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

Sepehr Assadi
sepehr@assadi.info
University of Waterloo
Waterloo, ON, Canada

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

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

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 $\widetilde{O}(n)$ size for n-vertex graphs, from which one can recover with high probability an $(\alpha + o(1))$-approximate correlation clustering in polynomial time, where α 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 $\widetilde{O}(n)$ 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.

CCS Concepts

• **Theory of computation** → **Sketching and sampling**.

Keywords

Correlation clustering, sparsification

ACM Reference Format:
Sepehr Assadi, Sanjeev Khanna, and Aaron Putterman. 2025. Correlation Clustering and (De)Sparsification: Graph Sketches Can Match Classical Algorithms. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3717823.3718194

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[1] 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 full version of the paper will appear on arxiv.

[1] By a non-edge, we mean a pair of vertices with no edges between them in the graph.

STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718194

A highly successful paradigm here, especially when it comes to flexibility and portability across different models, is *graph sketching* pioneered by [2]: 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 [2], vertex connectivity [6], cut sparsification [3], spectral sparsification [33], densest subgraph [39], maximum matchings [5], and subgraph counting [3], 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 [11] (building on [1]): 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* [13], which are weighted subgraphs of the input with only $\widetilde{O}(n/\epsilon^2)$ edges that preserve the value of every cut to within a $(1 \pm \epsilon)$ factor, and already admit efficient graph sketches [3]. Thus, we can also find $(1+\epsilon)$-approximate correlation clusterings using graph sketching.

There is a 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 [30][2]. 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; 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 [4] but also recently more improved combinatorial algorithms in [28]). 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 α_{BEST} denote the best approximation ratio possible for correlation clustering in polynomial time (via classical algorithms), which satisfies

$$1 + \Omega(1) \underset{[20]}{\leq} \alpha_{\text{BEST}} \underset{[18]}{\leq} 1.437. \tag{1}$$

due to the APX-hardness established in [20] and the recent approximation algorithm of [18].

Result 1. *For any n-vertex graph G, there is a randomized linear sketch of $\widetilde{O}(n)$ 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 Desparsification?

Let us revisit the approaches of [1, 11] that designed $(1+\epsilon)$-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[3] 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 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" [32]. 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. This suggests that

[2] In fact, it is shown by [30] 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.

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

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* [42] that are generally known to be a robust strengthening of cut sparsifiers?[4]

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

Result 2. *For any n-vertex unweighted graph G and any $\epsilon \in (0,1)$, there is a randomized linear sketch of $\widetilde{O}(n/\epsilon^2)$ 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 $(1\pm\epsilon)$-spectral sparsifier[a] of G.*

[a] 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 $\epsilon = o(1)$, and then running any α-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 [33] 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 $\epsilon \in (0,1)$ and n-vertex unweighted graph G, there is a randomized polynomial-time algorithm that given any $(1\pm\epsilon)$-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 $(1\pm 2\epsilon)$-cut sparsifier of G, provided the minimum cut in G is $\Omega(\log n/\epsilon^2)$.*
- *(b) an n-vertex unweighted graph $\tilde{G}$ with the same number of edges as G such that $\tilde{G}$ is a $(1\pm 2\epsilon)$-spectral sparsifier of G, provided maximum effective resistance in G is $O(\epsilon^2/\log n)$.*

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 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 [14], or for approximating weighted matching via unweighted matching algorithms [15, 16].

[4] Formally, using L_G and L_H to denote the Laplacian matrix of G and H, respectively, a cut sparsifier H of G satisfies $x^\top \cdot L_H \cdot x = (1\pm\epsilon)\cdot x^\top L_G \cdot x$ 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 $(1\pm\epsilon)$ factor of each other, which can be easily done in polynomial time.

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 α_{BEST} defined in Equation (1).

Our first algorithm is in the distributed communication model, studied for various clustering problems in [8, 22, 44] (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 $\widetilde{O}(nk)$ 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 [9, 36]. 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 $\widetilde{O}(n)$-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 $\widetilde{O}(n)$-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 [28] (given Equation (1)), although we note that the algorithm of [28] runs in $O(1)$ rounds even when memory per machine is n^δ 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 $\widetilde{O}(n)$ 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 [17] in dynamic streams and 1.84-approximation algorithm of [28] 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 [12] in the strongest possible sense as it works even against an adversary that sees its internal state; see also [19].

