Spanners and Sparsifiers in Dynamic Streams

Michael Kapralov[*]
MIT

David P. Woodruff
IBM Almaden

ABSTRACT

Linear sketching is a popular technique for computing in dynamic streams, where one needs to handle both insertions and deletions of elements. The underlying idea of taking randomized linear measurements of input data has been extremely successful in providing space-efficient algorithms for classical problems such as frequency moment estimation and computing heavy hitters, and was very recently shown to be a powerful technique for solving graph problems in dynamic streams [AGM'12]. Ideally, one would like to obtain algorithms that use one or a small constant number of passes over the data and a small amount of space (i.e. sketching dimension) to preserve some useful properties of the input graph presented as a sequence of edge insertions and edge deletions.

In this paper, we concentrate on the problem of constructing linear sketches of graphs that (approximately) preserve the spectral information of the graph in a few passes over the stream. We do so by giving the first sketch-based algorithm for constructing multiplicative graph spanners in only *two passes* over the stream. Our spanners use $\widetilde{O}(n^{1+1/k})$ bits of space and have stretch 2^k. While this stretch is larger than the conjectured optimal $2k-1$ for this amount of space, we show for an appropriate k that it implies the first 2-pass spectral sparsifier with $n^{1+o(1)}$ bits of space. Previous constructions of spectral sparsifiers in this model with a constant number of passes would require n^{1+c} bits of space for a constant $c > 0$. We also give an algorithm for constructing spanners that provides an *additive* approximation to the shortest path metric using *a single pass* over the data stream, also achieving an essentially best possible space/approximation tradeoff.

1. INTRODUCTION

Massive graphs are now a common way of representing real world data. Search engines and social networks require supporting various queries on large-scale graphs efficiently, and in particular, without having to store the entire graph in memory. An important type of query is a distance query between nodes in the graph. Various notions of distance can be used depending on the application, e.g., shortest path distances, minimum cuts, and effective resistances.

For each of these notions of distance it is known how to *compress* a graph to a small representation that allows for computing distance queries approximately given only the compressed representation. For shortest path distances this is achieved by the spanner construction algorithms of Thorup and Zwick[TZ01] (see also [TZ06, MN06, Bas08, BKMP10, BS03, BS06]), for cuts this is provided by the sparsifiers of Benczúr and Karger[BK96] (see also [FHHP11]) and for effective resistances this is given by the spectral sparsifiers of Spielman and Srivastava [SS08] (see also [BSS09]).

While these compression schemes have been quite successful, when processing massive graphs the data is often distributed and presented online as a long stream of insertions and deletions to its edges on multiple servers. The servers would like to determine aggregate properties of the underlying graph with low communication. Existing compression methods no longer apply, and this has motivated techniques for developing short synopses which are efficiently updatable in these settings. Linear sketching provides such a tool for compressing an object, represented as a vector x, where one chooses a random, succinctly representable matrix S and projects x to a vector $S \cdot x$. If S has fewer rows than columns, this can provide a significant compression. Moreover, given a positive or negative update to the i-th coordinate of x, denoted (i, Δ) and indicating that $x_i \leftarrow x_i + \Delta$, one can add $\Delta \cdot S_i$ to the current sketch $S \cdot x$ to efficiently update the sketch of the underlying vector. Here S_i denotes the i-th column of S. In the distributed setting, the servers can agree upon a sketching matrix S, which is often efficient to communicate if the entries of S are pseudorandom, and then locally compute Sx^i, where $x = x^1 + x^2 + \cdots x^s$ and x^i is the vector held by the i-th server, for $i = 1, 2, \ldots, s$. Communicating Sx^i rather than x^i itself can lead to significant savings.

Linear sketching has only recently emerged as a powerful technique for compressing graphs, pioneered in the work of Ahn, Guha, and McGregor [AGM12a]. In this setting, one views an underlying multigraph on n vertices as an $\binom{n}{2}$-dimensional vector x, indicating the multiplicity of each edge. A stream $\mathcal{S} = a_1, \ldots, a_t$ is given where each $a_k \in [n] \times [n] \times \{-1, 1\}$, and where $x_{i,j} = |\{k : a_k = (i, j, +)\}| - |\{k : a_k = (i, j, -)\}|$ for $i < j$. We assume that the edge multiplicity is non-negative and that the multigraph has no self-loops. This model is known as the dynamic streaming model. In a sequence of work [AGM12a, AGM12b, AGM13], Ahn, Guha, and McGregor show that it is possible to evaluate properties such as bipartiteness, connectivity, k-connectivity, dense subgraphs, maximum weighted matchings, minimum spanning trees, spanners and

[*]This research was supported by NSF award CCF-1065125, MADALGO center and Simons Foundation. We also acknowledge financial support from grant #FA9550-12-1-0411 from the U.S. Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Projects Agency (DARPA).

sparsifiers with near linear space in n (as opposed to n^2, which one can do simply by storing each of the edge multiplicities). See also the survey by Guha and McGregor [GM12].

Despite this progress, several important open questions remain. For instance, for multiplicative graph spanners, one seeks to find a sparse subgraph H of a graph G such that for all pairs of vertices u, v one has $d_G(u,v) \leq d_H(u,v) \leq t \cdot d_G(u,v)$, where $d_G(u,v)$ denotes the shortest path distance in G, and $t \geq 1$ is known as the distortion factor. One of the main applications of multiplicative graph spanners is to building spectral sparsifiers. If L_G is the Laplacian of a graph G (see Section 2 for background), a weighted graph H on the same vertex set is said to be a spectral sparsifier of G if $x^T L_H x = (1 \pm \epsilon) x^T L_G x$ simultaneously for all unit vectors x. Spectral sparsifiers approximately preserve the value of all cuts in a graph, by restricting x to binary vectors, but in fact preserve the entire quadratic form associated with the Laplacian of G, and are the focus of a very active line of research [ST04, SS08, BSS09, ST11, AGM12b, AGM13]. They have been instrumental in obtaining the first near-linear time algorithm for solving SDD linear systems [ST11], and have led to many other beautiful ideas [KMP10, KMP11].

Recently, it was shown how to find a spectral sparsifier in a single pass in the dynamic streaming model using $\tilde{O}(n^{5/3})$ bits of space. By combining work of [AGM12b] with the connection between multiplicative spanners and spectral sparsifiers in [KP12], it is possible to construct a spectral sparsifier in the dynamic streaming model with $\tilde{O}(n^{1+1/k})$ bits of space using $O(\log k)$ passes, for any $k \geq 2$. Note that if one wants a constant number of passes, this approach would require at least n^{1+c} bits of space for some positive constant $c > 0$. A natural question is if a sparsifier can be constructed in less space in a constant number of passes.

While multiplicative spanners are well-studied in the dynamic streaming model, another natural question is whether it is possible to design additive spanners in this model. For an additive spanner, one seeks to find a sparse subgraph H of a graph G such that for all pairs of vertices u, v one has $d_G(u,v) \leq d_H(u,v) \leq d_G(u,v) + t$, where t is the distortion factor. Note that additive spanners provide a much stronger guarantee on the distortion than multiplicative spanners. Compression schemes for these spanners have been extensively studied in the algorithms community, where surprisingly, one can achieve $\tilde{O}(n^{4/3})$ space and $O(1)$ distortion (see, e.g. [ACIM99, BKMP10, Che13] and references therein). A natural question is if such results are possible in the dynamic streaming model.

Our results.

In this paper we study the same dynamic streaming model defined above, as in [AGM12a, AGM12b, AGM13]. Note that the edge multiplicity in this model should not to be confused with the *weight* of an edge, in the case that we consider algorithms for weighted graphs. In the case of weighted graphs, the underlying stream can either add a weighted edge or completely remove the weighted edge that has already been added (i.e., set its weight to 0), that is, we do not allow turnstile updates which increment and decrement the weights [1].

First, for multiplicative spanners we show that for any $k \geq 1$ a 2^k-spanner can be constructed in $\tilde{O}(n^{1+1/k})$ space in the dynamic streaming model if only *two passes* over the stream are allowed:

THEOREM 1. *There exists an algorithm for constructing a 2^k-spanner of a weighted graph G using $\tilde{O}(n^{1+1/k})$ bits of space and two passes in the dynamic streaming model.*

This gives a different tradeoff than the multiplicative spanner constructions of [AGM12b], who show how to achieve $\tilde{O}(n^{1+1/k})$ bits of space using either $O(k)$ passes and $O(k)$ distortion, or $O(\log k)$ passes and poly(k) distortion. Our result shows that $O(1)$ passes and 2^k distortion are also possible. Unlike the algorithms in [AGM12b], our algorithm does not seem to be a less adaptive implementation of a non-streaming algorithm of Baswana and Sen [BS07]. Rather, it requires a different way of growing clusters, based on connecting clusters via randomly sampled vertices chosen ahead of time, which causes the diameter of our clusters to grow exponentially. The algorithm and its analysis are given in Section 3. While the distortion is rather large, we show that it is sufficient in order to use the reduction from spectral sparsification to spanner construction presented in [KP12]. By doing so we obtain the first algorithm for constructing ϵ-spectral sparsifiers in two passes over the input stream and $O(n^{1+o(1)}/\epsilon^4)$ space, for any $\epsilon > 0$:

COROLLARY 2. *For any $\epsilon > 0$ there exists an algorithm for constructing an ϵ-spectral sparsifier of a graph G presented in the dynamic streaming model using $\frac{1}{\epsilon^4} n 2^{O(\sqrt{\log n})} \log(w_{max}/w_{min})$ space and two passes over the stream (here w_{min}, w_{max} are the minimum and maximum edge weights in the input graph).*

We also consider the related problem of constructing additive spanners. For this problem, we present a single pass algorithm:

THEOREM 3. *For each $d \geq 1$ there exists an $\tilde{O}(nd)$-space algorithm for constructing an n/d-additive spanner of an unweighted undirected graph G in a single pass in the dynamic streaming model.*

This space versus approximation tradeoff is optimal even for the insertion-only model, in which each edge is inserted only once and there are no deletions:

THEOREM 4. *In the insertion-only model, any 1-pass streaming algorithm returning a spanner with additive distortion at most n/d with probability at least $6/7$ requires $\Omega(nd)$ bits of space.*

All our algorithms are randomized and succeed with high probability.

