Near-optimal Linear Sketches and Fully-Dynamic Algorithms for Hypergraph Spectral Sparsification

Sanjeev Khanna[*]   Huan Li[†]   Aaron (Louie) Putterman[‡]

February 7, 2025

Abstract

A hypergraph spectral sparsifier of a hypergraph G is a weighted subgraph H that approximates the Laplacian of G to a specified precision. Recent work has shown that similar to ordinary graphs, there exist $\widetilde{O}(n)$-size hypergraph spectral sparsifiers. However, the task of computing such sparsifiers turns out to be much more involved, and all known algorithms rely on the notion of balanced weight assignments, whose computation inherently relies on repeated, complete access to the underlying hypergraph. We introduce a significantly simpler framework for hypergraph spectral sparsification which bypasses the need to compute such weight assignments, essentially reducing hypergraph sparsification to repeated effective resistance sampling in *ordinary graphs*, which are obtained by *oblivious vertex-sampling* of the original hypergraph.

Our framework immediately yields a simple, new nearly-linear time algorithm for nearly-linear size spectral hypergraph sparsification. Furthermore, as a direct consequence of our framework, we obtain the first nearly-optimal algorithms in several other models of computation:

1. The first nearly-optimal size *linear sketches* for spectral hypergraph sparsification. For hypergraphs on n vertices, with hyperedges of arity $\leq r$ and with $\leq m$ hyperedges, these sketches require only $\widetilde{O}(nr \operatorname{polylog}(m)/\epsilon^2)$ bits and recover a $(1 \pm \epsilon)$ spectral-hypergraph sparsifier with high probability. It is known that linear sketches require $\Omega(nr \log(m))$ bits even for the easier task of cut sparsification (Khanna-Putterman-Sudan FOCS 2024).

2. The first nearly-optimal *fully dynamic* $(1 \pm \epsilon)$ spectral (and cut) hypergraph sparsification algorithm. Our algorithm has an amortized, expected update time of $\widetilde{O}(r \operatorname{polylog}(m)/\epsilon^2)$, and produces sparsifiers with $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges. This is nearly-optimal as even to read a single hyperedge takes time $\Omega(r)$.

3. The first nearly-optimal algorithm for *online* hypergraph spectral sparsification. On a sequence of m (unweighted) hyperedges, our algorithm creates a $(1 \pm \epsilon)$ hypergraph spectral sparsifier with $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges in an online manner. When $m \leq \operatorname{poly}(n)$, this improves upon the work of Soma, Tung, and Yoshida (IPCO 2024) by a factor of r, who created online sparsifiers with $\widetilde{O}(n(r+\log(m))/\epsilon^2)$ hyperedges. We complement this result with an $\Omega(n \log(m))$ lower-bound for any online sparsifier, thus provably separating the classical and online settings.

Our main conceptual and technical contributions are introduction of (a) the *vertex sampling* framework to reduce spectral sparsification in hypergraphs to ordinary graphs, and (b) a notion of *collective energy* in hypergraphs that may be seen as a continuous generalization of k-cuts.

---

[*]School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia, PA. Email: `sanjeev@cis.upenn.edu`. Supported in part by NSF awards CCF-2008305 and CCF-2402284.

[†]School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia, PA. Email: `huanli@cis.upenn.edu`. Supported in part by NSF awards CCF-2008305 and CCF-2402284.

[‡]School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts, USA. Supported in part by the Simons Investigator Awards of Madhu Sudan and Salil Vadhan, NSF Award CCF 2152413 and a Hudson River Trading PhD Research Scholarship. Email: `aputterman@g.harvard.edu`.

Contents

1 Introduction

Ever since its conception in the work of Karger [Kar93], *graph sparsification* has been a powerful algorithmic tool in the design of efficient graph algorithms. Roughly speaking, given a graph $G = (V, E)$, sparsification is the process of choosing a re-weighted subgraph $G' \subseteq G$ such that certain properties of G are preserved. In the works of Karger [Kar93] and Benczúr and Karger [BK96], these properties took the form of the *cut sizes in* the graph, and this research culminated in the design of an $\widetilde{O}(n + m)$ time algorithm that produces a "nearly-linear size cut-sparsifier." Specifically, they showed that there exists a re-weighted subgraph of G with only $O(n \log(n)/\epsilon^2)$ edges that preserves the weight of every cut to a $(1 \pm \epsilon)$-factor. This result continues to play a central role in obtaining significantly more space-efficient and time-efficient algorithms for cut and flow problems on graphs across many models of computation.

Subsequently, other works extended the notion of cut sparsification in different directions. Perhaps most notably, the landmark works of Spielman and Teng [ST11], Spielman and Srivastava [SS11], and Batson, Spielman, and Srivastava [BSS12] studied *spectral* graph sparsification. Here, instead of simply preserving the cuts in the graph, their sparsifiers preserve the entire spectrum of the graph Laplacian. Recall that for a graph G, the Laplacian $L_G = D - A$, where D is a diagonal matrix of vertex degrees, and A is the adjacency matrix. One can check that for $S \subseteq [n]$, $\mathbf{1}_S^T L_G \mathbf{1}_S = |\mathrm{cut}_G(S)|$. Thus, preserving the spectrum of L_G to a $(1 \pm \epsilon)$ factor is a *stronger* notion than $(1 \pm \epsilon)$ cut-sparsification. Yet, [ST11, SS11, BSS12] still showed the existence of spectral sparsifiers of (nearly) linear size.

Paralleling the work in graphs, recent efforts in sparsification have focused heavily on the *hypergraph* setting. Originally proposed in the cut setting by Kogan and Krauthgamer [KK15], here one is tasked with selecting a re-weighted *sub-hypergraph*, such that all cuts have their weight preserved to a $(1 \pm \epsilon)$-factor. As in graphs, a cut is given by a subset $S \subseteq V$ of vertices, and a hyperedge $e \subseteq V$ is said to be crossing the cut if $e \cap S \neq \emptyset$ and $e \cap (V - S) \neq \emptyset$. Likewise, there is a notion of a hypergraph Laplacian (see [CLTZ18]), where for a hypergraph $H = (V, E)$ and a vector $x \in \mathbb{R}^n$, we say that

$$Q_H(x) = \sum_{e \in E} w_e \cdot \max_{(u,v) \in e} (x[u] - x[v])^2.$$

Initially studied by Soma and Yoshida [SY19] and Bansal, Svensson, and Trevisan [BST19], a $(1 \pm \epsilon)$ hypergraph spectral sparsifier H' of a hypergraph H is thus a re-weighted sub-hypergraph such that $\forall x \in \mathbb{R}^n, (1 - \epsilon) Q_H(x) \leq Q_{H'}(x) \leq (1 + \epsilon) Q_H(x)$.

As with graphs, for a set $S \subseteq V$, it is also the case that $Q_H(\mathbf{1}_S) = |\mathrm{cut}_H(S)|$, and hence hypergraph spectral sparsification generalizes cut sparsification. In a long line of works [CKN20, KKTY21b, KKTY21a, Lee23, JLS23, JLLS23, OST23, Qua24, KPS24a] the existence of nearly-optimal size $\widetilde{O}(n/\epsilon^2)$ cut and spectral sparsifiers was established, along with a host of efficient algorithms for computing such sparsifiers.

1.1 Computing Sparsifiers in Modern Computational Settings

However, the extensive research into efficient algorithms for hypergraph sparsifiers has so far primarily focused on the classical model of computation where the underlying hypergraph is static and the algorithm has unrestricted random access to the hypergraph. In contrast, one highlight of the extensive literature on graph sparsification algorithms has been the deployment of these algorithms in modern computational settings where either the underlying graph is not static or the algorithm has only a restricted access to the underlying graph, say, via *linear measurements* only. For instance, the work of [AGM12], and [KLM+14] established algorithms for creating cut and spectral sparsifiers of graphs in the *linear sketching* model of computation which require only $\widetilde{O}(n/\epsilon^2)$ bits of space. These algorithms are then immediately amenable to other computational settings such as the massively parallel computation (MPC) model (see e.g., [ACL+22, AKLP22]) and the *dynamic streaming* setting (where edges in the stream can be both inserted and deleted).

In a different vein, the work of [ADK+16] studied algorithms for *dynamically* maintaining sparsifiers of ordinary graphs. In this model, the graph is no longer static but is undergoing edge insertions and deletions, and the goal is to maintain a sparsifier without recomputing it from scratch after each update. The work of [ADK+16] showed that one can maintain a $(1 \pm \epsilon)$-cut sparsifier of a graph with worst-case update time $\text{poly}(\log(n), 1/\epsilon)$, and spectral sparsifiers with an amortized update time of $\text{poly}(\log(n), 1/\epsilon)$. Beyond this, there has also been work on graph sparsification in the *small space* regime by [DMVZ20] (i.e., designing spectral sparsification algorithms that use a small work-tape but have an unrestricted access to the input graph) and development of *deterministic* algorithms for computing spectral sparsifiers of graphs by Batson, Spielman, and Srivastava [BSS09].

In contrast, hypergraph sparsification is well-understood only in a small number of settings. Guha, McGregor, and Tench [GMT15] gave the first *linear sketches* for hypergraph cut-sparsifiers. However, the linear sketches of [GMT15] require $\widetilde{O}(nr^2 \text{polylog}(m)/\epsilon^2)$[1] bits to recover a $(1 \pm \epsilon)$ cut-sparsifier for hypergraphs on n vertices, with at most m hyperedges of arity bounded by r. A very recent work of Khanna, Putterman, and Sudan [KPS24b] constructed linear sketches for hypergraph cut-sparsifiers that achieve a nearly-optimal size of $\widetilde{O}(nr \log(m)/\epsilon^2)$ bits. As mentioned above, this immediately yields space-efficient dynamic streaming algorithms and communication-efficient MPC algorithms. Finally, the very recent work of Soma, Tung, and Yoshida [STY24] studied hypergraph spectral sparsification in the *online* model. Here, hyperedges arrive one at a time, and the sparsifier must decide, upon each arrival, whether to keep the hyperedge or not: if the hyperedge is kept, the algorithm must decide its weight immediately. The work of [STY24] shows that one can compute $(1 \pm \epsilon)$ hypergraph spectral sparsifiers with $\widetilde{O}(n(r + \log(m))/\epsilon^2)$ hyperedges, where r is the maximum arity of any hyperedge. However, to the best of our knowledge, neither hypergraph cut-sparsification nor spectral-sparsification has been studied in the fully dynamic setting.

To summarize, there are many computational models where our understanding of hypergraph sparsification *significantly* lags behind our understanding of the graph setting, with either no non-trivial results known or there is a striking gap between upper bounds achieved by algorithmic results and known lower bound results.

Because spectral hypergraph sparsification has found numerous applications in e.g., semi-supervised learning [ZHTC20], clustering [LVS+21, ZLS22], and practical compression [BWG+19], designing efficient sparsifiers is ever more important. Indeed, in these modern applications, one is often facing a staggering volume of data, which is itself evolving in real-time. Thus, these applications vastly benefit from the ability to process this kind of data in a space- and time-efficient manner.

Motivated by this, our work introduces a new unifying framework that significantly narrows these gaps in our understanding by providing nearly-tight algorithms for hypergraph sparsification (both cut and spectral) in the settings of linear sketching, fully-dynamic algorithms, and online algorithms. In the following section, we describe in detail our contributions to hypergraph sparsification in these settings.

1.2 Our Contributions

As a step towards overcoming the gap in our understanding of computing hypergraph sparsifiers in restricted settings, we introduce a general framework for reducing hypergraph sparsification to an inherently graph-theoretic task. Our starting point, as in prior works on spectral hypergraph sparsification, is the notion of a *weight assignment*. Vaguely speaking, hypergraph sparsification relies on computing appropriate estimates of the *importance* of each hyperedge. After computing a probability distribution p_e over the hyperedges $e \in E$, one then samples every hyperedge e with probability p_e, and reweighs the hyperedge to have weight $\frac{1}{p_e}$. Weight assignments provide a mechanism for computing these probabilities by carefully studying an associated ordinary *multi-graph* for the hypergraph at hand. Unfortunately, these algorithms *iteratively* refine the weight assignment to the underlying multi-graph, and therefore rely on repeated, complete access to the entire hypergraph, which we do *not* have in restricted models of

---

[1] Throughout the paper, we use $\widetilde{O}$ to hide $\text{polylog}(n)$ factors.

computation such as linear sketching. Our first contribution is to provide a simple algorithm for static hypergraph sparsification which removes the need for computing these weight assignments to identify important hyperedges, and instead relegates them entirely to the analysis.

Vertex-Sampling Framework for Hypergraph Sparsification. We introduce a notion of *vertex-sampling* whereby we repeatedly, independently, sample subsets of the vertex set V. On these vertex-sampled hypergraphs, we then associate a canonical *multi-graph* which replaces each hyperedge with a clique on the corresponding vertices. We show that sampling the multi-edges in proportion to their effective resistance (and then recovering the corresponding hyperedges for whichever multi-edges survive) suffices for recovering the hyperedges with "high importance". Further, when we build sparsifiers using this framework, the sparsity of the returned hypergraphs is *nearly-optimal*. While the technique of sampling vertices and looking at the resulting induced subgraphs has been used previously for ordinary graph sparsification [FKN21, CKL22], to the best of our knowledge, we are the first to use it for sparsification of *hypergraphs*.

Note that we only present our results for unweighted hypergraphs, but one can extend them to weighted graphs by standard geometric weight grouping tricks (see [KLM$^+$14] or [ADK$^+$16]). Specifically, we show the following result:

Theorem 1.1 (Informal). *There is a randomized algorithm that reduces the task of creating a $(1 \pm \epsilon)$ spectral-sparsifier for a hypergraph H with at most m hyperedges to the task of sampling multi-edges at rate $\mathrm{polylog}(m, n)/\epsilon^2$ times their effective resistance in a collection $\mathcal{G}$ of ordinary graphs. The graphs in $\mathcal{G}$ are created in an oblivious manner, and moreover, the total number of vertices in graphs in $\mathcal{G}$ is bounded by $n \, \mathrm{polylog}(m, n)$, thus ensuring that the sparsifier has only $\widetilde{O}(n \, \mathrm{polylog}(n, m)/\epsilon^2)$ hyperedges.*

Since our algorithm is conceptually fairly simple, we present its full pseudocode in Section 2.1. Our framework immediately yields a novel and simple nearly-linear time algorithm for computing hypergraph spectral sparsifiers of nearly-linear size (see Section 4.3).

Linear Sketches for Hypergraph Spectral Sparsification. Furthermore, due to the simplicity of our framework, we are able to directly take advantage of many well-developed techniques for estimating effective resistances in ordinary graphs and extend these to the hypergraph setting. Our first such extension is in designing *linear sketches* for hypergraph spectral sparsification, where we take advantage of the prior work of [KLM$^+$14] which designed linear sketches for effective resistance sampling in ordinary graphs. A linear sketch for a hypergraph H is specified by a matrix $P \in \mathbb{R}^{s \times 2^n}$, where a hypergraph is represented by an indicator vector in $\{0, 1\}^{2^n}$. The sketch itself is then $P \cdot H$, and the number of bits required to store the sketch is typically $\widetilde{O}(s)$ (there are only s entries in the sketch, but they may require more precision to represent). In this regime, we show the following:

Theorem 1.2. *There is a linear sketch for hypergraphs on n vertices, $\leq m$ hyperedges, and arity $\leq r$ which uses $\widetilde{O}(nr \, \mathrm{polylog}(m)/\epsilon^2)$ bits of space, and can be used to recover a $(1 \pm \epsilon)$ spectral-sparsifier with probability $1 - 1/\mathrm{poly}(n, m)$.*

Recall that the work of [KPS24b] showed a lower bound of $\Omega(nr \log(m))$ bits even for the simpler task of computing a $(1 \pm \epsilon)$ *cut-sparsifier* of a hypergraph using a linear sketch. Thus, a dependence on $\log m$ is unavoidable in the linear sketching setting. Our linear sketch has nearly the same size and succeeds even in computing $(1 \pm \epsilon)$ *spectral*-sparsifiers. As an immediate corollary of the above theorem, we also get the first nearly-optimal space complexity algorithm for recovering hypergraph spectral sparsifiers from dynamic streams, and show that its complexity nearly-matches that of the cut-sparsification setting:

Corollary 1.3. *For any $\epsilon \in (0, 1)$, there is a randomized dynamic streaming algorithm using $\widetilde{O}(nr \, \mathrm{polylog}(m)/\epsilon^2)$ bits of space that, for any sequence of insertions / deletions of hyperedges in an n-vertex unweighted hypergraph H with at most m edges of arity bounded by r, allows recovery of a $(1 \pm \epsilon)$ spectral-sparsifier of H with probability $1 - 1/\mathrm{poly}(n, m)$ at the end of the stream.*

3

Fully Dynamic Hypergraph Spectral Sparsification. We also present the first construction of *fully-dynamic* hypergraph sparsifiers with nearly-optimal update time and sparsity (for both the cut and spectral settings). Recall that in the fully-dynamic setting, there is a sequence of insertion / deletion operations of hyperedges, and after each such operation, the algorithm must output a list of changes that must be made to the existing hypergraph sparsifier such that it maintains a $(1 \pm \epsilon)$-approximation to the spectrum (or cut-sizes). Along these lines, we show the following:

Theorem 1.4. *There is a fully dynamic data structure that for a hypergraph H on n vertices with $\leq m$ hyperedges, and arity $\leq r$ undergoing a sequence of updates maintains a $(1 \pm \epsilon)$-spectral-sparsifier (and thus cut-sparsifier too) with probability $1 - 1/\text{poly}(n, m)$. The expected, amortized update time of the data structure is $O(r \, \text{polylog}(m, n) \cdot (1/\epsilon)^2 \log(1/\epsilon))$, and it maintains a sparsifier with $\widetilde{O}(n \, \text{polylog}(m)/\epsilon^2)$ hyperedges.*

Note that even to maintain a $(1 \pm \epsilon)$ *ordinary graph* spectral sparsifier in the fully dynamic setting, the best known result is an *amortized* update time of $\text{poly}(\log n, \epsilon^{-1})$ by [ADK+16]. We also observe that our update-time is essentially the best possible, as simply to read each hyperedge requires $\Omega(r)$ time. This result also resolves in the affirmative the open question 6.3 of [STY24] about the existence of efficient fully dynamic hypergraph spectral sparsification algorithms.

In real-world applications, fully-dynamic sparsifiers allow for maintaining a sparse representation of the hypergraph, even as it undergoes hyperedge insertions / deletions, without re-computing the sparsifier from scratch. Because of the broad applicability of hypergraph spectral sparsifiers for faster clustering algorithms [LM17, LM18, VBK20, VBK21, LVS+21, ZLS22], this immediately yields faster clustering as the underlying hypergraph evolves. We note that our results hold only for the oblivious adversary fully-dynamic model, where the hyperedge updates are independent of the current sparsifier (as opposed to adaptive adversary models, such as the one considered in the work of [BvdBG+22] for ordinary graphs).

Online Hypergraph Spectral Sparsification. Finally, we also present a new nearly-optimal algorithm for online hypergraph spectral sparsification. Here, the algorithm is presented with a stream of hyperedges to be inserted. After seeing each hyperedge, the algorithm must immediately decide whether or not to keep the hyperedge (as well as what weight should be assigned if kept). Once a hyperedge is kept, the algorithm cannot remove it and cannot change its weight. In the context of hypergraphs, this problem was first studied in [STY24], who showed the existence of online hypergraph spectral sparsification algorithms which require only $\widetilde{O}(n^2 \log(m))$ bits of space, and produce sparsifiers with $\widetilde{O}(n(r + \log(m))/\epsilon^2)$ hyperedges. Using our framework, we are able to improve the sparsity by a factor of r in the case when $m \leq \text{poly}(n)$:

Theorem 1.5. *There is an online hypergraph spectral sparsification algorithm, which for hypergraphs H on n vertices with $\leq m$ hyperedges, and arity $\leq r$ undergoing a sequence of insertions maintains a $(1 \pm \epsilon)$-spectral-sparsifier (and thus cut-sparsifier too) with probability $1 - 1/\text{poly}(n, m)$. The space complexity of the algorithm is $\widetilde{O}(nr \, \text{polylog}(m)/\epsilon^2)$ bits and the sparsifier contains $\widetilde{O}(n \, \text{polylog}(m)/\epsilon^2)$ hyperedges.*

The space complexity of $\widetilde{O}(nr \, \text{polylog}(m)/\epsilon^2)$ bits of our online algorithm matches *exactly* the bound proposed in the open question 6.2 of [STY24], thus resolves it affirmatively.

We complement the algorithmic result of Theorem 1.5 with an $\Omega(n \log(m))$ lower-bound on the number of hyperedges that must be kept by any online sparsifier in the worst-case (Section 10.1). Hence, unlike the static setting, in the online setting a dependence on $\log m$ is unavoidable.

The following table depicts the landscape of hypergraph sparsification in different models of computation, with our contributions highlighted in red which includes as a reference a linear sketching algorithm for hypergraph spectral sparsification that can be easily derived from known results (see Appendix A).

Work	Setting	Sparsification	Space Complexity	Time Complexity
[KPS24b]	Linear Sketching	Cut	$\widetilde{O}(nr\log(m)/\epsilon^2)$ bits	N/A
Naive (Appendix A)	Linear Sketching	Spectral	$\widetilde{O}(nr^2\operatorname{polylog}(m)/\epsilon^2)$ bits	N/A
This work	Linear Sketching	Spectral	$\widetilde{O}(nr\operatorname{polylog}(m)/\epsilon^2)$ bits	N/A
This work	Fully-Dynamic	Cut	$\widetilde{O}(n\operatorname{polylog}(m)/\epsilon^2)$ edges	$\widetilde{O}(r\operatorname{polylog}(m)/\epsilon^2)$
This work	Fully-Dynamic	Spectral	$\widetilde{O}(n\operatorname{polylog}(m)/\epsilon^2)$ edges	$\widetilde{O}(r\operatorname{polylog}(m)/\epsilon^2)$
[STY24]	Online	Spectral	$O(n(r+\log(m))/\epsilon^2)$ edges	N/A
This work	Online	Spectral	$\widetilde{O}(n\operatorname{polylog}(m)/\epsilon^2)$ edges	N/A

Table 1: Summary of our results.

2 Technical Overview

In this section, we give a high level overview of our techniques. Throughout this section, we focus on unweighted hypergraphs that only have hyperedges of arity (cardinality) between $[r, 2r]$, as one can extend our results to general weighted hypergraphs of arbitrary arity by geometrically grouping hyperedges by arity and weights.

2.1 A New Hypergraph Spectral Sparsification Framework

We now present our new unified framework for spectral hypergraph sparsification which as we will see later, lends itself easily to efficient implementation in a variety of different computational models. At a high-level, our meta-algorithm is based on a very natural strategy for sparsification. Given a hypergraph $H(V, E)$, identify a set $F \subseteq E$ of *critical* hyperedges that almost certainly needs to be included in any sparsifier. The remaining hyperedges in $E - F$ are not critical, and we can afford to subsample them with probability $1/2$ to create a new hypergraph $H'(V, E')$. One can then focus on the task of recursively building a sparsifier for the graph H' which contains only half as many hyperedges as the starting hypergraph H. The main challenge then is in efficiently identifying the set of critical hyperedges.

Our meta-algorithm is completely specified by a pair of subroutines that implement the strategy above. The first subroutine, given by Algorithm 1, reduces the problem of identifying critical hyperedges to effective resistance sampling in *ordinary* graphs. The second subroutine, given by Algorithm 2, then recursively builds a sparsifier by invoking Algorithm 1 to identify critical hyperedges at each level of recursion. The heart of our approach is Algorithm 1 for recognizing critical hyperedges, and we describe it next.

As mentioned before, our key new insight is that the task of identifying critical hyperedges can be reduced to effective resistance sampling in a suitable collection of ordinary graphs, generated by sub-sampling vertices of the hypergraph. Specifically, for a hypergraph $H = (V, E)$, *vertex-sampling* at rate p refers to sampling each vertex in V independently with probability p. If we denote the resulting vertex set by V', then it defines a new hypergraph $H' = (V', E')$, where $E' = \{e \cap V' : e \in E\}$. Another key notion that we will utilize is the notion of a *multi-graph* of a hypergraph H, denoted by $\Phi(H)$, which is an *ordinary graph* obtained by replacing each hyperedge with a clique on the vertices of the hyperedge. Note that since a pair of vertices may be contained in multiple hyperedges, this gives rise to a multi-graph. Notationally, we also write $R^G(u, v)$ for an ordinary (multi-)graph G to denote the effective resistance between u, v in G.

Algorithm 1: $\mathrm{VS}(H, \lambda)$

Input : A hypergraph H with $\leq m$ hyperedges of arity $[r, 2r]$, and an oversampling rate $\lambda \geq 1$.
Output: A set of hyperedges F.

1 Initialize $F \leftarrow \emptyset$.
2 for $r \operatorname{polylog}(n, m)$ rounds **do**
3 $\quad$ Vertex sample H at rate $1/r$ to get H', with $\Phi(H')$ being its multi-graph.
4 $\quad$ Independently sample each multi-edge (u, v) of $\Phi(H')$ with probability $\lambda \cdot R^{\Phi(H')}(u, v)$.
5 $\quad$ Let F' contain all hyperedges of H for which at least one associated multi-edge got sampled.
6 $\quad$ Let $F \leftarrow F \cup F'$ and delete F' from H.
7 end
8 return F.

Algorithm 2: HypergraphSpectralSparsify(H, ϵ, r, m)

Input : A hypergraph H with $\leq m$ hyperedges of arity $[r, 2r]$, and a parameter ϵ.
Output: A reweighted subgraph of H.

1 Let $H_0 = H$.
2 for $i = 0, \ldots, \log(m)$ **do**
3 $\quad$ Let $F_i = \mathrm{VS}(H_i, \operatorname{polylog}(n, m)/\epsilon^2)$.
4 $\quad$ Let $H_{i+1} = H_i - F_i$ with its hyperedges sampled at rate $1/2$.
5 end
6 return $\bigcup_{i=0}^{\log(m)} 2^i \cdot F_i$, i.e. put weight 2^i on hyperedges in F_i.

2.2 Analysis of Our Meta-Algorithm

In spectral sparsification of ordinary graphs, critical/important edges are defined to be edges whose effective resistance is $\Omega(1/\log n)$ [SS11]. For spectral sparsification of hypergraphs, previous work [KKTY21a, JLS23, Lee23] determined important edges based on the notion of a weight assignment which are computed by an iterative refinement approach. In contrast, Algorithm 1 above identifies important hyperedges by implementing effective resistance sampling in ordinary graphs that are obtained by *oblivious* vertex sampling of the original hypergraph, completely bypassing the need for computing a weight assignment. The analysis of Algorithm 1, however, will rely on the notion of weight assignments, to argue correctness of our approach.

Weight Assignments. Recall that given a hypergraph $H = (V, E)$, we can associate with it a *multigraph*, denoted by $G = \Phi(H)$, where for each hyperedge $e \in E$, we replace it with a clique K_e on the corresponding vertices. A *weight assignment* W assigns a weight w_f to every multi-edge in G such that for every hyperedge $e \in H$, the sum of the weights of the corresponding multi-edges in G is 1.

As shown in [KKTY21a, JLS23, Lee23], for any weight assignment W of G, sampling each hyperedge e with probability given by $p_e = \epsilon^{-2} \operatorname{polylog}(n) \cdot \max_{(u,v) \in e} R^{G(W)}(u, v)$ suffices to yield a $(1 \pm \epsilon)$-spectral sparsifier, where we write $G(W)$ to denote G with edges weighted by W, and write $R^{G(W)}(u, v)$ to denote the effective resistance between u, v in $G(W)$.

That is, the sampling rate for each hyperedge e is proportional to the *maximum* over all multi-edges $f = (u, v) \in e$ of the effective resistance between u, v in $G(W)$. Consequently, the important hyperedges are the ones that contain some u, v between which the effective resistance is at least $1/\operatorname{polylog}(n, m)$ in the graph $G(W)$. Naturally then, the difficulty comes in showing that one can compute weight assignments such that $\sum_{e \in E} p_e$ is not too large (in particular, the number of important hyperedges is $\widetilde{O}(n)$), as this then immediately yields small spectral hypergraph sparsifiers. Unfortunately, finding such weight

assignments requires an iterative refinement procedure that needs repeated, complete access to the entire hypergraph, which we do not have in modern models of computation such as linear sketching.

Bypassing Weight Assignments Algorithmically. Instead, we present a sampling procedure that operates on the underlying (unweighted) multi-graph, i.e. $\mathtt{VS}(H, \lambda)$ in Algorithm 1, to recover the important hyperedges. This procedure samples multi-edges without needing to compute a weight assignment. Then, for each sampled multi-edge f, we recover the corresponding hyperedge (i.e., the $e \in H$ such that $f \in e$). In order to show the correctness of our procedure, the key is to show that *there exists* a single weight assignment W^* to the multi-graph $\Phi(H)$ such that every hyperedge e with a large sampling rate under W^* is recovered by our procedure. For the remaining hyperedges which are not recovered, this weight assignment W^* is a certificate to the fact that their sampling rates are small, and therefore they do not yet need to be recovered. Crucially, we do *not* need to explicitly compute such a W^*, but only need its existence for our analysis.

