Sparsification of Directed Graphs via Cut Balance

Ruoxu Cen
Duke University, Durham, NC, USA

Yu Cheng
University of Illinois at Chicago, IL, USA

Debmalya Panigrahi
Duke University, Durham, NC, USA

Kevin Sun
Duke University, Durham, NC, USA

Abstract

In this paper, we consider the problem of designing cut sparsifiers and sketches for directed graphs. To bypass known lower bounds, we allow the sparsifier/sketch to depend on the *balance* of the input graph, which smoothly interpolates between undirected and directed graphs. We give nearly matching upper and lower bounds for both *for-all* (cf. Benczúr and Karger, STOC 1996) and *for-each* (Andoni et al., ITCS 2016) cut sparsifiers/sketches as a function of cut balance, defined the maximum ratio of the cut value in the two directions of a directed graph (Ene et al., STOC 2016). We also show an interesting application of digraph sparsification via cut balance by using it to give a very short proof of a celebrated maximum flow result of Karger and Levine (STOC 2002).

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Graph sparsification, directed graphs, cut sketches, space complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.45

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2006.01975`

Funding *Debmalya Panigrahi and Kevin Sun*: This work was supported in part by NSF grants CCF-1535972, CCF-1955703, and an NSF CAREER Award CCF-1750140.

Acknowledgements Part of the work was done while Yu Cheng was visiting the Institute for Advanced Study.

1 Introduction

Graph sparsification, originally introduced by Benczúr and Karger as a means of obtaining faster maximum flow algorithms [8], has become a fundamental tool in graph algorithms. The goal of graph sparsification is to replace an arbitrary graph with a sparse graph (called the graph sparsifier) on the same set of n vertices but with only $O(n \cdot \text{poly}(\log n, 1/\epsilon))$ edges, while approximately preserving the value of every cut up to a factor of $1 \pm \epsilon$ for any given $\epsilon > 0$. Since their work, several graph sparsification techniques have been discovered (e.g., [17]), the idea has been extended to other models of computation such as data streaming (e.g., [1]) and sketching (e.g., [6]), stronger notions such as spectral sparsification that preserves all quadratic forms have been proposed (e.g., [43]), and far-reaching generalizations such as the Kadison-Singer conjecture have been established [37]. On the applications front, graph sparsification has been heavily used to obtain a tradeoff between algorithmic accuracy and efficiency for a variety of "cut-based" problems such as maximum flows, minimum cuts, balanced separators, etc.

In spite of its widespread use, one restriction is that most sparsification techniques only apply to undirected graphs. There is a fundamental reason for this restriction – there are directed graphs that *cannot* be sparsified (see Fig. 1 for an example). Indeed, the lower

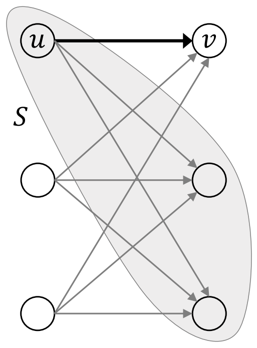

Figure 1 The graph is a complete bipartite graph where all edges go from the left vertices L to the right vertices R. For any $u \in L$ and $v \in R$, the only edge that leaves the set $S = \{u\} \cup (R \setminus \{v\})$ is the edge (u, v). Consequently, if we want to preserve the value of all directed cuts in this graph, we have to (approximately) store the weight of every edge.

bound holds even for *cut sketches*, where one does not insist on a graph being output as the sparsifier, but simply a succinct data structure from which the cut values of the original graph can be (approximately) retrieved.

A qualitative distinction between directed and undirected graphs is in terms of the *balance* of cuts, i.e., the ratio between incoming and outgoing edges in any given cut. An equivalent view of an undirected graph is by bi-directing its edges, which results in a graph with perfect balance, i.e., every cut has exactly the same number of incoming and outgoing edges.[1] Cut balance, therefore, smoothly interpolates between undirected and directed graphs, which leads to the question: *can we design cut sparsifiers/sketches for directed graphs that depend on cut balance?* We answer this question in the affirmative in this paper, and show that this view of sparsification leads to interesting consequences.

We note that the use of cut balance to bridge between undirected and directed graphs predates our work. Ene et al. [15] introduced the notion of parameterizing digraphs by their cut balance (or simply balance) and defined it as the largest ratio between the value of a cut in its two directions. Using this view, they extended two classic operations on undirected graphs – oblivious routing and fast approximate maximum flows – to directed graphs with a dependence on the balance. In this paper, we show that this phenomenon is exhibited by cut sparsification as well.

1.1 Our Results

We consider the two canonical forms of cut sparsification considered in the literature. The first is the classic version introduced by Benczúr and Karger [8], where all cuts must simultaneously be approximately preserved *whp*;[2] we call this *for-all* sparsification. The second, more relaxed, notion is due to Andoni et al. [6], where *any* cut must be approximately preserved *whp* instead of all cuts simultaneously; we call this *for-each* sparsification. For both these notions of sparsification, previous results on undirected graphs can be extended to β-balanced graphs by boosting sampling probabilities in undirected sparsification algorithms by a factor of β, thereby losing an additional factor of β in the size of the sparsifier/sketch (see also Ikeda and Tanigawa [20]). Is it possible to do better than losing a factor of β?

[1] Note that all Eulerian digraphs, whether or not derived from undirected graphs, exhibit perfect cut balance.
[2] with high probability

Our first result sharpens this naïve bound in for-each sparsification, by constructing cut sketches that improve the dependence on β to $\sqrt{\beta}$. We also show that this dependence is tight by constructing a matching lower bound. This pair of results resolves the precise dependence of for-each sparsification in directed graphs on the balance of the graph.

▶ **Theorem 1** (Upper Bound). *For any β-balanced graph with n vertices, m edges, and polynomially-bounded edge weights, there is an $\widetilde{O}(m + \sqrt{\beta}n/\epsilon)$-time algorithm*[3] *that constructs a $(1 \pm \epsilon)$ for-each cut sketch of size $\widetilde{O}(\sqrt{\beta}n/\epsilon)$ bits.*

(Lower Bound) Fix any $\beta \geq 1$, $0 < \epsilon < 1$, and n such that $(\beta/\epsilon)^{1/2} \leq n/2$. Any $(1 \pm \epsilon)$ for-each cut sketching algorithm for β-balanced graphs with n vertices must output at least $\Omega(\sqrt{\beta}n/\sqrt{\epsilon})$ bits in the worst case.

In for-all sparsification, we are not as lucky; we show that the linear dependence on β is tight in this case. (In fact, Ikeda and Tanigawa [20] had conjectured that better for-all sparsifiers can be constructed by sampling edges according to directed connectivity parameters; our lower bound construction refutes this conjecture and shows that such more aggressive sampling may not produce a sparsifier at all.)

▶ **Theorem 2.** *Fix any $\beta \geq 1$, $0 < \epsilon < 1$, and n such that $\beta/\epsilon \leq n/2$. Any $(1 \pm \epsilon)$ for-all cut sketching algorithm for n-node β-balanced graphs must output at least $\Omega(n\beta/\epsilon)$ bits.*

But, we note that the upper bound only applies to digraphs where *all* cuts are β-balanced. In general, the balance parameter for different cuts in a digraph may be highly non-uniform: some cuts could be very balanced and some others very unbalanced. For such graphs, we show a more refined result: for *any* value $\beta \geq 1$, we construct a sparsfier that approximately preserves all β-balanced cuts losing only an additional factor β in the size of the sparsifier. Note that this result holds for any value of β irrespective of the balance parameter of the graph; if β is the balance parameter, then it recovers the tight bound for β-balanced graphs.

▶ **Theorem 3.** *For any directed graph with n vertices, m edges, and non-negative edge weights, and any $\beta \geq 1$, there is an $\widetilde{O}(m)$-time algorithm that returns a (weighted) subgraph with $\widetilde{O}(\beta n/\epsilon^2)$ edges and preserves the values of all β-balanced cuts up to a factor of $1 \pm \epsilon$.*

We remark that digraph sparsification using cut balance has interesting consequences. In particular, note that for residual graphs produced by *s*-*t* maximum flow algorithms in undirected graphs, we can precisely bound the balance parameter on all cuts separating the source and the sink. Using this observation, we give a very short proof of the celebrated maximum flow result of Karger and Levine [27] via the digraph sparsification results.

1.2 Our Techniques

First, we outline the main ideas in our for-each cut sketch. In previous results on for-each cut sketches of undirected graphs [6, 21], the main idea was to (recursively) partition the graph into "sparse" and "dense" parts, and then maintain the sparse parts exactly along with a sample of the dense parts. A directed subgraph, however, can simultaneously be too dense to preserve exactly but also not amenable to sampling (e.g., a complete bipartite digraph). Of course, the balance parameter helps bridge this gap, but the cut balance of a subgraph that the algorithm encounters during recursion can be much worse than that of the original

[3] This runtime bound assumes that the value of β is known. If not, then β can be computed using an algorithm of Ene et al. [15] in $\widetilde{O}(\beta^2 m)$ time.

graph. Indeed, individual subgraphs might not even be strongly connected (i.e., have balance ∞), even if the original graph were Eulerian (i.e., has balance 1). This makes the (recursive) local sketching techniques in previous works unusable for directed graphs.

