Vertex Sparsifiers for Hyperedge Connectivity

Han Jiang
University of Michigan, Ann Arbor, MI, USA

Shang-En Huang
University of Michigan, Ann Arbor, MI, USA

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Tian Zhang
University of Michigan, Ann Arbor, MI, USA

Abstract

Recently, Chalermsook et al. [SODA'21] introduces a notion of *vertex sparsifiers for c-edge connectivity,* which has found applications in parameterized algorithms for network design and also led to exciting dynamic algorithms for c-edge st-connectivity [Jin and Sun FOCS'22].

We study a natural extension called *vertex sparsifiers for c-hyperedge connectivity* and construct a sparsifier whose size matches the state-of-the-art for normal graphs. More specifically, we show that, given a hypergraph $G = (V, E)$ with n vertices and m hyperedges with k terminal vertices and a parameter c, there exists a hypergraph H containing only $O(kc^3)$ hyperedges that preserves all minimum cuts (up to value c) between all subset of terminals. This matches the best bound of $O(kc^3)$ edges for normal graphs by [Liu'20]. Moreover, H can be constructed in almost-linear $O(p^{1+o(1)} + n(rc \log n)^{O(rc)} \log m)$ time where $r = \max_{e \in E} |e|$ is the rank of G and $p = \sum_{e \in E} |e|$ is the total size of G, or in $\text{poly}(m, n)$ time if we slightly relax the size to $O(kc^3 \log^{1.5}(kc))$ hyperedges.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Vertex sparsifier, hypergraph, connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.70

Related Version *Full Version*: `https://arxiv.org/abs/2207.04115`

Acknowledgements We thank Sorrachai Yingchareonthawornchai, Yang Liu, Yunbum Kook, and Richard Peng for the discussion that inspires the notion of the pruned auxiliary graph in this paper. We also thank anonymous reviewers for their valuable comments.

1 Introduction

Graph sparsification has played a central role in graph algorithm research in the last two decades. Prominent examples include spanners [1], cut sparsifiers [3], and spectral sparsifiers [22]. Recently, there has been significant effort in generalizing the graph sparsification results to hypergraphs. For cut sparsifiers, Kogan and Krauthgamer [13] generalized the Benczúr and Karger's cut sparsifiers [3] by showing that, given any hypergraph $G = (V, E)$ with n vertices, there is a $(1+\epsilon)$-approximate cut sparsifier H containing $\tilde{O}(nr/\epsilon^2)$ hyperedges where $r = \max_{e \in E} |e|$ denotes the *rank* of the hypergraph. After some follow-up work [5, 2], Chen, Khanna, and Nagda [6] finally improved the sparsifier size to $\tilde{O}(n/\epsilon^2)$ hyperedges, matching the optimal bound for normal graphs. Another beautiful line of work generalizes Speilman and Teng's spectral sparsifiers [22] to hypergraphs [2, 21, 10] and very recently results in spectral sparsifiers with $\tilde{O}(n/\text{poly}(\epsilon))$ hyperedges [11]. We also mention that the classical sparse connectivity certificates by Nagamochi and Ibaraki [19] were also generalized to hypergraphs by Chekuri and Xu [5].

This paper studies a graph sparsification problem recently introduced by Chalermsook et al. [4] called *vertex sparsifiers for c-edge connectivity*. It is closely related to the vertex sparsifiers for edge cuts [15, 12] and vertex cuts [14]. In this problem, we are given an

unweighted undirected graph $G = (V, E)$ and a set of terminals $\mathcal{T} \subseteq V$. For any disjoint subsets $A, B \subseteq \mathcal{T}$, let $\text{mincut}_G(A, B)$ denote the size of a minimum (edge-)cut that disconnects A and B. Now, a graph $H = (V_H, E_H)$ with $\mathcal{T} \subseteq V_H$ is a $(\mathcal{T}, c)$-*sparsifier* of G if for any disjoint subsets $A, B \subseteq \mathcal{T}$, $\min\{c, \text{mincut}_G(A, B)\} = \min\{c, \text{mincut}_H(A, B)\}$. Basically, H preserves all minimum cut structures between the terminals $\mathcal{T}$ up to the value c. This notion of graph sparsifiers has found interesting applications in offline dynamic algorithms and network design problems [4]. Moreover, the very recent breakthrough on dynamic c-edge st-connectivity by Jin and Sun [9] is also crucially based on dynamic algorithms for maintaining $(\mathcal{T}, c)$-sparsifiers.

In the original paper by [4], they showed that, for any graph $G = (V, E)$ and terminal set $\mathcal{T}$ of size k, there exists a $(\mathcal{T}, c)$-sparsifier containing $O(kc^4)$ edges (which can be constructed in $O(m(c \log n)^{O(c)})$ time) and also showed fast algorithms for constructing $(\mathcal{T}, c)$-sparsifiers of size $k \cdot O(c)^{2c}$ in $mc^{O(c)} \log^{O(1)} n$ time. Then, Liu [16] improved the size bound to $O(kc^3)$ together with polynomial-time algorithms (no exponential dependency on c) for constructing $(\mathcal{T}, c)$-sparsifiers with $O(kc^3 \log^{1.5} n)$ edges.

A natural question is then whether these results can be extended to hypergraphs. The notion of $(\mathcal{T}, c)$-sparsifiers itself can be naturally extended to hypergraphs by allowing G and H to be hypergraphs and letting $\text{mincut}_G(A, B)$ denote the value of the minimum hyperedge-cut instead. However, it is conceivable that there might not exist a $(\mathcal{T}, c)$-sparsifier with $\text{poly}(k, c)$. This bound might require bad dependency on the rank r, for example.