Organization.

We start with preliminaries and notation in section 2. Our two-pass spanner construction is presented in section 3. Section 4 gives the algorithm for additive spanner construction in a single pass, and section 5 shows that our approximation versus space tradeoff for this problem is near-optimal even for the insertion-only model. Finally, the spectral sparsification algorithm that we obtain from our multiplicative spanner construction is discussed in Appendix 6.

2. PRELIMINARIES

DEFINITION 5 (t-SPANNER). *A t-spanner of a weighted undirected graph G is a subgraph H of G such that the distances in H are stretch t estimates of the distances in G, i.e. for all $u, v \in V$ one has $d_G(u,v) \leq d_H(u,v) \leq t \cdot d_G(u,v)$.*

For a weighted undirected graph $G = (V, E, w)$ the Laplacian matrix of G is the matrix defined as $L_G(i,j) = -w_{(i,j)}$ and $L_G(i,i) = \sum_{j \neq i} w_{ij}$.

[1]This is consistent with the algorithms in [AGM12b], e.g., in Section 3.5 of that paper in their proof of Theorem 3.8 for sparsifiers the authors assume the weights are given beforehand (otherwise they cannot perform the partitioning into weights given after their Lemma 3.7), while in Section 5 of that paper their algorithm for spanners is only for unweighted graphs.

DEFINITION 6 (SPECTRAL ORDERING OF GRAPHS). *We define a partial ordering $\prec$ on graphs by letting $G \prec H$ if and only if $x^T L_G x \le x^T L_H x \, \forall x \in \mathbb{R}^{|V|}$.*

A weighted undirected graph G can be associated with an electrical network with link e having conductance w_e (i.e. corresponding to a resistor of resistance $1/w_e$). Then the effective resistance R_e across an edge e is the potential difference induced across it when a unit of current is injected at one end of e and extracted at the other end of e. The best known parameters for spectral sparsification are achieved by the algorithm of Spielman and Srivastava:

THEOREM 7 (SPECTRAL SPARSIFICATION, [SS08]). *Let H be obtained by sampling edges of G independently with probability $p_e = \Theta(w_e R_e \log n/\epsilon^2)$ for some $\epsilon > 1/\sqrt{n}$ and giving each sampled edge weight $1/p_e$. Then whp $(1-\epsilon)G \prec H \prec (1+\epsilon)G$.*

For a graph $G = (V, E)$ and a set $S \subseteq V$ we let $N(S) = \{v \in V : (u, v) \in E \text{ for some } u \in S\}$ denote the vertex neighborhood of S. We will use the following results from sparse recovery and sketching distinct elements:

THEOREM 8 ([CM06]). *We can construct a randomized $0/1$ matrix T of dimension $O(ck \log^3 n) \times n$ such that for any k-sparse signal A of dimension N, given the transformation TA we can reconstruct A exactly with probability at least $1 - n^{-c}$ in time $O(c^2 k \log^3 n)$. The matrix T is constructed using $O(1)$-wise independent hash functions, and individual entries can be queried efficiently.*

We note that we could also use other sketches, such as CountSketch instead of Theorem 8, improving upon the logarithmic factors in the space, though the reconstruction time will be larger.

THEOREM 9 ([KNW10]). *There is a linear sketch-based algorithm using $O(\epsilon^{-2} \log^2 n \log 1/\delta)$ bits of space that can estimate the number of distinct elements of an n-dimensional vector with integer entries bounded by a polynomial in n to within a factor of $(1 \pm \epsilon)$ with probability $1 - \delta$.*

Theorem 9 follows by repeating the algorithm of [KNW10] $O(\log 1/\delta)$ times and taking the median (note also, one can implement that algorithm as a linear sketch over the integers rather than over a finite field if desired, incurring the above space bound).

It is convenient to introduce notation for the linear sketches guaranteed by Theorem 8. In what follows, we assume that we have a (randomized) linear function $\text{SKETCH}_B : \mathbb{R}^n \to \mathbb{R}^r$ with $r = O(B \log^3 n)$ and a function $\text{DECODE} : \mathbb{R}^r \to \mathbb{R}^n$ such that for every $x \in \mathbb{R}^n$ with $||x||_\infty \le \text{poly}(n)$ and $||x||_0 \le B$ one has $\text{DECODE}(\text{SKETCH}_B(x)) = x$ with probability at least $1 - 1/\text{poly}(n)$. In general, most primitives that we use have a $1/\text{poly}(n)$ failure probability that can be made arbitrarily small at the expense of increased constants in space bounds, so we will always condition on the failure event not happening. In what follows we assume that we always know if a $\text{SKETCH}_B(x)$ can be decoded. This is easily achieved by maintaining a distinct elements sketch for each instantiation of $\text{SKETCH}_B(x)$, and declaring the sketch to be not decodable when the number of distinct elements is estimated to be above $2B$, for example. This involves an $O(\log^3 n)$ overhead for each instantiation of $\text{SKETCH}_B(x)$ by Theorem 9, and hence does not influence the asymptotic space requirement of $\text{SKETCH}_B(x)$.

AGM sketches.

In [AGM12a] Ahn et al gave an algorithm for constructing connectivity certificates from linear sketches of graphs. We will need

THEOREM 10 (AGM SKETCH, [AGM12A]). *There is a single-pass, linear sketch-based algorithm supporting edge additions and deletions that uses $O(n \log^3 n)$ space and returns a spanning forest of the graph with high probability.*

It will be very useful for our application that the sketches are linear. In particular, we will maintain AGM sketches as in Theorem 10 for a graph G and use them for finding a spanning forest of a graph G' obtained by subtracting a set of edges from G. Besides linearity, AGM sketches have the following property, which we use in our additive spanner construction. The AGM sketch is in fact a collection of vertex neighborhood sketches, and if a graph H is obtained from G by collapsing some sets of nodes into supernodes, an AGM sketch for H can be obtained from an AGM sketch for G (by adding sketches of vertex neighborhoods appropriately).

3. TWO PASSES: BASIC ALGORITHM

In this section we first describe a simple algorithm for constructing spanners in unweighted graphs, and then implement it in the dynamic streaming model using two passes over the stream. Our algorithm extends trivially to weighted graphs by partitioning edges into a geometric sequence of weight classes (this can be done since we are assuming that edges are either added or completely deleted, i.e. the weight of the edge is known during each update).

3.1 Basic algorithm.

Let $\mathcal{C}_i, i = 0, \dots, k-1$ be a subset of V where nodes are present independently with probability $n^{-i/k}$. In what follows $C > 0$ is a sufficiently large constant. Our algorithm runs in two phases. In the *first phase* for each $i = 0, \dots, k-2$ we cluster vertices in $\mathcal{C}_i$ around vertices in $\mathcal{C}_{i+1}$. The clusters will be defined by a forest $F \subseteq E$ on V, with the edges of each tree in F always connecting a node in $u \in \mathcal{C}_i$ to at most one node in $v \in \mathcal{C}_{i+1}$[2], $i = 0, \dots, k-2$. Then v is a parent of u in F. For a node $u \in V$ we denote the subtree of u in F by T_u. For a node $u \in V$ we let $p(u)$ denote the parent of u in F ($p(u) = \perp$ if u is the root of its subtree). For each $u \in \mathcal{C}_i$ we either make u a child of a node $v \in \mathcal{C}_{i+1}$ if $N(T_u) \cap \mathcal{C}_{i+1} \neq \emptyset$ and make u the root of its own component otherwise. As we will show below, if u ends up a root of its own component, then the vertex expansion of its subtree T_u cannot be large (in particular, it cannot be that $|N(T_u)| \gg n^{(i+1)/k}$). We note that the forest F is only a logical construction. Its edges may not necessarily be a subset of the graph G. However, each edge e in the forest F will be associated with an edge $\eta(e)$ of G (which we refer to as the *witness edge* for e; see below for formal definition). For each sutree T_u of F, the set of corresponding witness edges will provide connectivity to nodes in T_u using edges of the graph G.

The *second phase* selects a set of edges to be added to the spanner: we include a set of edges F' determined by F, and a sufficient number of edges to preserve the vertex neighborhood $N(T_u)$ for each $u \in V$ (namely one edge from each $v \in N(T_u) \setminus T_u$ to some $w \in T_u$). We now describe both phases formally, specify parameters and prove correctness and space bounds.

First phase – constructing the clusters.

