O(\frac{1}{\epsilon^2} n \operatorname{polylog}(n))$. Using this sketch, the algorithm can output a $(1 \pm \epsilon)$ spectral sparsifier for the graph with high probability.**

While $O(\frac{1}{\epsilon^2} n \operatorname{polylog}(n))$ space algorithms are known for computing *cut sparsifiers* in dynamic streams [1], [2], and spectral sparsifiers in *insertion-only* streams [3], prior to our work, the best known single pass algorithm for maintaining spectral sparsifiers in dynamic streams required sketches of dimension $\Omega(\frac{1}{\epsilon^2} n^{5/3})$ [4].

To achieve our result, we show that, using a coarse sparsifier of G and a linear sketch of G's incidence matrix, it is possible to sample edges by effective resistance, obtaining a spectral sparsifier of arbitrary precision. Sampling from the sketch requires a novel application of ℓ_2/ℓ_2 sparse recovery, a natural extension of the ℓ_0 methods used for cut sparsifiers in [1]. Recent work of [5] on row sampling for matrix approximation gives a recursive approach for obtaining the required coarse sparsifiers.

Under certain restrictions, our approach also extends to the problem of maintaining a spectral approximation for a general matrix $A^\top A$ given a stream of updates to rows in A.

***Keywords*-streaming; sketching; spectral sparsification; sparse recovery; dimensionality reduction**

I. Introduction

A. The Dynamic Semi-Streaming Model

When processing massive graph datasets arising from social networks, web topologies, or interaction graphs, computation may be as limited by space as it is by runtime. To cope with this issue, one might hope to apply techniques from the streaming model of computation, which 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 [6] for an overview. However, the model has proven too restrictive for many graph algorithms. For example, testing $s-t$ connectivity requires $\Omega(n)$ space [7].

Thus, the less restrictive semi-streaming model, in which the algorithm is allowed $\tilde{O}(n)$ space [8], has received significant attention in recent years. In this model, a processor receives a stream of edges over a fixed set of n 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 [9]. 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, and spanners among other problems [1], [8]–[11]. In [12], McGregor surveys much of this progress and provides a more complete list of citations.

B. Streaming Sparsification

First introduced by Benczúr and Karger [13], a *cut sparsifier* of a graph G is a weighted subgraph with only $O(\frac{1}{\epsilon^2} n \operatorname{polylog}(n))$ edges that preserves the total edge weight over every cut in G to within a $(1 \pm \epsilon)$ multiplicative factor. Cut sparsifiers can be used to compute approximations for minimum cut, sparsest cut, maximum flow, and a variety of other problems. In [14], 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.

The problem of computing graph sparsifiers in the streaming model has received a lot of attention. Ahn and Guha give the first single pass, insertion-only algorithm for cut sparsifiers [15]. Kelner and Levin give a single pass, insertion-only algorithm for spectral sparsifiers [3]. This algorithm stores 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

0272-5428/14 $31.00 © 2014 IEEE
DOI 10.1109/FOCS.2014.66

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

In [4], 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 $O(\frac{1}{\epsilon^2}n^{5/3}\,\mathrm{polylog}(n))$ 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 $O(\frac{1}{\epsilon^2}n\,\mathrm{polylog}(n))$ 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 [12]. A two-pass algorithm for constructing a spectral sparsifier in the dynamic streaming model using $O\left(\frac{1}{\epsilon^2}n^{1+o(1)}\right)$ space is presented in [16]. The approach is very different from ours: it leverages a reduction from spanner constructions to spectral sparsification presented in [17]. It is not known if this approach extends to a space efficient single pass algorithm.

C. Our Contribution

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 VI, we show that it is easily extended to weighted graphs, as long as an edge's weight is specified when it is deleted. This model matches what is standard for dynamic cut sparsifiers [1], [2].

Theorem 1 (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 $O\left(\frac{1}{\epsilon^2}n\,\mathrm{polylog}(n)\right)$ space. From these sketches, it is possible to recover, with high probability, a weighted subgraph H with $O(\frac{1}{\epsilon^2}n\log n)$ edges such that H is a $(1\pm\epsilon)$ spectral sparsifier of G. The algorithm recovers H in $O\left(\frac{1}{\epsilon^2}n^2\,\mathrm{polylog}(n)\right)$ time.*

It is well known that independently sampling edges from a graph G according to their *effective resistances* gives a $(1\pm\epsilon)$ spectral sparsifier of G with $O(\frac{1}{\epsilon^2}n\log n)$ edges [18]. 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 ([1], [2]) and introducing a new sampling technique based on an ℓ_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 a preprint of a recent paper on iterative row sampling for matrix approximation [5].

The fact that our algorithm maintains a linear sketch of the streamed graph allows for the simple handling of edge deletions, which are treated as negative edge insertions. Additionally, due to their linearity, 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 procesors hold different subgraphs or each processes different edge substreams.

Our 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 ϵ factor, all spectral information about the Laplacian using just $O(\frac{1}{\epsilon^2}n\log n)$ space. 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, the non-adaptive linear sketches used in our algorithm are completely *oblivious*: at each edge insertion or deletion, we do not need to examine the current compression at all to make the appropriate update. As in sparse recovery or dimensionality reduction, we essentially just multiply the vertex edge incidence matrix by a random projection matrix, decreasing its height drastically in the process. Nevertheless, the oblivious compression obtained 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 $O(\frac{1}{\epsilon^2}n\log n)$ by log factors. Our result is the first of this kind in the spectral domain. The only other streaming algorithm for spectral sparsification that uses $O(\frac{1}{\epsilon^2}n\,\mathrm{polylog}(n))$ space is distinctly non-oblivious [3] and oblivious subspace embeddings for compressing general matrices inherently require $\Omega(n^2)$ space, even when the matrix is sparse (as in the case of an edge vertex incidence matrix) [19], [20].

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 $O(\frac{1}{\epsilon^2}n\,\mathrm{polylog}(n))$ sized sketch from which a spectral approximation to $A^\top A$ can be recovered. By structured, we mean any matrix whose rows are selected from some fixed dictionary of size $\mathrm{poly}(n)$. 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 $\binom{n}{2}$ edge rows that may appear in A and $A^\top A$

is the graph Laplacian.

D. Road Map

Section II Lay out notation, build linear algebraic foundations for spectral sparsification, and present lemmas for graph sampling and sparse recovery required by our algorithm.

Section III Give an overview of our central algorithm, providing intuition and motivation.

Section IV Present an algorithm of Miller and Peng ([5]) for building a chain of coarse sparsifiers and prove our main result, assuming a primitive for sampling edges by effective resistance in the streaming model.

Section V Develop this sampling primitive, our main technical contribution.

Section VI Show how to extend the algorithm to weighted graphs.

Section VII Show how to extend the algorithm to general structured matrices.

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

II. Notation and Preliminaries

A. Graph Notation

Let $\mathbf{B}_n \in \mathbb{R}^{\binom{n}{2}\times n}$ be the vertex edge incidence matrix of the undirected, unweighted complete graph over n vertices. $\mathbf{b}_e$, 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 any unweighted, undirected graph $G(V, E)$ as $\mathbf{B} = \mathbf{S}\mathbf{B}_n$ where $\mathbf{S}$ is an $\binom{n}{2} \times \binom{n}{2}$ diagonal matrix with ones at positions corresponding to edges contained in G and zeros elsewhere.[1] The $n \times n$ Laplacian matrix of G is given by $\mathbf{K} = \mathbf{B}^\top\mathbf{B}$.