Corollary 1.4. *There is a polynomial-time streaming algorithm for correlation clustering that uses $\widetilde{O}(n)$ 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 $\widetilde{O}(n)$ (but not the internal memory) which is inline with the original definitions in [9] (see also [40]) 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 $\widetilde{O}(n)$ 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[5].

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 3.9) relying on the assumption that minimum cut is not too small, or, following the standard effective resistance sampling approach (see Proposition 4.2) 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 [2, 37] 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 [33] 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).

Finally, to obtain Result 1 from our Result 2, we follow previous arguments in [1, 11] 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 [1, 11]), 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, [25] designed $O(1)$-round MPC algorithms for correlation clustering, and building on this, [7] obtained single-pass streaming and sublinear time $O(1)$-approximation algorithms for this problem[6] (see also [1, 24] and references therein for earlier work on this problem). These results were subsequently improved in a series of work in [10, 11, 17, 28, 29, 38] culminating in the work of [28] 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 $(3+\epsilon)$-approximation were also developed in [17, 38] by adapting the landmark Pivot algorithm of [4] 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 [4], which was then improved to a 2.06 [21]. Recently, [27] broke the 2-approximation barrier—the integrality gap of the LP of [4]—and achieved a 1.995-approximation which was then improved to a 1.73-approximation in [26] and subsequently 1.437-approximation in [18]. Finally, [28] 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 $w_G : E \to \mathbb{R}$ to denote the weights. We often treat unweighted graphs as weighted graphs with weight

[5] 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.

[6] The constants in these algorithms are quite large, around 700 for [25] and more than 10000 for [7].

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 $a \in (1 \pm \epsilon)b$ to mean that $(1-\epsilon)b \le a \le (1+\epsilon)b$.

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.

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 [13] and spectral sparsifiers [42]. We review their definitions here.

Cut sparsifiers. A basic notion of sparsification is *cut sparsification* introduced by [13].

Definition 2.2 ([13]). *Given a graph $G = (V, E)$ and $\epsilon \in (0, 1)$, a graph $\widetilde{G}$ is said to be a $(1 \pm \epsilon)$ **cut sparsifier** of G iff for every cut $S \subseteq V$,*

$$(1-\epsilon) \cdot \text{cut}_G(S) \le \text{cut}_{\widetilde{G}}(S) \le (1+\epsilon) \cdot \text{cut}_G(S).$$

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* [42].

Definition 2.4 ([42]). *Given a graph $G = (V, E)$, a graph $\widetilde{G}$ is considered a $(1 \pm \epsilon)$ **spectral sparsifier** of G iff for every vector $x \in \mathbb{R}^V$,*

$$(1-\epsilon) \cdot x^\top L_G x \le x^\top L_{\widetilde{G}} x \le (1+\epsilon) \cdot x^T L_G x,$$

where L_G and $L_{\widetilde{G}}$ denote the Laplacian matrix of G and $\widetilde{G}$, 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:*

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

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

Definition 2.6. *Given a $(1 \pm \epsilon)$ cut/spectral sparsifier $\widetilde{G}$ of a graph $G = (V, E)$, we say that $\widetilde{G}$ is **total weight preserving** if it additionally satisfies $\sum_{e \in G} w_G(e) = \sum_{e \in \widetilde{G}} w_{\widetilde{G}}(e)$. Similarly, we say $\widetilde{G}$ 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 [2] (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 [23] for more details).

3 Desparsification for Correlation Clustering: Proof of Result 1

In this section, we prove the following theorem that formalizes Result 1.

Theorem 3.1. *Let α_{BEST} be the best possible approximation ratio for correlation clustering on simple graphs in polynomial time. There is a linear sketch of size $\widetilde{O}(n)$ 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 3.1 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 3.2. *There is a (randomized) linear sketch using $\widetilde{O}(n/\epsilon^2)$ bits of space which, for any simple graph G, can be used to recover with high probability a simple, $(1 \pm \epsilon)$ 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 3.1 using Theorem 3.2. We then switch to the proof of Theorem 3.2 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 3.3. *Let G and H be graphs on the same vertex set such that H is a $(1 \pm \epsilon)$ total weight preserving cut sparsifier of G. Then, for any partition $V_1, \ldots V_k$ of vertices,*

$$\mathrm{CC}_H(V_1, \ldots V_k) \in (1 \pm 2\epsilon) \cdot \mathrm{CC}_G(V_1, \ldots V_k).$$

We note that similar but not identical statements as Lemma 3.3 have been used in prior work in [1, 11]; as such, we omit the proof of this lemma, and leave it to the full version of the paper. With this lemma, we can immediately obtain Theorem 3.1, assuming Theorem 3.2.