There is a delicate and complex argument needed to show the existence of a weight assignment W^* to which we can couple the success of the scheme. In particular, our key technical theorem is the following:

Theorem 2.1 (Key Technical Theorem). *For any $\theta \in (0, 1)$, given a hypergraph H and its multi-graph $G = \Phi(H)$, there exists a single weight assignment W^* of G such that for every hyperedge $e \in H$, one of the following statements holds:*

(a) either $\max_{(u,v) \in e} R^{G(W^*)}(u, v) \le \theta$, or

(b) we have that $\Pr\left[e \in \mathtt{VS}(H, \lambda = \theta^{-1} \operatorname{polylog}(n, m))\right] \ge 1 - 1/\operatorname{poly}(n, m)$, where the $\operatorname{poly}(n, m)$ in the success probability depends on the $\operatorname{polylog}(n, m)$ in the parameter λ of Algorithm 1.

By setting $\theta = \epsilon^2 / \operatorname{polylog}(n, m)$ in Theorem 2.1, we get that there exists a weight assignment W^* such that each hyperedge either (i) can be recovered by our vertex sampling algorithm $\mathtt{VS}(H, \operatorname{polylog}(n, m))$ with high probability, or (ii) can be sampled with probability $1/2$ while preserving the entire spectrum to within $1 \pm \epsilon / \operatorname{polylog}(n, m)$. We thus classify the hyperedges satisfying (i) as important ones, and sample the rest with probability $1/2$ in Algorithm 2.

At a high level, the proof of Theorem 2.1 consists of two parts. First, we connect important hyperedges to a new notion of multi-way energy that we call *collective energy*[2], defined for a set of potential vectors $x_1, \ldots, x_k \in \mathbb{R}^n$ as

$$\mathcal{E}^H(x_1, \ldots, x_k) \overset{\text{def}}{=} \sum_{e \in H} \max_{(a,b) \in e} \left(\sum_{i=1}^{k} (x_i[a] - x_i[b])^2 \right). \tag{1}$$

Notably, $\mathcal{E}^H(x_1, \ldots, x_k)$ is solely determined by the hypergraph H and the potential vectors, without reference to any weight assignments. We derive that each important hyperedge is "witnessed" by a set of k potential vectors with small collective energy (in particular, $\le k \operatorname{polylog}(n, m)$). Then in the second part, we show that the existence of such a set of potential vectors witnessing important hyperedges can be in turn used to show that the witnessed hyperedges can be recovered by our vertex sampling algorithm with high probability. Crucially, we do *not* ever need to compute these potential vectors explicitly - they are only used in our analysis.

The formal statement of the first part is summarized in our *Collective Energy Lemma* (Lemma 2.4), and the formal statement of the second part is summarized in our *Vertex Sampling Lemma* (Lemma 2.5). We also refer the reader to our Proof of Theorem 2.1 for how we combine the two parts to prove the above theorem.

Due to space limitations, we defer a detailed discussion of the proof of Theorem 2.1 to Section 2.6, with the first part discussed in Section 2.6.1 and the second part discussed in Section 2.6.2.

For now, we proceed by understanding the correctness of Algorithm 2, as well as describing the implementation of our framework in modern models of computation.

---

[2]This may be seen as a continuous generalization of k-cuts, just as ordinary energy vs. 2-cuts; see Remark 2.1 below.

Correctness of Algorithm 2. We now show how Algorithm 2 leverages Theorem 2.1 into a simple algorithm for sparsifying the overall hypergraph. This type of framework is relatively standard, appearing in many works ([KX16, GMT15, ADK+16, CKL22, KPS24b] to name a few).

To understand the above algorithm, let us focus on the first iteration, as it turns out the reasoning extends to future iterations as well. When we run the vertex-sampling algorithm on $H_0 = H$, we recover some set $F_0 \subseteq H_0$ of hyperedges. By Theorem 2.1, we can then argue that with high probability, there exists a single weight assignment W^* for which *all remaining non-recovered hyperedges $H_0 - F_0$* would have maximum pairwise resistance $\leq \frac{\epsilon^2}{\text{polylog}(m,n)}$ in the multi-graph weighted by W^*. Per [JLS23], by choosing an appropriate polylog(n,m), when we now sample at rate $1/2$ to get H_1, it will be the case that $F_0 \cup 2 \cdot H_1$ is a $(1 \pm \epsilon)$ spectral-sparsifier for H_0 with high probability.

In general, we can extend this reasoning beyond the first iteration: in the ith iteration $F_i \cup 2 \cdot H_{i+1}$ is a $(1 \pm \epsilon)$-spectral sparsifier for H_i with high probability. By composing the sparsifiers, if $\epsilon \leq O(1/\log m)$, we see that the final returned result is a $(1 \pm O(\epsilon \log(m)))$-sparsifier to H with high probability. By instead running the sparsification algorithm with $\epsilon' = \epsilon/\log(m)$, we then get our desired result.

To see the sparsity of the sparsifier, we see that each hyperedge which is recovered must correspond to a multi-edge that is recovered. But, recall that we are doing effective resistance sampling on $r \, \text{polylog}(m,n)$ multi-graphs, each on $n \, \text{polylog}(n,m)/r$ vertices, with an oversampling factor of $\text{polylog}(m,n)/\epsilon^2$. On each multi-graph, we therefore recover $\widetilde{O}(\frac{n \, \text{polylog}(m)}{r\epsilon^2})$ multi-edges, and in total across all rounds, and all $\log(m)$ levels of sampling, we recover $\widetilde{O}(n \, \text{polylog}(m)/\epsilon^2)$ multi-edges. Because hyperedges are recovered only if a corresponding multi-edge is recovered, the final sparsity is therefore $\widetilde{O}(n \, \text{polylog}(m)/\epsilon^2)$ hyperedges. To summarize, we get the following lemma:

Lemma 2.2. *Given as input a hypergraph H and parameter $\epsilon \in (0,1)$, Algorithm 2 creates a $(1 \pm \epsilon)$ spectral sparsifier $\widetilde{H}$ of H with only $\widetilde{O}(n \, \text{polylog}(m)/\epsilon^2)$ hyperedges with probability $1 - 1/\text{poly}(n,m)$.*

In the next few sections, we show how to implement our new sparsification framework in various modern models of computation.

2.3 Linear Sketching Hypergraph Spectral Sparsifiers

The key observation is that Algorithm 2 relies only on the ability to sample ordinary edges in proportion to their effective resistance in various vertex-sampled multi-graphs. The work of [KLM+14] initiated the study of linear sketches for creating spectral-sparsifiers, and in fact, this work already shows the existence of a linear sketch (in small space) which can sample edges at rates proportional to their effective resistance.

Specifically, [KLM+14] showed that for a graph G on n vertices and a parameter $\epsilon > 0$, there is a linear sketch of size $\widetilde{O}(n/\epsilon^2)$ bits which can be used to recover a $(1 \pm \epsilon)$ spectral-sparsifier of G with high probability. Extending this linear sketch to work for our multi-graphs is straightforward, and requires only blowing up the space by a factor of $\widetilde{O}(r \log(m))$ bits (a factor of roughly r for the "universe size", i.e., number of distinct multi-edge slots, and a factor of roughly $\log(m)$ for the support size, i.e., the maximum number of multi-edges ever present in one multi-graph). Because this linear sketch already enables sampling in proportion to effective resistance, all that remains to implement Algorithm 2 is to recover the indices of the corresponding hyperedges whenever we recover a multi-edge. However, this is essentially canonical: the hypergraph is represented by a vector in $\{0,1\}^{\binom{n}{r}}$, and the corresponding multi-graph is represented by a vector in $\{0,1\}^{\binom{n}{r} \cdot r^2}$ (where each hyperedge creates $O(r^2)$ multi-edge slots). Whenever the linear sketch recovers a multi-edge, this is reported as an index in $[\binom{n}{r} \cdot r^2]$. But, given this index, there is a single hyperedge which corresponds to this index, and therefore this hyperedge must be present.

The total space required for the linear sketch is just that of $\widetilde{O}(r \, \text{polylog}(m,n))$ linear sketches for multi-graph effective resistance sampling, with each multi-graph defined on $O(\binom{n}{r} \cdot r^2)$ multi-edge slots, a support of $O(mr^2)$ multi-edges, and $O(n \, \text{polylog}(n,m)/r)$ vertices, and each linear sketch using an over-sampling rate of $\widetilde{O}(\text{polylog}(m,n)/\epsilon^2)$. For each multi-graph, the space required is $\widetilde{O}(\frac{n}{r} \cdot r \, \text{polylog}(m)/\epsilon^2)$, and

across all levels of sampling, and iterations of recovery, the number of multi-graphs is $O(r\,\mathrm{polylog}(m, n))$. Thus, the total space required by the linear sketch is $\widetilde{O}(nr\,\mathrm{polylog}(m)/\epsilon^2)$ bits. This yields Theorem 1.2.

2.4 Recursive Recovery Framework

Unfortunately, re-using the same strategy for the fully-dynamic and online settings does not work. This is primarily due to the fact that while there are linear sketches which can accomplish effective-resistance sampling of multi-edges, there are no known fully-dynamic (or online) algorithms which can maintain effective-resistance samples of a graph as it undergoes insertions and deletions. In prior works on designing spectral sparsifiers of graphs in the fully-dynamic setting [ADK+16], this is tackled by maintaining disjoint spanners at different levels of sampling.

More clearly, given a graph G, one can store a sequence $T = T_1 \cup \cdots \cup T_{\mathrm{polylog}(n,m)/\epsilon^2}$ of disjoint spanners, where each T_i is a $\log(n)$-spanner of $G - T_1 - \ldots T_{i-1}$. In [ADK+16], the authors showed via a simple combinatorial argument that any edge of effective resistance $\geq \frac{\epsilon^2}{\mathrm{polylog}(n)}$ *must* be contained in this set T of spanners. Immediately, this implies a simple algorithm for computing spectral sparsifiers whereby one recursively stores these disjoint spanners, and then subsamples the remaining edges at rate $1/2$. The correctness follows from the fact that the remaining edges have small effective resistance, whereby one can argue the concentration from [ST11, SS11], who showed that effective resistance sampling creates spectral sparsifiers. This leads to the following algorithm for creating spectral sparsifiers:

Algorithm 3: GraphSpannerSparsification(G, n, m, ϵ)

---

1 Let $G_1 = G$.
2 for $i \in [\log(m)]$ **do**
3   Let $T^{(i)} = T_1^{(i)} \cup \ldots T_{\mathrm{polylog}(n,m)/\epsilon^2}^{(i)}$ be a sequence of $\mathrm{polylog}(n,m)/\epsilon^2$ disjoint spanners of G_i.
4   Let G_{i+1} be the result of sampling $G_i - T^{(i)}$ at rate $1/2$.
5 end
6 return $T^{(1)} \cup 2 \cdot T^{(2)} \cup \ldots 2^{\log(m)} \cdot T^{(\log(m))}$

In [ADK+16], the authors showed that this framework leads to fully-dynamic algorithms for spectral sparsifiers, as one can design decremental algorithms for maintaining these disjoint collections of spanners (and then bootstrap the decremental solution into a fully-dynamic one). Unfortunately for us however, our analysis still relies on being able to do *effective-resistance sampling*, which is not directly in line with the above procedure.

In this direction, another one of our contributions is to show that the above procedure essentially simulates effective resistance sampling, up to some small degradations in the sampling probability. Specifically, we show the following:

Lemma 2.3. *Let $Q \subseteq G$ denote a set of multi-edges, and let $\epsilon > 0$ be a parameter. Then, if we let S denote the set of multi-edges recovered in spanners as a result of running Algorithm 3 on G, we have that*

$$\Pr[S \cap Q \neq \emptyset] \geq \min\left(2/3, \sum_{e \in Q} R_{\mathrm{eff}, G}(e)/\epsilon^2\right) - \frac{\log(m)}{n^{20}}.$$

In particular, for a hypergraph H, and for each hyperedge $e \in H$, we let the set $Q = K_e$, i.e., the set of multi-edges corresponding to e. Per Theorem 2.1, if the hyperedge e is to be recovered by vertex sampling and effective-resistance sampling, then the constituent multi-edges in K_e must at some have a significant effective resistance $(\geq \frac{\epsilon^2}{\mathrm{polylog}(n,m)})$. Then, by instead running our *recursive* recovery procedure that stores spanners, we will be guaranteed that *some multi-edge* in K_e is recovered, which then suffices for recovering the hyperedge e itself. Due to the degradation in error probability, we run the recursive recovery procedure some $\mathrm{polylog}(n, m)$ times to boost the probability of recovering all necessary hyperedges.

2.5 Fully Dynamic Sparsification and Online Sparsification

As a consequence of the above recursive recovery framework, we are able to construct fully-dynamic algorithms for building hypergraph sparsifiers and online algorithms for building hypergraph spectral sparsifiers. Roughly speaking, the key intuition here is that we have reduced the task of spectrally sparsifying a hypergraph to the task of maintaining a set of disjoint spanners of some corresponding multi-graphs.

Fully-Dynamic In the fully-dynamic setting, our starting point is existing constructions of *decremental* spanners of simple graphs [ADK$^+$16]. By developing some supplementary data structures, we are in fact able to extend these decremental spanners to arbitrary multi-graphs, and thus create a decremental implementation of hypergraph spectral sparsifiers. As in [ADK$^+$16], we are then able to leverage this decremental sparsifier data structure into a *fully-dynamic* sparsifier data structure using a well-known reduction. All that remains then is to calculate the time complexity and space complexity of the fully-dynamic algorithm: it turns out that we store $r \operatorname{polylog}(m,n)/\epsilon^2$ vertex-sampled multi-graphs (each on n/r vertices). For any hyperedge e of arity r, this means that we expect there to be at most $r \operatorname{polylog}(m,n)/\epsilon^2$ corresponding multi-edges across *all* multi-graphs. Upon removing a hyperedge, this leads to an amortized expected time-complexity of $O(r \operatorname{polylog}(m,n)/\epsilon^2)$, as removing a single multi-edge from a single spanner requires time $O(\operatorname{polylog}(n,m))$. The sparsity of the construction will be $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges, as it is exactly in line with Lemma 2.2. This yields a proof of Theorem 1.4.

Online In the online setting, the algorithm is presented with a stream of hyperedge insertions and must decide immediately after seeing each hyperedge, whether or not to keep it, and decide on the corresponding weight to assign it. Fortunately, we have already reduced the task of hypergraph spectral sparsification to storing spanners, and creating spanners in an online manner is exceptionally straightforward: given a spanner constructed so far T, and the new edge f to be inserted, we simply check if f creates any cycles of length $\leq \log(n)$ in T. If not, we include f in the spanner, and otherwise, we do not include f.

From a global perspective, whenever a hyperedge e arrives, we try inserting all of the corresponding multi-edges to e in each of the respective vertex-sampled multi-graphs. If, in any vertex-sampled multi-graph, a multi-edge e is included in one of the spanners, then we know that we must keep the hyperedge e with weight 1. Otherwise, we flip a coin; with probability $1/2$ the hyperedge e is deleted, and with probability $1/2$ we instead try inserting the hyperedge e into the second level of spanners. We continue on in this manner for $\log(m)$ levels of spanners, until the hyperedge e does not "survive" one of the coin flips. This reasoning yields the proof of Theorem 1.5. Observe that the space of the online sparsifier follows from the fact that we store $r \operatorname{polylog}(m,n)/\epsilon^2 \, O(\log(n))$ spanners on graphs with $O(\frac{n}{r})$ vertices. This requires storing at most $O(n \operatorname{polylog}(m)/\epsilon^2)$ multi-edges, which stores at most $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges.

Thus, we have seen how the statement of Theorem 2.1 is powerful, and yields a host of novel sublinear spectral sparsification results for hypergraphs. In the following section, we provide more insight into the proof of Theorem 2.1.

2.6 Overview of Key Technical Theorem (Theorem 2.1)

We now give an overview for our key technical theorem (Theorem 2.1). We first discuss our *Collective Energy Lemma* (Lemma 2.4) in Section 2.6.1, and then discuss our *Vertex Sampling Lemma* (Lemma 2.5) in Section 2.6.2. We refer the reader to our Proof of Theorem 2.1 for how we combine the two lemmas to prove our key technical theorem.

Before diving into the discussion, we first set up some definitions and notation, as well as present the formal statements of the lemmas we are about to discuss. For reasons that will be clear later in our

discussion in Section 2.6.2, we restrict to only vertex potentials with entries in $[0,1]$ when presenting the statements of the two lemmas.

Definition 2.1 (Vertex Potentials and Spanning Hyperedges). A set of vertex potentials is a real-valued vector $x \in [0,1]^n$ supported on the n vertices of a hypergraph H (or its multi-graph G which has the same set of vertices). We say a hyperedge $e \in H$ *spans* x if $\exists (u,v) \in e$ such that $x[u] = 1$ and $x[v] = 0$.

Recall that given vertex potentials $x_1, \ldots, x_k \in \mathbb{R}^n$, their collective energy is

$$\mathcal{E}^H(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \sum_{e \in H} \max_{(u,v) \in e} \left(\sum_{i=1}^k (x_i[u] - x_i[v])^2 \right).$$

We also define for a set of hyperedges F the following minimax resistance optimization:

$$\text{OPT}_F \stackrel{\text{def}}{=} \min_{W \in \mathcal{W}} \max_{e \in F} \max_{(u,v) \in e} R^{G(W)}(u,v),$$

where $\mathcal{W}$ denotes the set of all valid weight assignments of the multi-graph $G = \Phi(H)$ of H. That is, OPT_F characterizes the minimum max sampling rates of the hyperedges in F over all valid weight assignments.

We present the formal statements of our two key lemmas below.

Lemma 2.4 (Collective Energy Lemma). *For any $\theta \in (0,1)$, given a hypergraph H, and a hyperedge set F in H, if $\text{OPT}_F \geq \theta$, then there exists a set of potentials $x_1, \ldots, x_k \in [0,1]^n$ with $k \leq \text{poly}(n)$ such that the following two statements both hold:*

1. *Each x_i is spanned by at least one hyperedge in F (cf. Definition 2.1).*

2. *$\mathcal{E}^H(x_1, \ldots, x_k) \leq k \operatorname{polylog}(n,m)/\theta$.*

Lemma 2.5 (Vertex Sampling Lemma). *Suppose we are given a $\theta \in (0,1)$ satisfying $\theta \geq 1/\text{poly}(n)$, a hypergraph H, and a hyperedge set F in H, along with sets of potentials $x_1, \ldots, x_k \in [0,1]^n$ such that the following both hold:*

1. *Each x_i is spanned by at least one hyperedge in F (cf. Definition 2.1).*

2. *$\mathcal{E}^H(x_1, \ldots, x_k) \leq k \operatorname{polylog}(n,m)/\theta$.*

Then, there (deterministically) exists an $f \in F$ that spans at least one x_i for which we have

$$\Pr\left[f \in \text{VS}(H, \theta^{-1} \operatorname{polylog}(n,m) \right] \geq 1 - 1/\text{poly}(n,m),$$

where the $\text{poly}(n,m)$ in the success probability depends on the $\operatorname{polylog}(n,m)$ in the vertex-sampling scheme.

2.6.1 Important Hyperedges and Collective Energy

Our goal in this section is to connect important hyperedges to a new notion of multi-way energy that we call *collective energy*, defined for a set of potential vectors $x_1, \ldots, x_k$ as

$$\mathcal{E}^H(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \sum_{e \in H} \max_{(a,b) \in e} \left(\sum_{i=1}^k (x_i[a] - x_i[b])^2 \right). \tag{2}$$

We will eventually derive that each important hyperedge is witnessed by a set of k potential vectors with small collective energy (in particular, $\leq k \operatorname{polylog}(n,m)$). We refer the reader to the *Collective Energy Lemma* (Lemma 2.4) for the formal statement.

This can be seen as the spectral analog of the work by Quanrud [Qua24], where they showed how to identify important hyperedges for cut sparsification by looking at the *ratio k-cuts*. As we discuss in Remark 2.1 below, their notion of ratio k-cuts can indeed be seen as special cases of our collective energy. However, we will take an entirely different approach, as the cut counting bound derivation in [Qua24] does not extend to spectral sparsification where we have to deal with real-valued vectors.

Remark 2.1 (Collective Energy and k-Cuts)**.** We highlight an interesting connection between our notion of collective energy and the size of k-cuts in hypergraphs.

Specifically, consider the case when $x_1, \ldots, x_k$ are the indicator vectors of a k-way partition $V_1, \ldots, V_k$ of the vertices. That is, $x_1, \ldots, x_k$ are all 0/1 vectors with disjoint supports that form a partition of V. Then, one can verify that their collective energy $\mathcal{E}^H(x_1, \ldots, x_k)$ equals *exactly* twice the size of the corresponding k-cut in H. Consequently, in this special case, our small collective energy condition means that this k-cut has size at most $k \operatorname{polylog}(n, m)$, which is exactly the criteria for determining important hyperedges for cut sparsification by Quanrud [Qua24] based on their notion of ratio k-cuts.

We start by discussing a conceptual idea for identifying the important hyperedges that leads us to a certain optimization problem of effective resistances.

Minimax Optimization of Resistances. Consider the following idea of identifying the important hyperedges. First, if we were able to choose a weight assignment W such that *simultaneously for all hyperedges e* we have

$$\max_{(u,v) \in e} R^{G(W)}(u, v) < 1/\operatorname{polylog}(n, m),$$

then it would be great since we would have concluded that there are *no* important hyperedges that we need to keep in our sparsifier. On the other hand, if no such weight assignment exists, we would like to find a certain "certificate" that certifies the absence of such weight assignments, which can in turn help us find the "bottleneck" hyperedges preventing us from finding the desired W. Ideally, the certificate should be oblivious to any weight assignment. We will then classify the bottleneck hyperedges as important hyperedges.

Let us now rephrase the idea above from an optimization point of view. Consider the following minimax optimization problem of effective resistances, whose optimal value we denote by $\mathtt{OPT}_H$:

$$\mathtt{OPT}_H \overset{\text{def}}{=} \min_{W \in \mathcal{W}} \max_{e \in H} \max_{(u,v) \in e} R^{G(W)}(u, v), \tag{3}$$

where we write $\mathcal{W}$ to denote the collection of all possible valid weight assignments. If $\mathtt{OPT}_H < 1/\operatorname{polylog}(n, m)$, then we conclude that there are *no* important hyperedges that we are bound to keep in our sparsifier. Otherwise, we want to find some certificate certifying $\mathtt{OPT}_F \geq 1/\operatorname{polylog}(n, m)$, which can in turn help us find the "bottleneck" hyperedges that prevent $\mathtt{OPT}_F$ from ever going below $1/\operatorname{polylog}(n, m)$. Since we want the certificate to be oblivious to any weight assignments, a natural candidate is *vertex potentials*, which are known to characterize effective resistances through their *energies*.

Indeed, we will exploit the following well-known connection between effective resistances and energies of vertex potentials. Specifically, for any $u, v \in V$, we have the following energy minimization view of effective resistance:

$$1/R^{G(W)}(u, v) = \min_{x \in \mathbb{R}^n : x[u]=1, x[v]=0} \mathcal{E}^W(x), \tag{4}$$

where we call x a set of *vertex potentials*, and define $\mathcal{E}^W(x) = \sum_{(a,b) \in G} w(a, b)(x[a] - x[b])^2$ to be the *energy* of x in $G(W)$, with w being the edge weight function specified by W.

Let us now define a few useful notations that will simplify the presentation below. Let $(u_1, v_1), \ldots, (u_t, v_t)$ be the set of *all* vertex pairs such that each $(u_i, v_i) \in e$ for some $e \in H$. Furthermore, let $\mathcal{E}_i^W$ be the

minimum energy for the effective resistance optimization problem (4) for $R^{G(W)}(u_i, v_i)$, i.e.

$$\mathcal{E}_i^W \stackrel{\text{def}}{=} \min_{x \in \mathbb{R}^n : x[u_i]=1, x[v_i]=0} \mathcal{E}^W(x).$$

Then $\mathtt{OPT}_H$ satisfies the following identity:

$$\frac{1}{\mathtt{OPT}_H} = \max_{W \in \mathcal{W}} \min_i \mathcal{E}_i^W. \tag{5}$$

Our goal then is to find a certificate that certifies (5) $\leq \text{polylog}(n, m)$. One attempt to do so is to look at the optimal W^* of the outer maximization problem, as well as the corresponding vertex potentials achieving the energies $\mathcal{E}_i^{W^*}$'s, hoping to use the optimality conditions of W^* to connect the energies of the latter to the value of (5). This however turns out to be difficult as the inner minimization problem is not strictly convex, and in particular may admit multiple minimizers.

To get around this, we will instead look at the *dual* of (5).

Dual of (5). In order to take the dual of (5), a first technicality we have to resolve is to make the inner minimization problem over a convex domain. We do so by perhaps the most straightforward way where we extend the inner domain to all convex combinations of $\mathcal{E}_i^W$'s, which does not change the optimal value:

$$\frac{1}{\mathtt{OPT}_H} = \max_{W \in \mathcal{W}} \min_{\beta \in \Delta^{t-1}} \sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W, \tag{6}$$

where Δ^{t-1} denotes the simplex of dimension $t - 1$, containing all convex combination coefficients of t numbers. This reformulation has the following nice properties:

Claim 2.6 (Proposition 5.2 and Proposition 5.3)**.** The objective function $\sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W$ is convex in β and concave in W.

The claim above allows us to invoke (generalizations of) von Neumann's Minimax Theorem (see e.g., Corollary 37.3.2 of [Roc70]) and deduce strong duality for (6). In particular, we have

$$\frac{1}{\mathtt{OPT}_H} = \min_{\beta \in \Delta^{t-1}} \max_{W \in \mathcal{W}} \sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W. \tag{7}$$

Now, letting β^* to be the minimizer of the outer minimization problem, we can rewrite the dual (7) as

$$\frac{1}{\mathtt{OPT}_H} = \max_{W \in \mathcal{W}} \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^W. \tag{8}$$

That is, we have ended up with a single maximization problem over all valid weight assignments, with the objective function being a fixed convex combination of the optimal energies between the u_i, v_i's. Moreover, the objective function is friendly, as its partial derivatives w.r.t. W have simple forms:

Claim 2.7 (Proposition 5.5)**.** We have for any valid weight assignment $W \in \mathcal{W}$, any $i \in [t]$, and any multi-edge $(a, b) \in G$ in the multi-graph,

$$\frac{\partial \mathcal{E}_i^W}{\partial w(a, b)} = \left(x_i^W[a] - x_i^W[b] \right)^2, \tag{9}$$

where x_i^W is defined to be the energy-minimizing potential vector for u_i, v_i in $G(W)$ (cf. (4)). That is, the derivative of $\mathcal{E}_i^W$ w.r.t. $w(a, b)$ is exactly the energy contribution by the multi-edge (a, b).

Finally, by exploiting the KKT optimality conditions (see e.g., Theorem 28.2 of [Roc70]) on (8), we show that whenever $\mathtt{OPT}_H \geq 1/\operatorname{polylog}(n, m)$, there are potential vectors $\{x_1, \ldots, x_k\} \subseteq \{x_1^{W^*}, \ldots, x_t^{W^*}\}$ satisfying

$$\frac{1}{k} \cdot \mathcal{E}^H(x_1, \ldots, x_k) \leq \operatorname{polylog}(n, m). \tag{10}$$

Here, we define

$$\mathcal{E}^H(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \sum_{e \in H} \max_{(a,b) \in e} \left(\sum_{i=1}^{k} (x_i[a] - x_i[b])^2 \right) \tag{11}$$

to be the *collective energy* of $x_1, \ldots, x_k$ in H, W^* to be the maximizer of (8), and $x_i^{W^*}$ to be the energy-minimizing potential vector for u_i, v_i in $G(W^*)$ (cf. (4)). These potential vectors $x_1, \ldots, x_k$ thus together serve as certificate for $\mathtt{OPT}_H \geq 1/\operatorname{polylog}(n, m)$, and the hyperedges containing (u_i, v_i)'s corresponding to $x_1, \ldots, x_k$ are considered the "bottleneck" hyperedges.

We refer the reader to the *Collective Energy Lemma* (Lemma 2.4) for a formal statement. What we discussed above is of course a vast simplification of the proof of the collective energy lemma. We refer the reader to Section 5 for the full derivation, in particular Proposition 5.7 and Corollary 5.8 which show how KKT conditions allow us to bypass weight assignments completely.