In this paper, we show that the state-of-the-art for normal graphs indeed extend to hypergraphs and we can even slightly improve the bounds:

▶ **Theorem 1.** *Let $G = (V, E)$ be a hypergraph with n vertices, m hyperedges, rank r and total size p. Let $\mathcal{T} \subseteq V$ be the set of k terminals. There are algorithms for computing the following:*

1. *a $(\mathcal{T}, c)$-sparsifier H of G with $O(kc^3)$ hyperedges in $O(p^{1+o(1)} + n(rc \log n)^{O(rc)} \log m)$ time, which is almost-linear in the input size when both r and $c = O(1)$, and*
2. *a $(\mathcal{T}, c)$-sparsifier H of G with $O(kc^3 \log^{1.5}(kc))$ hyperedges in $\text{poly}(m, n)$ time.*

The first result matches the best known bound of $O(kc^3)$ edges for normal graphs [16]. When $r = O(1)$, the first time bound slightly improves the $O(m(c \log n)^{O(c)})$ bound of [4] for normal graphs. The second result removes the exponential dependency on r and c after relaxing the size by a $\log^{1.5}(kc)$ factor. The number of hyperedges in our sparsifier is completely independent from n, while the polynomial time algorithm by Liu [16] gives the size of $O(kc^3 \log^{1.5} n)$. So this implies the first polynomial time construction of sparsifiers of size near-linear in k and independent of n, even for normal graphs.

Open Problems. Can we construct vertex sparsifiers for c-hyperedge connectivity of $k \cdot \text{poly}(c)$ size in near-linear time even when the rank is unbounded? This is a prerequisite to near-linear time algorithms for computing vertex sparsifiers for *c-vertex connectivity* of $k \cdot \text{poly}(c)$ size. Such a result might lead to dynamic c-vertex st-connectivity algorithm similar to the previous development where a near-linear time construction of vertex sparsifiers for c-edge connectivity leads to a dynamic algorithm for c-edge st-connectivity [9]. As dynamic c-vertex st-connectivity is one of the major open problems in dynamic graph algorithms (known solutions only works for very small $c \leq 3$ [7, 8, 20]), we view this work as a stepping stone towards this goal.

1.1 Technical Challenges

There are two main obstacles that prevent us extending the results of [16, 4] directly from normal graphs to hypergraphs. First, if we follow the divide and conquer framework of Chalermsook et al. [4] in a straightforward way, then we would end up with a much larger $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|(rc)^3)$ hyperedges. This is because, in [4], all vertices incident to the boundary edges are declared as new terminal vertices in the recursion. However in our case, each hyperedge may contain r vertices and this yields the dependency of r. To handle this issue, we instead introduce only two *anchor vertices* for each boundary hyperedge. Our divide and conquer framework requires slightly more careful analysis, but this naturally gives a $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges.

The second obstacle is the near-linear time algorithm, Part (1) of Theorem 1. Chalermsook et al. [4] introduced *auxiliary graphs* and apply the ϕ-Sparsify procedure on it to identify all *essential* hyperedges, which roughly are hyperedges that will be kept in the sparsifier. However, there is a subtle small gap in [4]: their ϕ-Sparsify procedure could erroneously identify non-essential hyperedges as essential hyperedges. This is explained in more detail in the full version of the paper. This bug results in a much larger $(\mathcal{T}, c)$-sparsifier. In this paper we fix the bug by (1) introducing a notion of *useful* partitions of the terminal set and (2) providing an efficient algorithm that discards all non-useful partitions from the auxiliary graph. Then, we show that, after our modification, this approach indeed gives a small $(\mathcal{T}, c)$-sparsifier as desired.

1.2 Organization

In Section 2 we review some basic definitions of hypergraphs. In Section 3 we define contraction based $(\mathcal{T}, c)$-sparsifiers and introduce the divide and conquer framework. In Section 4 we show the existence of a $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges. In Section 5 we give a near-linear-time algorithm that computes a $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges, proving Part (1) of Theorem 1. For Part (2) of Theorem 1 we refer readers to the full version of the paper.

2 Preliminary

Let $G = (V, E)$ be a hypergraph. V is the set of vertices and E is a multiset of hyperedges with each hyperedge e being a subset of V. The *rank* $r \coloneqq \max_{e \in E} |e|$ of a hypergraph is the size of the largest hyperedge, and the *total size* $p \coloneqq \sum_{e \in E} |e|$ is the sum of all edge sizes.

For any two disjoint sets of vertices $A, B \subseteq V$, let $E_G(A, B)$ denote the set of hyperedges with at least one endpoint in A and at least one endpoint in B. For any set of vertices $X \subseteq V$, we denote the *boundary* of X of the graph G by $\partial_G X \coloneqq E_G(X, V \setminus X)$. If the context is clear then we will omit the graph G and write ∂X instead.

Restrictions and Induced Sub-Hypergraphs. Let $\mathcal{T} \subseteq V \cup E$ be a mixed multiset of vertices and hyperedges, for any set of vertices $X \subseteq V$, we define the *restriction* of the multiset $\mathcal{T}$ on X to be $\mathcal{T}|_X = (\mathcal{T} \cap X) \cup \{e \cap X \mid e \in (\mathcal{T} \cap E) \text{ and } e \cap X \neq \emptyset\}$. The *induced sub-hypergraph* $G[X]$ is then defined over the vertex set X with the restriction of all hyperedges $E|_X$, that is, $G[X] \coloneqq (X, E|_X)$.

Incident Edges and Vertices. For any set of vertices $X \subseteq V$, define $E(X)$ to be the set of all hyperedges that incident to at least one vertex in X. For any set of hyperedges $Y \subseteq E$, define $V(Y) = \bigcup_{y \in Y} y$ to be the set of vertices incident to hyperedges in Y. Similarly, for any mixed set of vertices and hyperedges $\mathcal{T} \subseteq V \cup E$ we define $V(\mathcal{T}) = (\mathcal{T} \cap V) \cup V(\mathcal{T} \cap E)$ to be the set of all vertices that are in the set or incident to any hyperedge in the set.