B. Spectral Sparsification

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

Note that we are giving these definitions for a general matrix $\mathbf{B}$, but we will often work with a $\mathbf{B}$ that is the vertex edge incidence matrix of a graph G, with $\mathbf{K}$ is the graph Laplacian. We will not always require our approximation $\tilde{\mathbf{K}}$ to be the graph Laplacian of a weighted subgraph, which is a standard assumption. For this reason, we avoid the standard $\mathbf{L}_G$ notation for the Laplacian. For our purposes, $\tilde{\mathbf{K}}$ will always be a sparse symmetric diagonally dominant matrix, containing no more than $O(n \log n)$ 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 - the Laplacian matrix of a weighted subgraph of G.

[1] Typically the rows of $\mathbf{B}$ that are all 0 are removed, however we find this formulation more convenient for our purposes.

C. Leverage Scores and Row Sampling

For any $\mathbf{B} \in \mathbb{R}^{m\times n}$ with rank r, let $\mathbf{K}^+$ denote the Moore-Penrose pseudoinverse of $\mathbf{K} = \mathbf{B}^\top\mathbf{B}$. Consider the reduced singular value decomposition, $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. $\mathbf{U} \in \mathbb{R}^{m\times r}$ and $\mathbf{V} \in \mathbb{R}^{n\times r}$ have orthonormal columns and $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ is diagonal and contains the nonzero singular values of $\mathbf{B}$. $\mathbf{K} = \mathbf{B}^\top\mathbf{B} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top$. It follows that:

$$\mathbf{K}^+ = \mathbf{V}(\mathbf{\Sigma}^{-1})^2\mathbf{V}^\top$$

The leverage score, τ_i, for a row $\mathbf{b_i}$ in $\mathbf{B}$ is defined as:

$$\tau_i \stackrel{\text{def}}{=} \mathbf{b}_i^\top\mathbf{K}^+\mathbf{b}_i = \mathbf{u}_i^\top\mathbf{\Sigma}\mathbf{V}^\top(\mathbf{V}\mathbf{\Sigma}^{-2}\mathbf{V}^\top)\mathbf{V}\mathbf{\Sigma}\mathbf{u}_i = \|\mathbf{u}_i\|_2^2 \leq 1$$

The last inequality follows from the fact that every row in a matrix with orthonormal columns has norm less than 1. In a graph, $\tau_i = r_i w_i$, where r_i is the *effective resistance* of edge i and w_i is its weight. Furthermore:

$$\sum_{i=1}^{m} \tau_i = \text{tr}(\mathbf{B}\mathbf{K}^+\mathbf{B}^\top) = \|\mathbf{U}\|_F^2 = r = \text{rank}(\mathbf{B})$$

It is well known that by sampling the rows of $\mathbf{B}$ according to their leverage scores it is possible to obtain a matrix $\tilde{\mathbf{B}}$ such that $\tilde{\mathbf{K}} = \tilde{\mathbf{B}}^\top\tilde{\mathbf{B}} \approx_\epsilon \mathbf{K}$ with high probability. Furthermore, if obtaining exact leverage scores is computationally difficult, it suffices to sample by upper bounds on the scores. Typically, rows are sampled with replacement with probability proportional to their leverage score [18], [21]. We give an alternative independent sampling procedure based off the matrix concentration results of [22], which is more amenable to our application.

Lemma 1 (Spectral Sparsifier via Leverage Score Sampling)**.** *Let $\tilde{\boldsymbol{\tau}}$ be a vector of m estimated leverage scores for the rows of $\mathbf{B}$, such that $1 \geq \tilde{\tau}_i \geq \tau_i$ for all $i \in [m]$. For some known constant c, let $\mathbf{W}_1, \mathbf{W}_2, ...\mathbf{W}_{c\log n\epsilon^{-2}}$ be diagonal matrices with independently chosen entries such that $\mathbf{W}_j(i,i) = \frac{1}{\tilde{\tau}_i}$ with probability $\tilde{\tau}_i$ and $\mathbf{W}_j(i,i) = 0$ otherwise. Letting $\bar{\mathbf{W}} = \frac{1}{c\log n\epsilon^{-2}} \cdot \sum_j \mathbf{W}_j$ then with high probability,*

$$\tilde{\mathbf{K}} = \mathbf{B}^\top\bar{\mathbf{W}}\mathbf{B} \approx_\epsilon \mathbf{K}$$

Furthermore, $\bar{\mathbf{W}}$ has $O(\|\tilde{\boldsymbol{\tau}}\|_1 \log n\epsilon^{-2})$ nonzeros with high probability. That is, if we sample each each row of $\mathbf{B}$

independently with probability $\tilde{\tau}_i$, reweight selected rows by $\frac{1}{\sqrt{\tilde{\tau}_i}}$, and average over $c \log n \epsilon^{-2}$ trials, we obtain a matrix $\tilde{\mathbf{B}} = \bar{\mathbf{W}}^{1/2}\mathbf{B}$ such that $\tilde{\mathbf{B}}^\top \tilde{\mathbf{B}} = \mathbf{B}^\top \bar{\mathbf{W}} \mathbf{B} = \tilde{\mathbf{K}} \approx_\epsilon \mathbf{K}$ and $\tilde{\mathbf{B}}$ contains just $O(\|\tilde{\boldsymbol{\tau}}\|_1 \log n \epsilon^{-2})$ reweighted rows of $\mathbf{B}$ with high probability.

Lemma 1 follows from a standard Matrix Chernoff bound – for completeness a proof is included in [23]. We use a variant of Corollary 5.2 from [22], given by Harvey in [24].

D. Sparse Recovery

We now give a sparse recovery primitive that is used to sample edges from our linear sketches. We use an ℓ_2 heavy hitters algorithm that, for any vector $\mathbf{x}$, lets us recover from a small size sketch $\mathbf{\Phi x}$, the index i and the approximate value of $\mathbf{x}_i$ for all i such that $\mathbf{x}_i > \frac{1}{O(\text{polylog}(n))}||\mathbf{x}||_2$.

Lemma 2 (ℓ_2 Heavy Hitters)**.** *For each $\eta > 0$, there is a decoding algorithm D and a distribution on matrices $\mathbf{\Phi}$ in $\mathbb{R}^{O(\eta^{-2}\,\text{polylog}(N)) \times N}$ such that, for any $\mathbf{x} \in \mathbb{R}^N$, with probability $1 - N^{-c}$ over the choice of $\mathbf{\Phi}$, given $\mathbf{\Phi x}$, 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 $1 - N^{-c}$. The sketch $\mathbf{\Phi x}$ can be maintained and decoded in $O(\eta^{-2}\,\text{polylog}(N))$ space.

Note that setting $\eta = \frac{\epsilon}{C \log n}$ for any $0 < \epsilon < 1/2$ and $C > 0$, guarantees that $\mathbf{w}_i$ must be a $(1 \pm \epsilon)$ approximation of $\mathbf{x}_i$ for any i with $\mathbf{x}_i \geq \frac{1}{C \log n}\|\mathbf{x}\|_2$. It also guarantees that we can distinguish using $\mathbf{w}_i$ whether $\mathbf{x}_i \geq \frac{1}{C \log n}\|\mathbf{x}\|_2$ or $\mathbf{x}_i < \frac{1}{2C \log n}\|\mathbf{x}\|_2$. Lemma 2 is based on an ℓ_2/ℓ_2 sparse recovery result given in [25] – we include the full reduction in the full version [23].

III. Algorithm Overview

Before providing a formal presentation and proof of our main result, Theorem 1, we would like to give an informal overview of the algorithm to provide intuition.

A. Effective Resistances

As explained in Section II-C, spectral sparsifiers can be generated by sampling edges, i.e. rows of the vertex edge incidence matrix. For an unweighted graph G, each edge is sampled independently with probability equal to its leverage score, τ_e. After appropriate repetition of the sampling, we reweight and combine any sampled edges. The result is a subgraph of G containing, with high probability, $O(\frac{1}{\epsilon^2} n \log n)$ edges and spectrally approximating G.