Proof of Theorem 3.1. The linear sketch is exactly the one of Theorem 3.2. Let $\widetilde{G}$ denote the recovered simple graph which is a $(1 \pm \epsilon)$ total weight preserving cut sparsifier of G. By Lemma 3.3, we see that for any clustering $V_1, \ldots V_k$,

$$\mathrm{CC}_{\widetilde{G}}(V_1, \ldots V_k) \in (1 \pm 2\epsilon) \cdot \mathrm{CC}_G(V_1, \ldots V_k).$$

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

$$\mathrm{OPT}(\widetilde{G}) \leq (1 + 2\epsilon) \cdot \mathrm{OPT}(G).$$

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

$$\mathrm{CC}_{\widetilde{G}}(P) \leq \alpha \cdot \mathrm{OPT}(\widetilde{G}).$$

Returning P as the answer on G, by Lemma 3.3 satisfies

$$\mathrm{CC}_G(P) \leq (1 + 2\epsilon) \cdot \mathrm{CC}_{\widetilde{G}}(P) \leq$$

$$(1 + 2\epsilon) \cdot \alpha \cdot \mathrm{OPT}(\widetilde{G}) \leq \alpha \cdot (1 + 2\epsilon)^2 \cdot \mathrm{OPT}(G).$$

Finally, by setting $\epsilon = o(1)$, the linear sketch we use requires only $\widetilde{O}(n)$ 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 3.2

We now switch to proving Theorem 3.2 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 $\lambda = \Theta(\log(n)/\epsilon^2)$ to be chosen later, can be used to (exactly) recover all edges of strength at most λ in the graph G, denoted by $S_1(G)$. This is done via the following result.

Proposition 3.4 (cf. [2],[37, 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 $\widetilde{O}(n\lambda)$ space.*

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

Proposition 3.5 [34]). *For any given $\epsilon \in (0, 1)$, there is a linear sketch for (unweighted) graphs G on n vertices for recovering a $(1 \pm \epsilon)$ spectral sparsifier of G with high probability in polynomial time, using $\widetilde{O}(n/\epsilon^2)$ 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 these three linear sketches can be used to recover structural information about G that will be sufficient for de-sparsification.

Lemma 3.6. *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 $(1 \pm \epsilon)$-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 $\widetilde{O}(n\lambda + n/\epsilon^2)$ bits.

Proof. We start by using $S_1(G)$ to recover the edges of strength at most λ in G, denoted by T. Importantly, because we recover *only* these edges, the resulting graph $G - T$ has all edges of strength greater than λ. This step implicitly partitions the vertex set V into $V_1, V_2, ..., V_k$ such that for $i \in [k]$, the subgraph of $G - T$ induced by V_i has minimum cut greater than λ. We rely here on the basic property of edge strengths, namely, the certificate of an edge having strength greater than λ in G never uses an edge of strength at most λ. 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 3.5, we can recover a $(1 \pm \epsilon)$-spectral sparsifier for each of $G[V_1], G[V_2], ..., G[V_k]$.

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 3.4 and Proposition 3.5, 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 $\widetilde{G}$. We say $\widetilde{G}$ is fractional in the sense that every edge $e \in \widetilde{G}$ 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 $\widetilde{G}$ 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 3.7. *Given a (potentially weighted) graph H which is promised to be a $(1\pm\epsilon)$ spectral sparsifier of some unweighted graph G, along with the number of edges in G, one can recover (in polynomial time) a $(1\pm 3\epsilon)$ 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:

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

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 $(1\pm\epsilon)$ 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 $(1\pm\epsilon)$ spectral sparsifier of L_H. That is, we wish to check whether

