

Fully Dynamic Spectral Sparsification for Directed Hypergraphs

Sebastian Forster 

Department of Computer Science, University of Salzburg, Austria

Gramoz Goranci 

Faculty of Computer Science, University of Vienna, Austria

Ali Momeni 

Faculty of Computer Science, UniVie Doctoral School Computer Science DoCS,
University of Vienna, Austria

Abstract

There has been a surge of interest in spectral hypergraph sparsification, a natural generalization of spectral sparsification for graphs. In this paper, we present a simple fully dynamic algorithm for maintaining spectral hypergraph sparsifiers of *directed* hypergraphs. Our algorithm achieves a near-optimal size of $O(n^2/\varepsilon^2 \log^7 m)$ and amortized update time of $O(r^2 \log^3 m)$, where n is the number of vertices, and m and r respectively upper bound the number of hyperedges and the rank of the hypergraph at any time.

We also extend our approach to the parallel batch-dynamic setting, where a batch of any k hyperedge insertions or deletions can be processed with $O(kr^2 \log^3 m)$ amortized work and $O(\log^2 m)$ depth. This constitutes the first spectral-based sparsification algorithm in this setting.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Spectral sparsification, Dynamic algorithms, (Directed) hypergraphs, Data structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.38

Related Version *Full Version*: <https://arxiv.org/abs/2512.21671>

Funding *Sebastian Forster*: This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 947702).

1 Introduction

Sparsification – the process of approximating a graph with another that has fewer edges while preserving a key property – is a central paradigm in the design of efficient graph algorithms. A fundamental property of interest is graph cuts, which are not only foundational in graph theory and serve as duals to flows, but also have widespread applications in areas such as graph clustering [39, 46, 17] and image segmentation [10, 34], to mention a few. In their seminal work, Benczúr and Karger [7] initiated the study of sparsifiers in the context of graph cuts. They showed that any graph admits a sparse reweighted cut-sparsifier whilst paying a small loss in the approximation. Spielman and Teng [52] introduced a variant of sparsification known as spectral sparsification, which generalizes cut sparsification and measures graph similarity via the spectrum of their Laplacian matrices. The development of such sparsification techniques has had a profound impact across algorithm design [42, 48, 53, 45], with the Laplacian paradigm standing out as a key example [9, 51, 14, 31, 36].

Motivated by the need to capture complex interdependencies in real-world data, beyond the pairwise relationships modeled by traditional graphs, there has been a surge of interest in recent years in developing spectral sparsifiers for hypergraphs. These efforts have led



© Sebastian Forster, Gramoz Goranci, and Ali Momeni;
licensed under Creative Commons License CC-BY 4.0

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).

Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thăng

Article No. 38; pp. 38:1–38:20



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



to algorithmic constructions that achieve near-optimal size guarantees [24, 25, 44, 23, 38]. However, a common limitation of these algorithms is the assumption that the input graph is static – an assumption that does not hold in many practical settings. For example, real-world graphs, such as those modeling social networks, are inherently dynamic and undergo continual structural changes. For undirected hypergraphs, this limitation has been addressed in two recent independent works [22, 32], which show that spectral hypergraph sparsifiers can be maintained dynamically, supporting both hyperedge insertions and deletions in polylogarithmic time with respect to input parameters. This naturally leads to the question of whether *directed* hypergraphs also admit similarly efficient dynamic sparsification algorithms.

In this paper, we study problems at the intersection of dynamic graph-based data structures and spectral sparsification for directed hypergraphs. Specifically, we consider the setting where a directed hypergraph undergoes hyperedge insertions and deletions, and the goal is to efficiently process these updates while maintaining a spectral sparsifier that approximates the input hypergraph. Our work builds upon variants of two key algorithmic constructions: the static spectral sparsification framework for directed hypergraphs developed by Oko et al. [44], and the dynamic sparsifier maintenance techniques for ordinary graphs by Abraham et al. [1]. Leveraging insights from both lines of work and modifying them to our setting, we design efficient dynamic algorithms for maintaining spectral sparsifiers of directed hypergraphs, as formalized in the theorem below.

► **Theorem 1.** *Given a directed hypergraph $H = (V, E, \mathbf{w})$ with n vertices, rank r , and at most m hyperedges (at any time), there is a fully dynamic data structure that, with high probability, maintains a $(1 \pm \epsilon)$ -spectral hypersparsifier \tilde{H} of H of size $O(n^2/\epsilon^2 \log^7 m)$ in $O(r^2 \log^3 m)$ amortized update time.*

The guarantees provided by the above theorem are nearly tight, for the following reasons: (1) even reading the hyperedges in a hypergraph of rank r requires $\Theta(r)$ time, which means any update time must inherently depend on r , and (2) in the setting of directed graphs, it is folklore that a complete bipartite graph on n vertices, with all edges directed from one partition to the other, gives an $\Omega(n^2)$ lower bound on the size of any directed sparsifier. Moreover, Oko et al. [44] established an even stronger lower bound, showing that any spectral sparsifier must also incur a $\Omega(\epsilon^{-1})$ dependence. The latter implies that the size of our dynamic sparsifier is optimal up to a factor of ϵ^{-1} and polylogarithmic terms.

Our algorithmic construction also extends naturally to the closely related batch-dynamic setting. In this model – similar to the fully dynamic setting – updates consist of hyperedge insertions and deletions, but are processed in batches, enabling the exploitation of parallelism. This approach is particularly well-suited for handling high-throughput update streams and may more accurately reflect how dynamic changes are managed in real-time systems. Our result in this model is formalized in the theorem below.

► **Theorem 2.** *Given a directed hypergraph $H = (V, E, \mathbf{w})$ with n vertices, rank r , and at most m hyperedges (at any time) undergoing batches of k hyperedge additions or deletions, there is a parallel fully dynamic data structure that, with high probability, maintains a $(1 \pm \epsilon)$ -spectral hypersparsifier \tilde{H} of H of size $O(n^2/\epsilon^2 \log^7 m)$ in $O(kr^2 \log^3 m)$ amortized work and $O(\log^2 m)$ depth.*

The data structure of Theorem 2, along with its analysis, is explained in the full version of the paper (Appendix A).

1.1 Related Work

We briefly discuss the related work for spectral sparsification on both graphs and hypergraphs below.

Static Algorithms

Starting with Spielman and Teng [52], spectral sparsification has been extensively studied on graphs [5, 29, 6, 57, 35, 40, 41]. Recently, the concept has been extended to undirected and directed hypergraphs [50, 4, 24, 25, 47, 44, 38, 23, 33].

Dynamic Algorithms

For undirected graphs, there have been several results for dynamic spectral sparsifiers that use a similar approach to ours. This includes the work of Abraham, Durfee, Koutis, Krinninger, and Peng [1], which achieves polylogarithmic update time using t -bundle spanners, and its extension against an adaptive adversary by Bernstein, Brand, Gutenberg, Nanongkai, Saranurak, Sidford, and Sun [8] and to directed graphs by Zhao [55]. More recently, Khanna, Li, and Putterman [32] achieved a fully dynamic spectral sparsifier with near-optimal size and update time for undirected hypergraphs. A closely related notion of sparsification, namely vertex sparsifiers, has also been studied in the dynamic setting [21, 16, 11, 19, 3, 54, 15].

Distributed and Parallel Algorithms

Koutis and Xu [37] achieved simple algorithms for spectral graph sparsification that can be implemented in many computational models, including both parallel and distributed settings, with sub-optimal guarantees on the sparsifier size. Very recently, Ghaffari and Koo [20] developed a parallel batch-dynamic algorithm for spanners. Other related works include distributed vertex sparsifiers [56, 18].

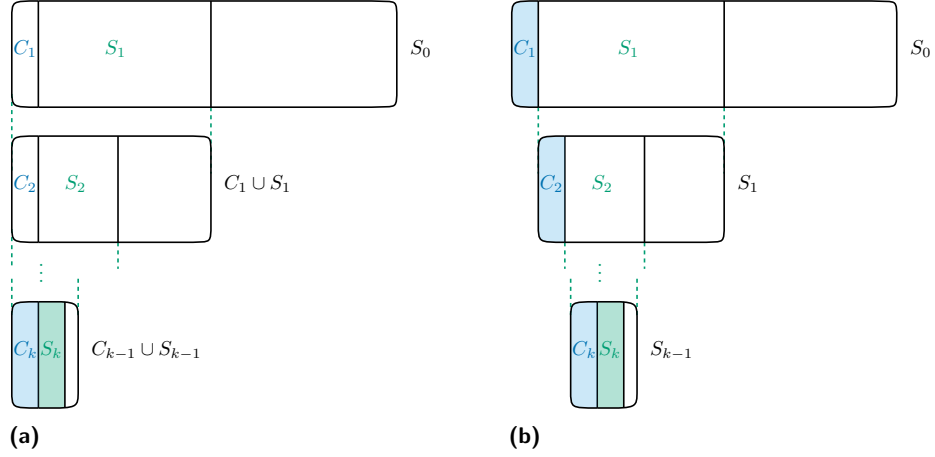
Online and Streaming Algorithms

For graphs, Kelner and Levin [30] extended the sampling scheme based on effective resistances [51] to the semi-streaming setting. Cohen, Musco, and Pachocki [13] obtained an online spectral sparsification algorithm for graphs. Recently, Soma, Tung, and Yoshida [49] proposed an online algorithm for spectral hypergraph sparsification. For graphs in dynamic streams, Ahn, Guha, and McGregor [2] developed a spectral sparsifier. Also, there has been a series of work on spectral sparsification in dynamic streams for both graphs and hypergraphs [26, 28, 27].

