

# Correlation Clustering and (De)Sparsification: Graph Sketches Can Match Classical Algorithms

Sepehr Assadi\*  
University of Waterloo

Sanjeev Khanna†  
University of Pennsylvania

Aaron Putterman‡  
Harvard University

## Abstract

Correlation clustering is a widely-used approach for clustering large data sets based only on pairwise similarity information. In recent years, there has been a steady stream of better and better classical algorithms for approximating this problem. Meanwhile, another line of research has focused on porting the classical advances to various sublinear algorithm models, including semi-streaming, Massively Parallel Computation (MPC), and distributed computing. Yet, these latter works typically rely on ad-hoc approaches that do not necessarily keep up with advances in approximation ratios achieved by classical algorithms. Hence, the motivating question for our work is this: can the gains made by classical algorithms for correlation clustering be ported over to sublinear algorithms in a *black-box manner*? We answer this question in the affirmative by introducing the paradigm of graph de-sparsification.

A versatile approach for designing sublinear algorithms across various models is the graph (linear) sketching. It is known that one can find a cut sparsifier of a given graph—which approximately preserves cut structures—via graph sketching, and that this is sufficient information-theoretically for recovering a near-optimal correlation clustering solution. However, no efficient algorithms are known for this task as the resulting cut sparsifier is necessarily a weighted graph, and correlation clustering is known to be a distinctly harder problem on weighted graphs.

Our main result is a randomized linear sketch of  $\tilde{O}(n)$  size for  $n$ -vertex graphs, from which one can recover with high probability an  $(\alpha + o(1))$ -approximate correlation clustering in polynomial time, where  $\alpha$  is the best approximation ratio of any polynomial time classical algorithm for (unweighted) correlation clustering. This is proved via our new de-sparsification result: we recover in *polynomial-time* from some  $\tilde{O}(n)$  size linear sketch of a graph  $G$ , an *unweighted*, simple graph that approximately preserves the cut structure of  $G$ . In fact we show that under some mild conditions, *any* spectral sparsifier of a graph  $G$  can be de-sparsified into an unweighted simple graph with nearly the same spectrum. We believe the de-sparsification paradigm is interesting in its own right as a way of reducing graph complexity when weighted version of a problem is harder than its unweighted version.

Finally, we use our techniques to get efficient algorithms for correlation clustering that match the performance of best classical algorithms, in a variety of different models, including dynamic streaming, MPC, and distributed communication models.

---

\*Supported in part by a Sloan Research Fellowship, an NSERC Discovery Grant (RGPIN-2024-04290) and a Faculty of Math Research Chair grant. (sepehr@assadi.info).

†Research supported in part by NSF grants CCF-2008305, CCF-2402284 (sanjeev@cis.upenn.edu).

‡Supported in part by the Simons Investigator Awards of Madhu Sudan and Salil Vadhan and NSF Award CCF-2152413 (aputterman@g.harvard.edu).

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Correlation Clustering and Graph Sketches . . . . .	1
1.2	A New Question: Graph Simplification via de-sparsification? . . . . .	2
1.3	Implication to Sublinear Algorithms . . . . .	4
1.4	Our Techniques . . . . .	5
1.5	Related Work . . . . .	6
<b>2</b>	<b>Preliminaries</b>	<b>6</b>
2.1	Cut and Spectral Sparsifiers . . . . .	7
2.2	Graph Sketches . . . . .	8
<b>3</b>	<b>Desparsification for Correlation Clustering: Proof of Result 1</b>	<b>8</b>
3.1	Correlation Clustering from Total Weight Preserving Sparsifiers . . . . .	8
3.2	Building the Linear Sketch used in <a href="#">Theorem 2</a> . . . . .	9
3.3	Recovering Fractional Total Weight-preserving Sparsifiers . . . . .	10
3.4	Rounding the Fractional Sparsifier . . . . .	12
3.5	Concluding the Proof of <a href="#">Theorem 2</a> . . . . .	13
<b>4</b>	<b>De-sparsifying Spectral Sparsifiers: Proofs of Result 2 and 3</b>	<b>14</b>
4.1	De-sparsifying With Small Effective Resistances . . . . .	14
4.2	Proof of <a href="#">Theorem 3</a> . . . . .	15
<b>5</b>	<b>Sublinear Algorithms</b>	<b>17</b>
5.1	Sublinear Algorithms Models . . . . .	17
5.2	Distributed Communication Model . . . . .	18
5.3	MPC Algorithms . . . . .	18
5.4	Dynamic Streaming . . . . .	19
5.5	Deterministic Algorithms for Insertion-Only Streams . . . . .	19
<b>6</b>	<b>Acknowledgements</b>	<b>23</b>
<b>A</b>	<b>Omitted Proofs</b>	<b>29</b>
A.1	Proof of <a href="#">Lemma 3.1</a> . . . . .	29
A.2	NP-Hardness of Certifying Cut Sparsification . . . . .	30

# 1 Introduction

Correlation clustering is a widely studied problem in theoretical computer science with applications to various areas. Given an undirected graph  $G = (V, E)$ , the goal is to cluster the vertices in a way that minimizes the cost, defined as the number of edges between the clusters and the number of non-edges<sup>1</sup> inside the clusters. We present a new approach for solving correlation clustering via graph sketching with approximation guarantees that can match the performance of *any* polynomial time algorithm for this problem. This immediately leads to improved algorithms for this problem across different sublinear algorithms models for processing massive graphs. The core to our approach is a new problem of its own independent interest: how do we de-sparsify an already sparsified graph in an efficient manner? We now elaborate more on our results and their context.

## 1.1 Correlation Clustering and Graph Sketches

Motivated by applications to processing massive graphs, there has been a rapidly growing interest in algorithms for correlation clustering across various *sublinear* algorithms models such as semi-streaming, Massively Parallel Computation (MPC), distributed computing, and alike. The key challenge in these models is that the resources available to the algorithm, say, its space or communication, is much smaller than the input size. Thus, the algorithms often need to ‘compress’ or ‘sparsify’ the input graphs before they are able to solve the problem in these models.

A highly successful paradigm here, especially when it comes to flexibility and portability across different models, is *graph sketching* pioneered by [AGM12a]: one compresses the graph into a sketch through a small number of linear measurements (say, of its adjacency or Laplacian matrix) and then solve the original problem given only this sketch (see [Definition 2.7](#)). Graph sketching has been quite successful for various graph problems including edge connectivity [AGM12a], vertex connectivity [AS23], cut sparsification [AGM12b], spectral sparsification [KLM<sup>+</sup>14a], densest subgraph [MTVV15], maximum matchings [AKLY16], and subgraph counting [AGM12b], among many others. This leads to the following natural question:

*Can we approximate correlation clustering via graph sketching?*

In some sense, this question was already settled in [BCMT23] (building on [ACG<sup>+</sup>15]): it turns out the cost of any clustering can be specified as a sum of cut sizes of the clusters plus some normalization; as such, to preserve (near-)optimal correlation clusterings, it suffices to preserve cut values of the graph in the sketch. But this latter task is precisely the goal of *cut sparsifiers* [BK96], which are weighted subgraphs of the input with only  $\tilde{O}(n/\varepsilon^2)$  edges that preserve the value of every cut to within a  $(1 \pm \varepsilon)$  factor, and already admit efficient graph sketches [AGM12b]. Thus, we can also find  $(1 + \varepsilon)$ -approximate correlation clusterings using graph sketching.

There is however a serious caveat with this approach: while information-theoretically we can recover a near-optimal solution from the sparsifier, we do not know how to do this in polynomial-time. In particular, recovering any solution from the sparsifier essentially amounts to solving a *weighted* version of correlation clustering (given that sparsification necessarily generates a weighted graph in general), which currently only admits an  $O(\log n)$ -approximation [DEFI06]<sup>2</sup>. Thus, when it comes to polynomial time algorithms, the above approach appears to hit a dead end.

As a result of the above shortcoming, recent work has come up with different graph sketches, or more often even entirely different techniques, for solving correlation clustering in this context;

---

<sup>1</sup>By a non-edge, we mean a pair of vertices with no edges between them in the graph.

<sup>2</sup>In fact, it is shown by [DEFI06] that weighted correlation clustering is equivalent to the minimum multicut problem and is thus difficult to approximate better than a  $\Theta(\log n)$  factor, and does not admit any constant factor approximation under the Unique Games Conjecture.

these results mostly collect “enough” information from the graph through the compression so as to simulate a *specific* classical algorithm (often, the pivot algorithm of [ACN08] but also recently more improved combinatorial algorithms in [CLP<sup>+</sup>24]). These approaches then lead to a host of different sublinear algorithms for this problem with different guarantees across many of these models; again, see Section 1.5 for a brief summary. However, this means that these techniques do not necessarily keep up with the improvements on classical algorithms on this problem—which have seen many exciting developments just recently; see Section 1.5—and rely on an ad hoc approach each time for porting these improvements to sublinear algorithms as well. Thus, we can ask a more nuanced version of our original question:

*Can we approximate correlation clustering via graph sketching in polynomial time, matching the approximation ratio of best polynomial-time classical algorithms?*

We show that the answer to this question is indeed *yes*. Let  $\alpha_{\text{BEST}}$  denote the best approximation ratio possible for correlation clustering in polynomial time (via classical algorithms), which satisfies

$$1.042 \underset{[\text{CCL}^{+}24]}{\leqslant} \alpha_{\text{BEST}} \underset{[\text{CCL}^{+}24]}{\leqslant} 1.437. \quad (1)$$

due to the APX-hardness established in [CGW03, CCL<sup>+</sup>24] and the recent approximation algorithm of [CCL<sup>+</sup>24].

**Result 1.** *For any  $n$ -vertex graph  $G$ , there is a randomized linear sketch of  $\tilde{O}(n)$  size from which one can recover with high probability an  $(\alpha_{\text{BEST}} + o(1))$ -approximate correlation clustering of  $G$  in polynomial time.*

We establish this result by introducing a new direction of research, termed *de-sparsification*, and then use it for our particular application to correlation clustering.

## 1.2 A New Question: Graph Simplification via de-sparsification?

Let us revisit the approaches of [ACG<sup>+</sup>15, BCMT23] that designed  $(1 + \varepsilon)$ -approximate graph sketches for correlation clustering (information-theoretically) via *weighted* cut sparsifiers. As stated earlier, the weights in the sparsifier forces us to solve a weighted correlation clustering instance which is a much harder version of the problem than the unweighted one. But what if these instances can be made unweighted<sup>3</sup> while preserving their correlation clustering structure? This will then allow us to run any approximation algorithm for unweighted correlation clustering on this instance and recover essentially the same approximation guarantee on the original graph.

To address this, we ask a general question that is entirely independent of correlation clustering:

*Can we efficiently de-sparsify a weighted cut sparsifier  $H$  to an unweighted, simple (but not necessarily sparse) graph  $G$  while (nearly) preserving the value of every cut?*

Two important remarks are in order: (a) firstly, the cut structure of weighted graphs is more general than unweighted ones, and thus in general, we cannot hope for approximating an arbitrary weighted graph with a (simple) unweighted graph; however, in our case, the weighted graph  $H$  is a sparsifier of an unweighted graph and thus certainly can be approximated by an unweighted

---

<sup>3</sup>Here, and throughout the paper, we use unweighted to also mean *simple* (i.e., that there are no multi-edges permitted).

graph; (b) secondly, information-theoretically it is impossible to recover the *original* graph from its weighted sparsifier due to the many-to-one nature of the sparsification; but here our goal is to find *some* graph that approximates the cut structure of  $H$  (and by extension the original graph), and hence, we do not run into this information-theoretic barrier. Note that there has been prior work on a topic called “densification” [HST12]. This line of work seeks to understand when weighted graphs *admit* dense sparsifiers. In our case, the question is not an existential one, but an algorithmic one. I.e., how do we *efficiently recover* these denser sparsifiers.

Unfortunately, we do not know a definitive answer to the above question in *this* formulation. It seems likely that the answer is *no* as it is even NP-hard to determine if a given weighted graph  $H$  is a cut sparsifier of a given graph  $G$  or not<sup>4</sup>. This suggests that preserving only the structure of the cuts may not allow for an efficient recovery/de-sparsification. But this then naturally suggests an alternative direction: what if we instead use *spectral sparsifiers* [ST11] that are generally known to be a robust strengthening of cut sparsifiers?<sup>5</sup>

Leveraging spectral sparsifiers, we obtain the following general de-sparsification result:

**Result 2.** *For any  $n$ -vertex unweighted graph  $G$  and any  $\varepsilon \in (0, 1)$ , there is a randomized linear sketch of  $\tilde{O}(n/\varepsilon^2)$  size from which one can recover in polynomial-time with high probability another  $n$ -vertex unweighted graph  $\tilde{G}$  with the same number of edges as  $G$  such that  $\tilde{G}$  is a  $(1 \pm \varepsilon)$ -spectral sparsifier<sup>6</sup> of  $G$ .*

---

<sup>6</sup>We note that using the term ‘sparsifier’ might be an abuse of notation here: graph  $\tilde{G}$  has the same exact number of edges as  $G$  and thus is not sparser than  $G$  in any way (nor is a subgraph of  $G$ ). We only use the term spectral sparsifier, here and throughout the paper, to mean that its spectrum is nearly the same as  $G$ .

As we will show later, **Result 2** turns out to be sufficient to obtain **Result 1** by simply setting  $\varepsilon = o(1)$ , and then running any  $\alpha$ -approximation correlation clustering algorithm on the graph  $\tilde{G}$ .