3 Structural Properties on Hypergraphs

In this section, we explore more structural properties on hypergraphs. In particular, we introduce *anchored separated hyperedges*, and describe useful properties in a divide and conquer framework that leads to a construction of $(\mathcal{T}, c)$-sparsifiers.

3.1 Cuts in Hypergraphs

Let u and v be two elements in V. We say that u and v are *connected* in a hypergraph G, if there is a path connecting u and v. Let $A, B \subseteq V$ be two disjoint sets of vertices. A and B are *disconnected* if for any $a \in A$ and $b \in B$, a and b are not connected.

▶ **Definition 2** (Cuts and Minimum Cuts). *A* cut *is a bipartition $(X, V \setminus X)$ of vertices. The* value *of the cut is $|\partial X| = |E_G(X, V \setminus X)|$. For any disjoint subsets $A, B \subseteq V$, if $A \subseteq X$ and $B \subseteq (V \setminus X)$ then we say that $(X, V \setminus X)$ is an (A, B)-cut. A* minimum (A, B)-cut *or* (A, B)-mincut *is any (A, B)-cut with minimum value. Its value is denoted as* $\text{mincut}_G(A, B)$. *Given a parameter c, a* c-thresholded (A, B)-mincut cut value *is defined as*

$$\text{mincut}^c_G(A, B) \coloneqq \min(\text{mincut}_G(A, B), c).$$

3.2 $(\mathcal{T}, c)$-Equivalency and $(\mathcal{T}, c)$-Sparsifiers

Our vertex sparsifier algorithms are based on identifying a set of hyperedges and contract them. Given a hypergraph $G = (V, E)$ and a hyperedge $e \in E$, the *contracted hypergraph* G/e is defined by identifying all incident vertices $V(e)$ as one vertex, and then remove e itself from the graph. For any set of terminals $T \subseteq V$, the effect of contracting an hyperedge e is denoted as T/e. Similarly, for any set $\hat{E} \subseteq E$, we denote $G/\hat{E}$ the hypergraph obtained from G by contracting all hyperedges in $\hat{E}$ (notice that all hyperedges in $\hat{E}$ are removed after the contraction.)

▶ **Definition 3** ($(\mathcal{T}, c)$-Sparsifiers). *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two hypergraphs. Let $\mathcal{T} \subseteq V_G$ be the set of terminals. We say H is a* contraction based $(\mathcal{T}, c)$-sparsifier *of G, if there exists a surjective (onto) projection $\pi : V_G \to V_H$, such that for any $e \in E_H$ there is an edge $f \in E_G$ such that $\pi(f) = \cup_{v \in f}\{\pi(v)\} = e$, and for any two subsets $T_1, T_2 \subseteq \mathcal{T}$,*

$$\text{mincut}^c_G(T_1, T_2) = \text{mincut}^c_H(\pi(T_1), \pi(T_2)).$$

Furthermore, if the terminals are not affected by the projection, i.e., $\pi(\mathcal{T}) = \mathcal{T}$, then we say that G and H are $(\mathcal{T}, c)$-equivalent.

Remark. A more general $(\mathcal{T}, c)$-sparsifier would allow an arbitrary mapping π on both vertices and edges. However, we note that all $(\mathcal{T}, c)$-sparsifiers constructed in this paper are always contraction based. Therefore, for the ease of the presentation we will omit the term "contraction based" when we mention $(\mathcal{T}, c)$-sparsifiers.

For the ease of the reading, we define the following set operations that allow us to add/remove hyperedges of G into a sparsifier H.

▶ **Definition 4.** *Let $G = (V, E)$ be a hypergraph. For any multiset X of hyperedges over the vertices V, and any contraction based $(\mathcal{T}, c)$-sparsifier H with the projection π, define*

- (Adding contracted hyperedges) $H \cup X := H \cup \pi(X)$, *and*
- (Removing contracted hyperedges) $H - X := H - \pi(X)$.

3.3 $(\mathcal{T}, c)$-Sparsifiers from a Divide and Conquer Framework

Another important concept to our contraction based $(\mathcal{T}, c)$-sparsifier construction is that we apply a divide and conquer framework to G. Let $G = (V, E)$ be a hypergraph and let (V_1, V_2) be a bipartition of vertices. We note that our divide and conquer framework is slightly different than just recurse on the induced sub-hypergraphs $G[V_1]$ and $G[V_2]$. In particular, for each *separated hyperedge* e we add two new *anchor vertices* to e, ended up slightly increasing the size of the vertex set in the next-level recursion.

Separated Hyperedges and Anchor Vertices. Let $e \in E(V_1, V_2)$ be a hyperedge across the bipartition. The *separated hyperedges* of e with respect to this bipartition (V_1, V_2) is the set composed of hyperedges of e restricted on both V_1 and V_2. The *anchored separated hyperedges* are separated hyperedges with additional *anchor vertices*: let $e_1 = e|_{V_1}$ and $e_2 = e|_{V_2}$ be the separated hyperedges of e, then we introduce four new anchored vertices $v_{e,1}, v_{e,2}, v_{e,3}$, and $v_{e,4}$ and define $\hat{e}_1 := e_1 \cup \{v_{e,1}, v_{e,2}\}$ and $\hat{e}_2 := e_2 \cup \{v_{e,3}, v_{e,4}\}$. Let $S = E(V_1, V_2)$ be the set of crossing hyperedges, in this paper the set of anchored separated hyperedges respect to bipartition (V_1, V_2) are denoted by $Sep(S, V_1, V_2) := \{\hat{e}_1, \hat{e}_2 \mid e \in S\}$.