We start with $F = (V, \emptyset)$, i.e. $T_u = \{u\}$ for all $u \in V$. Then for each $i = 0, \dots, k-1$ and for each $u \in \mathcal{C}_i$ **(1)** if $i = k-1$ or $\mathcal{C}_{i+1} \cap N(T_u) = \emptyset$, mark u as a *terminal* node; **(2)** otherwise make an arbitrary $w \in \mathcal{C}_{i+1} \cap N(T_u)$ the parent of u, i.e. add

[2]In fact, a node u could belong to multiple sets $\mathcal{C}_i$, so F is not necessarily a forest on V, but a forest on $V \times \{0, \dots, k-1\}$, where each node $u \in V$ is present via its k copies. We do not make this explicit in the interest of keeping notation simple.

the edge (u, w) to F. Let $v \in T_u$ denote an arbitrary vertex such that $(v, w) \in E$. Define $\eta((u, w)) = (v, w)$ (we say that the edge $\eta((u, w))$ is the edge that *witnesses* the connection of T_u to w).

Second phase – adding edges to the spanner.

In the second phase we construct the set E' of edges of the spanner. We start by setting $E' := \emptyset$. Then for each $u \in V$ **(1)** if u is non-terminal, add the edge $\eta((u, p(u)))$ to E'; **(2)** if u is terminal, for each $v \in N(T_u) \setminus T_u$ add exactly one edge (v, w) to E', where w is an arbitrary node in T_u. First, we have

CLAIM 11. *For every $i = 0, \ldots, k-1$ and every non-terminal node in $\mathcal{C}_i$ one has $|N(T_u)| \leq (C \log n)n^{(i+1)/k}$ with high probability as long as $C > 0$ is a sufficiently large constant.*

PROOF. Suppose that for a node $u \in \mathcal{C}_i$ one has $|N(T_u)| \leq (C \log n)n^{(i+1)/k}$. Then $\mathbf{E}[|N(T_u) \cap \mathcal{C}_{i+1}|] \geq (C \log n)n^{(i+1)/k} \cdot n^{-(i+1)/k} \geq C \log n$. Furthermore, the construction of T_u only depends on the random choices made for $\mathcal{C}_j, j = 0, \ldots, i$, and $\mathcal{C}_{i+1}$ is independent of $\mathcal{C}_j, j = 0, \ldots, i$, so by Chernoff bounds $N(T_u) \cap \mathcal{C}_{i+1} \neq \emptyset$ with probability $1 - n^{-10}$, say, whenever $C > 0$ is a sufficiently large constant, so such a node u would not be terminal. Thus, for every terminal node in $\mathcal{C}_i$ one has $|N(T_u)| \leq (C \log n)n^{(i+1)/k}$ with high probability, as required. $\square$

The number of edges added to the spanner is small:

LEMMA 12. *One has $|E'| = O(kn^{1+1/k} \log n)$.*

PROOF. The set E' is a union of the witness edges $\eta(F)$ and the edges added in step 2 of the second phase, where for each terminal node we add exactly one edge to each $v \in N(T_u) \setminus T_u$.

It remains to bound the number of edges added to the spanner in the second step of the algorithm. By Claim 11 for every non-terminal node in $\mathcal{C}_i$ one has $|N(T_u)| \leq (C \log n)n^{(i+1)/k}$, and hence the total number of edges is bounded by $|\eta(F)| + \sum_{i=0}^{k-1} |\mathcal{C}_i| \cdot O(n^{(i+1)/k} \log n) \leq k(n-1) + \sum_{i=0}^{k-1} O(n^{1-i/k} \cdot n^{(i+1)/k} \log n) = O(kn^{1+1/k} \log n)$. $\square$

Approximation quality is guaranteed by

LEMMA 13. *$H = (V, E')$ is a 2^k-spanner of $G = (V, E)$.*

PROOF. In what follows we abuse notation somewhat by writing, for a node $u \in \bigcup_{i \geq 0} \mathcal{C}_i$, $\eta(T_u)$ to denote the subgraph formed by edges in $\bigcup_{e \in T_u} \eta(e)$. We prove by induction on $j \geq 1$ that for every $u \in \mathcal{C}_j$ the diameter of $\eta(T_u)$ is bounded by $2^{j+1} - 2$.

Base:$j = 1$ The tree $\eta(T_u)$ is a star when $u \in \mathcal{C}_1$, so the diameter is $2 = 2^{1+1} - 2$.

Inductive step:$j \to j+1$ Consider $u \in \mathcal{C}_{j+1}$. We need to show that any two vertices $a, b \in T_u$ are connected by a path of length at most $2^{j+2} - 2$. Let w be the least common ancestor of a and b in F. If $w \neq u$, then we are done since the diameter of $\eta(T_w)$ is bounded by $2^{j+1} - 2 \leq 2^{j+2} - 2$ by the inductive hypothesis (since w is necessarily at level no higher than i).

Now suppose that u is the least common ancestor of a and b. Let (u, q_a) (resp. (u, q_b)) denote the edges connecting u to the subtrees that a and b belong to. By the inductive hypothesis the distance from q_a to a and the distance from q_b to b is bounded by $2^{j+1} - 2$. Thus, the distance from a to b is bounded by $2(2^{j+1} - 2) + 2 \leq 2^{j+2} - 2$ as required.

We now show that each edge of G is stretched by at most a factor of 2^k. Consider an edge $(u, v) \in E$. Let u' denote the terminal parent of u and v' denote the terminal parent of v. If $u' = v'$, then u and v are connected by a path of length at most $2^k - 2$ since the highest level that $u' = v'$ could be at is $k-1$, so we are done. Now suppose that $u' \neq v'$. Let (w, v) denote the edge from $w \in T_u$ to v that was added to E' by our algorithm (such an edge exists since the edge (u, v) is in E). Then the path $v \to w \to u$ has length at most $(2^k - 2) + 1 \leq 2^k$, yielding the result. $\square$

REMARK 14. *We note that our algorithm extends to weighted graphs by the simple reduction: round weights to the nearest power of $1 + \epsilon$ for some $\epsilon > 0$, and run the unweighted spanner construction on each weight class. This requires at most a factor of $O(\frac{1}{\epsilon} \log(w_{max}/w_{min}))$ more space.*

In the next section we give a two-pass dynamic streaming implementation of our algorithm.

3.2 Streaming implementation using two passes

Our streaming implementation uses two passes, one for each phase of the algorithm above.

First pass – constructing the clusters.

Recall that in the first phase we construct the forest F bottom up. For each $i = 0, \ldots, k-2$ and $u \in \mathcal{C}_i$ one makes an arbitrary node $v \in N(T_u) \cap \mathcal{C}_{i+1}$ the parent of u if this intersection is non-empty. Otherwise u is declared to be a terminal node, i.e. the root of its own component in F. Since later we will also need to recover an edge $e = (v, w)$ for a node $w \in T_u$, a natural approach to implementing this in small space as follows. We sample random subsets $E_j, j = 0, \ldots, \log_2 n^2$ by including each pair $(a, b) \in \binom{V}{2}$ into E_j independently with probability 2^{-j} and sketch the vector $E_j \cap E \cap (\mathcal{C}_{i+1} \times T_u)$. We assume that the input graph does not have multiple edges to simplify notation, even though the same analysis holds if edges can have multiplicities (one needs to replace sets by multisets in this case, but this does not affect the performance of our sketches since they can handle vectors with polynomially large entries).

We accomplish the first task by keeping linear sketches for $E_j \cap E \cap (\mathcal{C}_r \times T_u)$ for $r = 1, \ldots, k-1, j = 0, \ldots, \log_2 n^2$, i.e. by maintaining $\text{SKETCH}_B(E_j \cap E \cap (\mathcal{C}_r \times T_u))$ for $B = O(\log n)$. That way we can keep decoding sketches starting from the largest $j = \log_2 n^2$ until one of the sketches gives us a nonempty set of edges. We will abuse notation somewhat by using $\text{SKETCH}_B(A)$ to denote the sketch of the indicator vector of $A \subset \binom{V}{2}$, i.e. the vector in $\mathbb{R}^{\binom{V}{2}}$ with ones in positions that belong to A and zeros everywhere else.

We now give formal definitions. Let $E_j, j = 0, \ldots, \log_2 n^2$ denote random subsets of $\binom{V}{2}$ where each pair of nodes is present with probability 2^{-j}. Now each node $u \in \mathcal{C}_i$ computes

$$S_j^r(u) \leftarrow \text{SKETCH}_{O(\log n)}^{r,j}((\{u\} \times \mathcal{C}_r) \cap E \cap E_j)$$

for all $j = 0, \ldots, \log_2 n^2$ and $r \in [0 : k-1]$. The superscript r, j corresponds to the fact that the random bits used by SKETCH are a function of r, j, and independent for different (r, j). To construct the forest, for each $i = 0, \ldots, k-2$ each $u \in \mathcal{C}_i$ computes $Q_j^{i+1}(u) := \sum_{v \in T_u} S_j^{i+1}(v)$ for each $j = 0, \ldots, \log_2 n^2$. By linearity, this is a sketch for $(T_u \times \mathcal{C}_{i+1}) \cap E \cap E_j$, so we can detemine if there exists an edge from T_u to $\mathcal{C}_{i+1}$ by decoding this sketch for at least one value of j between 0 and $\log_2 n^2$. We note that it is sufficient to use $O(\log n)$-wise independent random variables to generate the sets E_j. We summarize this in Algorithm 1.