Our main technical contribution is a new *global* cut sketch construction. We design a cut sketch whose variance can be large on individual dense regions of the input digraph that are well-connected in an undirected sense, but we crucially show that the *cumulative variance of our estimator across all these well-connected regions of the digraph is small.* This helps eliminate the need for local cut sketches in each dense subgraph, and simplifies the recovery algorithm to the natural estimator that appropriately scales the number of sampled edges in the queried cut. Moreover, to obtain the right dependence on β, we need to carefully analyze the variance of our estimator. Our new variance analysis works for undirected graphs as well, which tightens the analysis of [6] and consequently leads to undirected cut sketching algorithms that do not require downsampling or low-accuracy for-all sparsifiers.

Next, we turn to for-all sparsification. Our first result is the lower bound on for-all sparsifiers and cut sketches. For any β and n, we construct a family $\mathcal{G}$ of β-balanced graphs on n vertices that satisfies two conflicting properties: $\mathcal{G}$ is a large family, yet for each graph $G \in \mathcal{G}$, the number of graphs in $\mathcal{G}$ that approximate all cuts of G is small. For any graph G and cut S, let $\delta_G(S)$ denote the value of S and $E_G(S)$ denote the edges crossing S. Notice that there are many possible graphs H such that $\delta_H(S) \approx \delta_G(S)$, because $E_H(S)$ and $E_G(S)$ could differ in numerous ways. Thus, to ensure that the number of cut approximators is small, we carefully design $\mathcal{G}$ such that for any $G, H \in \mathcal{G}$ and cut S, if $\delta_H(S) \approx \delta_G(S)$, then $E_H(S) \approx E_G(S)$. We show that this can be done by considering a large family of bipartite graphs that all contain a fixed (directed) matching. Consequently, any sketching algorithm must produce a large number of different cut sketches for the graphs in $\mathcal{G}$, which translates to a lower bound on the size of the cut sketches using standard information theory.

Finally, we refine for-all sparsification in digraphs by showing that we can preserve all balanced cuts, irrespective of the balance parameter of the entire graph G. More specifically, at a cost of an additional factor of β in the size of the sparsifer, we can preserve all β-balanced cuts, and provide an approximation for α-balanced cuts with $\alpha > \beta$ that degrades gracefully as α gets larger. For this purpose, we adopt a (recursive) graph decomposition due to Benczúr and Karger [8] that expresses a graph as a weighted sum of subgraphs, each of which corresponds to a particular edge sampling rate. Now we can boost the (undirected) sampling rate by a factor of β. If the balance of every subgraph in the decomposition is also β, then the undirected analysis carries over to the directed case. However, in general, each subgraph can be very unbalanced, so we cannot bound the estimation error in each individual subgraph. Our main technical contribution is to show that even though we do not preserve the cut values in individual subgraphs, we do so globally across all the subgraphs.

1.3 Related Work

Graph Sparsification. Graph sparsification was introduced by Benczúr and Karger [8] ("for-all" cut sparsification), and has led to research in a number of directions: Fung et al. [17] and Kapralov and Panigrahy [26] gave new algorithms for preserving cuts in a sparsifier; Spielman and Teng [43] generalized to spectral sparsfiers that preserved all quadratic forms, which led to further research both in reducing the size of the sparsifier [42, 7] and developing faster algorithms (e.g., [34, 5, 35, 11, 33, 30, 32, 31]); faster algorithms for fundamental graph problems such as maximum flow utilized sparsification results (e.g., [8, 40]); Ahn and Guha [1] introduced sparsification in the streaming model, which has led to a large body of work for both cut (e.g., [2, 3, 18]) and spectral sparsifiers (e.g., [25, 24, 23, 4]) in graph streams; both cut [29, 38] and spectral [41] sparsification have been studied in hypergraphs. For lower bounds, Andoni et al. [6] showed that any data structure that $(1 \pm \epsilon)$-approximately stores the sizes of all cuts in an undirected graph must use $\Omega(n/\epsilon^2)$ bits. Carlson et al. [10] improved this lower bound to $\Omega(n \log n/\epsilon^2)$ bits, matching existing upper bounds.

Andoni et al. [6] first proposed the notion of "for-each" cut (and spectral) sketches, where the sparsifier preserves the value of *any* cut rather than all cuts simultaneously. They showed that for any undirected graph with n vertices, a $(1 \pm \epsilon)$ for-each cut sketch of size $\widetilde{O}(n/\epsilon)$ exists and can be computed in polynomial time. Subsequently, Jambulapati and Sidford [21] gave the first nearly-linear time algorithm for constructing $(1 \pm \epsilon)$ for-each graph sketches of size $\widetilde{O}(n/\epsilon)$. Their sketch not only approximates cut values, but also approximately preserves the quadratic form of any undirected Laplacian matrix (and its pseudoinverse). Chu et al. [11] showed how to construct a *graph* containing $\widetilde{O}(n^{1+o(1)}/\epsilon)$ edges that satisfies the "for-each" requirement for spectral queries.

Directed Graphs. Cohen et al. [14, 13] proposed a directed notion of spectral sparsifiers and used it to obtain nearly-linear time algorithms for solving directed Laplacian linear systems and computing various properties of directed random walks. However, their directed spectral sparsifiers only work for Eulerian graphs, i.e., for $\beta = 1$. Zhang et al. [44] proposed a notion of spectral sparsification that works for all directed graphs, but their definition does not preserve cut values. More generally, there have been attempts at bridging the divide between directed and undirected graphs for other problems. For instance, Lin [36] defined the imbalance of a graph as the sum of the absolute difference of in- and out-capacities at all vertices, and used it to generalize the max-flow algorithm of Karger and Levine [27] from undirected graphs to digraphs. Digraphs have also been parameterized by directed extensions of treewidth [22], and similar notions of DAG-width [9, 39] and Kelly-width [19], which led to FPT algorithms based on these parameters, much like for undirected bounded treewidth graphs. In spectral graph theory, directed analogs of Cheeger's inequality have been defined [12], particularly in the context of analyzing the spectrum of digraphs. Closest to our work is that of Ene et al. [15] who proposed cut balance of digraphs that we use in this paper, although in the context of oblivious routing and max-flow algorithms.

2 Preliminaries

Basic Notations. Let $G = (V, E, w)$ be a weighted *directed* graph with $n = |V|$ vertices and $m = |E|$ edges. Every edge $e \in E$ has a given non-negative weight $w_e \geq 0$. When working with unweighted graphs, (i.e., $w_e = 1$ for all $e \in E$), we will omit the edge weights w_e.

For two sets of vertices $S \subseteq V$ and $T \subseteq V$, we use $E(S,T) = \{(u,v) \in E : u \in S, v \in T\}$ to denote the set of edges in E that go from S to T. We use $w(S,T) = \sum_{e \in E(S,T)} w_e$ to denote the total weight of the edges from S to T. For a vertex $u \in V$ and a set of vertices $S \subseteq V$, we write $E(u,S)$ for $E(\{u\},S)$, and we define $E(S,u)$, $w(u,S)$, and $w(S,u)$ similarly.

We often write $\overline{S}$ as a shorthand for $V \setminus S$. Given a component V_i and a subset of its vertices $S_i \subseteq V_i$, we can similarly define $\overline{S_i} = V_i \setminus S_i$. For example, using this notation, we write $w(S, \overline{S})$ for $w(S, V \setminus S)$ and similarly $w(S_i, \overline{S_i}) = w(S_i, V_i \setminus S_i)$.

The *conductance* of an undirected graph $G = (V, E, w)$ is defined as

$$\phi(G) = \min_{\varnothing \neq S \subset V} \frac{w(S, \overline{S})}{\min\bigl(w(S,V), w(\overline{S}, V)\bigr)}. \tag{1}$$

▶ **Definition 4** (β-Balanced). *A strongly connected digraph $G = (V, E, w)$ is* β-balanced *if, for all $\varnothing \subsetneq S \subsetneq V$, it holds that $w(S, \overline{S}) \leq \beta \cdot w(\overline{S}, S)$.*

Directed Cut Sparsifiers and Cut Sketches. We consider two notions of sparsification. The first is the classic “for-all” sparsifier that approximately preserves the values of all cuts.

▶ **Definition 5** (For-All Cut Sparsifier). *Let $G = (V, E_G, w_G)$ and $H = (V, E_H, w_H)$ be two weighted directed graphs. Fix $0 < \epsilon < 1$. We say H is a $(1 \pm \epsilon)$ for-all cut sparsifier of G iff the following holds for all $S \subseteq V$:*

$$(1-\epsilon) \cdot w_G(S, V \setminus S) \leq w_H(S, V \setminus S) \leq (1+\epsilon) \cdot w_G(S, V \setminus S).$$

Instead of a graph that preserves cut values, if we allow any data structure from which the cut values can be (approximately) recovered, we call it a cut sketch.