Let $A_1 = \{v_{e,1}, v_{e,2} \mid e \in E(V_1, V_2)\}$ and let $A_2 = \{v_{e,3}, v_{e,4} \mid e \in E(V_1, V_2)\}$ be the set of newly introduced anchor vertices. These anchor vertices will be added to the terminal set in order to correctly preserve the mincut values. That is, the terminal sets defined for the subproblems are $\mathcal{T}_1 := \mathcal{T}|_{V_1} \cup A_1$ and $\mathcal{T}_2 := \mathcal{T}|_{V_2} \cup A_2$. Now, we define the anchored induced sub-hypergraphs, which are useful when applying the divide and conquer framework.

▶ **Definition 5** (Anchored Induced Sub-Hypergraphs). *Let $G = (V, E)$ be a hypergraph and $V_1 \subseteq V$ be a subset of vertices. Define $V_2 = V \setminus V_1$, $G^{\mathsf{sep}} = G \cup Sep(E(V_1, V_2), V_1, V_2) - E(V_1, V_2)$, and the set of anchored vertices to be $A_1 \cup A_2$. Then, the* anchored induced sub-hyperegraph *for V_1 is defined as $\hat{G}[V_1] := G^{\mathsf{sep}}|_{V_1 \cup A_1}$.*

The Divide and Conquer Framework. The most generic divide and conquer method works as the follows. First, a bipartition (V_1, V_2) are determined. Then, the algorithm performs recursion on the anchored induced sub-hypergraphs $\hat{G}[V_1]$ and $\hat{G}[V_2]$ with terminal sets $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. After obtaining the $(\mathcal{T}_1, c)$-sparsifier and $(\mathcal{T}_2, c)$-sparsifier from the subproblems, the algorithm combines them by replacing the anchored separated hyperedges with the original hyperedges.

Algorithm 1 A Divide and Conquer Framework.

Input: Hypergraph G, terminal set $\mathcal{T}$, bipartition (V_1, V_2) of vertices, parameter c.
Output: A $(\mathcal{T}, c)$-sparsifier H for G.

1 (**Divide**) Construct subproblems $(\hat{G}[V_1], \mathcal{T}_1)$ and $(\hat{G}[V_2], \mathcal{T}_2)$.
2 (**Conquer**) For $i \in \{1, 2\}$, obtain H_i, a $(\mathcal{T}_i, c)$-sparsifier of $\hat{G}[V_i]$.
3 (**Combine**) Return $H := H_1 \cup H_2 \cup E(V_1, V_2) - Sep(E(V_1, V_2), V_1, V_2)$.

We summarize the divide and conquer framework in Algorithm 1. The following lemma states the correctness of the framework.

▶ **Lemma 6.** *H returned from Algorithm 1 is a $(\mathcal{T}, c)$-sparsifier.*

Proof. Let $\pi_1 : V_1 \cup A_1 \to V_{H_1}$ and $\pi_2 : V_2 \cup A_2 \to V_{H_2}$ be the projection maps on H_1 and H_2 respectively. Since $V_1 \cup A_1$ and $V_2 \cup A_2$ are disjoint, it is natural to define $\pi : V \to V_H$ by simply combining both maps where $\pi(v) = \pi_1(v)$ if $v \in V_1$, and $\pi(v) = \pi_2(v)$ if $v \in V_2$.

Now, it suffices to show that for any two disjoint subsets $A, B \subseteq \mathcal{T}$, we have $\text{mincut}^c_G(A, B) = \text{mincut}^c_H(\pi(A), \pi(B))$.

Part 1. We first show that $\text{mincut}^c_G(A, B) \geq \text{mincut}^c_H(\pi(A), \pi(B))$. Let $(X, V \setminus X)$ be a minimum (A, B)-cut on G with size $|\partial X| \leq c$. Intuitively, we will construct the cuts in the subproblems $\hat{G}[V_1]$ and $\hat{G}[V_2]$ using $(X, V \setminus X)$. Then we will argue that the preserved mincuts in $\hat{G}[V_1]$ and $\hat{G}[V_2]$ can be merged back, proving that there is a $(\pi(A), \pi(B))$-mincut in H with size no larger than $|\partial X|$.

Let $S = E_G(V_1, V_2)$ be the set of hyperedges across the bipartition in the divide and conquer framework, and let $\hat{V} = V \cup A_1 \cup A_2$ be the vertex set in G^{sep}. We define the set of vertices X^{sep} that contains X and all newly created anchor vertices that belongs to the X side: for any $e \in S$, we add $\{v_{e,1}, v_{e,2}, v_{e,3}, v_{e,4}\}$ to X^{sep} if $e \subseteq X$ (the hyperedge is fully in the X side). We add $\{v_{e,1}, v_{e,3}\}$ to X^{sep} if $e \in S$. We add nothing if $e \subseteq V \setminus X$.

Now, we have $\partial X^{\text{sep}} = (\partial X) \cup Sep((\partial X) \cap S, V_1, V_2) - (\partial X) \cap S$. Moreover, $(X^{\text{sep}}, \hat{V} \setminus X^{\text{sep}})$ is an $(A^{\text{ext}}, B^{\text{ext}})$-cut in G^{sep} of size $|\partial X| + |(\partial X) \cap S|$, where

$$\begin{cases} A^{\text{ext}} := A \cup \{v_{e,1}, v_{e,3} \mid e \in ((\partial X) \cap S)\}, \text{ and} \\ B^{\text{ext}} := B \cup \{v_{e,2}, v_{e,4} \mid e \in ((\partial X) \cap S)\}. \end{cases}$$

Intuitively, by carefully extend the pair (A, B) to a larger pair $(A^{\text{ext}}, B^{\text{ext}})$ we ensure that *all* separated hyperedges $Sep((\partial X) \cap S, V_1, V_2)$ appear in every $(A^{\text{ext}}, B^{\text{ext}})$-mincut on G^{sep}. See Figure 1.