If we view 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:

$$\tau_e = \mathbf{b}_e^\top \mathbf{K}^+ \mathbf{b}_e$$

Note that τ_e can be computed for any pair of vertices, (i, j), or in other words, for any possible edge in G. We can evaluate $\mathbf{b}_e^\top \mathbf{K}^+ \mathbf{b}_e$ 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.

B. 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 τ_e for any e. Known sampling algorithms, including our Lemma 1, 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 [5]. We will give the overview shortly but, for now, assume that we have access to a coarse sparsifier, $\tilde{\mathbf{K}} \approx_{1/2} \mathbf{K}$. Computing $\mathbf{b}_e^\top \tilde{\mathbf{K}}^+ \mathbf{b}_e$ gives a 2 factor multiplicative approximation of τ_e for each e. Furthermore, as long as $\tilde{\mathbf{K}}$ has sparsity $O(n\,\text{polylog}(n))$, the computation can be done in small space using any nearly linear time solver for symmetric diagonally dominant linear systems (e.g. [26]).

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

$$\tau_e = \mathbf{b}_e^\top \mathbf{K}^+ \mathbf{K} \mathbf{K}^+ \mathbf{b}_e = \|\mathbf{B}\mathbf{K}^+ \mathbf{b}_e\|_2^2 = \|\mathbf{S}\mathbf{B}_n \mathbf{K}^+ \mathbf{b}_e\|_2^2$$

Referring to Section II, recall that $\mathbf{B} = \mathbf{S}\mathbf{B}_n$ is exactly the same as a standard vertex edge incidence matrix except that rows in $\mathbf{B}_n$ corresponding to nonexistent edges are zeroed out instead of removed. Denote $\mathbf{x}_e = \mathbf{S}\mathbf{B}_n \mathbf{K}^+ \mathbf{b}_e$. Each nonzero entry in $\mathbf{x}_e$ 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, the e^{th} entry of $\mathbf{x}_e$, $\mathbf{x}_e(e)$, is 0. However, if e is in G, then $\mathbf{x}_e(e)$ is τ_e. Furthermore, $\|\mathbf{x}_e\|_2^2 = \tau_e$. So, if we could access a sketch of $\mathbf{x}_e$, could we determine whether or not $e \in G$ using our ℓ_2 sparse recovery primitive?

Not quite - to determine whether an index in $\mathbf{x}_e$ is nonzero, the recovery primitive, Lemma 2, requires it to

account for an $O(1/\operatorname{polylog}(n))$ fraction of the total ℓ_2 norm. Currently, $\mathbf{x}_e(e)/\|\mathbf{x}_e\|_2 = \sqrt{\tau_e}$, which could be much smaller than $O(1/\log n)$. However, suppose we had a sketch of $\mathbf{x}_e$ with all but τ_e fraction of edges randomly sampled out. Then, we would expect $\|\mathbf{x}_e\|_2^2 \approx \tau_e^2$ and, in fact, we can show that it would equal $o(\tau_e \log n)$ with high probability. Thus, $\mathbf{x}_e(e)/\|\mathbf{x}_e\|_2 = \Omega(1/\operatorname{polylog}(n))$ and sparse recovery would successfully indicate whether or not $e \in G$. What's more, randomly zeroing out edges of $\mathbf{x}_e$ can serve as our main sampling routine for edge e. This process will set $\mathbf{x}_e(e) = 0$ with probability $(1 - \tau_e)$, exactly what we wanted to sample by in the first place.

However, how do we go about sketching every appropriately sampled $\mathbf{x}_e$? Well, consider subsampling our graph at geometrically decreasing rates, $1/2^s$ for $s \in \{0, 1, ...O(\log n)\}$. Maintain linear sketches $\mathbf{\Pi}_1\mathbf{B}_1, ...\mathbf{\Pi}_{O(\log n)}\mathbf{B}_{O(\log n)}$ of the vertex edge incidence matrix for every subsampled graph using the ℓ_2 sparse recovery sketch distribution from Lemma 2. When asked to output a spectral sparsifier, for every possible edge e, we compute its approximate effective resistance τ_e using $\tilde{\mathbf{K}}$ and determine a rate $1/2^s$ that approximates τ_e.

Next, since our sketches are linear, for every edge, we can just multiply $\mathbf{\Pi}_{1/2^s}\mathbf{B}_{1/2^s}$ on the right by $\tilde{\mathbf{K}}^+\mathbf{b}_e$. We get:

$$\mathbf{\Pi}_{1/2^s}\mathbf{B}_{1/2^s}\tilde{\mathbf{K}}^+\mathbf{b}_e \approx \mathbf{\Pi}_{1/2^s}\mathbf{x}_e^{1/2^s}$$

where $\mathbf{x}_e^{1/2^s}(e)$ is $\mathbf{x}_e$ sampled at rate $1/2^s \approx \tau_e$. This sketch is equivalent to what would be obtained if we had been able to sketch $\mathbf{x}_e^{1/2^s}$ in the first place. Thus, as explained, we can just 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!

C. A Chain of Coarse Sparsifiers

The final required component is access to some sparse $\tilde{\mathbf{K}} \approx_{1/2} \mathbf{K}$. This coarse sparsifier is obtained recursively by constructing a chain of matrices, $\left[\mathbf{K}(0), \mathbf{K}(1), \ldots, \mathbf{K}(d), \mathbf{K}\right]$ each weakly approximating the next. Specifically, imagine producing $\mathbf{K}(d)$ by adding a fairly light identity matrix to $\mathbf{K}$. As long as the identity's weight is small compared to $\mathbf{K}$'s spectrum, $\mathbf{K}(d)$ approximates $\mathbf{K}$. Add even more weight to the diagonal to form $\mathbf{K}(d-1)$. Again, as long as the increase is small, $\mathbf{K}(d-1)$ approximates $\mathbf{K}(d)$. We continue down the chain until $\mathbf{K}(0)$, which will actually have a heavy diagonal after all the incremental increases. Thus, $\mathbf{K}(0)$ can be approximated by an appropriately scaled identity matrix, which is clearly sparse. Miller and Peng show that parameters can be chosen such that $d = O(\log n)$ [5].

Putting everything together, we maintain $O(\log n)$ sketches for $\left[\mathbf{K}(0), \mathbf{K}(1), \ldots, \mathbf{K}(d), \mathbf{K}\right]$. We first use a weighted identity matrix as a coarse approximation for $\mathbf{K}(0)$, which allows us to recover a good approximation to $\mathbf{K}(0)$ from our sketch. This approximation will in turn be a coarse approximation for $\mathbf{K}(1)$, so we can recover a good sparsifier of $\mathbf{K}(1)$. Continuing up the chain, we eventually recover a good sparsifier for our final matrix, $\mathbf{K}$. This approach is formalized in the next section.

IV. Recursive Sparsifier Construction

In this section, we describe the recursive procedure for obtaining a chain of coarse sparsifiers using a technique introduced by Miller and Peng - "Introduction and Removal of Artificial Bases" [5]. We then formally prove Theorem 1 by combining this technique with the sampling algorithm developed in Section V.