2.6.2 Recovering Important Hyperedges by Vertex Sampling

Our next step is then to show that the certificate $x_1, \ldots, x_k$ could help us recover the important hyperedges. This can be seen as the spectral analog of the work by Khanna, Putterman, and Sudan [KPS24b], where they showed how to recover important hyperedges to cut sparsification by carefully exploiting the (small) ratio k-cuts, which are special cases of our certificate (cf. Remark 2.1). Although our task is more challenging as the certificates are arbitrarily real-valued as opposed to $0/1$-valued, we will nonetheless use a drastically simpler algorithm, namely vertex sampling, that readily reduces the task to effective resistance sampling in *ordinary graphs*, and relegate much of the complication to the analysis. We refer the reader to the *Vertex Sampling Lemma* (Lemma 2.5) for the formal statement.

From now on, we restrict our attention to potential vectors in $[0, 1]^n$. This is without loss of generality since we only deal with energy-minimizing potential vectors x with the constraints that $x[u] = 1$ and $x[v] = 0$ for some vertex pair u, v. As such, any vector can be made in $[0, 1]^n$ by setting all entries > 1 to 1 and all entries < 0 to 0 without increasing its energy or violating the constraints.

Let us now introduce the notion of *potential-spanning hyperedges* that will facilitate our presentation. For a given potential vector $x \in [0, 1]^n$, we say a hyperedge e *spans* x if there exist $u, v \in e$ with $x[u] = 1, x[v] = 0$. For the vectors $x_1, \ldots, x_k$ serving as certificate in (10), each x_i is bound to be spanned by at least one hyperedge, in particular the hyperedge that contains the vertex pair u_j, v_j for which x_i is energy-minimizing.

Extending $\mathtt{OPT}_H$ to Subsets of Hyperedges. Observe that in our definition of $\mathtt{OPT}_H$ in (3), the inner maximization is over multiedges within the *entire* set of hyperedges in H. However, there is nothing special about the entire set of hyperedges - one can in fact change the inner maximization domain to any subset of hyperedges F and obtain a new minimax optimization problem whose optimal value we denote by $\mathtt{OPT}_F$:

$$\mathtt{OPT}_F \stackrel{\text{def}}{=} \min_{W \in \mathcal{W}} \max_{e \in F} \max_{(u,v) \in e} R^{G(W)}(u, v). \tag{12}$$

Similarly, when $\mathtt{OPT}_F \geq 1/\operatorname{polylog}(n, m)$, by taking the dual and invoke the KKT optimality conditions, we get a certificate $x_1, \ldots, x_k$ such that each x_i is spanned by a hyperedge $f \in F$, and moreover

$$\frac{1}{k} \cdot \mathcal{E}^H(x_1, \ldots, x_k) \leq \operatorname{polylog}(n, m). \tag{13}$$

Once again, we refer the reader to Lemma 2.4 for the formal statement and to Section 5 for its proof.

Thought Process for Identifying Important Hyperedges. Now consider the following thought process for identifying important hyperedges:

1. Initially, let F_0 contain all hyperedges in H.

2. While $\mathtt{OPT}_{F_0} \geq 1/\operatorname{polylog}(n,m)$:

    (a) Find certificate $x_1, \ldots, x_k$ with collective energy $\leq k \operatorname{polylog}(n,m)$ such that each x_i is spanned by some $f_i \in F$.

    (b) Remove a (deterministically chosen) hyperedge in $f_1, \ldots, f_k$ from F_0.

3. Identify all hyperedges *not* in F_0 as important.

No matter how we choose the hyperedge at Line 2b to remove, by the condition of the while loop, for the final F_0, there exists a single weight assignment W^* such that *all* multiedges within hyperedges in F_0 have effective resistances at most $1/\operatorname{polylog}(n,m)$ in $G(W^*)$. Thus we do not consider hyperedges in F_0 as important. It then remains to show a good way to choose a hyperedge f_i at Line 2b to remove from F_0 so that (i) we are able to recover f_i algorithmically, and (ii) the number of important hyperedges is bounded by $n \operatorname{polylog}(n,m)$. These will both follow by analyzing our vertex sampling algorithm $\mathtt{VS}(H, \lambda)$ presented in Algorithm 1, which we restate below.

Remark 2.2. Crucially, the thought process itself, along with the certificate $x_1, \ldots, x_k$, are only used in our *analysis* of the vertex sampling algorithm. Algorithmically speaking, $\mathtt{VS}(H)$ is the only action we take to recover important hyperedges.

Algorithm 1: $\mathtt{VS}(H, \lambda)$

Input : A hypergraph H with $\leq m$ hyperedges of arity $[r, 2r]$, and an oversampling rate $\lambda \geq 1$.
Output: A set of hyperedges F.
1 Initialize $F \leftarrow \emptyset$.
2 **for** $r \operatorname{polylog}(n,m)$ rounds **do**
3   Vertex sample H at rate $1/r$ to get H', with $\Phi(H')$ being its multi-graph.
4   Independently sample each multi-edge (u,v) of $\Phi(H')$ with probability $\lambda \cdot R^{\Phi(H')}(u,v)$.
5   Let F' contain all hyperedges of H for which at least one associated multi-edge got sampled.
6   Let $F \leftarrow F \cup F'$ and delete F' from H.
7 **end**
8 **return** F.

It is clear that the total number of hyperedges recovered by this process is at most $n \lambda \operatorname{polylog}(n,m)$ due the resistance sampling. We also show in Section 6 that given any certificate $x_1, \ldots, x_k$ for $\mathtt{OPT}_F \geq 1/\operatorname{polylog}(n,m)$, there *deterministically exists* a hyperedge $f_i \in F$ spanning x_i such that

$$\Pr\left[f_i \in \mathtt{VS}(H)\right] \geq 1 - 1/\operatorname{poly}(n,m). \tag{14}$$

Thus this f_i will be the one we remove from F_0 in the above thought process that identifies important hyperedges. We refer the reader to the *Vertex Sampling Lemma* (Lemma 2.5) for a formal statement.

Our proof of Lemma 2.5 is rather delicate and contains several ideas carefully pieced together, thus we do not intend to cover it fully in this technical overview. Rather, in an effort to provide intuition behind the proof, we will explain how to prove a (distinctly) weaker version of the claim. Then we will discuss the challenges we face in extending this warm-up solution to the full generality, and sketch what more we have to do to overcome them.

Warm-Up: Proving the $0/1$-Case. Specifically, recall that in Remark 2.1, we noted that the ratio k-cuts in [Qua24] are special cases where a collection of k $0/1$-valued vectors with disjoint supports have small collective energy. In light of this connection, for illustration purpose, we consider proving the following $0/1$-version of the vertex sampling lemma as a warm-up.

Lemma 2.8 ($0/1$-Vertex Sampling Lemma). *Suppose we are given $0/1$-valued vectors $x_1, \ldots, x_k \in \{0, 1\}^n$ with disjoint supports, such that each x_i is spanned by at least one hyperedge in H, and*

$$\frac{1}{k} \cdot \mathcal{E}^H(x_1, \ldots, x_k) \le \operatorname{polylog}(n, m). \tag{15}$$

Then there (deterministically) exists an $f \in H$ spanning at least one x_i such that

$$\Pr\left[f \in \mathtt{VS}(H, \operatorname{polylog}(n, m))\right] \ge 1 - 1/\operatorname{poly}(n, m).$$

Note that we are doing effective resistance sampling in each vertex sampled multi-graph in $\mathtt{VS}$. Thus intuitively, in order to recover a hyperedge $f \in H$, it only helps if (i) at least one multi-edge within f survives the vertex sampling, and (ii) the surviving multi-edge(s) within f has high resistance so they are likely to be sampled. If, during one round of vertex sampling, a certain multi-edge $(u, v) \in f$ survives and it spans some x_i, we then hope to resort to the energy minimization view (cf. (4)) of effective resistance to show (u, v) has large resistance.

Specifically, let H' be the vertex-sampled hypergraph that contains $(u, v) \in f$, and let $G' = \Phi(H')$ be its multi-graph. Suppose some x_i satisfies $x_i[u] = 1$ and $x_i[v] = 0$. Recall the energy minimization view of effective resistance:

$$
\begin{aligned}
1/R^{G'}(u, v) &= \min_{x : x[u]=1, x[v]=0} \mathcal{E}^{G'}(x) \\
&= \min_{x : x[u]=1, x[v]=0} \sum_{(a,b) \in G'} (x[a] - x[b])^2.
\end{aligned}
$$

This in particular means that the energy $\mathcal{E}^{G'}(x_i)$ of x_i in G' is an upper bound on $1/R^{G'}(u, v)$. As a result, it suffices to show that $\mathcal{E}^{G'}(x_i)$ is small, which will imply $R^{G'}(u, v)$ is large.

An immediate takeaway of the above discussion is that only multi-edges that span at least one x_i are useful. We call these *spanning multi-edges*. Our first step of the analysis is to characterize these multi-edges by *stars*, whose structure is friendly to analysis under vertex sampling.

Characterizing Spanning Multi-edges by Stars. Consider a potential vector x_i and a hyperedge f spanning it. Then f has at least one vertex with potential 0 and at least one vertex with potential 1 in x_i. If the majority of them have potential 1 in x_i, then for each vertex $u \in f$ with potential 0 in x_i, we create a star with u being the center and all vertices in e with potential 1 in x_i as leaves. Otherwise, conversely, for each vertex $u \in f$ with potential 1 in x_i, we create a star with u being the center and all vertices in e with potential 0 in x_i as leaves. This way each star has $\Omega(r)$ multi-edges each has a potential difference 1 between its endpoints in x_i. Let $\mathcal{S}(x_i)$ be the set of all stars we create from x_i for all hyperedges. Clearly, $\mathcal{S}(x_i)$ captures *all* multi-edge energy contributions of x_i.

For ease of our analysis, we want to restrict to a subset of x_i's each with a roughly similar number of stars. By a standard geometric grouping trick, there exists $\{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\}$ such that $\ell \ge k/\operatorname{polylog}(n, m)$ and each $|\mathcal{S}(y_j)|$ is within $\operatorname{polylog}(n, m)$ of some number h. We now consider the hyperedge $f^* \in H$ that contains the most number of stars across all $y_1, \ldots, y_\ell$. Note that $\ell \ge k/\operatorname{polylog}(n, m)$ implies that $\mathcal{E}(y_1, \ldots, y_\ell) \le \ell \operatorname{polylog}(n, m)$, which in turn yields that at most $\ell \operatorname{polylog}(n, m)$ hyperedges can contain stars, as each of them contributes at least energy 1 to their collective energy $\mathcal{E}^H(y_1, \ldots, y_\ell)$ (cf. (11)). Thus by an averaging argument, there exists one hyperedge f^* that contains at least $h/\operatorname{polylog}(n, m)$ stars across all $y_1, \ldots, y_\ell$. Since $y_1, \ldots, y_\ell$ have disjoint supports, the

stars in f^* must have at least $h/\operatorname{polylog}(n,m)$ different centers. We next show that f^* can be recovered by $\mathtt{VS}(H, \operatorname{polylog}(n,m))$ with high probability.

Within one round of vertex sampling at rate $1/r$, the probability that one (out of $h/\operatorname{polylog}(n,m)$) star center in f^* survives is $\Omega(h/(r\operatorname{polylog}(n,m)))$. Conditioned on a specific center being sampled, one of its star multi-edges gets sampled with constant probability, as there are $\Omega(r)$ of them. On the other hand, conditioned on a specific star center being sampled, the total energy of the y_j corresponding to the star center becomes $\leq 1/r$ times smaller in expectation - this is because multi-edges incident on the center survive with probability $1/r$, whereas other multi-edges survive with probability $1/r^2 \leq 1/r$. Since the $\leq h\operatorname{polylog}(n,m)$ stars in $\mathcal{S}(y_j)$ capture all energy of y_j, we expect the total energy $\mathcal{E}^{G'}(y_j)$ to be $\leq h\operatorname{polylog}(n,m)$. Thus, we expect the survival star multi-edge in f^* to have effective resistance at least $\Omega(1/(h\operatorname{polylog}(n,m)))$. Together, in one round of vertex sampling, f^* gets recovered with probability at least

$$\Omega(h/(r\operatorname{polylog}(n,m))) \cdot \Omega(1/(h\operatorname{polylog}(n,m))) = \Omega(1/(r\operatorname{polylog}(n,m))).$$

As a result, f^* gets recovered with high probability in $r\operatorname{polylog}(n,m)$ rounds of vertex sampling.

Proof Sketch of the General Vertex Sampling Lemma. In the proof of the 0/1-vertex sampling lemma above, we are exploiting the fact all multi-edges with non-zero energy contribution to the energy of the multi-graph can be useful in recovering their parent hyperedges. This is however not true when the vertex potentials have arbitrary real values. In particular, multi-edges of small energy contribution are not helpful in our recovery since the corresponding vertex potential vector does not serve as a certificate of their resistances being high. Yet, if their number is large, their total energy contribution may still be large enough to overshadow the interesting multi-edges that do have large energy contribution.

There is however a remedy for this, namely *rounding* of the vertex potential vector. For instance, if the multi-edges contributing small energy have potentials between $[0, \epsilon]$ on their endpoints for some small $\epsilon > 0$, then we can *round* the potential vector by setting all potentials $\leq \epsilon$ to ϵ. This can only decrease the total energy, while also preserving the multi-edges contributing large energy (e.g., $\Omega(1)$).

Making the rounding work in general turns out to require a more delicate analysis, due the fact that we do not have this clear separation of the vertex potentials in general. Our recipe for the general case is to build a *hierarchy* of star collections. Specifically, we start from the star multi-edges we want to recover, which we call the top-level stars. If there is a way to round vertex potentials to make these multi-edges stand out in terms of their energy contribution, then we are in good shape; otherwise, we show that out of the multi-edges overshadowing the top-level stars, we are able to create another level of stars of geometrically more total energy, thus effectively expanding our star hierarchy by a new level. We show that by choosing the potential rounding scheme carefully, we can make sure (i) all stars we add to our hierarchy have large enough energy contribution to make themselves useful in our recovery algorithm, and (ii) the hierarchy can have at most $\operatorname{polylog}(n,m)$ levels. Our recovery then proceeds in a bottom-up manner. Each level takes $r\operatorname{polylog}(n,m)$ rounds of vertex sampling to recover. So $\operatorname{polylog}(n,m)$ repetitions of these rounds can finally recover the top-level star multi-edges we are interested in.

Once again, this is a vast simplification of the actual proof. We refer the reader to Section 6 for the full proof detail.

2.7 Organization

Section 3 establishes preliminary definitions and concepts that will be used throughout the paper. In Section 4, we present our main vertex sampling algorithm for spectral hypergraph sparsification and its analysis assuming the collective energy lemma and vertex sampling lemma, which are then proved in Section 5 and Section 6 respectively. Section 7 shows how to implement our framework in the linear sketching model. Section 8 develops our recursive recovery framework that forms the basis for both our online and dynamic algorithms. In Section 9, we generalize a result of [ADK$^+$16] to recovering large effective resistance multi-edges in a multi-graph. Section 10 presents our algorithm for online spectral

hypergraph sparsification and establishes its near-optimality via a matching lower bound in Section 10.1. Section 11 extends our framework to the fully dynamic setting.

3 Preliminaries

3.1 Vertex Sampling

In this paper we will make use of the notion of vertex-sampling.

Definition 3.1. For a hypergraph $H = (V, E)$, a **vertex-sampling** at rate p is the result of keeping every vertex (independently) with probability p. We denote the resulting vertex set by V'. The new hypergraph $H' = (V', E')$, where $E' = \{e \cap V' : e \in E\}$.

Note that if $e \cap V' = \emptyset$, then the hyperedge disappears after vertex-sampling. If $e \cap V' \neq \emptyset$, we call $e \cap V'$ the *projected hyperedge* of e.

We also make use of the corresponding multi-graph for any hypergraph.

Definition 3.2. For a hypergraph $H = (V, E)$, the corresponding multi-graph for H, denoted by $\Phi(H)$ has the same vertex set V, and edge set $\bigcup_{e \in E} K_e$, where K_e is the clique on vertices in e.

Note that if a hypergraph has arity bounded by r, then the number of multi-edges in the corresponding hypergraph is at most $|E| \cdot \binom{r}{2}$.

3.2 Linear Sketches

As an application of our results, we create *linear sketches* for hypergraph spectral sparsification. We define linear sketches below:

Definition 3.3. A *linear sketch* of a hypergraph G is given by a (randomized) matrix S of dimension $s \times 2^n$, chosen independently of the hypergraph. We associate the hypergraph G with its indicator vector $\mathbf{1}_G \in \{0,1\}^{2^n}$, and then the *sketch* of the graph is given by $M \cdot \mathbf{1}_G$. The space complexity of the sketch is given by the number of bits required to represent $M \cdot \mathbf{1}_G$.

Note that linear sketches satisfy many convenient properties. For instance, when we wish to add or remove a hyperedge e from G, we can simply add or delete the linear sketch of this hyperedge. That is,

$$M \cdot \mathbf{1}_{G \pm e} = M \cdot \mathbf{1}_G \pm M \cdot \mathbf{1}_e.$$

3.3 Dynamic Algorithms

In this paper we will be studying *dynamic* algorithms for hypergraph problems. In general, for a graph (or hypergraph), a sparsifier may require $\Omega(n)$ (hyper)edges. Thus, it is infeasible to return the entire description of a hypergraph after each update to the underlying graph (i.e., each insertion or deletion of a (hyper)edge). Instead, in a *dynamic* sparsifier, upon any update to the underlying graph, the algorithm responds with the necessary updates to the corresponding sparsifier. We make this more formal below:

Definition 3.4. An algorithm A is a $(1 \pm \epsilon)$ dynamic hypergraph sparsification algorithm if:

1. At every timestep t, A receives a new update to the underlying hypergraph H_{t-1}. The update is either the insertion of a new hyperedge, or deletion of an existing hyperedge. We denote the new resulting hypergraph by H_t.

2. At every timestep t, A outputs a sequence of changes to update the previous sparsifier $\widetilde{H}_{t-1}$ to $\widetilde{H}_t$. This sequence of changes is either re-weighting of hyperedges, the addition of new hyperedges to the sparsifier, or the removal of hyperedges from the sparsifier. With probability $\geq 1 - 1/\text{poly}(n)$, $\widetilde{H}_t$ must be a $(1 \pm \epsilon)$ sparsifier to H_t.

18

There are two key quantities that we will be interested in with respect to a dynamic hypergraph sparsification algorithm. First, we want to understand the *sparsity* of the resulting hypergraph. It is trivial to see that if we do not care about the sparsity, then we can simply keep / delete every hyperedge as it is inserted / deleted. Second, we care about the *update time* after each new insertion / deletion. If we allow the update time to be too large, we can simply re-calculate the entire sparsifier after each new insertion / deletion, therefore getting optimal sparsity, but going against the spirit of a dynamic algorithm. So, we add the following quantifiers to our description of a dynamic algorithm:

Definition 3.5. We say that an algorithm A is a $(1 \pm \epsilon)$ dynamic hypergraph sparsification algorithm of sparsity s and update time f if the algorithm satisfies the conditions of Definition 3.4, while also maintaining $\leq s$ hyperedges in each $\widetilde{H}_t$ and each update requiring amortized, expected time $\leq f$.

We will use the following result from the dynamic *graph* algorithms literature.

Theorem 3.1. *[ADK+16] For every $k \geq 2$ there is a decremental algorithm for maintaining a $2k - 1$ spanner H of expected size $O(k^2 n^{1+1/k} \log(n))$ for an undirected graph G on n vertices with non-negative edge weights that has an expected total update time of $O(k^2 m \log(n))$. Additional H has the following property: Every time an edge is added to H, it stays in H until it is deleted from G.*

Remark 3.1. Of particular note in this theorem is the final sentence. This *monotonicity* (or what we call "laziness" later in the paper) property will be very useful for us as we design dynamic hypergraph sparsification algorithms, as it was in [ADK+16].

4 Spectral Hypergraph Sparsification by Vertex Sampling

We give a meta algorithm for spectral hypergraph sparsification by vertex sampling which is for now stated without regard to any specific algorithmic implementation. We will discuss implementation details in various settings in the following sections.

Notation. We use H to denote a hypergraph of arity $\in [r, 2r]$ and use $G = \Phi(H)$ to denote H's underlying multi-graph where we replace each hyperedge with a clique (of unit edge weights) on those vertices. We write W to denote a weight assignment of the edges in G where the edge weights of each hyperedge's clique sum up to exactly 1, and write $\mathcal{W}$ to denote the collection of all such weight assignments. We write $G(W)$ to denote the multi-graph weighted by W.

For an ordinary multi-graph G, we write $R^G(u, v)$ to denote the effective resistance between u and v. We also write $R^G(e)$ to denote the effective resistance between the endpoints of edge $e \in G$.

4.1 Vertex-Sampling Algorithm

We will use the following result from [JLS23] in a black-box manner.

Theorem 4.1 ([JLS23])**.** *For any weight assignment $W \in \mathcal{W}$ of the underlying multigraph G of a hypergraph H, and $\epsilon \in (0, 1)$, importance sampling each hypergraph $e \in H$ with probability*

$$p_e \geq \max_{(u,v) \in e} R^{G(W)}(u, v) \operatorname{polylog}(n, m)/\epsilon^2$$

gives a $(1 + \epsilon)$-spectral sparsifier of H with high probability in n, m.

Now consider the following vertex sampling algorithm. Note that the output set F contains at most $O(n\lambda \operatorname{polylog}(n, m))$ hyperedges.

Algorithm 1: VS(H, λ)

Input : A hypergraph H with $\leq m$ hyperedges of arity $[r, 2r]$, and an oversampling rate $\lambda \geq 1$.
Output: A set of hyperedges F.

1 Initialize $F \leftarrow \emptyset$.
2 for $r \operatorname{polylog}(n, m)$ rounds **do**
3   Vertex sample H at rate $1/r$ to get H', with $\Phi(H')$ being its multi-graph.
4   Independently sample each multi-edge (u, v) of $\Phi(H')$ with probability $\lambda \cdot R^{\Phi(H')}(u, v)$.
5   Let F' contain all hyperedges of H for which at least one associated multi-edge got sampled.
6   Let $F \leftarrow F \cup F'$ and delete F' from H.
7 end
8 return F.

We will prove the following theorem, which essentially shows that whenever a hyperedge f cannot be recovered by vertex sampling at oversampling rate $\theta^{-1} \operatorname{polylog}(n, m)$, we must be able to sample f with probability $\theta \operatorname{polylog}(n, m)/\epsilon^2$ while preserving the spectrum of the hypergraph. Thus by setting θ to some $O((\epsilon/\log m)^2/\operatorname{polylog}(n, m))$ and repeating $O(\log m)$ times we get a $(1 + \epsilon)$-spectral sparsifier of $O(n \operatorname{polylog}(n, m)/\epsilon^2)$ hyperedges.

Theorem 2.1 (Key Technical Theorem). *For any $\theta \in (0, 1)$, given a hypergraph H and its multi-graph $G = \Phi(H)$, there exists a single weight assignment W^* of G such that for every hyperedge $e \in H$, one of the following statements holds:*

(a) *either $\max_{(u,v) \in e} R^{G(W^*)}(u, v) \leq \theta$, or*

(b) *we have that $\Pr\left[e \in \text{VS}(H, \lambda = \theta^{-1} \operatorname{polylog}(n, m))\right] \geq 1 - 1/\operatorname{poly}(n, m)$, where the $\operatorname{poly}(n, m)$ in the success probability depends on the $\operatorname{polylog}(n, m)$ in the parameter λ of Algorithm 1.*

To prove Theorem 2.1, we need to connect resistance to vertex potentials and their (collective) energies.

Definition 2.1 (Vertex Potentials and Spanning Hyperedges). A set of vertex potentials is a real-valued vector $x \in [0, 1]^n$ supported on the n vertices of a hypergraph H (or its multi-graph G which has the same set of vertices). We say a hyperedge $e \in H$ *spans* x if $\exists (u, v) \in e$ such that $x[u] = 1$ and $x[v] = 0$.

Definition 4.1 (Collective Energies). Given vertex potentials $x_1, \ldots, x_k \in \mathbb{R}^n$, their collective energy is

$$\mathcal{E}^H(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \sum_{e \in H} \left\{ \max_{(u,v) \in e} \left[\sum_{i=1}^k (x_i[u] - x_i[v])^2 \right] \right\}.$$

Definition 4.2 (Minimax Resistance Characterization). For a set of hyperedges F, define

$$\text{OPT}_F \stackrel{\text{def}}{=} \min_{W \in \mathcal{W}} \max_{e \in F} \max_{(u,v) \in e} R^{G(W)}(u, v).$$

That is, OPT_F characterizes the minimum max sampling rates of the hyperedges in F over all valid weight assignments.

The following lemma will be proved in Section 5.

Lemma 2.4 (Collective Energy Lemma). *For any $\theta \in (0, 1)$, given a hypergraph H, and a hyperedge set F in H, if $\text{OPT}_F \geq \theta$, then there exists a set of potentials $x_1, \ldots, x_k \in [0, 1]^n$ with $k \leq \operatorname{poly}(n)$ such that the following two statements both hold:*

1. *Each x_i is spanned by at least one hyperedge in F (cf. Definition 2.1).*

2. *$\mathcal{E}^H(x_1, \ldots, x_k) \leq k \operatorname{polylog}(n, m)/\theta$.*

We prove the following lemma in Section 6.

Lemma 2.5 (Vertex Sampling Lemma). *Suppose we are given a $\theta \in (0,1)$ satisfying $\theta \geq 1/\operatorname{poly}(n)$, a hypergraph H, and a hyperedge set F in H, along with sets of potentials $x_1, \ldots, x_k \in [0,1]^n$ such that the following both hold:*

1. Each x_i is spanned by at least one hyperedge in F (cf. Definition 2.1).

2. $\mathcal{E}^H(x_1, \ldots, x_k) \leq k \operatorname{polylog}(n,m)/\theta$.

Then, there (deterministically) exists an $f \in F$ that spans at least one x_i for which we have

$$\Pr\left[f \in \mathtt{VS}(H, \theta^{-1}\operatorname{polylog}(n,m)\right] \geq 1 - 1/\operatorname{poly}(n,m),$$

where the $\operatorname{poly}(n,m)$ in the success probability depends on the $\operatorname{polylog}(n,m)$ in the vertex-sampling scheme.

Now, we show how to prove our main theorem using this lemma. Roughly, the intuition is that we start by considering $F = E$ (i.e., the entire hyperedge set). Now, either there is a hyperedge which is recovered with high probability from F, or every hyperedge in F can afford to be sampled at probability $1/2$ while still preserving the spectrum of the hypergraph. In the second case, we are already done, and in the first case, we then simply define $F = F - \{e\}$, where e is a hyperedge which is recovered with high probability. Then, we continue on inductively. Observe that the construction of this set F is deterministic, and independent of the randomness used for vertex sampling. Thus, when we actually begin vertex sampling, we must only take a union bound over the $\leq m$ hyperedges that are explicitly recovered.

Proof of Theorem 2.1. For any given $\theta \in (0,1)$, we prove that there exists a hyperedge set F_0 such that $\mathtt{OPT}_{F_0} \leq \theta$ and for each hyperedge $e \notin F_0$ we have

$$\Pr\left[e \in \mathtt{VS}(H, \theta^{-1}\operatorname{polylog}(n,m))\right] \geq 1 - 1/\operatorname{poly}(n,m).$$

This will immediately imply the proposition. To see why such an F_0 exists, we consider the following process for (implicitly) constructing it.

1. Initially, let F_0 contain all hyperedges in H.

2. While $\mathtt{OPT}_{F_0} \geq \theta$:

   (a) By composing Lemma 2.4 and Lemma 2.5, $\exists f \in F_0$ such that

   $$\Pr\left[f \in \mathtt{VS}(H, \theta^{-1}\operatorname{polylog}(n,m))\right] \geq 1 - 1/\operatorname{poly}(n,m).$$

   (b) Remove such an f from F_0.

3. Return F_0.

Then it is clear that the while loop terminates in $\leq m$ iterations, and the returned F_0 has our desired property, completing the proof. $\qquad\square$

In the following sections, we first describe the algorithmic implementations / consequences of vertex-sampling. Then, we provide the proofs of Lemma 2.4, Lemma 2.5.

4.2 Sparsification via Vertex-Sampling

In this section, we show how to use Algorithm 1 to create $(1 \pm \epsilon)$ spectral sparsifiers for hypergraphs. We present the algorithm below:

Algorithm 2: HypergraphSpectralSparsify(H, ϵ, r, m)

Input : A hypergraph H with $\leq m$ hyperedges of arity $[r, 2r]$, and a parameter ϵ.
Output: A reweighted subgraph of H.

1 Let $H_0 = H$.
2 **for** $i = 0, \ldots, \log(m)$ **do**
3   Let $F_i = \mathtt{VS}(H_i, \mathrm{polylog}(n, m)/\epsilon^2)$.
4   Let $H_{i+1} = H_i - F_i$ with its hyperedges sampled at rate $1/2$.
5 **end**
6 **return** $\bigcup_{i=0}^{\log(m)} 2^i \cdot F_i$, i.e. put weight 2^i on hyperedges in F_i.