**Result 2** gives efficient de-sparsification for sparsifiers obtained by a particular linear sketching scheme. Specifically, it relies on the linear sketches for spectral sparsification given in [KLM<sup>+</sup>14a] that faithfully implement effective resistance-based sampling. One may ask the question if in fact any *arbitrary* spectral sparsifier can be efficiently de-sparsified, no matter how it was created. We show that the answer to this question is in the affirmative as well, assuming some mild conditions on the underlying graph.

**Result 3.** *For any  $\varepsilon \in (0, 1)$  and  $n$ -vertex unweighted graph  $G$ , there is a randomized polynomial-time algorithm that given any  $(1 \pm \varepsilon)$ -spectral sparsifier  $H$  of  $G$ , recovers with high probability*

- (a) *an  $n$ -vertex unweighted graph  $\tilde{G}$  with the same number of edges as  $G$  such that  $\tilde{G}$  is a  $(1 \pm 2\varepsilon)$ -cut sparsifier of  $G$ , provided the minimum cut in  $G$  is  $\Omega(\log n/\varepsilon^2)$ .*
- (b) *an  $n$ -vertex unweighted graph  $\tilde{G}$  with the same number of edges as  $G$  such that  $\tilde{G}$  is a  $(1 \pm 2\varepsilon)$ -spectral sparsifier of  $G$ , provided maximum effective resistance in  $G$  is  $O(\varepsilon^2/\log n)$ .*

We believe the de-sparsification question posed in this paper to be of its own independent interest specifically from the view point of *graph simplification*: while traditionally one often considers

---

<sup>4</sup>We suspect this result is folklore but do not know a reference for it and thus we prove it in [Appendix A.2](#).

<sup>5</sup>Formally, using  $L_G$  and  $L_H$  to denote the Laplacian matrix of  $G$  and  $H$ , respectively, a cut sparsifier  $H$  of  $G$  satisfies  $x^\top \cdot L_H \cdot x = (1 \pm \varepsilon) \cdot x^\top L_G \cdot x$  for all  $x \in \{0, 1\}^n$  whereas a spectral sparsifier satisfies the same for all  $x \in \mathbb{R}^n$ . Note that the evidence earlier no longer holds as checking if  $H$  is a spectral sparsifier of  $G$  boils down to checking if all singular values of  $L_H$  and  $L_G$  are within  $(1 \pm \varepsilon)$  factor of each other, which can be easily done in polynomial time.

simplifying a graph as making it sparser, it is also quite natural to simplify a graph by making it *unweighted* even at the cost of increasing its density (given that many problems are easier to solve on unweighted graphs such as correlation clustering considered here). Such simplification questions have also recently been considered for other graph problems in entirely different contexts, e.g., for preserving shortest path structures without using very large weights [BBW24], or for approximating weighted matching via unweighted matching algorithms [BDL21, BCD<sup>+</sup>25].

### 1.3 Implication to Sublinear Algorithms

Finally, we can use our efficient graph sketches for correlation clustering in [Result 1](#) to obtain new sublinear algorithms for this problem across a variety of different models, that can achieve approximation ratios nearly matching  $\alpha_{\text{BEST}}$  defined in [Eq \(1\)](#).

Our first algorithm is in the distributed communication model, studied for various clustering problems in [CSWZ16, ABB<sup>+</sup>19, ZZL<sup>+</sup>19] (although we are not aware of prior work on correlation clustering here). In this model, the input graph  $G = (V, E)$  is edge-partitioned across  $k$  machines plus a coordinator that receives no input. The machines and the coordinator can communicate in a distributed point-to-point manner. The goal is to limit the total communication while allowing the coordinator to output a correlation clustering of the entire input.

**Corollary 1.1.** *There is a polynomial-time randomized algorithm for correlation clustering in the distributed communication model with  $k$  machines that uses  $\tilde{O}(nk)$  communication in total, and with high probability, achieves an  $(\alpha_{\text{BEST}} + o(1))$ -approximation.*

The second algorithm is in the Massively Parallel Computation (MPC) model [KSV10, BKS17]. Here, the input graph  $G = (V, E)$  is edge-partitioned across multiple machines. Computation happens in synchronous rounds wherein each machine can send and receive  $\tilde{O}(n)$ -size messages. After the last round, one designated machine outputs a solution to the problem.

**Corollary 1.2.** *There is a polynomial-time randomized algorithm for correlation clustering in the MPC model that uses  $O(1)$  rounds and  $\tilde{O}(n)$ -size messages per-machine, and with high probability, achieves an  $(\alpha_{\text{BEST}} + o(1))$ -approximation.*

[Corollary 1.2](#) improves upon the 1.87-approximation MPC algorithm of [CLP<sup>+</sup>24] (given [Eq \(1\)](#)), although we note that the algorithm of [CLP<sup>+</sup>24] runs in  $O(1)$  rounds even when memory per machine is  $n^\delta$  for any constant  $\delta \in (0, 1)$ . But importantly, our algorithm in [Corollary 1.2](#) has the benefit of automatically improving in future using any other advances on classical algorithms for correlation clustering.

We can also implement our algorithm in the dynamic streaming model. Here, the input graph  $G = (V, E)$  is presented to the algorithm as a stream of edge insertions and deletions, and the algorithm can make a single pass (or a few passes) over this stream and should output the answer to the problem on the graph  $G$  at the end.

**Corollary 1.3.** *There is a polynomial-time randomized streaming algorithm for correlation clustering that uses  $\tilde{O}(n)$  memory when making a single pass over a dynamic stream, and with high probability, achieves an  $(\alpha_{\text{BEST}} + o(1))$ -approximation.*

Again, our result improves upon the prior 3-approximation algorithm of [CKL<sup>+</sup>24] in dynamic streams and 1.84-approximation algorithm of [CLP<sup>+</sup>24] in insertion-only streams.

Finally, our approach also have an interesting consequence to insertion-only streams using non-sketching techniques (in particular using [Result 3](#) and not [Result 2](#) used in our other algorithms):

it provides the first polynomial time algorithm that processes the stream *deterministically* and uses randomness only at the end of the stream. This guarantee in particular satisfies the notion of adversarially-robust streaming algorithms [BJWY22] in the strongest possible sense as it works even against an adversary that sees its internal state; see also [CGS22].

**Corollary 1.4.** *There is a polynomial-time streaming algorithm for correlation clustering that uses  $\tilde{O}(n)$  memory to deterministically build a data structure  $D$  using a single pass over an insertion-only stream, and only at the end, uses randomization to, with high probability, recover from  $D$  an  $(\alpha_{\text{BEST}} + o(1))$ -approximation for correlation clustering.*

Before moving on, an important remark about our sublinear algorithms is in order.

**Remark 1.5.** Our approach inherently bounds the size of the sketch it computes and not the post-processing algorithms (given we have no control over the space-complexity of the best classical algorithm we run at the end beside it being polynomial). For our distributed algorithms, this is inconsequential. In the MPC model, this means the in- and out-communication by each machine will be bounded by  $\tilde{O}(n)$  (but not the internal memory) which is inline with the original definitions in [BKS17] (see also [RVW18]) that allow for any complex operations to be done on each machine. For the streaming algorithms, this means that the memory of the algorithm *during* the stream is bounded by  $\tilde{O}(n)$  but after the stream finishes, to recover the solution, the space used by the algorithm may become larger. We note that to our knowledge, all existing streaming *lower bounds* only bound the space of the algorithm during the stream<sup>6</sup>.

## 1.4 Our Techniques

Our approach in establishing [Result 3](#) consists of two steps: (1) recovering a *fractional* sparsifier, namely, a graph with all edge-weights in  $[0, 1]$ , from the given spectral sparsifier  $H$  of  $G$ , and then, (2) *rounding* this fractional sparsifier into a simple unweighted graph to obtain the graph  $\tilde{G}$ . We implement the first step (for both parts of this result) by formulating the problem as a convex program and devising a separation oracle to run Ellipsoid algorithm on this program (the separation oracle crucially relies on  $H$  being a spectral sparsifier, as the oracle for cut sparsifiers is solving an NP-hard problem in general). The second part is done via a randomized rounding approach—which, additionally ensures the number of sampled edges *exactly* matches the original graph—but requires different analysis for each part: a union bound approach using Karger’s cut-bounding bound (see [Proposition 3.7](#)) relying on the assumption that minimum cut is not too small, or, following the standard effective resistance sampling approach (see [Proposition 4.1](#)) for constructing spectral sparsifiers, using the assumption that effective resistances are not too large.

To obtain our sketch in [Result 2](#) from [Result 3](#), we first use a sketch due to [AGM12a, KPS24] that identifies  $\tilde{O}(n)$  edges, whose removal partitions the graphs into subgraphs with large enough minimum cut as required by [Result 3](#); in parallel, we also use a sketch by [KLM<sup>+</sup>14a] for spectral sparsification, and use linearity of these sketches to recover a sketch for each of these large-min-cut subgraphs. A final argument then shows we can use the recovered edges plus the unweighted sparsifiers on each component obtained via [Result 3](#) to get an unweighted sparsifier of the entire graph as well (in [Section 3](#), we show how the plan outlined above can recover a cut sparsifier from the sketch, which is a weaker version of [Result 2](#) but is sufficient for proving [Result 1](#) for correlation clustering; we then improve this to recover a spectral sparsifier in [Section 4](#)).

---

<sup>6</sup>Specifically, the techniques in communication complexity and branching programs used for proving streaming lower bounds are inherently oblivious to the post-processing space of the algorithm.

Finally, to obtain [Result 1](#) from our [Result 2](#), we follow previous arguments in [[ACG<sup>+</sup>15](#), [BCMT23](#)] that show cut sparsifiers (information-theoretically) preserve correlation clustering structure; the new part here is to ensure the problem reduces to an instance of correlation clustering on the (unweighted) sparsifier (not some rather arbitrary computation as in [[ACG<sup>+</sup>15](#), [BCMT23](#)]), which further requires us to *exactly* match the number of edges in the original graph and the unweighted sparsifier (which was an additional property obtained in [Result 2](#)).

## 1.5 Related Work

The last couple of years has witnessed a flurry of results on correlation clustering both for sublinear as well as classical algorithms. For instance, [[CLM<sup>+</sup>21](#)] designed  $O(1)$ -round MPC algorithms for correlation clustering, and building on this, [[AW22](#)] obtained single-pass streaming and sublinear time  $O(1)$ -approximation algorithms for this problem<sup>7</sup> (see also [[ACG<sup>+</sup>15](#), [CDK14](#)] and references therein for earlier work on this problem). These results were subsequently improved in a series of work in [[BCMT22](#), [BCMT23](#), [DMM24](#), [MC23](#), [CKL<sup>+</sup>24](#), [CLP<sup>+</sup>24](#)] culminating in the work of [[CLP<sup>+</sup>24](#)] that achieves a 1.847-approximation via single-pass streaming or sublinear time algorithms and 1.876-approximation in  $O(1)$  MPC rounds (considerably simpler algorithms achieving a  $(3 + \varepsilon)$ -approximation were also developed in [[MC23](#), [CKL<sup>+</sup>24](#)] by adapting the landmark Pivot algorithm of [[ACN08](#)] to these models).

Meanwhile, there has also been exciting progress on classical algorithms for correlation clustering. Early work on this problem led to 3-approximation combinatorial and 2.5-approximation LP based algorithms for this problem [[ACN08](#)], which was then improved to a 2.06 [[CMSY15](#)]. Recently, [[CLN22](#)] broke the 2-approximation barrier—the integrality gap of the LP of [[ACN08](#)]—and achieved a 1.995-approximation which was then improved to a 1.73-approximation in [[CLLN23](#)] and subsequently 1.437-approximation in [[CCL<sup>+</sup>24](#)]. Finally, [[CLP<sup>+</sup>24](#)] gave a combinatorial 1.84-approximation algorithm which, as stated earlier, can also be implemented in streaming and sublinear time (and with some small loss MPC) models.

## 2 Preliminaries

**Notation.** Throughout, we work with undirected graphs  $G = (V, E)$  and use  $n$  to denote the number of vertices in  $G$ . For a set  $S \subseteq V$ , we use  $G[S]$  to denote the induced subgraph of  $G$  on  $S$ , and  $\delta(S)$  to denote the edges crossing the cut induced by  $S$ . For a vertex  $v \in V$ , we use  $N(v)$  and  $\deg(v) = |N(v)|$  to denote its neighbors and its degree, respectively.

For weighted graphs, we use  $w_G : E \rightarrow \mathbb{R}$  to denote the weights. We often treat unweighted graphs as weighted graphs with weight one on every edge (primarily, to avoid repeating definitions for them separately). For a cut  $S \subseteq V$ , we use  $\text{cut}_G(S) := \sum_{e \in \delta(S)} w_G(e)$  to denote the weight of the edges in the cut. We use  $\text{mincut}(G)$  to denote the minimum cut value in  $G$ . Additionally, we use  $L_G$  to denote the Laplacian matrix of the graph  $G$ , where  $(L_G)_{u,u}$  is the weighted degree of each vertex  $u \in V$ , and  $(L_G)_{u,v} = -w_{u,v}$  for each edge  $(u, v) \in E$  and 0 otherwise.

We say an event happens *with high probability* if its probability is at least  $1 - 1/\text{poly}(n)$  where  $n$  is the number of vertices in the underlying graph (which will be clear from the context).

Likewise, we will often use the shorthand  $a \in (1 \pm \varepsilon)b$  to mean that  $(1 - \varepsilon)b \leq a \leq (1 + \varepsilon)b$ .

**Correlation clustering.** For a partition  $V_1, \dots, V_k$ , we use  $E_G^+(V_1, \dots, V_k)$  to denote all the edges in  $G$  which are crossing between  $V_1, \dots, V_k$ . Likewise, we use  $E_G^-(V_i)$  to denote the set of non-edges (i.e., not present edges in  $G$ ) which are contained in  $V_i$ .