Theorem 2 (Recursive Sparsification ([5], Section 4))**.** *Consider any PSD matrix $\mathbf{K}$ with maximum eigenvalue bounded from above by λ_u and minimum nonzero eigenvalue bounded from below by λ_l. Let $d = \lceil \log_2(\lambda_u/\lambda_l) \rceil$. For $\ell \in \{0, 1, 2, ..., d\}$, define:*

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

So, $\gamma(d) \leq \lambda_l$ and $\gamma(0) = \lambda_u$. Then the chain of PSD matrices, $\left[\mathbf{K}(0), \mathbf{K}(1), \ldots, \mathbf{K}(d)\right]$ with:

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

satisfies the following relations:

1) $\mathbf{K} \preceq_r \mathbf{K}(d) \preceq_r 2\mathbf{K}$
2) $\mathbf{K}(\ell) \preceq \mathbf{K}(\ell-1) \preceq 2\mathbf{K}(\ell)$ *for all* $\ell \in \{1, \ldots, d\}$
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, $\lambda_{max} < 2n$ and $\lambda_{min} > 8/n^2$ (where here λ_{min} is the smallest nonzero *eigenvalue). Thus the length of our chain, $d = \lceil \log_2 \lambda_u/\lambda_l \rceil$, is $O(\log n)$.*

For completeness, we've included a proof of Theorem 2 in the full version of the paper [23]. Now, to prove our main result, we need to state the sampling primitive for streams that will be developed in Section V. 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 1, to obtain a better sparsifier of $\mathbf{K}(\ell)$.

Theorem 3. *Let $\mathbf{B} \in \mathbb{R}^{n\times m}$ be the vertex edge incidence matrix of an unweighted graph G, specified by an insertion-deletion graph stream. Let $\gamma = O(\operatorname{poly} n)$ be a fixed parameter and consider $\mathbf{K} = \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I}$. For any $0 < \epsilon < 1$, there exists a sketching procedure* `MaintainSketches`$(\mathbf{B}, \epsilon)$ *that outputs an $O(n \operatorname{polylog}(n))$ sized sketch $\mathbf{\Pi}\mathbf{B}$.*

There exists a corresponding recovery algorithm `RefineSparsifier`*, such that, if $\tilde{\mathbf{K}}$ is a spectral approximation to $\mathbf{K}$ with $O(n \operatorname{polylog}(n))$ nonzeros and $c\mathbf{K} \preceq_r \tilde{\mathbf{K}} \preceq_r \mathbf{K}$ for some constant $0 < c < 1$ then:*

`RefineSparsifier`$(\mathbf{\Pi}\mathbf{B}, \tilde{\mathbf{K}}, \gamma, \epsilon, c)$ *returns, with high probability, $\tilde{\mathbf{K}}_\epsilon = \tilde{\mathbf{B}}_\epsilon^\top\tilde{\mathbf{B}}_\epsilon + \gamma\mathbf{I}$, where*

$(1-\epsilon)\mathbf{K} \preceq_r \tilde{\mathbf{K}}_\epsilon \preceq_r (1+\epsilon)\mathbf{K}$, *and* $\tilde{\mathbf{B}}_\epsilon$ *contains only* $O(\epsilon^{-2}c^{-1}n\log n)$ *reweighted rows of* $\mathbf{B}$ *with high probability.* `RefineSparsifier` *runs in* $O(n^2\,\mathrm{polylog}(n))$ *time.*

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

Proof of Theorem 1: Let $\mathbf{K}$ be the Laplacian of our graph G. Process all edge insertions and deletions, using `MaintainSketches` to produce a separate sketch, $(\mathbf{\Pi B})_\ell$ for each $\ell \in \{0, 1, \ldots, \lceil \log_2 \lambda_u/\lambda_l \rceil + 1\}$.

We can use Theorem 3 to recover an ϵ approximation, $\tilde{\mathbf{K}}(\ell)$, for any $\mathbf{K}(\ell)$ given an ϵ approximation for $\mathbf{K}(\ell-1)$. First, consider the base case, $\mathbf{K}(0)$. Let:

$$\tilde{\mathbf{K}}(0) = \texttt{RefineSparsifier}((\mathbf{\Pi B})_0, \gamma(0)\mathbf{I}, \gamma(0), \epsilon, \frac{1}{2})$$

By Theorem 2, Relation 3:

$$\frac{1}{2}\mathbf{K}(0) \preceq \gamma(0)\mathbf{I} \preceq \mathbf{K}(0)$$

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

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

$$\tilde{\mathbf{K}}(\ell) = \texttt{RefineSparsifier}\left((\mathbf{\Pi B})_\ell, \frac{1}{2(1+\epsilon)}\tilde{\mathbf{K}}(\ell-1), \gamma(\ell), \epsilon, \frac{1-\epsilon}{2(1+\epsilon)}\right)$$

By Theorem 2, Relation 2:

$$\frac{1}{2}\mathbf{K}(\ell) \preceq \frac{1}{2}\mathbf{K}(\ell-1) \preceq \mathbf{K}(\ell)$$

Furthermore, by assumption we have the inequalities:

$$\frac{1-\epsilon}{1+\epsilon}\mathbf{K}(\ell-1) \preceq_r \frac{1}{1+\epsilon}\tilde{\mathbf{K}}(\ell-1) \preceq_r \mathbf{K}(\ell-1)$$

Thus:

$$\frac{1-\epsilon}{2(1+\epsilon)}\mathbf{K}(\ell) \preceq_r \frac{1}{2(1+\epsilon)}\tilde{\mathbf{K}}(\ell-1) \preceq_r \mathbf{K}(\ell)$$

So, with high probability `RefineSparsifier` returns $\tilde{\mathbf{K}}(\ell)$ such that $(1-\epsilon)\mathbf{K}(\ell) \preceq_r \tilde{\mathbf{K}}(\ell) \preceq_r (1+\epsilon)\mathbf{K}(\ell)$ and $\tilde{\mathbf{K}}(\ell)$ contains just $O((\frac{2(1+\epsilon)}{1-\epsilon})^2\epsilon^{-2}n\log n) = O(\epsilon^{-2}n\log n)$ nonzero elements. It is important to note that there is no "compounding of error" in this process. Every $\tilde{\mathbf{K}}(\ell)$ is an ϵ approximation for $\mathbf{K}(\ell)$. Error from using $\tilde{\mathbf{K}}(\ell-1)$ instead of $\mathbf{K}(\ell-1)$ is absorbed by a constant factor increase in the number of rows sampled from $\mathbf{B}$. The corresponding increase in sparsity for $\mathbf{K}(\ell)$ does not compound - in fact Theorem 3 is completely agnostic to the sparsity of the coarse approximation $\tilde{\mathbf{K}}$ used.

Finally, to obtain a bonafide spectral graph sparsifier (a weighted subgraph of our streamed graph), let:

$$\tilde{\mathbf{K}} = \texttt{RefineSparsifier}\left((\mathbf{\Pi B})_{d+1}, \frac{1}{2(1+\epsilon)}\tilde{\mathbf{K}}(d), 0, \epsilon, \frac{1-\epsilon}{2(1+\epsilon)}\right)$$

As in the inductive case,

$$\frac{1-\epsilon}{2(1+\epsilon)}\mathbf{K} \preceq_r \frac{1}{2(1+\epsilon)}\tilde{\mathbf{K}}(d) \preceq_r \mathbf{K}$$

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

$$(1-\epsilon)\mathbf{K} \preceq \tilde{\mathbf{K}} \preceq (1+\epsilon)\mathbf{K}$$

We conclude that $\tilde{\mathbf{K}}$ is the Laplacian of some graph H containing $O(\epsilon^{-2}n\log n)$ rescaled edges of G and approximating G spectrally to precision ϵ. Finally, note that we only required $d+1 = O(\log n)$ recovery steps, each running in $O(n^2\,\mathrm{polylog}(n))$ time. Thus, the complete recovery time is $O(n^2\,\mathrm{polylog}(n))$. ∎

V. Streaming Row Sampling

In this section, we develop the sparsifier refinement subroutine required for the proof of Theorem 1 in Section IV.