We have

LEMMA 15. *Algorithm 1 constructs the forest F together with witness edges whp. The space requirement is $\tilde{O}(kn)$.*

PROOF. The space complexity is immediate. We now argue correctness. Recall that to construct the forest, for each $i = 0, \ldots, k-2$ each $u \in \mathcal{C}_i$ computes $Q_j^{i+1}(u) := \sum_{v \in T_u} S_j^{i+1}(v)$, for j from $\log_2 n^2$ down to 1 until one of the sketches gives a nonempty set of edges or j reaches 0. By linearity, this is a sketch for $(T_u \times \mathcal{C}_{i+1}) \cap E \cap E_j$, so we can detemine if there exists an edge from T_u to $\mathcal{C}_{i+1}$ by decoding this sketch for at least one value of j between $\log_2 n^2$ and 0 unless decoding of a sketch fails (which happens with probability at most $1/\text{poly}(n)$). The edge recovered from the sketch serves as the witness for $(u, p(u))$. Thus, Algorithm 1 correctly emulates the first phase of our basic algorithm whp. □

Note that our algorithm runs the DECODE procedure on various sketches. The decoding procedure is guaranteed to succeed with $1-1/\text{poly}(n)$ probability as long as the sparsity of the sketched signal is smaller than the budget and its entries are properly bounded. In what follows we condition on the event that all such invocations of the decoding procedure succeed. We will need the following claim in section 6.

CLAIM 16. *Consider the execution of Algorithm 1 on a graph $G = (V, E)$. Let R denote the random seed used by the algorithm, and let $\Lambda_1(R) \subset \binom{V}{2}$ denote the set of locations of the adjacency matrix of G whose content the execution path of the algorithm depends on. Then Algorithm 1 can be augmented to output the set of all edges of G that belong to $\Lambda_1(R)$ with high probability.*

PROOF. The algorithm accesses sketches starting from small samples of the adjacency matrix and stops as soon as one of the sketches is nonzero. Thus, it is able to decode all sketches that its execution depends on, and output the corresponding edges. □

Second pass – recovering spanner edges.

We recover the edges of the spanner in the second pass. Recall that in order to do that, we need to recover: **(1)** for each non-terminal node $u \in V$ an edge $(w, p(u)) \in E$, for an arbitrary node $w \in T_u$ (such witness edges have been constructed in our implementation of the first phase already); **(2)** for each terminal node $u \in V$ and each node $v \in V \setminus T_u$ at least one edge from v to T_u.

We only need to show how to achieve (2). For each terminal node $u \in \mathcal{C}_i$ one has $|N(T_u)| \leq C n^{(i+1)/k} \log n$, so it is sufficient to keep a *linear* hash table with $\tilde{O}(n^{(i+1)/k})$ cells such that each cell can hold for each $v \in N(T_u)$ a $O(\text{poly}(\log n))$-size sketch of $N(v) \cap T_u$ (we define the properties of the desired hash table and give an implementation below). Note that sketches are only maintained at terminal nodes, and whenever an update to an edge incident on $w \in T_u$ for a *terminal* node u is received, this update is added directly to the sketch maintained by u. We now give formal definitions (the algorithm as given as Algorithm 2).

First, for each $j = 0, \ldots, \log_2 n$ let Y_j contain each node in V independently with probability 2^{-j}. For each $i \in [0 : k-1]$, each j, terminal node $u \in \mathcal{C}_i$ and $v \in V \setminus T_u$ we store $\text{SKETCH}_{O(\log n)}(N(v) \cap T_u \cap Y_j)$ in a linear hash table H_j^u of size $\tilde{O}(n^{(i+1)/k})$ using v as the key. We will be able to retrieve the contents of H_j^u since for a terminal node $u \in \mathcal{C}_i$ we have $|N(T_u)| = O(n^{(i+1)/k} \log n)$ with high probability. Then edges of the spanner are reconstructed as in the offline algorithm. We summarize this in Algorithm 2. We note that the use of the sets Y_j could be eliminated by using L0-SAMPLER in a similar way as [AGM12a] does, but we find it notationally simpler to present the algorithm in this way.

Algorithm 1 First pass (constructing clusters)

```
1: procedure CONSTRUCTCLUSTERS(k)
2:     $F \leftarrow (V, \emptyset)$
3:     Let $E_j$ contain each $(a,b) \in \binom{V}{2}$ independently with probability $2^{-j}$, for $j = 0, \ldots, \log_2 n^2$
4:     Let $\mathcal{C}_r$ contain each $v \in V$ independently with probability $n^{-r/k}$, for $r = 0, \ldots, k-1$
5:     for $u \in V$ do                                    ▷ During the first pass
6:         Maintain $S_j^r(u) \leftarrow \text{SKETCH}^{r,j}_{O(\log n)}((\{u\} \times \mathcal{C}_r) \cap E \cap E_j), j = 0, \ldots, \log_2 n^2, r \in [0 : k-1]$
7:     end for
8:     for $i = 0$ to $k-2$ do                              ▷ After the first pass
9:         for $u \in \mathcal{C}_i$ do
10:            $Q_j^{i+1}(u) \leftarrow \sum_{v \in T_u} S_j^{i+1}(v)$ for $j = 0, \ldots, \log_2 n^2$   ▷ Sketch for $(T_u \times \mathcal{C}_{i+1}) \cap E \cap E_j$
11:            for $j = \log_2 n^2$ down to 0 do
12:                $x_u \leftarrow \text{DECODE}(Q_j^{i+1}(u))$
13:                if DECODE succeeded and $x_u \neq \mathbf{0}$ then
14:                    $(a,b) \leftarrow$ an arbitrary element in the support of $x_u$, $a \in T_u, b \in \mathcal{C}_{i+1}$
15:                    Make $b$ parent of $u$ in $F$,
16:                    and let $\eta((u,b)) := (a,b)$
17:                end if
18:            end for
19:        end for
20:    end for
21:    return $F$
22: end procedure
```

We now also provide an outline of an implementation of such a linear hash table H_j^u in space $\tilde{O}(n^{(i+1)/k})$. Recall that the hash table H_j^u needs to support recovery of $\tilde{O}(n^{(i+1)/k})$ values indexed by elements of V, where each value is a string of poly$(\log n)$ bits (this is the amount of space that $\text{SKETCH}_{O(\log n)}$ needs). The hash table can be implemented by treating the sketches associated with nodes $v \in V$ as poly$(\log n)$-length bit numbers and sketching this vector $x \in \mathbb{R}^V$ using $\text{SKETCH}_{\tilde{O}(n^{(i+1)/k})}(x)$. It remains to note that we can recover the contents of this hash table whenever the number of inserted elements is $\tilde{O}(n^{(i+1)/k})$, and this is the case by Claim 11 because we only need to decode the table for terminal nodes.

LEMMA 17. *Algorithm 2 correctly executes the second phase of spanner construction whp using $\tilde{O}(kn^{1+1/k})$ space.*

PROOF. Recall that the algorithm needs to recover for each terminal node u and each $v \in V \setminus T_u$ that is connected to T_u at least one edge $(w, v), w \in T_u$. Algorithm 2 for each $j = 0, \ldots, \log_2 n$ stores sketches of neighborhoods of each $v \in V \setminus T_u$ sampled at rate 2^{-j} in the hash table H_j^u. Since u is a terminal node, at most $O(n^{(i+1)/k} \log n)$ nodes $v \in V \setminus T_u$ are connected to T_u at the end of the stream. Thus, each table H_j^u contains $O(n^{(i+1)/k} \log n)$ keys at the end of the stream and hence can be decoded. Now consider $v \in \text{keys}(H_j^u)$. Since $N(v) \cap T_u \cap Y_j$ is sketched to allow reconstruction of $O(\log n)$ sparse vectors, there exists at least one j such that reconstruction is possible whp, and it provides the necessary edge (w, v). The space complexity follows since a hash table of size $\tilde{O}(n^{(i+1)/k})$ is stored for each node in $\mathcal{C}_i$, for a total space of $\sum_{i=0}^{k-1} \tilde{O}(n^{(i+1)/k}) \cdot n^{1-i/k} = \tilde{O}(kn^{1+1/k})$. □

We will need the following claim in section 6.

CLAIM 18. *Consider the execution of Algorithm 2 on a graph $G = (V, E)$. Let R denote the random seed used by the algorithm, and let $\Lambda_2(R) \subset \binom{V}{2}$ denote the set of locations of the adjacency matrix of G whose content the execution path of the algorithm depends on. Then Algorithm 1 can be augmented to output the set of all edges of G that belong to $\Lambda_2(R)$ with high probability.*

PROOF. Algorithm 2 stores for each terminal node $u \in V$ hash tables $H_j^u, j = 0, \ldots, \log_2 n$. For a terminal node u each hash table is guaranteed to be decodable whp. For each key $v \in V$ of hash table H_j^u the entry stored is a sketch of $N(v) \cap T_u \cap Y_j$, where Y_j is a random sample of V at rate 2^{-j}. Our algorithm for each $v \in \text{keys}(H_j^u)$ starts with the largest j and decreases j until it gets a nonempty sketch (say, when $j = j^*$). This sketch can be decoded whp. Crucially, for each v the algorithm ignores the sketches $N(v) \cap T_u \cap Y_j$ for $j < j^*$, and hence its execution path only depends on the sketches that it can completely decode (we condition on the failures not happening for sketches that are guaranteed to be decoded with $1 - 1/\text{poly}(n)$ probability, and thus treat invocations of the decoding procedure as a black box). □

Proof of Theorem 1: Follows by putting together Lemma 13, Lemma 15 and Lemma 17. □

4. $O(N/D)$-ADDITIVE SPANNERS IN $\tilde{O}(ND)$ SPACE IN A SINGLE PASS

In this section we give an algorithm for constructing additive spanners in the dynamic stream model. The algorithm is similar in spirit to the multiplicative spanner construction in section 3, with the important distinction that it samples just one set of centers instead of a hierarchy, like in section 3, and, most importantly, uses the approach of [AGM12a] to connect the sampled centers by a spanning tree.