---

<sup>7</sup>The constants in these algorithms are quite large, around 700 for [[CLM<sup>+</sup>21](#)] and more than 10000 for [[AW22](#)].

**Definition 2.1.** Let  $G = (V, E)$  be an arbitrary unweighted graph. Then, for a partition  $V_1, \dots, V_k$ , the value of the partition under the correlation clustering objective is:

$$\text{CC}_G(V_1, \dots, V_k) = \sum_{i \in [k]} |E^-(V_i)| + |E^+(V_1, \dots, V_k)|.$$

The goal in the correlation clustering problem is to find a partition that minimizes this objective.

## 2.1 Cut and Spectral Sparsifiers

We will frequently be concerned with graph sparsifiers and specifically cut sparsifiers [BK96] and spectral sparsifiers [ST11]. We review their definitions here.

**Cut sparsifiers.** A basic notion of sparsification is *cut sparsification* introduced by [BK96].

**Definition 2.2** ([BK96]). Given a graph  $G = (V, E)$  and  $\varepsilon \in (0, 1)$ , a graph  $\tilde{G}$  is said to be a  $(1 \pm \varepsilon)$  **cut sparsifier** of  $G$  iff for every cut  $S \subseteq V$ ,

$$(1 - \varepsilon) \cdot \text{cut}_G(S) \leq \text{cut}_{\tilde{G}}(S) \leq (1 + \varepsilon) \cdot \text{cut}_G(S).$$

We note that one often requires a cut sparsifier of a graph to be its subgraph. However, as stated in [Result 2](#), this is not the case in our paper due to our de-sparsification approach (which does not require this guarantee, nor can provide it without trivializing the problem).

A key quantity of interest when designing cut sparsifiers is known as the *strength* of an edge:

**Definition 2.3.** Given a graph  $G = (V, E)$ , the **strength** of an edge  $e \in E$  is defined as

$$\lambda_e = \max_{S \subseteq V : e \subseteq S} \text{mincut}(G[S]).$$

**Spectral sparsifiers.** A strictly stronger notion than cut sparsifiers are *spectral sparsifiers* [ST11].

**Definition 2.4** ([ST11]). Given a graph  $G = (V, E)$ , a graph  $\tilde{G}$  is considered a  $(1 \pm \varepsilon)$  **spectral sparsifier** of  $G$  iff for every vector  $x \in \mathbb{R}^V$ ,

$$(1 - \varepsilon) \cdot x^\top L_G x \leq x^\top L_{\tilde{G}} x \leq (1 + \varepsilon) \cdot x^\top L_G x,$$

where  $L_G$  and  $L_{\tilde{G}}$  denote the Laplacian matrix of  $G$  and  $\tilde{G}$ , respectively.

Similar to strength of edges defined in the context of cut sparsifiers, we have effective resistances for spectral sparsifiers.

**Definition 2.5.** For a graph  $G = (V, E)$ , and a pair of vertices  $(u, v) \in \binom{V}{2}$ , we say that the **effective resistance** of  $(u, v)$  in  $G$  is:

$$R_{\text{eff}, G}(u, v) = \max_{x \in \mathbb{R}^V, x \neq 0} \frac{(x_u - x_v)^2}{x^\top L_G x}.$$

Finally, we need some additional properties from sparsifiers captured in the following definition.

**Definition 2.6.** Given a  $(1 \pm \varepsilon)$  cut/spectral sparsifier  $\tilde{G}$  of a graph  $G = (V, E)$ , we say that  $\tilde{G}$  is **total weight preserving** if it additionally satisfies  $\sum_{e \in G} w_G(e) = \sum_{e \in \tilde{G}} w_{\tilde{G}}(e)$ . Similarly, we say  $\tilde{G}$  is **simple** iff it is an unweighted simple graph.

## 2.2 Graph Sketches

In this work, we will frequently be concerned with designing (linear) graph sketches, introduced in the work of [AGM12a] (for graph problems).

**Definition 2.7.** A *linear sketch* of a graph  $G$  is identified by a (possibly randomized) matrix  $M$  of dimensions  $s \times \binom{n}{2}$ , chosen independently of the graph. Then, given an unweighted graph  $G$  with edge incidence vector  $\mathbf{1}_G \in \{0, 1\}^{\binom{n}{2}}$ , the *sketch* of the graph is given by  $M \cdot \mathbf{1}_G$ . Finally, there is a *recovery algorithm* that given only the sketch and the sketching matrix, with no direct access to  $G$ , outputs the solution to a given problem on  $G$ .

The convention is that the entries in the linear sketch should be bounded in magnitude by  $\text{poly}(n)$ , and thus the space complexity of the linear sketch is  $O(s \log(n))$  bits.

We note that even though we work with both weighted and unweighted graphs in this work, we have opted to define the sketching only for unweighted graphs given certain subtleties in the definition for weighted graphs, which will not be relevant to our work (see [CKL22] for more details).

Linearity of these sketches allows one to use them in various sublinear algorithms models; we elaborate more on this in [Section 5](#) when designing our sublinear algorithms.

## 3 Desparsification for Correlation Clustering: Proof of [Result 1](#)

In this section, we prove the following theorem that formalizes [Result 1](#).

**Theorem 1.** Let  $\alpha_{\text{BEST}}$  be the best possible approximation ratio for correlation clustering on simple graphs in polynomial time. There is a linear sketch of size  $\tilde{O}(n)$  bits, which for any simple graph  $G$  on  $n$  vertices can be used to recover an  $(\alpha_{\text{BEST}} + o(1))$  approximation to correlation clustering in  $G$  in polynomial time with high probability.

The key building block in the proof of [Theorem 1](#) is the following general de-sparsification result, which is a weaker version of [Result 2](#) (we opted to start with this weaker version as it suffices for our application and contains many of ideas for the full result as well).

**Theorem 2.** There is a (randomized) linear sketch using  $\tilde{O}(n/\varepsilon^2)$  bits of space which, for any simple graph  $G$ , can be used to recover with high probability a simple,  $(1 \pm \varepsilon)$  total weight preserving cut sparsifier of  $G$  in polynomial time.

In the rest of this section, we first show how to use total weight preserving sparsifiers to solve correlation clustering and prove [Theorem 1](#) using [Theorem 2](#). We then switch to the proof of [Theorem 2](#) by presenting its sketch first, and then going through the two separate steps outlined in [Section 1.4](#) needed for its proof.

### 3.1 Correlation Clustering from Total Weight Preserving Sparsifiers

The following lemma motivates total weight preserving cut sparsifiers for correlation clustering.

**Lemma 3.1.** Let  $G$  and  $H$  be graphs on the same vertex set such that  $H$  is a  $(1 \pm \varepsilon)$  total weight preserving cut sparsifier of  $G$ . Then, for any partition  $V_1, \dots, V_k$  of vertices,

$$\text{CC}_H(V_1, \dots, V_k) \in (1 \pm 2\varepsilon) \cdot \text{CC}_G(V_1, \dots, V_k).$$

We note that similar but not identical statements as [Lemma 3.1](#) have been used in prior work in [ACG<sup>+</sup>15, BCMT23]; as such, we postpone the proof of this lemma to [Appendix A](#). With this lemma, we can immediately obtain [Theorem 1](#), assuming [Theorem 2](#).

*Proof of Theorem 1.* The linear sketch is exactly the one of [Theorem 2](#). Let  $\tilde{G}$  denote the recovered simple graph which is a  $(1 \pm \varepsilon)$  total weight preserving cut sparsifier of  $G$ . By [Lemma 3.1](#), we see that for any clustering  $V_1, \dots, V_k$ ,

$$\text{CC}_{\tilde{G}}(V_1, \dots, V_k) \in (1 \pm 2\varepsilon) \cdot \text{CC}_G(V_1, \dots, V_k).$$

So, if we let  $\text{OPT}(G)$  denote the minimum correlation clustering value on  $G$ , we know that

$$\text{OPT}(\tilde{G}) \leq (1 + 2\varepsilon) \cdot \text{OPT}(G).$$

Now, let us run any black-box  $\alpha_{\text{BEST}}$ -approximation, polynomial time algorithm for correlation clustering on  $\tilde{G}$  in (crucially using the fact that  $\tilde{G}$  is simple). We are guaranteed that this recovers a partition  $P = (V_1, \dots, V_k)$  of vertices such that

$$\text{CC}_{\tilde{G}}(P) \leq \alpha \cdot \text{OPT}(\tilde{G}).$$

Returning  $P$  as the answer on  $G$ , by [Lemma 3.1](#) satisfies

$$\text{CC}_G(P) \leq (1 + 2\varepsilon) \cdot \text{CC}_{\tilde{G}}(P) \leq (1 + 2\varepsilon) \cdot \alpha \cdot \text{OPT}(\tilde{G}) \leq \alpha \cdot (1 + 2\varepsilon)^2 \cdot \text{OPT}(G).$$

Finally, by setting  $\varepsilon = o(1)$ , the linear sketch we use requires only  $\tilde{O}(n)$  bits, yet still recovers an  $(\alpha_{\text{BEST}} + o(1))$ -approximate solution to correlation clustering on  $G$  in polynomial time. ■

### 3.2 Building the Linear Sketch used in [Theorem 2](#)

We now switch to proving [Theorem 2](#) which is the main technical contribution of this section. We will require three distinct linear sketches for constructing our total-weight preserving sparsifier:

1. First, we require a linear sketch which, for some parameter  $\lambda = \Theta(\log(n)/\varepsilon^2)$  to be chosen later, can be used to (exactly) recover all edges of strength at most  $\lambda$  in the graph  $G$ , denoted by  $S_1(G)$ . This is done via the following result.

**Proposition 3.2** (cf. [\[AGM12a\]](#), [\[KPS24, Claim 4.9\]](#)). *For any given  $\lambda \geq 1$ , there is a linear sketch for (unweighted) graphs  $G$  on  $n$  vertices for recovering all edges of strength  $\leq \lambda$  with high probability in polynomial time, using  $\tilde{O}(n\lambda)$  space.*

2. Second, we require a linear sketch which recovers a  $(1 \pm \varepsilon)$  spectral sparsifier of the graph  $G$ , denoted by  $S_2(G)$ . This is done via the following result.

**Proposition 3.3** ([\[KLM<sup>+</sup>14b\]](#)). *For any given  $\varepsilon \in (0, 1)$ , there is a linear sketch for (unweighted) graphs  $G$  on  $n$  vertices for recovering a  $(1 \pm \varepsilon)$  spectral sparsifier of  $G$  with high probability in polynomial time, using  $\tilde{O}(n/\varepsilon^2)$  space.*

3. Finally, our remaining linear sketch is simply the total number of edges present in the graph. This is a deterministic linear sketch that simply tracks the size of the support of the  $\binom{n}{2}$  dimensional vector describing the graph  $G$ , and we denote this sketch by  $S_3(G)$ , but will often implicitly refer to this quantity as  $m$ .

The lemma below shows how these three linear sketches can be used to recover structural information about  $G$  that will be sufficient for de-sparsification.

**Lemma 3.4.** *Given the linear sketches  $S_1(G), S_2(G), S_3(G)$ , one can recover:*

1. *a set of edges  $T \subseteq G$  such that  $G - T$  has minimum cut  $> \lambda$ ,*
2. *a  $(1 \pm \varepsilon)$ -spectral sparsifier of the graph  $G - T$ , and*
3. *the total number of edges in  $G - T$ .*

*Further, the space complexity of these sketches is  $\tilde{O}(n\lambda + n/\varepsilon^2)$  bits.*

*Proof.* We start by using  $S_1(G)$  to recover the edges of strength at most  $\lambda$  in  $G$ , denoted by  $T$ . Importantly, because we recover *only* these edges, the resulting graph  $G - T$  has all edges of strength greater than  $\lambda$ . This step implicitly partitions the vertex set  $V$  into  $V_1, V_2, \dots, V_k$  such that for  $i \in [k]$ , the subgraph of  $G - T$  induced by  $V_i$  has minimum cut greater than  $\lambda$ . We rely here on the basic property of edge strengths, namely, the certificate of an edge having strength greater than  $\lambda$  in  $G$  never uses an edge of strength at most  $\lambda$ . In other words, any edge in  $T$  necessarily connects a vertex in some  $V_i$  to a vertex in some  $V_j$  for  $1 \leq i \neq j \leq k$ .

Next, because we have recovered the set  $T \subseteq G$  of edges, we can simply delete these edges from the linear sketch  $S_2(G)$ , yielding a linear sketch  $S_2(G - T)$ . Now, invoking the recovery algorithm of [Proposition 3.3](#), we can recover a  $(1 \pm \varepsilon)$ -spectral sparsifier for each of  $G[V_1], G[V_2], \dots, G[V_k]$ .

Finally, number of edges in  $G - T$  is obtained by subtracting the number of edges in  $T$  from  $m$ .

The space complexities of linear sketches  $S_1(G)$  and  $S_2(G)$  follow from [Proposition 3.2](#) and [Proposition 3.3](#), respectively. The sketch  $S_3(G)$  takes only  $O(\log n)$  space. ■

The combination of  $S_1, S_2, S_3$  is the entirety of the linear sketches that we will store. The rest of the complexity in our procedure is in *recovering* a specific type of sparsifier. We discuss this more in the coming subsections.

### 3.3 Recovering Fractional Total Weight-preserving Sparsifiers

We now describe how given a total weight preserving spectral sparsifier  $H$  of some unweighted graph  $G$  with  $m$  edges, we can recover in polynomial time a *fractional*, total weight preserving sparsifier  $\tilde{G}$ . We say  $\tilde{G}$  is fractional in the sense that every edge  $e \in \tilde{G}$  will have a fraction  $Y_e \in [0, 1]$  assigned to it, which can be seen as its weight. In the next subsection we will show how  $\tilde{G}$  can be rounded to an unweighted graph that is a total weight-preserving cut-sparsifier of  $H$  and as such  $G$  as well.

