



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

Sanjeev Khanna
sanjeev@cis.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Huan Li
huanli@cis.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Aaron Putterman
aputterman@g.harvard.edu
Harvard University
Cambridge, MA, USA

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 $\tilde{O}(n)$ -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

$$\tilde{O}(nr \text{ polylog}(m)/\epsilon^2)$$

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 $\tilde{O}(r \text{ polylog}(m)/\epsilon^2)$, and produces sparsifiers with $\tilde{O}(n \text{ polylog}(m)/\epsilon^2)$ 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 $\tilde{O}(n \text{ polylog}(m)/\epsilon^2)$ hyperedges in an online manner. When $m \leq \text{poly}(n)$, this improves upon the work of Soma, Tung, and Yoshida (IPCO 2024) by a factor of r , who created online sparsifiers with $\tilde{O}(n(r + \log(m))/\epsilon^2)$ 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.

CCS Concepts

• Theory of computation → Sketching and sampling.

Keywords

Hypergraph, sparsification, streaming

ACM Reference Format:

Sanjeev Khanna, Huan Li, and Aaron Putterman. 2025. Near-Optimal Linear Sketches and Fully-Dynamic Algorithms for Hypergraph Spectral Sparsification. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25)*, June 23–27, 2025, Prague, Czechia. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3717823.3718239>

1 Introduction

Ever since its conception in the work of Karger [22], *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 [22] and Benczúr and Karger [8], these properties took the form of the *cut sizes* in the graph, and this research culminated in the design of an $\tilde{O}(n + m)$ 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 [37], Spielman and Srivastava [36], and Batson, Spielman, and Srivastava [7] studied *spectral graph*

*For the full version of the paper, see <https://arxiv.org/pdf/2502.03313>.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
STOC '25, Prague, Czechia

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1510-5/25/06
<https://doi.org/10.1145/3717823.3718239>

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 $S \subseteq [n]$, $\mathbf{1}_S^T L_G \mathbf{1}_S = |\text{cut}_G(S)|$. Thus, preserving the spectrum of L_G to a $(1 \pm \epsilon)$ factor is a *stronger* notion than $(1 \pm \epsilon)$ cut-sparsification. Yet, [7, 36, 37] 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 [25], 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 [11]), 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 [35] and Bansal, Svensson, and Trevisan [5], 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 $Q_H(\mathbf{1}_S) = |\text{cut}_H(S)|$, and hence hypergraph spectral sparsification generalizes cut sparsification. In a long line of works [13, 17–20, 23, 27, 31, 32] the existence of nearly-optimal size $\tilde{O}(n/\epsilon^2)$ 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 [3], and [21] established algorithms for creating cut and spectral sparsifiers of graphs in the *linear sketching* model of computation which require only $\tilde{O}(n/\epsilon^2)$ bits of space. These algorithms are then immediately amenable to other computational settings such as the massively parallel computation (MPC) model (see e.g., [2, 4]) and the *dynamic streaming* setting (where edges in the stream can be both inserted and deleted).

In a different vein, the work of [1] 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 [1] 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 [14] (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 [6].

In contrast, hypergraph sparsification is well-understood only in a small number of settings. Guha, McGregor, and Tench [16] gave the first *linear sketches* for hypergraph cut-sparsifiers. However, the linear sketches of [16] require $\tilde{O}(nr^2 \text{polylog}(m)/\epsilon^2)$ 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 [24] constructed linear sketches for hypergraph cut-sparsifiers that achieve a nearly-optimal size of $\tilde{O}(nr \log(m)/\epsilon^2)$ 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 [34] 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 [34] shows that one can compute $(1 \pm \epsilon)$ hypergraph spectral sparsifiers with $\tilde{O}(n(r + \log(m))/\epsilon^2)$ 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 [40], clustering [30, 41], and practical compression [10], 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

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

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 reweights 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 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 [12, 15], 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 [21] or [1]). 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 $\text{polylog}(m, n)/\epsilon^2$ 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 $n \text{polylog}(m, n)$, thus ensuring that the sparsifier has only $\tilde{O}(n \text{polylog}(n, m)/\epsilon^2)$ 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.