The key claim is the following:

Claim 4.2. Let $i \in [\log(m)]$, and let H_i be as constructed by an iteration of Algorithm 2. Then, with probability $1 - 1/\mathrm{poly}(m)$, $F_i \cup 2 \cdot H_{i+1}$ is a $(1 \pm \epsilon)$ hypergraph spectral sparsifier for H_i.

Proof. Given the hypergraph H_i, Algorithm 2 recovers a set of hyperedges of H_i via running Algorithm 1. Now, by Theorem 2.1, there exists a single weight assignment W_i^* of the multi-graph of H_i, such that every hyperedge satisfying

$$\max_{(u,v) \in e} R^{\Phi(H_i)(W_i^*)} \geq \frac{\epsilon^2}{\mathrm{polylog}(n, m)}$$

is recovered with probability $\geq 1 - 1/\mathrm{poly}(m)$. Now, every hyperedge satisfying the above condition is essentially sampled with probability 1, while all remaining hyperedges are sampled at rate $1/2$. In particular, for each hyperedge e, this means that the sampling rates we use satisfy

$$p_e \geq \frac{C \log(n) \log(m)}{\epsilon^2} \cdot \max_{(u,v) \in e} R_{\mathrm{eff}}^{\Phi(H_i)(W_i^*)}(u, v).$$

By [JLS23], this then means that the above sampling scheme (whereby we sample edges with probability p_e, and give weight $1/p_e$), yields a $(1 \pm \epsilon)$ spectral-sparsifier with probability $1 - 1/\mathrm{poly}(m)$. $2 \cdot H_{i+1}$ is exactly the hyperedges who survive the sampling at rate $1/2$, and F_i is exactly the hyperedges that we keep with probability 1 (and hence weight 1). This then yields the desired claim. $\square$

Now, we can conclude that Algorithm 2 produces spectral sparsifiers:

Lemma 4.3. *With probability $1 - 1/\mathrm{poly}(m)$, the result of Algorithm 2 called on a hypergraph H with parameter ϵ is a $(1 \pm \epsilon)$-spectral sparsifier of H.*

Proof. Let us suppose by induction that $F_1 \cup 2 \cdot F_2 \cup \cdots \cup 2^i \cdot F_i \cup 2^{i+1} \cdot H_{i+1}$ is a $(1 \pm 2i\epsilon)$ spectral sparsifier of H. Then, in the $i+1$st iteration of sparsification, we replace H_{i+1} with $F_{i+1} \cup 2 \cdot H_{i+2}$. By composition of sparsifiers, if $F_{i+1} \cup 2 \cdot H_{i+2}$ is a $(1 \pm \epsilon)$ spectral sparsifier of H_{i+1}, then $F_1 \cup 2 \cdot F_2 \cup \cdots \cup 2^{i+1} \cdot F_{i+1} \cup 2^{i+2} \cdot H_{i+2}$ is a $(1 \pm 2(i+1)\epsilon)$ spectral-sparsifier of H.

By Claim 4.2, and a union over all $\log(m)$ levels, this yields a $(1 \pm O(\epsilon \log(m)))$ sparsifier with probability $1 - 1/\mathrm{poly}(m)$. Further, observe that after some $i = O(\log(m))$ iterations, every hyperedge in the hypergraph will be removed (i.e., sampled away) with probability $1 - 1/\mathrm{poly}(m)$, and hence H_i will be empty. Thus, all that remains is the F_i's, and their union is the sparsifier.

Finally, observe that we can use an error parameter $\epsilon' = \epsilon/\log(m)$ while only incurring a $\log^2(m)$ blow-up in the vertex-sampling parameter (which we absorb into the $\mathrm{polylog}(m)$). Thus, the result of Algorithm 2 on H will be a $(1 \pm \epsilon)$ spectral sparsifier with probability $1 - 1/\mathrm{poly}(m)$, as we desire. $\square$

4.3 Nearly-Linear Time Implementation

Corollary 4.4 (Nearly-Linear Time Implementation). *Algorithm 2 can be implemented on a static hypergraph H with $\leq m$ hyperedges in time $\widetilde{O}((\sum_{e \in H} |e|) \operatorname{polylog}(m)/\epsilon^2)$ to get a $(1 \pm \epsilon)$-spectral sparsifier of H with $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges, where $|e|$ denotes the arity of hyperedge e. The algorithm succeeds with probability $1 - 1/\operatorname{poly}(m)$.*

Proof. Note that we can always group hyperedges by their arity into $O(\log n)$ geometric groups such that hyperedges in the same group have arity within a factor of 2. Thus, we focus on implementing Algorithm 2 in $\widetilde{O}((mr + n) \operatorname{polylog}(m)/\epsilon^2)$ time, where the input hypergraph only has arity $[r, 2r]$. Our implementation is fairly simple, using only linked list data structures and effective resistance sampling in [SS11].

Note that all Algorithm 2 does is, for $O(\log m)$ adaptive iterations, sampling/deleting hyperedges and repeatedly calling Algorithm 1. Thus it suffices to show that Algorithm 1 can be implemented in $\widetilde{O}((mr + n) \operatorname{polylog}(m)/\epsilon^2)$ time. We describe an implementation of Algorithm 1 as follows.

We first simulate the $N := r \operatorname{polylog}(n, m)$ rounds of vertex-sampling upfront as they are oblivious to other algorithmic actions. Our goal is to obtain a linked list γ_u for each vertex u that denotes the rounds in which u gets sampled. To do so, for each vertex u, we first sample from Binomial distribution a number K_u that corresponds to the number of successes when a coin with success prob. $p = 1/r$ is tossed N times. We then repeatedly sample a random number between 1 and N until we have collected $K(u)$ distinct numbers. This corresponds to the rounds in which u is selected, and we store them as a linked list γ_u. This process takes $n \operatorname{polylog}(n, m)$ time in expectation, since $K_u \leq \operatorname{polylog}(n, m)$ with high probability in m.

We then do another preprocessing step that will be helpful for us to build each vertex-sampled multigraphs. Specifically, we go through each hyperedge one by one. For each hyperedge e, we obtain linked lists $\ell_1^e, \ldots, \ell_N^e$ such that ℓ_i^e contains all e's endpoints that were sampled in the i-th round of vertex sampling. This can be done by going through the endpoints of e one by one and for each endpoint u appending u to the linked lists ℓ_i^e for which $i \in \gamma_u$. This preprocessing step takes running time $\widetilde{O}(mr \log(m))$ with high probability, since with high probability in m the arity of each hyperedge is bounded by $O(\log(m))$ in each round of vertex-sampling.

We then simulate the $r \operatorname{polylog}(n, m)$ adaptive rounds of effective resistance sampling. For each round i, we first create the vertex-sampled multigraph. We do so by going through each remaining hyperedge e that has not been deleted yet, and creating a clique supported on e's vertices sampled in round i which we have stored in linked list ℓ_i^e. We also label each (multi-)edge in the created clique by the identifier of the edge e. Note that the creation of the vertex-sampled multigraph takes running time $\widetilde{O}(m \log^2 m)$ since with high probability in m each hyperedge's arity becomes $O(\log(m))$ after vertex sampling. Then we run the nearly-linear time effective resistance sampling scheme in [SS11] to sample the multi-edges, which has a running time of $\widetilde{O}(m\epsilon^{-2} \operatorname{polylog}(n, m))$ since with high probability in m the vertex-sampled multi-graph has $\widetilde{O}(m \log(m)^2)$ edges, and we over-sample the multi-edges by $\lambda = \operatorname{polylog}(n, m)$. By looking at the labels of the sampled multi-edges, we recover the corresponding hyperedges as well. We then simply delete the recovered hyperedges from the graph and proceed to the next round.

Our total running time adds up to $\widetilde{O}((mr + n) \operatorname{polylog}(m)/\epsilon^2)$ as desired. $\qquad\square$

5 From Minimax Resistance to Collective Energies

In this section, we will prove Lemma 2.4.

We consider a valid weight assignment to be valid only if (i) the weights of the multi-edges within the clique of any single hyperedge sums up to exactly 1, and (ii) every multi-edge in the multi-graph has weight at least[3] $1/(8r^2)$. We denote the collection of valid weight assignments by $\mathcal{W}$.

---

[3]This is to make the multi-graph always connected under any valid weight assignment, assuming the original hypergraph

Consider the following definition:

Definition 5.1 (Minimax Resistance Characterization)**.** For a set of hyperedges F, define

$$\mathtt{OPT}_F \overset{\text{def}}{=} \min_{W \in \mathcal{W}} \max_{e \in F} \max_{(u,v) \in e} R^{G(W)}(u,v).$$

That is, $\mathtt{OPT}_F$ characterizes the minimum max sampling rates of the hyperedges in F over all valid weight assignments.

Also consider the following collective energy lemma, which we seek to prove in this section.

Lemma 5.1 (Collective Energy Lemma)**.** *For any $\theta \in (0,1)$ and hyperedge set F, if $\mathtt{OPT}_F \geq \theta$, then there exists sets of potentials $x_1, \ldots, x_k$ with $k \leq \mathrm{poly}(n)$ such that the following two statements both hold:*

1. *Each x_i is spanned by at least one hyperedge in F.*

2. *$\mathcal{E}^H(x_1, \ldots, x_k) \leq k \, \mathrm{polylog}(n,m)/\theta$.*

Let us fix a hyperedge set F and a $\theta \in (0,1)$ such that $\mathtt{OPT}_F \geq \theta$. Let P_F be the set of vertex pairs (u,v)'s such that u,v both belong to a same hyperedge in F. Note that even if (u,v) appears in more than one hyperedge in F, we only keep one copy of it in P_F. As a result, $|P_F| \leq \binom{n}{2}$. Then we can rewrite the minimax optimization problem as

$$\mathtt{OPT}_F = \min_{W \in \mathcal{W}} \max_{(u,v) \in P_F} R^{G(W)}(u,v). \tag{16}$$

Our first step is to write the effective resistance in the optimization problem as energy of vertex potentials. To this end, let us set up some notation for vertex potentials and their energies.

Let $(u_1, v_1), \ldots, (u_t, v_t)$ where $t = |P_F|$ be the vertex pairs in P_F.

Vertex Potentials and Energies. Given a weight assignment W of the underlying multigraph, we write the energy of a vector $x \in \mathbb{R}^n$ in the multigraph G weighted by W as

$$\mathcal{E}^W(x) \overset{\text{def}}{=} \sum_{(a,b) \in G} w(a,b)(x[a] - x[b])^2. \tag{17}$$

It is known that the reciprocal of effective resistance of an edge (u_i, v_i) is equal to the minimum energy of any vertex potential spanned by (u_i, v_i):

$$\frac{1}{R^{G(W)}(u_i, v_i)} = \min_{x : x[u_i] = 1, x[v_i] = 0} \mathcal{E}^W(x).$$

We write x_i^W to denote the optimal potentials for $R^{G(W)}(u_i, v_i)$, and $\mathcal{E}_i^W \overset{\text{def}}{=} \mathcal{E}^W(x_i^W)$ to denote the corresponding minimum energy. Then $\mathtt{OPT}_F$ can alternatively be written as

$$\frac{1}{\mathtt{OPT}_F} = \max_{W \in \mathcal{W}} \min_{i \in [t]} \min_{x : x[u_i] = 1, x[v_i] = 0} \mathcal{E}^W(x)$$
$$= \max_{W \in \mathcal{W}} \min_{i \in [t]} \mathcal{E}_i^W. \tag{18}$$

is connected to start with.

Dual of $1/\texttt{OPT}_F$. We will now look at the dual of $1/\texttt{OPT}_F$. First we make the minimization problem have a convex feasible solution space by equivalently writing $1/\texttt{OPT}_F$ as

$$\frac{1}{\texttt{OPT}_F} = \max_{W \in \mathcal{W}} \min_{\beta \in \Delta^{t-1}} \sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W. \tag{19}$$

where Δ^{t-1} denotes the simplex of dimension $t-1$ (which contains all vectors denoting a distribution over $[t]$). Note that now both domains $\mathcal{W}$ and Δ^{t-1} are convex and compact. We then prove that the objective function $\left\langle \beta, \vec{\mathcal{E}}_F^W \right\rangle$ is convex in β and concave in W.

Proposition 5.2. *For any fixed $W \in \mathcal{W}$, $\sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W$ is convex in β on Δ^{t-1}.*

Proof. For any fixed $W \in \mathcal{W}$, $\sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W$ is a linear function of β, and thus is also convex. $\square$

Proposition 5.3. *For any fixed $\beta \in \Delta^{t-1}$, $\sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W$ is concave in W on $\mathcal{W}$.*

Proof. It suffices to show that each $\mathcal{E}_i^W$ is concave on $\mathcal{W}$. Note that we have

$$\mathcal{E}_i^W = \min_{x: x[u_i] = 1, x[v_i] = 0} \sum_{(a,b) \in G} w(a,b)(x[a] - x[b])^2.$$

It suffices to show $\mathcal{E}_i^W$ is concave in W. To this end, consider any weight assignments $U, V, W \in \mathcal{W}$ such that $W = \lambda U + (1-\lambda)V$ for $\lambda \in [0,1]$. We have

$$\begin{aligned}
\mathcal{E}_i^W &= \mathcal{E}^W(x_i^W) \\
&= \lambda \mathcal{E}^U(x_i^W) + (1-\lambda)\mathcal{E}^V(x_i^W) && (\mathcal{E}^W(x) \text{ is linear in } W \text{ when } x \text{ fixed}) \\
&\geq \lambda \mathcal{E}^U(x_i^U) + (1-\lambda)\mathcal{E}^V(x_i^V) && (x_i^U, x_i^V \text{ minimize corresponding energies}) \\
&= \lambda \mathcal{E}_i^U + (1-\lambda)\mathcal{E}_i^V.
\end{aligned}$$

This implies that $\mathcal{E}_i^W$ is concave in W, as desired. $\square$

The above propositions allow us to invoke (generalizations of) von Neumann's Minimax Theorem (see Corollary 37.3.2 of [Roc70]) and deduce strong duality for (19):

Proposition 5.4 (Strong Duality for $1/\texttt{OPT}_F$). *We have*

$$\begin{aligned}
\frac{1}{\texttt{OPT}_F} &= \max_{W \in \mathcal{W}} \min_{\beta \in \Delta^{t-1}} \sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W \\
&= \min_{\beta \in \Delta^{t-1}} \max_{W \in \mathcal{W}} \sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W. \tag{20}
\end{aligned}$$

If we define β^* to be the minimizer of (20):

$$\beta^* = \arg\min_{\beta \in \Delta^{t-1}} \max_{W \in \mathcal{W}} \sum_{i \in [t]} \beta[i] \cdot \mathcal{E}_i^W.$$

then we have

$$\frac{1}{\texttt{OPT}_F} = \max_{W \in \mathcal{W}} \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^W. \tag{21}$$

We will then analyze the optimality conditions of (21) and connect it to vertex potentials and their collective energy.

Optimality Conditions. We now analyze the optimality conditions of (21). To this end, we first show that the derivative of the energy with respect to W has the following simple form.

Proposition 5.5 (Energy's Derivative with respect to Weight Assignment)**.** *We have for any valid weight assignment $W \in \mathcal{W}$, any $i \in [t]$, and any multi-edge $(a, b) \in G$ in the multi-graph,*

$$\frac{\partial \mathcal{E}_i^W}{\partial w(a,b)} = \left(x_i^W[a] - x_i^W[b]\right)^2. \tag{22}$$

Proof. We resort to the fact that

$$\mathcal{E}_i^W = \frac{1}{R^{G(W)}(u_i, v_i)} = \frac{1}{\chi_i^T L_W^\dagger \chi_i}.$$

Here, we define χ_i to be the incidence vector of (u_i, v_i), where we have $\chi_i[u_i] = 1, \chi_i[v_i] = -1$ and $\chi_i[u] = 0, \forall u \neq u_i, v_i$. We also define L_W to be the Laplacian matrix of the weighted multi-graph $G(W)$, and $L_W^\dagger$ to denote its pseudo-inverse. Note that since any valid weight assignment ensures that the corresponding weighted multi-graph is connected, the pseudo-inverse here always means taking the inverse in the maximal subspace orthogonal to the all-one vector, with no exceptions.

Using the chain rule, we get

$$\frac{\partial \mathcal{E}_i^W}{\partial w(a,b)} = -\frac{1}{\left(\chi_i^T L_w^\dagger \chi_i\right)^2} \cdot \frac{\partial \chi_i^T L_w^\dagger \chi_i}{\partial w(a,b)}.$$

Computing the derivative via rank-one update formula for the pseudo-inverse [SM50, Mey73], we get

$$\begin{aligned}
\frac{\partial \chi_i^T L_w^\dagger \chi_i}{\partial w(a,b)} &= -\chi_i^T \frac{\partial L_w^\dagger}{\partial w(a,b)} \chi_i \\
&= -\chi_i^T L_W^\dagger \left(\chi_{ab} \chi_{ab}^T\right) L_w^\dagger \chi_i \\
&= -\left(\chi_i^T L_w^\dagger \chi_{ab}\right)^2,
\end{aligned}$$

where χ_{ab} denotes the incidence vector of (a, b). Plugging this back into the chain rule we get

$$\begin{aligned}
\frac{\partial \mathcal{E}_i^W}{\partial w(a,b)} &= \left(\frac{\chi_i^T L_w^\dagger \chi_{ab}}{\chi_i^T L_w^\dagger \chi_i}\right)^2 \\
&= \left(\chi_{ab}^T \frac{L_w^\dagger \chi_i}{\chi_i^T L_w^\dagger \chi_i}\right)^2.
\end{aligned}$$

If we view $z := \frac{L_w^\dagger \chi_i}{\chi_i^T L_w^\dagger \chi_i}$ as a potential vector, then the above equals exactly the squared difference between $z[a]$ and $z[b]$. Note that by definition, $z[u_i] - z[v_i] = 1$. Thus it suffices to show z minimizes the energy, which is indeed true since

$$\mathcal{E}^W(z) = z^T L_W z = \frac{\chi_i^T L_w^\dagger L_w L_w^\dagger \chi_i}{\left(\chi_i^T L_w^\dagger \chi_i\right)^2} = \frac{\chi_i^T L_w^\dagger \chi_i}{\left(\chi_i^T L_w^\dagger \chi_i\right)^2} = \frac{1}{\chi_i^T L_w^\dagger \chi_i} = \mathcal{E}_i^W.$$

This finishes the proof. $\qquad\square$

We will then use KKT conditions (see Theorem 28.2 of [Roc70]) to reason about optimality. We first show that the optimization problem (21) satisfy some regularity conditions.

Proposition 5.6. *The value of (21) is finite and there is feasible point in the relative interior of $\mathcal{W}$ (i.e. Slater's condition is met).*

Proof. The value of (21) is bounded since each hyperedge contributes at most $\binom{n}{2}$ to the total energy, leading to a $m\binom{n}{2}$ upper bound on the value of (21). Moreover, the weight assignment that assigns uniform weight $1/\binom{|e|}{2}$ to the multiedges of a hyperedge e is feasible and in the relative interior of $\mathcal{W}$ (since all weights are strictly bigger than $1/(8r^2)$ by $|e| \in [r, 2r]$). $\qquad\square$

Now we can invoke KKT conditions and get the following. Here we let W^* be the optimal weight assignment minimizing (21):

$$W^* = \arg\max_{W \in \mathcal{W}} \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^W,$$

and w^* be the weight function corresponding to W^*. Also Define $x_i^* \stackrel{\text{def}}{=} x_i^{W^*}$ and $\mathcal{E}_i^* \stackrel{\text{def}}{=} \mathcal{E}_i^{W^*}$. That is, we have

$$\frac{1}{\mathtt{OPT}_F} = \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^*. \tag{23}$$

Proposition 5.7 (Optimality Conditions for (21))**.** *For any hyperedge $f \in H$ and any $(a,b), (c,d) \in f$, we have*

1. *If $w^*(a,b) > 1/(8r^2)$ and $w^*(c,d) > 1/(8r^2)$, then*

$$\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b]\right)^2 = \sum_{i \in [t]} \beta^*[i] \left(x_i^*[c] - x_i^*[d]\right)^2. \tag{24}$$

2. *If $w^*(a,b) > 1/(8r^2)$ and $w^*(c,d) = 1/(8r^2)$, then*

$$\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b]\right)^2 \geq \sum_{i \in [t]} \beta^*[i] \left(x_i^*[c] - x_i^*[d]\right)^2. \tag{25}$$

Proof. We first explicitly write (21) as a constrained maximization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^W \\ \text{subject to} \quad & \\ & \forall (a,b) \in G, \ w(a,b) \geq 1/(8r^2) \\ & \forall f \in H, \ \sum_{(a,b) \in f} w(a,b) = 1. \end{aligned} \tag{26}$$

The corresponding Lagrangian function is

$$L(W; \mu, \lambda) = \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^W + \sum_{(c,d) \in G} \mu_{ab} \cdot \left(w(c,d) - \frac{1}{8r^2}\right) + \sum_{f \in H} \lambda_f \cdot \left(\sum_{(c,d) \in f} w(c,d) - 1\right), \tag{27}$$

where we have that the optimal value to (26) is equal to

$$\max_W \min_{\mu \geq 0, \lambda} L(W; \mu, \lambda).$$

By KKT conditions (see Theorem 28.2 of [Roc70]), there exist $\mu^* \geq 0$ and λ^* such that:

1. (Stationarity) For every hyperedge $f \in H$ and $(a, b) \in f$,

$$\frac{\partial}{\partial w(a,b)} L(W^*; \mu^*, \lambda^*) = 0.$$

2. (Complementary Slackness) For every hyperedge $f \in H$ and $(a, b) \in f$,

$$\mu_{ab}^* > 0 \implies w^*(a, b) = 0.$$

By Proposition 5.5, for a multi-edge (a, b) in a hyperedge $f \in H$, we have

$$\frac{\partial}{\partial w(a,b)} \sum_{i \in [t]} \beta^*[i] \cdot \mathcal{E}_i^W = \sum_{i \in [t]} \beta^*[i] \left(x_i^W[a] - x_i^W[b] \right)^2.$$

As a result, we have

$$\frac{\partial}{\partial w(a,b)} L(W; \mu, \lambda) = \sum_{i \in [t]} \beta^*[i] \left(x_i^W[a] - x_i^W[b] \right)^2 + \mu_{ab} + \lambda_f.$$

By the stationarity condition above, this partial derivative is zero at W^*, μ^*, λ^*, implying that

$$\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 = -\mu_{ab}^* - \lambda_f^*.$$

If $w^*(a, b) > 1/(8r^2)$, then by the complementary slackness condition above, we have $\mu_{ab}^* = 0$, and thus

$$\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 = -\lambda_f^*,$$

This is the same value for all $(a, b) \in f$ with $w^*(a, b) > 1/(8r^2)$, proving the first claim of the proposition. Otherwise ($w^*(a, b) = 1/(8r^2)$), using the fact that $\mu_{ab}^* \geq 0$, we get

$$\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 = -\mu_{ab}^* - \lambda_f^* \leq -\lambda_f^*,$$

proving the second claim of the proposition. □

Getting Rid of Weight Assignments. We then prove the following useful inequality that characterizes OPT_F without resorting to any weight assignment at all.

Corollary 5.8 (of Proposition 5.7). *The vectors* $x_i^*, (u_i, v_i) \in P_F$ *satisfy that*

$$\sum_{f \in H} \max_{(a,b) \in f} \left(\sum_{i=1}^{t} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 \right) \leq \frac{2}{\mathsf{OPT}_F}. \tag{28}$$

Proof. Consider the clique of a hyperedge f and the multiedges (a, b) therein satisfying

$$\sum_{i=1}^{t} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 < \max_{(c,d) \in f} \left(\sum_{i=1}^{t} \beta^*[i] \left(x_i^*[c] - x_i^*[d] \right)^2 \right).$$

By Proposition 5.7, each such multi-edge (a,b) has weight $1/(8r^2)$ with respect to W^*, and thus these multi-edges have total weight at most $1/2$ with respect to W^*. As a result we have

$$
\sum_{f \in H} \max_{(a,b) \in f} \left(\sum_{i=1}^{t} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 \right)
$$

$$
\leq 2 \sum_{f \in H} \sum_{(a,b) \in f} w^*(a,b) \left(\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 \right) \qquad \text{(Proposition 5.7)}
$$

$$
= 2 \sum_{i \in [t]} \beta^*[i] \sum_{f \in H} \sum_{(a,b) \in f} w^*(a,b) \left(x_i^*[a] - x_i^*[b] \right)^2 \qquad \text{(reordering summation)}
$$

$$
= 2 \sum_{i \in [t]} \beta^*[i] \mathcal{E}_i^*
$$

$$
= \frac{2}{\mathtt{OPT}_F}, \tag{29}
$$

as desired. $\qquad\square$

Proof of Collective Energy Lemma. We now prove the collective energy lemma, which we recall below.

Lemma 5.9 (Collective Energy Lemma). *For any $\theta \in (0,1)$ and hyperedge set F, if $\mathtt{OPT}_F \geq \theta$, then there exists sets of potentials $x_1, \ldots, x_k$ with $k \leq \mathrm{poly}(n)$ such that the following two statements both hold:*

1. *Each x_i is spanned by at least one hyperedge in F.*

2. *$\mathcal{E}^H(x_1, \ldots, x_k) \leq k \operatorname{polylog}(n,m)/\theta$.*

Proof. It suffices to exhibit vectors $x_1, \ldots, x_k$ such that

$$
\frac{1}{k} \cdot \mathcal{E}^H(x_1, \ldots, x_k) \leq \frac{12 \log_2 n}{\mathtt{OPT}_F} \tag{30}
$$

and each x_i is spanned by at least one hyperedge in F. To this end, let us consider the set of vectors $x_1^*, \ldots, x_t^*$, which are optimal vertex potentials for each vertex pair $(u_i, v_i) \in P_F$ when the multi-edges are weighted by W^* (optimal weight to (21)). Clearly, each x_i^* is spanned by at least one hyperedge in F which contains u_i, v_i both.

By Corollary 5.8, we know

$$
\sum_{f \in H} \max_{(a,b) \in f} \left(\sum_{i \in [t]} \beta^*[i] \left(x_i^*[a] - x_i^*[b] \right)^2 \right) \leq \frac{2}{\mathtt{OPT}_F}. \tag{31}
$$

The difference between the LHS of the above equation and the definition of collection energy is the parameters $\beta^*[i]$'s. In particular, if β^* corresponded to the uniform distribution (i.e. $\beta^*[i] = 1/t$ for all i), then the LHS would be the collective energy of x_i^*'s divided by t, and we would be done. To deal with the general case, consider doing geometric grouping of indices $i \in [t]$ based on their $\beta^*[i]$ values. Specifically, for $j = 0, 1, \ldots, \log_2 n^3 - 1$, the j-th group R_j contains indices i with

$$
\beta^*[i] \in (2^{-j-1}, 2^{-j}],
$$

whereas the last group $R_{\log_2 n^3}$ contains indices i with

$$
\beta^*[i] \in (-\infty, n^{-3}].
$$

Then we pick a group R_{j^*} whose total β^* values is at least $1/\log_2 r^3$. Note that it must be the case that $j^* < \log_2 n^3$, since there are $\binom{n}{2}$ pairs (u_i, v_i) in total and thus the last group can have total β^* values at most $1/n$. Now restricting the inner summation of (31) to group R_{j^*}, we have

$$
\begin{aligned}
\frac{2}{\mathsf{OPT}_F} &\geq \sum_{f \in H} \max_{(a,b) \in f} \left(\sum_{i \in [t]} \beta^*[i]\, (x_i^*[a] - x_i^*[b])^2 \right) && \text{(by (31))} \\
&\geq \sum_{f \in H} \max_{(a,b) \in f} \left(\sum_{i \in R_{j^*}} \beta^*[i]\, (x_i^*[a] - x_i^*[b])^2 \right) && \text{(restricting inner sum to } R_{j^*}) \\
&\geq \sum_{f \in H} \max_{(a,b) \in f} \left(\sum_{i \in R_{j^*}} 2^{-j^*-1}\, (x_i^*[a] - x_i^*[b])^2 \right) && (\beta^*[i] > 2^{-j^*-1}, \forall i \in R_{i^*}) \\
&\geq \frac{1}{\log_2 n^3} \cdot \frac{1}{2\,|R_{j^*}|} \sum_{f \in G} \max_{(a,b) \in f} \left(\sum_{i \in R_{j^*}} (x_i^*[a] - x_i^*[b])^2 \right) && (R_{j^*} \text{ has total } \beta^* \text{ value} \geq 1/\log_2 n^3) \\
&= \frac{1}{2\log_2 n^3} \cdot \frac{1}{|R_{j^*}|} \cdot \mathcal{E}^H\left(\{x_i^*\}_{i \in R_{j^*}} \right).
\end{aligned}
$$

This implies

$$
\frac{1}{|R_{j^*}|} \cdot \mathcal{E}_G\left(\{x_i^*\}_{i \in R_{j^*}} \right) \leq \frac{12 \log_2 n}{\mathsf{OPT}_F},
$$

recovering (30) as desired. $\qquad\square$

6 Proof of the Vertex Sampling Lemma

In this section, we will provide a complete proof of Lemma 2.5. To start, let us recall some definitions:

Definition 2.1 (Vertex Potentials and Spanning Hyperedges). A set of vertex potentials is a real-valued vector $x \in [0,1]^n$ supported on the n vertices of a hypergraph H (or its multi-graph G which has the same set of vertices). We say a hyperedge $e \in H$ *spans* x if $\exists (u,v) \in e$ such that $x[u] = 1$ and $x[v] = 0$.