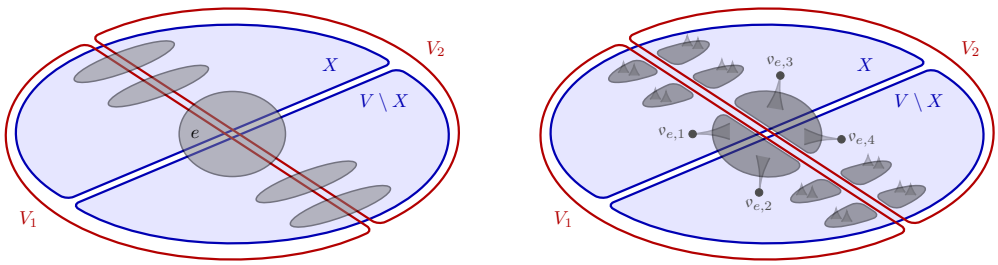

Figure 1 An illustration to the proof of Lemma 6. The gray circles represent hyperedges that cross the bipartition (V_1, V_2) in the divide and conquer framework. When these hyperedges are separated, new anchor vertices are introduced and added to the terminal sets. The newly created terminal vertices are forced to join different sides of the cut, if and only if the separated hyperedge crosses the (A, B)-mincut $(X, V \setminus X)$.

Suppose H_1 is a $(\mathcal{T}_1, c)$-sparsifier of $\hat{G}[V_1]$ and H_2 is a $(\mathcal{T}_2, c)$-sparsifier of $\hat{G}[V_2]$ obtained from the conquer step (Algorithm 1). Let $(Y_1, \pi(V_1 \cup A_1) \setminus Y_1)$ and $(Y_2, \pi(V_2 \cup A_2) \setminus Y_2)$ be a $(\pi(A^{\text{ext}}|_{V_1 \cup A_1}), \pi(B^{\text{ext}}|_{V_1 \cup A_1}))$-mincut on H_1 and a $(\pi(A^{\text{ext}}|_{V_2 \cup A_2}), \pi(B^{\text{ext}}|_{V_2 \cup A_2}))$-mincut on H_2 respectively. Notice that every hyperedge in $Sep((\partial X) \cap S, V_1, V_2)$ are in $\partial(Y_1 \cup Y_2)$. Let $Y_0 := Y_1 \cup Y_2$ and after removing all anchor vertices we get $Y = Y_0 \cap V$. Now $(Y, \pi(V) \setminus Y)$ is a $(\pi(A), \pi(B))$-cut. Since for each hyperedge $e \in (\partial_G X) \cap S$, e is separated into two hyperedges and both of them are in $\partial_{H_1 \cup H_2} Y_0$, we have

$$|\partial_H Y| \le |\partial_{H_1 \cup H_2} Y_0| - |(\partial_G X) \cap S|. \tag{1}$$

Notice that the inequality in Equation (1) comes from the fact that $\partial_{H_1 \cup H_2} Y_0$ may or may not contain more separated hyperedges from $Sep(S \setminus \partial X, V_1, V_2)$.

Finally we obtain

$$\begin{aligned}
\text{mincut}^c_H(\pi(A), \pi(B)) &\le |\partial_H Y| && (Y \text{ is some } (\pi(A), \pi(B))\text{-cut on } H)\\
&\le |\partial_{H_1 \cup H_2} Y_0| - |(\partial X) \cap S| && (\text{by Equation (1)})\\
&= |\partial_{H_1} Y_1| + |\partial_{H_2} Y_2| - |(\partial X) \cap S| \\
& && (Y_0 \text{ is the disjoint union } Y_1 \cup Y_2)\\
&\le |\partial_{G^{\mathsf{sep}}} X^{\mathsf{sep}}| - |(\partial X) \cap S| && (X^{\mathsf{sep}} \text{ is some } (A^{\mathsf{ext}}, B^{\mathsf{ext}})\text{-cut})\\
&= |\partial X| && (\text{exactly } |(\partial X) \cap S| \text{ hyperedges were separated})\\
&= \text{mincut}^c_G(A, B) && (X \text{ is an } (A, B)\text{-mincut})
\end{aligned}$$

as desired.

Part 2. The proof of $\text{mincut}^c_H(\pi(A), \pi(B)) \ge \text{mincut}^c_G(A, B)$ is very similar to Part 1, so we defer the proof (for completeness) in the full version. ◀

3.4 $(5c, c)$-Edge-Unbreakable Terminals

Let $G = (V, E)$ be a hypergraph and let $\mathcal{T} \subseteq V$ be the set of terminals. By adopting the notations from [17], we say that a terminal set $\mathcal{T}$ is $(5c, c)$-*edge-unbreakable* on G if for any bipartition (V_1, V_2) of V with no more than c crossing edges $|E_G(V_1, V_2)| \le c$, either $|\mathcal{T}|_{V_1}| < 5c$ or $|\mathcal{T}|_{V_2}| < 5c$. That is, if there is a cut of size at most c, then at least one of the sides has less than $5c$ induced terminals.

Liu [16] obtained an $(\mathcal{T}, c)$-sparsifier of size $O(|\mathcal{T}|c^2)$ with a $(5c, c)$-edge-unbreakable terminal set $\mathcal{T}$ where each terminal vertex $v \in \mathcal{T}$ has degree 1. It turns out that Liu's techniques naturally extend to hypergraphs. We prove the following in the full version of the paper.

▶ **Lemma 7.** *Let $G = (V, E)$ be a hypergraph and let $\mathcal{T} \subseteq V$ be a set of degree-1 terminals. If $\mathcal{T}$ is $(5c, c)$-edge-unbreakable on G, then there exists a subset $E' \subseteq E$ with $O(|\mathcal{T}|c^2)$ hyperedges, such that $G/(E - E')$ is a $(\mathcal{T}, c)$-sparsifier of G.*

4 Existence of $(\mathcal{T}, c)$-Sparsifiers with $O(kc^3)$ Hyperedges

With all the tools equipped in the previous section, we are able to prove the existence of a $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges.