Proof of Theorem 3:

Outside of the streaming model, given full access to $\mathbf{B}$ rather than just a sketch $\mathbf{\Pi B}$ it is easy to implement `RefineSparsifier` via leverage score sampling. Letting $\oplus$ denote appending the rows of one matrix to another, we can define $\mathbf{B}_\gamma = \mathbf{B} \oplus \sqrt{\gamma(\ell)}\cdot\mathbf{I}$, so $\mathbf{K} = \mathbf{B}^\top\mathbf{B} + \gamma\mathbf{I} = \mathbf{B}_\gamma^\top\mathbf{B}_\gamma$. Since $\tau_i = \mathbf{b}_i^\top\mathbf{K}^+\mathbf{b}_i$ and $c\mathbf{K} \preceq_r \tilde{\mathbf{K}} \preceq_r \mathbf{K}$, for any row of $\mathbf{B}_\gamma$ we have

$$\tau_i \leq \mathbf{b}_i^\top\tilde{\mathbf{K}}^+\mathbf{b}_i \leq \frac{1}{c}\tau_i.$$

Let $\tilde{\tau}_i = \mathbf{b}_i^\top\tilde{\mathbf{K}}^+\mathbf{b}_i$ be the leverage score of $\mathbf{b}_i$ approximated using $\tilde{\mathbf{K}}$. Let $\tilde{\boldsymbol{\tau}}$ be the vector of approximate leverage scores, with the leverage scores of the n rows corresponding to $\sqrt{\gamma(\ell)}\cdot\mathbf{I}$ rounded up to 1. This will include the rows of the identity with probability 1 in each independent sampling. While not strictly necessary, doing so simplifies our analysis in the streaming setting. Using this $\tilde{\boldsymbol{\tau}}$ in Lemma

1, we can obtain $\tilde{\mathbf{K}}_\epsilon \approx_\epsilon \mathbf{K}$ with high probability. Since $\|\tilde{\boldsymbol{\tau}}\|_1 \leq \frac{1}{c}\|\boldsymbol{\tau}\|_1 + n \leq \frac{1}{c} \cdot \text{rank}(\mathbf{B}) + n \leq \frac{n+1}{c}$, we can write $\tilde{\mathbf{K}}_\epsilon = \tilde{\mathbf{B}}_\epsilon^\top \tilde{\mathbf{B}}_\epsilon + \gamma \mathbf{I}$, where $\tilde{\mathbf{B}}_\epsilon$ contains $O(\epsilon^{-2}c^{-1}n \log n)$ reweighted rows of $\mathbf{B}$ with high probability.

The challenge in the semi-streaming setting is actually performing the independent edge sampling given only a sketch of $\mathbf{B}$. The general idea is explained in our overview Section III, with detailed pseudocode included below. We show that each required computation is possible in the dynamic semi-streaming model, and then prove the correctness of the sampling procedure.

Streaming Sparsifier Refinement

`MaintainSketches`$(\mathbf{B}, \epsilon)$:

1) For $j \in 1, 2, ...c_0 \frac{1}{\epsilon^2} \log n$
   a) For $s \in \{1, ...O(\log n)\}$ let $h_s : E \to \{0, 1\}$ be a uniform hash function. Let $\mathbf{B}_s$ be $\mathbf{B}$ with all rows except those with $\prod_{j \leq s} h_j(e) = 0$ zeroed out. So $\mathbf{B}_s$ is $\mathbf{B}$ with rows sampled independently at rate $\frac{1}{2^s}$. $\mathbf{B}_0$ is simply $\mathbf{B}$.
   b) Maintain $O(\log n)$ sketches $\boldsymbol{\Pi}_0\mathbf{B}_0, \boldsymbol{\Pi}_1\mathbf{B}_1, ..., \boldsymbol{\Pi}_{O(\log n)}\mathbf{B}_{O(\log n)}$ where $\{\boldsymbol{\Pi}_0, \boldsymbol{\Pi}_1, ...\boldsymbol{\Pi}_{O(\log n)}\}$ are drawn from the distribution from Lemma 2 with $\eta = \frac{1}{4c_1 \log n}$.

`RefineSparsifier`$(\boldsymbol{\Pi}\mathbf{B}, \tilde{\mathbf{K}}, \gamma, \epsilon, c)$:

1) For $j \in 1, 2, ...c_0 \frac{1}{\epsilon^2} \log n$
   a) Compute $\boldsymbol{\Pi}_s\mathbf{B}_s\tilde{\mathbf{K}}^+$ for each $s \in \{0, 1, 2, ...O(\log n)\}$.
   b) For each possible edge e:
      i) Compute $\tilde{\tau}_e = \mathbf{b}_e^\top \tilde{\mathbf{K}}^+ \mathbf{b}_e$. Choose s such that $\min\{1, \tilde{\tau}_e\} \leq \frac{1}{2^s} \leq 2 \cdot \min\{1, \tilde{\tau}_e\}$.
      ii) Compute the vector $\boldsymbol{\Pi}_s\mathbf{x}_e = \boldsymbol{\Pi}_s\mathbf{B}_s\tilde{\mathbf{K}}^+\mathbf{b}_e$, and perform the heavy hitters algorithm of Lemma 2, recovering with high probability elements with a $\geq \frac{1}{c_1 \log n}$ fraction of the ℓ_2 weight of $\mathbf{x}_e$, and throwing out any recovered elements with a $< \frac{1}{2c_1 \log n}$ fraction of the weight.
      iii) If $\mathbf{x}_e(e)$ is recovered set $\mathbf{W}_j(e, e) = 2^s$
2) Set $\bar{\mathbf{W}} = \frac{1}{c_0 \epsilon^{-2} \log n} \sum_j \mathbf{W}_j$ and output $\tilde{\mathbf{K}}_\epsilon = \mathbf{B}^\top \bar{\mathbf{W}} \mathbf{B} + \gamma \mathbf{I}$.

Implementation Details in the Semi-Streaming Model.: Note that the iterations of main loop of `MaintainSketches` can be done simultaneously in a single pass over the data stream. The sketches $\boldsymbol{\Pi}_0\mathbf{B}_0, \ldots, \boldsymbol{\Pi}_{O(\log n)}\mathbf{B}_{O(\log n)}$ can be stacked and the entire $O(n \,\text{polylog}(n))$ sized compression is output as $\boldsymbol{\Pi}\mathbf{B}$.

`MaintainSketches` requires $O(n \,\text{polylog}(n))$ space in total, and can be implemented in the dynamic streaming model. When an edge insertion comes in, use $\{h_s\}$ to compute which $\mathbf{B}_s$'s should contain the inserted edge, and update the corresponding sketches. An edge deletion can be performed simply by updating the sketches to reflect adding $-\mathbf{b}_e$ to $\mathbf{B}_s$.

Step 1(a) of `RefineSparsifier` can also be implemented in $O(n \,\text{polylog}\, n)$ space. Since $\tilde{\mathbf{K}}$ has $O(n \,\text{polylog}\, n)$ nonzeros and since each $\boldsymbol{\Pi}_s\mathbf{B}_s$ has $O(\text{polylog}\, n)$ rows, this step simply requires solving $O(\text{polylog}\, n)$ linear systems on $\tilde{\mathbf{K}}$, which can be performed in $O(n \,\text{polylog}\, n)$ time by using a nearly linear time SDD system solver [26]. From this time bound we immediately know that this computation can be performed in $O(n \,\text{polylog}\, n)$ space.