1.2 Technical Overview

In this section, we present the main ideas behind our fully dynamic algorithm for maintaining a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of a directed hypergraph H . Our algorithm builds on the static algorithm of [44], which we briefly review.

The algorithm of [44] constructs \tilde{H} by computing a sequence of hypergraphs H_1, \dots, H_k , where H_1 is a spectral hypersparsifier of H , H_2 is a spectral hypersparsifier of H_1 , and so on, until $H_k = \tilde{H}$. Each H_i is obtained via simple sampling scheme (discussed shortly) that guarantees, with high probability, H_i is proportionally smaller than H_{i-1} . As a result, the number of iterations is bounded by $k = O(\log m)$. In constructing H_i from H_{i-1} , the algorithm obtains two sub-hypergraphs of H_{i-1} : the *coreset* hypergraph C_i and the *sampled* hypergraph S_i .



■ **Figure 1** Comparison of (a) the algorithm of [44] and (b) our static algorithm. In each iteration i , their algorithm recurses on $H_{i-1} = C_{i-1} \cup S_{i-1}$ and computes $H_i = C_i \cup S_i$ for the next iteration. In contrast, our algorithm recurses solely on S_{i-1} , adds the coreset C_i to the sparsifier \tilde{H} , and computes the sampled hypergraph S_i for the next iteration. After $k = O(\log m)$ iterations, our algorithm terminates and returns $\tilde{H} = C_1 \cup \dots \cup C_k \cup S_k$, whereas the algorithm of [44] returns $\tilde{H} = C_k \cup S_k$ (the shaded parts in the figures). The increase in the size of \tilde{H} in our algorithm allows us to maintain \tilde{H} efficiently in the dynamic setting, as detailed in Section 3.2.

At a high level, C_i consists of a sufficient number of heavy-weight hyperedges of H_{i-1} (to be specified shortly). This ensures that, when hyperedges are sampled uniformly at random from the remaining hypergraph $H_{i-1} \setminus C_i$ to construct S_i , the resulting union $H_i := C_i \cup S_i$ forms a spectral hypersparsifier of H_{i-1} . More precisely, C_i is constructed as follows. For every pair $(u, v) \in V \times V$, the algorithm selects the $O(\log^3 m / \varepsilon^2)$ heaviest hyperedges whose tail¹ contains u and whose head contains v .² These hyperedges are then added to C_i .

To construct S_i , each hyperedge in $H_{i-1} \setminus C_{i-1}$ is sampled independently with probability $1/2$, and its weight is doubled. They prove that, with high probability, this simple sampling scheme produces a $(1 \pm \varepsilon)$ -spectral hypersparsifier $H_i = C_i \cup S_i$ of H_{i-1} . Moreover, with high probability, $|E(H_i)| \leq 3|E(H_{i-1})|/4$ and so $k = O(\log m)$. See Figure 1a for an illustration of their approach.

Unfortunately, the static algorithm cannot be directly converted into an efficient dynamic algorithm. This is mostly due to the way the hypergraphs in the sequence H_1, \dots, H_k are built on top of each other, as a single change in H can propagate into $O(m)$ changes across the sequence. As an example, we explain how the removal of a hyperedge e from H_i can cause at least two changes in H_{i+1} , which can eventually lead to $O(2^k) = O(m)$ changes throughout the sequence. Assume a hyperedge e is removed from H and that e also belongs to the coreset hypergraph C_i of H_{i-1} . As H_{i+1} is a sub-hypergraph of $H_i = C_i \cup S_i$, e may also belong to H_{i+1} , necessitating its removal from H_{i+1} as well. To maintain C_i as a coreset of H_{i-1} , the algorithm must replace e with a next heaviest hyperedge e' from $H_{i-1} \setminus C_i$. If e' is an unsampled hyperedge (i.e., belongs to $H_{i-1} \setminus S_i$), its addition to C_i (and so H_i) may require updating H_{i+1} as well: as H_{i+1} is a sub-hypergraph of H_i , the (newly added)

¹ In a directed hypergraph, each hyperedge e is a pair $(t(e), h(e))$, with the tail $t(e) \subseteq V$ and the head $h(e) \subseteq V$ reflecting the direction of e . See Section 2 for further details.

² If multiple hyperedges have equal weight, the algorithm picks them arbitrarily.

hyperedge e' could be heavier than some hyperedge e'' in C_{i+1} , necessitating the replacement of e'' in C_{i+1} . Thus, the removal of a hyperedge e from H_i can trigger at least two removals (namely, e and e'') and one insertion (namely, e') in H_{i+1} . See Section 3.1 for further details.

To overcome this issue, our static algorithm deviates from that of [44] in the following way. We ensure that each hyperedge is included in at most one coreset by recursing on the sampled hypergraph S_i rather than on $C_i \cup S_i$. At the same time, we add C_i to the sparsifier \tilde{H} . i.e., we set $\tilde{H} = C_1 \cup \dots \cup C_k \cup S_k$, which can be verified to remain a spectral sparsifier of H (as discussed in Lemma 5). See Figure 1b for an illustration. Using this approach, after the deletion of e from H_i , each hypergraph H_{i+1}, \dots, H_k in the sequence will undergo at most one change. Specifically, in the case of replacing a hyperedge e' by e as in the high-recourse example above, there are two possibilities: either (1) e' does not belong to H_{i+1} (i.e., it is not in the sampled hypergraph S_i), in which case the replacement does not affect H_{i+1} , or (2) e' belongs to H_{i+1} , in which case the update is interpreted as the removal of e' from H_{i+1} (note that since C_i is excluded from H_{i+1} , the hyperedge e no longer belongs to H_{i+1}, \dots, H_k). The downside of this approach is that it increases the size of \tilde{H} from $O(n^2/\varepsilon^2 \log^3(n/\varepsilon))$ to $O(n^2/\varepsilon^2 \text{poly}(\log m))$. This increase becomes noticeable only when m is exponential in n . Even in that case, however, the size of \tilde{H} stays $\text{poly}(n)$, which is asymptotically much smaller than the exponential size of H .

To dynamize our static algorithm, we adapt an approach similar to that of [1] for graphs. We first design a decremental data structure (Algorithm 4 in Section 3.2.1), where the updates consist of only hyperedge deletions, and then use it to design a fully dynamic data structure (Algorithm 5 in Section 3.2.2) via a reduction technique.

Our decremental algorithm leverages the fact that a hyperedge deletion in H causes at most one hyperedge deletion in each hypergraph in the sequence H_1, \dots, H_k . The removal of a hyperedge e from $H_i = C_i \cup S_i$ is handled using a straightforward replacement scheme. If e belongs to the sampled hypergraph S_i , then S_i remains *valid* after the removal of e , in the sense that it is still a set of hyperedges sampled uniformly at random from the updated $H_i \setminus C_i$. The update procedure is more involved if e belongs to the coreset hypergraph C_i . Recall that e was added to C_i through a pair $(u, v) \in V \times V$, where u belongs to the tail of e and v belongs to the head of e . The replacement of e is handled by removing it from all sets representing such pairs and then selecting a hyperedge with high weight that is not already in C_i . Since there are $O(r^2)$ such pairs, this results in an $\tilde{O}(r^2)$ update time for maintaining H_i . See Section 3.2.1 for more details.

To convert our decremental data structure to a fully dynamic one, we leverage the decomposability property of spectral sparsifiers: the union of spectral sparsifiers of hyperedge partitions of H forms a spectral sparsifier of H (see Lemma 3 for a formal statement). The main idea is to maintain a hyperedge partition I_1, \dots, I_l of H where $|E(I_i)| \leq 2^i$ at any time. Deletions are handled by passing them to the respective I_i , whereas insertions are more difficult to handle: the data structure finds an integer j and moves all the hyperedges in I_1, \dots, I_{j-1} to I_j along with the inserted hyperedge. The choice of j depends on the number of insertions so far, with smaller values of j (corresponding to hypergraphs considerably smaller than H) being chosen more frequently. On each I_i , we run our decremental data structure to maintain a spectral sparsifier \tilde{I}_i of I_i , and upon an insertion, we reinitialize it for I_j . The algorithm sets $\tilde{H} = \tilde{I}_1 \cup \dots \cup \tilde{I}_l$, and since $k = O(\log m)$ the desired size of \tilde{H} follows. See Section 3.2.2 for further discussion.