In this section, we prove the following lemma:

**Lemma 3.5.** *Given a (potentially weighted) graph  $H$  which is promised to be a  $(1 \pm \varepsilon)$  spectral sparsifier of some unweighted graph  $G$ , along with the number of edges in  $G$ , one can recover (in polynomial time) a  $(1 \pm 3\varepsilon)$  fractional total weight preserving spectral sparsifier of the graph  $G$ .*

*Proof.* Consider the following convex program in  $\mathbb{R}^{\binom{V}{2}}$ , for which we wish to find a feasible point:

$$\begin{aligned} Y_e &\in [0, 1] \quad \forall e \in \binom{V}{2}, \\ \sum_{e \in \binom{V}{2}} Y_e \cdot z^\top L_e z &\geq (1 - \varepsilon) z^\top L_H z \quad \forall z \in \mathbb{R}^V : \|z\|_2 = 1, \end{aligned}$$

$$\boxed{\begin{aligned} \sum_{e \in \binom{V}{2}} Y_e \cdot z^\top L_e z &\leq (1 + \varepsilon) z^\top L_H z \quad \forall z \in \mathbb{R}^V : \|z\|_2 = 1, \\ \sum_{e \in \binom{V}{2}} Y_e &= m. \end{aligned}}$$

Here,  $L_e$  is the Laplacian matrix of the  $n$ -vertex graph consists of only the single edge  $e$ .

Each of the infinitely many constraints of this program are linear in the variables  $Y_e$  since  $z^\top L_H z$  (and similar for  $L_G$ ) are simply numbers for each fixed  $z$ . This program also has a feasible solution, as the original graph  $G$  is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $H$  (and vice versa), with  $m$  edges and weights that are  $\{0, 1\}$  and hence the characteristic vector of its edges form a feasible solution to this program.

We now show there is a polynomial time separation oracle for this program. Fix any assignment to  $Y_e$ 's yielding a fractional graph which we will denote by  $G(Y)$ , where the weight of edge  $e$  is  $Y_e$ . We can check in polynomial time whether the total edge weights in  $G(Y)$  equal  $m$ , that each  $Y_e \in [0, 1]$ , and that  $G(Y)$  and  $H$  have the same connected components. If any of these checks fail, we have found a violated constraint. We now focus on verifying that  $L_{G(Y)}$  is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $L_H$ . That is, we wish to check whether

$$(1 - \varepsilon) \cdot L_H \preceq L_{G(Y)} \preceq (1 + \varepsilon) \cdot L_H,$$

where  $\preceq$  refers to the Loewner order of PSD matrices. Observe that this condition passes if and only if  $G(Y)$  is a  $(1 \pm \varepsilon)$ -spectral sparsifier of  $H$ , as

$$(1 - \varepsilon) \cdot L_H \preceq L_{G(Y)} \preceq (1 + \varepsilon) \cdot L_H \iff \forall z \in \mathbb{R}^V : (1 - \varepsilon) \cdot z^\top L_H z \leq z^\top L_{G(Y)} z \leq (1 + \varepsilon) \cdot z^\top L_H z,$$

by the definition of the Loewner order.

By left and right multiplying by  $L_H^{\dagger/2}$ , where  $L_H^\dagger$  is the pseudo-inverse of  $L_H$  (and restricting our attention to the image of  $L_H$ ), this is equivalent to checking whether

$$(1 - \varepsilon) \cdot I_{\text{Im}(L_H)} \preceq L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2} \preceq (1 + \varepsilon) \cdot I_{\text{Im}(L_H)}, \tag{2}$$

where  $I_{\text{Im}(L_H)}$  is simply the projection operator on to  $\text{Im}(L_H)$ <sup>8</sup>. Next, we note that Eq (2) is true if and only if all non-trivial eigenvalues of  $L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2}$  are in  $(1 \pm \varepsilon)$ . This shows that  $G(Y)$  is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $H$  if and only if every (non-trivial) eigenvalue of  $L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2}$  is in the range  $(1 \pm \varepsilon)$ .

Next, we want to show that if we identify an eigenvector  $v$  of  $L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2}$  with eigenvalue  $\lambda \notin (1 \pm \varepsilon)$ , then we can use this to find a violated constraint in our convex program.

Indeed, let us suppose that we recover such a  $v$  and  $\lambda$ :

$$L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2} \cdot v = \lambda \cdot v;$$

by left multiplying with  $v^\top$  this yields

$$v^\top \cdot L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2} \cdot v = \lambda \cdot v^\top v.$$

---

<sup>8</sup>This technicality is due to the fact that  $L_H$  and  $L_H^\dagger$  have a non-trivial null-space (i.e., they will always contain the vector of all 1's).

However, for the vector  $L_H^{\dagger/2} \cdot v$ , we also see that

$$v^\top \cdot L_H^{\dagger/2} \cdot L_H \cdot L_H^{\dagger/2} \cdot v = v^\top \cdot L_H^{\dagger/2} \cdot L_H^{1/2} \cdot L_H^{1/2} \cdot L_H^{\dagger/2} \cdot v = v^\top v.$$

Thus, if  $\lambda \notin (1 \pm \varepsilon)$ , we have

$$v^\top \cdot L_H^{\dagger/2} \cdot L_{G(Y)} \cdot L_H^{\dagger/2} \cdot v = \lambda \cdot v^\top v \notin (1 \pm \varepsilon) \cdot v^\top v = (1 \pm \varepsilon) \cdot v^\top \cdot L_H^{\dagger/2} \cdot L_H \cdot L_H^{\dagger/2} \cdot v,$$

and so we can use the constraint specified by  $z = L_H^{\dagger/2} \cdot v$  as a violated constraint in the convex program. Since calculating pseudo-inverses, multiplying matrices, and finding eigenvalues can all be done in polynomial time, the procedure above gives a polynomial-time separation oracle.

To conclude, since the feasible region for the above convex program is non-empty, and we have a polynomial time separation oracle, we can find an assignment to the  $Y_e$ 's which satisfies all of the above constraints in polynomial time by using the ellipsoid method (see [GLS88, Theorem 6.4.1], for instance). For the feasible solution  $Y$  found at the end, the fractional graph  $G(Y)$  defined by the assignment  $Y$  is a  $(1 \pm \varepsilon)^2$  spectral sparsifier of  $G$  (it is a  $(1 \pm \varepsilon)$  sparsifier of  $H$ , which is in turn a  $(1 \pm \varepsilon)$  sparsifier of  $G$ ), with the same total weight as the graph  $G$ . ■

### 3.4 Rounding the Fractional Sparsifier

Finally, in this section we will show how, given a fractional total weight preserving graph  $H$ , we can round the weights in such a way that we get a simple (unweighted) graph which is a  $(1 \pm \varepsilon)$  cut-sparsifier of  $H$ , while still preserving the total weight exactly. For any edge  $e$  in  $H$ , we will denote by  $w_H(e)$  the fractional weight assigned to the edge  $e$ , that is,  $w_H(e) \in [0, 1]$ . Towards this goal, we establish the following lemma.

**Lemma 3.6.** *Let  $H$  be a fractional graph whose minimum cut is at least  $\lambda \geq 200 \log(n)/\varepsilon^2$ , and whose total edge weight sums to an integer. Then, there is a polynomial time randomized rounding scheme which with high probability recovers a simple graph  $\tilde{H}$  which is a total weight preserving  $(1 \pm \varepsilon)$  cut sparsifier of  $H$ .*

To prove this lemma, we require the following classic result due to [Kar93].

**Proposition 3.7** (Karger's Cut-counting Bound [Kar93]). *Let  $G$  be any (potentially weighted) graph on  $n$  vertices with minimum cut size  $\lambda(G)$ . Then, the number of cuts in  $G$  of size  $\leq \alpha \cdot \lambda$  is at most  $n^{2\alpha}$ .*

We are now ready to prove Lemma 3.6.

*Proof of Lemma 3.6.* The rounding scheme itself is elementary: we create a simple, unweighted graph  $\tilde{H}$ , where for every edge  $e \in H$ , we keep  $e$  independently with probability  $w_H(e)$ . First, we will show that this procedure recovers a  $(1 \pm \varepsilon)$  sparsifier with probability  $\geq 1 - 1/n^5$  (although it may not be total weight preserving).

Let  $t := 200 \log(n)/\varepsilon^2$ . Because the minimum cut in  $H$  is of size  $\geq t$ , by Karger's cut-counting bound (Proposition 3.7), we know that for any  $\alpha \in \mathbb{Z}^+$ , the number of cuts of size at most  $\alpha \cdot t$  is bounded by  $n^{2\alpha}$ . Now, fix a cut  $S$  of size  $\in [\alpha \cdot t, 2\alpha \cdot t]$ . It follows from Chernoff bound that

$$\Pr(\text{cut}_{\tilde{H}}(S) \notin (1 \pm \varepsilon) \cdot \text{cut}_H(S)) \leq \exp(-\varepsilon^2 \cdot \alpha \cdot t / 12) \leq \exp\left(-\varepsilon^2 \cdot \alpha \cdot \frac{200 \log(n)}{12\varepsilon^2}\right) < n^{-10\alpha};$$

(to apply Chernoff bound, we can simply view the weight contributed by each edge as a Bernoulli random variable. The expected weight of a cut under the sampling procedure is exactly equal to its current weight, and the Chernoff bound then follows simply).

Taking a union bound over all  $n^{4\alpha}$  possible cuts, this then yields that every cut of size  $\in [\alpha \cdot t, 2\alpha \cdot t]$  is preserved to within a  $(1 \pm \varepsilon)$  factor with probability at least  $1 - 1/n^{6\alpha}$ . Finally, integrating over all choices of  $\alpha \geq 1$ , we get that every cut is preserved to a factor of  $(1 \pm \varepsilon)$  with probability at least  $1 - 1/n^5$ .

The next step is to show that we can also sample exactly  $\sum_e w_H(e)$  edges in  $\tilde{H}$  in our randomized rounding approach. For this, we need the following auxiliary claim.

**Claim 3.8.** *Let  $p_1, \dots, p_m$  each be in  $[0, 1]$ , and let  $K = \sum_{i=1}^m p_i$  be an integer. Now, let  $X_i = \text{Bern}(p_i)$  and let the  $X_i$ 's be independently distributed. Then,*

$$\Pr\left[\sum_{i=1}^m X_i = K\right] \geq \frac{1}{m+1}.$$

*Proof.* This follows from the fact that the mode of a Poisson binomial distribution is either its mean, or differs from its mean by at most 1. In particular, in our case when the mean is an integer ( $K$ ), it must be the case that the mode  $\ell = K$  also (see [TT23], page 2, Darroch's rule for instance). Now, because the support of the distribution has size at most  $m+1$  (i.e.,  $0, \dots, m$ ), it follows that the mode must occur with probability  $\geq 1/(m+1)$ , yielding our claim above. ■ [Claim 3.8](#)

To conclude the proof of [Lemma 3.6](#), we can apply [Claim 3.8](#) to our randomized rounding procedure. We see that with probability  $\geq \frac{1}{n^2}$ , we will sample *exactly*  $\sum_{e \in H} w_H(e)$  edges in  $\tilde{H}$ . This is because our edge sampling procedure is exactly a Poisson binomial distribution fitting the form of [Claim 3.8](#).

So, we employ the following simple procedure: for  $n^3$  rounds, we randomly sample edges in accordance with the above scheme. With probability  $1 - (1 - 1/n^2)^{n^3} = 1 - 2^{-\Omega(n)}$ , we know that in at least one round, we will recover a graph  $\tilde{H}$  which exactly preserves the total edge mass compared to  $H$ . Further, by a union bound over all  $n^3$  graphs generated in these rounds, we know that with probability  $1 - 1/n^2$  every single graph we generate will be a  $(1 \pm \varepsilon)$  cut sparsifier of  $H$ . Thus, the graph  $\tilde{H}$  that we return is the first graph which preserves the total weight, and it will be a  $(1 \pm \varepsilon)$  total weight preserving cut sparsifier of  $H$  with high probability. ■ [Lemma 3.6](#)

### 3.5 Concluding the Proof of [Theorem 2](#)

Finally, in this section we synthesize our claims to conclude the proof of [Theorem 2](#).

*Proof of [Theorem 2](#).* First, we initialize the linear sketch of [Lemma 3.4](#), using  $\lambda = 200 \log(n)/\varepsilon^2$ . The space complexity of our sketch then follows from [Lemma 3.4](#).

Using our sketch, we can recover all edges of strength at most  $\lambda$  (we denote this set by  $T$ ), as well as a  $(1 \pm \varepsilon)$  spectral sparsifier of the graph  $G - T$ , and the total number of edges in  $G - T$ . Next, using [Lemma 3.5](#), we find a fractional  $(1 \pm 3\varepsilon)$  total weight preserving spectral sparsifier  $H$  of  $G - T$  in polynomial time. Finally, we use [Lemma 3.6](#) to, with high probability, round this into a simple  $(1 \pm \varepsilon)$  total weight preserving cut sparsifier  $\tilde{H}$  of  $H$  in polynomial time. By composition of the sparsifier approximations, we also get that  $\tilde{H}$  is a  $(1 \pm 5\varepsilon)$  simple total weight-preserving cut sparsifier of  $G - T$ . Finally, we add back the edges from  $T$ , and conclude that  $\tilde{H} \cup T$  is a  $(1 \pm 5\varepsilon)$  simple total weight-preserving cut sparsifier of  $G$ .