In particular, we start by choosing a set of $O(n/d)$ centers $\mathcal{C} \subseteq V$ uniformly at random, so that each node of degree above $O(d \log n)$ will have a neighbor in $\mathcal{C}$ with high probability, and let such nodes $u \in V \setminus \mathcal{C}$ select a neighbor in $\mathcal{C}$ as their parent. This way we construct a forest F such that all subtrees of F are stars with nodes in $\mathcal{C}$ at their center. Constructing such a forest is easy – it is sufficient to sketch for every $u \in V \setminus \mathcal{C}$ the set of neighbors of u in (a subsampled version of) $\mathcal{C}$, taking polylogarithmic space per node.

Further, for each node $u \in V$ we maintain a sketch of the set of edges incident on u so that we can reconstruct all edges incident on u if there are at most d of them. Finally, we also store a polylogarithmic number of independent AGM sketches. The edges of the spanner are constructed as follows. First, we let E_{low} denote the set of edges incident on low degree nodes of G (i.e. nodes whose degree was $O(d \log n)$). Then we associate each node with degree above $Cd \log n$ with a neighbor in $\mathcal{C}$ (such a neighbor exists whp by our choice of parameters). This yields $O(n/d)$ clusters $T_u, u \in \mathcal{C}$. We now let G' be the graph obtained from G by subtracting all edges in E_{low}. We then collapse all clusters $T_u, u \in \mathcal{C}$, into supernodes in the new graph G'; let F' denote a spanning forest in the contracted G'. Note that the spanning forest in G' can be computed using Theorem 10 since, starting with AGM sketches for G, we can first subtract all edges in E_{low}, and then invoke Theorem 10 on G'. Finally, we output $E_{low} \cup F \cup F'$. We formalize this in Algorithm 3. Note that as before, we introduce auxiliary sets $Z^r \subseteq V, r = 0, \ldots, \log_2 n$, and then maintain a sketch of $N(u) \cap \mathcal{C} \cap Z^r$ for each $r = 0, \ldots, \log_2 n$ in order to be able to recover a neighbor in $\mathcal{C}$ for each u with degree $\Omega(d)$. We formalize this in Algorithm 3. The guarantees provided by the algorithm are summarized in

THEOREM 19. *Algorithm 3 outputs an $O(n/d)$-additive spanner using space $\tilde{O}(nd)$.*

PROOF. The space bound follows immediately from the definition of Algorithm 3. Let $P = (u = p_0 \to p_1 \to \ldots \to p_t = v)$ be a shortest path between u and v in G. Let H denote the spanner output by Algorithm 3. We show that there exists a path of length at most $t + O(n/d)$ in H.

We first show that P visits every cluster $T_w, w \in \mathcal{C}$ at most once, i.e. there are no two indices $0 \le t_1, t_2 \le t, t_1 < t_2 - 1$ such that $p_{t_1}, p_{t_2} \in T_w$ and $p_{t_1+1} \notin T_w$. Indeed, in that case we can construct the path P' by removing the edges (p_i, p_{i+1}) for $i \in [t_1 : t_2 - 1]$(of which there are at least two) and replace it with two edges $(p_{t_1}, w), (w, p_{t_2})$, preserving (and possibly reducing) path length.

Algorithm 2 Second pass (constructing spanner edges)

```
 1: procedure CONSTRUCTSPANNER($\{\mathcal{C}_r\}_{r=0}^{k-1}, F, k$)
 2:     $E' \leftarrow \emptyset$
 3:     $Y_j \leftarrow$ sample of $V$ at rate $2^{-j}, j = 0, \ldots, \log_2 n$
 4:     for $u \in \mathcal{C}_i$, $u$ a terminal node, $i \in [0 : k-1]$ do
 5:         for $j = 0$ to $\log_2 n$ do
 6:             $H_j^u \leftarrow$ hash table with $\tilde{O}(n^{(i+1)/k})$ cells of size $O(\text{poly}(\log n))$    ▷ Create empty hash table
 7:         end for
 8:     end for
 9:                                                  ▷ During the second pass
10:     for each update $\delta \cdot (a, b)$ do    ▷ $\delta \in \{-1, 1\}$ is the sign of the update, $a, b \in V$
11:         $u \leftarrow$ terminal parent of $a$, $v \leftarrow$ terminal parent of $b$
12:         for $j = 0$ to $\log_2 n$ do
13:             If $a \in T_u \cap Y_j, b \in V \setminus T_u$,
14:             add SKETCH$_{O(\log n)}(\delta \cdot a)$ to $b$-th entry of $H_j^u$
15:             If $b \in T_v \cap Y_j, a \in V \setminus T_v$,
16:             add SKETCH$_{O(\log n)}(\delta \cdot b)$ to $a$-th entry of $H_j^v$
17:         end for
18:     end for
19:                                                  ▷ After the second pass
20:     for $u \in \mathcal{C}_i$, $u$ a non-terminal node, $i \in [0 : k-1]$ do
21:         Add $\eta((u, p(u)))$ to $E'$    ▷ The edge $\eta((u, p(u)))$ was constructed in first pass
22:     end for
23:     for $u \in \mathcal{C}_i$, $u$ a terminal node, $i \in [0 : k-1]$ do
24:         for $v \in V \setminus T_u$ do
25:             for $j = \log_2 n$ down to 0 do
26:                 If $v \notin \text{keys}(H_j^u)$ continue
27:                 $S \leftarrow H_j^u(v)$         ▷ retrieve sketch with key $v$
28:                 $w \leftarrow$ DECODE($S$)
29:                 If $w \neq \mathbf{0}$ then $E' \leftarrow E' \cup \{(w, v)\}$, break
30:                         ▷ $w$ is an arbitrary decoded neighbor of $v$
31:             end for
32:         end for
33:     end for
34:     return $E'$
35: end procedure
```

We now assume that P visits every cluster at most once and show how to construct a path P' between u and v that has length at most $t + O(n/d)$. We consider edges $e = (a, b) \in P$ of types:
Type 1. e is incident on a low degree node, or belongs to F. Then the edge is available in E^*;
Type 2. $a, b \in T_u \setminus \{u\}$ for some $u \in \mathcal{C}$. Then we can replace the edge by a pair of edges $(a, u), (u, b)$, since they are available in

$F \subseteq E^*$. Since by the argument above P contains $O(n/d)$ such edges, (a, b), in total, this only increases the path length by an additive $O(n/d)$ term;

Type 3. $a \in T_u, b \in T_v$ for some $u, v \in \mathcal{C}, u \neq v$. We call such pairs (u, v) *cross-edges* arising from P (note that (u, v) may not belong to P, only the edge (a, b) is guaranteed to). We now consider the graph $G' = (V, E')$ defined above, i.e. the graph obtained by removing edges in E_{low}. Let G'' be obtained by collapsing all clusters $T_x, x \in \mathcal{C}$ into supernodes. Consider paths P_{uv} from u to v in the spanning forest F'' of G'' for all cross-edges arising from P. We can assume that these paths do not overlap, since otherwise they can be shortcut to provide the same connectivity. It remains to note that the union of these paths contains $O(n/d)$ edges, and can be extended to paths in E^* by using edges of F.

We have shown how to construct a path P' in E^* that has length $t + O(n/d)$, which completes the proof. □

5. LOWER BOUNDS FOR ADDITIVE SPANNERS

In this section we prove Theorem 4, which we restate here for convenience of the reader.

Theorem 4 *In the insertion-only model, any* 1*-pass streaming algorithm returning a spanner with additive distortion at most* n/d *with probability at least* $6/7$ *requires* $\Omega(nd)$ *bits of space.*

PROOF. We use a 2-player *distributional* communication game to prove this. We reduce from the distributional version of the indexing problem IND, in which Alice is given a bit string X of length r, while Bob is given an index $I \in \{1, 2, \ldots, r\}$, and Bob's goal is to output X_I. We endow X with the uniform distribution on $\{0, 1\}^r$, and I with the uniform distribution on $\{1, 2, \ldots, r\}$. Further, X and I are independent. It is known that in any, possibly randomized, protocol in which Alice sends a single message $M(X)$ to Bob, and Bob succeeds in outputting X_I with probability at least $2/3$, where the probability is taken both over Alice's coin tosses and the random choice of X and I, that in expectation Alice's message $M(X)$ must be $\Omega(r)$ bits long [KNR99].