Our approach in designing a decremental data structure and extending it to a fully dynamic one is similar to recent work on dynamic sparsification for undirected hypergraphs [22, 32]. Moreover, [22] also builds on the framework of [44]. The key difference is that we

adapt the directed framework of [44] (specifically, λ -coresets), whereas [22] employs their undirected framework (specifically, t -bundle hyperspanners, a concept also used in [32]). Both [22] and [32] rely on spanner-based techniques to bound the effective resistances in the *associated graph* of the hypergraph as a crucial step in enabling their simple sampling scheme. For the directed case, however, [44] shows that this translates to using coresets, which are structurally simpler than hyperspanners. This structural simplicity allows us to design relatively simpler algorithms compared to those in [22, 32].

It is noteworthy to mention the recent work of [33] that reduces directed hypergraph sparsification to undirected hypergraph sparsification. Combining this reduction with the fully dynamic undirected hypergraph sparsification result of [32] yields a dynamic algorithm for directed hypergraph sparsification with guarantees similar to ours. However, the algorithm of [32] is substantially more involved: (i) it relies on vertex-sampling steps, which our approach does not require, and (ii) it uses dynamic graph spanner constructions in a black-box manner. Moreover, it does not seem straightforward to extend their algorithm to the batch-parallel setting. In comparison, our coreset-based construction is significantly simpler and potentially more practically relevant. It readily extends to the batch-parallel setting and achieves a better and explicit polylogarithmic overhead in both sparsifier size and update time.

Lastly, in the full version of the paper (Appendix A), we show how to parallelize our data structures when H undergoes batches of k hyperedge deletions or additions as a single update. This discussion also explains how to adapt our fully dynamic data structure to support batch updates rather than single updates.

2 Preliminaries

Hypergraphs

A hypergraph $H = (V, E, \mathbf{w})$ consists of a set of vertices V , a set of hyperedges E , and a weight vector $\mathbf{w} \in \mathbb{R}_+^{|E|}$. The direction of a hyperedge $e \in E$ is defined by sets $t(e)$ and $h(e)$ as follows. Each hyperedge $e \in E$ is a pair $(t(e), h(e))$, where both $t(e)$ and $h(e)$ are non-empty subsets of V . The sets $t(e)$ and $h(e)$ are called the *tail* and *head* of e , respectively; they indicate the direction of e . Note that $t(e)$ and $h(e)$ may overlap.

We use $E(H)$ to denote the set E of hyperedges of H whenever necessary, to avoid possible confusion. We define $n = |V|$ and $m = |E|$ and call H an m -edge n -vertex hypergraph. We say H is of rank r if, for every hyperedge $e \in E$, $|t(e) \cup h(e)| \leq r$.

Given a vector $\mathbf{x} \in \mathbb{R}^n$ defined on the set of vertices V , we define x_v to be the value of vector \mathbf{x} at the element associated with vertex $v \in V$. Similarly, for a vector $\mathbf{w} \in \mathbb{R}_+^m$ defined on the set of hyperedges E , we define w_e to be the value of vector \mathbf{w} at the element associated with hyperedge $e \in E$.

Spectral Sparsification of Directed Hypergraphs

We define the spectral property of a directed hypergraph $H = (V, E, \mathbf{w})$ using the energy function defined in the following. For a vector $\mathbf{x} \in \mathbb{R}^n$, we define the energy of \mathbf{x} with respect to H as

$$Q_H(\mathbf{x}) = \sum_{e \in E} w_e \max_{u \in t(e), v \in h(e)} (x_u - x_v)_+^2,$$

where $(x_u - x_v)_+ = \max\{x_u - x_v, 0\}$ and $(x_u - x_v)_+^2 = ((x_u - x_v)_+)^2$. Note that this definition generalizes a similar definition for graphs, given by $Q_G(\mathbf{x}) = \sum_{uv \in E} w_{uv} (x_u - x_v)^2$, which represents the total energy dissipated in G when viewed as an electrical network. In

such electrical network, the endpoints of each edge uv have potentials x_u and x_v , respectively, and the edge itself has resistance $1/w_{uv}$. This interpretation is closely related to the notion of electrical flows, a concept that is deeply intertwined with spectral analysis.

The central object of this paper is the notion of a spectral hypersparsifier. A hypergraph $\tilde{H} = (V, \tilde{E}, \tilde{\mathbf{w}})$ is called a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H if for every vector $\mathbf{x} \in \mathbb{R}^n$,

$$(1 - \varepsilon)Q_{\tilde{H}}(\mathbf{x}) \leq Q_H(\mathbf{x}) \leq (1 + \varepsilon)Q_{\tilde{H}}(\mathbf{x}).$$

The following lemma will be useful later in proving the guarantees of our algorithm.

► **Lemma 3 (Decomposability).** *Let H_1, \dots, H_k partition the hyperedges of a hypergraph H . For each $1 \leq i \leq k$, let \tilde{H}_i be a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H_i . Then, the union $\bigcup_{i=1}^k \tilde{H}_i$ is a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H .*

Proof. By definition, for every vector $\mathbf{x} \in \mathbb{R}^n$, we have

$$(1 - \varepsilon)Q_{\tilde{H}_i}(\mathbf{x}) \leq Q_{H_i}(\mathbf{x}) \leq (1 + \varepsilon)Q_{\tilde{H}_i}(\mathbf{x}).$$

Summing over all $1 \leq i \leq k$, results in

$$(1 - \varepsilon) \sum_{i=1}^k Q_{\tilde{H}_i}(\mathbf{x}) \leq \sum_{i=1}^k Q_{H_i}(\mathbf{x}) \leq (1 + \varepsilon) \sum_{i=1}^k Q_{\tilde{H}_i}(\mathbf{x}),$$

which means

$$(1 - \varepsilon)Q_{\tilde{H}}(\mathbf{x}) \leq Q_H(\mathbf{x}) \leq (1 + \varepsilon)Q_{\tilde{H}}(\mathbf{x})$$

as H_1, \dots, H_k partition H . ◀

Chernoff Bound [12, 43]

Let X_1, \dots, X_k be independent random variables, where each X_i equals 1 with probability p_i , and 0 otherwise. Let $X = \sum_{i=1}^k X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^k p_i$. Then, for all $\delta \geq 0$,

$$\mathbb{P}[X \geq (1 + \delta)\mu] \leq \exp\left(-\frac{\delta^2 \mu}{2 + \delta}\right). \quad (1)$$

Parallel Batch-Dynamic Model

We use the work-depth model to analyze our parallel algorithm. Work is defined as the total number of operations done by the algorithm, and depth is the length of the longest chain of dependencies. Intuitively, work measures the time required for the algorithm to run on a single processor, whereas depth measures the optimal time assuming the algorithm has access to an unlimited number of processors.

In the batch-dynamic setting, each update consists of a batch of k insertions or deletions, and the goal is to take advantage of performing several hyperedge insertions or deletions as a single update to improve the work and depth of the parallel algorithm. Note that this model is equivalent to the one with mixed updates (i.e., when the batch consists of both insertions and deletions), as this can be transformed into two steps, each consisting of insertions and deletions separately, without asymptotically increasing the work or depth.

■ **Algorithm 1** Coreset-And-Sample(H, ε).

Input: an m -edge hypergraph $H = (V, E, \mathbf{w})$ and $0 < \varepsilon < 1$
Output: a coreset C of H and a sampled hypergraph S

- 1 $\lambda \leftarrow c_\lambda \log^3 m / \varepsilon^2$ /* c_λ is a sufficiently large constant */
- 2 $C, S \leftarrow (V, \emptyset, \mathbf{0})$
- 3 Order the hyperedges of H by their weights in decreasing order
- 4 **foreach** pair $(u, v) \in V \times V$ **do** /* compute C */
- 5 Construct the set $E(u, v) \subseteq E$ such that $e \in E(u, v)$ iff $u \in t(e)$ and $v \in h(e)$
- 6 Add the first λ hyperedges (if any) in the ordering from $E(u, v) \setminus C$ to C while retaining their weights
- 7 **foreach** hyperedge e of $H \setminus C$ **do** /* compute S */
- 8 With probability $1/2$, add e to S and double its weight
- 9 **return** (C, S)

3 Dynamic Spectral Sparsification

In this section, we present our fully dynamic algorithm of Theorem 1 for maintaining a $(1 \pm \varepsilon)$ -spectral hypersparsifier of a directed hypergraph $H = (V, E, \mathbf{w})$. To achieve this goal, we use a static algorithm as the cornerstone for our dynamic algorithm. The static algorithm is explained in Section 3.1 and is followed by the dynamic data structure in Section 3.2.