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

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

$$
\begin{aligned}
&(1-\epsilon)\cdot L_H \preceq L_{G(Y)} \preceq (1+\epsilon)\cdot L_H \iff \\
&\forall z \in \mathbb{R}^V : (1-\epsilon)\cdot z^T L_H z \leq z^T L_{G(Y)} z \leq (1+\epsilon)\cdot z^T L_H z,
\end{aligned}
$$

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

$$(1-\epsilon)\cdot I_{\mathrm{Im}(L_H)} \preceq L_H^{\dagger/2}\cdot L_{G(Y)}\cdot L_H^{\dagger/2} \preceq (1+\epsilon)\cdot I_{\mathrm{Im}(L_H)}, \quad (2)$$

where $I_{\mathrm{Im}(L_H)}$ is simply the projection operator on to $\mathrm{Im}(L_H)$[7]. Next, we note that Equation (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 $(1\pm\epsilon)$. This shows that $G(Y)$ is a $(1\pm\epsilon)$ 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 $(1\pm\epsilon)$.

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 $\lambda \notin (1\pm\epsilon)$, 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 λ:

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

by left multiplying with v^T this yields

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

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

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

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

$$
\begin{aligned}
v^\top\cdot L_H^{\dagger/2}\cdot L_{G(Y)}\cdot L_H^{\dagger/2}\cdot v &= \lambda\cdot v^\top v \notin (1\pm\epsilon)\cdot v^\top v \\
&= (1\pm\epsilon)\cdot v^\top\cdot L_H^{\dagger/2}\cdot L_H\cdot L_H^{\dagger/2}\cdot v,
\end{aligned}
$$

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 [31, 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 $(1\pm\epsilon)^2$ spectral sparsifier of G (it is a $(1\pm\epsilon)$ sparsifier of H, which is in turn a $(1\pm\epsilon)$ 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 $(1\pm\epsilon)$ 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.

[7] 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).

Lemma 3.8. *Let H be a fractional graph whose minimum cut is at least $\lambda \geq 200\log(n)/\epsilon^2$, 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 $\widetilde{H}$ which is a total weight preserving $(1\pm\epsilon)$ cut sparsifier of H.*

To prove this lemma, we require the following classic result due to [35].

Proposition 3.9 (Karger's Cut-counting Bound [35])**.** *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 3.8.

Proof of Lemma 3.8. The rounding scheme itself is elementary: we create a simple, unweighted graph $\widetilde{H}$, 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 $(1\pm\epsilon)$ sparsifier with probability $\geq 1-1/n^5$ (although it may not be total weight preserving).

Let $t := 200\log(n)/\epsilon^2$. Because the minimum cut in H is of size $\geq t$, by Karger's cut-counting bound (Proposition 3.9), 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

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

(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 $(1\pm\epsilon)$ 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 $(1\pm\epsilon)$ 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 $\widetilde{H}$ in our randomized rounding approach. For this, we need the following auxiliary claim.

Claim 3.10. *Let $p_1,\ldots p_m$ 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,*

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

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 [43], page 2, Darroch's rule for instance). Now, because the support of the distribution has size at most $m+1$ (i.e., $0,\ldots m$), it follows that the mode must occur with probability $\geq 1/(m+1)$, yielding our claim above. ∎ Claim 3.10

To conclude the proof of Lemma 3.8, we can apply Claim 3.10 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 $\widetilde{H}$. This is because our edge sampling procedure is exactly a Poisson binomial distribution fitting the form of Claim 3.10.

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 $\widetilde{H}$ 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 $(1\pm\epsilon)$ cut sparsifier of H. Thus, the graph $\widetilde{H}$ that we return is the first graph which preserves the total weight, and it will be a $(1\pm\epsilon)$ total weight preserving cut sparsifier of H with high probability. ∎ Lemma 3.8

3.5 Concluding the Proof of Theorem 3.2

Finally, in this section we synthesize our claims to conclude the proof of Theorem 3.2.

Proof of Theorem 3.2. First, we initialize the linear sketch of Lemma 3.6, using $\lambda = 200\log(n)/\epsilon^2$. The space complexity of our sketch then follows from Lemma 3.6.