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 [21] 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 $\tilde{O}(s)$ (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 $\tilde{O}(nr \text{polylog}(m)/\epsilon^2)$ bits of space, and can be used to recover a $(1 \pm \epsilon)$ spectral-sparsifier with probability $1 - 1/\text{poly}(n, m)$.*

Recall that the work of [24] 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 $\tilde{O}(nr \text{polylog}(m)/\epsilon^2)$ 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 $1 - 1/\text{poly}(n, m)$ at the end of the stream.*

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 $\tilde{O}(n \text{polylog}(m)/\epsilon^2)$ 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 [1]. 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 [34] 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 [28–30, 38, 39, 41], 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 [9] 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 [34], who showed the existence of online hypergraph spectral sparsification algorithms which require only $\tilde{O}(n^2 \log(m))$ bits of space, and produce sparsifiers with $\tilde{O}(n(r + \log(m))/\epsilon^2)$ 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 $\tilde{O}(nr \text{polylog}(m)/\epsilon^2)$ bits and the sparsifier contains $\tilde{O}(nr \text{polylog}(m)/\epsilon^2)$ hyperedges.*

The space complexity of $\tilde{O}(nr \text{polylog}(m)/\epsilon^2)$ bits of our online algorithm matches *exactly* the bound proposed in the open question 6.2 of [34], 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. 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.

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. This recursive recover-and-sample procedure serves as the foundation for many works in graph and hypergraph sparsification [16, 24].

Our meta-algorithm is completely specified by a pair of sub-routines 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: 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** $r \text{polylog}(n, m)$ 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 .
-

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 $\Omega(1/\log n)$ [36]. For spectral sparsification of hypergraphs, previous work

Table 1: Summary of our results.

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

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** $i = 0, \dots, \log(m)$ **do**
- 3 Let $F_i = \text{VS}(H_i, \text{polylog}(n, m)/\epsilon^2)$.
- 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 .

[18, 19, 27] 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 *multi-graph*, 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 [18, 19, 27], for any weight assignment W of G , sampling each hyperedge e with probability given by $p_e = \epsilon^{-2} \text{polylog}(n) \cdot \max_{(u,v) \in e} R^{G(W)}(u, v)$ 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 $1/\text{polylog}(n, m)$ 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 $\tilde{O}(n)$), 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. $\text{VS}(H, \lambda)$ 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 $\max_{(u,v) \in e} R^{G(W^*)}(u, v) \leq \theta$, or
- (b) we have that $\Pr[e \in \text{VS}(H, \lambda = \theta^{-1} \text{polylog}(n, m))] \geq 1 - 1/\text{poly}(n, m)$, where the $\text{poly}(n, m)$ in the success probability depends on the $\text{polylog}(n, m)$ in the parameter λ of Algorithm 1.

By setting $\theta = \epsilon^2/\text{polylog}(n, m)$ 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 $\text{VS}(H, \text{polylog}(n, m))$ with high probability, or (ii) can be sampled with probability $1/2$ while preserving the entire spectrum to within $1 \pm \epsilon/\text{polylog}(n, m)$. 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 $x_1, \dots, x_k \in \mathbb{R}^n$ as

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

Notably, $\mathcal{E}^H(x_1, \dots, x_k)$ 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, $\leq k \text{polylog}(n, m)$). 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](#)).

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.

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 ([1, 12, 16, 24, 26] 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 [18], by choosing an appropriate $\text{polylog}(n, m)$, 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 i th 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.