3.1 The Static Algorithm

We start by briefly explaining the algorithm of [44], which serves as a foundation for our algorithm. Their algorithm constructs a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H using an iterative approach: starting with $H_0 = H$, each iteration i computes a sub-hypergraph H_i of H_{i-1} , until the final iteration computes $H_{i_{\text{last}}}$, where $\tilde{H} = H_{i_{\text{last}}}$.

To compute H_i from H_{i-1} , the algorithm first obtains a λ -coreset C_i of H_{i-1} . Roughly speaking, C_i is a sub-hypergraph containing “heavyweight” hyperedges of H_{i-1} and is defined as follows. The algorithm defines an ordering on hyperedges in H by their weights in decreasing order. Note that this ordering naturally extends to every hypergraph H_i as a sub-hypergraph of H . To construct C_i , the algorithm examines every pair $(u, v) \in V \times V$ and selects the first λ hyperedges in the ordering (if any) that are not already in C_i , whose tail contains u and whose head contains v . These hyperedges are then added to C_i . The second building-block of H_{i-1} is the sampled hypergraph S_i of H_{i-1} defined as follows. The algorithm samples the non-heavy hyperedges (i.e., the ones in $H_{i-1} \setminus C_i$) with probability $1/2$ and adds them to S_i while doubling their weight. Consequently, $H_i = C_i \cup S_i$. See Algorithm 1 for a pseudocode.

Having heavyweight hyperedges in C_i ensures that the simple sampling scheme used for constructing S_i results in $H_i = C_i \cup S_i$, which, with high probability, is a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H_{i-1} [44, Lemma 4.3]. Due to the sampling scheme, H_i is roughly half the size of H_{i-1} , which ensures the termination of the algorithm after $i_{\text{last}} = O(\log m)$ iterations. Using this sequence $H_1, \dots, H_{i_{\text{last}}}$ of hypergraphs, they achieve a sparsifier \tilde{H} with an almost optimal size of $O(n^2/\varepsilon^2 \log^3(n/\varepsilon))$ [44, Theorem 1.1]. See Figure 1a for an illustration.

Unfortunately, employing the sequence of hypergraphs $H_1, \dots, H_{i_{\text{last}}}$ can result in $O(m)$ recourse (and consequently, update time) in the dynamic setting. For example, consider the removal of a hyperedge e from H that also belongs to the coreset C_i of H_{i-1} . In this scenario, to ensure C_i remains a λ -coreset and so $H_i = C_i \cup S_i$ remains a $(1 \pm \varepsilon_i)$ -spectral

■ **Algorithm 2** Spectral-Sparsify(H, ε).

Input: hypergraph $H = (V, E, \mathbf{w})$ and $0 < \varepsilon < 1$
Output: a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H

- 1 $i \leftarrow 0$
- 2 $k \leftarrow \lceil \log_{3/4} m \rceil$
- 3 $m^* \leftarrow n^2 / \varepsilon^2 \log^3 m$
- 4 $\tilde{H} \leftarrow (V, \emptyset, \mathbf{0})$
- 5 $S_0 \leftarrow H$
- 6 **while** $i \leq k$ and $|E(H_i)| \geq 32cm^*$ **do** /* c is from Lemma 4 */
- 7 $(C_{i+1}, S_{i+1}) \leftarrow \text{CORESET-AND-SAMPLE}(S_i, \varepsilon / (2k))$
- 8 $\tilde{H} \leftarrow \tilde{H} \cup C_{i+1}$
- 9 $i \leftarrow i + 1$
- 10 $i_{\text{last}} \leftarrow i$
- 11 $\tilde{H} \leftarrow \tilde{H} \cup S_{i_{\text{last}}}$
- 12 **return** \tilde{H}

hypersparsifier of H_{i-1} , we need to replace e with another hyperedge e' from $H_{i-1} \setminus C_i$. i.e., if e was added to C_i through the pair (u, v) , hyperedge e' in $H_{i-1} \setminus C_i$ is the next hyperedge in the ordering whose tail contains u and whose head contains v . Since S_i is a set of sampled hyperedges uniformly at random, it may be the case that e' does not belong to S_i , and therefore to none of H_j for $j > i$. Since e' is added to H_i after the update (as it now belongs to C_i), it must be taken into account in the update of H_{i+1} as a sub-hypergraph of (newly updated) H_i . But now, e' may be heavier than another hyperedge e'' in C_{i+1} , which necessitates the replacement of e'' with e' . Since e can be present in C_{i+1} as well, this means that the deletion of e from H_i can result in at least 2 changes in H_{i+1} , and at least $2^{k-i} = O(m)$ changes in the sequence, which is too expensive to afford.

To alleviate this issue, our static algorithm deviates from that of [44] by recursing on S_i instead of $C_i \cup S_i$ as follows. At each iteration i , the algorithm recurses on S_{i-1} , adds the coreset C_i of S_{i-1} to \tilde{H} , and samples the rest, $S_{i-1} \setminus C_i$, to obtain a smaller hypergraph S_i for the next iteration. More precisely, the algorithm starts with $S_0 = H$ and an empty sparsifier \tilde{H} . In the first iteration, it computes the coreset C_1 of S_0 , and adds it to \tilde{H} . The algorithm then samples the remaining hypergraph $S_0 \setminus C_1$ to compute S_1 by sampling every hyperedge in $S_0 \setminus C_1$ with probability $1/2$ and doubling their weight. Similar to [44], with high probability, $C_i \cup S_i$ is a $(1 \pm \varepsilon_i)$ -spectral hypersparsifier of S_{i-1} (Lemma 4). The algorithm then recurses on S_1 , and so on. Thus, our algorithm adds the sequence of coresets $C_1, \dots, C_{i_{\text{last}}}$ to \tilde{H} while recursing on the sequence of hypergraphs $S_1, \dots, S_{i_{\text{last}}}$. In the last iteration i_{last} , our algorithm adds $C_{i_{\text{last}}}$ as well as $S_{i_{\text{last}}}$ to \tilde{H} . See Algorithm 2 for a pseudocode and Figure 1b for an illustration.

This technique, which is similar to the one used in [1] for graphs, ensures that each hyperedge of \tilde{H} is associated with at most *one* coreset. This guarantees $O(\log m)$ hyperedge deletions from $S_1, \dots, S_{i_{\text{last}}}$ after a hyperedge deletion in H (Lemma 8), but in return, results in a poly($\log m$) overhead in the size of \tilde{H} as we include the coresets $C_1, \dots, C_{i_{\text{last}}}$ in \tilde{H} . i.e., our algorithm ensures that \tilde{H} is a desired sparsifier of size $O(n^2 / \varepsilon^2 \text{poly}(\log m))$, which is asymptotically much smaller than H even when m is exponential in n .

The rest of this section examines the correctness of Algorithms 1 and 2. Algorithm 1, used as a subroutine of Algorithm 2, computes a coreset and a sampled hypergraph of the input hypergraph. The guarantees of Algorithm 1 are stated in the following lemma.

38:10 Fully Dynamic Spectral Sparsification for Directed Hypergraphs

► **Lemma 4.** *Let $0 < \varepsilon < 1$ and let $H = (V, E, \mathbf{w})$ be an m -edge n -vertex hypergraph. For any positive constant $c \geq 1$, if $m \geq c \log m$, then Algorithm 1 returns a coresset C and a sampled hypergraph S such that $C \cup S$ is a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H of size $O(m/2 + (2cm \log m)^{1/2} + \lambda n^2)$ with probability at least $1 - 1/m^c$.*

Proof. The lemma is derived from [44, Lemma 4.3]. The only difference is that we guarantee a probability of success of at least $1 - 1/\text{poly}(m)$, instead of $1 - 1/\text{poly}(n)$. Thus, we only prove the claim about the probability using an argument similar to that in [44, Lemma 4.4].

High probability claim. For every hyperedge e in $H \setminus C$, we define the random variable X_e to be equal 1 if e is sampled to be in S , and 0 otherwise. Let $X = \sum_{e \in S} X_e$. Since each hyperedge in S is sampled independently with probability $1/2$, we can use Equation (1) and we have $\mu = \mathbb{E}[X] = 1/2|H \setminus C| \leq m/2$. By substituting $\delta = (2c \log m/m)^{1/2} \leq 2$ in Equation (1),

$$\mathbb{P} \left[X \geq \left(1 + \left(\frac{8c \log m}{m} \right)^{1/2} \right) \frac{m}{2} \right] \leq \exp \left(-\frac{1}{4} \left(\frac{8c \log m}{m} \right) \frac{m}{2} \right) = \exp(-c \log m) = \frac{1}{m^c},$$

or equivalently

$$\mathbb{P} \left[X \leq \frac{m}{2} + (2cm \log m)^{1/2} \right] \geq 1 - \frac{1}{m^c}. \quad (2)$$

Since each pair $(u, v) \in V \times V$ adds at most λ hyperedges to S , we have $|S| = O(\lambda n^2)$. Together with Equation (2), it follows that $\tilde{H} = C \cup S$ has size $O(m/2 + (2cm \log m)^{1/2} + \lambda n^2)$ with probability at least $1 - 1/m^c$. ◀

The guarantees of Algorithm 2 are stated in the following lemma.