We remark on a minor subtlety here. It seems possible that an edge in  $T$  is also included in our rounded solution  $\tilde{H}$ , and thus appears twice in  $\tilde{H} \cup T$ . However, recall that when we remove edges in  $T$ , this partitions  $G$  into connected components  $V_1, \dots, V_k$  each with minimum cut greater than  $\lambda$ . The set of edges  $T$  is exactly the set of all edges in  $G$  that go across these components. We observe that our spectral sparsifier for the graph  $G - T$  will thus not contain any edges crossing  $V_1, \dots, V_k$ . This in turn implies that the fractional graph  $H$  generated by our convex program will not have any edges crossing between  $V_1, \dots, V_k$  also, as otherwise, for some  $V_i$ ,  $\text{cut}_H(V_i)$  would be non-zero, whereas  $\text{cut}_{G-T}(V_i) = 0$ , violating the spectral approximation constraint in our convex program. So, for every edge  $e$  in  $T$ ,  $e$  is not present in the rounded graph  $\tilde{H}$ .

Finally, we can re-parameterize  $\varepsilon$  to some  $\Theta(\varepsilon)$  to obtain a  $(1 \pm \varepsilon)$  cut sparsifier with high probability, concluding the proof. ■

Before moving on from this section, we note that by combining [Lemma 3.5](#) and [Lemma 3.6](#), we get the following general de-sparsification corollary that formalizes part 1 of [Result 3](#).

**Corollary 3.9** (Part 1 of [Result 3](#)). *Given a (potentially weighted) graph  $H$  which is promised to be a  $(1 \pm \varepsilon)$  spectral sparsifier of some unweighted simple graph  $G$  with minimum cut  $\lambda \geq 200 \log(n)/\varepsilon^2$ , along with the number of edges in  $G$ , we can recover in polynomial time a simple graph  $\tilde{G}$  which is a  $(1 \pm O(\varepsilon))$  total weight preserving cut sparsifier of  $G$ , with high probability.*

## 4 De-sparsifying Spectral Sparsifiers: Proofs of [Result 2](#) and [3](#)

In the previous section, we focused on recovering an unweighted, simple total weight preserving *cut* sparsifiers of our original graph. This was motivated by applications to correlation clustering but also leads to a simpler analysis, as the rounding procedure yields correct outputs so long as the minimum cut value is sufficiently large. We now show that we can also recover a spectral sparsifier of the original graph via de-sparsification.

**Theorem 3.** *There is a randomized linear sketch of size  $\tilde{O}(n/\varepsilon^2)$  bits which, on any graph  $G$ , can be used to recover, with high probability, an unweighted, simple  $(1 \pm \varepsilon)$  total weight preserving spectral sparsifier of  $G$  in polynomial time.*

To prove [Theorem 3](#), we follow the strategy of [\[SS11\]](#) in constructing spectral sparsifiers by sampling edges proportional to their effective resistance ([Definition 2.5](#)). Formally,

**Proposition 4.1** ([\[SS11\]](#)). *Let  $G$  be an arbitrary graph on  $n$  vertices, let  $\varepsilon \in (0, 1)$ , and let  $C$  be a sufficiently large constant. Then, independently sampling each edge with probability*

$$p_e \geq w_e \cdot \frac{C \log(n) \cdot R_{\text{eff}, G}(e)}{\varepsilon^2}$$

(and assigning weight  $w_e/p_e$  if sampled) yields a  $(1 \pm \varepsilon)$  spectral sparsifier of  $G$  with high probability.

### 4.1 De-sparsifying With Small Effective Resistances

In this subsection, we prove part 2 of [Result 3](#), as its intuition will be very valuable in the proof of [Result 2](#). We first restate the result before providing our proof:

**Theorem 4** (Part 2 of [Result 3](#)). *For any  $\varepsilon \in (0, 1)$  and  $n$ -vertex unweighted graph  $G$ , there is a randomized polynomial-time algorithm that given any  $(1 \pm \varepsilon)$ -spectral sparsifier  $H$  of  $G$ , recovers with high probability an  $n$ -vertex, simple unweighted graph  $\tilde{G}$  such that  $\tilde{G}$  is a  $(1 \pm 5\varepsilon)$ -spectral sparsifier of  $G$ , provided the effective resistance of every pair  $(u, v)$  in  $G$  is  $\leq \frac{\varepsilon^2}{2C \log(n)}$ , where  $C$  is the constant from [Proposition 4.1](#).*

*Proof.* First, recall that by [Lemma 3.5](#) we can recover a fractional, total weight preserving graph  $\tilde{H}$  which is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $H$  and thus a  $(1 \pm 3\varepsilon)$  spectral sparsifier of  $G$ . All that remains is to perform a randomized rounding of  $\tilde{H}$  which yields a spectral sparsifier and preserves the total weight. We use the same rounding procedure as in [Lemma 3.6](#), and thus, just as in the proof of the lemma, by repeating the procedure a polynomial number of times, we can guarantee that the number of edges in our rounding matches the number of edges in the original graph. All that remains to be shown is that the rounded graph is a spectral sparsifier with high probability.

For this, by [Proposition 4.1](#), we know that sampling every edge  $e$  with probability

$$p_e \geq \frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot R_{\text{eff}}(e),$$

and assigning weight  $w_e/p_e$  to the sampled edges yields a  $(1 \pm \varepsilon)$  spectral sparsifier with high probability. By the hypothesis of our theorem, it follows that every edge  $e \in \tilde{H}$  will have effective resistance at most

$$(1 + 2\varepsilon) \cdot \frac{\varepsilon^2}{2C \log(n)}.$$

This is because  $\tilde{H}$  is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $G$ , and so for every pair of vertices  $(u, v) \in \binom{V}{2}$ , it is the case that  $R_{\text{eff}, \tilde{H}}(u, v) \in (1 \pm 2\varepsilon)R_{\text{eff}, G}(u, v)$  (see [Definition 2.5](#)).

So, by [Proposition 4.1](#) we must only sample each edge  $e \in \tilde{H}$  with probability

$$p_e \geq (1 + 2\varepsilon) \cdot \frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot \frac{\varepsilon^2}{2C \log(n)} = \frac{1 + 2\varepsilon}{2} \cdot w_e.$$

Since  $w_e \geq p_e$ , we can keep each edge  $e$  independently with probability  $w_e$ , while still yielding a  $(1 \pm \varepsilon)$  spectral sparsifier of  $H$  with high probability. Indeed, because  $\tilde{H}$  was already a  $(1 \pm 3\varepsilon)$  spectral sparsifier to  $G$ , the resulting graph is a  $(1 \pm 5\varepsilon)$  spectral sparsifier of  $G$  while also being a simple graph. ■

## 4.2 Proof of [Theorem 3](#)

We now provide a formal proof of [Theorem 3](#). The main challenge here is to handle the pairs of vertices whose effective resistances will be higher than the bounds in [Theorem 4](#) which requires a non-black-box modification of our approach in establishing [Theorem 2](#).

To prove [Theorem 3](#), we need to use the following more detailed analysis from [[KLM<sup>+</sup>14b](#)].

**Proposition 4.2** ([\[KLM<sup>+</sup>14b\]](#)). *Given any parameter  $\phi \geq 0$ , there is a (randomized) linear sketch  $\mathcal{S}$  such that for a graph  $G$  on  $n$  vertices,  $\mathcal{S}(G)$  can be used to recover each edge  $e$  independently with probability at least  $\phi \cdot r_{\text{eff}, G}(u, v)$ , and likewise assigns appropriate weights to create a  $(1 \pm \varepsilon)$  spectral sparsifier with probability  $1 - 1/\text{poly}(n)$ . Further,  $\mathcal{S}$  requires only  $\tilde{O}(n\phi)$  bits of space to store.*

We are now ready to start the proof.

**Linear sketch.** The linear sketch in [Theorem 3](#) is very simple: we simply store the sketch of [Proposition 4.2](#) with parameter  $\phi = \frac{C \log^3(n)}{\varepsilon^2}$ , for  $C$  a sufficiently large constant. It follows then by [Proposition 4.2](#) that this allows us to recover a  $(1 \pm \varepsilon)$  spectral sparsifier of the graph  $G$  with high probability. Further, since the linear sketch of [Proposition 4.2](#) implicitly performs effective-resistance sampling, it follows that every edge  $e$  with effective resistance  $\geq \frac{1}{\phi} = \frac{\varepsilon^2}{C \log^3(n)}$  is necessarily recovered, as each such edge is sampled with probability  $\geq \frac{\phi}{\phi} = 1$ .

**Recovery from the sketch.** Now, let  $\tilde{H}$  denote the weighted spectral sparsifier recovered by the above sketch, and let  $\tilde{H}_U$  denote the corresponding unweighted version of  $\tilde{H}$  (where every edge in  $\tilde{H}$  is given weight 1). Observe that it must be the case that  $\tilde{H}_U \subseteq G$ , as it is the result of sub-sampling  $G$ . Observe also that because  $\tilde{H}$  is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $G$ , for every pair of vertices  $(u, v) \in \binom{V}{2}$ , it is the case that  $R_{\text{eff}, \tilde{H}}(u, v) \in (1 \pm 2\varepsilon)R_{\text{eff}, G}(u, v)$  (see Definition 2.5).

Now, we define the set  $\hat{E} \subseteq \binom{V}{2}$ :

$$\hat{E} = \left\{ (u, v) \in \binom{V}{2} : (u, v) \notin \tilde{H}_U, R_{\text{eff}, \tilde{H}}(u, v) \leq \frac{\varepsilon^2}{100 \log^2(n)} \right\}.$$

In words, this is simply the set of pairs of vertices which have small effective resistance with respect to the recovered spectral sparsifier  $\tilde{H}$ . Using this we create our convex program, for which we wish to find a feasible point:

$$Y_e \in [0, 1] \quad \forall e \in \hat{E},$$

$$z^T L_{\tilde{H}_U} z + \sum_{e \in \binom{V}{2}} Y_e \cdot z^T L_e z \geq (1 - \varepsilon) z^T L_{\tilde{H}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1,$$

$$z^T L_{\tilde{H}_U} z + \sum_{e \in \binom{V}{2}} Y_e \cdot z^T L_e z \leq (1 + \varepsilon) z^T L_{\tilde{H}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1,$$

$$|\tilde{H}_U| + \sum_{e \in \binom{V}{2}} Y_e = m.$$

As before, we must show that this program is feasible:

**Claim 4.3.** *The stated convex program is feasible.*

*Proof.* This follows because the original graph  $G$  will be a  $(1 \pm \varepsilon)$  spectral sparsifier of  $\tilde{H}$  which preserves the total weight. Because we are already including the contribution of  $\tilde{H}_U$  in each of the constraints, there is a feasible solution corresponding to  $G - \tilde{H}_U$ , which will contain only edges that are in  $\hat{E}$ . This is because any edge in  $G$  with effective resistance larger than  $\frac{\varepsilon^2}{100 \log^2(n)}$  is already in  $\tilde{H}_U$ , so the entire support of  $G - \tilde{H}_U$  is thus in  $\hat{E}$ .  $\blacksquare$

Likewise, the separation oracle is efficiently implementable:

**Claim 4.4.** *There is a polynomial time separation oracle for the above convex program.*

*Proof.* This follows from all of the same reasons as in Lemma 3.5. Indeed certifying the constraints that  $Y_e \in [0, 1]$  and that  $\sum_{e \in L_{\tilde{H}_U}} 1 + \sum_{e \in \binom{V}{2}} Y_e = m$  are both trivial. Thus, it remains only to check whether

$$z^T L_{\tilde{H}_U} z + \sum_{e \in \binom{V}{2}} Y_e \cdot z^T L_e z \in (1 + \varepsilon) z^T L_{\tilde{H}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1.$$

Letting  $\hat{G}$  denote the graph whose edge weights are given by  $Y_e$  (and is 0 otherwise), this constraint is equivalent to

$$z^T L_{\tilde{H}_U} z + z^T L_{\hat{G}} z \in (1 \pm \varepsilon) z^T L_{\tilde{H}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1.$$

By linearity of the Laplacians, this can be re-written as

$$z^T(L_{\tilde{H}_U} + L_{\hat{G}})z \in (1 \pm \varepsilon)z^T L_{\tilde{H}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1,$$

which is now exactly in the same form as the constraints of [Lemma 3.5](#), and so can be checked by calculating the eigenvalues of  $L_{\tilde{H}}^{\dagger/2}(L_{\tilde{H}_U} + L_{\hat{G}})L_{\tilde{H}}^{\dagger/2}$ .  $\blacksquare$

Now, because of [Claim 4.3](#) and [Claim 4.4](#), we can use the ellipsoid method to find a feasible solution in polynomial time [[GLS88](#)]. So, let  $\hat{G}$  then denote this feasible solution recovered by the above program, where the edge set is  $\hat{E}$ , and the corresponding weight on each edge  $e \in \hat{E}$  is  $Y_e$ . Observe that  $\hat{G} \cup \tilde{H}_U$  is a fractional  $(1 \pm \varepsilon)$  spectral sparsifier of  $\tilde{H}$ , and thus a fractional total weight preserving  $(1 \pm 3\varepsilon)$  spectral sparsifier of  $G$ .