Set $r = Cnd$, for an appropriate constant $C > 0$ to be determined. Let us interpret Alice's input X as $s = 18n/d$ disjoint and independent random graphs $G_1, \ldots, G_s$ each drawn from $G(d, 1/2)$, that is, each G_ℓ has exactly d vertices and each edge is present independently in each G_ℓ with probability $1/2$. Note that each G_ℓ requires $\binom{d}{2} = \Theta(d^2)$ bits to specify, each bit indicating the presence or absence of each potential edge. For $C > 0$ a suitable constant, it is indeed possible to interpret Alice's string in this way.

Suppose we have a 1-pass streaming algorithm returing a spanner with additive distortion n/d. Alice runs this algorithm on the disjoint union of $G_1, \ldots, G_s$, and sends the state of the algorithm to Bob.

Suppose Bob is interested in the bit X_I, which can be interpreted as a specific pair of verices $\{U, V\}$ in a specific graph G_J, for some $J \in [1 : s]$. Note that with our choice of input distribution for IND, J is uniformly random, and U, V are uniformly random distinct vertices in G_J.

For each $\ell \in [1 : s]$, Bob chooses a uniformly random pair $\{U_\ell, V_\ell\}$ of distinct vertices in G_ℓ, with the exception that in G_J, Bob sets $\{U_J, V_J\} = \{U, V\}$. Bob then inserts the edges $\{V_1, U_2\}$, $\{V_2, U_3\}, \ldots, \{V_{s-1}, U_s\}$ into the input stream, and continues the computation of the streaming algorithm on this input stream, starting with the state of the streaming algorithm that Alice sent him. Suppose that with probability at least $6/7$, the output of the streaming algorithm is an additive spanner H with additive distortion at most n/d between all pairs of vertices.

The shortest path from U_1 to V_s follows the edges that Bob inserted into the stream, together with as many of the pairs $\{U_\ell, V_\ell\}$ as possible which correspond to actual edges in G_ℓ. For those G_ℓ for which $\{U_\ell, V_\ell\}$ is not an edge, then the path length is at least 2 inside of G_ℓ.

By Chernoff bounds (over the choice of edges in the different G_ℓ), with probability at least $1 - \exp(-\Theta(n/d))$, at least $s/3$ of the different $\{U_\ell, V_\ell\}$ pairs occur in the stream, as we range over $\ell \in \{1, 2, \ldots, s\}$. Hence, to achieve additive distortion at most n/d, since $s/3 = 6n/d$, at least a $5/6$ fraction of the different $\{U_\ell, V_\ell\}$ which occur in the stream must also occur in H. Since J is uniformly random, $\{U_J, V_J\} = \{U, V\}$ is uniformly random in G_J, and all other $\{U_\ell, V_\ell\}$ are uniformly random edges in G_ℓ for $\ell \neq J$, it follows that the algorithm has no information about J, and therefore if $\{U, V\}$ is an edge in G_J then with probability at least $1 - 1/6 - 1/7 - \exp(-\Theta(n/d)) > 2/3$ it occurs in H, while if $\{U, V\}$ is not an edge in G_J, then it occurs in H with probability at most $1/7$, i.e., if the streaming algorithm fails.

Algorithm 3 Construction of an $O(n/d)$-additive spanner

```
procedure AdditiveSpanner($d$)
    $F \leftarrow (V, \emptyset)$
    $\mathcal{C} \leftarrow$ sample of $V$ at rate $O(\frac{1}{d})$
    $Z^r \leftarrow$ sample of $V$ at rate $2^{-r}$, $r = 0, \ldots, \log_2 n$
        ▷ Auxiliary variables for recovering a node in $N(u) \cap \mathcal{C}$
    For each $u \in V$ maintain:
    $S(u) \leftarrow \text{Sketch}_{\tilde{O}(d)}(N(u))$ ▷ Sketching neighborhood of all nodes $u \in V$
    $A^r(u) \leftarrow \text{Sketch}_{O(\log n)}(N(u) \cap \mathcal{C} \cap Z^r)$ for $r = 0, \ldots, \log_2 n$
    Sketch $\hat{d}_u$ to degree of $u$ (using Theorem 9)
    AGM sketches as per Theorem 10
    $E_{low} \leftarrow \emptyset$
    for $u \in V$ do
        if $\hat{d}_u \leq O(d \log n)$ then
            $W \leftarrow \text{Decode}(S(u))$
            $E_{low} \leftarrow E_{low} \cup \{(u, v)\}_{v \in W}$
        else
            for $r = 0$ to $\log_2 n$ do▷ Associate $u$ with a node in $\mathcal{C}$
                $w \leftarrow \text{Decode}(A^r(u))$
                If $w \neq \perp$ then make (an arbitrary element of) $w$ parent of $u$ in $F$, break
            end for
        end if
    end for
    $E' \leftarrow E \setminus E_{low}$    ▷ Subtract edges incident on low-degree nodes from $E$
    $F' \leftarrow \text{SpanningForest}(E', \{T_u\}_{u \in \mathcal{C}})$ ▷ Spanning forest on clusters $T_u, u \in \mathcal{C}$ using AGM sketches
    $E^* \leftarrow E_{low} \cup F \cup F'$
    return $E^*$
end procedure
```

It follows that if Bob outputs 1 iff $\{U, V\}$ occurs in H, he will have solved IND with probability at least $2/3$. By the abovementioned communication lower bound for IND, it follows that the space complexity of the streaming algorithm is $\Omega(r) = \Omega(nd)$. In the above lower bound the graph is on $sd = 18n$ vertices; rescaling n by a factor of 18 establishes the theorem. □

6. SPECTRAL SPARSIFIERS

In this section we show that Algorithm 3 of [KP12] can be easily implemented in the dynamic stream model using our multiplicative spanner construction in place of the Thorup-Zwick distance oracles used in [KP12]. The oracle required by [KP12] needs to output, given a pair of nodes $u, v \in V$, an estimate $\hat{d}(u, v)$ that satisfies $d(u, v) \leq \hat{d}(u, v) \leq \kappa d(u, v)$. Note that our multiplicative spanner construction provides such an estimate with $\kappa \leq 2^k$ when $\tilde{O}(n^{1+1/k})$ space is used, by our analysis in section 3.

There is an omission in the proof of a sampling lemma in [KP12] (Lemma 21), so we reprove this lemma here for our application. Also, we outline small space implementation details for other subroutines in [KP12] since [KP12] does not stress the small space requirement. We state the algorithms of [KP12] here for completeness. First, we round all edge weights to the nearest power of $(1+\epsilon)$ (our algorithm will produce a $(1 + O(\epsilon))$-sparsifier, which is fine by rescaling variables). Thus, it is sufficient to construct sparsifiers of unweighted graphs (at the expense of a loss of a factor of $\frac{1}{\epsilon}\log(w_{max}/w_{min})$ in runtime and space complexity).

6.1 Estimating connectivities

We first give a version of the ESTIMATE algorithm from [KP12] that we can use to estimate robust connectivities in the dynamic streaming model. Note that we cannot execute this algorithm directly in the dynamic streaming model since we do not have access to the edge set E of the graph explicitly. This, however, is not necessary, since we can perform estimates for any pair $(u, v) \in \binom{V}{2}$ on demand, i.e. we obtain an oracle for computing estimates of robust connectivities $\hat{q}_{\kappa,\delta}$. This version of the ESTIMATE algorithm of [KP12] is given as Algorithm 4 below.

Algorithm 4 Estimation of robust connectivities

```
1: procedure ESTIMATE($G, \kappa, \delta$)
2:                                                    ▷ Preprocessing
3:     for $j = 1$ to $J$ do                           ▷ $J = O(\log n/\delta^2)$
4:         Set $E_t^j \leftarrow \emptyset$ for $t \in [1 : T]$.        ▷ $T = \log n^4$
5:         For each $e \in E$ add $e$ to $E_1^j$.
6:         for $t = 1$ to $T - 1$ do
7:             Add each $e \in E_t^j$ to $E_{t+1}^j$ independently with probability 1/2.
8:         end for
9:     end for
10:    for $t = 1$ to $T$ do
11:        Construct a distance oracle $O_t^j$ for $E_t^j$, $j \in [1 : J]$.
12:    end for
13:                                           ▷ Query time, on input $(u, v)$
14:    for $t = 1$ to $T$ do
15:        for $j = 1$ to $J$ do                       ▷ $J = O(\log n/\delta^2)$
16:            If $O_t^j(u, v) > \kappa^2$ then $\eta^j(t) \leftarrow 1$
17:            else $\eta^j(t) \leftarrow 0$
18:        end for
19:    end for
20:    $\hat{q}_{\kappa,\delta}(e) \leftarrow 2^{-t}$, where $t$ is the smallest such that $|\{j : \eta^j(t) = 1\}| \geq (1 - \delta)J$
21:    return $\hat{q}_{\kappa,\delta}$
22: end procedure
```

These estimates can be used for sampling as shown in Algorithm 5. Algorithm 5 uses a sequence of subsets of edges of G. Let $E_j, j = 0, \ldots, H = \log_2 n^2$ contain edges of G sampled independently at rate 2^{-j}. Let $q(e) = \hat{q}_{\kappa,\epsilon}(e), e \in E$ be the vector of sampling parameters obtained from ESTIMATE(G, κ, ϵ). We now show how to use the spanner construction primitive to sample edges $e \in E$ with probability proportional to $q(e)$. Since we use the spanner construction algorithm to sample, the sampling is imperfect. However, we show that the quality of our sampling procedure is sufficient to produce a spectral sparsifier.