► **Lemma 5.** *Let $0 < \varepsilon < 1$ and let $H = (V, E, \mathbf{w})$ be an m -edge n -vertex hypergraph. Then, with high probability, Algorithm 2 returns a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H of size $O(n^2/\varepsilon^2 \log^6 m)$.*

Proof. We prove each guarantee separately below.

Approximation guarantee. We prove that $H'_l = \bigcup_{j=1}^l C_j \cup S_l$ is a $(1 \pm \varepsilon/(2k))^l$ -spectral hypersparsifier of H by induction on l .

If $l = 1$, by Lemma 4, $H_1 = C_1 \cup S_1$ is a $(1 \pm \varepsilon/(2k))$ -spectral hypersparsifier of H .

Suppose that, for an integer $l > 1$, H'_l be a $(1 \pm \varepsilon/(2k))^l$ -spectral hypersparsifier of H . To compute H'_{l+1} , the algorithm computes a $(1 + \varepsilon/(2k))$ -spectral hypersparsifier $\tilde{S}_l = C_{l+1} \cup S_{l+1}$ of S_l . Since $\bigcup_{j=1}^l C_j \cup S_l$ partition H'_l , by Lemma 3, $H'_{l+1} = \bigcup_{j=1}^l C_j \cup (C_{l+1} \cup S_{l+1})$ is a $(1 \pm \varepsilon/(2k))$ -spectral hypersparsifier of H'_l . For every vector $\mathbf{x} \in \mathbb{R}^n$, we have

$$Q_{H'_{l+1}}(\mathbf{x}) \leq (1 + \varepsilon/(2k)) Q_{H'_l}(\mathbf{x}) \leq (1 + \varepsilon/(2k)) (1 + \varepsilon/(2k))^l Q_H(\mathbf{x}) = (1 + \varepsilon/(2k))^{l+1} Q_H(\mathbf{x}).$$

Similarly, $(1 - \varepsilon/(2k))^{l+1} Q_H(\mathbf{x}) \leq Q_{H'_{l+1}}(\mathbf{x})$, and so H'_{l+1} is a $(1 \pm \varepsilon/(2k))^{l+1}$ -spectral hypersparsifier of H .

The desired bound follows from the fact that $i_{\text{last}} \leq k$, and

$$(1 + \varepsilon/(2k))^k \leq (1 + \varepsilon) \quad \text{and} \quad (1 - \varepsilon) \leq (1 - \varepsilon/(2k))^k.$$

Size of \tilde{H} . Let m_i be the number of hyperedges present in S_i , where $1 \leq i \leq i_{\text{last}}$. We first show that $m_i \leq 3m_{i-1}/4$. By Equation (2), S_i has size $O(m_{i-1}/2 + (2cm_{i-1} \log m_{i-1})^{1/2})$, so it suffices to show that $(2cm_{i-1} \log m_{i-1})^{1/2} \leq m_{i-1}/4$. By the assumption of the loop, we have

$$m_{i-1} \geq 32cm^* = 32cn^2/\varepsilon^2 \log^3 m \geq 32c \log m_{i-1},$$

and thus $(2cm_{i-1} \log m_{i-1})^{1/2} \leq m_{i-1}/4$. This means that the size of $S_{i_{\text{last}}}$ is

$$O(\max\{\log m, m^*\}) = O(n^2/\varepsilon^2 \log^3 m).$$

The rest of \tilde{H} consists of the coresets $C_1, \dots, C_{i_{\text{last}}}$. By Lemma 4, each C_i has size $O(\lambda_i n^2) = n^2/(\varepsilon/2k)^2 \log^3 m$. Since $k = O(\log m)$, the total size of the coresets is

$$O\left(\sum_{l=1}^k k^2 n^2/\varepsilon^2 \log^3 m\right) = O(n^2/\varepsilon^2 \log^6 m).$$

Therefore, the total size of \tilde{H} is

$$O(n^2/\varepsilon^2 \log^3 m + n^2/\varepsilon^2 \log^6 m) = O(n^2/\varepsilon^2 \log^6 m).$$

High probability claim. Since $i_{\text{last}} = O(\log m)$ and from Lemma 4, \tilde{H} is a $(1 \pm \varepsilon)$ -spectral hypersparsifier with probability at least $1 - O(\log m/m^c)$. The claim follows by choosing $c \geq 2$. \blacktriangleleft

3.2 The Dynamic Algorithm

In this section, we dynamize our static algorithm (Algorithm 2). We first design a decremental data structure in Section 3.2.1, where H undergoes only hyperedge deletions. Then, we reduce it to a fully dynamic data structure in Section 3.2.2 using the following lemma.

► **Lemma 6.** *Assume that there is a decremental algorithm that with probability at least $1 - 1/\text{poly}(m')$, maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier of any hypergraph with m' initial hyperedges in $T(m', n, \varepsilon^{-1})$ amortized update time and of size $S(m', n, \varepsilon^{-1})$, where S and T are monotone non-decreasing functions. Then, the algorithm can be transferred into a fully dynamic algorithm that, with high probability, maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier of any hypergraph with m hyperedges (at any point) in $O(T(m, n, \varepsilon^{-1}) \log m)$ amortized update time of size $O(S(m, n, \varepsilon^{-1}) \log m)$.*

The fully dynamic to decremental reduction (Lemma 6) uses the batching technique [1, 22, 32] and is explained in Section 3.2.2.

3.2.1 Decremental Spectral Sparsifier

In this section, we explain our data structure (Algorithm 4) to decrementally maintain a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H . As Algorithm 1 is used as a subroutine of Algorithm 2, we begin by explaining its decremental implementation (presented in Algorithm 3).

Decremental Implementation of Algorithm 1

To decrementally maintain a coreset C and a sampled hypergraph S of H , we need to ensure that after each deletion, the maintained $C \cup S$ remains a valid sparsifier for H . i.e., it continues to be a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H (Lemma 4).

Recall that, for every pair $(u, v) \in V \times V$, Algorithm 1 defines the set $E(u, v)$ of hyperedges where $e \in E(u, v)$ iff $u \in t(e)$ and $v \in h(e)$. It then constructs a coreset C by adding λ (defined in Algorithm 1) heaviest hyperedges of $E(u, v) \setminus C$ to C . The hypergraph S is then obtained by sampling each hyperedge in $H \setminus C$ with probability $1/2$ while doubling its weight.

To construct C , the algorithm greedily chooses a pair (u, v) , adds its heavyweight hyperedges to C , and continues with another pair (u', v') . Note that the order of choosing the pairs might affect the choice of hyperedges included in C . Nevertheless, the guarantee of the algorithm (Lemma 4) is independent of this order. For example, it does not matter whether (u, v) is chosen first or (u', v') is. We will use this fact later to maintain a valid C and S after each deletion.

We use the same procedure to initialize the decremental implementation. Since we would need to find the heaviest hyperedges in $E(u, v) \setminus C$ after a deletion, we also order each $E(u, v)$.

Suppose that hyperedge e has been removed from H . Then, e must belong to one of the three cases below, which together cover all hyperedges of H .

- If e belongs to neither C nor to S , then its removal does not affect $C \cup S$. This is because C still contains the heaviest hyperedges associated with each pair $(u, v) \in V \times V$, and each hyperedge in S has been independently sampled.
- If e belongs to S , similar to the previous case, the sampled hypergraph S after the removal of e is still a valid sample. Thus, no further changes are required to maintain $C \cup S$.
- If e belongs to C , then the deletion from C translates to undoing the addition of e to C from the specific set $E(u, v)$ that originally added e to C . In this case, since e no longer exists in $E(u, v)$, we add a heaviest hyperedge e' from $E(u, v) \setminus C$ to C . Since the order of hyperedge addition to C does not affect its guarantees (Lemma 4), the updated C remains valid. For bookkeeping reasons, if e' was previously sampled, we remove it from S . Based on the previous discussion, the updated S is also valid.

In addition to the changes explained above, the maintenance involves the removal of e from every set $E(u, v)$ containing e , which, as explained in the lemma below, adds $O(\log m)$ overhead to the update time. Putting it all together, Algorithm 3 is our decremental data structure for maintaining $C \cup S$. We have the following.

► **Lemma 7.** *Given a constant $c \geq 2$ and an m -edge n -vertex hypergraph $H = (V, E, \mathbf{w})$ of rank r undergoing hyperedge deletions. If $m \geq c \log m$, then Algorithm 3 maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier $C \cup S$ of H of size $O(m/2 + (2cm \log m)^{1/2} + n^2/\varepsilon^2 \log^3 m)$ in $O(r^2 \log m)$ amortized update time with probability at least $1 - 1/m^{c-1}$.*

Proof. Suppose the hyperedge e is removed from H .