▶ **Definition 6** (For-All Cut Sketch). *Let $G = (V, E, w)$ be a weighted directed graph. Fix $0 < \epsilon < 1$. A (deterministic) function g outputs a $(1 \pm \epsilon)$ for-all cut sketch of G if there exists a recovering function f such that, for all $S \subseteq V$:*

$$(1-\epsilon) \cdot w(S, V \setminus S) \leq f(S, \mathrm{sk}(G)) \leq (1+\epsilon) \cdot w(S, V \setminus S).$$

Next we consider a weaker notion of graph sparsification, where instead of approximating the value of all cuts, we only require the value of any individual cut to be approximately preserved with (high) constant probability.

▶ **Definition 7** (For-Each Cut Sketch). *Let $G = (V, E, w)$ be a weighted directed graph. Fix $0 < \epsilon < 1$. A function g outputs a for-each $(1 \pm \epsilon)$-cut sketch of G if there exists a recovering function f such that, for each $S \subseteq V$, with probability at least $2/3$,*

$$(1-\epsilon) \cdot w(S, V \setminus S) \leq f(S, g(G)) \leq (1+\epsilon) \cdot w(S, V \setminus S).$$

3 For-All Sparsification: $\widetilde{O}(n \cdot \beta/\epsilon^2)$ Upper Bound

In this section, we extend the seminal work of Bencúr and Karger [8] to directed graphs using cut balance. For undirected graphs, they showed that sampling every edge inversely proportional to a quantity known as its strength (see Definition 11) preserves all cuts with high probability. We show that, by boosting this sampling probability by a factor of β, this procedure can preserve the value of β-balanced cuts in any directed graph.

We then show that this sampling theorem can be applied in a black-box manner to recover the analysis of a celebrated maximum flow algorithm for undirected graphs given by Karger and Levine [27]. At each step of the algorithm, they sample edges from the residual network (which is directed) of an undirected graph. Using a customized version of the sparsification result from Benczúr and Karger [8], they show that with high probability, the sample contains an augmenting path. In contrast, our sampling procedure can be applied directly to the residual network, which simplifies the analysis of the algorithm.

The following theorem is our main result of this section:

▶ **Theorem 8.** *Let $G = (V, E)$ be a directed graph where each edge e has weight $u_e \geq 0$, and let ϵ, β be parameters. There is an $\widetilde{O}(m)$-time algorithm that returns a weighted subgraph H that satisfies the following with high probability: for every α-balanced cut U,*

$$\left(1 - \epsilon\sqrt{\frac{\alpha+1}{\beta+1}}\right) \cdot \delta_G(U) \leq \delta_H(U) \leq \left(1 + \epsilon\sqrt{\frac{\alpha+1}{\beta+1}}\right) \cdot \delta_G(U).$$

where $\delta_G(U)$ and $\delta_H(U)$ denote the cut value of U in G and H, respectively. Furthermore, H contains $O(\beta n \log n/\epsilon^2)$ edges in expectation.

Note that for the special case where the graph G is β-balanced, Theorem 3 is implied by Theorem 8: all cut values are preserved. This is the main result of Ikeda and Tanigawa [20].

▶ **Corollary 9** (Ikeda and Tanigawa [20]). *Consider the same setting as Theorem 8. If G is β-balanced, then with high probability, H approximates every cut of G up to a $(1 \pm \epsilon)$ factor.*

Before proving Theorem 8, we give an application of digraph sparsification to the maximum flow problem. In particular, we prove the correctness of the $\widetilde{O}(m + nv)$-time maximum flow algorithm given by Karger and Levine [27], where v is the value of the maximum flow. This algorithm is an adaptation of the classic augmenting paths algorithm of Ford and Fulkerson [16], but with the following crucial observation. Let f denote the current flow value in any iteration, and $\gamma = (v - f)/v$ denote the fraction of remaining flow in the residual network. Karger and Levine [27] show that, by boosting the undirected sampling procedure of Benczúr and Karger [8] by a factor of $1/\gamma$ and applying it to the residual network, the resulting sample contains an augmenting path with high probability. This saves on running time since the search for an augmenting path can then be performed on the sampled graph instead of the entire residual network. (See the full paper for more details.)

In contrast, we show that we can directly apply digraph sparsification to the residual network, with $\beta = 2/\gamma$ to obtain a short proof of the Karger-Levine theorem:

▶ **Theorem 10** (Karger and Levine [27]). *Suppose we apply the algorithm in Theorem 8 to the residual network in a maximum flow computation, with $\epsilon = 0.1$ and $\beta = 2/\gamma$, where $\gamma = (v - f)/v$ is the fraction of flow remaining in the residual network. Then with high probability, there is an augmenting path in the sample.*

Proof. We claim that every s-t cut S in the residual graph is β-balanced, where $\beta = 2/\gamma$. Suppose S initially contains capacity $c \geq v$, and currently, x units of flow are entering S. Since the flow value is $(1-\gamma)v$, the amount of flow leaving S is $x + (1-\gamma)v$. At the same time, the x units of flow entering S create a residual capacity of x leaving S. Thus, the total residual capacity leaving S is $c - x - (1-\gamma)v + x = c - (1-\gamma)v$. We can similarly show that the residual capacity entering S is $c + x + (1-\gamma)v - x = c + (1-\gamma)v$. Thus, in the residual graph, the balance of S is at most $\frac{c+(1-\gamma)v}{c-(1-\gamma)v} \leq \frac{2-\gamma}{\gamma} \leq \frac{2}{\gamma}$.

Now by setting $\epsilon = 0.1$ and $\beta = 2/\gamma$, Theorem 8 implies that the sparsifier H preserves all $(2/\gamma)$-balanced cuts up to a (1 ± 0.1) factor with high probability. Since every s-t cut is $(2/\gamma)$-balanced, this implies that there exists an augmenting path in H, as desired. ◀

In the rest of this section, we prove Theorem 8. Before we give our algorithm, we state the definitions and results that we need from previous work.

▶ **Definition 11** (Strength and strong components). *The* strength *of an edge e, denoted by k_e, is the largest k such that there exists a k-edge-connected vertex-induced subgraph of G containing e. A k-*strong component *is the subgraph induced by edges with strength at least k.*

▶ **Lemma 12** (Benczúr and Karger [8]). *The strong components of an undirected graph form a laminar family, and a graph on n vertices has at most $n - 1$ nontrivial strong components.*

▶ **Lemma 13** (Benczúr and Karger [8]). *In any graph with edge weights u_e and strengths k_e, we have $\sum_e u_e/k_e \leq n - 1$. Furthermore, there exists an $O(m \log^3 n)$-time algorithm that returns, for every edge e, an estimate $\widetilde{k}_e$ of k_e satisfying $\widetilde{k}_e \leq k_e$ and $\sum_e u_e/\widetilde{k}_e = O(n)$.*

We now describe our algorithm (Algorithm 1). The input is a directed graph where each edge e has weight u_e. We first compute approximate edge strengths $\widetilde{k}_e$ as given in Lemma 13. Then we sample each edge e proportional to $(\beta+1)u_e/\widetilde{k}_e$, where $\beta \geq 1$ is a chosen parameter. We choose the weight of the sampled edges so that we get an unbiased estimator.

Algorithm 1 For-all sparsification for directed graphs.

```
  Input  : An $n$-vertex directed graph $G = (V, E, u)$ with edge weights $u_e$, $0 < \epsilon < 1$,
           $\beta \geq 1$, and a constant $d > 2$.
  Output : A subgraph $H$ that satisfies Theorem 8.
1 Use Lemma 13 to compute an estimated edge strength $\widetilde{k}_e \leq k_e$ for every edge $e \in E$.
2 Let $\rho = 3d(\beta+1)\log n/\epsilon^2$.
3 for each edge $e \in E$ do
4     Sample $e$ with probability $p_e = \rho \cdot u_e/\widetilde{k}_e$.
5     if $e$ is sampled then add $e$ to $H$ with weight $w_e = \widetilde{k}_e/\rho$.
6 return $H$.
```

Now we analyze the output H of Algorithm 1. Without loss of generality, we assume that the algorithm uses the actual edge strengths k_e rather than the estimates $\widetilde{k}_e$. This is because $\widetilde{k}_e \leq k_e$ and it does not hurt to oversample in importance sampling.

For each strong component G_i of G (see Definition 11), let H_i denote the corresponding component in H. Because the way we choose sampling probabilities and edge weights in H, we have $\mathbb{E}[H_i] = G_i$. Let $\alpha_i = (k_i - k_{p(i)})/\rho$ where $p(i)$ is G_i's parent in the laminar family formed by strong components (see Lemma 12). As shown by Benczúr and Karger [8], this results in a decomposition of G into its strong components, that is, $G = \sum_i \alpha_i G_i$.

For a component G_i and a cut U, let $\delta_{G_i}(U)$ be the total capacity of edges leaving U in G_i, and let $\delta^{\text{un}}_{G_i}(U)$ be the corresponding value for the undirected version of G_i. The following lemma shows that for every strong component G_i, with high probability, $\delta_{G_i}(U)$ is preserved in H up to a relative error for every cut U.