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

To see the sparsity of the sparsifier, we see that each hyper-edge 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 $\tilde{O}\left(\frac{n \text{polylog}(m)}{r \epsilon^2}\right)$ multi-edges, and in total across all rounds, and all $\log(m)$ levels of sampling, we recover $\tilde{O}(n \text{polylog}(m)/\epsilon^2)$ multi-edges. Because hyperedges are recovered only if a corresponding multi-edge is recovered, the final sparsity is therefore $\tilde{O}(n \text{polylog}(m)/\epsilon^2)$ 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 \tilde{H} of H with only $\tilde{O}(n \text{polylog}(m)/\epsilon^2)$ 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 [21] 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, [21] showed that for a graph G on n vertices and a parameter $\epsilon > 0$, there is a linear sketch of size $\tilde{O}(n/\epsilon^2)$ 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 $\tilde{O}(r \log(m))$ 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 $\tilde{O}(r \text{polylog}(m, n))$ linear sketches for multi-graph effective resistance sampling, with each multi-graph defined on $O\left(\binom{n}{r} \cdot r^2\right)$ 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 oversampling rate of $\tilde{O}(\text{polylog}(m, n)/\epsilon^2)$. For each multi-graph, the space required is $\tilde{O}\left(\frac{n}{r} \cdot r \text{polylog}(m)/\epsilon^2\right)$, and across all levels of sampling, and iterations of recovery, the number of multi-graphs

is $O(r \text{polylog}(m, n))$. Thus, the total space required by the linear sketch is $\tilde{O}(nr \text{polylog}(m)/\epsilon^2)$ 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 [1], this is tackled by maintaining disjoint spanners at different levels of sampling.