All that remains is to show that efficiently rounding this solution is possible. For this, by [Proposition 4.1](#), we know that sampling each edge  $e$  with probability  $p_e \geq \frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot R_{\text{eff}}(e)$ , and giving weight  $\frac{w_e}{p_e}$  yields a  $(1 \pm \varepsilon)$  spectral sparsifier with high probability. Now, because  $\hat{G} \cup \tilde{H}_U$  is a  $(1 \pm \varepsilon)$  spectral sparsifier of  $\tilde{H}$ , it follows that for every pair of vertices  $(u, v)$ ,

$$R_{\text{eff}, \hat{G} \cup \tilde{H}_U}(u, v) \leq (1 + 2\varepsilon)R_{\text{eff}, \tilde{H}}(u, v).$$

In particular, for every edge  $(u, v) \in \hat{E}$ , we have

$$R_{\text{eff}, \hat{G} \cup \tilde{H}_U}(u, v) \leq (1 + 2\varepsilon) \frac{\varepsilon^2}{100 \log^2(n)}.$$

Thus, in the graph  $\hat{G} \cup \tilde{H}_U$ , for every edge  $e \in \hat{E}$ , we calculate the minimal sampling rate as

$$p_e = \frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot R_{\text{eff}, \hat{G} \cup \tilde{H}_U}(e) \leq \frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot \frac{\varepsilon^2}{C \log(n)} \leq w_e.$$

So, we can independently keep each edge  $e \in \hat{G}$  with probability  $w_e$  while still creating a  $(1 \pm \varepsilon)$  spectral sparsifier of  $\hat{G} \cup \tilde{H}_U$ . By composing sparsifiers (as before) this yields a  $(1 \pm 5\varepsilon)$  spectral sparsifier of the original graph  $G$ , while also yielding a simple, unweighted graph (the edges in  $\tilde{H}_U$  are already unweighted).

By starting with an error parameter of  $\varepsilon/5$ , we then obtain the stated accuracy of our spectral sparsifier. Likewise, because we are performing the simple, independent Bernoulli rounding, we can repeat this procedure  $n^3$  times and be guaranteed by [Claim 3.8](#) that in some round, the total weight is exactly preserved.

This concludes the proof of [Theorem 3](#).  $\blacksquare$

## 5 Sublinear Algorithms

In this section, we provide the proofs of [Corollaries 1.1](#) to [1.4](#). We start with a brief review of the sublinear algorithms models we consider as well as formalizing the role of linear sketching for solving problems in these models. We then provide the proofs of the corollaries.

### 5.1 Sublinear Algorithms Models

In this paper, we will be working with the following sublinear algorithms models.

**Distributed communication.** In this model, the input graph  $G = (V, E)$  is edge partitioned across  $k$  machines. The machines can communicate with each other in a message-passing model (i.e., the communication is point to point), and the goal is to minimize the total communication between the machines. At the end, one designated machine should output the answer.

**Massively Parallel Computation (MPC).** In this model, the input graph  $G = (V, E)$  is edge partitioned across multiple machines initially. Computation happens in synchronous rounds wherein each machine can send and receive  $\tilde{O}(n)$ -size messages. The goal is to have a small number of rounds (ideally a small constant) and compute the final outcome on a designated machine.

We note that this model is often referred to as near-linear-memory MPC as opposed to the fully scalable MPC wherein the memory of each machine can be any  $n^\delta$  for constant  $\delta > 0$ .

**Dynamic Streams.** In this model, the input graph  $G = (V, E)$  is specified as a sequence of insertion and deletion to its edges in a stream. Specifically, each entry of the stream is of the form  $(u, v, +)$  or  $(u, v, -)$  which either inserts a new edge  $(u, v)$  to  $G$  or remove an already existing edge  $(u, v)$  from  $G$ , respectively. The algorithm can make one or a few passes over the stream and needs to output the final outcome on the resulting graph at the end of the last pass.

## 5.2 Distributed Communication Model

We start with proving [Corollary 1.1](#) restated below.

**Corollary** (Restatement of [Corollary 1.1](#)). *There is a polynomial-time randomized algorithm for correlation clustering in the distributed communication model with  $k$  machines that uses  $\tilde{O}(nk)$  communication in total, and with high probability, achieves an  $(\alpha_{\text{BEST}} + o(1))$ -approximation.*

*Proof.* One of the simplest ways of using linear sketching is to design distributed communication protocols. The coordinator samples the sketching matrix  $\mathcal{S}(\cdot)$  and shares it with all the other machines. Then, each machine  $i \in [k]$ , computes  $\mathcal{S}(G_i)$  on its own subgraph  $G_i$  and sends it to the coordinator. Finally, the coordinator forms  $\mathcal{S}(G) = \mathcal{S}(G_1 + \dots + G_k)$  using the linearity of the sketches. We can thus use our [Theorem 1](#) and achieve a polynomial time protocol with  $\tilde{O}(n)$  communication per machine and so  $\tilde{O}(nk)$  communication in total.

The only subtlety here is due to the randomness of the linear sketch and how the sketching matrix  $\mathcal{S}$  can be communicated efficiently between the machines. Because the linear sketches we are using in [Lemma 3.4](#) are just the spectral sparsification sketch of [Proposition 3.3](#) [[KLM<sup>+</sup>14b](#)] and the spanning forest sketch of [Proposition 3.2](#) [[AGM12a, KPS24](#)], and both these works have already shown how to use only  $\tilde{O}(n)$  many random bits to create the sketch, sharing the sketching matrix is also possible in  $\tilde{O}(n)$  communication. ■

## 5.3 MPC Algorithms

We next prove [Corollary 1.2](#) restated below.

**Corollary** (Restatement of [Corollary 1.2](#)). *There is a polynomial-time randomized algorithm for correlation clustering in the MPC model that uses  $O(1)$  rounds and  $\tilde{O}(n)$ -size messages per machine, and with high probability, achieves an  $(\alpha_{\text{BEST}} + o(1))$ -approximation.*

To prove this theorem, we need to use the specific form of graph sketching used by our algorithms, referred to as vertex-incidence sketches.

**Definition 5.1.** We say a linear sketch  $\mathcal{S}(G)$  is a **vertex-incidence sketch** iff

$$\mathcal{S}(G) = \left[ \mathcal{S}_1(N(v_1)), \mathcal{S}_1(N(v_2)), \dots, \mathcal{S}_n(N(v_n)) \right],$$

where each  $\mathcal{S}_i$  is also a linear sketch. Namely, the entire sketch is obtained as a linear sketch of neighborhood of each vertex separately.

The following is a well-known fact about MPC protocols for combining vertex-incidence sketches.

**Proposition 5.2** ([AGM12b, AKLP22]). Let  $\mathcal{S}(G)$  denote a vertex-incidence linear sketch which requires  $\leq s$  bits of space for each vertex-neighborhood sketch. Then, there is a 2 round,  $\tilde{O}(n \cdot s)$  communication MPC protocol which results in a single machine containing the entire sketch  $\mathcal{S}(G)$ .

This will be useful for us as the linear sketches of [Proposition 3.2](#) and [Proposition 3.3](#) used in our [Lemma 3.4](#) are vertex-incidence sketches.

*Proof of Corollary 1.2.* Because [Proposition 3.2](#) and [Proposition 3.3](#) are both vertex-incidence sketches that require  $O(\text{polylog}(n)/\varepsilon^2)$  bits for each vertex-neighborhood, this implies a simple  $\tilde{O}(n/\varepsilon^2)$ -size message MPC protocol which terminates with a single machine  $M_1$  containing  $S_1(G), S_2(G)$ , and  $S_3(G)$  as in [Lemma 3.4](#).

From here, we must only set  $\varepsilon = 1/\log(n)$  and have  $M_1$  run the recovery algorithm from [Theorem 1](#) to conclude our corollary. ■

## 5.4 Dynamic Streaming

Next, we provide the proof of [Corollary 1.3](#).

**Corollary** (Restatement of [Corollary 1.3](#)). *There is a polynomial-time randomized streaming algorithm for correlation clustering that uses  $\tilde{O}(n)$  memory when making a single pass over a dynamic stream, and for any constant  $\varepsilon > 0$ , with high probability, achieves an  $(\alpha_{\text{BEST}} + o(1))$ -approximation.*

*Proof.* Another very simple application of linear sketching is to dynamic streaming algorithms. At the beginning of the stream, the algorithm can sample the sketching matrix  $\mathcal{S}$  and compute  $\mathcal{S}(\emptyset)$  as the sketch. Then, whenever a new edge  $e$  is inserted or deleted, the algorithm can update the sketch by  $\mathcal{S}(+e)$  or  $\mathcal{S}(-e)$  using the linearity of the sketch. This way, at the end of the stream, the algorithm is left with a sketch of the final graph.

In our context, we simply run this approach using our sketching algorithm in [Theorem 1](#) and at the end, in polynomial time, return the solution. We note that similar to [Corollary 1.1](#), here also, we additionally exploit the fact that the description of the sketching matrix can be stored in  $\tilde{O}(n)$  bits. ■

## 5.5 Deterministic Algorithms for Insertion-Only Streams

Finally, we show that by leveraging known results on *deterministic* algorithms for spectral sparsification in insertion-only streams, we can achieve the following result.

**Corollary** (Restatement of [Corollary 1.4](#)). *There is a polynomial-time streaming algorithm for correlation clustering that uses  $\tilde{O}(n)$  memory to deterministically build a data structure  $D$  using a single pass over an insertion-only stream, and only at the end, uses randomization to, with high probability, recover from  $D$  an  $(\alpha_{\text{BEST}} + o(1))$ -approximation for correlation clustering.*

**Remark 5.3.** Note that the high probability in Corollary 1.4 is with respect to the post-processing, not the data structure  $D$  created during the stream itself, which is deterministic. Moreover because the algorithm is deterministic during the stream, it works even against an adversary that sees the internal state of the algorithm, namely, is adversarially robust in the strongest possible sense.

The key to the proof of Corollary 1.4 is the following lemma.

**Lemma 5.4.** *There is a polynomial time streaming algorithm that uses  $\tilde{O}(n/\varepsilon^2)$  bits of memory to deterministically build a data structure  $D$  using a single pass over an insertion-only stream of edges of  $G$ , and, only at the end, uses randomization to recover a simple graph  $H$  from  $D$  which is a  $(1 \pm \varepsilon)$  total weight preserving spectral sparsifier of  $G$  with high probability.*

Before proving this lemma, we show how it implies Corollary 1.4.

*Proof of Corollary 1.4.* We just run Lemma 5.4. The space required is only  $\tilde{O}(n/\varepsilon^2)$  bits.

By the statement of Lemma 5.4, this yields a simple graph  $H$  which is a  $(1 \pm \varepsilon)$  total weight preserving spectral sparsifier of  $G$ , and in particular, also a  $(1 \pm \varepsilon)$  cut sparsifier of  $G$ .

Now, on  $H$ , by Lemma 3.1, we see that for any clustering  $V_1, \dots, V_k$ ,

$$\text{CC}_H(V_1, \dots, V_k) \in (1 \pm 2\varepsilon)\text{CC}_G(V_1, \dots, V_k).$$

So, if we let  $\text{OPT}$  denote the minimum correlation clustering value, we know that

$$\text{OPT}(H) \leq (1 + 2\varepsilon) \cdot \text{OPT}(G).$$

Now, let us run the  $\alpha_{\text{BEST}}$ -approximation, polynomial time algorithm for correlation clustering on  $H$  (here, we are using the fact that  $H$  is simple). We are guaranteed that this recovers a partition  $\hat{P}$  such that

$$\text{CC}_H(\hat{P}) \leq \alpha_{\text{BEST}} \cdot \text{OPT}(H).$$

Finally, we note that the solution we recover satisfies

$$\text{CC}_G(\hat{P}) \leq (1 + 2\varepsilon)\text{CC}_H(\hat{P}) \leq (1 + 2\varepsilon) \cdot \alpha_{\text{BEST}} \cdot \text{OPT}(H) \leq \alpha_{\text{BEST}} \cdot (1 + 2\varepsilon)^2 \cdot \text{OPT}(G).$$

By setting  $\varepsilon = 1/\log(n)$ , the total space required is only  $\tilde{O}(n)$  bits, yet still recovers an  $(\alpha_{\text{BEST}} + o(1))$ -approximate solution to correlation clustering. ■

In the rest of this section, we focus on a proof of Lemma 5.4. The key building block here will be the existence of deterministic spectral sparsifiers of [BSS09]. It is also known that these can be leveraged into deterministic insertion-only spectral sparsification algorithms as follows.

**Proposition 5.5** (cf. [McG14]). *There is a deterministic single-pass streaming algorithm on insertion-only streams that computes a  $(1 \pm \varepsilon)$  spectral sparsifier using  $\tilde{O}(n/\varepsilon^2)$  space.*

Note that it is tempting to try to invoke the same reasoning that was used in the proof of Theorem 3 to argue that the deterministic spectral sparsification algorithm recovers edges with large effective resistance. Unfortunately however, this is not necessarily true with the algorithm of Proposition 5.5. Instead, our algorithm needs to explicitly recover these edges. To do this we first introduce a few definitions:

**Definition 5.6.** For a graph  $G$ , we say that a subgraph  $T \subseteq G$  is a  $\log(n)$ -spanner of  $G$ , if for every  $u, v$ :

$$\text{dist}_T(u, v) \leq \log(n) \cdot \text{dist}_G(u, v),$$

where  $\text{dist}_G(u, v)$  refers to the length of the shortest path between  $(u, v)$  in  $G$ .

**Definition 5.7.** We say that  $T_1, \dots, T_\ell$  are  $\ell$  form a sequence of disjoint  $\log(n)$ -spanners of  $G$  if  $\forall i \in [\ell]$ ,  $T_i$  is a  $\log(n)$ -spanner of the graph  $G - T_1 - T_2 - \dots - T_{i-1}$ .

We also rely on the following two propositions, one detailing a standard process for creating  $\log(n)$ -spanners, and another connecting  $\log(n)$ -spanners to effective resistance.