▶ **Lemma 14.** *Let H be the output of Algorithm 1. For each strong component G_i (defined in Definition 11), the following holds with probability at least $1 - O(n^{-d+2})$: for any cut U, we have $\left|\delta_{H_i}(U) - \delta_{G_i}(U)\right| \leq \zeta(U) \cdot \delta_{G_i}(U)$ where $\zeta(U) = \epsilon\sqrt{\delta^{\text{un}}_{G_i}(U)/(\delta_{G_i}(U)(\beta+1))}$.*

Proof. For any cut U, let $\delta_p(U)$ denote the sum of u_e/k_e over the (undirected) edges crossing U_j in G_i. Order the r cuts intersecting G_i such that $1 = \delta_p(U_1) \leq \cdots \leq \delta_p(U_r)$, and let

$$q_j = \Pr\left(\left|\delta_{H_i}(U_j) - \delta_{G_i}(U_j)\right| > \zeta(U) \cdot \mathbb{E}\left[\delta_{H_i}(U_j)\right]\right).$$

By a Chernoff bound, we have

$$q_j \leq 2\exp\left(-\frac{(\zeta(U))^2 \cdot \delta_{G_i}(U_j)}{3}\right) = 2\exp\left(-\frac{\epsilon^2 \cdot \delta^{\text{un}}_{G_i}(U_j)}{3(\beta+1)}\right), \tag{2}$$

where the equality follows substituting the definition of $\zeta(U)$. Let $E(i,j)$ denote the set of edges crossing U_j in G_i. Since each edge e in G_i has weight p_e, we have

$$\delta^{\text{un}}_{G_i}(U_j) = \sum_{e \in E(i,j)} p_e = \sum_{e \in E(i,j)} \frac{3d(\beta+1)u_e}{\epsilon^2 k_e} \cdot \log n,$$

Substituting this into Eq. (2) shows

$$q_j \le 2\exp\left(-d \sum_{e\in E(i,j)} \frac{u_e}{k_e} \cdot \log n\right) = 2n^{-d\cdot\delta_p(U_j)}.$$

Since $\delta_p(U_j) \ge 1$, we have $q_j \le 2n^{-d}$, so

$$\sum_{j\le n^2} q_j \le n^2 \cdot 2n^{-d} = 2n^{-d+2} = O(n^{-d+2}). \tag{3}$$

For $j \ge n^2$, we express j as $j = n^{2\lambda}$. The number of λ-minimum cuts is at most j (see, e.g., Karger and Stein [28]) and $\delta_p(U_1) = 1$, so $\delta_p(U_j) \ge \lambda$, so $\delta_p(U_j) \ge \frac{\log j}{2\log n}$. This implies, for $j \ge n^2$, $q_j \le 2n^{-(d\log j)/(2\log n)} = 2j^{-d/2}$. Combining this with Eq. (3), we can conclude

$$\sum_{j\ge 1} q_j \le O(n^{-d+2}) + 2\int_{n^2}^{\infty} j^{-d/2} dj = O(n^{-d+2}). \qquad \blacktriangleleft$$

Now we are ready to prove Theorem 8.

Proof of Theorem 8. By Lemma 13, the expected number of edges in H is $\sum_e p_e = O(\beta n \log n/\epsilon^2)$, as claimed. Now consider an α-balanced cut U. We have

$$\left|\delta_H(U) - \delta_G(U)\right| = \left|\sum_i \alpha_i \delta_{H_i}(U) - \alpha_i \delta_{G_i}(U)\right| \le \sum_i \alpha_i \left|\delta_{H_i}(U) - \delta_{G_i}(U)\right|.$$

Taking a union bound over all strong components, we know that with high probability, Lemma 14 holds for every strong component. Thus, the quantity above is at most

$$\begin{aligned}\sum_i \alpha_i \zeta(U)\cdot\delta_{G_i}(U) &= \sum_i \alpha_i \epsilon \sqrt{\delta^{\text{un}}_{G_i}(U)\delta_{G_i}(U)/(\beta+1)} \\ &\le \frac{\epsilon}{\sqrt{\beta+1}}\sqrt{\sum_i \alpha_i \delta^{\text{un}}_{G_i}(U) \sum_i \alpha_i \delta_{G_i}(U)} && \text{(Cauchy-Schwarz)} \\ &= \frac{\epsilon}{\sqrt{\beta+1}}\sqrt{\delta^{\text{un}}_G(U)\delta_G(U)}. && \left(\textstyle\sum_i \alpha_i\delta_{G_i}(U) = \delta_G(U)\right)\end{aligned}$$

We conclude the proof by noting that $\delta^{\text{un}}_G(U) \le (\alpha+1)\cdot\delta_G(U)$, since U is α-balanced. ◀

4 For-All Sparsification: $\Omega(n \cdot \beta/\epsilon)$ Lower Bound

Our goal in this section to prove a lower bound whose dependence on β matches the linear upper bound given by Ikeda and Tanigawa [20] on the size of for-all cut sketches:

▶ **Theorem 15.** *Fix $\beta \ge 1$ and $0 < \epsilon < 1$ where $\beta/\epsilon \le n/2$. Any $(1 \pm \epsilon)$ for-all cut sketching algorithm for n-node β-balanced graphs must output $\Omega(n \cdot \beta/\epsilon)$ bits in the worst case.*

We prove a special case of our lower bound for $\beta = \Theta(n)$ and $\epsilon = \Theta(1)$ (Lemma 16). The proof for this special case contains the main ideas of our lower-bound construction for general values of β and ϵ. We defer the proof of Theorem 15 to the full paper.

▶ **Lemma 16.** *Let $\beta = 8n$ and let ϵ be a sufficiently small universal constant. Any $(1 \pm \epsilon)$ for-all cut sketching algorithm for n-node β-balanced graphs must output $\Omega(\beta n)$ bits in the worst case.*

We give an overview of how we prove Lemma 16. We can w.l.o.g focus on deterministic cut sketching algorithms, because running time is not a concern in Lemma 16, any randomized sketching algorithm can be derandomized by enumerating all possible coin flips.

We will choose a set of graphs $\mathcal{G}$ such that the following conditions hold:

- Every graph in $\mathcal{G}$ is β-balanced.
- The size of $\mathcal{G}$ is large (Lemma 17).
- There exists a ℓ with $|\mathcal{G}|/\ell = 2^{\Omega(\beta n)}$ such that, for every graph $G \in \mathcal{G}$, there are at most ℓ graphs in $\mathcal{G}$ that can share a $(1 \pm \epsilon)$-cut sketch with G (Lemma 18).

This way, each cut sketch works for at most ℓ graphs in $\mathcal{G}$, so any algorithm must produce at least $|\mathcal{G}|/\ell = 2^{\Omega(\beta n)}$ different cut sketches for all graphs in $\mathcal{G}$, which implies that the algorithm must output at least $\log_2(|\mathcal{G}|/\ell) = \Omega(\beta n)$ bits.

Formally, consider the set of graphs $\mathcal{G}_{2n}$ with $2n$ vertices defined as follows: every graph $G \in \mathcal{G}_{2n}$ is an unweighted bipartite graph with bipartitions L, R satisfying $|L| = |R| = n$. Fix a perfect matching from L to R. The set $\mathcal{G}_{2n}$ is defined to contain all graphs G such that the edges from L to R is exactly this perfect matching (and the set of edges from R to L are arbitrary). Let $\mathcal{G}_{2n,\beta} \subseteq \mathcal{G}_{2n}$ be the subset of graphs in $\mathcal{G}_{2n}$ that are β-balanced.

As described above, Lemma 17 gives a lower bound on the size of $\mathcal{G}_{2n,\beta}$.

▶ **Lemma 17.** *Let n_0 be a sufficiently large universal constant. If $n \geq n_0$ and $\beta = 8n$, then $|\mathcal{G}_{2n,\beta}| \geq 2^{n^2/2}$.*

The next lemma upper bounds the maximum number of graphs in $\mathcal{G}$ that can share an $(1 \pm \epsilon)$-cut sketch. Notice that if G and H have the same $(1 \pm \epsilon)$-cut sketch, then H must be a $(1 \pm 3\epsilon)$-cut sparsifier of G.

▶ **Lemma 18.** *Let $\epsilon > 0$ be a sufficiently small universal constant. For every $G \in \mathcal{G}_{2n}$, the number of graphs in $\mathcal{G}_{2n}$ that are $(1 \pm 3\epsilon)$-cut sparsifiers of G is at most $2^{n^2/4}$.*

We now prove Lemma 16 and defer the proofs of Lemmas 17 and 18 to the full paper.

Proof of Lemma 16. We work with graphs with $2n$ vertices (rather than n vertices) to make the presentation easier. This is equivalent because we aim to prove a lower bound of $\Omega(\beta n)$.

Fix any $(1 \pm \epsilon)$ for-all cut sketching algorithm. Consider running this algorithm on all graphs in $\mathcal{G}_{2n,\beta}$. Every graph in $\mathcal{G}_{2n,\beta}$ is β-balanced, so the algorithm must map every $G \in \mathcal{G}_{2n,\beta}$ to a bit string (i.e., cut sketch), and graphs that are not $(1 \pm 3\epsilon)$-cut sparsifiers of each other must be mapped to different strings. By Lemma 17, there are at least $2^{n^2/2}$ graphs in $\mathcal{G}_{2n,\beta}$, and by Lemma 18, at most $2^{n^2/4}$ graphs can be mapped to the same bit string. Therefore, the algorithm must output at least $\frac{2^{n^2/2}}{2^{n^2/4}} = 2^{n^2/4}$ distinct bit strings. This implies that the algorithm must output at least $\frac{n^2}{4} = \frac{1}{32}\beta n$ bits in the worst case. ◀

5 For-Each Cut Sketch: $\widetilde{O}(n \cdot \sqrt{\beta}/\epsilon)$ Upper Bound

In this section, we give an upper bound on the size of cut sketches in the for-each setting.