Definition 6.1 (Collective Energies). Given vertex potentials $x_1, \ldots, x_k \in \mathbb{R}^n$, their collective energy is

$$
\mathcal{E}^H(x_1, \ldots, x_k) \overset{\text{def}}{=} \sum_{e \in H} \left\{ \max_{(u,v) \in e} \left[\sum_{i=1}^k (x_i[u] - x_i[v])^2 \right] \right\}.
$$

Definition 6.2 (Minimax Resistance Characterization). For a set of hyperedges F, define

$$
\mathsf{OPT}_F \overset{\text{def}}{=} \min_{W \in \mathcal{W}} \max_{e \in F} \max_{(u,v) \in e} R^{G(W)}(u,v).
$$

That is, OPT_F characterizes the minimum max sampling rates of the hyperedges in F over all valid weight assignments.

Now, we let $x_1, \ldots x_k$ be real-valued vectors with entries in $[0,1]$. We will show that there exist $\{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\}$ with $\ell = \Omega(k)$ such that for all $j \in [\ell]$, the energy of y_j in the multi-graph G is bounded by

$$
\mathcal{E}^G(y_j) \leq O(r^2 \theta^{-1} \operatorname{polylog}(n, m)). \tag{32}
$$

Recall that for a vector x, we define its energy in a multi-graph G by

$$\mathcal{E}^G(x) = \sum_{e=(u,v)\in G} (x[u] - x[v])^2.$$

In what follows, whenever we say "energy", we refer to the energy in a multi-graph; we will explicitly say "collective energy" to refer to the collective energy in a hypergraph.

Note that because $y_1, \ldots, y_\ell$ is a subset of $x_1, \ldots, x_k$ and $\ell = \Omega(k)$, together these imply that the collective energy of the former is bounded by

$$\mathcal{E}^H(y_1, \ldots, y_\ell) \leq \ell \operatorname{polylog}(n, m)/\theta, \tag{33}$$

with the $\operatorname{polylog}(n, m)$ factor being potentially larger than the $\operatorname{polylog}(n, m)$ promised by the condition of the lemma.

We will from now on restrict our attention to $y_1, \ldots, y_\ell$, with a goal to prove that $\exists f \in F$ that spans at least one y_j for which we have

$$\Pr\left[f \in \mathtt{VS}(H, \theta^{-1} \operatorname{polylog}(n, m)\right] \geq 1 - 1/\operatorname{poly}(n, m).$$

We start by supplying the proof for (32).

Lemma 6.1. *There exist* $\{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\}$ *with* $\ell = \Omega(k)$ *such that for all* $j \in [\ell]$

$$\mathcal{E}^G(y_j) \leq O(r^2 \theta^{-1} \operatorname{polylog}(n, m)).$$

Proof. Note that by definition, we have

$$\sum_{i=1}^k \mathcal{E}^G(x_i) = \sum_{i=1}^k \sum_{e \in H} \sum_{(u,v) \in e} (x_i[u] - x_i[v])^2$$

$$= \sum_{e \in H} \sum_{(u,v) \in e} \sum_{i=1}^k (x_i[u] - x_i[v])^2 \qquad \text{(reordering summations)}$$

$$\leq \sum_{e \in H} \binom{r}{2} \left\{ \max_{(u,v) \in e} \left[\sum_{i=1}^k (x_i[u] - x_i[v])^2 \right] \right\}$$

$$= \binom{r}{2} \mathcal{E}^H(x_1, \ldots, x_k)$$

$$\leq \binom{r}{2} k \operatorname{polylog}(n, m)/\theta.$$

Then the lemma follows by a standard application of Markov's inequality. $\square$

Characterization of Potentials by Stars. Now, let us consider a single hyperedge e, as well as a potential vector y_j. We will be interested in characterizing the energy contributed in the multi-edges corresponding to e. To this end, we introduce the notion of a *star*. We establish a single vertex $c \in e$ as the *center* of the star, and all other vertices are *leaf* vertices of the star. Clearly, if we choose every single vertex to be a center, then our set of stars will capture all of the energy contributed by the multi-edges of the multi-graph. Our goal will be to carefully construct a set of stars which still captures the majority of the energy, yet also allows us to argue that certain hyperedges will be recovred with high probability.

Hence, we now construct a hierarchy of collections of stars for each potential vector y_j. We want that for each star center, within its incident multi-edges in G, the ones that appear in our star collections contribute overwhelmingly more energy compared to the ones that do not.

We always create stars with the following structure - each star contains multi-edges within a single hyperedge e, with one endpoint as its center, and *all* other endpoints as its leaves. We will also ensure that the center has potential 0, and at least $|e|/2$ leaves have potential $\Omega(1)$. For each star, we call the leaf with the $\lceil|e|/2\rceil$th largest potential the *median* leaf. We always guarantee that each star's median leaf has potential $\Omega(1)$. Additionally, for a *single* potential vector, all stars we create will have the *same* center (though the stars will be for different hyperedges).

More specifically, consider a potential vector $y_j \in [0,1]^n$ that is spanned by some hyperedge $f \in F$. If f has more endpoints with potentials in $[0, 1/2]$ than in $(1/2, 1]$, then we modify y_j by letting $y_j[u] \leftarrow 1 - y_j[u]$ for all vertices u. Note that this does not change the collective energy of $y_1, \ldots, y_\ell$ or the energy $\mathcal{E}^G(y_j)$ at all. Now let $(c^*, d^*) \in f$ be such that $y_j[c^*] = 0$ and $y_j[d^*] = 1$. For a $\eta \in [0,1]$, we write y_j^η to denote the rounded version of y_j where we set all potentials below η to η. We use the following process to construct a hierarchy $\mathcal{S}_{y_j} = \mathcal{S}_{y_j}^1 \cup \mathcal{S}_{y_j}^2 \cup \ldots \mathcal{S}_{y_j}^{O(\log n)}$ of collections of stars. Crucially, we will ensure that all stars we create throughout the hierarchy will have c^* as their center.

1. **Initializing $\mathcal{S}_{y_j}$:** We create the first level of stars, denoted $\mathcal{S}_{y_j}^1$, which contains the single star star with c^* as the center, and f's other endpoints as leaves.

2. **Checking if the top level is good:** Let $\mathcal{S}_{y_j}^p$ be the current top level of stars in our hierarchy (i.e. p is the number of levels which have been created so far). Among all median leaves in $\mathcal{S}_{y_j}^p$, let b be the one with smallest potential, and define

$$\eta_p \stackrel{\text{def}}{=} \left(1 - \frac{1}{\log^2 n}\right) y_j[b].$$

We then check, for the rounded potential vector $y_j^{\eta_p}$, whether the following inequality is true:

$$\sum_{u:(c^*,u)\in G\setminus\mathcal{S}_{y_j}} \left(y_j^{\eta_p}[u] - y_j^{\eta_p}[c^*]\right)^2 \leq 16^{p+1} \cdot r. \tag{34}$$

If (34) is true, then we terminate with our current hierarchy; otherwise we proceed.

3. **Expanding our hierarchy:** In the case that (34) is false, we next expand our hierarchy by adding another level $\mathcal{S}_{y_j}^{p+1}$ of stars that is initialized to be empty.

Define $\eta_p' \stackrel{\text{def}}{=} (1 - 1/\log^2 n)\eta_p$. Let us restrict our attention to the multi-edges $(c^*, u) \in G \setminus \mathcal{S}_{y_j}$ satisfying $y_j[u] > \eta_p$, which contribute non-zero energy to the violation of (34). We say one such multi-edge (c^*, u) is *expandable* if, for the hyperedge e that contains (c^*, u), the majority of its vertices are assigned potentials $\geq \eta_p'$ by y_j, and *non-expandable* otherwise (i.e. majority of e's vertices have potentials $< \eta_p'$ in y_j). If a hyperedge e contains one multi-edge $(c^*, u) \in G \setminus \mathcal{S}_{y_j}$ that is expandable, then we add to the next level of collection $S_{y_j}^{p+1}$ the star with c^* being center and all of e's other vertices as leaves. Otherwise, we ignore e and the non-expandable multi-edges therein, if any.

4. **Repeat:** Go back to Step 2 with $\mathcal{S}_{y_j}^{p+1}$ being the new top level.

At first glance, it may be unclear why we round the values in the potential vector, as this kinda of rounding can only decrease the energy contained in any particular multi-edge. However, the key point is that when it comes to sampling multi-edges and ultimately recovering a hyperedge, we only care about the *relative* amount of energy contained in a specific multi-edge. Creating the star hierarchy as described above ensures that the energy of the stars contributes a large *fraction* of the energy under the specific rounded potential vector that we create. Constructing the hierarchy is essential, as it allows us to define a new set of stars at each level, where each group of stars now uses a *different* potential vector.

We now make some simply observations and remarks about the star hierarchy.

Remark 6.1. In each iteration, every star we add to $\mathcal{S}_{y_j}^{p+1}$ has the same center c^*, and always has median leaf having potential at least η'_p with respect to y_j.

Remark 6.2. We define expandability at *multi-edge* level because this will make it easier to talk about their energy when we analyze vertex sampling later. However, expandability is in fact entirely determined by the *hyperedge* that contains the multi-edge of interest - a hyperedge either only has expandable multi-edges or only has non-expandable multi-edges (or has neither), where "neither" happens when c^* does not belong to the hyperedge, or all its vertices have potentials $\leq \eta_p$ in y_j.

Remark 6.3. Note that for each level p, the expandable and non-expandable multi-edges together *entirely* capture the non-zero energy contribution to the energy of $y_j^{\eta_p}$ from multi-edges incident on c^* but outside of $\mathcal{S}_{y_j}^1 \cup \ldots \mathcal{S}_{y_j}^p$.

We show that the reason we can ignore non-expandable multi-edges is because their number is small.

Claim 6.2. For the first $O(\log n)$ iterations of the above procedure, the number of non-expandable multi-edges is at most $O(r\theta^{-1} \operatorname{polylog}(n, m))$.

Proof. For each non-expandable multi-edge (c, u) belonging to some hyperedge e, the multi-edges connecting u to the $\Omega(r)$ (majority) vertices of e with potentials $< \eta'$ in y_j each has potential difference (in y_j) at least

$$\eta - \eta' = \eta / \log^2 n \geq \left(1 - \frac{1}{\log^2 n}\right)^{O(\log n)} / \log^2 n \geq \Omega(1/\log^2 n),$$

resulting in a total energy contribution of $\Omega(r/\log^4 n)$ to the energy of y_j. Moreover, if we were to sum up this contribution over all non-expandable multi-edges (c, u), we would only count the contribution from each multi-edge at most twice, as it only has two endpoints. This coupled with the fact that $\mathcal{E}^G(y_j) \leq O(r^2\theta^{-1} \operatorname{polylog}(n, m))$ implies that the number of non-expandable multi-edges (c, u) can be at most $O(r\theta^{-1} \operatorname{polylog}(n, m))$. $\square$

We show that whenever we expand, we obtain geometrically more new stars in our collection.

Claim 6.3. For the first $q = O(\log n)$ iterations with $q \geq 2$, the number of stars in $\mathcal{S}_{y_j}^q$ is at least

$$\left|\mathcal{S}_{y_j}^q\right| \geq 16^{q-1} - O(\theta^{-1} \operatorname{polylog}(n, m)).$$

Proof. By the violation of (34), we know that, during the q-th iteration, the incident multi-edges of c not in $\mathcal{S}_{y_j}$ have energy at least $16^q \cdot r$. Thus the expandable multi-edges therein have energy $\geq 16^q \cdot r - O(r\theta^{-1} \operatorname{polylog}(n, m))$ by Claim 6.2. Finally, since each new star in $\mathcal{S}_{y_j}^q$ collects at most $2r$ energy from the expandable multi-edges, we have the desired lower bound on the number of stars. $\square$

The above claim coupled with the energy upper bound of y_j gives a total iteration count of $O(\log n)$.

Claim 6.4. Our process for constructing $\mathcal{S}_{y_j}$ terminates in $O(\log n)$ iterations.

Proof. By Remark 6.1, for $q = O(\log n)$, each median leaf b of our stars in $\mathcal{S}_{y_j}^{q+1}$ has potential in y_j at least

$$\eta'_q \geq \left(1 - \frac{1}{\log^2 n}\right)^{O(\log n)} \geq \Omega(1).$$

Thus by Claim 6.3 and that $\mathcal{E}^G(y_j) \leq O(r^2\theta^{-1} \operatorname{polylog}(n, m))$, the process must terminate in $O(\log n)$ iterations before violating the energy upper bound. $\square$

Claim 6.4 immediately implies the following:

Claim 6.5. Each star in our star collection hierarchy has $\Omega(r)$ multi-edges each with $\Omega(1)$ energy in y_j.

For each potential vector y_j, let $\left|\texttt{levels}\left(\mathcal{S}_{y_j}\right)\right|$ be the number of levels of the star collection hierarchy we created for y_j, and let $\left|\texttt{stars}\left(\mathcal{S}_{y_j}\right)\right|$ be the total number of stars across all levels.

Consider the following process for recovering a hyperedge in F. In the process, for the sake of analysis, we will make changes to the set of potential vectors, our star collection hierarchy, and the hypergraph. However, only Step 4 is what we actually do in our algorithm, and the other steps are for analysis only.

1. Let μ be the average of $\left|\texttt{stars}\left(\mathcal{S}_{y_j}\right)\right|$ over the remaining potential vectors in $y_1, \ldots, y_\ell$.

2. Discard all potential vectors y_j with $\left|\texttt{stars}\left(\mathcal{S}_{y_j}\right)\right| \geq \mu \log^2 n$ and all their stars.

3. For each y_j:

   (a) While $p := \left|\texttt{levels}\left(\mathcal{S}_{y_j}\right)\right| \geq 2$ and the current last level only has $|\mathcal{S}_{y_j}^p| \leq 16^p$ stars:

      i. Delete the current last level of $\mathcal{S}_{y_j}^p$ and all stars therein from our star collection hierarchy.
      ii. Repeat until the while condition is false.

4. $F_1 \leftarrow \texttt{VS}(H, \theta^{-1}\operatorname{polylog}(n, m))$ and delete hyperedges in F_1 from the hypergraph H.

5. Delete the stars in hyperedges in F_1 from our star collection hierarchy.

6. If F_1 contains a hyperedge also in F, then we terminate; otherwise, we go back to Step 1.

We first show that whenever a hyperedge contains sufficiently many stars compared to the current upper bound we have on the total number of stars, then we can recover it by vertex sampling. We will next show that if no such hyperedges exist, then we must be able to get a geometrically smaller upper bound on the total number of stars, and can then make progress by analyzing the next level of stars.

Claim 6.6. Whenever we execute Step 4 in an iteration of the above process, for any hyperedge $e \in H$ containing $\Omega(\mu\theta/\operatorname{polylog}(n, m))$ stars of different centers, we have

$$\Pr\left[e \in \texttt{VS}(H, \theta^{-1}\operatorname{polylog}(n, m))\right] \geq 1 - 1/\operatorname{poly}(n, m).$$

Proof. This proof relies on several ingredients: First, we perform some simplifications to reduce our analysis to the case where only a single center $c_i \in e$ survives the vertex-sampling. Our goal is to show that when this center c_i survives, then with relatively high probability, our effective resistance sampling will recover a multi-edge from e which is incident on c_i. For this, we recall that the effective resistance of a multi-edge is equal to the maximum over all vertex-potentials of the fraction of energy contributed by this multi-edge. Naturally then, there are two quantities we want to bound: 1) an upper bound on the total energy of the entire vertex-sampled multi-graph, and 2) a lower bound on the energy contributed by the multi-edges corresponding to e. For the upper bound, we break the multi-edges into two parts, namely the multi-edges incident on c_i, and those not incident on c_i. For the lower bound, we simply argue that *some* multi-edge in e incident on c_i will survive the vertex sampling. Now, we begin the rigorous proof:

Note that we can assume wlog that the $\mu\theta/\operatorname{polylog}(n, m)$ in the claim is $O(r)$, since e can only contain $\leq 2r$ stars of different centers (it only has $\leq 2r$ vertices by our assumption on the arity of the hyperedge). Let $N = \Omega(\mu\theta/\operatorname{polylog}(n, m))$ be the number of stars of e of different centers. Let $S_1, \ldots, S_N$ be stars of e where each S_i has a unique center $c_i \in e$. If $N > r/4$, we discard $N - r/4$ arbitrary stars and their respective centers from this collection, so that the total number of stars become exactly $r/4$ (doing so only changes the number of stars by a constant factor), and update $N \leftarrow r/4$.

Let $\boldsymbol{E}_1$ denote the event that in one round of vertex sampling, at least one star center gets sampled. Since the stars have different centers, we have

$$
\begin{aligned}
\Pr\left[\boldsymbol{E}_1\right] &= 1 - \left(1 - \frac{1}{r}\right)^N \\
&\geq 1 - \exp\{-N/r\} \\
&\geq \frac{1}{e} \cdot N/r \qquad (\text{as } N \leq r/4) \\
&\geq \Omega(\mu\theta r^{-1}/\operatorname{polylog}(n, m)).
\end{aligned}
$$

For an $i \in [N]$, let $\boldsymbol{E}_2^i$ be the event that in a single round of vertex sampling, the center c_i gets sampled. but all other star centers $c_j, j \neq i$ do not get sampled. Note that events $\boldsymbol{E}_2^i$'s are mutually disjoint for different indices i's, and their probabilities are all equal by symmetry. Moreover, letting N' be the number of star centers sampled in $c_1, \ldots, c_N$, we have

$$
\begin{aligned}
\Pr\left[\bigcup_{i=1}^N \boldsymbol{E}_2^i \,\middle|\, \boldsymbol{E}_1\right] &= \Pr[N' = 1 \mid N' > 0] \\
&= \frac{N(1/r)(1 - 1/r)^{N-1}}{1 - (1 - 1/r)^N} \qquad (\text{binomial distribution}) \\
&\geq \frac{N(1/r)(1 - (N-1)/r)}{1 - (1 - N/r)} \qquad (\text{first Bernoulli ineq. in here}) \\
&\geq \frac{N/(4r)}{N/r} \qquad (N \leq r/4) \\
&= 1/4.
\end{aligned}
$$

As a result,

$$
\Pr\left[\bigcup_{i=1}^N \boldsymbol{E}_2^i\right] = \Pr\left[\bigcup_{i=1}^N \boldsymbol{E}_2^i \,\middle|\, \boldsymbol{E}_1\right] \cdot \Pr\left[\boldsymbol{E}_1\right] = \Omega(\mu\theta r^{-1}/\operatorname{polylog}(n, m)).
$$

We divide the rest of the proof into the $\mu \geq \theta^{-1}$ case and the $\mu < \theta^{-1}$ case.

The $\mu \geq \theta^{-1}$ Case: Consider conditioning on $\boldsymbol{E}_2^i$ for a single arbitrary fixed $i \in [N]$. Let y_j be the potential vector from which the star S_i is created, let q be the level where the star is created, and let η_q be the threshold we used to check if level q is good. Thus $S_i \in \mathcal{S}_{y_j}^q$. We then look at the rounded potential vector $y_j^{\eta_q}$. We consider how much $y_j^{\eta_q}$'s energy becomes after vertex sampling conditioned on $\boldsymbol{E}_2^i$. For multi-edges not incident on c_i, their survival probability is either $1/r^2$ or 0, depending on whether they are incident on another star center, so by (32) their expected total energy after vertex sampling conditioned on $\boldsymbol{E}_2^i$ is

$$
O(1/r^2) \cdot \mathcal{E}^G(y_j^\eta) \leq O(1/r^2) \cdot \mathcal{E}^G(y_j) \leq O(\theta^{-1}\operatorname{polylog}(n, m)).
$$

We then restrict our attention to multi-edges incident on c_i only. Specifically, we consider the following two cases:

1. **q is the last level of $\mathcal{S}_{y_j}$ that hasn't been deleted:** In this case, either q is the last level of $\mathcal{S}_{y_j}$ we have ever created, or there was a level $q + 1$ in $\mathcal{S}_{y_j}$ but it has already been deleted. In the former case, we know the multi-edges incident on c_i but outside of $\mathcal{S}_{y_j}$ contribute energy at most $16^{q+1}r$, which was the reason we stopped expanding $\mathcal{S}_{y_j}$. In the latter case, we know the multi-edges

35

incident on c_i but outside of $\mathcal{S}_{y_j}$ contributing non-zero energy are the non-expandable multi-edges, plus those that were in $\mathcal{S}_{y_j}^{q+1}$, which were deleted because it has $\leq 16^{q+1}$ stars. Therefore, in either case, we know that the total energy of $y_j^{\eta_q}$ contributed by multi-edges incident on c_i is at most

$$O(\mu r \log^2 n) + O(16^{q+1} \cdot r) + O(r\theta^{-1} \operatorname{polylog}(n,m)), \tag{35}$$

with the first term contributed by multi-edges still remaining in $\mathcal{S}_{y_j}$, the second term contributed by the (expandable) multi-edges outside of $\mathcal{S}_{y_j}$, and the third contributed by the non-expandable multi-edges outside of $\mathcal{S}_{y_j}$.

Note that

$$\mu \geq |\mathcal{S}_{y_j}|/\log^2 n \geq |\mathcal{S}_{y_j}^q|/\log^2 n \geq 16^q/\log^2 n,$$

where the first inequality follows from Step 2, and the last inequality follows since level q is the last level that hasn't been deleted. Also, recall that we are in the $\mu \geq \theta^{-1}$ case. So the total energy of $y_j^{\eta_q}$ of multi-edges incident on c_i is less than or equal to $\mu r \operatorname{polylog}(n,m)$.

2. **There is a level $q+1$ in $\mathcal{S}_{y_j}$ that hasn't been deleted:** In this case, all multi-edges incident on c_i contributing nonzero energy to the energy of $y_j^{\eta_q}$ are either in $\mathcal{S}_{y_j}^{q+1}$ or non-expandable. Thus the total energy of multi-edges incident on c_i can be bounded by

$$O(\mu r \log^2 n) + O(r\theta^{-1} \operatorname{polylog}(n,m)). \tag{36}$$

Recall once again that we are in the $\mu \geq \theta^{-1}$ case. So the total energy of $y_j^{\eta_q}$ of multi-edges incident on c_i can similarly bounded by $\mu r \operatorname{polylog}(n,m)$.

Conditioned on $\boldsymbol{E}_2^i$, each multi-edge incident on c_i survives vertex sampling with probability either $1/r$ or 0, depending on whether they are incident on other star centers. Thus the expected total energy of these multi-edges becomes at most $\mu \operatorname{polylog}(n,m)$ after vertex sampling conditioned on $\boldsymbol{E}_2^i$.

Overall, in the case when $\mu \geq \theta^{-1}$, we can bound the expected total energy of y_j^{η} of all multi-edges by $\mu \operatorname{polylog}(n,m)$ after vertex sampling conditioned on $\boldsymbol{E}_2^i$.

On the other hand, we also have that with $\Omega(1)$ probability conditioned on $\boldsymbol{E}_2^i$, we sample one of S_i's star leaves with potential in y_j bigger than that of its median leaf. This is because there are at least $r/2$ such leaves, and at least $r/2 - N \geq r/4$ of them are not star centers, each of which survives vertex sampling with probability $1/r$ even conditioned on $\boldsymbol{E}_2^i$. As a result, the corresponding star multi-edge to the sampled leaf contributes $\Omega(1/\log^4 n)$ energy to $y_j^{\eta_q}$ via the corresponding star multi-edge. Thus with oversampling rate $\lambda = \theta^{-1} \operatorname{polylog}(n,m)$, conditioned on $\boldsymbol{E}_2^i$, we recover hyperedge e with probability $\Omega(\mu^{-1}\theta^{-1}/\operatorname{polylog}(n,m))$. Since this is the case if conditioned on any single $\boldsymbol{E}_2^i$ for all indices $i \in [N]$, the overall probability we recover e in a single round of vertex sampling is at least

$$\Omega(\mu^{-1}\theta^{-1}/\operatorname{polylog}(n,m)) \cdot \Pr\left[\bigcup_{i=1}^N \boldsymbol{E}_2^i\right]$$
$$=\Omega(\mu^{-1}\theta^{-1}/\operatorname{polylog}(n,m)) \cdot \Omega(\mu\theta r^{-1}/\operatorname{polylog}(n,m))$$
$$=\Omega(r^{-1}/\operatorname{polylog}(n,m)).$$

Thus in $r \operatorname{polylog}(n,m)$ rounds, we recover e with probability $1 - 1/\operatorname{poly}(n,m)$, as desired.

The $\mu < \theta^{-1}$ Case: Finally we discuss the case when $\mu < \theta^{-1}$. In this case, the lower bound on the number of stars e has becomes 1 since it is an integral number. With $\mu < \theta^{-1}$, the dominating part in the energy in (35) and (36) is now $r\theta^{-1} \operatorname{polylog}(n,m)$. Thus, the expected energy of $y_j^{\eta_q}$ after vertex sampling is now $O(\theta^{-1} \operatorname{polylog}(n,m))$. Fixing an arbitrary star center (which is potentially the only star center) of e, we know that with high probability, we will sample it in at least one round of vertex sampling,

conditioned on which happening, we also sample a star multi-edge of $\Omega(1/\log^4 n)$ energy in $y_j^{\eta_q}$ with $\Omega(1)$ probability. This coupled with the $\theta^{-1} \operatorname{polylog}(n, m)$ oversampling rate implies e will be recovered with high probability. $\qquad \square$

As promised, we show that we get a geometrically smaller upper bound on the number of stars after each iteration.

Claim 6.7. At the end of each iteration, the total number of remaining stars satisfies with probability $1 - 1/\operatorname{poly}(n, m)$

$$\sum_{\text{remaining } y_j} |\mathcal{S}_{y_j}| \le \mu\ell/\log n.$$

Proof. Define each hyperedge e's contribution to the collective energy as

$$\mathcal{E}_e^H(\{y_j\}) \stackrel{\text{def}}{=} \max_{(u,v)\in e} \left[\sum_{\text{remaining } y_j} (y_j[u] - y_j[v])^2 \right].$$

Note that we always have

$$\sum_{e\in H} \mathcal{E}_e^H(\{y_j\}) = \mathcal{E}^H(\{y_j\}) \le \ell \operatorname{polylog}(n, m)/\theta, \tag{37}$$

where the inequality follows from (33). Let $f(n)$ be the $\operatorname{polylog}(n, m)$ factor in (37) (which is also the factor in (33)) multiplied with another $\log n$. Consider all hyperedges e whose total number of stars across all remaining potential vectors is at least

$$\frac{\mu\theta}{f(n)} \cdot \mathcal{E}_e^H(\{y_j\}). \tag{38}$$

Since $\mathcal{E}_e^H(\{y_j\})$ captures the maximum multi-edge energy contribution, this implies that e much have at least $\Omega(1) \cdot \frac{\mu\theta}{f(n)}$ stars of different centers. By Claim 6.6, all those hyperedges can be recovered by vertex sampling with probability $1 - 1/\operatorname{poly}(n, m)$. Thus, all remaining hyperedges must contain at most (38) stars. Summing this upper bound over all remaining hyperedges and plugging in the upper bound of collective energy in (37), we get an upper bound on the total number of remaining stars as $\mu\ell/\log n$ as desired. $\qquad \square$

We finally finish the proof of Lemma 2.5 with the following claim.

Claim 6.8. With probability $1 - 1/\operatorname{poly}(n, m)$, the above process terminates in $O(\log n/\log\log n)$ iterations, with at least one hyperedge in F recovered.

Proof. Since each time we only discard potential vectors whose number of stars is more than $O(\log^2 n)$ times of the average, we discard $O(1/\log^2 n)$ fraction of the vectors by Markov's inequality. Thus for the first $O(\log n)$ iterations, we have $\Omega(1)$ fraction of the vectors remaining. Thus by Claim 6.7, the total number of stars drops from $\Omega(\mu\ell)$ to $O(\mu\ell/\log n)$, reducing by a $O(1/\log n)$ factor after each one of the first $O(\log n)$ iterations. Thus we only have $O(\log n)$ iterations before μ drops to θ^{-1} or below.