6.2 Sampling using augmented spanners

Consider an execution of our 2-pass spanner construction algorithm (Algorithms 1 and 2) and let $\Lambda(R) \subseteq \binom{V}{2}$ define the set of locations in the adjacency matrix of the graph accessed by the algorithm, where R is the random seed. Recall that $\Lambda(R) = \Lambda_1(R) \cup \Lambda_2(R)$ as defined in Claims 16 and 18.

CLAIM 20. *Algorithms 1 and 2 can be augmented to output all edges in the set of locations $\Lambda(R)$ that their execution path depends on with probability $1 - 1/poly(n)$ over the choice of the random seed.*

PROOF. Follows by Claim 16 and Claim 18. □

We denote this augmented construction by AUGMENTEDSPANNER(E, κ), where E is the input set of edges, and κ is the stretch parameter. (note that this algorithm outputs all edges of G that belong to $\Lambda(R)$). As in [KP12], our sampling procedure is given by

Algorithm 5 Sampling using an augmented spanner construction

```
1: procedure SAMPLE-AUGMENTED-SPANNER($G$, $q$, $\kappa$)
2:     for $j = 1, \ldots, H$ do                       ▷ $H \leftarrow \log_2 n^2$
3:         $E_j \leftarrow$ random sample of edges of $G$ at rate $2^{-j}$
4:     end for
5:     for $j = 1, \ldots, H$ do
6:         $S_j \leftarrow$ AUGMENTED-SPANNER$(V, E_j, \kappa)$     ▷ $\kappa$ is the stretch parameter
7:         For each $e \in S_j$, assign weight 0 to $e$ if $q(e) \neq 2^{-j}$, otherwise assign weight $2^j$.
8:     end for
9:     Return the weighted collection $S_1 \cup \ldots \cup S_H$.
10: end procedure
```

The sampling process is then given by Algorithm 6. Note that Algorithm 6 differs from the corresponding algorithm in [KP12] in that parameter Z is chosen as $Z \leftarrow \Theta(\kappa^2 \log n/((1 - \epsilon)\epsilon^3))$ as opposed to $Z \leftarrow \Theta(\log^3 n/((1 - \epsilon)\epsilon^3))$. This is because κ was instantiated to $\log n$ for this algorithm in [KP12].

Algorithm 6 Sparsification via spanners

```
1: procedure AUGMENTED-SPANNER-SPARSIFY($G$, $q$, $\epsilon$, $\kappa$)
2:     $q \leftarrow$ ESTIMATE$(G, \kappa, \epsilon)$.
3:     $Z \leftarrow \Theta(\kappa^2 \log n/((1 - \epsilon)\epsilon^3))$
4:     for $t = 1, \ldots, Z$ do
5:         $X_t \leftarrow$ SAMPLE-AUGMENTED-SPANNER$(G, q, \kappa)$
6:     end for
7:     return $\frac{1}{Z}(X_1 + \ldots + X_H)$
8:     ▷ Addition above refers to taking a union of $X_1, \ldots, X_H$ and scaling weights by $1/Z$
9: end procedure
```

We now prove that AUGMENTED-SPANNER-SPARSIFY(G, q, ϵ, Z) produces a spectral sparsifier whp. We will use the following

THEOREM 21. *Let $G = (V, E)$ be an unweighted graph, $\epsilon > 0$ a precision parameter. Let sampling parameters τ_e satisfy $\tau_e \geq R_e/M$ for a parameter M. Let $C > 0$ be a sufficiently large* *constant, and for $j = 1, \ldots, (C/\epsilon^2)M \log n$, $e \in E$ let Y_e^j be independent random variables that equal $1/\tau_e$ with probability τ_e and 0 otherwise. Let G' contain each edge $e \in E$ with weight $\sum_{j=1}^{(C/\epsilon^2)M \log n} Y_e^j$. Then $(1-\epsilon)G \prec G' \prec (1+\epsilon)G$ whp.*

A proof of this theorem follows easily using concentration inequalities for positive semidefinite matrix-valued random variables proved in [Tro12]. We now show that our sampling procedure works nearly as well as independent sampling analyzed in Theorem 21.

LEMMA 22. *Algorithm 6 produces a $(1 \pm O(\epsilon))$-spectral sparsifier of the input graph G whp.*

PROOF. Set $Z = O(\kappa^2 \log n/((1-\epsilon)\epsilon^3))$, as in line 3 of Algorithm 6. By Lemma 19 of [KP12] we have that

$$\hat{q}_{\kappa,\epsilon}(e) = \Omega(\epsilon R_e/\kappa^2) \qquad (1)$$

for all $e \in E$. The complication is that in an invocation of SAMPLE-AUGMENTED-SPANNER(G, q, κ) the function AUGMENTED-SPANNER(E_j, κ) may not necessarily output all edges $e \in E_j$ such that $\hat{q}_{\kappa,e} = 2^{-j}$. It is, however, guaranteed to output at least an $1-\epsilon$ fraction of such edges in expectation by the definition of $\hat{q}_{\kappa,e}$ (which we denote by $q(e)$ in what follows). More precisely, for each $e = (u,v)$ such that $q(e) = 2^{-j}$ the set E_j contains no path between u and v of length less than κ with probability at least $1-2\epsilon$. Indeed, since the estimation procedure invokes the spanner construction algorithm with stretch κ, the pair (u,v) would not have been assigned $q(e) = 2^{-j}$ otherwise. Thus, the set $E_j \setminus \{e\}$ contains no path between u and v of length less than κ with probability at least $1-2\epsilon$, and hence our κ-stretch spanner outputs e with probability at least $1-2\epsilon$ over the choice of $E_j \setminus \{e\}$. We now introduce relevant notation and show that this implies that our algorithm outputs a spectral sparsifier.

For each $s = 1, \ldots, Z$ and each $j = 1, \ldots, H = \log_2 n^2$ let $E_{s,j}$ denote the sets sampled in the s-th invocation of AUGMENTED-SPANNER in line 5 of Algorithm 6. For $e \in E$ let $j(e)$ be such that $q(e) = 2^{-j(e)}$. Recall that each invocation of AUGMENTED-SPANNER outputs all edges e in $E_{s,j} \cap \Lambda(R_{s,j})$, where $R_{s,j}$ is the random seed used. Let $X_e^s = 1$ if $e \in E_{s,j(e)}$ and 0 otherwise. First, we have

CLAIM 23. *Fix j. Conditional on $\Lambda(R)$ the random variables $\{X_e^j\}_{e \in E \setminus \Lambda(R)}$ are statistically n^{-C}-close to independent Bernoulli 0/1 with expectation 2^{-j}, where the choice of $C > 0$ only affects the space requirement of our algorithm by a constant factor.*

PROOF. Recall that $\Lambda_1(R)$ and $\Lambda_2(R)$ are the locations in the adjacency matrix that affect the execution path of our algorithm, conditional on our sketches succeeding. The probability of at least one sketch failing can be chosen to be n^{-C} by choosing parameters appropriately (and the choice of C only affects space complexity by constant factors). □

For each s, j and $\{u,v\} \in \binom{V}{2}$ let $c_{uv}^{s,j} = 1$ if $\{u,v\} \in \binom{V}{2} \setminus \Lambda(R_{s,j})$ and 0 otherwise. We say that a pair $\{u,v\}$ is *covered* in invocation (s,j) if $c_{uv}^{s,j} = 1$. Note that if an edge $e = (u,v)$ is covered, then the constructed spanner must contain a path of length bounded by κ between u and v. Thus, we have $\mathbf{Pr}[c_{uv}^{s,j} = 1] \leq 2\epsilon$ for any $e = (u,v)$ with $q(e) = 2^{-j}$ whp.

Since samples are independent for different $s = 1, \ldots, Z$ and $Z \geq (C/\epsilon)\log n$ for a sufficiently large constant $C > 0$, one has for any $e = (u,v)$ that

$$\sum_{s=1}^{Z} c_e^{s,j(e)} \leq 4\epsilon Z \qquad (2)$$

with probability at least $1 - n^{-20}$.