More clearly, given a graph G , one can store a sequence $T = T_1 \cup \dots \cup T_{\text{polylog}(n, m)/\epsilon^2}$ of disjoint spanners, where each T_i is a $\log(n)$ -spanner of $G - T_1 - \dots - T_{i-1}$. In [1], the authors showed via a simple combinatorial argument that any edge of effective resistance $\geq \frac{\epsilon^2}{\text{polylog}(n)}$ 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 [36, 37], 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  $T^{(i)} = T_1^{(i)} \cup \dots \cup T_{\text{polylog}(n, m)/\epsilon^2}^{(i)}$  be a sequence of
   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  $T^{(1)} \cup 2 \cdot T^{(2)} \cup \dots \cup 2^{\log(m)} \cdot T^{(\log(m))}$ 

```

In [1], 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*

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

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 ($\geq \frac{\epsilon^2}{\text{polylog}(n, m)}$). 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 $\text{polylog}(n, m)$ 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 [1]. 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 [1], 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 $r \text{polylog}(m, n)/\epsilon^2$ 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 $r \text{polylog}(m, n)/\epsilon^2$ corresponding multi-edges across *all* multi-graphs. Upon removing a hyperedge, this leads to an amortized expected time-complexity of $O(r \text{polylog}(m, n)/\epsilon^2)$, as removing a single multi-edge from a single spanner requires time $O(\text{polylog}(n, m))$. The sparsity of the construction will be $\tilde{O}(n \text{polylog}(m)/\epsilon^2)$ 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 $r \text{polylog}(m, n)/\epsilon^2$ $O(\log(n))$ spanners on graphs with $O(\frac{n}{r})$ vertices. This requires storing at most $O(n \text{polylog}(m)/\epsilon^2)$ multi-edges, which stores at most $O(n \text{polylog}(m)/\epsilon^2)$ 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](#).

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 $x_1, \dots, x_k \in \mathbb{R}^n$, their collective energy is

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

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 $x_1, \dots, x_k \in [0, 1]^n$ 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) $\mathcal{E}^H(x_1, \dots, x_k) \leq k \text{polylog}(n, m)/\theta$.

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 $x_1, \dots, x_k \in [0, 1]^n$ such that the following both hold:*

- (1) *Each x_i is spanned by at least one hyperedge in F (cf. [Definition 2.1](#)).*
- (2) $\mathcal{E}^H(x_1, \dots, x_k) \leq k \text{polylog}(n, m)/\theta$.

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

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

where the $\text{poly}(n, m)$ in the success probability depends on the $\text{polylog}(n, m)$ 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, \dots, x_k as

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

We will eventually derive that each important hyperedge is witnessed by a set of k potential vectors with small collective energy (in particular, $\leq k \text{polylog}(n, m)$). 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 [\[32\]](#), 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 [\[32\]](#) 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, \dots, x_k are the indicator vectors of a k -way partition V_1, \dots, V_k of the vertices. That is, x_1, \dots, x_k are all $0/1$ vectors with disjoint supports that form a partition of V . Then, one can verify that their collective energy $\mathcal{E}^H(x_1, \dots, x_k)$ 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 $k \text{polylog}(n, m)$, which is exactly the criteria for determining important hyperedges for cut sparsification by Quanrud [\[32\]](#) 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

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

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 OPT_H :

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

where we write \mathcal{W} to denote the collection of all possible valid weight assignments. If $\text{OPT}_H < 1/\text{polylog}(n,m)$, 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 $\text{OPT}_F \geq 1/\text{polylog}(n,m)$, which can in turn help us find the “bottleneck” hyperedges that prevent OPT_F from ever going below $1/\text{polylog}(n,m)$. 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:

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

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 $(u_1, v_1), \dots, (u_t, v_t)$ 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 OPT_H satisfies the following identity:

$$\frac{1}{\text{OPT}_H} = \max_{W \in \mathcal{W}} \min_i \mathcal{E}_i^W. \quad (5)$$

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:

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

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. 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 [33]) and deduce strong duality for (6). In particular, we have

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

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

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

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. 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,

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