If μ does not drop below θ^{-1}, it must be the case that we have already recovered some hyperedge in F and thus terminated. If μ does drop below θ^{-1}, then by Claim 6.6, we can recover all hyperedges that contain at least one star, among which one must be in F since the first level star collection of every y_j contains a single star in a hyperedge in F. $\qquad \square$

7 Linear Sketching Hypergraph Spectral Sparsifiers

In this section, we will use the following linear sketch:

Theorem 7.1 (Multi-graph Effective Resistance Sampler). *[KLM+14] Given any parameter $\phi \in \mathbb{R}$, there is a linear sketch $\mathcal{S}$ such that for any multi-graph G on ℓ vertices with $\leq u$ potential edge slots, and $\leq \kappa$ edges, for any edge $e = (u, v) \in G$, $\mathcal{S}(G)$ recovers e (independently) with probability at least $\phi \cdot r_{eff,G}(u,v)$. Further, $\mathcal{S}$ requires only $\widetilde{O}(\ell\phi \log(u) \operatorname{polylog}(\kappa))$ bits of space to store.*

The above theorem follows from the linear sketch of [KLM+14] (among others). Although not explicit in their theorem statement, they independently sample each edge with probability at least as large as the leverage score (equal to the effective resistance for unweighted graphs), (see page 11, under "Correctness" section for more details). We discuss this theorem more completely in the appendix, see Appendix B.

Now, we will show that by storing copies of the above sketch for the vertex-sampled multi-graphs, we can create a linear sketch that can be used to recover a hypergraph sparsifier. We present an outline of the algorithm below:

Algorithm 4: LinearSketchSpectralSparsificationConstruction(H, ϵ)

---
1 Let $H_0 = H$.
2 for $i = 0, 1 \ldots 10 \log(m)$ **do**
3 **for** $j \in [r \operatorname{polylog}(n, m)]$ **do**
4   Vertex sample H_1 at rate $1/r$ to get $H'_{(i,j)}$, with $\Phi(H'_{(i,j)})$ being its multi-graph.
5   Store $\mathcal{S}(\Phi(H'_{(i,j)}), R_{i,j})$ with parameter $\phi = C \log(n) \log^2(m)/\epsilon^2$, and new randomness $R_{i,j}$.
6 **end**
7 end

Algorithm 5: LinearSketchSpectralSparsificationRecovery(H, ϵ)

---
1 Let $H_0 = H$.
2 for $i = 0, 1 \ldots 10 \log(m)$ **do**
3 $F_i = \emptyset$.
4 **for** $j \in [r \operatorname{polylog}(n, m)]$ **do**
5   Open $\mathcal{S}(\Phi(H'_{(i,j)} - \cup_{\ell \leq i} F_\ell, R_{i,j}))$ to yield a set of multi-edges $\widetilde{F}_i^{(j)}$.
6   Let $F_i^{(j)}$ be the set of corresponding hyperedges for the multi-edges in $\widetilde{F}_i^{(j)}$.
7 $F_i \leftarrow F_i \cup \widetilde{F}_i^{(j)}$.
8 **end**
9 end
10 return $\cup_i 2^i \cdot F_i$.

Claim 7.2. Algorithm 5 yields a $(1 \pm \epsilon)$ spectral-sparsifier for the hypergraph H with probability $1 - 1/\operatorname{poly}(m)$.

Proof. First, observe that $\mathcal{S}(\Phi(H'_{(i,j)} - \cup_{\ell \leq i} F_\ell, R_{i,j}))$ is efficiently computable given our linear sketch. Indeed, because we store $\mathcal{S}(\Phi(H'_{(i,j)}), R_{i,j})$, and we have already explicitly recovered $\cup_{\ell \leq i} F_\ell$, it follows that $\mathcal{S}(\Phi(H'_{(i,j)} - \cup_{\ell \leq i} F_\ell), R_{i,j}) = \mathcal{S}(\Phi(H'_{(i,j)}), R_{i,j}) - \mathcal{S}(\Phi(\cup_{\ell \leq i} F_\ell), R_{i,j})$. Here, we use our knowledge of the vertex-sampled partitions to compute the corresponding multi-edges that should be removed in correspondence with $\cup_{\ell \leq i} F_\ell$.

Next, observe that because $R_{i,j}$ is a random string independent of all other $R_{a,b}$, $a \neq i, b \neq j$, this means that the set of recovered edges $\cup_{\ell \leq i} F_\ell$ is independent of $R_{i,j}$. In particular, when we calculate $\mathcal{S}(\Phi(H'_{(i,j)} - \cup_{\ell \leq i} F_\ell), R_{i,j}) = \mathcal{S}(\Phi(H'_{(i,j)}), R_{i,j}) - \mathcal{S}(\Phi(\cup_{\ell \leq i} F_\ell), R_{i,j})$, this does not change the failure

probability of the sketch, as we are simply performing an update operation which is independent of the randomness it is using (a similar fact is used in [AGM12]).

We must also show that we are able to recover $F_i^{(j)}$, which is the set of corresponding hyperedges for the multi-edges in $\widetilde{F}_i^{(j)}$. This is essentially a triviality: the linear sketch operates over a universe of size $O(\binom{n}{r} \cdot r^2)$, where there is one index in the universe for each multi-edge slot. We can view this universe as first choosing a hyperedge in $\binom{n}{r}$ ways, and then choosing a constituent multi-edge (in $\binom{r}{2}$ ways). Thus, whenever the linear sketch recovers an index $\in [O(\binom{n}{r} \cdot r^2)]$, it is explicit what the parent hyperedge that should be recovered is.

Finally, to conclude the correctness, we must only observe that Algorithm 5 is exactly implementing Algorithm 2, as the sketch $\mathcal{S}$ performs the multi-edge effective resistance sampling, and the hyperedge recovery is exactly $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Claim 7.3. Algorithm 5 can be implemented with a (randomized) linear sketch of size $\widetilde{O}(nr\,\mathrm{polylog}(m)/\epsilon^2)$.

Proof. The space required by the above sketch is that of storing $r\,\mathrm{polylog}(m,n)$ multi-graph effective resistance samplers, each on a vertex set of size $\widetilde{O}(\frac{n}{r})$ with $\leq O(n^r \cdot r^2)$ multi-edge slots, $\leq O(mr^2)$ edges, and $\phi = \mathrm{polylog}(m,n)/\epsilon^2$. This translates to a space complexity of $\widetilde{O}(\frac{n}{r} \cdot r\,\mathrm{polylog}(m,n)/\epsilon^2)$ bits per multi-graph sketch (using Theorem 7.1), and thus a total space requirement of $\widetilde{O}(nr\,\mathrm{polylog}(m)/\epsilon^2)$ across all the sketches, as we desire. $\qquad\qquad\qquad$ $\square$

With this, we can state our final theorem for this section:

Theorem 7.4. *For hypergraphs of arity $\leq r$, with $\leq m$ hyperedges and an accuracy parameter $\epsilon \in (0,1)$, there is a linear sketch using $\widetilde{O}(nr\,\mathrm{polylog}(m)/\epsilon^2)$ bits that can be used to recover a $(1 \pm \epsilon)$-spectral sparsifier with probability $1 - 1/\mathrm{poly}(n)$.*

Proof. The linear sketch itself is given by Algorithm 4. The accuracy of the recovery procedure is given by Claim 7.2, and the size bound on the linear sketch is given by Claim 7.3. $\qquad\qquad\qquad$ $\square$

We also discuss the application of this linear sketch to the dynamic streaming setting:

Corollary 7.5. *For any $\epsilon \in (0,1)$, there is a randomized dynamic streaming algorithm in $\widetilde{O}(nr\,\mathrm{polylog}(m)/\epsilon^2)$ bits of space that, for any sequence of insertions / deletions of hyperedges in an n-vertex unweighted hypergraph H with at most m edges of arity bounded by r, allows recovery of a $(1 \pm \epsilon)$ spectral-sparsifier of H with probability $1 - 1/\mathrm{poly}(n,m)$ at the end of the stream.*

Proof. We use the linear sketch from Theorem 7.4. Let us denote this sketch by $\mathcal{S}$ and denote its randomness by R. Observe that for a hypergraph H and a hyperedge e, we have

$$\mathcal{S}(H \pm e, R) = \mathcal{S}(H, R) \pm \mathcal{S}(e, R),$$

by the definition of a linear sketch. Thus, given a dynamic stream of a hypergraph, we must only store the randomness R, in addition to the current linear sketch $\mathcal{S}(H, R)$, and add / subtract the sketch of hyperedges that are inserted or deleted.

The only remaining subtlety is to argue that the randomness R need not be too large (namely, that it requires $\widetilde{O}(nr\,\mathrm{polylog}(m)/\epsilon^2)$ bits). For this, we refer the reader to Claim 7.1 in [KPS24b], which provides a general non-uniform derandomization result for hypergraph linear sketches. This then yields the corollary. $\qquad\qquad\qquad$ $\square$

8 Recursive Recovery Framework

8.1 General Framework

From the previous sections, we have established that Algorithm 1 recovers all hyperedges whose sampling rate would be $\geq \frac{\epsilon^2}{\text{polylog}(n,m)}$ in accordance with a weight assignment and Theorem 4.1. In this section, we will show how we can implement Algorithm 1 in an *recursive* manner which will be amenable to both online and fully-dynamic algorithms for sparsification.

Specifically, we will use the following algorithm, which operates on a multi-graph G on n vertices, with $\leq m$ multi-edges, and accuracy parameter ϵ.

Algorithm 6: RecursiveRecovery(G, n, m, ϵ)

---
1 Let $G_1 = G$.
2 **for** $i \in [\log(m)]$ **do**
3   Let F_i be such that $\{e \in G_i : R_{\text{eff},G_i}(e) \geq \frac{(1/2)^2}{1000 \log(n) \log^2(m)}\} \subseteq F_i$.
4   Let G_{i+1} be the result of sampling $G_i - F_i$ at rate $1/2$.
5 **end**
6 **return** $S = F_1 \cup F_2 \cup \ldots F_{\log(m)}$

First, we argue that the above procedure yields $(1 \pm \epsilon)$ spectral sparsifiers:

Claim 8.1. For every $i \in [\log(m)]$, $F_1 \cup \cdots \cup 2^i \cdot F_i \cup 2^{i+1} \cdot G_{i+1}$ is a $(1 \pm O(i \cdot 1/2 \log(m)))$-spectral sparsifier for G, with probability $1 - \log(m)/n^{20}$.

Proof. Let us assume inductively that $F_1 \cup \cdots \cup 2^i \cdot F_i \cup 2^{i+1} \cdot G_{i+1}$ is a $(1 \pm 2(i \cdot 1/2 \log(m)))$-spectral sparsifier for G with probability $1 - i/n^{20}$. Then, in the next iteration, observe that $F_{i+1} \cup 2 \cdot G_{i+2}$ is a $(1 \pm 1/2 \log(m))$-spectral sparsifier for G_{i+1} with probability $1 - 1/n^{20}$, as we are only sampling each edge e independently with marginal probability $\geq \frac{1000 R_e \log^2(m) \log(n)}{\epsilon^2}$, where $\epsilon = 1/2$. Composing sparsifiers, and then taking a union bound over all $\log(m)$ levels then yields our claim. $\square$

Using the above, we can obtain simple bounds on the effective resistance of each edge as we go through levels of sampling.

Claim 8.2. With probability $\geq 1 - \log(m)/n^{20}$, for every $i \in [\log(m)]$, and for every $u, v \in \binom{V}{2}$, it is the case that
$$R_{\text{eff},G_i}(u, v) \geq 2^i \cdot 1/2 \cdot R_{\text{eff},G}(u, v).$$

Proof. First, from the previous claim, we know that with probability $\geq 1 - \log(m)/n^{20}$, for every $i \in [\log(m)]$, it is the case that $F_1 \cup \cdots \cup 2^{i-1} \cdot F_{i-1} \cup 2^i \cdot G_i$ is a $(1 \pm 1/2)$-spectral sparsifier for G. In particular, this means that
$$R_{\text{eff},F_1 \cup \cdots \cup 2^{i-1} \cdot F_{i-1} \cup 2^i \cdot G_i}(u, v) \geq 1/2 \cdot R_{\text{eff},G}(u, v),$$
and therefore,
$$R_{2^i \cdot G_i}(u, v) \geq 1/2 \cdot R_{\text{eff},G}(u, v).$$
Finally, when we remove the weight from G^i, we get that
$$R_{G_i}(u, v) \geq 2^i 1/2 \cdot R_{\text{eff},G}(u, v),$$
as we desire. $\square$

Now, going forward, we will adopt a different perspective for the sub-sampling of edges. Indeed, instead of viewing the sub-sampling operation as taking place *after* certain edges are recovered, let us instead fix a set of "filter" functions beforehand, where for each edge, a filter function keeps the edge with probability $1/2$, and deletes it otherwise. If edges are recovered in a round i, then we simply do not apply the filter functions for future levels.

Claim 8.3. Let $Q \subseteq G$ denote a set of multi-edges. Then, if we let S denote the result of running Algorithm 6 on G, we have that

$$\Pr[S \cap Q \neq \emptyset] \geq \min\left(2/3, \sum_{e \in Q} R_{\text{eff},G}(e)/\epsilon^2\right) - \frac{\log(m)}{n^{20}}.$$

Proof. Let $W = \sum_{e \in Q} R_{\text{eff},G}(e)$. First, observe that there must be some level of sampling $i \in [\log(m)]$ such that edges $e \in Q$ with effective resistance in $[\frac{1}{2^i}, \frac{1}{2^{i-1}}]$ contribute a $\geq 1/\log(m)$ fraction of W. Denoting this set by $Q^{(i)}$, we specifically are saying that

$$\sum_{e \in Q^{(i)}} R_{\text{eff},G}(e) \geq \frac{W}{\log(m)}.$$

We remark that this immediately implies that there are $\geq \frac{W \cdot 2^i}{2 \log(m)}$ edges in $Q^{(i)}$.

Next, let us consider the graph after sampling at rate

$$\frac{1}{2^j} = \frac{100 \log(n) \log(m)}{2^i (1/2)^2}.$$

By the previous claim, with probability $1 - \log(m)/n^{20}$, for every $(u, v) \in Q$, it will be the case that

$$R_{\text{eff},G_i}(u, v) \geq \frac{2^i(1/2)^2}{100 \log(n) \log(m)}(1 - \epsilon) \cdot \frac{1}{2^i} \geq \frac{(1/2)^2}{100 \log(n) \log(m)}.$$

In particular, conditioned on this event happening, as long as *some* edge in $Q^{(i)}$ survives the first j filter functions, it will be the case that an edge in $Q^{(i)}$ is present and has large enough effective resistance, and therefore is recovered (and hence $S \cap Q \neq \emptyset$).

Thus, it remains only to compute the probability that an edge in $Q^{(i)}$ survives the first j filter functions. For this observe that when combined, the first j filter functions are sampling at rate $1/2^j = \frac{100 \log(n) \log(m)}{2^i(1/2)^2}$. Further, the number of edges in $Q^{(i)}$ is $\geq \frac{2^i \cdot W}{2 \log(m)}$. Thus,

$$\Pr[\text{edge in } Q^{(i)} \text{ survives filters}] \geq 1 - (1 - \frac{100 \log(n) \log(m)}{2^i(1/2)^2})^{\frac{2^i \cdot W}{\log(m)2}}$$

$$\geq 1 - e^{-\frac{50 W \log(n)}{(1/2)^2}}.$$

Now, we observe that if $e^{-x} \geq 1/3$, then $e^{-x} \leq 1 - x/2$. This means that either $\Pr[\text{edge in } Q^{(i)} \text{ survives filters}] \geq 2/3$, or

$$\Pr[\text{edge in } Q^{(i)} \text{ survives filters}] \geq \frac{50 W \log(n)}{2(1/2)^2},$$

when $\frac{50 W \log(n)}{(1/2)^2} \leq \ln(3)$. Thus, we can conclude that

$$\Pr[\text{edge in } Q^{(i)} \text{ survives filters}] \geq \min(2/3, \frac{50 W \log(n)}{2(1/2)^2}).$$

In particular, this means that

$$\Pr[\text{edge in } Q^{(i)} \text{ survives filters}] \geq \min(2/3, W).$$

Finally, we observe that the only failure case for our recovering an edge from Q is when *either* the sequence of graphs we construct fails to be a spectral sparsifier, or an edge in $Q^{(i)}$ fails to survive the j filter functions. Thus,

$$\Pr[S \cap Q = \emptyset] \leq \Pr[\text{Sparsifier fail}] + \Pr[\text{Filter fail}] \leq (1 - \min(2/3, W)) + \frac{\log(m)}{n^{20}}.$$

This yields our stated claim. $\qquad\square$

8.2 Re-Implementing Vertex-Sampling Recovery

In this subsection, we will show that a variant of the above recursive recovery procedure can be used to implement the vertex-sampling recovery procedure. Specifically, for a fixed hyperedge f, let us set Q to be the set of multi-edges corresponding to the clique-expansion of f. Observe that if Q is to be recovered in the $r \operatorname{polylog}(m, n) \leq r^2 \leq n^2$ rounds of vertex-sampling with high probability, then in certain rounds of vertex-sampling, the multi-edges in Q must have a lot of effective resistance (i.e., the sum of their effective resistances must be large). This is because multi-edges are sampled directly proportional to their effective resistance.

Finally, once we have established that the multi-edges have a lot of effective resistance, then we can use the previous section to argue that a multi-edge in Q is recovered with high probability. Below, we make this intuition more formal.

First, we introduce some notation: for a hyperedge f, and a round of vertex-sampling ℓ, we let $Q^{(\ell,f)}$ denote the multi-edges correpsonding to f in the ℓth round. We let $W^{(\ell,f)}$ denote the combined effective resistances for multi-edges in $Q^{(\ell,f)}$ with respect to the ℓth vertex-sampled multi-graph.

Claim 8.4. Suppose that a hyperedge f is recovered with probability $\geq 1/2$ in $K \leq n^2$ rounds of vertex sampling (as defined in Algorithm 1). Then,

$$\sum_{\ell \in [K]: W^{(\ell,f)} \geq 1/n^3} W^{(\ell,f)} \geq \frac{\epsilon^2}{\operatorname{polylog}(m, n)}.$$

Proof. First, observe that if a hyperedge is recovered with probability $\geq 1/2$, then this must mean across all rounds $\ell \in [K]$,

$$\sum_{\ell \in [K]} W^{(\ell,f)} \geq \frac{\epsilon^2}{\operatorname{polylog}(m, n)}.$$

Indeed, if we suppose for the sake of contradiction that

$$\sum_{\ell \in [K]} W^{(\ell,f)} < \frac{\epsilon^2}{\operatorname{polylog}(m, n)},$$

then

$$\Pr[\text{sample an edge from } Q] \leq \sum_{\ell \in [K]} \sum_{u,v \in Q^{(\ell,f)}} r_{\text{eff}}(u, v) \cdot \operatorname{polylog}(m, n)/\epsilon^2$$

$$\leq \sum_{\ell \in [K]} W^{(\ell,f)} \cdot \operatorname{polylog}(m, n)/\epsilon^2 < 1/2.$$

Next, observe that

$$\sum_{\ell \in [K]: W^{(\ell,f)} \leq 1/n^3} W^{(\ell,f)} \leq 1/n,$$

as there are at most n^2 rounds total, and under the above assumption, each such round is contributing at most $1/n$ to the total sampling mass.

Finally, putting these together gives that

$$\sum_{\ell \in [K]: W^{(\ell,f)} \geq 1/n^3} W^{(\ell,f)}$$

$$\geq \sum_{\ell \in [K]} W^{(\ell,f)} - \sum_{\ell \in [K]: W^{(\ell,f)} \leq 1/n^3} W^{(\ell,f)} \geq \frac{\epsilon^2}{\operatorname{polylog}(m, n)}.$$

$\square$

With this, we now consider the following algorithm:

Algorithm 7: RepeatedRecursiveRecovery(G, n, m, ϵ)

---

1 Initialize $\widetilde{F} \leftarrow \emptyset$ and $H_1 \leftarrow H$.

2 for $r \operatorname{polylog}(n, m)/\epsilon^2$ *rounds* **do**

3   Vertex sample H_1 at rate $1/r$ to get H', with G being its multi-graph.

4   Let $G_1 = G$.

5   **for** $i \in [\log(m)]$ **do**

6   Let F_i be such that $\{e \in G_i : R_{\mathrm{eff}, G_i}(e) \geq \frac{(1/2)^2}{1000 \log(n) \log^2(m)}\} \subseteq F_i$.

7   Let G_{i+1} be the result of sampling $G_i - F_i$ at rate $1/2$.

8   **end**

9   Let $S = F_1 \cup F_2 \cup \ldots F_{\log(m)}$.

10   Let $\widetilde{S}$ contain all hyperedges of H_1 who have at least one clique multi-edge in S.

11   Let $\widetilde{F} \leftarrow \widetilde{F} \cup \widetilde{S}$ and delete $\widetilde{S}$ from H_1.

12 end

13 return $\widetilde{F}$

Lemma 8.5. *Let f be any hyperedge which is recovered by Algorithm 1 with probability $\geq 1/2$ (for a fixed set of vertex-samplings). Then, f is recovered by Algorithm 7 (with the same fixed set of vertex-samplings) with probability $\geq 1 - 1/\operatorname{poly}(n, m)$.*

Proof. First, by Claim 8.4, we know that this means

$$\sum_{\ell \in [K]: W^{(\ell,f)} \geq 1/n^3} W^{(\ell,f)} \geq \frac{\epsilon^2}{\operatorname{polylog}(m, n)}.$$

Now, by Claim 8.3, in each iteration where $W^{(\ell,f)} \geq 1/n^3$, we know that a multi-edge from $Q^{(\ell,f)}$ is recovered with probability $\geq \min(2/3, W^{(\ell,f)})/2$. Thus, because

$$\sum_{\ell \in [K]: W^{(\ell,f)} \geq 1/n^3} W^{(\ell,f)} \geq \frac{\epsilon^2}{\operatorname{polylog}(m, n)},$$

we also know that across these same rounds of vertex-sampling,

$$\Pr[\text{edge from } Q \text{ is recovered}] \geq \frac{\epsilon^2}{\operatorname{polylog}(m, n)}.$$

Finally, because we repeat this procedure some $\operatorname{polylog}(m, n)/\epsilon^2$ times, we get that any hyperedge which is recovered by Algorithm 1 with probability $\geq 1/2$ is also recovered by Algorithm 7 with probability $1 - 1/\operatorname{poly}(m, n)$. $\qquad\square$

8.3 Sparsification Algorithm

The preceding subsections showed that we can use this recursive recovery procedure to recover the "sensitive" hyperedges (i.e., those with large sampling rate). Next, we show how we can turn this into an actual sparsification algorithm. The key observation is that we must only bootstrap the above algorithm $\log(m)$ times. In each iteration, we are sub-sampling the edges at rate $1/2$ and recovering the important hyperedges in the resulting hypergraph.

Now, we re-analyze Algorithm 2 but with Algorithm 7 as a sub-routine:

Algorithm 8: HypergraphSparsify(H, m, n, ϵ)

---

1 Let $H_i = H$.

2 **for** $i \in [\log(m)]$ **do**

3   Let $F_i = \text{RepeatedIterativeRecovery}(H_i, \lambda = \frac{\text{polylog}(m,n)}{\epsilon^2})$.

4   Let H_{i+1} be the result of keeping each hyperedge in $H_i - F_i$ with probability $1/2$.

5 **end**

6 **return** $\bigcup_{i \in [\log(m)]} 2^i \cdot F_i$.

Claim 8.6. With probability $1 - 1/\text{poly}(n)$, for every $i \in [\log(m)]$, $2^{i+1} \cdot H_{i+1} \bigcup_{j \leq i} 2^i \cdot F_i$ as returned from Algorithm 8 is a $(1 \pm O(i \cdot \epsilon))$ hypergraph spectral sparsifier for H.

Proof. Recall that given the hypergraph H_i, Algorithm 2 recovers a set of hyperedges of H_i via running Algorithm 1. Now, by Theorem 2.1, there exists a single weight assignment W_i^* of the multi-graph of H_i, such that every hyperedge satisfying

$$\max_{(u,v) \in e} R^{\Phi(H_i)(W_i^*)} \geq \frac{\epsilon^2}{\text{polylog}(n,m)}$$

is recovered with probability $\geq 1 - 1/\text{poly}(m)$. Now, every hyperedge satisfying the above condition is essentially sampled with probability 1, while all remaining hyperedges are sampled at rate $1/2$. The only new aspect of Algorithm 8 is that we have used Algorithm 7 as a sub-routine. By Lemma 8.5 and taking a union bound over every hyperedge recovered by Algorithm 2, we know that with probability $1 - 1/\text{poly}(n, m)$, every hyperedge recovered by Algorithm 2 is also recovered in Algorithm 8.

In particular, for each hyperedge e, this means that the sampling rates we use satisfy

$$p_e \geq \frac{C \log(n) \log(m)}{\epsilon^2} \cdot \max_{(u,v) \in e} R_{\text{eff}}^{\Phi(H_i)(W_i^*)}(u, v).$$

By Theorem 4.1[JLS23], this then means that the above sampling scheme (whereby we sample edges with probability p_e, and give weight $1/p_e$), yields a $(1 \pm \epsilon)$ spectral-sparsifier with probability $1 - 1/\text{poly}(m)$. $2 \cdot H_{i+1}$ is exactly the hyperedges who survive the sampling at rate $1/2$, and F_i is exactly the hyperedges that we keep with probability 1 (and hence weight 1). This then yields the desired claim. $\qquad\square$

Lemma 8.7. *With probability $1 - 1/\text{poly}(m)$, the result of Algorithm 8 called on a hypergraph H with parameter ϵ is a $(1 \pm \epsilon)$-spectral sparsifier of H.*

Proof. Let us suppose by induction that $F_1 \cup 2 \cdot F_2 \cup \cdots \cup 2^i \cdot F_i \cup 2^{i+1} \cdot H_{i+1}$ is a $(1 \pm 2i\epsilon)$ spectral sparsifier of H. Then, in the $i + 1$st iteration of sparsification, we replace H_{i+1} with $F_{i+1} \cup 2 \cdot H_{i+2}$. By composition of sparsifiers, if $F_{i+1} \cup 2 \cdot H_{i+2}$ is a $(1 \pm \epsilon)$ spectral sparsifier of H_{i+1}, then $F_1 \cup 2 \cdot F_2 \cup \cdots \cup 2^{i+1} \cdot F_{i+1} \cup 2^{i+2} \cdot H_{i+2}$ is a $(1 \pm 2(i + 1)\epsilon)$ spectral-sparsifier of H.

By Claim 8.6, and a union over all $\log(m)$ levels, this yields a $(1 \pm O(\epsilon \log(m)))$ sparsifier with probability $1 - 1/\text{poly}(m)$. Further, observe that after some $i = O(\log(m))$ iterations, every hyperedge in the hypergraph will be removed (i.e., sampled away) with probability $1 - 1/\text{poly}(m)$, and hence H_i will be empty. Thus, all that remains is the F_i's, and their union is the sparsifier.

Finally, observe that we can use an error parameter $\epsilon' = \epsilon/\log(m)$ while only incurring a $\log^2(m)$ blow-up in the vertex-sampling parameter (which we absorb into the $\text{polylog}(m)$). Thus, the result of Algorithm 8 on H will be a $(1 \pm \epsilon)$ spectral sparsifier with probability $1 - 1/\text{poly}(m)$, as we desire. $\qquad\square$

9   Disjoint Spanners for Recovering High Effective Resistance Multi-Edges

In this section, we will generalize a result of [ADK+16] to recovering large effective resistance multi-edges in a multi-graph. To this end, we introduce some definitions first:

Definition 9.1. A subgraph $T \subseteq G$ is a $\log(n)$-spanner of G if for every pair u, v, $d_T(u, v) \leq \log(n) \cdot d_G(u, v)$. Here, $d_G(u, v)$ is understood to be the distance from u to v in G.

We generalize this to disjoint spanners as follows:

Definition 9.2. We say that $T_1, \ldots T_\ell$ form a disjoint collection of $\log(n)$-spanners if for every $i \in [\ell]$, $T_i \subseteq G - T_1 - \ldots T_{i-1}$, and T_i is a $\log(n)$-spanner of $G - T_1 - \ldots T_{i-1}$.

Our key claim (as in [ADK+16]) is the following:

Claim 9.1. Let $T_1, \ldots T_\ell$ be a disjoint collection of $\log(n)$-spanners of a multi-graph G. Then, every multi-edge $e = (u, v)$ for which $R_{\text{eff},G}(u, v) \geq \frac{\log(n)}{\ell}$ is contained in $T_1 \cup \cdots \cup T_\ell$.