▶ **Theorem 8.** *Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$ be the set of terminals. Then there is a subset $E' \subseteq E$ such that $|E'| = O(|\mathcal{T}|c^3)$ and the contracted hypergraph $G/(E - E')$ is $(\mathcal{T}, c)$-equivalent to G.*

To prove Theorem 8, it suffices to prove the following Lemma 9 where every terminal vertex has degree 1:

▶ **Lemma 9.** *Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$ be the set of degree 1 terminals. Then there is a subset $E' \subseteq E$ such that $|E'| = O(|\mathcal{T}|c^2)$ and the contracted hypergraph $G/(E - E')$ is $(\mathcal{T}, c)$-equivalent to G.*

Proof of Theorem 8. Without loss of generality, we may assume that each vertex in $\mathcal{T}$ has degree at most c, by duplicating each terminal vertex and add c parallel edges between the duplicated vertex and the original vertex. Let $\mathcal{T}$ be the terminal set of an input instance. Now, assuming each terminal has degree at most c, we can further duplicate each of these terminals c times so we have a set $\mathcal{T}'$ of at most $|\mathcal{T}|c$ degree-1 terminal vertices. By Lemma 9, there exists a subset $E' \subseteq E$ such that $|E'| = O(|\mathcal{T}'|c^2) = O(|\mathcal{T}|c^3)$ and the contracted hypergraph $G/(E - E')$ is $(\mathcal{T}, c)$-equivalent to G. ◀

To prove Lemma 9, we first present an algorithm SparsifySlow (See Algorithm 2). The algorithm recursively apply divide and conquer framework until the terminal set is $(5c, c)$-edge-unbreakable as the base case. After applying Lemma 7 on each base case, the algorithm combines the sparsifiers from the subproblems using Lemma 6.

Algorithm 2 SparsifySlow SparsifySlow$(G, \mathcal{T}, c)$.

Input: An undirected unweighted multi-hypergraph G, a set of degree-1 vertex terminals $\mathcal{T} \subseteq V$, and a constant c.
Output: A $(\mathcal{T}, c)$-sparsifier H for G.

if $\mathcal{T}$ *is* $(5c, c)$*-edge-unbreakable* **then**
  Construct H, a $(\mathcal{T}, c)$-sparsifier of G using Lemma 7.
  return H.
else
  Let (V_1, V_2) be a bipartition of $V(G)$ that refutes the $(5c, c)$-edge-unbreakable property. That is, $|E_G(V_1, V_2)| \leq c$ but $|\mathcal{T} \cap V_1| \geq 5c$ and $|\mathcal{T} \cap V_2| \geq 5c$.
  Obtain $\begin{cases} H_1 \leftarrow \text{SparsifySlow}(\hat{G}[V_1], \mathcal{T}_1, c), \text{ and} \\ H_2 \leftarrow \text{SparsifySlow}(\hat{G}[V_2], \mathcal{T}_2, c). \end{cases}$.
  return $H \leftarrow H_1 \cup H_2 \cup E(V_1, V_2) - Sep(S, V_1, V_2)$.
end

Lemma 10 and Lemma 11 give the correctness proof and the size to the returned $(\mathcal{T}, c)$-sparsifier from Algorithm 2.

▶ **Lemma 10.** *Algorithm 2 returns a* $(\mathcal{T}, c)$*-sparsifier of* G*.*

Proof. First we notice that all vertices in $\mathcal{T}_1$ and $\mathcal{T}_2$ have degree 1 in $\hat{G}[V_1]$ and $\hat{G}[V_2]$ respectively: the anchor vertices have degree 1 and so the recursive calls in Algorithm 2 are valid. The correctness is then recursively guaranteed by Lemma 6 (divide-and-conquer step) and Lemma 7 (base case). ◀

▶ **Lemma 11.** *Let* G *be a hypergraph,* $\mathcal{T} \subseteq V$ *is the set of degree-1 terminal vertices, and let* c *be a constant. Let* $H = \text{SparsifySlow}(G, \mathcal{T}, c)$ *be the output of Algorithm 2. Then* H *has at most* $O(|\mathcal{T}|c^2)$ *hyperedges.*

The proof to Lemma 11 is via a potential function similarly defined in Liu [16].

Proof. The execution to Algorithm 2 defines a recursion tree. If $|\mathcal{T}| < 5c$, then the recursion terminates immediately because $\mathcal{T}$ is trivially $(5c, c)$-edge-unbreakable by definition and a $(\mathcal{T}, c)$-sparsifier of $O(|\mathcal{T}|c^2)$ hyperedges is returned by Lemma 7. Assume that $|\mathcal{T}| \geq 5c$, then each recursive call on the subproblem $(G', \mathcal{T}')$ guarantees that $|\mathcal{T}'| \geq 5c$.

Now, it suffices to use the following potential function to prove that the total number of terminal vertices in all recursion tree leaves can be bounded by $O(|\mathcal{T}|)$. Define a potential function for each subproblem $(G', \mathcal{T}')$ to be $\Phi(G', \mathcal{T}') := |\mathcal{T}'| - 5c$. Then, according to Algorithm 2, whenever $(G', \mathcal{T}')$ splits into two subproblems $(\hat{G}'[V_1], \mathcal{T}'_1)$ and $(\hat{G}'[V_2], \mathcal{T}'_2)$ we have

$$\begin{aligned}\Phi(\hat{G}'[V_1], \mathcal{T}'_1) + \Phi(\hat{G}'[V_2], \mathcal{T}'_2) &\leq |\mathcal{T}' \cap V_1| + |\mathcal{T}' \cap V_2| + 4|E_{G'}(V_1, V_2)| - 10c \\ &\leq \Phi(G', \mathcal{T}') - c.\end{aligned}$$

Since every subproblem has a non-negative potential, and the sum of potential decreases by c at each divide-and-conquer step, the total number of leaf cases do not exceed $\Phi(G, \mathcal{T})/c \leq |\mathcal{T}|/c$. Hence, the total size from the base case is at most $\sum_{(G', \mathcal{T}'): \text{ base case}} |\mathcal{T}'| \leq \Phi(G, \mathcal{T}) + (5c)(\#\text{ of leaf cases}) = O(|\mathcal{T}|)$.