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 [33]) on (8), we show that whenever $\text{OPT}_H \geq 1/\text{polylog}(n,m)$, there are potential vectors

$\{x_1, \dots, x_k\} \subseteq \{x_1^{W^*}, \dots, x_t^{W^*}\}$ satisfying

$$\frac{1}{k} \cdot \mathcal{E}^H(x_1, \dots, x_k) \leq \text{polylog}(n,m). \quad (10)$$

Here, we define

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

to be the *collective energy* of x_1, \dots, 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, \dots, x_k thus together serve as certificate for $\text{OPT}_H \geq 1/\text{polylog}(n,m)$, and the hyperedges containing (u_i, v_i) 's corresponding to x_1, \dots, x_k are considered the “bottleneck” hyperedges.

2.6.2 Recovering Important Hyperedges by Vertex Sampling. Our next step is then to show that the certificate x_1, \dots, 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 [24], 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 defer a full discussion of the approach to the full version of the paper.

Acknowledgements

S. K. and H.L. are supported by NSF awards CCF-2008305 and CCF-2402284. A. P. is 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.

References

- [1] Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. 2016. On Fully Dynamic Graph Sparsifiers. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, Irit Dinur (Ed.). IEEE Computer Society, 335–344. doi:10.1109/FOCS.2016.44
- [2] Arpit Agarwal, Sanjeev Khanna, Huan Li, and Prathamesh Patil. 2022. Sub-linear Algorithms for Hierarchical Clustering. In *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*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/16466b6c95c5924784486ac5a3feeb65-Abstract-Conference.html
- [3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17–19, 2012*, Yuval Rabani (Ed.). SIAM, 459–467. doi:10.1137/1.9781611973099.40
- [4] Sepehr Assadi, Vaggos Chatziafratis, Jakub Lacki, Vahab Mirrokni, and Chen Wang. 2022. Hierarchical Clustering in Graph Streams: Single-Pass Algorithms and Space Lower Bounds. In *Conference on Learning Theory, 2–5 July 2022, London, UK (Proceedings of Machine Learning Research, Vol. 178)*, Po-Ling Loh and Maxim Raginsky (Eds.). PMLR, 4643–4702. <https://proceedings.mlr.press/v178/assadi22a.html>
- [5] Nikhil Bansal, Ola Svensson, and Luca Trevisan. 2019. New Notions and Constructions of Sparsification for Graphs and Hypergraphs. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9–12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 910–928. doi:10.1109/FOCS.2019.00059
- [6] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. 2009. Twice-Ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 – June 2, 2009*, Michael Mitzenmacher (Ed.). ACM, 255–262. doi:10.1145/1536414.1536451
- [7] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. 2012. Twice-Ramanujan Sparsifiers. *SIAM J. Comput.* 41, 6 (2012), 1704–1721. doi:10.1137/090772873
- [8] András A. Benczúr and David R. Karger. 1996. Approximating s - t Minimum Cuts in $\tilde{O}(n^2)$ Time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22–24, 1996*, Gary L. Miller (Ed.). ACM, 47–55. doi:10.1145/237814.237827
- [9] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thataphol Saranurak, Aaron Sidford, and He Sun. 2022. Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4–8, 2022, Paris, France (LIPIcs, Vol. 229)*, Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 20:1–20:20. doi:10.4230/LIPICS.ICALP.2022.20
- [10] Maciej Besta, Simon Weber, Lukas Gianinazzi, Robert Gerstenberger, Andrey Ivanov, Yishai Oltchik, and Torsten Hoefler. 2019. Slim graph: practical lossy graph compression for approximate graph processing, storage, and analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17–19, 2019*, Michela Tauber, Pavan Balaji, and Antonio J. Peña (Eds.). ACM, 35:1–35:25. doi:10.1145/3295500.3356182
- [11] T.-H. Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. 2018. Spectral Properties of Hypergraph Laplacian and Approximation Algorithms. *J. ACM* 65, 3 (2018), 15:1–15:48. doi:10.1145/3178123
- [12] Yu Chen, Sanjeev Khanna, and Huan Li. 2022. 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*. IEEE, 474–485. doi:10.1109/FOCS54457.2022.00052
- [13] Yu Chen, Sanjeev Khanna, and Ansh Nagda. 2020. Near-linear Size Hypergraph Cut Sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16–19, 2020*, Sandy Irani (Ed.). IEEE, 61–72. doi:10.1109/FOCS46700.2020.00015
- [14] Dean Doron, Jack Murtagh, Salil P. Vadhan, and David Zuckerman. 2020. Spectral Sparsification via Bounded-Independence Sampling. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8–11, 2020, Saarbrücken, Germany (Virtual Conference) (LIPIcs, Vol. 168)*, Artur Czumaj, Anuj Dawar, and Emanuela Merelli (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:21. doi:10.4230/LIPICS.ICALP.2020.39
- [15] Arnold Filtser, Michael Kapralov, and Navid Nouri. 2021. Graph Spanners by Sketching in Dynamic Streams and the Simultaneous Communication Model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, Dániel Marx (Ed.). SIAM, 1894–1913. doi:10.1137/1.9781611976465.113
- [16] Sudipto Guha, Andrew McGregor, and David Tench. 2015. Vertex and Hyper-edge Connectivity in Dynamic Graph Streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 – June 4, 2015*, Tova Milo and Diego Calvanese (Eds.). ACM, 241–247. doi:10.1145/2745754.2745763
- [17] Arun Jambulapati, James R. Lee, Yang P. Liu, and Aaron Sidford. 2023. Sparsifying Sums of Norms. CoRR abs/2305.09049 (2023). doi:10.48550/arXiv.2305.09049 arXiv:2305.09049