Proof. Our proof proceeds in the same manner as [ADK+16]. Indeed, consider any multi-edge (u, v) which is not contained in $T_1 \cup \cdots \cup T_\ell$. We will show every such multi-edge has effective resistance $\leq \frac{\log(n)}{\ell}$. Indeed, if a multi-edge (u, v) is not recovered in any of $T_1, \ldots T_\ell$, then it must be the case that (u, v) is present in each graph $G - T_1 - \ldots T_{i-1}$ when the ith spanner T_i is being created. This means that there must be a path from u to v of length $\leq \log(n)$ in T_i. Because the T_i's are disjoint, this means that across $T_1, \ldots T_\ell$ there are ℓ disjoint paths of length $\leq \log(n)$ from u to v. By the resistor network interpretation of graphs, this means that the effective resistance between u and v in G must be $\leq \frac{\log(n)}{\ell}$, yielding our claim. $\qquad\square$

10 Online Spectral Hypergraph Sparsification

In this section, we will show the following theorem:

Theorem 10.1. *Let H be a hypergraph on n vertices, $\leq m$ hyperedges, and arity $[r, 2r]$, whose hyperedges are revealed one by one. Then, there is an* online *algorithm for computing a $(1 \pm \epsilon)$ spectral sparsifier for H which retains only $\widetilde{O}(n \log(m)/\epsilon^2)$ hyperedges.*

Recall that in the online sparsification setting, the hyperedges are presented in a stream $e_1, \ldots e_m$ (where we know the value m before-hand). After each hyperedge e_i is revealed, the algorithm must immediately decide whether or not to include e_i in the sparsifier, and if included, must also decide on its weight. In particular, the sparsifier for e_i must be decided without having seen the future hyperedges $e_{i+1}, \ldots e_m$.

To this end, we use the following claim:

Claim 10.2. Given a sequence of multi-edge insertions, there is an online algorithm which creates a $\log(n)$-spanner by keeping only $O(n)$ multi-edges.

Proof. The algorithm is very simple: the algorithm stores the spanner it has built so far, which is denoted by T. For the ith multi-edge (u, v) that is revealed, the algorithm checks if (u, v) are already connected by a path of length $\leq \log(n)$. If so, the algorithm does not include (u, v), and otherwise the algorithm does include (u, v). Clearly, this builds a $\log(n)$-spanner, as for any edge (u, v) in G, (u, v) are connected by a path of length $\leq \log(n)$ in T.

Next, we show that T will never have a cycle of length $\leq \log(n)$. Indeed, in order for this to be the case, there must have been some time step i where adding the edge (u, v) created a cycle of length $\leq \log(n)$. This means that (u, v) were connected by a path of length $\leq \log(n) - 1$, and the cycle was then completed by adding (u, v). But, if there is already such a path of length $\leq \log(n) - 1$, then the edge (u, v) would not be included. Thus, there must not be any cycles of length $\leq \log(n)$ in T.

Finally, it remains to bound the number of edges in T. This follows because any graph on n vertices with no cycles of length $\leq k$ must have $n^{1+O(1/k)}$ edges (see for instance, Proposition 2.3 in [ABS+20]). Plugging in $k = \log(n)$ then yields that T must have $O(n)$ edges. $\qquad\square$

Now, by a simple bootstrapping of the above claim, we can also get the following:

Claim 10.3. Given a sequence of multi-edge insertions, there is an online algorithm which creates a disjoint collection of $\ell \log(n)$-spanners, keeping only $O(\ell n)$ multi-edges.

Proof. The algorithm again is very simple. The algorithm maintains the spanners $T_1, \ldots T_\ell$ with the invariant that T_i is a $\log(n)$-spanner of $G - T_1 - \cdots - T_{i-1}$.

Upon receiving a multi-edge (u, v), the algorithm checks if (u, v) can be added to T_1 without creating a cycle of length $\log(n)$. If so, the (u, v) is added to T_1, otherwise, the algorithm tries adding (u, v) to T_2, and so on.

Each T_i has $O(n)$ edges by the same logic as the previous claim. Likewise, each T_i is a $\log(n)$-spanner of $G - T_1 - \cdots - T_{i-1}$, as we can simply view T_i as the spanner that is build by applying the previous algorithm Claim 10.2 to the sequence of online edges with $T_1, \ldots, T_{i-1}$ removed. Thus, we get a set of ℓ disjoint $\log(n)$-spanners, each with $O(n)$ edges, yielding our claim. $\square$

Now we are ready to present our online algorithm for spectral hypergraph sparsification. We adopt the mold of Algorithm 7 and Algorithm 8, but replace the black-box step of storing large-effective resistance multi-edges with storing a set of disjoint spanners:

Algorithm 9: RepeatedRecursiveRecoverySpanner(G, n, m, ϵ)

---

1 Initialize $\widetilde{F} \leftarrow \emptyset$ and $H_1 \leftarrow H$.
2 for $r \operatorname{polylog}(n, m)/\epsilon^2$ *rounds* **do**
3   Vertex sample H_1 at rate $1/r$ to get H', with G being its multi-graph.
4   Let $G_1 = G$.
5   **for** $i \in [\log(m)]$ **do**
6   Store a set $T_1, \ldots T_{4000 \log^2(n) \log^2(m)}$ of $4000 \log^2(n) \log^2(m)$ disjoint spanners of G_i.
7   Let F_i be the set of multi-edges $\{f : \exists j \in [4000 \log^2(n) \log^2(m)], f \in T_j\}$.
8   Let G_{i+1} be the result of sampling $G_i - F_i$ at rate $1/2$.
9   **end**
10   Let $S = F_1 \cup F_2 \cup \ldots F_{\log(m)}$.
11   Let $\widetilde{S}$ contain all hyperedges of H_1 who have at least one clique multi-edge in S.
12   Let $\widetilde{F} \leftarrow \widetilde{F} \cup \widetilde{S}$ and delete $\widetilde{S}$ from H_1.
13 end
14 return $\widetilde{F}$

We can then adopt the following sparsification algorithm:

Algorithm 10: HypergraphSparsifySpanner(H, m, n, ϵ)

---

1 Let $H_i = H$.
2 for $i \in [\log(m)]$ **do**
3   Let $F_i = \text{RepeatedIterativeRecoverySpanner}(H_i, \lambda = \frac{\operatorname{polylog}(m, n)}{\epsilon^2})$.
4   Let H_{i+1} be the result of keeping each hyperedge in $H_i - F_i$ with probability $1/2$.
5 end
6 return $\bigcup_{i \in [\log(m)]} 2^i \cdot F_i$.

Claim 10.4. Algorithm 10 produces a $(1 \pm \epsilon)$ spectral hypergraph sparsifier of H with probability $1 - 1/\operatorname{poly}(m)$.

Proof. First, by Claim 9.1, it follows that in the multi-graph G_i, any multi-edge of effective resistance $\geq$ $\frac{\log(n)}{4000 \log^2(n) \log^2(m)} = \frac{(1/2)^2}{1000 \log(n) \log^2(m)}$ is recovered in F_i, as the disjoint $\log(n)$-spanners $T_1, \ldots T_{4000 \log^2(n) \log^2(m)}$ will include these multi-edges. Thus, Algorithm 9 is a concrete instantiation of an algorithm which implements Algorithm 7.

Then, we must only observe that Algorithm 10 is exactly implementing Algorithm 8. The correctness directly follows. □

Claim 10.5. Algorithm 10 can be implemented as an online algorithm, storing only $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges.

Proof. First we show the bound on the number of hyperedges that are stored. The algorithm stores $O(r \operatorname{polylog}(m, n)/\epsilon^2) \log(n)$-spanners, each on a vertex set of size $\widetilde{O}(n/r)$. Each spanner therefore stores $\widetilde{O}(n/r)$ multi-edges, and in total, the number of multi-edges stored is $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$. Because each hyperedge that is recovered requires one of its corresponding multi-edges to be recovered, this means that the total number of recovered hyperedges is $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$.

Next, we show that the algorithm can be implemented in an online manner. This follows from the fact that we can construct the disjoint collections of spanners in an online manner (as per Claim 10.3) *and* perform the sampling in an online manner. Before starting, the algorithm initializes all spanners to be empty, and uses public randomness to construct the vertex-sampled subsets of the original hypergraph. We will show that Algorithm 9 can be implemented in an online manner. From there, observe that implementing Algorithm 10 online is straightforward, as we simply check if a hyperedge e is stored by Algorithm 9. If so, the hyperedge is kept with weight 1. Otherwise, we flip a coin: with probability $1/2$ we delete the hyperedge, and with probability $1/2$ we check if e is stored by Algorithm 9 at the second level of sampling. If it is recovered here, then we assign it weight 2, and so on.

Finally, to implement Algorithm 9 as an online algorithm, let us use $T_1^{(i,j)}, \ldots T_{4000 \log^2(n) \log^2(m)}^{(i,j)}$ to denote the disjoint spanners for the jth round of vertex sampling (out of $r \operatorname{polylog}(n, m)/\epsilon^2$ rounds), and $i \in [\log(m)]$ round of sub-sampling multi-edges. Whenever a new hyperedge e arrives, we first calculate the vertex-sampled version of the hyperedge $e^{(1)}$, and attempt to insert the multi-edges corresponding to $K_{e^{(1)}}$ into $T_1^{(1,1)}, \ldots T_{4000 \log^2(n) \log^2(m)}^{(1,1)}$. This can be done online as per Claim 10.3. Next, whichever multi-edges are not recovered, we sample at rate $1/2$ before attempting to insert into $T_1^{(2,1)}, \ldots T_{4000 \log^2(n) \log^2(m)}^{(2,1)}$ (via Claim 10.3) and so for $i \in [\log(m)]$. After the $\log(m)$ levels, if any multi-edge from $K_{e^{(j)}}$ has been recovered, then we terminate. Otherwise, we continue on to the second vertex-sampled multi-graph and repeat trying to insert $K_{e^{(2)}}$ into $T_1^{(1,2)}, \ldots T_{4000 \log^2(n) \log^2(m)}^{(1,2)}$, repeating the same logic. If no spanner ever recovers a multi-edge from $K_{e^{(j)}}$ across all rounds, then we do not store the hyperedge. Otherwise, we report it as being stored. This yields the online implementation. □

Finally, we can conclude:

Theorem 10.6. *There is an online hypergraph spectral sparsification algorithm, which for hypergraphs H on n vertices with $\leq m$ hyperedges, and arity $\leq r$ undergoing a sequence of insertions maintains a $(1 \pm \epsilon)$-spectral-sparsifier (and thus cut-sparsifier too) with probability $1 - 1/\operatorname{poly}(m)$. The space complexity of the algorithm is $\widetilde{O}(nr \operatorname{polylog}(m)/\epsilon^2)$ bits and the sparsity of the sparsifier is $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges.*

Proof. The algorithm and size of the sparsifier follows from Claim 10.5 (the bit complexity follows because there are $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ stored multi-edges, but each multi-edge has a label of size $O(r)$ bits to its parent hyperedge). The correctness of the sparsifier follows from Claim 10.4. □

10.1 Online Sparsification Lower Bound

In this section, we detail a lower bound for online hypergraph (cut) sparsification in the model where we can re-weigh hyperedges that have already been included in the sparsifier. Note that this is a strictly easier setting than the setting discussed above, where required spectral sparsification (stronger than cut), and did not allow the sparsifier to alter the weights of the hyperedges it kept.

Further, we restrict the model to only allow insertions of one copy of each hyperedge, and that inserted hyperedges must be unweighted.

10.1.1 Lower Bound Set-up

Now, we explain the set-up for the lower bound. We will let k denote the number of rounds of insertions that we will perform. We partition the vertex set into a left side $L = [n]$ and a right side $R = [n]$. For each round $j \in [k]$, we randomly partition the left side L into two pieces L_j^1, L_j^2, and the right side into two pieces R_j^1, R_j^2. Then, for each left vertex $u \in L_j^1$, we add n^{2j} random hyperedges of arity $n/4$ from to R_j^1, and likewise for every $u \in L_j^2$ we add n^{2j} random hyperedges from to R_j^2.

10.2 Lower Bound Analysis

First, we establish that if the sparsifier is small, then necessarily all of its edges must be crossing between the partition established in R with high probability.

Claim 10.7. Suppose that after round $j - 1$, the sparsifier contains $\leq n^2$ hyperedges. Then, with probability $1 - n^2/2^{\Omega(n)}$, each hyperedge in the sparsifier crosses from R_j^1 to R_j^2.

Proof. Indeed, consider any hyperedge e in the sparsifier. Note that the partition R_j^1, R_j^2 is chosen independently of the sparsifier. So, the probability that e is completely contained in exactly one of R_j^1, R_j^2 is bounded by $(1/2)^{n/4-1} = 1/2^{\Omega(n)}$. Taking the union bound over all edges e in the sparsifier yields our claim. $\square$

Next, we show that with high probability, there is no way to re-weigh existing hyperedges to match certain cut sizes without necessarily violating other cuts.

Claim 10.8. Suppose that after round $j - 1$, the sparsifier contains $\leq n^2$ hyperedges. Then, in round j, the sparsifier must include $\Omega(n)$ new hyperedges in the sparsifier.

Proof. Indeed, consider a single vertex $u \in L$, and let us assume WLOG that $u \in L_j^1$. Now, in the jth round, we insert n^j hyperedges that are incident upon u. For the sake of contradiction, let us assume that the sparsifier does not include any of these hyperedges, and instead re-weighs existing hyperedges. Because the degree of u must be preserved in order to be a $(1 \pm \epsilon)$ cut-sparsifier, this means that the hyperedges incident to u in the sparsifier must be re-weighed by an average factor of $\Omega(n)$ to compensate. However, now let us consider cut defined by $S = L_j^1 \cup R_j^1$. In the actual hypergraph (not the sparsifier), the size of this cut is bounded by $\sum_{i \leq j-1} n^{2i+1} = O(n^{2j-1})$. However, in the sparsifier, with probability $1 - 1/2^{\Omega(n)}$ (from the previous claim), every hyperedge incident to u is now crossing from R_j^1 to R_j^2. The sum of the weights of these hyperedges is $\Omega(n^{2j})$, and therefore the cut is exceeded by a factor of n, and therefore not a sparsifier.

This means that the sparsifier *must* select a new hyperedge among those incident on u to be included. Hence, n new hyperedges must be included, as there is one for each left-vertex. $\square$

Claim 10.9. After $k \leq n/10$ rounds, the size of the sparsifier is $\Omega(nk)$.

Proof. For each of the k rounds, the sparsifier adds $\Omega(n)$ hyperedges. $\square$

Theorem 10.10. *Online hypergraph cut-sparsifiers with re-weighing allowed require $\Omega(n \log(m)/\log(n))$ hyperedges .*

Proof. Given an allotment of m hyperedges, this allows us to perform the above procedure for $\Omega(\log_n(m)) = \Omega(\log(m)/\log(n))$ rounds. By the previous claim, this leads to a total of $\Omega(n \log(m)/\log(n))$ hyperedges that must be in the sparsifier. $\square$

11 Fully Dynamic Algorithms for Spectral Hypergraph Sparsification

In this section, we explain how to implement a fully-dynamic algorithm for spectral hypergraph sparsification. We will use Algorithm 10 as a starting point, and show how we can leverage decremental algorithms for maintain spanners in multi-graphs to create a fully-dynamic sparsification algorithm. To this end, we start by explaining the well-known reduction from fully-dynamic sparsification algorithms to decremental sparsification algorithms.

11.1 Reducing Fully-Dynamic to Decremental

We use the well-known reduction for turning decremental sparsifiers into fully dynamic sparsifiers (see for instance, [ADK$^+$16]). Note that the previous reductions were specifically done for graphs, hence our motivation here is to re-produce it for hypergraphs. We follow the exposition and statement of [ADK$^+$16] closely.

We show the following lemma:

Lemma 11.1. *[ADK$^+$16] Given a decremental algorithm for maintaining a $(1 \pm \epsilon)$ spectral-sparsifier of size $S(m, n, \epsilon)$ for an undirected hypergraph with total update time $m \cdot T(m, n, \epsilon)$, there is a fully dynamic algorithm for maintaining a $(1 \pm \epsilon)$-cut sparsifier of size $O(S(m, n, \epsilon) \log(m))$ with amortized update time $O(T(m, n, \epsilon) \log(m))$.*

As in [ADK$^+$16], this lemma relies on a notion of decomposability.

Claim 11.2. Let $H = (V, E)$ be a hypergraph, and let $E_1, \ldots E_k$ be a partition of the edge-set into k pieces. Let $\widetilde{H}_i$ be a $(1 \pm \epsilon)$-spectral sparsifier for $H_i = (V, E_i)$. Then, $\widetilde{H} = \cup_{i=1}^k \widetilde{H}_i$ is a $(1 \pm \epsilon)$-spectral sparsifier for H.

Proof. Consider any vector $x \in \mathbb{R}^V$. We have that $Q_H(S) = \sum_{i=1}^k Q_{H_i}(S)$. Thus, because $Q_{\widetilde{H}_i}(S) \in (1 \pm \epsilon) Q_{H_i}(S)$, we see that

$$Q_{\widetilde{H}}(S) = \sum_{i=1}^k Q_{\widetilde{H}_i}(S) \in (1 \pm \epsilon) Q_H(S),$$

as we desire. $\qquad \square$

Now, we are ready to prove the overall lemma.

Proof of Lemma 11.1. Set $k = \lceil \log(m) \rceil$. For each $1 \le i \le k$, we maintain a set $E_i \subseteq E$ of edges, and an instance A_i of the decremental algorithm running on the hypergraph $H_i = (V, E_i)$. We also keep a binary counter C that counts the number of insertions $\mod m$ with the least significant bit in C being the right most one.

A deletion of an edge e is carried out by simply deleting e from the set E_i it is contained in, and propagating the deletion to instance A_i of the decremental algorithm. An insertion of an edge e is more involved: we let j be the highest (i.e., left-most) bit that gets flipped in the counter when increasing the number of insertions. Thus, in the updated counter, the jth bit is 1, and all lower bits are 0. We first add the edge e, as well as all edges in $\cup_{i=1}^{j-1} E_i$ to E_j. Then, we set $E_i = \emptyset$ for all $1 \le i \le j - 1$. Finally, we re-initialize the instance A_j on the hypergraph $H_j = (V, E_j)$.

Now, we can bound the total update time for each instance A_i of the decremental algorithm. First, we observe that the ith bit of the binary counter is reset after every 2^i edge insertions. A simple induction then shows that at any time $|E_i| \le 2^i$, for all $1 \le j \le k$. Now, consider an arbitrary sequence of updates of length ℓ. The instance A_i is re-initialized after every 2^i insertions. It therefore is re-initialized at most $\ell / 2^i$ times. For every re-initialization, we pay a total update time of $|E_i| \cdot T(|E_i|, n, \epsilon) \le 2^i \cdot T(m, n, \epsilon)$. For the entire sequence of ℓ updates, the total time spent for instance A_i is therefore $(\ell / 2^i) \cdot 2^i \cdot T(m, n\epsilon) =$

$\ell \cdot T(m, n, \epsilon)$. Thus, we spend total time $O(\ell \cdot T(m, n, \epsilon) \log(m))$ for the entire algorithm, which gives an amortized update time of $O(T(m, n, \epsilon) \log(m))$. $\qquad\square$

11.2 Building Decremental Disjoint Multi-graph Spanners

Recall that the key building block in Algorithm 10 and Algorithm 9 is the construction of disjoint spanners in a multi-graph. The work of [ADK+16] already studied constructing these in *ordinary* (simple) graphs. In this section, we will show how we can extend their result to work for multi-graphs. Roughly speaking, the difficulty comes from the fact that whenever we recover a multi-edge, we must also recover the corresponding *label* (i.e., parent hyperedge) corresponding to it.

We start with the spanners of [ADK+16]:

Theorem 11.3. *[ADK+16] For every $k \geq 2$ there is a decremental algorithm for maintaining a $\log(n)$ spanner H of expected size $O(n \log^3(n))$ for an undirected graph G on n vertices with non-negative edge weights that has an expected total update time of $O(m \log^3(n))$. Additionally, H has the following property: Every time an edge is added to H, it stays in H until it is deleted from G.*

Recall that we refer to this additional property as the "lazy" property. Next, recall our notion of disjoint spanners:

Definition 11.1. Let $G = (V, E)$ be a multi-graph. We say that a set of spanners $T_1, \ldots T_\ell$ is an ℓ-bundle of $\log(n)$-spanners for G if T_i is a $\log(n)$-spanner of $G - T_1 - T_2 - \cdots - T_{i-1}$.

Notationally, for $i \in [\ell]$, we will let G_i denote the (simple) graph for which T_i is a spanner. That is, G_i is the result of taking the multi-graph $G - T_1 - \cdots - T_{i-1}$, and then putting only a single copy of any present multi-edge. Note that the same edge (u, v) may be used in different spanners if there are multiple copies of the edge present. Now, as a warm-up, we show that there is a decremental data structure for maintaining ℓ-bundles of spanners in multi-graphs.

Claim 11.4. There is a decremental algorithm for maintaining an ℓ-bundle of $\log(n)$-spanners of an undirected, unweighted multi-graph G with amortized update time $O(\ell \log(\ell) \log^3(n))$. Moreover, the algorithm satisfies the "lazy" property, in the following sense: Every time an edge e is inserted, the only potential change is the insertion of e to the bundle of spanners. Whenever an edge e is *deleted*, the only potential changes are the removal of e from the bundle of spanners, and the addition of a single new edge to the bundle of spanners.

Proof. In addition to maintaining the ℓ disjoint spanners (using Theorem 11.3), we maintain a simple table data structure. This table data structure maps each possible edge $e = (u, v)$ to a tuple with three elements. The first element in the tuple is the number of copies of the multi-edge, denoted w_e, the second is the number of spanners which contain the multi-edge t_e, and the final element is a self-balancing binary search tree (e.g., AVL tree) containing the indices $\subseteq [\ell]$ of all the spanners which contain this multi-edge, as well as a pointer to the final index $f_e \in [\ell]$ in the AVL tree. Note that since different multi-edges connecting the same pair of vertices are interchangeable for the purpose of building spanners, we only keep a counter for the number of spanners using e rather than track which copies they are using.

Now, let us consider what happens when an edge e is inserted. First, if $w_e > t_e$, then the multi-edge is already present in every copy of the graph (i.e., every G_i), and all we do is increment w_e. So, we consider the case when $w_e = t_e$, that is, every copy of the multi-edge is already being used in spanners. We first increment w_e by 1. Then, we access f_e. Starting $i = f_e + 1$, we insert the edge e into G_i. If T_i updates to include the edge (u, v), then increment t_e, and set $f_e = i$. Otherwise, we continue on, incrementing i. Eventually, either every graph G_i has e inserted, or some spanner T_i adds the edge, in which case we increment t_i and set $f_e = i$, and add i to the AVL tree and stop the process. Thus, the total update time for an insertion is bounded by the table updates, which are constant time, insertion of i into an AVL tree that takes $O(\log \ell)$ time, and the $\leq \ell$ insertions into the graphs G_i for which we are maintaining the

50

spanners (amortized time $O(\ell \log^2(n))$). Note that we trivially satisfy the laziness condition, i.e., either one tree adds the edge e, or no tree adds the edge e.

Next, we consider deletions. Indeed, let some multi-edge $e = (u, v)$ be deleted. First, we access the table. If $w_e \geq t_e + 1$, then this means that not every copy of the edge is being used. So, we simply decrement w_e by 1. If $w_e = t_e + 1$, then we must also go to f_e, and starting with $i = f_e + 1$, remove the edge e from $G_i, G_{i+1}, \ldots$. However, since $e \notin T_i, T_{i+1}, \ldots$, this leads to no cascading changes, as we are removing an edge not present in the spanners.

So, let us consider the remaining case, when $w_e = t_e$, which means that every copy of the multi-edge is being used in the spanners. As we mentioned at the beginning of the proof, all copies of e are interchangeable, so we can always assume without loss of generality that the f_eth spanner is using the copy of the multi-edge that is being deleted. When we remove the edge e, we first decrement w_e. Then, we store the index f_e, and also delete f_e from the AVL tree. We then access the *new* index of the final spanner that contains the edge e, which we denote by f'_e (and update this in the table to be the new final spanner index). For all the indices $i \in [f'_e + 1, f_e - 1]$, we remove the edge e from G_i. Note that this causes no cascading changes, as e was not used in any of these spanners. Then, we remove the edge e from G_{f_e}. Because e was also used in T_{f_e}, this does cause a change, as T_{f_e} now potentially adds a new edge. Indeed, if T_{f_e} does not add a new edge, then nothing changes, so the only interesting case is when T_{f_e} does add a new edge, let us call this e'. We insert f_e into the AVL tree of spanners containing e'. But, because we inserted a copy of e' into T_{f_e}, we must also ensure that there are not too many copies of e' present in the spanners. So, we find $w_{e'}$ and compare this with $t_{e'}$. If $w_{e'} = t_{e'}$ (i.e., every copy of e' was being used), then we simply go to $f_{e'}$, and remove e' from $T_{f_{e'}}$, repeating the above procedure. Note that the number of deletions is bounded by ℓ, as each time a new edge is inserted (i.e., a spanner deletes e and adds e'), the indices we consider go from $f_e \to f_{e'} \to f_{e''}$. So, the indices of the spanners we update are monotonically increasing. Thus we have total amortized update time $O(\ell \log(\ell) \log^2(n))$ for the deletion. $\square$

However, this still is not enough for our purposes. Recall that in Algorithm 10, our target is not only to store spanners, but to have spanners *report* the corresponding hyperedges for each edge. So, to capture this, we introduce the notion of a *labelling* for each multi-edge.

Definition 11.2. Let $G = (V, E)$ be a multi-graph, where for each multi-edge there is a corresponding label $\in \mathcal{U}$ where $\mathcal{U}$ admits a total ordering. We say that a set of spanners $T = T_1, \ldots T_\ell$ is a label-respecting ℓ-bundle of spanners for G if T_i is a spanner of $G - T_1 - T_2 - \cdots - T_{i-1}$, and for any update to T, T reports the change in edge $e = (u, v)$ *as well as* the corresponding label for the edge (u, v) that was changed.

We now show how to implement such a label-respecting ℓ-bundle of spanners in the *decremental* model.

Claim 11.5. There is a decremental algorithm for maintaining a label-respecting ℓ-bundle of spanners of an undirected, unweighted multi-graph G with label set $\mathcal{U}$, with $\leq m$ edges total. The amortized update time is $O(\ell \log(\ell) \log^2(n) + \log(m) \log(\mathcal{U}))$. Moreover, the algorithm satisfies the "lazy" property, in the following sense. Whenever an edge e with corresponding label x is *deleted*, the only potential changes are the removal of e with label x from the bundle of spanners, and the addition of a single new edge (with label) to the bundle of spanners. Further, the data structure can be initialized on an instance with m edges in time $O(m\ell \log(\ell) \log^2(n) + m \log(m) \log(|\mathcal{U}|))$.

Proof. In addition to maintaining the spanners (using Theorem 11.3), we maintain a simple table data structure. This table data structure maps each possible edge $e = (u, v)$ to a tuple with four elements. The first element in the tuple is the number of copies of the multi-edge w_e, the second is the number of spanners which contain the multi-edge t_e, the third element is an AVL tree containing the indices $\subseteq [\ell]$ of all the spanners which contain this multi-edge, as well as a pointer to the final index $f_e \in [\ell]$ in the AVL tree, and the final element is an AVL tree of all valid labels for edge (u, v). After the initialization of

the data structure, and throughout the entire process of processing the (decremental) updates, we always associate the labels for edge $e = (u, v)$ in the t_e spanners using e with the first t_e labels (i.e., the first t_e labels in the sorted order).

Now we show how to handle deletions. Let some multi-edge $e = (u, v)$ with corresponding label x be deleted. First, we access the table. If $w_e \geq t_e + 1$, then this means that not every copy of the edge is being used. So, we simply decrement w_e by 1. If $w_e = t_e + 1$, then we must also go to f_e, and starting with $i = f_e + 1$, remove the edge e from $G_i, G_{i+1}, \dots$. We also search in the AVL tree of valid labels for (u, v) to find the label x, and remove x. If x is among the first t_e labels, we also access the new t_eth label, x'. The data structure reports a change that edge (u, v) has changed from label x to label x'.

Now, let us consider the remaining case, when $w_e = t_e$, which means that every copy of the multi-edge is being used. When we remove the edge e, we first decrement w_e. Then, we store the index f_e, and also delete f_e from the AVL tree. We then access the *new* index of the final spanner to contain the edge e, which we denote by f'_e (and update this in the table to be the new final spanner index). We also remove the label x from the AVL tree. The data structure reports that edge e with label x has been removed from the spanner.

For all the indices $i \in [f'_e + 1, f_e - 1]$, we remove the edge e from G_i. Note that this causes no cascading changes, as e was not used in any of these spanners. Then, we remove the edge e from G_{f_e}. Because e was also used in T_{f_e}, this does cause a change, as T_{f_e} now potentially adds a new edge. Indeed, if T_{f_e} does not add a new edge, then nothing changes, so the only interesting case is when T_{f_e} does add a new edge, let us call this e', so we focus on this case: First, we insert f_e into the AVL tree of the indices of which spanners contain e'. But, because we inserted a copy of e' into T_{f_e}, we must also ensure that there are not too many copies of e' present in the spanners. So, we find $w_{e'}$ and compare this with $t_{e'}$:

1. If $w_{e'} = t_{e'}$ (i.e., every copy of e' was being used), then we simply go to $f_{e'}$, and remove e' from $T_{f_{e'}}$, repeating the above procedure. Note that this cascading effect can only happen ℓ times at most, as each time we go from $f_e \to f_{e'}$, the value increases by at least one.