**Proposition 5.8** (cf. [ABS<sup>+</sup>20]). Let  $G$  be a graph, and let  $T$  be the result of iteratively removing an arbitrary single edge from any cycle of length at least  $\log(n)$  that remains in  $G$ . Then,  $T$  is an  $\log(n)$ -spanner of  $G$ , and  $T$  has only  $O(n)$  edges.

The work of [ADK<sup>+</sup>16] provided the following characterization of effective resistance:

**Proposition 5.9** ([ADK<sup>+</sup>16]). Let  $G$  be a simple graph, and let  $T_1, \dots, T_\ell$  be  $\ell$  be a sequence of disjoint  $\log(n)$  spanners. Then, for any edge  $e = (u, v) \in G - T_1 - \dots - T_\ell$ , the effective resistance between  $u$  and  $v$  is at most  $\frac{\log(n)}{\ell}$  in the graph  $T_1 \cup T_2 \cup \dots \cup T_\ell$ .

Using this, we can proceed to the proof of Lemma 5.4:

*Proof of Lemma 5.4.* Our data structure  $D$  will consist of a sequence of disjoint  $\log(n)$ -spanners as well as a deterministic spectral sparsifier of the remaining edges. Specifically, we create a sequence of disjoint  $\log(n)$ -spanners by processing edges as they arrive in the stream. This procedure is straightforward: whenever a new edge  $e$  arrives, we attempt to insert  $e$  into the first spanner  $T_1$ . If this creates a cycle of length  $\leq \log(n)$  in  $T_1$ , then we remove  $e$  from  $T_1$  and instead try inserting it into  $T_2$ , and so on. If  $e$  does not get included into  $T_1, \dots, T_\ell$ , then we insert it into the algorithm of Proposition 5.5.

Since this process is exactly implementing Proposition 5.8, each  $T_i$  is a  $\log(n)$ -spanner of  $G - T_1 - \dots - T_{i-1}$  (again, see [ABS<sup>+</sup>20] for a discussion). In particular, this means that every edge  $e \in G - \bigcup_{i=1}^\ell T_i$  has effective resistance  $\leq \frac{\log(n)}{\ell}$  in  $G$  by Proposition 5.9.

At this point, the algorithm has recovered the sequence of  $\ell$  spanners  $T_1, \dots, T_\ell$ , as well as a  $(1 \pm \varepsilon)$  spectral sparsifier of  $G - \bigcup_{i=1}^\ell T_i$ , which we denote by  $\widetilde{G - T}$ . However, observe that we still cannot use the convex program of Lemma 3.5. Although we are guaranteed that every edge in  $G - \bigcup_{i=1}^\ell T_i$  has small effective resistance, if we just run the original convex program, it is possible that it assigns fractional mass to edges which are not in the original graph  $G$ , and therefore have large effective resistances (and hence the randomized rounding scheme would not work). Instead, given the spanners  $T_1, \dots, T_\ell$ , we define a new set  $\hat{E} \subseteq \binom{V}{2}$  as

$$\hat{E} = \left\{ e \in \binom{V}{2} : e \notin \bigcup_{i=1}^\ell T_i \wedge R_{\text{eff}, (\bigcup_{i=1}^\ell T_i)}(e) \leq \frac{\log(n)}{\ell} \right\}.$$

Here, we use  $R_{\text{eff}, (\bigcup_{i=1}^\ell T_i)}(e)$  to denote the effective resistance of an edge  $e$  in the graph formed only by the edges contained in the sequence of spanners  $T_1, \dots, T_\ell$ . We can now introduce a modified convex program whose solution support is guaranteed to be in  $\hat{E}$  while remaining feasible:

$$\begin{aligned}
Y_e &\in [0, 1] \quad \forall e \in \hat{E}, \\
\sum_{e \in \binom{V}{2}} Y_e \cdot z^T L_e z &\geq (1 - \varepsilon) z^T L_{\widetilde{G-T}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1, \\
\sum_{e \in \binom{V}{2}} Y_e \cdot z^T L_e z &\leq (1 + \varepsilon) z^T L_{\widetilde{G-T}} z \quad \forall z \in \mathbb{R}^n : \|z\|_2 = 1, \\
\sum_{e \in \binom{V}{2}} Y_e &= m - \sum_{i=1}^{\ell} |T_i|.
\end{aligned}$$

**Claim 5.10.** *The convex program above is feasible.*

*Proof.* This follows because all edges  $G - \bigcup_{i=1}^{\ell} T_i$  are in  $\hat{E}$  (Proposition 5.9), and hence constitute a  $(1 \pm \varepsilon)$  spectral sparsifier of  $\widetilde{G-T}$ . Moreover, this solution also preserves the total weight, and only uses  $\{0, 1\}$  valued edge weights. ■

We next observe that the separation oracle for the above convex program is efficiently implementable:

**Claim 5.11.** *There is a polynomial-time separation oracle for the above convex program.*

*Proof.* This follows from essentially same reasoning as done in Lemma 3.5. Indeed certifying the constraints that  $Y_e \in [0, 1]$  and that  $\sum_{e \in \binom{V}{2}} Y_e = m - \sum_{i=1}^{\ell} |T_i|$  are both trivial. Thus, it remains only to check the spectral approximation conditions.

This can be done by checking the eigenvalues of  $L_{\widetilde{G-T}}^{\dagger/2} L_{\hat{G}} L_{\widetilde{G-T}}^{\dagger/2}$  using the same logic as Lemma 3.5 and Claim 4.4. ■

Now, because of Claim 5.10 and Claim 5.11, we can use the ellipsoid method to find a feasible solution in polynomial-time [GLS88]. So, let  $\hat{G}$  then denote this feasible solution recovered by the above program, where the edge set is  $\hat{E}$ , and the corresponding weight on each edge  $e \in \hat{E}$  is  $Y_e$ . Observe that  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$  is a fractional  $(1 \pm \varepsilon)$  spectral sparsifier of  $\widetilde{G-T} \cup \bigcup_{i=1}^{\ell} T_i$ , and thus a fractional total weight preserving  $(1 \pm 3\varepsilon)$  spectral sparsifier of  $G$ .

All that remains is to show that efficiently rounding our fractional  $(1 \pm \varepsilon)$  spectral sparsifier  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$  is possible. By Proposition 4.1, sampling every edge  $e$  in  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$  with probability  $p_e \geq \frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot R_{\text{eff}, \hat{G} \cup \bigcup_{i=1}^{\ell} T_i}(e)$ , and giving weight  $\frac{w_e}{p_e}$  yields a  $(1 \pm \varepsilon)$  spectral-sparsifier with high probability.

By setting  $\ell = \frac{C \log^2(n)}{\varepsilon^2}$  (using the constant  $C$  in Proposition 4.1), it follows that every edge  $e \in \hat{E}$  will have effective resistance  $\leq \frac{\log(n)\varepsilon^2}{C \log^2(n)} = \frac{\varepsilon^2}{C \log(n)}$  in  $\bigcup_{i=1}^{\ell} T_i$ . Because effective resistance only decreases as one adds more edges, it also follows that every edge  $e \in \hat{E}$  will have effective resistance  $\leq \frac{\varepsilon^2}{C \log(n)}$  in  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$ . Thus, in the graph  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$ , if we keep every edge  $\bigcup_{i=1}^{\ell} T_i$  with probability 1, and then sample every edge  $e \in \hat{G}$  with probability  $\frac{C \log(n)}{\varepsilon^2} \cdot w_e \cdot \frac{\varepsilon^2}{C \log(n)} = w_e$ , we will get with high probability a  $(1 \pm \varepsilon)$  spectral sparsifier of  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$ . We denote this resulting simple graph by  $H$ . Now since  $\hat{G} \cup \bigcup_{i=1}^{\ell} T_i$  is a  $(1 \pm 3\varepsilon)$  spectral sparsifier to  $G$ , it follows that  $H$

is a  $(1 \pm 5\epsilon)$  spectral sparsifier of  $G$ . Thus running the above procedure with an error parameter of  $\epsilon/5$  yields a spectral sparsifier with desired accuracy.

Finally, because we are performing the simple, independent Bernoulli rounding, we can repeat this procedure  $n^3$  times and be guaranteed by [Claim 3.8](#) that in some round, the total weight is exactly preserved. The memory required by the algorithm follows from the  $\tilde{O}(n/\epsilon^2)$  edges stored in the spanners (by our choice of  $\ell$  and [Proposition 5.8](#)), and the complexity of the deterministic spectral sparsifier ([Proposition 5.5](#)). This yields the lemma.  $\blacksquare$

## 6 Acknowledgements

Part of this work was conducted while the authors were visiting the Simons Institute for the Theory of Computing as part of the Sublinear Algorithms program.

## References

- [ABB<sup>+</sup>19] P. Awasthi, A. Bakshi, M. Balcan, C. White, and D. P. Woodruff. Robust communication-optimal distributed clustering algorithms. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICS*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. [doi:10.4230/LIPIcs.ICALP.2019.18](https://doi.org/10.4230/LIPIcs.ICALP.2019.18). 4
- [ABS<sup>+</sup>20] A. R. Ahmed, G. Bodwin, F. D. Sahneh, K. Hamm, M. J. L. Jebelli, S. G. Kobourov, and R. Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020. [doi:10.1016/J.COSREV.2020.100253](https://doi.org/10.1016/J.COSREV.2020.100253). 21
- [ACG<sup>+</sup>15] K. J. Ahn, G. Cormode, S. Guha, A. McGregor, and A. Wirth. Correlation clustering in data streams. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2237–2246. JMLR.org, 2015. 1, 2, 6, 8
- [ACN08] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. [doi:10.1145/1411509.1411513](https://doi.org/10.1145/1411509.1411513). 2, 6
- [ADK<sup>+</sup>16] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng. On fully dynamic graph sparsifiers. In I. Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 335–344. IEEE Computer Society, 2016. [doi:10.1109/FOCS.2016.44](https://doi.org/10.1109/FOCS.2016.44). 21
- [AGM12a] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In Y. Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012. [doi:10.1137/1.9781611973099.40](https://doi.org/10.1137/1.9781611973099.40). 1, 5, 8, 9, 18
- [AGM12b] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In M. Benedikt, M. Krötzsch, and M. Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–12. ACM, 2012. 1

*Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012.  
doi:10.1145/2213556.2213560. 1, 19

- [AKLP22] A. Agarwal, S. Khanna, H. Li, and P. Patil. Sublinear algorithms for hierarchical clustering. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. URL [http://papers.nips.cc/paper\\_files/paper/2022/hash/16466b6c95c5924784486ac5a3feeb65-Abstract-Conference.html](http://papers.nips.cc/paper_files/paper/2022/hash/16466b6c95c5924784486ac5a3feeb65-Abstract-Conference.html). 19
- [AKLY16] S. Assadi, S. Khanna, Y. Li, and G. Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In R. Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.CH93. 1
- [AS23] S. Assadi and V. Shah. Tight bounds for vertex connectivity in dynamic streams. In T. Kavitha and K. Mehlhorn, editors, *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 213–227. SIAM, 2023. doi:10.1137/1.9781611977585.CH20. 1
- [AW22] S. Assadi and C. Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In M. Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICS*, pages 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.10. 6
- [BBPP12] P. S. Bonsma, H. Broersma, V. Patel, and A. V. Pyatkin. The complexity of finding uniform sparsest cuts in various graph classes. *J. Discrete Algorithms*, 14:136–149, 2012. doi:10.1016/J.JDA.2011.12.008. 30
- [BBW24] A. Bernstein, G. Bodwin, and N. Wein. Are there graphs whose shortest path structure requires large edge weights? In V. Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPICS*, pages 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.12. 4
- [BCD<sup>+</sup>25] A. Bernstein, J. Chen, A. Dudeja, Z. Langley, A. Sidford, and T. Tu. Matching composition and efficient weight reduction in dynamic matching. In Y. Azar and D. Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2991–3028. SIAM, 2025. doi:10.1137/1.9781611978322.97. 4
- [BCMT22] S. Behnezhad, M. Charikar, W. Ma, and L. Tan. Almost 3-approximate correlation clustering in constant rounds. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 720–731. IEEE, 2022. doi:10.1109/FOCS54457.2022.00074. 6
- [BCMT23] S. Behnezhad, M. Charikar, W. Ma, and L. Tan. Single-pass streaming algorithms for correlation clustering. In N. Bansal and V. Nagarajan, editors, *Proceedings of the 2023*

*ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 819–849. SIAM, 2023. doi:10.1137/1.9781611977554.CH33. 1, 2, 6, 8

- [BDL21] A. Bernstein, A. Dudeja, and Z. Langley. A framework for dynamic matching in weighted graphs. In S. Khuller and V. V. Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 668–681. ACM, 2021. doi:10.1145/3406325.3451113. 4
- [BJWY22] O. Ben-Eliezer, R. Jayaram, D. P. Woodruff, and E. Yoge. A framework for adversarially robust streaming algorithms. *J. ACM*, 69(2):17:1–17:33, 2022. doi:10.1145/3498334. 5
- [BK96] A. A. Benczúr and D. R. Karger. Approximating  $s$ - $t$  minimum cuts in  $\tilde{O}(n^2)$  time. In G. L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827. 1, 7
- [BKS17] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. *J. ACM*, 64(6):40:1–40:58, 2017. doi:10.1145/3125644. 4, 5
- [BSS09] J. D. Batson, D. A. Spielman, and N. Srivastava. Twice-ramanujan sparsifiers. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 255–262. ACM, 2009. doi:10.1145/1536414.1536451. 20
- [CCL<sup>+</sup>24] N. Cao, V. Cohen-Addad, E. Lee, S. Li, A. Newman, and L. Vogl. Understanding the cluster linear program for correlation clustering. In B. Mohar, I. Shinkar, and R. O’Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1605–1616. ACM, 2024. doi:10.1145/3618260.3649749. 2, 6
- [CDK14] F. Chierichetti, N. N. Dalvi, and R. Kumar. Correlation clustering in mapreduce. In S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’14, New York, NY, USA - August 24 - 27, 2014*, pages 641–650. ACM, 2014. doi:10.1145/2623330.2623743. 6
- [CGS22] A. Chakrabarti, P. Ghosh, and M. Stoeckl. Adversarially robust coloring for graph streams. In M. Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICS*, pages 37:1–37:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.37. 5
- [CGW03] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 524–533. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238225. 2
- [CKL22] Y. Chen, S. Khanna, and H. Li. On weighted graph sparsification by linear sketching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*