▶ **Theorem 19.** *Let G be an n-vertex β-balanced graph with edge weights in $[1, \mathrm{poly}(n)]$. There exists a $(1 \pm \epsilon)$ for-each cut sketch of size $O(n\beta^{1/2} \log^3 n/\epsilon)$ bits that approximates the value $w(S, V \setminus S)$ of every directed cut $S \subseteq V$ with high probability.*

We will prove a lower bound of $\Omega(n \cdot (\beta/\epsilon)^{1/2})$ bits in Section 6.

Overview of Our Approach. Our approach is inspired by the cut sketching algorithm for undirected graphs by Andoni et al. [6]. We first partition the edges into $\ell = O(\log n)$ disjoint sets $(E_i)_{i=1}^{\ell}$ based on their weights. Edges in $G_i = (V, E_i, w)$ have roughly the same weight and we can essentially treat G_i as an unweighted graph. For each graph G_i, we ignore edge directions, and iteratively remove and store edges belonging to sparse cuts.

Note that G_i may not be balanced. Even when G_i is balanced, the dense components of G_i may not be balanced. Despite this, we show that we can estimate the dense components' contribution to the cut value via random sampling. This is because we can bound the variance within each component, and in the end, upper bound their sum (i.e., the overall variance) using the β-balance condition.

One of our main technical contributions is to derive a tighter upper bound on the variance of random sampling. If we trace our analysis back to the undirected case, we remove some redundant terms in the analysis of [6]. This tighter variance bound is critical, because we cannot obtain the right space dependence on β without it (see the full paper). In addition, our new analysis can be traced back to the undirected case, which will simplify the algorithm of [6]. We can obtain a for-each sketching algorithms for undirected graphs without downsampling or low-accuracy for-all sparsifiers, and the output is a graph.

5.1 Sketching and Recovery Algorithms

Let $G = (V, E, w)$ be an n-node β-balanced directed graph with edge weights $w_e \in [1, \mathrm{poly}(n)]$. It is worth noting that, when constructing the cut sketch, we do not know the cut query $S \subseteq V$. The cut query S is only given as input to the recovery algorithm.

The Sketching Algorithm. We describe our overall cut sketching algorithm (Algorithm 2). We partition the edges into $O(\log n)$ weight classes. For each weight class, we iteratively store and remove all edges that belong to some λ-sparse cut (defined in Equation 4). When there are no λ-sparse cuts remain, we sample α incoming and outgoing edges at each vertex among the remaining edges. The values of λ and α will be specified later in our analysis.

For a directed graph $G = (V, E, w)$, we say a cut $(S, \overline{S})$ is λ-*sparse* if the following holds:

$$|E(S, \overline{S})| + |E(\overline{S}, S)| \le \lambda \cdot \min\big(|S|, |\overline{S}|\big). \tag{4}$$

The Recovery Algorithm. Algorithm 3 is our recovery algorithm that queries the cut sketch $\mathrm{sk}(G)$ (i.e., the output of Algorithm 2). We first establish some notation. Recall that Algorithm 2 decomposes G into $(G_i)_{i=1}^{\ell}$ according to the edge weights. Let $V_i = (V_{ij})_j$ denote the set of dense components in G_i after we iteratively remove the sparse cuts in G_i.

Algorithm 3 approximates $w(S, \overline{S})$ by adding the total contribution of the sparse-cut edges and the dense-component edges. Let J_S denote the total weight of sparse-cut edges that go from S to $\overline{S}$ in all of the graphs G_i (which we store deterministically). Let I_S be the estimator for the total weight of dense-component edges leaving S in all G_i as defined in Algorithm 3. Algorithm 3 returns $I_S + J_S$ as the final answer.

Correctness and Size Guarantees. We state the correctness of our recovery algorithm (Algorithm 3) in Lemma 20 and the output size of our sketching algorithm (Algorithm 2) in Lemma 21. Theorem 19 follows immediately from Lemmas 20 and 21; we prove the latter before proving the former.

▶ **Lemma 20** (Correctness of Algorithm 3). *Let* $\mathrm{sk}(G)$ *be the output of Algorithm 2. Fix a cut query* $S \subseteq V$*. With probability at least* $2/3$*, the value* $(I_S + J_S)$ *returned by Algorithm 3 on input* $(S, \mathrm{sk}(G))$ *satisfies* $\left|(I_S + J_S) - w(S, \overline{S})\right| \le O(\epsilon) \cdot w(S, \overline{S})$.

Algorithm 2 Compute a $(1 \pm O(\epsilon))$ for-each cut sketch.

```
Input  : An $n$-vertex $\beta$-balanced graph $G = (V, E, w)$ with edge weights
         $w_e \in [1, \mathrm{poly}(n)]$, and $0 < \epsilon < 1$.
Output : A $(1 \pm O(\epsilon))$ for-each cut sketch $\mathrm{sk}(G)$ of size $\widetilde{O}(n\beta^{1/2}/\epsilon)$.
1  Set $\alpha = \lambda = \beta^{1/2}/\epsilon$.
2  Partition the edges into $\ell = O(\log n)$ weight classes $E_1, \dots, E_\ell$ where
     $E_i = \{e : w_e \in [2^{i-1}, 2^i)\}$.
3  Each weight class $E_i$ defines a (possibly unbalanced) graph $G_i = (V, E_i, w)$.
4  for $i = 1$ to $\ell$ do
5      while there exists a $\lambda$-sparse cut (defined in Equation (4)) in $G_i$ do
6          Remove all edges (in both directions) in this cut and store them in $\mathrm{sk}(G_i)$.
7      In $\mathrm{sk}(G_i)$, store the (dense) components $\{V_{ij}\}_j$ of $G_i$.
8      For every $V_{ij}$ and every $u \in V_{ij}$, store the number of (remaining) incoming and
         outgoing edges at $u$ in $G_i$, i.e., $d^{\mathrm{in}}_{ij}(u) = |E_i(V_{ij}, u)|$ and $d^{\mathrm{out}}_{ij}(u) = |E_i(u, V_{ij})|$.
9      At each vertex $u \in V$, sample with replacement $\alpha$ edges from the (remaining)
         outgoing edges $(u, v)$ and store them in $\mathrm{sk}(G_i)$. Do the same for incoming edges.
10 return $\mathrm{sk}(G) = \bigcup_i \mathrm{sk}(G_i)$.
```

Algorithm 3 Query the cut value $w(S, \overline{S})$ from $\mathrm{sk}(G)$.

```
Input  : A cut query $S \subseteq V$ and a cut sketch $\mathrm{sk}(G)$ (output of Algorithm 2).
1  for each $\mathrm{sk}(G_i)$ in $\mathrm{sk}(G)$ do
2      for each dense component $V_{ij}$ in $G_i$ do
3          Let $S_{ij}$ denote the smaller set of $(V_{ij} \cap S)$ and $(V_{ij} \cap \overline{S})$.
4          if $S_{ij} = V_{ij} \cap S$ then
5              Estimate the total weight of edges leaving $S_{ij}$: For every $u \in S_{ij}$, set

                   $$I_{S_{ij}}(u) = \frac{d^{\mathrm{out}}_{ij}(u)}{\alpha} \sum_{q=1}^{\alpha} \chi_{ij}(u,q) w_{ij}(u,q) \qquad (5)$$

               where $d^{\mathrm{out}}_{ij}(u)$ is the out-degree of $u$ in $V_{ij}$, $\chi_{ij}(u,q) = 1$ if the $q$-th
               sampled outgoing edge at $u$ crosses $S$ and $\chi_{ij}(u,q) = 0$ otherwise, and
               $w_{ij}(u,q)$ is the weight of the $q$-th sampled edge.
6          else
7              Estimate the total weight of edges entering $S_{ij} = V_{ij} \cap \overline{S}$ instead:
8              For every $u \in S_{ij}$, set $I_{S_{ij}}(u)$ as in (5), using $d^{\mathrm{in}}_{ij}(u)$ instead of $d^{\mathrm{out}}_{ij}(u)$, and
                 $\chi_{ij}(u,q)$ indicates if the $q$-th sampled incoming edge at $u$ crosses $S$.
9          The estimated contribution from $V_{ij}$ is $I_{V_{ij}} = \sum_{u \in S_{ij}} I_{S_{ij}}(u)$.
10     The estimated contribution from $G_i$ is $I_{G_i} = \sum_{V_{ij} \in G_i} I_{V_{ij}}$.
11 Compute $I_S = \sum_i I_{G_i}$, the estimate of the cut value from all dense-component edges.
12 Compute $J_S$, the total weight of $\lambda$-sparse cut edges that leaves $S$ in all $G_i$'s.
13 return $I_S + J_S$.
```

▶ **Lemma 21** (Output Size of Algorithm 2). *The output* $\mathrm{sk}(G)$ *of Algorithm 2 has size* $\widetilde{O}(n(\lambda+\alpha)) = \widetilde{O}(n\beta^{1/2}/\epsilon)$.