2. If $w_{e'} > t_{e'}$ (i.e. there were available copies of e'), the data structure increments $t_{e'}$ by 1, and sets $f_{e'} = \max(f_{e'}, f_e)$. If $w_{e'}$ now equals $t_{e'}$, then we additionally remove e' from the final $\ell - f_{e'}$ graphs (but this causes no cascading changes, as none of these graphs were using e'). The data structure outputs that edge e' has been added to the bundle of spanners with label x', where x' is the $t_{e'}$st label among the sorted labels for edge e'.

The total run-time thus sums up to $O(\ell \log(\ell) \log^2(n))$. For maintaining the labels, observe that there are $\leq m$ labels present in any edge slot. Assuming that the label universe is $\mathcal{U}$ and supports comparison operations in time $O(\log(|\mathcal{U}|))$, we can insert / delete the labels in time $O(\log(m) \log(|\mathcal{U}|))$. Since at most $O(1)$ labels are updated, this yields our final update-time.

Finally, to see the initialization time, observe that given a list of multi-edges / labels, we can construct the data structure of Claim 11.4 in total time $O(m \ell \log(\ell) \log^2(n))$. From this point, all that remains is to create the AVL tree of labels. This can be done by (for instance) inserting the labels of each (u, v) into the AVL tree one by one. This requires time $O(m \log(m) \log(|\mathcal{U}|))$ to construct, yielding our run-time. $\qquad \square$

Finally, we require one more tool before being able to present our decremental sparsifier. Recall that we repeatedly vertex-sample the hypergraph H to get a smaller vertex set V', and so for any hyperedge e, we must insert the multi-edges in $K_{V' \cap e}$ into the graph. However, because there are $r \operatorname{polylog}(m, n)$ different vertex sampled graphs, if we just naively compute this intersection separately for each graph, we will be doomed to have a running time of $\Omega(r^2)$ (since each intersection requires time $\Omega(r)$ to calculate). Thus, we now show that just using a table suffices for computing all of these intersections in time $\widetilde{O}(r \operatorname{polylog}(m, n))$.

Claim 11.6. Let $V^{(1)}, \dots V^{r \operatorname{polylog}(m,n)}$ denote the result of independently vertex sampling V at rate $\frac{1}{r}$. There exists a data structure such that given any hyperedge $e \subseteq V$ of arity r, we can compute $e \cap V^{(i)}, \forall i \in [r \operatorname{polylog}(m, n)]$ in time $O(r \operatorname{polylog}(m, n))$.

Proof. We store a table that for every vertex $v \in [n]$, maps it to a list of the corresponding indices $i \in [r \operatorname{polylog}(m, n)]$ for which $v \in V^{(i)}$.

Now, for any hyperedge e, we first initialize $r \operatorname{polylog}(m, n)$ empty lists (one for each $V^{(i)}$), and denote these by $L_1, \ldots L_{r \operatorname{polylog}(m,n)}$. Then, we iterate through e, and for each $v \in e$, we access the table to find which $V^{(i)}$ contain v. For any such i, we then add v to corresponding list L_i. Clearly then, at the end of this procedure L_i contains exactly $e \cap V^{(i)}$.

Next, we bound the run-time. Initializing the lists takes time $O(r \operatorname{polylog}(m, n))$. For each $v \in e$, we access the table to see which $V^{(i)}$ contain v. The time for this is bounded by the number of $V^{(i)}$ that contain v. For this, observe that we are vertex sampling at rate $\frac{1}{r}$ for $r \operatorname{polylog}(m, n)$ rounds. Hence, the number of such $V^{(i)}$ is bounded by $O(\operatorname{polylog}(m, n))$ in expectation. Repeating this for all $v \in e$ (of which there are $\leq 2r$), yields an expected running time of $O(r \operatorname{polylog}(m, n))$ (or also a worst-case run-time with probability $1 - 2^{-\Omega(\log(n) \log(m))}$). $\qquad\square$

11.3 Decremental Recovery Algorithm

Finally, we are now able to present a decremental data structure for Algorithm 9.

Claim 11.7. There is a decremental data structure which, given parameters r, ϵ, maintains a set of hyperedges F in accordance with Algorithm 9. The data structure can be implemented with amortized update time $O(r \operatorname{polylog}(m, n, 1/\epsilon) \cdot (1/\epsilon)^2)$.

Proof. First, we describe the data structure. We store the data structure of Claim 11.6 so that we can efficiently compute the intersections of each hyperedge with the vertex-sampled subsets. We then additionally store the data structure of Claim 11.5, with parameter $\ell = \operatorname{polylog}(m, n)/\epsilon^2$ on each vertex-sampled multi-graph.

However, it remains to show how we can implement Line 11 of Algorithm 9. In particular, we need to show how, given only the (labelled) spanners, we can also exactly recover the corresponding set of hyperedges.

For this, we first explain how to implement the step using a naive method. Indeed, if we simply let the label of each multi-edge denote the corresponding hyperedge, then in addition to the spanners, we can maintain a sorted set of all the labels for which there is a multi-edge in the bundle of spanners using that label, along with a counter for how many occurrences there are for that label. After any deletion of an edge / label pair, the data structure of Claim 11.5 either does nothing (i.e., no changes to the data structure) or reports the inclusion of a new edge / changing of a label, and we simply increment / decrement the corresponding counters in our set of labels. The only overhead here beyond that of Claim 11.5 is in updating the counters of labels (which takes an additional time $O(r \log(n))$ to binary search, since $|\mathcal{U}| \approx n^r$). Note that whenever a new label is introduced for the first time, we also update all down-stream spanners (i.e., to remove the edges corresponding to this newly recovered hyperedge from the later multi-graphs $\Phi(H^{(i+1)}), \ldots$).

Now, we can bound the run-time. The key claim is the following, which is a consequence of the laziness of our data structure.

Claim 11.8. Let $F^{(i)}$ be as defined in Algorithm 10 when the disjoint spanners are implemented with the above data structure. If no hyperedges are inserted into H, then this also holds for $H - \bigcup_{j<i} F^{(j)}$ for every $i \in [16r \log(m) \log(n)^2]$.

Proof. We can prove this by induction on i. The claim is trivially true for $i = 1$. Let us now assume it holds by induction up to $i-1$, and we will show it holds true for i. Indeed, suppose that if no hyperedges are inserted into H, then no hyperedges are inserted in to $H - \bigcup_{j<i-1} F^{(j)}$. Thus, the only way for a hyperedge to be inserted into $H - F^{(i)}$, would be to remove a hyperedge from the $i-1$st set of recovered hyperedges ($F^{(i)} - F^{(i-1)}$). However, this will never happen, as the only time a hyperedge is removed from a recovered set of hyperedges is when the hyperedge itself is deleted, by the laziness of our data structure. $\qquad\square$

53

Note that there might be cascading deletion happening in the vertex-sampled multi-graphs - if in a multi-graph we recover a new hyperedge due to deletion of another previously recovered hyperedge, then we shall remove this hyperedge from all subsequent multi-graphs, potentially causing $\text{polylog}(n, m)$ multi-edge deletions in those multi-graphs. And the latter might trigger further edge deletions for the same reason. However, as we argue below, this does not blow up our amortized runtime since each hyperedge (and their corresponding multi-edges) can only be deleted once from each multi-graph.

Now, the remainder of our proof follows from observing that the hyperedges (and their corresponding multi-edges) can *only be deleted once* from each vertex-sampled multi-graph. Thus, we finish by bounding the total run-time of deleting all the hyperedges once. Indeed, consider a single hyperedge e. To delete the hyperedge, we first compute all of the corresponding multi-edges for each sub-sampled hypergraph. This can be done via Claim 11.6 in time $O(r \, \text{polylog}(m, n))$ total. Next, each hyperedge becomes $\le r \, \text{polylog}(m, n)$ multi-edges. For each multi-edge, deleting it from each label-respecting spanner and reporting the changes can be done in time $O(\text{polylog}(m, n) \cdot (1/\epsilon)^2 \cdot \log(1/\epsilon) \log^2(n) + \log(m) \log(\mathcal{U})) = O(\text{polylog}(m, n) \cdot (1/\epsilon)^2 \log(1/\epsilon) + \log(m) \log(\mathcal{U}))$ via Claim 11.5. Thus, because there are $\le m \cdot r \, \text{polylog}(m, n)$ total multi-edges, the total-time spent by the data structure in removing *every* hyperedge is bounded by

$$O(m \cdot r \log^2(m) \log^3(n)) + O(m \cdot r \, \text{polylog}(m, n) \cdot (1/\epsilon)^2 \cdot \log(1/\epsilon) \log^2(n) + m \cdot r \log(m) \log(\mathcal{U}))$$

$$= O(mr \, \text{polylog}(m, n) \cdot (1/\epsilon)^2 \log(1/\epsilon) + mr \log(m) \log(\mathcal{U})).$$

Thus, the amortized update time, per hyperedge deletion, is $O(r \, \text{polylog}(m, n) \cdot (1/\epsilon)^2 \log(1/\epsilon) + r \log(m) \log(|\mathcal{U}|))$. Note however, that this is not quite the bound that we would like. Indeed, because we are simply using $\mathcal{U}$ to be the entire set of hyperedges of arity $[r, 2r]$, $\log(|\mathcal{U}|) = \Omega(r \log(n))$. Thus, the amortized update time would be $\Omega(r^2)$, when ideally we would only have a linear dependence on r.

To fix this, we introduce our final trick. Instead of associating each multi-edge of a hyperedge e with itself as the label, we instead associate e with a random $C \cdot \log(m)$ bit label, which we call x_e. We include a table which maps x_e to e (and like-wise one which maps e to x_e), and additionally, for the data structure Claim 11.6, instead of accessing this table with the hyperedge e, we instead access it with the label x_e. The main insight is that with overwhelmingly high probability, we have no collisions within the new (smaller) labels, because the hypergraph itself never has more than m hyperedges. Additionally, for each internal operation, we can use $\log(|\mathcal{U}|) = O(\log(m))$. Thus, the new amortized update (i.e., deletion) time is $O(r \, \text{polylog}(m, n) \cdot (1/\epsilon)^2 \log(1/\epsilon))$. $\qquad\square$

We also show that the data structure of Claim 11.7 can be initialized efficiently.

Claim 11.9. The data structure of Claim 11.7 can be initialized on a hypergraph H on n vertices, m hyperedges, arity $[r, 2r]$, and error parameter ϵ in time $O(mr \, \text{polylog}(m, n)(1/\epsilon)^2 \log(1/\epsilon))$.

Proof. Let E denote the edge set with m hyperedges. First, we choose the vertex-sample subsets which requires time $\widetilde{O}(n \log(m))$ (for each vertex v, choose which rounds of vertex sampling v will be present for). Then, we initialize the data structure of Claim 11.6, which takes time $O(r \, \text{polylog}(m, n))$. Then, we choose the random labels we assign to each hyperedge to decrease the time for binary searching. This takes time $O(r + \log(m))$ per hyperedge to store the label, hyperedge pair in the table. We also create a binary search tree of the hyperedges, such that for each hyperedge one can find the corresponding label assigned. This takes time $O(m \log(m) \cdot r \log(n))$ to compute (m edges to sort, time $r \log(n)$ per comparison).

Next, we also store a list of all the *active* hyperedge labels (i.e., labels for which the corresponding hyperedge has not already been recovered in some spanner). Initially, this is just the set of all hyperedge labels. Now, for the first vertex sampled hypergraph, we insert all of the corresponding multi-edges (with their corresponding labels) into the data structure of Claim 11.5, by using Claim 11.6 to find the multi-edges to insert. This takes time $O(m\ell \log(\ell) \log^2(n) + m \log(m) \log(|\mathcal{U}|)) = O(m \, \text{polylog}(m, n)(1/\epsilon)^2 \log(1/\epsilon))$, using the determined values of ℓ, $\log(|\mathcal{U}|)$. Note that this data structure will report some labels as being

used, because some hyperedges will have multi-edges stored in the spanner. Hence, for each of these labels which is used, we remove them from our list of *active* hyperedge labels. If we let κ_1 denote the number of hyperedges recovered in this round, this takes time $\kappa_1 \log^2(m)$ to remove (κ_1 times, binary search in the list which requires $\log(m)$ comparisons, each in time $\log(m)$). Now, we simply repeat this procedure for the second vertex-sampled graph, and so on up until the $r \operatorname{polylog}(m, n)$th round. It must be the case that $\kappa_1 + \kappa_2 + \cdots + \kappa_{r \operatorname{polylog}(m,n)} \leq m$, because no hyperedge can be stored in different rounds of vertex sampling. Thus, the total insertion time across all the rounds of vertex sampling is bounded by $O(mr \operatorname{polylog}(m, n)(1/\epsilon)^2 \log(1/\epsilon))$ Finally, to report the hyperedges which have been recovered, we simply take the labels of hyperedges which were recovered, and use our table to recover the corresponding hypergraph labels. This takes time $\leq m \cdot r \log(n)$ (m hyperedges to search, recovering $r \log(n)$ bits each). Thus, the total time to build this data structure is

$$O(mr \operatorname{polylog}(n, m)) + O(mr \operatorname{polylog}(m, n)(1/\epsilon)^2 \log(1/\epsilon)) = O(mr \operatorname{polylog}(m, n)(1/\epsilon)^2 \log(1/\epsilon)).$$

$\square$

Finally, we can use the data structure from the preceding section to implement our overall sparsifier data structure as in Algorithm 10. In particular, we simply store the data structure of Claim 11.7 for each of $\log(m)$ levels of sparsification. We formalize the claim below:

Claim 11.10. There is a decremental data structure maintaining a $(1 \pm \epsilon)$-spectral sparsifier of a hypergraph H of arity $[r, 2r]$, $\leq m$ hypredges (with probability $1 - 1/\operatorname{poly}(m)$) with amortized update time $O(r \operatorname{polylog}(m, n) \cdot (1/\epsilon)^2 \log(1/\epsilon))$. Further, the intialization time of the data structure is

$$O(mr \operatorname{polylog}(m, n)(1/\epsilon)^2 \log(1/\epsilon)).$$

Proof. We avoid repeating the details from the previous proof. However, the key insight remains the same. Because the data structures are lazy, hyperedges can only be removed a *single* time. In order to implement the $\log(m)$ levels of sampling, we associate each new hyperedge with $\log(m)$ random bits, where the ith bit dictates whether the hyperedge survives the ith level of sampling. Note that, as before, this property does not alter the laziness of the data structure. Thus, it suffices to bound the update time for the operations involving only a single hyperedge, ignoring any cascading effects. The update time for any single hyperedge deletion is simply $\log(m)$ times that of Claim 11.7 (for the at most $\log(m)$ rounds) thus yielding the above claim. Note that the correctness of the data structure follows from Claim 10.4 (and probability of correctness), as we are exactly implementing the same algorithm (only with a different spanner implementation).

To see the initialization time, we simply use the initialization time bound of Claim 11.9. Note that each time we store this data structure, it reports some hyperedges as being recovered. Thus, we maintain an "active list" of hyperedges as in the proof of Claim 11.9, when initializing the data structure at different levels. The initialization is thus at most $\log(m)$ times that of Claim 11.9. $\square$

We are thus ready to conclude:

Theorem 11.11. *For a hypergraph on n vertices and $\leq m$ hyperedges, there is a fully dynamic algorithm for maintaining a $(1 \pm \epsilon)$-spectral sparsifier of size $\widetilde{O}(n \operatorname{polylog}(m)/\epsilon^2)$ hyperedges with an amortized, expected update time of $\widetilde{O}(r \operatorname{polylog}(m)/\epsilon^2)$.*

Proof. This follows from Lemma 11.1, Claim 11.10, and Claim 11.9. The bound on the number of hyperedges follows from the same bound as in the online setting (since the number of spanners that are stored is small) Claim 10.5. $\square$

References

[ABS⁺20] Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020.

[ACL⁺22] Sepehr Assadi, Vaggos Chatziafratis, Jakub Lacki, Vahab Mirrokni, and Chen Wang. Hierarchical clustering in graph streams: Single-pass algorithms and space lower bounds. In Po-Ling Loh and Maxim Raginsky, editors, *Conference on Learning Theory, 2-5 July 2022, London, UK*, volume 178 of *Proceedings of Machine Learning Research*, pages 4643–4702. PMLR, 2022.

[ADK⁺16] Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 335–344. IEEE Computer Society, 2016.

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012.

[AKLP22] Arpit Agarwal, Sanjeev Khanna, Huan Li, and Prathamesh Patil. Sublinear algorithms for hierarchical clustering. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29. ACM, 1996.

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55. ACM, 1996.

[BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 255–262. ACM, 2009.

[BSS12] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012.

[BST19] Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 910–928. IEEE Computer Society, 2019.

[BvdBG⁺22] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. In Mikolaj Bojanczyk, Emanuela Merelli, and David P.

Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[BWG+19]   Maciej Besta, Simon Weber, Lukas Gianinazzi, Robert Gerstenberger, Andrey Ivanov, Yishai Oltchik, and Torsten Hoefler. Slim graph: practical lossy graph compression for approximate graph processing, storage, and analytics. In Michela Taufer, Pavan Balaji, and Antonio J. Peña, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*, pages 35:1–35:25. ACM, 2019.

[CKL22]   Yu Chen, Sanjeev Khanna, and Huan Li. On weighted graph sparsification by linear sketching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 474–485. IEEE, 2022.

[CKN20]   Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 61–72. IEEE, 2020.

[CLTZ18]   T.-H. Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. Spectral properties of hypergraph laplacian and approximation algorithms. *J. ACM*, 65(3):15:1–15:48, 2018.

[DMVZ20]   Dean Doron, Jack Murtagh, Salil P. Vadhan, and David Zuckerman. Spectral sparsification via bounded-independence sampling. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 39:1–39:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[FKN21]   Arnold Filtser, Michael Kapralov, and Navid Nouri. Graph spanners by sketching in dynamic streams and the simultaneous communication model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1894–1913. SIAM, 2021.

[GMT15]   Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In Tova Milo and Diego Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 241–247. ACM, 2015.

[JLLS23]   Arun Jambulapati, James R. Lee, Yang P. Liu, and Aaron Sidford. Sparsifying sums of norms. *CoRR*, abs/2305.09049, 2023.

[JLS23]   Arun Jambulapati, Yang P. Liu, and Aaron Sidford. Chaining, group leverage score overestimates, and fast spectral hypergraph sparsification. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 196–206. ACM, 2023.

[JW18]   Rajesh Jayaram and David P. Woodruff. Perfect lp sampling in a data stream. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 544–555. IEEE Computer Society, 2018.

[Kar93]   David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993.

[KK15]   Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 367–376. ACM, 2015.

[KKTY21a] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Spectral hypergraph sparsifiers of nearly linear size. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1159–1170. IEEE, 2021.

[KKTY21b] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 598–611. ACM, 2021.

[KLM+14] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014.

[KPS24a] Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. Code sparsification and its applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5145–5168. SIAM, 2024.

[KPS24b] Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. Near-optimal size linear sketches for hypergraph cut sparsifiers. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1669–1706. IEEE, 2024.

[KX16]   Ioannis Koutis and Shen Chen Xu. Simple parallel and distributed algorithms for spectral graph sparsification. *ACM Trans. Parallel Comput.*, 3(2):14:1–14:14, 2016.

[Lee23]   James R. Lee. Spectral hypergraph sparsification via chaining. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 207–218. ACM, 2023.

[LM17]   Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2308–2318, 2017.

[LM18]   Pan Li and Olgica Milenkovic. Submodular hypergraphs: p-Laplacians, cheeger inequalities and spectral clustering. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3020–3029. PMLR, 2018.

[LVS+21] Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. Strongly local hypergraph diffusions for clustering and semi-supervised learning. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 2092–2103. ACM / IW3C2, 2021.

[Mey73]   Carl D Meyer, Jr. Generalized inversion of modified matrices. *Siam journal on applied mathematics*, 24(3):315–323, 1973.

[OST23]   Kazusato Oko, Shinsaku Sakaue, and Shin-ichi Tanigawa. Nearly tight spectral sparsification of directed hypergraphs. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 94:1–94:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[Qua24]   Kent Quanrud. *Quotient sparsification for submodular functions*, pages 5209–5248. SIAM, 2024.

[Roc70]   R. Tyrrell Rockafellar. *Convex Analysis*. Princeton Mathematics Series. Princeton University Press, Princeton, New Jersey, 1970.

[SM50]   Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[SS11]   Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

[ST11]   Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

[STY24]   Tasuku Soma, Kam Chuen Tung, and Yuichi Yoshida. Online algorithms for spectral hypergraph sparsification. In Jens Vygen and Jaroslaw Byrka, editors, *Integer Programming and Combinatorial Optimization - 25th International Conference, IPCO 2024, Wrocław, Poland, July 3-5, 2024, Proceedings*, volume 14679 of *Lecture Notes in Computer Science*, pages 405–417. Springer, 2024.

[SY19]   Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2570–2581. SIAM, 2019.

[VBK20]   Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1708–1718. ACM, 2020.

[VBK21]   Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Approximate decomposable submodular function minimization for cardinality-based components. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 3744–3756, 2021.

[ZHTC20]   Chenzi Zhang, Shuguang Hu, Zhihao Gavin Tang, and T.-H. Hubert Chan. Re-revisiting learning on hypergraphs: Confidence interval, subgradient method, and extension to multiclass. *IEEE Trans. Knowl. Data Eng.*, 32(3):506–518, 2020.

[ZLS22]   Yu Zhu, Boning Li, and Santiago Segarra. Hypergraph 1-spectral clustering with general submodular weights. In *56th Asilomar Conference on Signals, Systems, and Computers, ACSSC 2022, Pacific Grove, CA, USA, October 31 - Nov. 2, 2022*, pages 935–939. IEEE, 2022.

A   A Naive Linear Sketching Algorithm

As before, we assume the input hypergraph H has hyperedges of arity within $[r, 2r]$ only, which is without loss of generality up to a $\log r$ factor in the sparsity of the sparisifer. As we mentioned in our technical overview, a balanced weight assignment of the multi-graph is hard to compute in the linear sketching setting as that requires repeat and complete access to the underlying hypergraph. However, one can naively sparsify the hypergraph using the uniform weight assignment, where for each multi-edge associated with a hyperedge e, we assign a weight $1/\binom{|e|}{2}$. We then resort to the sampling rate $p_e = \epsilon^{-2}\operatorname{polylog}(n) \cdot \max_{(u,v)\in e} R^{G(W)}(u, v)$ by [KKTY21a, Lee23, JLS23] with W being the uniform weight assignment. By the analysis in [BST19] (see Section 5.2), we know that

$$p_e \leq \epsilon^{-2}\operatorname{polylog}(n)\frac{1}{r}\sum_{(u,v)\in e} R^{G(W)}(u, v). \tag{39}$$

Note that in the uniform weight assignment W, all edge weights are within a constant factor of $1/r^2$, as the input hypergraph H has hyperedges of arity within $[r, 2r]$ only. Therefore we can in fact work with the unweighted multi-graph $\Phi(H)$ of H, and do effective resistance sampling with an oversampling rate of $r\operatorname{polylog}(n)\epsilon^{-2}$, and recover a hyperedge e if it has an associated multi-edge that got sampled. This is guaranteed to be an oversampling compared to p_e by (39). Moreover, by Theorem 7.1, this can be done by a linear sketch of $\widetilde{O}(nr^2\operatorname{polylog}(m)/\epsilon^2)$ bits using the algorithm in [KLM+14].

B   Space Bounds for [KLM+14]

B.1 ℓ_2 Heavy-Hitters

In this section, we explain in more detail how to derive the space bounds of [KLM+14] when dealing with multi-graphs. We will let u denote the universe size (i.e., the total number of possible multi-edge slots), and let m denote the support size (i.e., the actual maximum number of present multi-edges).

Recall that [KLM+14] require the following notion of an ℓ_2 heavy-hitter:

Definition B.1. For a parameter $\eta \in (0, 1)$ and a vector $x \in \mathbb{R}^u$, we say that a vector w is an ℓ_2 heavy-hitter for x with parameter η if

$$\|x - w\|_\infty \leq \eta\|x\|_2.$$

In [KLM+14], they show the existence of a linear sketch implementing an η ℓ_2 heavy-hitter that requires $O(\log^2(u)/\eta^2)$ bits to store. Here, we show that this bound can more tightly be written as $O(\log(u)\log(m)/\eta^2)$.

Claim B.1. For vectors $x \in \mathbb{R}^u$ with support size bounded by m, and all entries bounded in magnitude by C, there is a linear sketch (using public randomness) implementing an η ℓ_2 heavy-hitter that requires $O(\log(u)\log(mC)/\eta^2 + \log(u)\log(1/\delta))$ bits to represent, returns a vector of sparsity $O(1/\eta^2)$ and has success probability $1 - 1/\operatorname{poly}(u) - \delta$.

Proof. We simply follow the standard CountSketch protocol. In each copy of CountSketch, we initialize a uniformly random hash function h which maps $[u] \to [100/\eta^2]$, and for each $i \in [u]$, we also associate a random sign $\sigma_i \in \pm 1$. For each of the $j \in [100/\eta^2]$ buckets, we store $c_j = \sum_{i:h(i)=j}\sigma_i x_i$. From prior work ([JW18] for instance, or https://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/teaching/15859-fall17/weekEight.pdf), it is known that if we simply store $O(\log(u))$ rounds of this CountSketch (initialized independently), and let $c_j^{(p)}$, $h^{(p)}$ denote the sketch values, hash function respectively in the pth round, then we can acquire a good estimator for each value x_i. Indeed, we get an unbiased estimator for x_i by looking at $\sigma_i^{(j)} \cdot c_{h^{(j)}(i)}^{(j)}$ and taking the median over these values across $j = 1, \ldots, O(\log(u))$. In

these prior works, it is established that this yields estimates to each x_i with additive error at most $\frac{\eta\|x\|_2}{10}$ with probability $1 - 1/\text{poly}(u)$. Note then, by additionally storing a linear sketch for estimating the ℓ_2 norm of x (which can be done to accuracy $\eta/1000$ with probability $1 - \delta$ in space $O(\log(u)\log(1/\delta)/\eta^2)$ via [AMS96]), we can then return every element x_i which attains a value of at least $\frac{8\eta\hat{\ell}_2(x)}{10}$, where $\hat{\ell}_2(x)$ is our estimate of the ℓ_2 norm of x. Because we overestimate $\|x\|_2$ by at most $(\eta/1000)$, and we have additive error at most $\eta\|x\|_2/10$, any element x_i of size at least $\eta\|x\|_2$ will report a size which is $\geq \frac{9\eta\|x\|_2}{10} \geq \frac{8\eta(1+\eta/1000)\|x\|_2}{10}$, and thus be recovered.

Further, note that the sparsity of the returned vector will be $O(1/\eta)$, as at most $\frac{10}{7\eta}$ elements can contribute $\geq 7\eta\|x\|_2/10$ and thus be recovered.

Finally, assuming that the randomness is provided to us via a re-readable random tape, we can see that the total space required by this linear sketch is $O(\frac{1}{\eta^2} \cdot \log(u) \cdot \log(mC) + \log(u)\log(1/\delta))$, where the final $\log(mC)$ term is for storing the running sum c_j in each bucket, and δ is the failure probability of the [AMS96] sketch. $\qquad\square$

Claim B.2. For a multi-graph with u possible edge slots, and at most m multi-edges inserted, one can implement the sketch of [KLM$^+$14] in space $\widetilde{O}(n\log(u)\,\text{polylog}(m)/\epsilon^2)$ with failure probability at most $1 - 1/\text{poly}(n)$, where n is the number of vertices in the multi-graph.

Proof. Recall that [KLM$^+$14] simply stores heavy-hitter sketches of the down-sampled incidence matrices. Letting B denote the incidence matrix, there are at most $O(\log(m))$ levels of sampling we must do until the matrix is empty. For each level, we store a heavy-hitter sketch with parameter $\eta = \frac{\epsilon}{c_1 \cdot \sqrt{\log(m)}}$. We initialize the failure probability $\delta = 1/\text{poly}(n)$, where n is the number of vertices in the multi-graph. It follows then that the total space required is bounded by $O((n\log(m)) \cdot (\log(u)\,\text{polylog}(m)/\epsilon^2 + \log(u)\log(n))) = \widetilde{O}(n\log(u)\,\text{polylog}(m)/\epsilon^2)$ (here we are also using that the entries of the vector we sketch with ℓ_2 heavy hitters is bounded by $\text{poly}(m)$ as well). Correctness follows exactly from the work of [KLM$^+$14], as they use Claim B.1 as a black-box in their sketch. $\qquad\square$