- [18] Arun Jambulapati, Yang P. Liu, and Aaron Sidford. 2023. Chaining, Group Leverage Score Overestimates, and Fast Spectral Hypergraph Sparsification. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20–23, 2023*, Barna Saha and Rocco A. Servedio (Eds.). ACM, 196–206. doi:10.1145/3564246.3585136
- [19] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. 2021. 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*. IEEE, 1159–1170. doi:10.1109/FOCS52979.2021.00114
- [20] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. 2021. Towards tight bounds for spectral sparsification of hypergraphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21–25, 2021*, Samir Khuller and Virginia Vassilevska Williams (Eds.). ACM, 598–611. doi:10.1145/3406325.3451061
- [21] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. 2014. 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*. IEEE Computer Society, 561–570. doi:10.1109/FOCS.2014.66
- [22] David R. Karger. 1993. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, 25–27 January 1993, Austin, Texas, USA*, Vijaya Ramachandran (Ed.). ACM/SIAM, 21–30. <http://dl.acm.org/citation.cfm?id=313559.313605>
- [23] Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. 2024. Code sparsification and its applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 5145–5168. doi:10.1137/1.9781611977912.185
- [24] Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. 2024. Near-optimal Size Linear Sketches for Hypergraph Cut Sparsifiers. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1669–1706. doi:10.1109/FOCS61266.2024.00105
- [25] Dmitry Kogan and Robert Krauthgamer. 2015. Sketching Cuts in Graphs and Hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11–13, 2015*, Tim Roughgarden (Ed.). ACM, 367–376. doi:10.1145/2688073.2688093
- [26] Ioannis Koutis and Shen Chen Xu. 2016. Simple Parallel and Distributed Algorithms for Spectral Graph Sparsification. *ACM Trans. Parallel Comput.* 3, 2 (2016), 14:1–14:14. doi:10.1145/2948062
- [27] James R. Lee. 2023. Spectral Hypergraph Sparsification via Chaining. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20–23, 2023*, Barna Saha and Rocco A. Servedio (Eds.). ACM, 207–218. doi:10.1145/3564246.3585165
- [28] Pan Li and Olga Milenkovic. 2017. Inhomogeneous Hypergraph Clustering with Applications. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December*

- 4-9, 2017, Long Beach, CA, USA, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 2308–2318. <https://proceedings.neurips.cc/paper/2017/hash/a50abba8132a77191791390c3eb19fe7-Abstract.html>
- [29] Pan Li and Olgica Milenkovic. 2018. Submodular Hypergraphs: p-Laplacians, Cheeger Inequalities and Spectral Clustering. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 3020–3029. <http://proceedings.mlr.press/v80/li18e.html>
- [30] Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. 2021. Strongly Local Hypergraph Diffusions for Clustering and Semi-supervised Learning. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 2092–2103. doi:10.1145/3442381.3449887
- [31] Kazusato Oko, Shinsaku Sakaue, and Shin-ichi Tanigawa. 2023. Nearly Tight Spectral Sparsification of Directed Hypergraphs. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany (LIPIcs, Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 94:1–94:19. doi:10.4230/LIPIcs.ICALP.2023.94
- [32] Kent Quanrud. 2024. *Quotient sparsification for submodular functions*. SIAM, 5209–5248. doi:10.1137/1.9781611977912.187 arXiv:<https://epubs.siam.org/doi/pdf/10.1137/1.9781611977912.187>
- [33] R. Tyrrell Rockafellar. 1970. *Convex Analysis*. Princeton University Press, Princeton, New Jersey.
- [34] Tasuku Soma, Kam Chuen Tung, and Yuichi Yoshida. 2024. Online Algorithms for Spectral Hypergraph Sparsification. In *Integer Programming and Combinatorial Optimization - 25th International Conference, IPCO 2024, Wrocław, Poland, July 3-5, 2024, Proceedings (Lecture Notes in Computer Science, Vol. 14679)*, Jens Vygen and Jaroslav Byrka (Eds.). Springer, 405–417. doi:10.1007/978-3-031-59835-7_30
- [35] Tasuku Soma and Yuichi Yoshida. 2019. Spectral Sparsification of Hypergraphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 2570–2581. doi:10.1137/1.9781611975482.159
- [36] Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926. doi:10.1137/080734029
- [37] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral Sparsification of Graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025. doi:10.1137/08074489X
- [38] Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. 2020. Minimizing Localized Ratio Cut Objectives in Hypergraphs. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 1708–1718. doi:10.1145/3394486.3403222
- [39] Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. 2021. Approximate Decomposable Submodular Function Minimization for Cardinality-Based Components. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 3744–3756. <https://proceedings.neurips.cc/paper/2021/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html>
- [40] Chenzi Zhang, Shuguang Hu, Zhihao Gavin Tang, and T.-H. Hubert Chan. 2020. Re-Visiting Learning on Hypergraphs: Confidence Interval, Subgradient Method, and Extension to Multiclass. *IEEE Trans. Knowl. Data Eng.* 32, 3 (2020), 506–518. doi:10.1109/TKDE.2018.2880448
- [41] Yu Zhu, Boning Li, and Santiago Segarra. 2022. 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*. IEEE, 935–939. doi:10.1109/IEEECONF56349.2022.10052065