Proof. Without loss of generality, we assume $\epsilon = \Omega(1/n)$, otherwise we can store all edges exactly using $\widetilde{O}(n/\epsilon)$ bits. Algorithm 2 produces $\ell = O(\log n)$ weight classes; each weight class defines a graph G_i. In every G_i:

- First we iteratively store and remove edges in λ-sparse cuts. We can upper bound the total number of edge removed using the following charging argument: When a λ-sparse cut is removed, we charge the cut size evenly to the vertices on the smaller side of the cut. Since the cut is λ-sparse, every vertex on the smaller side gets charged at most λ edges. Each vertex can be charged at most $O(\log n)$ times because it can be in the smaller side $O(\log n)$ times. Therefore, $\mathrm{sk}(G)$ stores at most $O(\lambda n \log n)$ sparse edges, which takes $O(\lambda n \log^2 n)$ bits.
- On the remaining graph, the connected components are disjoint, so we can also store the partition of vertices into these dense components in $O(n \log n)$ bits.
- We can store the (remaining) in- and out-degree of every vertex in $O(n \log n)$ bits.
- We sample $O(\alpha)$ edges at each vertex in V, which requires $O(\alpha n \log n)$ bits.

Thus, for every G_i we store $O(\lambda n \log^2 n + n\log n + \alpha n \log n) = O(n(\lambda+\alpha)\log^2 n)$ bits. Since $\alpha = \lambda = \beta^{1/2}/\epsilon$, the size of $\mathrm{sk}(G_i)$ is $O(n\beta^{1/2}\log^2 n/\epsilon)$. The size of $\mathrm{sk}(G) = \bigcup_i \mathrm{sk}(G_i)$ is

$$O(\log n)\cdot O(n\beta^{1/2}\log^2 n/\epsilon) = O(n\beta^{1/2}\log^3 n/\epsilon). \qquad ◀$$

Now we prove Lemma 20 (the correctness of Algorithm 3). Note that it follows immediately from the following lemma and Chebyshev's inequality.

▶ **Lemma 22.** *The estimator returned in Algorithm 3 is unbiased, i.e.,* $\mathbb{E}[I_S] + J_S = w(S,\overline{S})$. *Moreover, the variance of* I_S *is* $\mathrm{Var}[I_S] \le O(\beta/\alpha\lambda) w(S,\overline{S})^2$.

Proof of Lemma 20. Algorithm 2 sets $\alpha = \lambda = \beta^{1/2}\epsilon^{-1}$, so Lemma 22 implies $\mathrm{Var}[I_S] \le O(\epsilon^2)\cdot w(S,\overline{S})^2$. By Chebyshev's inequality, with probability at least 2/3,

$$\left|(I_S + J_S) - w(S,\overline{S})\right| \le O(\epsilon)\cdot w(S,\overline{S}). \qquad ◀$$

To prove Theorem 19, all that remains is to prove Lemma 22.

Proof of Lemma 22. Recall that our estimator is

$$I_S = \sum_{G_i}\sum_{V_{ij}}\sum_{u\in S_{ij}} I_{S_{ij}}(u),$$

where i sums over the graphs G_i defined according to the edge weights, j sums over the dense components in each G_i after all sparse cuts are removed, and u sums over the vertices of S_{ij}. Without loss of generality, we can assume $|V_{ij}\cap S| \le |V_{ij}\cap \overline{S}|$ and hence $S_{ij} = V_{ij}\cap S$. Otherwise, Algorithm 3 works with $V_{ij}\cap\overline{S}$ and queries for the incoming edges instead. Under this assumption, we always work with outgoing edges:

$$I_{S_{ij}}(u) = \frac{d_{ij}^{\mathrm{out}}(u)}{\alpha}\sum_{q=1}^{\alpha}\chi_{ij}(u,q)w_{ij}(u,q).$$

Every edge $e \in E(S,\overline{S})$ belongs to exactly one G_i, and in that G_i it is either a sparse-cut edge, or a dense-component edge in exactly one V_{ij}. Consequently, to prove $I_S + J_S$ is unbiased, it suffices to prove that $I_{S_{ij}}(u)$ is unbiased. In the dense component V_{ij} of G_i, the total contribution of edges leaving u to $w(S,\overline{S})$ is $\sum_{r=1}^{d_{ij}^{\text{out}}(u)} \chi(u,r) \cdot w(u,r)$, where r indexes the edges leaving u, $w(u,r)$ is the weight of the r-th edge leaving u, and $\chi(u,r)$ indicates if this edge goes from S to $\overline{S}$. Let $e(u,r)$ denote the r-th edge leaving u and $e_{ij}(u,q)$ denote the q-th sampled edge leaving u within V_{ij}. Summing over the α sampled edges, we have

$$\begin{aligned}
\mathbb{E}\left[I_{S_{ij}}(u)\right] &= \frac{d_{ij}^{\text{out}}(u)}{\alpha} \cdot \sum_{q=1}^{\alpha} \mathbb{E}\left[\chi_{ij}(u,q) \cdot w_{ij}(u,q)\right] \\
&= \frac{d_{ij}^{\text{out}}(u)}{\alpha} \cdot \sum_{q=1}^{\alpha} \sum_{r=1}^{d_{ij}^{\text{out}}(u)} \Pr[e_{ij}(u,q) = e(u,r)] \cdot \chi(u,r) \cdot w(u,r) \\
&= \sum_{r=1}^{d_{ij}^{\text{out}}(u)} \chi(u,r) \cdot w(u,r),
\end{aligned}$$

where the last equality holds because each sample has the same variance, and the q-th sample is drawn uniformly among all outgoing edges at u, i.e., $\Pr[e_{ij}(u,q) = e(u,r)] = \frac{1}{d_{ij}^{\text{out}}(u)}$. The expectation of $I_{S_{ij}}(u)$ is exactly the contribution of edges leaving u to $w(S,\overline{S})$, so $I_{S_{ij}}(u)$ is unbiased.

For the rest of the proof, we upper bound the variance of I_S. We assume without loss of generality that $J_S = 0$, i.e., no sparse edges were ever stored and removed by Algorithm 2. This is because we are trying to prove the statement

$$\operatorname{Var}\left[I_S\right] \le O\left(\frac{\beta}{\alpha\lambda}\right) w(S,\overline{S})^2 = O\left(\frac{\beta}{\alpha\lambda}\right) \cdot \left(\mathbb{E}\left[I_S\right] + J_S\right)^2,$$

so setting $J_S = 0$ only makes the right-hand side smaller and hence, the proof more difficult.

We introduce some notation: recall that $(V_{ij})_j$ is the set of dense components of G_i and $S_{ij} = V_{ij} \cap S$. We use $X_{ij} = |E_i(S_{ij}, \overline{S_{ij}})|$ to denote the *number* of edges from S_{ij} to $V_{ij} \cap \overline{S}$ in G_i, and $\overline{X_{ij}} = |E_i(\overline{S_{ij}}, S_{ij})|$ the number of edges in the reverse direction. Let $X_i = \sum_j X_{ij}$ and $\overline{X_i} = \sum_j \overline{X_{ij}}$, so that X_i is the total number of dense-component edges that go from S to $\overline{S}$ in G_i.

Since there are no λ-sparse cuts (defined in (4)) at the end of Algorithm 2, we have $X_{ij} + \overline{X_{ij}} > \lambda \min(|S_{ij}|, |\overline{S_{ij}}|)$. Since we assume $|S_{ij}| \le |\overline{S_{ij}}|$, this condition implies the following for every dense component V_{ij}, $\lambda\left|S_{ij}\right| \le X_{ij} + \overline{X_{ij}}$.