In step 1(b)i. it is always possible to choose an appropriate s with $\min\{1, \tilde{\tau}_e\} \leq \frac{1}{2^s} \leq 2 \cdot \min\{1, \tilde{\tau}_e\}$. $\lambda_{max}(\mathbf{K}) \leq n + \gamma = O(\text{poly}(n))$. So $\lambda_{min}(\tilde{\mathbf{K}}^+) = \Omega(\text{poly}(n))$ so $\tilde{\tau}_e = \Omega(\text{poly}(n))$ for all e. So such an s always can be found if we have $O(\log n)$ samplings of $\mathbf{B}$.

Finally, with high probability, when running Step 1(b) for each edge, in total we only ever recover $O(n \log n)$ edges and so can store them in small space.

Correctness: We need to show that, with high probability, in each round of sampling, this algorithm independently samples each row of $\mathbf{B}$ with probability $\hat{\tau}_e$ where $\min\{1, \tilde{\tau}_e\} \leq \hat{\tau}_e \leq 2 \cdot \min\{1, \tilde{\tau}_e\}$. Given this fact, since the algorithm samples the n rows of $\sqrt{\gamma} \cdot \mathbf{I}$ with probability 1, and since $\tau_e \leq \min\{1, \tilde{\tau}_e\} \leq \frac{1}{c}\tau_e$ for all e, by Lemma 1, with high probability, $\tilde{\mathbf{K}}_\epsilon \approx_\epsilon \mathbf{K}$ and $\tilde{\mathbf{K}}_\epsilon = \mathbf{B}\bar{\mathbf{W}}\mathbf{B} + \gamma\mathbf{I} = \tilde{\mathbf{B}}_\epsilon^\top \tilde{\mathbf{B}}_\epsilon + \gamma\mathbf{I}$, where $\tilde{\mathbf{B}}_\epsilon$ contains $O(\epsilon^{-2}c^{-1}n \log n)$ reweighted rows of $\mathbf{B}$.

In the above algorithm, an edge is only included in $\tilde{\mathbf{K}}_\epsilon$ if it is included in the sampled matrix $\mathbf{B}_{s(e)}$ where

$$\min\{1, \tilde{\tau}_e\} \leq \frac{1}{2^{s(e)}} \leq 2 \cdot \min\{1, \tilde{\tau}_e\}$$

The probability of $\mathbf{b}_e$ being included in $\mathbf{B}_{s(e)}$ is simply $1/2^{s(e)}$, and sampling is done independently using uniform random hash functions. So, as long as we can show that with high probability, all $\mathbf{b}_e$ are recovered by the sparse recovery procedure if included in their respective $\mathbf{B}_{s(e)}$, then we are done.

Let $\mathbf{x}_e = \mathbf{B}\tilde{\mathbf{K}}^+\mathbf{b}_e$ and $\mathbf{x}_e^{s(e)} = \mathbf{B}_{s(e)}\tilde{\mathbf{K}}^+\mathbf{b}_e$. If e is not an edge in the original graph or $\mathbf{b}_e$ is not included in $\mathbf{B}_{s(e)}$ then $\mathbf{x}_e^{s(e)}(e) = 0$, so if index e is recovered, it will be discarded. We need to argue that, if $\mathbf{b_e}$ is in fact included in $\mathbf{B}_{s(e)}$, with high probability, $\|\mathbf{x}_e^{s(e)}\|^2$ is not too large, so we are able to identify $\mathbf{x}_e^{s(e)}(e)$. We have:

$$\mathbf{x}_e^{s(e)}(e) = \mathbf{x}_e(e) = \mathbf{1}_e\mathbf{B}\tilde{\mathbf{K}}^+\mathbf{b}_e = \mathbf{b}_e^\top\tilde{\mathbf{K}}^+\mathbf{b}_e = \tilde{\tau}_e \quad (1)$$

Further, we can compute:

$$\begin{aligned}\|\mathbf{x}_e\|^2 &= \mathbf{b}_e^\top \tilde{\mathbf{K}}^+ \mathbf{B}^\top \mathbf{B} \tilde{\mathbf{K}}^+ \mathbf{b}_e \\ &\leq \mathbf{b}_e^\top \tilde{\mathbf{K}}^+ \mathbf{B}_\gamma^\top \mathbf{B}_\gamma \tilde{\mathbf{K}}^+ \mathbf{b}_e \quad \text{(Since } \mathbf{B}^\top\mathbf{B} \preceq \mathbf{B}_\gamma^\top\mathbf{B}_\gamma\text{)} \\ &\leq \frac{1}{c} \cdot \mathbf{b}_e^\top \tilde{\mathbf{K}}^+ \mathbf{b}_e \quad \text{(Since } c\left(\mathbf{B}_\gamma^\top\mathbf{B}_\gamma\right) \preceq \tilde{\mathbf{K}}\text{)} \\ &\leq \frac{1}{c}\tilde{\tau}_e\end{aligned}$$

For any edge $e' \neq e$ we define:

$$\tilde{\tau}_{e',e} \stackrel{\text{def}}{=} \mathbf{x}_e^{s(e)}(e') = \mathbf{1}_{e'}\mathbf{B}\tilde{\mathbf{K}}^+\mathbf{b}_e = \mathbf{b}_{e'}^\top\tilde{\mathbf{K}}^+\mathbf{b}_e$$

Lemma 3. $\tilde{\tau}_{e',e} \leq \tilde{\tau}_e$

Proof: Consider $\tilde{\mathbf{v}}_e = \tilde{\mathbf{K}}^+\mathbf{b}_e$. When $\mathbf{K}^+$ is a graph Laplacian $\tilde{\mathbf{v}}_e$ can be interpreted as the approximate voltages induced over each vertex when we treat our edges as resistors and route one unit of current between the endpoints of e. Letting $e = (u_1, u_2)$ and $e' = (u_1', u_2')$, if we have $|\tilde{\mathbf{v}}_e(u_1') - \tilde{\mathbf{v}}_e(u_2)'| \leq |\tilde{\mathbf{v}}_e(u_1) - \tilde{\mathbf{v}}_e(u_2)|$ then:

$$\mathbf{b}_{e'}^\top\tilde{\mathbf{v}}_e = \mathbf{b}_{e'}^\top\tilde{\mathbf{K}}^+\mathbf{b}_e \leq \mathbf{b}_e^\top\tilde{\mathbf{K}}^+\mathbf{b}_e = \mathbf{b}_e^\top\tilde{\mathbf{v}}_e$$

So:

$$\tilde{\tau}_{e',e} \leq \tilde{\tau}_e$$

Now, $\tilde{\mathbf{K}}$ is a weighted graph Laplacian added to a weighted identity matrix. So it is full rank and diagonally dominant. So $\tilde{\mathbf{K}}\tilde{\mathbf{v}}_e = \tilde{\mathbf{K}}\tilde{\mathbf{K}}^+\mathbf{b}_e = \mathbf{b}_e$

Since $\tilde{\mathbf{K}}$ is diagonally dominant and since $\mathbf{b}_e$ is zero everywhere except at $\mathbf{b}_e(u_1) = 1$ and $\mathbf{b}_e(u_2) = -1$, it must be that $\tilde{\mathbf{v}}_e(u_1)$ is the maximum value of $\tilde{\mathbf{v}}_e$ and $\tilde{\mathbf{v}}_e(u_2)$ is the minimum value. So $|\tilde{\mathbf{v}}_e(u_1') - \tilde{\mathbf{v}}_e(u_2)'| \leq |\tilde{\mathbf{v}}_e(u_1) - \tilde{\mathbf{v}}_e(u_2)|$ and $\tilde{\tau}_{e',e} \leq \tilde{\tau}_e$.