2022, Denver, CO, USA, October 31 - November 3, 2022, pages 474–485. IEEE, 2022.  
doi:[10.1109/FOCS54457.2022.00052](https://doi.org/10.1109/FOCS54457.2022.00052). 8

- [CKL<sup>+</sup>24] M. Cambus, F. Kuhn, E. Lindy, S. Pai, and J. Uitto. A  $(3 + \varepsilon)$ -approximate correlation clustering algorithm in dynamic streams. In D. P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2861–2880. SIAM, 2024.  
doi:[10.1137/1.9781611977912.101](https://doi.org/10.1137/1.9781611977912.101). 4, 6
- [CLLN23] V. Cohen-Addad, E. Lee, S. Li, and A. Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1082–1104. IEEE, 2023.  
doi:[10.1109/FOCS57990.2023.00065](https://doi.org/10.1109/FOCS57990.2023.00065). 6
- [CLM<sup>+</sup>21] V. Cohen-Addad, S. Lattanzi, S. Mitrovic, A. Norouzi-Fard, N. Parotsidis, and J. Tarnawski. Correlation clustering in constant many parallel rounds. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021. 6
- [CLN22] V. Cohen-Addad, E. Lee, and A. Newman. Correlation clustering with sherali-adams. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 651–661. IEEE, 2022.  
doi:[10.1109/FOCS54457.2022.00068](https://doi.org/10.1109/FOCS54457.2022.00068). 6
- [CLP<sup>+</sup>24] V. Cohen-Addad, D. R. Lolck, M. Pilipczuk, M. Thorup, S. Yan, and H. Zhang. Combinatorial correlation clustering. In B. Mohar, I. Shinkar, and R. O’Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1617–1628. ACM, 2024.  
doi:[10.1145/3618260.3649712](https://doi.org/10.1145/3618260.3649712). 2, 4, 6
- [CMSY15] S. Chawla, K. Makarychev, T. Schramm, and G. Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 219–228. ACM, 2015.  
doi:[10.1145/2746539.2746604](https://doi.org/10.1145/2746539.2746604). 6
- [CSWZ16] J. Chen, H. Sun, D. P. Woodruff, and Q. Zhang. Communication-optimal distributed clustering. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3720–3728, 2016. 4
- [DEFI06] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006.  
doi:[10.1016/J.TCS.2006.05.008](https://doi.org/10.1016/J.TCS.2006.05.008). 1
- [DMM24] M. Dalirrooyfard, K. Makarychev, and S. Mitrovic. Pruned pivot: Correlation clustering algorithm for dynamic, parallel, and local computation models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. 6

- [GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4. 12, 17, 22
- [HST12] M. Hardt, N. Srivastava, and M. Tulsiani. Graph densification. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 380–392. ACM, 2012. doi:10.1145/2090236.2090266. 3
- [Kar93] D. R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In V. Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. URL <http://dl.acm.org/citation.cfm?id=313559.313605>. 12
- [KLM<sup>+</sup>14a] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.66. 1, 3, 5
- [KLM<sup>+</sup>14b] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.66. 9, 15, 18
- [KPS24] S. Khanna, A. Puterman, and M. Sudan. Near-optimal size linear sketches for hypergraph cut sparsifiers. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 1669–1706. IEEE, 2024. doi:10.1109/FOCS61266.2024.00105. 5, 9, 18
- [KSV10] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In M. Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010. doi:10.1137/1.9781611973075.76. 4
- [MC23] K. Makarychev and S. Chakrabarty. Single-pass pivot algorithm for correlation clustering. keep it simple! In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. 6
- [McG14] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694. 20
- [MTVV15] A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu. Densest subgraph in dynamic graph streams. In G. F. Italiano, G. Pighizzini, and D. Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 472–482. Springer, 2015. doi:10.1007/978-3-662-48054-0\_39. 1

- [RVW18] T. Roughgarden, S. Vassilvitskii, and J. R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6):41:1–41:24, 2018. [doi:10.1145/3232536](https://doi.org/10.1145/3232536). 5
- [SS11] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011. [doi:10.1137/080734029](https://doi.org/10.1137/080734029). 14
- [ST11] D. A. Spielman and S. Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. [doi:10.1137/08074489X](https://doi.org/10.1137/08074489X). 3, 7
- [TT23] W. Tang and F. Tang. The poisson binomial distribution—old & new. *Statistical Science*, 38(1):108–119, 2023. 13
- [ZZL<sup>+</sup>19] C. J. Zhu, T. Zhu, K. Lam, S. Han, and J. Bi. Communication-optimal distributed dynamic graph clustering. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 5957–5964. AAAI Press, 2019. 4

# Appendix

## A Omitted Proofs

### A.1 Proof of Lemma 3.1

We now provide a proof of Lemma 3.1, restated below.

**Lemma (Lemma 3.1).** *Let  $G$  and  $H$  be graphs on the same vertex set such that  $H$  is a  $(1 \pm \varepsilon)$  total weight preserving cut sparsifier of  $G$ . Then, for any partition  $V_1, \dots, V_k$  of vertices,*

$$\text{CC}_H(V_1, \dots, V_k) \in (1 \pm 2\varepsilon) \cdot \text{CC}_G(V_1, \dots, V_k).$$

*Proof.* Recall that by Definition 2.1,

$$\text{CC}_G(V_1, \dots, V_k) = \sum_{i \in [k]} |E^-(V_i)| + |E^+(V_1, \dots, V_k)|,$$

Observe that we can re-write  $|E^+(V_1, \dots, V_k)|$  as

$$|E^+(V_1, \dots, V_k)| = \frac{1}{2} \cdot \sum_{i \in [k]} \text{cut}_G(V_i),$$

as each crossing edge will be present in exactly two of the  $V_i$  cuts. Similarly,

$$|E^-(V_i)| = \binom{|V_i|}{2} - \frac{1}{2} \left( \sum_{v \in V_i} \deg(v) - \text{cut}_G(V_i) \right),$$

as the second term above counts the number of edges inside each  $V_i$ . Together, these mean

$$\text{CC}_G(V_1, \dots, V_k) = \sum_{i \in [k]} \text{cut}_G(V_i) + \sum_{i \in [k]} \binom{|V_i|}{2} - \frac{1}{2} \sum_{v \in V} \deg(v).$$

Moreover, since  $G$  and  $H$  have the same total weight, we have

$$\sum_{i \in [k]} \binom{|V_i|}{2} - \frac{1}{2} \sum_{v \in V} \deg_H(v) = \sum_{i \in [k]} \binom{|V_i|}{2} - \frac{1}{2} \sum_{v \in V} \deg_G(v).$$

Thus, the only difference between  $\text{CC}_G$  and  $\text{CC}_H$  is in the terms  $\sum_{i \in [k]} \text{cut}_G(V_i)$  vs.  $\sum_{i \in [k]} \text{cut}_H(V_i)$ . Because  $H$  is a  $(1 \pm \varepsilon)$  cut sparsifier of  $G$ , it follows that

$$|\text{CC}_H(V_1, \dots, V_k) - \text{CC}_G(V_1, \dots, V_k)| = \left| \sum_{i \in [k]} \text{cut}_H(V_i) - \sum_{i \in [k]} \text{cut}_G(V_i) \right| \leq \varepsilon \cdot \sum_{i \in [k]} \text{cut}_G(V_i).$$

Finally, we also have

$$\text{CC}_G(V_1, \dots, V_k) \geq \sum_{i \in [k]} |E^+(V_1, \dots, V_k)| = \frac{1}{2} \cdot \sum_{i \in [k]} \text{cut}_G(V_i),$$

and combining the previous two equations gives us,

$$|\text{CC}_H(V_1, \dots, V_k) - \text{CC}_G(V_1, \dots, V_k)| \leq 2\varepsilon \cdot \text{CC}_G(V_1, \dots, V_k),$$

concluding the proof. ■

## A.2 NP-Hardness of Certifying Cut Sparsification

As stated in [Section 1](#), it was important for us to work with spectral sparsifiers in our de-sparsification paradigm even though for our application, cut sparsifiers would have sufficed also. Intuitively, the reason for this is that it is simple to *certify* that two graphs  $G_1, G_2$  are spectral sparsifiers of one another. Naturally then, one may wonder if the same is true about certifying whether two graphs  $G_1, G_2$  are *cut* sparsifiers of one another. We now show that this is not true. Indeed, we show that being able to certify that arbitrary graphs are  $(1 \pm \varepsilon)$  cut sparsifiers of each other is **NP-hard** in general.

As a starting point, we recall the following about the sparsest cut problem:

**Definition A.1.** *For an arbitrary graph  $G = (V, E)$ , and a subset  $S \subseteq V$ , we say that the sparsity of the cut  $S$  is*

$$\Phi(S) = \frac{|E[S, \bar{S}]|}{|S||\bar{S}|}.$$

*The sparsity of the graph is defined as*

$$\Phi(G) = \min_{S \neq \emptyset, V} \Phi(S).$$

We will rely on the following result:

**Proposition A.2** ([BBPP12]). *It is NP-hard to calculate  $\Phi(G)$  on unweighted simple graphs  $G$ .*

Using this we prove the following result (as stated in [Section 1](#), we believe this result is folklore but we know no reference for it and as such are proving it here for completeness).

**Proposition A.3.** *It is NP-hard to check if two graphs are  $(1 \pm \varepsilon)$  cut sparsifiers of one another.*

*Proof.* Let  $G$  be an arbitrary simple graph for which we wish to approximate  $\Phi(G)$ .

Next, let us consider the complete graph  $K_n$ . In order for our graph  $G$  to be a  $(1 \pm \varepsilon)$  cut sparsifier of  $K_n$  it must be the case that for every  $S \subseteq V$

$$(1 - \varepsilon)\text{cut}_{K_n}(S) \leq \text{cut}_G(S) \leq (1 + \varepsilon)\text{cut}_{K_n}(S).$$

In particular however, we know that  $\text{cut}_{K_n}(S) = |S||\bar{S}|$ , and further,  $\text{cut}_G(S) \leq (1 + \varepsilon)\text{cut}_{K_n}(S)$  by definition, as  $K_n$  contains all edges (and more) of  $G$ . Thus, our graph  $G$  is a cut sparsifier of  $K_n$  if and only if for every  $S \subseteq V$ :

$$\text{cut}_G(S) \geq (1 - \varepsilon)|S||\bar{S}|,$$

equivalently, if and only if

$$\Phi(S) = \frac{|E[S, \bar{S}]|}{|S||\bar{S}|} \geq (1 - \varepsilon).$$

Now, suppose we have an algorithm which efficiently certifies whether  $G$  is a  $(1 \pm \varepsilon)$  cut sparsifier of  $K_n$  for any value of  $\varepsilon$ , and let us denote this algorithm by  $A(G, K_n, \varepsilon)$ . To start, we can query with  $\varepsilon = 1 - \frac{1}{n^2}$ . This will always return 1, as we are simply certifying that

$$\text{cut}_G(S) \geq \frac{1}{n^2}|S||\bar{S}|,$$

which is equivalent to checking that  $G$  is connected. Now, our goal is to calculate  $\Phi(G)$ , or equivalently, to find  $\min_{S \neq \emptyset, V} \frac{|E[S, \bar{S}]|}{|S||\bar{S}|}$ . Observe that this expression can only take on  $\leq n^3$  values, as the number of edges in  $E[S, \bar{S}]$  is  $\leq n^2$ , and the denominator can only take on  $n/2$  values (one for each value of  $|S| = 1, \dots, n/2$ ). So, there exists a set of  $\leq n^3$  possible values for  $\Phi(S)$ , which we denote by  $Q$ . Our goal will be to find the smallest value  $q \in Q$  for which there exists an  $S$  such that  $\Phi(S) = q$ .

So, let us sort the set of possible values  $Q$ , and denote the possible values by  $q_1, \dots, q_{\leq n^3}$ . To start, we set  $\varepsilon = (1 - q_2)$ , and query  $A(G, K_n, \varepsilon)$ . Then, we are certifying whether  $\forall S \subseteq V$ ,

$$\Phi(S) \geq q_2.$$

If this fails, then there must be a cut with smaller sparsity, and hence  $\Phi(G) = q_1$ . Otherwise, we instead query with  $\varepsilon = 1 - q_3$ , then  $\varepsilon = 1 - q_4$ ,  $\varepsilon = 1 - q_5$ , etc. until we find a value  $\varepsilon = 1 - q_i$  such that  $G$  is not a  $(1 \pm \varepsilon)$  cut sparsifier of  $K_n$ . For this value, we know that  $\forall S$ ,

$$\Phi(S) \geq q_{i-1},$$

but that there exists an  $S$  such that

$$\Phi(S) < q_i.$$

Thus,  $\Phi(G)$  will be exactly  $q_{i-1}$ . This requires only  $\text{poly}(n)$  oracle calls to the cut-certification algorithm  $A$ . Thus, calculating the sparsest cut efficiently reduces to cut certification, meaning the latter must be **NP-hard**. ■