Fix any $u \in S_{ij}$. We first upper bound $\operatorname{Var}\left[I_{S_{ij}}(u)\right]$ and then work our way up the definition of I_S. Since $\chi_{ij}(u,q)$ is a Bernoulli random variable with mean $\left|E_i(u, \overline{S_{ij}})\right| / d_{ij}^{\text{out}}(u)$, its variance is

$$\operatorname{Var}\left[\chi_{ij}(u,q)\right] = \frac{\left|E_i(u, \overline{S_{ij}})\right|}{d_{ij}^{\text{out}}(u)} \cdot \frac{\left|E_i(u, S_{ij})\right|}{d_{ij}^{\text{out}}(u)}.$$

Now from the definition of $I_{S_{ij}}(u)$, we have

$$\begin{aligned}
\operatorname{Var}\left[I_{S_{ij}}(u)\right] &= \frac{d_{ij}^{\text{out}}(u)^2}{\alpha^2}\sum_{q=1}^{\alpha}\operatorname{Var}\left[\chi_{ij}(u,q)\right]w_{ij}(u,q)^2 \\
&\le \frac{d_{ij}^{\text{out}}(u)^2}{\alpha^2}\cdot\alpha\cdot\frac{\left|E_i(u,\overline{S_{ij}})\right|}{d_{ij}^{\text{out}}(u)}\cdot\frac{\left|E_i(u,S_{ij})\right|}{d_{ij}^{\text{out}}(u)}\cdot 2^{2i} && (w_e \le 2^i \text{ for all } e\in E_i) \\
&= \frac{2^{2i}}{\alpha}\left|E_i(u,\overline{S_{ij}})\right|\cdot\left|E_i(u,S_{ij})\right| \\
&\le \frac{2^{2i}}{\alpha}\left|E_i(u,\overline{S_{ij}})\right|\cdot\left|S_{ij}\right|. && (\left|E_i(u,S_{ij})\right|\le|S_{ij}|)
\end{aligned}$$

In Algorithm 3, we set $I_{V_{ij}} = \sum_{u\in S_{ij}} I_{S_{ij}}(u)$, so

$$\begin{aligned}
\operatorname{Var}\left[I_{V_{ij}}\right] = \sum_{u\in S_{ij}}\operatorname{Var}\left[I_{S_{ij}}(u)\right] &\le \sum_{u\in S_{ij}}\frac{2^{2i}}{\alpha}\left|S_{ij}\right|\cdot\left|E_i(u,\overline{S_{ij}})\right| \\
&= \frac{2^{2i}}{\alpha}\cdot\left|S_{ij}\right|\cdot X_{ij} && (X_{ij} = \left|E_i(S_{ij},\overline{S_{ij}})\right|) \\
&\le \frac{2^{2i}}{\alpha\lambda}\left(X_{ij}+\overline{X_{ij}}\right)X_{ij}. && ((S_{ij},\overline{S_{ij}}) \text{ is not } \lambda\text{-sparse})
\end{aligned}$$

Summing across every dense component V_{ij} in G_i, we get

$$\begin{aligned}
\operatorname{Var}\left[I_{G_i}\right] = \sum_j \operatorname{Var}\left[I_{V_{ij}}\right] &\le \frac{2^{2i}}{\alpha\lambda}\sum_j\left(X_{ij}+\overline{X_{ij}}\right)X_{ij} \\
&\le \frac{2^{2i}}{\alpha\lambda}\left(X_i+\overline{X}_i\right)X_i. && (X_i = \textstyle\sum_j X_{ij} \text{ and } \overline{X_i} = \sum_j \overline{X_{ij}})
\end{aligned}$$

Finally, we sum across the weight classes indexed by i to obtain

$$\begin{aligned}
\operatorname{Var}\left[I_S\right] &= \sum_i \operatorname{Var}\left[I_{G_i}\right] \\
&\le \frac{1}{\alpha\lambda}\sum_i\left(2^i\right)^2\left(X_i+\overline{X}_i\right)X_i \\
&= \frac{1}{\alpha\lambda}\left[\sum_i\left(2^iX_i\right)^2 + \sum_i 2^i\overline{X}_i\cdot\sum_i 2^iX_i\right] \\
&\le \frac{4}{\alpha\lambda}\left[w(S,\overline{S})^2 + w(\overline{S},S)\cdot w(S,\overline{S})\right] && (w_e \ge 2^{i-1} \text{ for all } e\in E_i) \\
&\le \frac{4}{\alpha\lambda}\left[w(S,\overline{S})^2 + \beta w(S,\overline{S})^2\right] && (w(\overline{S},S)\le\beta w(S,\overline{S})) \\
&= O\left(\frac{\beta}{\alpha\lambda}\right)w(S,\overline{S})^2.
\end{aligned}$$

◀

5.2 For-Each Cut Sketch: Faster Algorithms

We can speed up the algorithms in the previous section and prove the following theorem.

▶ **Theorem 23.** *Consider the same setting as in Theorem 19. That is, a $(1\pm\epsilon)$ for-each cut sketch of size $O(\beta^{1/2}n\log^5 n/\epsilon)$ bits exists for any n-vertex β-balanced graph G. Now in addition, we can compute such a cut sketch in time $\widetilde{O}(m+\beta^{1/2}n/\epsilon)$.*

For undirected graphs, Jambulapati and Sidford [21] showed how to construct for-each cut sketches in nearly-linear time. Instead of trying to repeatedly find sparse cuts, they showed how to sketch expander graphs (graphs with high conductance) and then decompose the input graph using expander partitioning algorithms.

Intuitively, we should be able to speed up our algorithm using a similar approach, because when we partition the graph by removing sparse cuts, we do not look at the direction of the edges. However, the analysis in [21] does not apply to our setting because they focus on sketching quadratic forms. The quadratic form of a directed Laplacian ignores edge directions and hence does not preserve the directed cut values. On a more technical level, their analysis relies heavily on the notion of conductance, which is not canonically defined for directed graphs. In our setting, we cannot bound the variance of our estimator even if we have a directed graph whose undirected version is an expander (see the full paper for more details).

Because our main focus in this paper is to derive cut sketches with *optimal size* (especially with a tight dependence on β) rather than to obtain best runtime of computing such cut sketches, we defer the proof of Theorem 23 to the full paper.

6 For-Each Cut Sketch: $\Omega(n \cdot \sqrt{\beta/\epsilon})$ Lower Bound

In this section, we prove that the size of for-each cut sketches must scale with $\sqrt{\beta}$.

▶ **Theorem 24.** *Fix $\beta \geq 1$ and $0 < \epsilon < 1$ with $\left(\beta/\epsilon\right)^{1/2} \leq \frac{n}{2}$. A $(1 \pm \epsilon)$ for-each cut sketching algorithm for n-node β-balanced graphs must output $\Omega(n \cdot (\beta/\epsilon)^{1/2})$ bits in the worst case.*

To prove this, we will need the following folklore result from communication complexity:

▶ **Lemma 25.** *Given a bit string $s \in \{0,1\}^N$, if there is a data structure D that allows one to recover each bit of s with marginal probability at least $2/3$, then D must use $\Omega(N)$ bits.*

We first prove a special case of our lower bound for specific values of $\beta = \Theta(n^2)$ and $\epsilon = \Theta(1)$ (Lemma 26). The proof for this special case is easier to explain and it contains the key ingredients of our construction for the general lower bound.

▶ **Lemma 26.** *For $\beta = n^2$ and $\epsilon = \frac{1}{10}$, any $(1 \pm \epsilon)$ for-each cut sketching algorithm for n-node β-balanced graphs must output $\Omega(n \cdot \beta^{1/2})$ bits in the worst case.*

Proof. At a high level, we will encode a bit string s of length $\Omega(n^2)$ into an n-node β-balanced graph, such that given a $(1 \pm \epsilon)$ for-each cut sketch, we can recover each bit of s with high constant probability. Then by Lemma 25, the cut sketch must have use $\Omega(|s|) = \Omega(n^2) = \Omega(n\beta^{1/2})$ bits.

Given a bit string s of length $\frac{n^2}{4}$, we construct a graph as follows. We start with an $\frac{n}{2} \times \frac{n}{2}$ complete bipartite digraph where edges go from left to right. We set the weight of the i-th bipartite edge to $s_i + 1$ (so either 1 or 2). We add a unit-weight cycle that leaves each side exactly once. See Fig. 2 for an example of our construction.

We first show that the graph is β-balanced for $\beta = n^2$. The graph is strongly connected because it contains a cycle. Note that all edge weights are in $[1, 2]$ and there are in total $\frac{n^2}{4} + n \leq \frac{n^2}{2}$ edges in the graph. Therefore, for every non-empty set $S \subset V$, the total weight of edges leaving (or entering) S is at least 1 and at most n^2, so the graph is (n^2)-balanced.

It remains to show that we can recover each bit of s from a $(1 \pm \epsilon)$ cut sketch. Let L denote the left vertices and R the right vertices. Fix any coordinate of s and suppose it corresponds to the edge (u, v) for some $u \in L$ and $v \in R$. To recover this bit of s, we need to decide whether $w(u, v)$ is 1 or 2. Consider the cut value leaving $S = \{u\} \cup R \setminus \{v\}$. The cycle contributes a fixed amount to this cut (independent of the weights of the bipartite

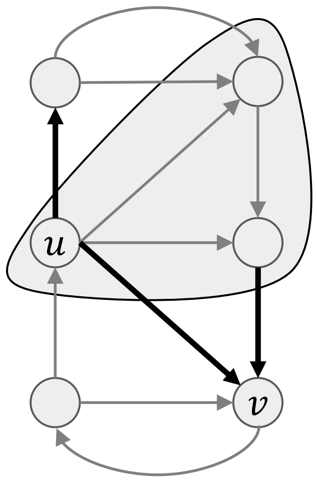

Figure 2 In this example, the cut value $w(S, \overline{S})$ is 2 or 3, depending on the value of $w(u, v)$. Thus, a (1 ± 0.1)-approximation to $w(S, \overline{S})$ allows us to decode the bit in s corresponding to edge (u, v). (For readability we omit other bipartite edges from L to R.)

edges), which is at most 3. More importantly, (u, v) is the only bipartite edge leaving S. Since $\epsilon = \frac{1}{10}$ and the sketch returns this cut value within a factor of $(1 \pm \epsilon)$ with probability at least $2/3$, we can recover $w(u, v)$ with probability at least $2/3$. ◀

Our lower bound construction for general values of β and ϵ builds on the one in the proof of Lemma 26. At a high level, instead of using a bipartite graph with two clusters, we will use multiple clusters where the size of each cluster depends on β and ϵ.