Using our sketch, we can recover all edges of strength at most λ (we denote this set by T), as well as a $(1\pm\epsilon)$ spectral sparsifier of the graph $G-T$, and the total number of edges in $G-T$. Next, using Lemma 3.7, we find a fractional $(1\pm 3\epsilon)$ total weight preserving spectral sparsifier H of $G-T$ in polynomial time. Finally, we use Lemma 3.8 to, with high probability, round this into a simple $(1\pm\epsilon)$ total weight preserving cut sparsifier $\widetilde{H}$ of H in polynomial time. By composition of the sparsifier approximations, we also get that $\widetilde{H}$ is a $(1\pm 5\epsilon)$ simple total weight-preserving cut sparsifier of $G-T$. Finally, we add back the edges from T, and conclude that $\widetilde{H}\cup T$ is a $(1\pm 5\epsilon)$ 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 $\widetilde{H}$, and thus appears twice in $\widetilde{H}\cup T$. However, recall that when we remove edges in T, this partitions G into connected components $V_1,\ldots V_k$ each with minimum cut greater than λ. 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 $V_1,\ldots V_k$. This in turn implies that the fractional graph H generated by our convex program will not have any edges crossing between $V_1,\ldots V_k$ 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 $\widetilde{H}$.

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

Before moving on from this section, we note that by combining Lemma 3.7 and Lemma 3.8, we get the following general de-sparsification corollary that formalizes part 1 of Result 3.

Corollary 3.11 (Part 1 of Result 3). *Given a (potentially weighted) graph H which is promised to be a $(1 \pm \epsilon)$ spectral sparsifier of some unweighted simple graph G with minimum cut $\lambda \geq 200\log(n)/\epsilon^2$, along with the number of edges in G, we can recover in polynomial time a simple graph $\widetilde{G}$ which is a $(1 \pm O(\epsilon))$ 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 4.1. *There is a randomized linear sketch of size $\widetilde{O}(n/\epsilon^2)$ bits which, on any graph G, can be used to recover, with high probability, an unweighted, simple $(1 \pm \epsilon)$ total weight preserving spectral sparsifier of G in polynomial time.*

To prove Theorem 4.1, we follow the strategy of [41] in constructing spectral sparsifiers by sampling edges proportional to their effective resistance (Definition 2.5). Formally,

Proposition 4.2 ([41]). *Let G be an arbitrary graph on n vertices, let $\epsilon \in (0,1)$, and let C be a sufficiently large constant. Then, independently sampling each edge with probability*

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

(and assigning weight w_e/p_e if sampled) yields a $(1 \pm \epsilon)$ 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 4.3 (Part 2 of Result 3). *For any $\epsilon \in (0,1)$ and n-vertex unweighted graph G, there is a randomized polynomial-time algorithm that given any $(1 \pm \epsilon)$-spectral sparsifier H of G, recovers with high probability an n-vertex, simple unweighted graph $\widetilde{G}$ such that $\widetilde{G}$ is a $(1 \pm 5\epsilon)$-spectral sparsifier of G, provided the effective resistance of every pair (u,v) in G is $\leq \frac{\epsilon^2}{2C\log(n)}$, where C is the constant from Proposition 4.2.*

Proof. First, recall that by Lemma 3.7 we can recover a fractional, total weight preserving graph $\widetilde{H}$ which is a $(1 \pm \epsilon)$ spectral sparsifier of H and thus a $(1 \pm 3\epsilon)$ spectral sparsifier of G. All that remains is to perform a randomized rounding of $\widetilde{H}$ which yields a spectral sparsifier and preserves the total weight. We use the same rounding procedure as in Lemma 3.8, 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 4.2, we know that sampling every edge e with probability

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

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

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

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

So, by Proposition 4.2 we must only sample each edge $e \in \widetilde{H}$ with probability

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

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

4.2 Proof of Theorem 4.1

We now provide a formal proof of Theorem 4.1. The main challenge here is to handle the pairs of vertices whose effective resistances will be higher than the bounds in Theorem 4.3 which requires a non-black-box modification of our approach in establishing Theorem 3.2.

To prove Theorem 4.1, we need to use the following more detailed analysis from [34].

Proposition 4.4 ([34]). *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 $(1 \pm \epsilon)$ spectral sparsifier with probability $1 - 1/\text{poly}(n)$. Further, $\mathcal{S}$ requires only $\widetilde{O}(n\phi)$ bits of space to store.*

We are now ready to start the proof.