■

Now we upper bound the probability that in step (b)ii of `RefineSparsifier` we can't recover edge e from $\mathbf{B}_{s(e)}$ given that it is included in the sample.

$$\begin{aligned}&\mathbb{P}\left(\frac{\mathbf{x}_e^{s(e)}(e)^2}{\|\mathbf{x}_e^{s(e)}\|^2} < \frac{1}{c_1 \cdot \log n}\middle| e \in \mathbf{B}_{s(e)}\right) \\ &= \mathbb{P}\left(\|\mathbf{x}_e^{s(e)}\|^2 > c_1 \log n \cdot \tilde{\tau}_e^2\right) \\ &= \mathbb{P}\left(\|\frac{1}{\tilde{\tau}_e}\mathbf{x}_e^{s(e)}\|^2 > c_1 \log n\right)\end{aligned}$$

Note that the vector $\frac{1}{\tilde{\tau}_e}\mathbf{x}_e^{s(e)}$ has all entries (and thus all squared entries) in $[0, 1]$ (by Lemma 3) so we can apply a Chernoff bound to show concentration for its norm. Specifically, we will use a common multiplicative bound [27]:

$$\mathbb{P}(X > (1+\delta)\,\mathbb{E}\,X) < e^{-\frac{\delta^2}{2+\delta}\,\mathbb{E}\,X} \tag{2}$$

Recall that:

$$\mathbb{E}\,\|\frac{1}{\tilde{\tau}_e}\mathbf{x}_e^{s(e)}\|^2 = \frac{1}{2^{s(e)}} \cdot \frac{\tilde{\tau}_e}{c} \cdot \frac{1}{\tilde{\tau}_e^2} \leq \frac{2}{c} = \Theta(1) \tag{3}$$

which gives:

$$\begin{aligned}&\mathbb{P}\left(\|\frac{1}{\tilde{\tau}_e}\mathbf{x}_e^{s(e)}\|^2 > c_1 \log n\right) \\ &\leq \mathbb{P}\left(\|\frac{1}{\tilde{\tau}_e}\mathbf{x}_e^{s(e)}\|^2 > \frac{c_1 c \log n}{2}\,\mathbb{E}\,\|\frac{1}{\tilde{\tau}_e}\mathbf{x}_e^{s(e)}\|^2\right) \\ &= O(n^{-\Theta(1)})\end{aligned}$$

since $\delta = \Theta(\log n)$.

Recall that c is the constant determined by our input coarse sparsifier and c_1 can be chosen by implementing our sparse recovery routine with a different parameter. If we set c_1 large enough, as long as edge e is included in $\mathbf{B}_{s(e)}$, it is recovered with high probability. This guarantee holds for all $\binom{n}{2}$ possible edges with high probability by a union bound. So with high probability, our sampling process is exactly equivalent to independently sampling each edge with probability $\frac{1}{2^{s(e)}}$ where $\min\{1, \tilde{\tau}_e\} \leq \frac{1}{2^{s(e)}} \leq 2 \cdot \min\{1, \tilde{\tau}_e\}$. So our algorithm returns the desired $\tilde{\mathbf{K}}_\epsilon$ with high probability.

■

VI. 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 [1], [2]. Assume that all edge weights and the desired approximation factor ϵ are polynomial in n, then we can consider the binary representation of each edge's weight, out to $O(\log n)$ bits. For each bit of precision, we maintain a separate unweighted graph $G_0, G_1, ...G_{O(\log n)}$. 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. We have that: G = $\sum_i 2^i \cdot G_i$, so given a $(1 \pm \epsilon)$ sparsifier $\tilde{\mathbf{K}}_i$ for each $\mathbf{K}_i$ we have:

$$(1-\epsilon)\sum_i 2^i \cdot \mathbf{K}_i \preceq \sum_i 2^i \cdot \tilde{\mathbf{K}}_i \preceq (1+\epsilon)\sum_i 2^i \cdot \mathbf{K}_i$$

$$(1-\epsilon)\mathbf{K} \preceq \sum_i 2^i \cdot \tilde{\mathbf{K}}_i \preceq (1+\epsilon)\mathbf{K}$$

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

VII. Sparsification of Structured Matrices

Here we show that our algorithm can be extended to handle certain general matrices rather than just graph Laplacians. There were only three places in our analysis where we used that $\mathbf{B}$ was not an arbitrary matrix. First, we needed that $\mathbf{B} = \mathbf{S}\mathbf{B}_n$, where $\mathbf{B}_n$ is the vertex edge incidence matrix of the unweighted complete graph on n vertices.

In other words, we assumed that we had some dictionary matrix $\mathbf{B}_n$ whose rows encompass every possible row that could arrive in the data stream. In addition to this dictionary assumption, we needed $\mathbf{B}$ 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 $\Omega(n^2 \operatorname{polylog}(n))$ lower bound for approximately solving regression on general $\mathbb{R}^{m\times n}$ matrices in the streaming model [28].

As such, to handle the general 'structured matrix' case, we assume that we have some dictionary $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{m\times n}$ containing rows $\mathbf{a}_i \in \mathbb{R}^n$ for each $i \in [m]$. We assume that $m = O(\operatorname{poly}(n))$. In the dynamic streaming model we receive insertions and deletions of rows from $\boldsymbol{\mathcal{A}}$ resulting in a matrix $\mathbf{A} = \mathbf{S}\boldsymbol{\mathcal{A}}$ where $\mathbf{S} \in \mathbb{R}^{m\times m}$ is a diagonal matrix such that $\mathbf{S}_{ii} \in \{0, 1\}$ for all $i \in [m]$. Our goal is to recover an $O(n \operatorname{polylog}(m))$ space compression a diagonal matrix $\mathbf{W}$ with at most $O(n \log(n))$ nonzero entries such that $\boldsymbol{\mathcal{A}}^\top \mathbf{W}^2 \boldsymbol{\mathcal{A}} \approx_\epsilon \boldsymbol{\mathcal{A}}^T \mathbf{S}^2 \boldsymbol{\mathcal{A}} = \mathbf{A}^\top \mathbf{A}$. Formally, we prove the following:

Theorem 4 (Streaming Structured Matrix Sparsification)**.** *Given a row dictionary $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{m\times n}$ containing all possible rows of the matrix $\mathbf{A}$, there exists an algorithm that, for any $\epsilon > 0$, processes a stream of row insertions and deletions for $\mathbf{A}$ in a single pass and maintains a set of linear sketches of this input in $O\left(\frac{1}{\epsilon^2} n \operatorname{polylog}(m, \kappa_u)\right)$ space where κ_u is an upper bound on the condition number of $\mathbf{A}^\top \mathbf{A}$. From these sketches, it is possible to recover, with high probability, a matrix $\tilde{\mathbf{A}}^\top \tilde{\mathbf{A}}$ such that $\tilde{\mathbf{A}}$ contains only $O(\epsilon^{-2} n \log n)$ reweighted rows of $\mathbf{A}$ and $\tilde{\mathbf{A}}^T \tilde{\mathbf{A}}$ is a $(1 \pm \epsilon)$ spectral sparsifier of $\mathbf{A}^T \mathbf{A}$. The algorithm recovers $\tilde{A}$ in $\operatorname{poly}(m, \epsilon, n, \log \kappa_u)$ time.*

Note that, when $\mathbf{A}, \kappa_u = o(\operatorname{poly}(n))$, the sketch space is $O\left(\frac{1}{\epsilon^2} n \operatorname{polylog}(n)\right)$. To prove Theorem 4, we need to introduce a more complicated sampling procedure than what was used for the graph case. In Lemma 3, for the correctness proof of `RefineSparsifier` in Section V, we relied on the structure of our graph Laplacian and vertex edge incidence matrix to show that $\tilde{\tau}_{e',e} \leq \tilde{\tau}_e$. This allowed us to show that the norm of a sampled $\mathbf{x}_e^{s(e)}$ concentrates around its mean. Thus, we could recover edge e with high probability if it was in fact included in the sampling $\mathbf{B}_{s(e)}$. Unfortunately, when processing general matrices, $\tilde{\tau}_e$ is not necessarily the largest element $\mathbf{x}_e^{s(e)}$ and the concentration argument falls apart.