Proof of Theorem 24. Let $k = \sqrt{\beta/\epsilon}$. We will encode a bit string s of length $\Omega(nk)$ into an n-node graph G such that (1) G is (3β)-balanced, and (2) we can recover each bit of s with high constant probability given a $(1 \pm c \cdot \epsilon)$ for-each cut sketch of G where $c = 10^{-2}$. By Lemma 25, the cut sketch must have at least $\Omega(nk) = \Omega(n \cdot (\beta/\epsilon)^{1/2})$ bits.

Without loss of generality, we assume k is an integer and n is a multiple of k. We partition n vertices into $t = n/k$ clusters of size k, which we denote by $V_1, \ldots, V_t$. Since we assume $n \geq 2k$, there are at least two clusters.

Let s be a bit string of length $k^2(t-1) = \Omega(nk)$. We partition s into $(t-1)$ blocks where each block has length k^2. We encode the i-th block of s in a $k \times k$ complete bipartite digraph where edges go from V_i to V_{i+1}. As in the proof of Lemma 26, a bipartite edge (u, v) for $u \in V_i$ and $v \in V_{i+1}$ has weight $s_{i,(u,v)} + 1$ (so either 1 or 2). For every $1 \leq i \leq t-1$, we add a cycle between V_i and V_{i+1} that leaves V_i and V_{i+1} exactly once. Now, in contrast to the previous construction, these cycle edges have weight $1/\epsilon$.

We first show that G is (3β)-balanced. Fix any non-empty set $S \subset V$. Let G_i denote the subgraph between V_i and V_{i+1} which contains k^2 bipartite edges and one cycle. Let $w_i(S, \overline{S})$ denote the total weight of edges leaving S in G_i. We will show that $w_i(S, \overline{S})$ and $w_i(S, \overline{S})$ are within a factor of 3β of each other. Because G is strongly connected and $w(S, \overline{S}) = \sum_{i=1}^{t-1} w_i(S, \overline{S})$, we can conclude that G is (3β)-balanced.

Without loss of generality, we assume both $w_i(S, \overline{S})$ and $w_i(\overline{S}, S)$ are positive. The cut value $w_i(S, \overline{S})$ remains the same if we restrict G_i on vertices $(V_i \cup V_{i+1})$ and consider the cut query $S \cap (V_i \cup V_{i+1})$. The cycle contributes equally in both directions, so without loss of generality, we can assume the cycle has minimum contribution, which is $\frac{1}{\epsilon}$. (If the cycle contributes more, the cut is more balanced.) The total weight of the bipartite edges is at most $2k^2 = \frac{2\beta}{\epsilon}$. Therefore, the ratio between the cut values in both directions is at most $\frac{(2\beta/\epsilon)+(1/\epsilon)}{1/\epsilon} = 2\beta + 1 \leq 3\beta$.

It remains to show that we can recover every bit of s from a cut sketch. Fix any bit of s. Suppose this bit $s_{i,(u,v)}$ corresponds to the edge (u,v) for some $u \in V_i$ and $v \in V_{i+1}$, we query the cut value leaving $S_{(u,v)} = \{u\} \cup \left(V_{i+1} \setminus \{v\}\right) \bigcup_{j=i+2}^{t-1} V_j$. The only bipartite edge leaving $S_{(u,v)}$ is the edge (u,v), which has weight either 1 or 2. There are at most 5 cycle edges leaving $S_{(u,v)}$ (at most 1 from G_{i-1}, 3 from G_i, and 1 from G_{i+1}), whose total weight is fixed and at most $\frac{5}{\epsilon}$. Therefore, if we can compute an $(1 \pm c \cdot \epsilon)$ approximation to the cut value for $c = 10^{-2}$, we can recover the corresponding bit of s. ◀

7 Conclusion

In this paper, we considered the question of sparsifying directed graphs. We focused on graphs that are β-balanced, where the ratio between the cut value in two directions is at most β. We gave upper and lower bounds on the size of the cut sketch with almost tight dependence on β, under both the standard "for-all" notion (i.e., simultaneously preserving the value of all cuts) and the "for-each" notion (introduced by Andoni *et. al* [6]) of cut sparsification. More specifically, we showed that under the "for-all" notion, the linear dependence on β obtained by Ikeda and Tanigawa [20] is tight. For the "for-each" notion, we gave a data structure that preserves cut values whose size scales as $\sqrt{\beta}$, thereby beating the "for-all" lower bound. We also showed that this dependence on $\sqrt{\beta}$ is tight. Our lower bounds hold not only for sparsifiers (i.e., graph encodings), but also for arbitrary data structures.

An interesting direction for future work is to consider the *spectral* sparsification of directed graphs. Cohen et al. [14, 13] (see also Chu et al. [11]) introduced a novel definition of directed sparsification and leveraged it to solve directed Laplacian linear systems. However, their work is not immediately relevant to ours because their directed spectral sparsifiers do not necessarily preserve directed cut values. This motivates the following natural question: *is there a notion of spectral sparsification that generalizes cut sparsification in directed graphs*? (Note that this is indeed the case for undirected graphs, where spectral sparsifiers also preserve cut values.) A natural candidate would be a sparse graph that preserves $\sum_{(u,v)\in E} \left((x_u - x_v)^+\right)^2$ for all real vectors x, where $y^+ = \max(0,y)$. Note that if $x \in \{0,1\}^{|V|}$, then this sum represents directed cut values, which is analogous to the correspondence between cut and spectral sparsification in undirected graphs. It would be interesting to explore if preserving this sum in directed graphs has interesting applications beyond preserving cuts, and if so, whether there exist sparse graphs that preserve this sum approximately for balanced directed graphs.

References

1 Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 328–338. Springer, 2009.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems*, pages 5–14, 2012.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *Proceedings of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and the 17th International Workshop on Randomization and Computation*, pages 1–10, 2013.

5 Zeyuan Allen Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. In *Proceedings of the 47th ACM Symposium on Theory of Computing*, pages 237–245, 2015.

6 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 311–319. ACM, 2016.

7 Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

8 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.

9 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The DAG-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.

10 Charles Carlson, Alexandra Kolla, Nikhil Srivastava, and Luca Trevisan. Optimal lower bounds for sketching graph cuts. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms*, pages 2565–2569, 2019.

11 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 361–372, 2018.

12 Fan Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

13 Michael B. Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed Laplacian systems in nearly-linear time through sparse LU factorizations. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science*, pages 898–909, 2018.

14 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th ACM Symposium on Theory of Computing*, pages 410–419, 2017.

15 Alina Ene, Gary Miller, Jakub Pachocki, and Aaron Sidford. Routing under balance. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 598–611. ACM, 2016.

16 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

17 Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 71–80. ACM, 2011.

18 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Graph sparsification via refinement sampling. *CoRR*, abs/1004.4915, 2010. `arXiv:1004.4915`.

19 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008.

20 Motoki Ikeda and Shin-ichi Tanigawa. Cut sparsifiers for balanced digraphs. In *International Workshop on Approximation and Online Algorithms*, pages 277–294. Springer, 2018.

21 Arun Jambulapati and Aaron Sidford. Efficient $\widetilde{O}(n/\epsilon)$ spectral sketches for the Laplacian and its pseudoinverse. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503, 2018.

22 Thor Johnson, Neil Robertson, Paul D Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

23 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.

24 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, and Navid Nouri. Faster spectral sparsification in dynamic streams. *CoRR*, abs/1903.12165, 2019. `arXiv:1903.12165`.

25 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`.

26 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 393–398, 2012.

27 David R Karger and Matthew S Levine. Random sampling in residual graphs. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 63–66. ACM, 2002.

28 David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

29 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.

30 Ioannis Koutis, Alex Levin, and Richard Peng. Improved spectral sparsification and numerical algorithms for SDD matrices. In *29th International Symposium on Theoretical Aspects of Computer Science*, pages 266–277, 2012.

31 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science*, pages 235–244, 2010.

32 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science*, pages 590–598, 2011.

33 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified Cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, pages 842–850, 2016.

34 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science*, pages 250–269, 2015.

35 Yin Tat Lee and He Sun. An SDP-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th ACM Symposium on Theory of Computing*, pages 678–687, 2017.

36 Henry Lin. Reducing directed max flow to undirected max flow. *Unpublished Manuscript*, 2009.

37 Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families II: Mixed characteristic polynomials and the kadison—singer problem. *Annals of Mathematics*, pages 327–350, 2015.

38 Ilan Newman and Yuri Rabinovich. On multiplicative Lambda-approximations and some geometric applications. *SIAM J. Comput.*, 42(3):855–883, 2013.

39 Jan Obdržálek. DAG-width: connectivity measure for directed graphs. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 814–821, 2006.

40 Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 263–269, 2013.

41 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 2570–2581, 2019.

42 Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

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

44 Ying Zhang, Zhiqiang Zhao, and Zhuo Feng. Towards scalable spectral sparsification of directed graphs. In *Proceedings of the 15th IEEE International Conference on Embedded Software and Systems*, pages 1–2, 2019.