Linear sketch. The linear sketch in Theorem 4.1 is very simple: we simply store the sketch of Proposition 4.4 with parameter $\phi = \frac{C\log^3(n)}{\epsilon^2}$, for C a sufficiently large constant. It follows then by Proposition 4.4 that this allows us to recover a $(1 \pm \epsilon)$ spectral sparsifier of the graph G with high probability. Further, since the linear sketch of Proposition 4.4 implicitly performs effective-resistance sampling, it follows that every edge e with effective resistance $\geq \frac{1}{\phi} = \frac{\epsilon^2}{C\log^3(n)}$ is necessarily recovered, as each such edge is sampled with probability $\geq \frac{\phi}{\phi} = 1$.

Recovery from the sketch. Now, let $\widetilde{H}$ denote the weighted spectral sparsifier recovered by the above sketch, and let $\widetilde{H}_U$ denote the corresponding unweighted version of $\widetilde{H}$ (where every edge in $\widetilde{H}$ is given weight 1). Observe that it must be the case that $\widetilde{H}_U \subseteq G$,

as it is the result of sub-sampling G. Observe also that because $\widetilde{H}$ is a $(1 \pm \epsilon)$ spectral sparsifier of G, for every pair of vertices $(u,v) \in \binom{V}{2}$, it is the case that $R_{\text{eff},\widetilde{H}}(u,v) \in (1 \pm 2\epsilon) R_{\text{eff},G}(u,v)$ (see Definition 2.5).

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

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

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

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

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

Claim 4.5. *The stated convex program is feasible.*

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

Likewise, the separation oracle is efficiently implementable:

Claim 4.6. *There is a polynomial time separation oracle for the above convex program.*

Proof. This follows from all of the same reasons as in Lemma 3.7. Indeed certifying the constraints that $Y_e \in [0,1]$ and that $\sum_{e \in L_{\widetilde{H}_U}} 1 + \sum_{e \in \binom{V}{2}} Y_e = m$ are both trivial. Thus, it remains only to check whether

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

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

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

By linearity of the Laplacians, this can be re-written as $\forall z \in \mathbb{R}^n : \|z\|_2 = 1$,

$$z^T (L_{\widetilde{H}_U} + L_{\hat{G}}) z \in (1 \pm \epsilon) z^T L_{\widetilde{H}} z,$$

which is now exactly in the same form as the constraints of Lemma 3.7, and so can be checked by calculating the eigenvalues of $L_{\widetilde{H}}^{\dagger/2} (L_{\widetilde{H}_U} + L_{\hat{G}}) L_{\widetilde{H}}^{\dagger/2}$. ∎

Now, because of Claim 4.5 and Claim 4.6, we can use the ellipsoid method to find a feasible solution in polynomial time [31]. 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 $\hat{G} \cup \widetilde{H}_U$ is a fractional $(1 \pm \epsilon)$ spectral sparsifier of $\widetilde{H}$, and thus a fractional total weight preserving $(1 \pm 3\epsilon)$ spectral sparsifier of G.

All that remains is to show that efficiently rounding this solution is possible. For this, by Proposition 4.2, we know that sampling each edge e with probability $p_e \geq \frac{C \log(n)}{\epsilon^2} \cdot w_e \cdot R_{\text{eff}}(e)$, and giving weight $\frac{w_e}{p_e}$ yields a $(1 \pm \epsilon)$ spectral sparsifier with high probability. Now, because $\hat{G} \cup \widetilde{H}_U$ is a $(1 \pm \epsilon)$ spectral sparsifier of $\widetilde{H}$, it follows that for every pair of vertices (u,v),

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

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

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

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

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

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

By starting with an error parameter of $\epsilon/5$, 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 3.10 that in some round, the total weight is exactly preserved.

This concludes the proof of Theorem 4.1. ∎

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.

S. A. is supported in part by a Sloan Research Fellowship, an NSERC Discovery Grant (RGPIN-2024-04290) and a Faculty of Math Research Chair grant. S. K. is supported in part by NSF awards CCF-2008305 and CCF-2402284. A. P. is supported in part by the Simons Investigator Awards of Madhu Sudan and Salil Vadhan and NSF Award CCF-2152413.