The Laplacian of the sampled graph G' output by AUGMENTED-SPANNER-SPARSIFY is given by

$$L_{G'} = \frac{1}{Z}\sum_{s=1}^{Z}\sum_{e \in E}(1 - c_e^{s,j(e)})\frac{1}{q(e)}X_e^s \cdot \mathbf{e} \cdot \mathbf{e}^T, \qquad (3)$$

where for an edge $e = (u,v)$ we denote the vector in $\mathbb{R}^V$ with -1 at u and $+1$ at v by $\mathbf{e}$. Let

$$L_{\widetilde{G}} = \frac{1}{Z}\sum_{s=1}^{Z}\sum_{e \in E}\frac{1}{q(e)}X_e^s \cdot \mathbf{e} \cdot \mathbf{e}^T. \qquad (4)$$

By Theorem 21 we have with high probability

$$(1 - O(\epsilon))L_G \prec L_{\widetilde{G}} \prec (1 + O(\epsilon))L_G. \qquad (5)$$

Let $L_H = \frac{1}{Z}\sum_{s=1}^{Z}\sum_{e \in E} c_e^{s,j(e)}\frac{1}{q(e)}X_e^s \cdot \mathbf{e} \cdot \mathbf{e}^T$. By Claim 23 conditional on $\Lambda(R_{s,j})$ and failure events for our sketching primitives not happening, the random variables $\{X_e^j\}$ for $e \in E \setminus \Lambda(R_{s,j})$ are statistically $n^{-C'}$-close to independent Bernoulli 0/1 with expectation 2^{-j} (where the choice of constant C' only affects our space requirements by constant factors). Let $\widetilde{X}_e^s$, $s = 1, \ldots, Z, e \in E$ be such Bernoulli 0/1 random variables coupled with X_e^s on a $1 - n^{-C'}$ fraction of probability space. Letting $L_{\widetilde{H}} = \frac{1}{Z}\sum_{s=1}^{Z}\sum_{e \in E} c_e^{s,j(e)}\frac{1}{q(e)}\widetilde{X}_e^s \cdot \mathbf{e} \cdot \mathbf{e}^T$, we have that $\mathbf{Pr}[L_H \neq L_{\widetilde{H}}] \leq n^{-C'}$. Let R' denote the random bits (random seed R of our algorithm and random choices made in sampling edges that belong to $\Lambda(R)$) that determine the values of $c_e^{s,j}$. By (2) with probability at least $1 - n^{-20}$ one has $\sum_{s=1}^{Z} c_e^{s,j(e)} \leq 4\epsilon Z$ for each $e \in E$ (call this event $\mathcal{E}$). Without loss of generality (and to simplify notation) we assume that conditional on this event one has $c_e^{s,j} = 0$ for all $s > 4\epsilon Z$ and all e (note that this does not change the distribution of $L_{\widetilde{H}}$ since all $\widetilde{X}_e^s$ are iid for fixed e and different s).

Thus, conditional on $\mathcal{E}$, $L_{\widetilde{H}}$ is stochastically dominated by $L_{H,*} := \frac{1}{Z}\sum_{s=1}^{4\epsilon Z}\sum_{e \in E}\frac{1}{q(e)}W_e^s \cdot \mathbf{e} \cdot \mathbf{e}^T$, where $W_e^s = 1$ independently with probability $2^{-j(e)}$ and 0 otherwise. This is because for each fixed $R' \in \mathcal{E}$ and each $e \in E$ the random variable $c_e^{s,j(e)}\widetilde{X}_e^s$ is either 0 or independent Bernoulli 0/1 with expectation $2^{-j(e)}$, and hence W_e^s and $c_e^{s,j(e)}\widetilde{X}_e^s$ can be coupled so that $c_e^{s,j(e)}\widetilde{X}_e^s \leq W_e^s$. Under this coupling we have that for each $R' \in \mathcal{E}$ the difference $L_{H,*} - L_{\widetilde{H}} = \frac{1}{Z}\sum_{s=1}^{4\epsilon Z}\sum_{e \in E}\frac{1}{q(e)}(W_e^s - c_e^{s,j(e)}\widetilde{X}_e^s) \cdot \mathbf{e} \cdot \mathbf{e}^T$ is a positive semidefinite matrix with probability 1 (since it is a sum of positive semidefinite rank-1 matrices).

By Theorem 21 together with (1) we have that $L_{\widetilde{H}} \prec L_{H,*} \prec O(\epsilon) \cdot L_G$ with high probability (we are using the fact that $\mathbf{Pr}[\mathcal{E}] \geq 1 - n^{-20}$). We also have $\mathbf{Pr}[L_H \neq L_{\widetilde{H}}] < n^{-C'}$, so $L_H \prec O(\epsilon) \cdot L_G$ with high probability. Finally, since $L_{G'} = L_{\widetilde{G}} - L_H$ by (3) and (4), this implies by (5) that $(1 - O(\epsilon))L_G \prec L_{G'} \prec (1 + O(\epsilon))L_G$ with high probability, as required.

6.3 Randomness in small space and setting parameters

Note that in order to implement our algorithm in the dynamic stream model it is sufficient to implement the partitioning of the set E of edges of G into classes $E_j, j = 1, \ldots, H$ needed in Algorithm 5 and into sets $E_j^t, j = 1, \ldots, J, t = 1, \ldots, T$ needed in Algorithm 4 (recall that $J = O(\log n/\epsilon^2)$ and $T = \log n^4$). We show how to generate the first sequence of sets in the dynamic streaming model. The second is analogous. We choose uniformly random numbers $h_{uv}^j, u, v \in V, j = 1, \ldots, H$ such that h_{uv}^j are independent and uniform in $[0,1]$ (rather, a sufficiently fine discretization of $[0,1]$ with step size $1/\text{poly}(n)$, say). Then a potential edge (u,v) belongs to E_j iff $h_{uv}^j \le 2^{-j}$. This yields a filtered input stream for each $j = 1, \ldots, H$, which is given as input to our 2-pass distance oracle. Since the approximation guarantees provided by our oracle satisfy the requirements of [KP12], the same analysis holds, and hence calls to ESTIMATE give the same guarantees. It remains to verify that SAMPLE-AUGMENTED-SPANNER and AUGMENTED-SPANNER-SPARSIFY can be implemented in the dynamic stream model with small space. Note that the sequence of sets E_j used in SAMPLE-AUGMENTED-SPANNER can be emulated as we just described. Finally, note that we assumed perfect randomness in the choice of the subsets E_j^t, as required by [KP12], which results in $\Omega(n^2)$ space requirement for the random bits. Since our algorithm works in space $S = O(n^{1+o(1)}/\epsilon^4)$ (after setting parametes appropriately, as we do below), we can replace this perfect randomness with Nisan's pseudoradom generator, taking space $O(n^{1+o(1)}/\epsilon^4)$, in the same way as in [AGM12b] (see section 3.4; the argument is identical, so we do not repeat it here).

It remains to set parameters. Let $\kappa = 2^k$ denote the upper bound on the stretch provided by our algorithm with $\tilde{O}(n^{1+1/k})$ space. Then Algorithm 6 requires $Z = O(\kappa^2(\log n)/\epsilon^3)$ invocations of SAMPLE-AUGMENTED-SPANNER, each of which requires $H = \log_2 n^2$ copies of our spanner construction algorithm, for a total space of $\tilde{O}(\kappa^2 n^{1+1/k}/\epsilon^3) = \tilde{O}(2^{2k} n^{1+1/k}/\epsilon^3)$. A similar calculation shows that the space complexity of ESTIMATE is $\tilde{O}(n^{1+1/k}(\log n^4)/\epsilon^2)$. We now choose $k = \sqrt{\log n}$ to obtain space complexity $\tilde{O}(2^{2\sqrt{\log n}} n^{1+1/\sqrt{\log n}}/\epsilon^4) = n \cdot 2^{O(\sqrt{\log n})}/\epsilon^4$ as desired (an extra factor of $\frac{1}{\epsilon}\log\frac{w_{max}}{w_{min}}$ appears for weighted graphs).

Corollary 2 *For any $\epsilon > 0$ there exists an algorithm for constructing an ϵ-spectral sparsifier of a graph G presented in the dynamic streaming model using $O(n 2^{O(\sqrt{\log n})}(\log(w_{max}/w_{min}))/\epsilon^4)$ space and two passes over the stream.*

7. REFERENCES

[ACIM99] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

[AGM12a] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. *SODA*, pages 459–467, 2012.

[AGM12b] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. *PODS*, pages 5–14, 2012.

[AGM13] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. *APPROX-RANDOM*, pages 1–10, 2013.

[Bas08] Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

[BK96] András A. Benczúr and David R. Karger. Approximating *s-t* minimum cuts in $\tilde{O}(n^2)$ time. *STOC*, pages 47–55, 1996.

[BKMP10] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (alpha, beta)-spanners. *ACM Transactions on Algorithms*, 7(1), 2010.

[BS03] Surender Baswana and Sandeep Sen. A simple linear time algorithm for computing a (2k-1)-spanner of o($n^{1+1/k}$) size in weighted graphs. *ICALP*, pages 384–296, 2003.

[BS06] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $o(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.

[BS07] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007.

[BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *STOC*, pages 255–262, 2009.

[Che13] Shiri Chechik. New additive spanners. In *SODA*, pages 498–512, 2013.

[CM06] G. Cormode and S. Muthukrishnan. Combinatorial algorithms for compressed sensing. *Structural Information and Communication Complexity*, 4056:280–294, 2006.

[FHHP11] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *STOC*, pages 71–80, 2011.

[GM12] Sudipto Guha and Andrew McGregor. Graph synopses, sketches, and streams: A survey. *PVLDB*, 5(12):2030–2031, 2012.

[KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *FOCS*, pages 235–244, 2010.

[KMP11] 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.

[KNR99] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52, 2010.

[KP12] Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. *ITCS*, pages 393–398, 2012.

[MN06] Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *FOCS*, pages 109–118, 2006.

[SS08] D.A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *STOC*, pages 563–568, 2008.

[ST04] D.A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. *STOC*, pages 81–90, 2004.

[ST11] D. Spielman and S. Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[Tro12] Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Found. Comput. Math.*, 12(4):389–434, August 2012.

[TZ01] M. Thorup and U. Zwick. Approximate distance oracles. *STOC*, 2001.

[TZ06] Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. *SODA*, pages 802–809, 2006.