Correctness. From the discussion above, it follows immediately that $C \cup S$, after the update, is a valid sparsifier for H ; the correctness of its spectral property is addressed in the high probability claim below.

Size of $C \cup S$. Since after each hyperedge deletion from $C \cup S$, the algorithm substitutes it with at most one other hyperedge from H , the size of $C \cup S$ is monotonically decreasing. Thus, $C \cup S$ has the same size guarantee as of Algorithm 1 explained in Lemma 4, which is

$$O(m/2 + (2cm \log m)^{1/2} + \lambda n^2) = O(m/2 + (2cm \log m)^{1/2} + n^2/\varepsilon^2 \log^3 m).$$

■ **Algorithm 3** Decremental-Coreset-And-Sample(H, ε).

Input: an m -edge hypergraph $H = (V, E, \mathbf{w})$ and $0 < \varepsilon < 1$
Maintain: a coreset C and a sampled hypergraph S of H

- 1 $C, S \leftarrow (V, \emptyset, \mathbf{0})$
- 2 **Procedure** INITIALIZE
- 3 $(C, S) \leftarrow \text{CORESET-AND-SAMPLE}(H, \varepsilon)$ /* initialize C and S */
- 4 **foreach** pair $(u, v) \in V \times V$ **do**
- 5 Order hyperedges in $E(u, v)$ from highest to lowest weight
- 6 **Procedure** DELETE(e)
- 7 Remove e from H
- 8 **if** C contains e **then**
- 9 Let $E(u, v)$ be the set from which e was added to C
- 10 Choose a hyperedge e' in $E(u, v) \setminus C$ with highest weight and add it to C
- 11 Remove e' from S
- 12 Remove e from C
- 13 **foreach** pair $(u, v) \in V \times V$ with $u \in t(e)$ and $v \in h(e)$ **do**
- 14 Remove e from $E(u, v)$
- 15 Remove e from S

Update time. Since each hyperedge e contains $O(r)$ vertices, the total number of $E(\cdot, \cdot)$'s containing e is $O(r^2)$. Thus, constructing $E(\cdot, \cdot)$'s, takes $O(mr^2)$ time, while ordering them adds $O(mr^2 \log m)$ time to the initialization time as well. As explained before, the deletion of e from H translates to its deletion from at most r^2 sets of $E(\cdot, \cdot)$'s, each of which takes $O(\log m)$ time, i.e., $O(mr^2 \log m)$ total time. Other changes, such as substituting another hyperedge in S , can be done by probing $E(\cdot, \cdot)$'s only once in total. We conclude that the total update time is $O(mr^2 \log m)$, resulting in $O(r^2 \log m)$ amortized update time.

High probability claim. By choosing $c \geq 2$ in Lemma 4, each update is guaranteed to succeed with probability at least $1 - 1/m^c$. Since there are at most m updates, it follows that Algorithm 3 succeeds with probability at least $1 - 1/m^{c-1}$. ◀

Decremental Implementation of Algorithm 2

Our decremental data structure for maintaining a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H (Algorithm 4) is a decremental implementation of Algorithm 2. Recall that, in Algorithm 2, we recurse on the sampled hypergraphs $S_1, \dots, S_{i_{\text{last}}}$ and add the coresets $C_1, \dots, C_{i_{\text{last}}}$ to \tilde{H} . By Lemma 5, $\tilde{H} = \bigcup_{j=1}^{i_{\text{last}}} C_j \cup S_{i_{\text{last}}}$ is a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H .

In Algorithm 4, we decrementally maintain the hypergraphs $S_1, \dots, S_{i_{\text{last}}}$ and the coresets $C_1, \dots, C_{i_{\text{last}}}$. Suppose that hyperedge e has been deleted from H as the most recent update. Below, we discuss how the data structure handles the deletion in all possible scenarios.

- If e does not belong to \tilde{H} , then e can only exist in the sets of sampled hyperedges $S_1, \dots, S_{i_{\text{last}}-1}$. In this case, removing e from all S_i 's does not affect \tilde{H} since the set of sampled hyperedges remains valid, as each hyperedge is sampled independently.
- If e belongs to $S_{i_{\text{last}}}$, then e is present in all sets of sampled hyperedges, i.e., in $S_1, \dots, S_{i_{\text{last}}}$. Note that, although e belongs to \tilde{H} in this case, since it is only present in sampled hypergraphs, its removal does not affect the coresets and can be handled similarly to the previous case.

■ **Algorithm 4** Decremental-Spectral-Sparsify(H, ε).

Input: an m -edge hypergraph $H = (V, E, \mathbf{w})$ and $0 < \varepsilon < 1$
Maintain: a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H

- 1 $i \leftarrow 0$
- 2 $k \leftarrow \lceil \log_{3/4} m \rceil$
- 3 $m^* \leftarrow n^2 / \varepsilon^2 \log^3 m$
- 4 $S_0 \leftarrow H$
- 5 **Procedure** INITIALIZE
- 6 **while** $i \leq k$ and $|E(H_i)| \geq 32cm^*$ **do** /* c is from Lemma 4 */
- 7 $\mathcal{A}_{i+1} \leftarrow$ initialize DECREMENTAL-CORESET-AND-SAMPLE($S_i, \varepsilon/(2k)$)
 /* \mathcal{A}_{i+1} decrementally maintains C_{i+1} and S_{i+1} */
- 8 $\tilde{H} \leftarrow \tilde{H} \cup C_{i+1}$
- 9 $i \leftarrow i + 1$
- 10 $i_{\text{last}} \leftarrow i$
- 11 $\tilde{H} \leftarrow \tilde{H} \cup S_{i_{\text{last}}}$
- 12 **Procedure** DELETE(e)
- 13 Remove e from H and \tilde{H}
- 14 $i \leftarrow 1$
- 15 **while** $i \leq i_{\text{last}}$ **do**
- 16 Pass the deletion of e to \mathcal{A}_i
- 17 **if** e' has been added to C_i due to the deletion of e **then**
- 18 Add e' to \tilde{H}
- 19 Remove e' from S_i
- 20 $e \leftarrow e'$
- 21 $i \leftarrow i + 1$

- If e belongs to a coreset C_i , then it belongs to the sets of sampled hyperedges S_1, \dots, S_{i-1} , and since it is in the coreset C_i of S_{i-1} , it cannot be present in $S_i, \dots, S_{i_{\text{last}}}$. Similar to the previous cases, we can simply remove e from S_1, \dots, S_{i-1} to maintain $C_1 \cup S_1, \dots, C_{i-1} \cup S_{i-1}$. For $C_i \cup S_i$, we need to maintain C_i , for which we use Algorithm 3 as a subroutine to find a hypergraph e' to be added to C_i as the substitution for e . By Algorithm 3, e' belongs to S_{i-1} and thus its addition to C_i does not affect $C_1 \cup S_1, \dots, C_{i-1} \cup S_{i-1}$. On the other hand, e' might be present in $C_{i+1} \cup S_{i+1}, \dots, C_{i_{\text{last}}} \cup S_{i_{\text{last}}}$, and we need to ensure they are still valid after adding e' to C_i . If e' appears in the sets of sampled hyperedges, then similar to the previous cases, simply removing e' from the sparsifiers makes them valid. Therefore, we pass the deletion of e' to the next sparsifiers until we reach $C_j \cup S_j$ containing e' in its coreset C_j . Again, by construction, e' cannot be present in $C_{j+1} \cup S_{j+1}, \dots, C_{i_{\text{last}}} \cup S_{i_{\text{last}}}$, and we need to remove e' from C_j . This procedure is similar to the one we just explained for the removal of e from C_i , except that we now use it to remove e' from C_j .

As explained above, to handle hyperedge deletions in each sparsifier $C_i \cup S_i$, the data structure utilizes Algorithm 3 and then transmits the changes in $C_i \cup S_i$ to $C_{i+1} \cup S_{i+1}$. The key point of our data structure is that it guarantees the transfer of at most *one* hyperedge deletion from one level to the next, thereby ensuring low recourse at each level. This results in at most $i_{\text{last}} = O(\log m)$ hyperedge deletions across all levels following a hyperedge deletion in H . The guarantees of the data structure are stated below.

► **Lemma 8.** *Given a constant $c \geq 3$ and an m -edge n -vertex hypergraph $H = (V, E, w)$ of rank r undergoing hyperedge deletions, Algorithm 4 maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H of size $O(n^2/\varepsilon^2 \log^6 m)$ in $O(r^2 \log^2 m)$ amortized update time with probability at least $1 - 1/m^{c-2}$. Additionally, each deletion in H results in $O(\log m)$ recourse in \tilde{H} .*

Proof. Suppose that a deletion has happened in H .

Correctness. From Lemma 7, each \mathcal{A}_i correctly handles the deletions. The fact that the data structure correctly transfers hyperedge deletions between \mathcal{A}_i 's and thus correctly maintains \tilde{H} follows from the discussion above.