References

[1] Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. 2015. Correlation Clustering in Data Streams. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015 (JMLR Workshop and Conference Proceedings, Vol. 37)*, Francis R. Bach and David M. Blei (Eds.). JMLR.org, 2237–2246.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, Yuval Rabani (Ed.). SIAM, 459–467. doi:10.1137/1.9781611973099.40

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini (Eds.). ACM, 5–14. doi:10.1145/2213556.2213560

[4] Nir Ailon, Moses Charikar, and Alantha Newman. 2008. Aggregating inconsistent information: Ranking and clustering. *J. ACM* 55, 5 (2008), 23:1–23:27. doi:10.1145/1411509.1411513

[5] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. 2016. Maximum Matchings in Dynamic Graph Streams and the Simultaneous Communication Model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, Robert Krauthgamer (Ed.). SIAM, 1345–1364. doi:10.1137/1.9781611974331.CH93

[6] Sepehr Assadi and Vihan Shah. 2023. Tight Bounds for Vertex Connectivity in Dynamic Streams. In *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, Telikepalli Kavitha and Kurt Mehlhorn (Eds.). SIAM, 213–227. doi:10.1137/1.9781611977585.CH20

[7] Sepehr Assadi and Chen Wang. 2022. Sublinear Time and Space Algorithms for Correlation Clustering via Sparse-Dense Decompositions. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA (LIPIcs, Vol. 215)*, Mark Braverman (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:20. doi:10.4230/LIPICS.ITCS.2022.10

[8] Pranjal Awasthi, Ainesh Bakshi, Maria-Florina Balcan, Colin White, and David P. Woodruff. 2019. Robust Communication-Optimal Distributed Clustering Algorithms. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece (LIPIcs, Vol. 132)*, Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 18:1–18:16. doi:10.4230/LIPICS.ICALP.2019.18

[9] Paul Beame, Paraschos Koutris, and Dan Suciu. 2017. Communication Steps for Parallel Query Processing. *J. ACM* 64, 6 (2017), 40:1–40:58. doi:10.1145/3125644

[10] Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. 2022. 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*. IEEE, 720–731. doi:10.1109/FOCS54457.2022.00074

[11] Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. 2023. Single-Pass Streaming Algorithms for Correlation Clustering. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, Nikhil Bansal and Viswanath Nagarajan (Eds.). SIAM, 819–849. doi:10.1137/1.9781611977554.CH33

[12] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. 2022. A Framework for Adversarially Robust Streaming Algorithms. *J. ACM* 69, 2 (2022), 17:1–17:33. doi:10.1145/3498334

[13] András A. Benczúr and David R. Karger. 1996. Approximating *s-t* Minimum Cuts in $\tilde{O}(n^2)$ Time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, Gary L. Miller (Ed.). ACM, 47–55. doi:10.1145/237814.237827

[14] Aaron Bernstein, Greg Bodwin, and Nicole Wein. 2024. Are There Graphs Whose Shortest Path Structure Requires Large Edge Weights?. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA (LIPIcs, Vol. 287)*, Venkatesan Guruswami (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 12:1–12:22. doi:10.4230/LIPICS.ITCS.2024.12

[15] Aaron Bernstein, Jiale Chen, Aditi Dudeja, Zachary Langley, Aaron Sidford, and Ta-Wei Tu. 2025. Matching Composition and Efficient Weight Reduction in Dynamic Matching. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, Yossi Azar and Debmalya Panigrahi (Eds.). SIAM, 2991–3028. doi:10.1137/1.9781611978322.97

[16] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. 2021. A framework for dynamic matching in weighted graphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, Samir Khuller and Virginia Vassilevska Williams (Eds.). ACM, 668–681. doi:10.1145/3406325.3451113

[17] Mélanie Cambus, Fabian Kuhn, Etna Lindy, Shreyas Pai, and Jara Uitto. 2024. A $(3+\epsilon)$-Approximate Correlation Clustering Algorithm in Dynamic Streams. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, David P. Woodruff (Ed.). SIAM, 2861–2880. doi:10.1137/1.9781611977912.101

[18] Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, Alantha Newman, and Lukas Vogl. 2024. Understanding the Cluster Linear Program for Correlation Clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, Bojan Mohar, Igor Shinkar, and Ryan O'Donnell (Eds.). ACM, 1605–1616. doi:10.1145/3618260.3649749