By Lemma 7, the total number of hyperedges returned from Algorithm 2 is at most $O(|\mathcal{T}|c^2)$. The total number of hyperedges added back at Algorithm 2 is at most the number of divide-and-conquer steps times the cut size, which is at most $|\mathcal{T}|$. Therefore, the output $(\mathcal{T}, c)$-sparsifier H has at most $O(|\mathcal{T}|c^2)$ hyperedges as desired. ◀

Proof of Lemma 9. Lemma 9 follows immediately after the correctness proof (Lemma 10) and upper bounding the number of hyperedges (Lemma 11). ◀

5 An Almost-linear-time Algorithm Constructing a Sparsifier

This section is devoted to proving part (1) in Theorem 1. That is, we give a almost-linear-time (assuming a constant rank) algorithm that constructs a contraction based $(\mathcal{T}, c)$-sparsifier of $O(|\mathcal{T}|c^3)$ hyperedges which matches with Theorem 8 up to a constant factor. We summarize the result in Theorem 12.

▶ **Theorem 12.** *Let $G = (V, E)$ be a hypergraph with n vertices, m hyperedges, and rank $r = \max_{e \in E} |e|$. Let $\mathcal{T} \subseteq V$ be a terminal set $\mathcal{T} \subseteq V$. Then there exists a randomized algorithm which constructs a $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges in $O(p + n(rc \log n)^{O(rc)} \log m)$ time.*

Overview of the algorithm. Although Algorithm 2 can construct a $(\mathcal{T}, c)$-sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges, it is slow because we do not have an efficient algorithm searching for a bipartition that violates the $(5c, c)$-edge-unbreakable property.

To construct our contraction-based $(\mathcal{T}, c)$-sparsifier, all we need to do is identifying *essential* hyperedges and contract non-essential ones. Essential hyperedges are indispensable to maintaining mincut between terminals. It seems to be challenging to identify essential hyperedges on an arbitrary graph without a $(5c, c)$-edge-unbreakable guarantee. Fortunately, we notice there is an efficient way to identify essential hyperedges in an expander.

Naturally, we can utilize ExpanderDecompose (where the version for hypergraphs is explicitly stated in [18]) which splits a hypergraph into expanders. Expander decomposition not only guarantees expander sub-hypergraphs, but also fits in the divide-and-conquer framework indicated by Lemma 6 with a favorable almost-linear time. Then, we can focus on identifying essential hyperedges in an expander.

To identify essential hyperedges in an expander, we first enumerate all *connected cuts*[1] with value at most c – the sub-hypergraph induced by the smaller side of a connected cut

[1] In Chalermsook et al. [4], the concept of connected cuts is not explicitly defined. We give a formal definition in the full version and hope it clarifies some ambiguity in their paper.

is connected; see Algorithm 3 and Algorithm 4. Then, we build a *pruned auxiliary graph* based on the cuts we have enumerated. The pruned auxiliary graph leads to an efficient way identifying essential hyperedges. Finally, we contract all detected non-essential hyperedges. We call the above procedure that sparsifies an expander ϕ-SPARSIFY. See Algorithm 5.

Algorithm 3 ENUMERATECUTS (G, ϕ, r, c).

Input: A ϕ-expander hypergraph $G = (V, E)$ with rank r, and a threshold parameter c.
Output: All connected cuts with value at most c.

$\mathcal{C} \leftarrow \emptyset$. // Stores all found connected cuts.
for *each* $v_{seed} \in V$ **do**
  /* Invokes a helper function to find all connected cuts involving v_{seed}. */
  $\mathcal{C} \leftarrow \mathcal{C} \cup$ ENUMERATECUTSHELP$(0, G, G, \phi, r, c, v_{\text{seed}})$. // See Algorithm 4.
end
return $\mathcal{C}$.

Algorithm 4 ENUMERATECUTSHELP $(depth, H, G, \phi, r, c, v_{\text{seed}})$.

Input: The current recursion depth *depth*. A hypergraph $H = (V, E)$ with rank r. The original hypergraph G. Parameters c and ϕ. A seed vertex $v_{\text{seed}} \in V$.
Output: All connected cut with value at most c so that v_{seed} is in the smaller side.

if *depth* $\leq rc$ **then**
  Run DFS from v_{seed} on H and **stop** as soon as visiting $c\phi^{-1} + 1$ hyperedges.
  Let $\hat{E}$ be the set of visited hyperedges and X be the set of visited vertices.
  if *DFS gets stuck before visiting* $c\phi^{-1} + 1$ *hyperedges* **then**
    if $|\partial_G X| \leq c$ **then**
      return $\{(X, V \setminus X)\}$. /* Some connected cut with value at most c. */
    else
      return $\emptyset$.
    end
  else
    $\mathcal{S} \leftarrow \emptyset$.
    for *each* $e \in \hat{E}$ *and for each* $v \in e$, $v \neq v_{\text{seed}}$ **do**
      Let $e' \leftarrow e \setminus v$./* modify the boundary hyperedge into a smaller one. */
      // A recursive call with v being removed from e.
      $\mathcal{S} \leftarrow \mathcal{S} \cup$ ENUMERATECUTSHELP$(depth + 1, H - e + e', G, \phi, r, c, v_{\text{seed}})$
    end
    return $\mathcal{S}$.
  end
else
  return $\emptyset$.
end

Algorithm 5 ϕ-SPARSIFY $(G, \mathcal{T}, \phi, r, c)$.