Size of \tilde{H} . The data structure includes $i_{\text{last}} = O(\log m)$ coresets $C_1, \dots, C_{i_{\text{last}}}$ in \tilde{H} . By Lemma 7, for each $1 \leq i \leq i_{\text{last}}$, C_i has a size of $O(n^2/\varepsilon_i^2 \log^3 m)$, where $\varepsilon_i = \varepsilon/(2k)$ and $k = O(\log m)$. As shown in the proof of Lemma 5, the size of $S_{i_{\text{last}}}$ is $O(n^2/\varepsilon^2 \log^3 m)$. It follows that the size of \tilde{H} is

$$O\left(\sum_{i=1}^k n^2/\varepsilon_i^2 \log^3 m + n^2/\varepsilon^2 \log^3 m\right) = O(n^2/\varepsilon^2 \log^6 m).$$

Update time. The data structure initializes and decrementally maintains $i_{\text{last}} = O(\log m)$ data structures of Algorithm 3. By Lemma 7, each data structure takes $O(mr^2 \log m)$ total update time. Also, at each update, the data structure transmits at most one hyperedge from one level to the next, which results in an $O(m \log m)$ total transmission. Thus, the algorithm takes $O(mr^2 \log^2 m)$ total update time, or equivalently, $O(r^2 \log^2 m)$ amortized update time.

High probability claim. From Lemma 7, for any constant $c \geq 2$, each \mathcal{A}_i correctly maintains its coreset and the set of sampled hyperedges with probability at least $1 - 1/m^{c-1}$. Since $1 \leq i \leq \log m$, by choosing $c \geq 3$, the claim follows.

Recourse bound. As explained before, every hyperedge deletion from H translates to at most one hyperedge deletion or addition in \mathcal{A}_i for $1 \leq i \leq i_{\text{last}}$. Since $i_{\text{last}} = O(\log m)$, it follows that the total number of hyperedge changes in \tilde{H} is $O(\log m)$. ◀

3.2.2 Fully Dynamic Spectral Sparsifier

With a decremental data structure (Algorithm 4) in hand, we now obtain the fully dynamic data structure. We begin by explaining the reduction from the fully dynamic to the decremental data structure and by proving Lemma 6. We then prove Theorem 1.

Reduction from Fully Dynamic to Decremental

We explain how to use the decremental data structure to design a fully dynamic data structure, as described in Algorithm 5. In a nutshell, the data structure uses the batching technique; it maintains a batch of $(1 \pm \varepsilon)$ -spectral hypersparsifiers $\tilde{H}_1, \dots, \tilde{H}_k$, each of which decrementally maintains sub-hypergraphs H_1, \dots, H_k , respectively, such that these sub-hypergraphs partition H . The union $\tilde{H} = \bigcup_{i=1}^k \tilde{H}_i$, by decomposability (Lemma 3), is a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H .

More specifically, the data structure starts with an empty H and empty H_1, \dots, H_k and ensures that each H_i contains at most 2^i hyperedges at all time (which we call the size constraint of H_i). Intuitively, a hyperedge e inserted in H is placed in H_1 as long as the size constraint of H_1 is not violated. If there are already two hyperedges in H_1 , the data structure moves *all* hyperedges of H_1 (along with e) to the empty H_2 . Note that H_2 can contain at most four hyperedges, and so its size constraint is not violated. However, if there

Algorithm 5 Fully-Dynamic-Spectral-Sparsify(H, ε).

Input: an empty n -vertex hypergraph $H = (V, E, \mathbf{w})$ with $|E| \leq m$ at any time
Maintain: a $(1 \pm \varepsilon)$ -spectral hypersparsifier \tilde{H} of H

- 1 **Procedure** INITIALIZE
- 2 **foreach** $1 \leq i \leq \log m$ **do**
- 3 $H_i, \tilde{H}_i \leftarrow (V, \emptyset, \mathbf{0})$
- 4 $t \leftarrow 0$
- 5 $i_{\text{last}} \leftarrow 1$
- 6 **Procedure** ADD(e)
- 7 $t \leftarrow t + 1$
- 8 $j \leftarrow \max\{i \mid t \text{ is divisible by } 2^{i-1}\}$
- 9 $i_{\text{last}} \leftarrow \max\{j, i_{\text{last}}\}$
- 10 $H_j \leftarrow H_j \cup H_{j-1} \cup \dots \cup H_1$
- 11 Add e to H_j
- 12 **foreach** $1 \leq i \leq j - 1$ **do**
- 13 $H_i, \tilde{H}_i \leftarrow (V, \emptyset, \mathbf{0})$
- 14 $\mathcal{A}_j \leftarrow$ (re)initialize DECREMENTAL-SPECTRAL-SPARSIFY(H_j, ε)
 /* \mathcal{A}_j decrementally maintains the *newly* computed \tilde{H}_j */
- 15 **Procedure** DELETE(e)
- 16 $j \leftarrow$ index of the specific hypergraph among H_1, \dots, H_n that contains e
- 17 Pass the deletion of e to \mathcal{A}_j
- 18 **Procedure** OUTPUTSPARSIFIER()
- 19 **return** $\tilde{H} = \bigcup_{i=1}^{i_{\text{last}}} \tilde{H}_i$

were already two hyperedges in H_2 , then the addition of three more hyperedges (from H_1 plus e) would violate its size constraint. In this case, the data structure would move the hyperedges of H_1 and H_2 plus e to H_3 , and so on.

To regularize the insertion process, we initialize a counter t as a binary sequence of $\log k$ zeros. After each insertion, the data structure increments t (in the binary format) by one and finds the highest index i whose bit flipped due to the increment. The data structure then moves the hyperedges in H_1, \dots, H_{i-1} , along with the inserted hyperedge e , to H_i and reinitializes the decremental data structure with the new H_i .

The deletion process can be done as before; since each hyperedge is associated with exactly one sub-hypergraph H_i , its deletion would be easily handled by passing it to the decremental data structure maintaining \tilde{H}_i .

Proof of Lemma 6. Suppose that the decremental data structure maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier of any hypergraph with m' initial hyperedges in $T(m', n, \varepsilon^{-1})$ amortized update time and of size $S(m', n, \varepsilon^{-1})$ with probability at least $1 - 1/\text{poly}(m')$.

Correctness. From the discussion above, the sub-hypergraphs H_1, \dots, H_k correctly partition the hyperedges of H . By Lemma 8, the sparsifiers $\tilde{H}_1, \dots, \tilde{H}_k$ are correctly maintained. It follows from Lemma 3 that $\tilde{H} = \bigcup_{i=1}^k \tilde{H}_i$ is a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H .

Size of \tilde{H} . By assumption, each \tilde{H}_i has size $S(m_i, n, \varepsilon^{-1})$, which is upper bounded by $S(m, n, \varepsilon^{-1})$ since $m_i \leq m$ and S is a monotone non-decreasing function. Since $k = O(\log m)$, the size of \tilde{H} is bounded by $\sum_{i=1}^{\log m} S(m_i, n, \varepsilon^{-1}) = O(S(m, n, \varepsilon^{-1}) \log m)$.

Update time. After l insertions, H_i has been reinitialized for at most $l/2^i$ times. This is because H_i can contain at most $m_i = 2^i$ hyperedges. By assumption, the total time for the initialization and maintenance of \mathcal{A}_i is bounded by $m_i T(m_i, n, \varepsilon^{-1}) = 2^i T(m_i, n, \varepsilon^{-1})$, which is bounded by $2^i T(m, n, \varepsilon^{-1})$ since $m_i \leq m$ and S is a monotone non-decreasing function. Thus, the total time for the initialization and maintenance of \mathcal{A}_i throughout l insertions is $O(\frac{l}{2^i} 2^i T(m, n, \varepsilon^{-1})) = O(lT(m, n, \varepsilon^{-1}))$. Therefore, the total update time for maintaining $\tilde{H}_1, \dots, \tilde{H}_k$ is

$$O\left(\sum_{i=1}^{\log m} lT(m, n, \varepsilon^{-1})\right) = O(lT(m, n, \varepsilon^{-1}) \log m),$$

resulting in an $O(T(m, n, \varepsilon^{-1}) \log m)$ amortized update time.

High probability claim. After l insertions, each H_i is reinitialized $O(l)$ times. By Lemma 8, the probability of failure is at most l/m^{c-2} , where $c \geq 3$. Since $k = \log m$, it follows that the probability of failure for maintaining $\tilde{H}_1, \dots, \tilde{H}_k$ is at most $l \log m / m^{c-2}$. Since $l = \text{poly}(m)$, the high probability claim follows by choosing c to be large enough. ◀

Proof of Theorem 1. Given an m -edge n -vertex hypergraph H of rank r , by Lemma 8, the decremental data structure (Algorithm 4) maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H of size $S(m', n, \varepsilon^{-1}) = O(n^2/\varepsilon^2 \log^6 m)$ in $T(m, n, \varepsilon^{-1}) = O(r^2 \log^2 m)$ amortized update time.