We overcome this problem by modifying our algorithm to compute more sketches. Rather than computing a single $\boldsymbol{\Pi}\mathbf{A}_s$, for every sampling rate $1/2^s$, we compute $O(\log n)$ sketches of different samplings of $\mathbf{A}$ at rate $1/2^s$. Each sampling is fully independent from the *all* others, including those at the same and different rates. This differs from the graph case, where $\mathbf{B}_{1/2^{s+1}}$ was always a subsampling of $\mathbf{B}_{1/2^s}$ (for ease of exposition). Our modified set up lets us show that, with high probability, the norm of $\mathbf{x_i}^{s(i)}$ 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 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 proceed as in the previous sections to prove Theorem 4. We defer the algorithm and the proofs to the full version of the paper [23] due to space constraints.

VIII. Using a Pseudorandom Number Generator

In the proof of our sketching algorithm, Theorem 3, we assume that `MaintainSketches` has access to $O(\log n)$ uniform random hash functions, $h_1, \ldots, h_{O(\log n)}$ mapping every edge to $\{0, 1\}$. These functions are used to subsample our vertex edge incidence matrix, $\mathbf{B}$, at geometrically decreasing rates. Storing the functions as described would require $O(n^2 \log n)$ space - we need $O(\log n)$ random bits for each possible edge.

To achieve $O(n \operatorname{polylog}(n))$ space, we need to compress the hash functions using Nisan's pseudorandom number generator [29]:

Theorem 5 (Corollary 1 in [29])**.** *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 $(O(S \log R))$)*

The application of Theorem 5 to our algorithms is not immediate. Our approach follows an argument in [1] (Section 3.4) that was originally introduced in [30] (Section 3.3), and relies crucially on the fact that our algorithms are based on linear sketches, i.e. their output does not depend on the order in which edges are inserted and deleted in the stream. We defer the details of this argument to the full version of the paper [23].

Acknowledgements

We would like to thank Richard Peng for pointing us to the recursive row sampling algorithm contained in [5], which became a critical component of our streaming algorithm. We would also like to thank Jonathan Kelner for useful discussions and Jelani Nelson for a helpful initial conversation on oblivious graph compression.

This work was partially supported by NSF awards 0843915, 1111109, and 0835652, CCF-1065125, CCF-AF-0937274, CCF-0939370, and CCF-1217506, NSF Graduate Research Fellowship grant 1122374, Hong Kong RGC grant 2150701, AFOSR grants FA9550-13-1-0042 and FA9550-12-1-0411, MADALGO center, Simons Foundation, and the Defense Advanced Research Projects Agency (DARPA).

REFERENCES

[1] K. J. Ahn, S. Guha, and A. McGregor, "Graph sketches: sparsification, spanners, and subgraphs," in *PODS*, 2012, pp. 5–14.

[2] A. Goel, M. Kapralov, and I. Post, "Single pass sparsification in the streaming model with edge deletions," *CoRR*, vol. abs/1203.4900, 2012.

[3] J. A. Kelner and A. Levin, "Spectral sparsification in the semi-streaming setting," in *STACS*, 2011, p. 440.

[4] K. J. Ahn, S. Guha, and A. McGregor, "Spectral sparsification in dynamic graph streams," in *APPROX-RANDOM*, 2013, pp. 1–10.

[5] G. L. Miller and R. Peng, "Iterative approaches to row sampling," *CoRR*, vol. abs/1211.2713v1, 2012.

[6] S. Muthukrishnan, "Data streams: Algorithms and applications," www.cs.rutgers.edu/~muthu/stream-1-1.ps, 2005.

[7] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, "External memory algorithms," J. M. Abello and J. S. Vitter, Eds. Boston, MA, USA: American Mathematical Society, 1999, ch. Computing on Data Streams, pp. 107–118. [Online]. Available: http://dl.acm.org/citation.cfm?id=327766.327782

[8] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "On graph problems in a semi-streaming model," *Theoretical Computer Science*, vol. 348, no. 2, pp. 207–216, 2005.

[9] K. J. Ahn, S. Guha, and A. McGregor, "Analyzing graph structure via linear measurements," in *SODA*, 2012, pp. 459–467. [Online]. Available: http://dl.acm.org/citation.cfm?id=2095116.2095156

[10] L. Epstein, A. Levin, J. Mestre, and D. Segev, "Improved approximation guarantees for weighted matching in the semi-streaming model," *SIAM Journal on Discrete Mathematics*, vol. 25, no. 3, pp. 1251–1265, 2011.

[11] M. Elkin, "Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners," *ACM Transactions on Algorithms*, vol. 7, no. 2, p. 20, 2011.

[12] A. McGregor, "Graph stream algorithms: A survey," http://people.cs.umass.edu/~mcgregor/papers/13-graphsurvey.pdf, 2013.

[13] A. Benczúr and D. Karger, "Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time," in *STOC*, 1996, pp. 47–55.

[14] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC*, 2004, pp. 81–90.

[15] K. J. Ahn and S. Guha, "Graph sparsification in the semi-streaming model," in *ICALP (2)*, 2009, pp. 328–338.

[16] M. Kapralov and D. Woodruff, "Spanners and sparsifiers in dynamic streams," *PODC*, pp. 107–118, 2014.

[17] M. Kapralov and R. Panigrahy, "Spectral sparsification via random spanners," *ITCS*, pp. 393–398, 2012.

[18] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," in *STOC*, 2008, pp. 563–568. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374456

[19] T. Sarlos, "Improved approximation algorithms for large matrices via random projections," in *FOCS*, 2006, pp. 143–152.

[20] J. Nelson and H. L. Nguyen, "Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings," in *FOCS*, 2013, pp. 117–126.

[21] M. Li, G. L. Miller, and R. Peng, "Iterative row sampling," in *FOCS*, 2013, pp. 127–136.

[22] J. A. Tropp, "User-friendly tail bounds for sums of random matrices," *Foundations of Computational Mathematics*, vol. 12, no. 4, pp. 389–434, 2012.

[23] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford, "Single pass spectral sparsification in dynamic streams," *CoRR*, vol. abs/1407.1289, 2014.

[24] N. Harvey, "Matrix concentration," http://www.cs.rpi.edu/~drinep/RandNLA/slides/Harvey_RandNLA@FOCS_2012.pdf, 2012.

[25] A. C. Gilbert and P. Indyk, "Sparse recovery using sparse matrices," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 937–947, 2010.

[26] I. Koutis, G. L. Miller, and R. Peng, "A nearly-m log n time solver for sdd linear systems," in *FOCS*, 2011, pp. 590–598.

[27] A. Blum and A. Gupta, "Randomized algorithms lecture notes," http://www.cs.cmu.edu/~avrim/Randalgs11/lectures/lect0124.pdf, 2011.

[28] K. Clarkson and D. Woodruff, "Numerical linear algebra in the streaming model," in *STOC*, 2009, pp. 205–214.

[29] N. Nisan, "Pseudorandom generators for space-bounded computation," *Combinatorica*, vol. 12, no. 4, pp. 449–461, 1992.

[30] P. Indyk, "Stable distributions, pseudorandom generators, embeddings and data stream computation," in *FOCS*, 2000, pp. 189–197.