Input: ϕ-expander hypergraph $G = (V, E)$ with rank r, terminal set $\mathcal{T}$,threshold parameter c.
Output: A $(\mathcal{T}, c)$-sparsifier of G.

Run EnumerateCuts and construct the pruned auxiliary graph $G^{\mathsf{aux}} = (V^{\mathsf{aux}}, E^{\mathsf{aux}})$, where $V^{\mathsf{aux}} = P_0 \cup C_0 \cup E_0$.
Let $E' \leftarrow E \setminus E_0$.
for *each* $e \in E_0$ *(in any order)* **do**
  Compute the set of partitions $P'_e := P_0 \cap N(N(e))$ who has at least one mincut that contains the edge e.
  if $\forall\ p \in P'_e$, $N(p) \not\subseteq N(e)$ **then**
    Remove $N(e)$ and all incident edges from G^{aux}.
    $E' \leftarrow E' \cup \{e\}$. // e is non-essential.
  end
end
return G/E'.

Algorithm 6 SPARSIFYFAST $(G, r, \mathcal{T}, c, C')$.

Input: hypergraph $G = (E, V)$ with rank r, terminal set $\mathcal{T}$, threshold parameter c, constant C'.
Output: a $(\mathcal{T}, c)$-sparsifier H.

$H \leftarrow G$.
$iter \leftarrow 0$. /* Number of iterations of the following while-loop. */
do
  $G \leftarrow H$
  $\phi^{-1} \leftarrow 4C'rc^4 \log^3 n$.
  $\{V_i\}_{i=1}^t \leftarrow$ ExpanderDecompose(G, ϕ).
  $G' \leftarrow G$ /* Anchored sub-hypergraphs will be separated from G' one by one. */
  for *each* $i = 1, 2, \ldots, t$ **do**
    Apply the divide step in Algorithm 1 to G' with terminal $\mathcal{T}$ and bipartition $(V_i, \bigcup_{\ell=i+1}^t V_\ell)$, and get $G_i \leftarrow \hat{G}'[V_i]$ and $G' \leftarrow \hat{G}'[\bigcup_{\ell=i+1}^t V_\ell]$.
    (For each boundary hyperedge e with anchor vertices $v_{e,3}$ and $v_{e,4}$ created on the $\bigcup_{\ell=i+1}^t V_\ell$ side, we assign both $v_{e,3}$ and $v_{e,4}$ to an arbitrary V_j such that $j > i$ and $e \cap V_j \neq \emptyset$.)
  $\{H_i\}_{i=1}^t \leftarrow \{\phi$-SPARSIFY$(G_i, V_i \cap \mathcal{T}, \phi, r, c)\}_{i=1}^t$. /* The conquer step. */
  $H \leftarrow H_t$ /* Each sparsifier H_i will be merged with H one by one. */
  for *each* $i = t-1, \ldots, 1$ **do**
    Apply the combine step in Algorithm 1 to merge H_i with H. That is, all anchor vertices introduced at the divide step are removed and all separated hyperedges are replaced by the boundary hyperedges before separating V_i from $\bigcup_{\ell=i}^t V_\ell$.
  $iter = iter + 1$.
while $iter < \log m$.
return H.

With ExpanderDecompose and ϕ-Sparsify procedures introduced above, we are able to construct the $(\mathcal{T}, c)$-sparsifier on general hypergraphs of size $O(|\mathcal{T}|c^3)$ efficiently. Our algorithm (Algorithm 6) is based on Chalermsook et al. [4] and consists of iterations of ExpanderDecompose and ϕ-Sparsify. Each iteration implements the divide-and-conquer framework shown by Algorithm 1: we first apply ExpanderDecompose and decompose the hypergraph into ϕ-expanders. Then we apply ϕ-Sparsify to sparsify the ϕ-expanders. Finally, we glue all sparsifiers of the ϕ-expanders by recovering the inter-cluster hyperedges between the ϕ-expanders. Similar to [4], we prove that $O(\log m)$ iterations suffice to obtain a $(\mathcal{T}, c)$-sparsifier of $O(|\mathcal{T}|c^3)$ hyperedges.

Due to the page limit, we refer the readers to the full version of this paper to see the details of the algorithms in this section.

References

1 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

2 Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 910–928. IEEE, 2019.

3 András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

4 Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1206–1225. SIAM, 2021.

5 Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018.

6 Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72. IEEE, 2020.

7 David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)*, 44(5):669–696, 1997.

8 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

9 Wenyu Jin and Xiaorui Sun. Fully dynamic st edge connectivity in subpolynomial time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 861–872. IEEE, 2022.

10 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 598–611, 2021.

11 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Spectral hypergraph sparsifiers of nearly linear size. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1159–1170. IEEE, 2022.

12 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014.

13 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 367–376, 2015.

14 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

15 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1789–1799. SIAM, 2013.

16 Yang P. Liu. Vertex sparsification for edge connectivity in polynomial time. *CoRR*, abs/2011.15101, 2020. `arXiv:2011.15101`.

17 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-cut. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2020.

18 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. *CoRR*, abs/2205.03930, 2022.

19 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparsek-connected spanning subgraph of ak-connected graph. *Algorithmica*, 7(1):583–596, 1992.

20 Richard Peng, Bryce Sandlund, and Daniel D Sleator. Optimal offline dynamic 2, 3-edge/vertex connectivity. In *Workshop on Algorithms and Data Structures*, pages 553–565. Springer, 2019.

21 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2570–2581. SIAM, 2019.

22 Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.