Since S and T are monotone non-decreasing functions, it follows from Lemma 6 that, for any hypergraph H containing at most m hyperedges at any point, the fully dynamic data structure (Algorithm 5) maintains a $(1 \pm \varepsilon)$ -spectral hypersparsifier of H of size $O(n^2/\varepsilon^2 \log^7 m)$ in $O(r^2 \log^3 m)$ amortized update time. ◀

References

- 1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *57th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 335–344, 2016. doi:10.1109/FOCS.2016.44.
- 2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and José D. P. Rolim, editors, *16th Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX)*, volume 8096 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2013. doi:10.1007/978-3-642-40328-6_1.
- 3 Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. In *62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 528–539, 2021. doi:10.1109/FOCS52979.2021.00059.
- 4 Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 910–928, 2019. doi:10.1109/FOCS.2019.00059.
- 5 Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012. doi:10.1137/090772873.
- 6 Joshua D. Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM*, 56(8):87–94, 2013. doi:10.1145/2492007.2492029.
- 7 András A. Benczúr and David R. Karger. Approximating s - t minimum cuts in $O(n^2)$ time. In *28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 47–55, 1996. doi:10.1145/237814.237827.

- 8 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 *49th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 229, pages 20:1–20:20, 2022. doi:10.4230/LIPIcs.ICALP.2022.20.
- 9 Erik G. Boman, Bruce Hendrickson, and Stephen A. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM J. Numer. Anal.*, 46(6):3264–3284, 2008. doi:10.1137/040611781.
- 10 Yuri Boykov and Olga Veksler. Graph cuts in vision and graphics: Theories and applications. In *Handbook of Mathematical Models in Computer Vision*, pages 79–96. 2006. doi:10.1007/0-387-28831-7_5.
- 11 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1135–1146, 2020. doi:10.1109/FOCS46700.2020.00109.
- 12 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952. URL: <http://www.jstor.org/stable/2236576>.
- 13 Michael B. Cohen, Cameron Musco, and Jakub Pachocki. Online row sampling. *Theory Comput.*, 16:1–25, 2020. doi:10.4086/TOC.2020.V016A015.
- 14 Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 451–460, 2008. doi:10.1145/1374376.1374441.
- 15 Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In *33rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 124–153, 2022. doi:10.1137/1.9781611977073.7.
- 16 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In *51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 914–925, 2019. doi:10.1145/3313276.3316379.
- 17 Zhuo Feng. Similarity-aware spectral sparsification by edge filtering. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018. doi:10.1145/3195970.3196114.
- 18 Sebastian Forster, Gramoz Goranci, Yang P. Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. Minor sparsifiers and the distributed laplacian paradigm. In *62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 989–999, 2021. doi:10.1109/FOCS52979.2021.00099.
- 19 Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 516–527, 2021. doi:10.1109/FOCS52979.2021.00058.
- 20 Mohsen Ghaffari and Jaehyun Koo. Parallel batch-dynamic algorithms for spanners, and extensions. In *37th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 299–313. ACM, 2025. doi:10.1145/3694906.3743322.
- 21 Gramoz Goranci, Monika Henzinger, and Pan Peng. Dynamic effective resistances and approximate schur complement on separable graphs. In *26th European Symposium on Algorithms (ESA)*, volume 112, pages 40:1–40:15, 2018. doi:10.4230/LIPIcs.ESA.2018.40.
- 22 Gramoz Goranci and Ali Momeni. Fully dynamic spectral sparsification of hypergraphs. *CoRR*, abs/2502.01421, 2025. doi:10.48550/arXiv.2502.01421.
- 23 Arun Jambulapati, Yang P. Liu, and Aaron Sidford. Chaining, group leverage score overestimates, and fast spectral hypergraph sparsification. In *55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 196–206, 2023. doi:10.1145/3564246.3585136.

- 24 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)*, pages 1159–1170, 2021. doi:10.1109/FOCS52979.2021.00114.
- 25 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 598–611, 2021. doi:10.1145/3406325.3451061.
- 26 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017. doi:10.1137/141002281.
- 27 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In Shuchi Chawla, editor, *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1814–1833. SIAM, 2020. doi:10.1137/1.9781611975994.111.
- 28 Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. Dynamic streaming spectral sparsification in nearly linear time and space. *CoRR*, abs/1903.12150, 2019. arXiv:1903.12150.
- 29 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science (ITCS)*, pages 393–398. ACM, 2012. doi:10.1145/2090236.2090267.
- 30 Jonathan A. Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory Comput. Syst.*, 53(2):243–262, 2013. doi:10.1007/S00224-012-9396-1.
- 31 Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *50th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 13–21, 2009. doi:10.1109/FOCS.2009.75.
- 32 Sanjeev Khanna, Huan Li, and Aaron Putterman. Near-optimal linear sketches and fully-dynamic algorithms for hypergraph spectral sparsification. In Michal Koucký and Nikhil Bansal, editors, *57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1190–1200. ACM, 2025. doi:10.1145/3717823.3718239.
- 33 Sanjeev Khanna, Aaron (Louie) Putterman, and Madhu Sudan. Almost-tight bounds on preserving cuts in classes of submodular hypergraphs. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 297 of *LIPICs*, pages 98:1–98:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ICALP.2024.98.
- 34 Pushmeet Kohli and Philip H. S. Torr. Dynamic graph cuts and their applications in computer vision. In *Computer Vision: Detection, Recognition and Reconstruction*, volume 285 of *Studies in Computational Intelligence*, pages 51–108. 2010. doi:10.1007/978-3-642-12848-6_3.
- 35 Ioannis Koutis, Alex Levin, and Richard Peng. Faster spectral sparsification and numerical algorithms for SDD matrices. *ACM Trans. Algorithms*, 12(2):17:1–17:16, 2016. doi:10.1145/2743021.
- 36 Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Comput. Vis. Image Underst.*, 115(12):1638–1646, 2011. doi:10.1016/J.CVIU.2011.05.013.
- 37 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. doi:10.1145/2948062.
- 38 James R. Lee. Spectral hypergraph sparsification via chaining. In *55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 207–218, 2023. doi:10.1145/3564246.3585165.
- 39 James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM*, 61(6):1–30, 2014. doi:10.1145/2665063.
- 40 Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 678–687. ACM, 2017. doi:10.1145/3055399.3055477.

- 41 Huan Li and Aaron Schild. Spectral subspace sparsification. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 385–396. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00044.
- 42 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *51st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 245–254, 2010. doi:10.1109/FOCS.2010.30.
- 43 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.
- 44 Kazusato Oko, Shinsaku Sakaue, and Shin-ichi Tanigawa. Nearly tight spectral sparsification of directed hypergraphs. In *50th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 94:1–94:19, 2023. doi:10.4230/LIPIcs.ICALP.2023.94.
- 45 Richard Peng. Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time. In *27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1862–1867, 2016. doi:10.1137/1.9781611974331.CH130.
- 46 Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! *SIAM Journal on Computing*, 46(2):710–743, 2017. doi:10.1137/15m1047209.
- 47 Akbar Rafiey and Yuichi Yoshida. Sparsification of decomposable submodular functions. In *36th Conference on Artificial Intelligence (AAAI)*, pages 10336–10344. AAAI Press, 2022. doi:10.1609/AAAI.V36I9.21275.
- 48 Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 263–269, 2013. doi:10.1109/FOCS.2013.36.
- 49 Tasuku Soma, Kam Chuen Tung, and Yuichi Yoshida. Online algorithms for spectral hypergraph sparsification. In Jens Vygen and Jaroslav Byrka, editors, *25th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 14679 of *Lecture Notes in Computer Science*, pages 405–417. Springer, 2024. doi:10.1007/978-3-031-59835-7_30.
- 50 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In Timothy M. Chan, editor, *30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2570–2581. SIAM, 2019. doi:10.1137/1.9781611975482.159.
- 51 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *40th Annual ACM Symposium on Theory of Computing (STOC)*, 2008. doi:10.1145/1374376.1374456.
- 52 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. doi:10.1137/08074489X.
- 53 Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Anal. Appl.*, 35(3):835–885, 2014. doi:10.1137/090771430.
- 54 Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 543–556, 2022. doi:10.1145/3519935.3520068.
- 55 Yibin Zhao. Fully dynamic spectral and cut sparsifiers for directed graphs. *CoRR*, 2025. doi:10.48550/arXiv.2507.19632.
- 56 Chunjiang Zhu, Qinqing Liu, and Jinbo Bi. Spectral vertex sparsifiers and pair-wise spanners over distributed graphs. In *38th International Conference on Machine Learning (ICML)*, volume 139, pages 12890–12900, 2021. URL: <https://proceedings.mlr.press/v139/zhu21c.html>.
- 57 Zeyuan Allen Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 237–245. ACM, 2015. doi:10.1145/2746539.2746610.