[19] Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. 2022. Adversarially Robust Coloring for Graph Streams. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA (LIPIcs, Vol. 215)*, Mark Braverman (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 37:1–37:23. doi:10.4230/LIPICS.ITCS.2022.37

[20] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. 2003. Clustering with Qualitative Information. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*. IEEE Computer Society, 524–533. doi:10.1109/SFCS.2003.1238225

[21] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. 2015. Near Optimal LP Rounding Algorithm for CorrelationClustering on Complete and Complete k-partite Graphs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM, 219–228. doi:10.1145/2746539.2746604

[22] Jiecao Chen, He Sun, David P. Woodruff, and Qin Zhang. 2016. Communication-Optimal Distributed Clustering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 3720–3728.

[23] Yu Chen, Sanjeev Khanna, and Huan Li. 2022. On Weighted Graph Sparsification by Linear Sketching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 474–485. doi:10.1109/FOCS54457.2022.00052

[24] Flavio Chierichetti, Nilesh N. Dalvi, and Ravi Kumar. 2014. Correlation clustering in MapReduce. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (Eds.). ACM, 641–650. doi:10.1145/2623330.2623743

[25] Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. 2021. Correlation Clustering in Constant Many Parallel Rounds. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 2069–2078.

[26] Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. 2023. 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*. IEEE, 1082–1104. doi:10.1109/FOCS57990.2023.00065

[27] Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. 2022. 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*. IEEE, 651–661. doi:10.1109/FOCS54457.2022.00068

[28] Vincent Cohen-Addad, David Rasmussen Lolck, Marcin Pilipczuk, Mikkel Thorup, Shuyi Yan, and Hanwen Zhang. 2024. Combinatorial Correlation Clustering. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, Bojan Mohar, Igor Shinkar, and Ryan O'Donnell (Eds.). ACM, 1617–1628. doi:10.1145/3618260.3649712

[29] Mina Dalirrooyfard, Konstantin Makarychev, and Slobodan Mitrovic. 2024. 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.

[30] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. 2006. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.* 361, 2-3 (2006), 172–187. doi:10.1016/J.TCS.2006.05.008

[31] Martin Grötschel, László Lovász, and Alexander Schrijver. 1988. *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics, Vol. 2. Springer. doi:10.1007/978-3-642-97881-4

[32] Moritz Hardt, Nikhil Srivastava, and Madhur Tulsiani. 2012. Graph densification. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, Shafi Goldwasser (Ed.). ACM, 380–392. doi:10.1145/2090236.2090266

[33] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. 2014. Single Pass Spectral Sparsification in Dynamic Streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 561–570. doi:10.1109/FOCS.2014.66

[34] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. 2014. Single Pass Spectral Sparsification in Dynamic Streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 561–570. doi:10.1109/FOCS.2014.66

[35] David R. Karger. 1993. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, Vijaya Ramachandran (Ed.). ACM/SIAM, 21–30. http://dl.acm.org/citation.cfm?id=313559.313605

[36] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, Moses Charikar (Ed.). SIAM, 938–948. doi:10.1137/1.9781611973075.76

[37] Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. 2024. 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*. IEEE, 1669–1706. doi:10.1109/FOCS61266.2024.00105

[38] Konstantin Makarychev and Sayak Chakrabarty. 2023. Single-Pass Pivot Algorithm for Correlation Clustering. Keep it simple!. In *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*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.).

[39] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. 2015. Densest Subgraph in Dynamic Graph Streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9235)*, Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella (Eds.). Springer, 472–482. doi:10.1007/978-3-662-48054-0_39

[40] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. 2018. Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation). *J. ACM* 65, 6 (2018), 41:1–41:24. doi:10.1145/3232536

[41] Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926. doi:10.1137/080734029

[42] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral Sparsification of Graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025. doi:10.1137/08074489X

[43] Wenpin Tang and Fengmin Tang. 2023. The Poisson binomial distribution—Old & new. *Statist. Sci.* 38, 1 (2023), 108–119.

[44] Chun Jiang Zhu, Tan Zhu, Kam-yiu Lam, Song Han, and Jinbo Bi. 2019. 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*. AAAI Press, 5957–5964.