Faster All-Pairs Minimum Cut: Bypassing Exact Max-Flow

Yotam Kenneth-Mordoch Robert Krauthgamer*
Weizmann Institute of Science
{yotam.kenneth,robert.krauthgamer}@weizmann.ac.il

November 14, 2025

Abstract

All-Pairs Minimum Cut (APMC) is a fundamental graph problem that asks to find a minimum s,t-cut for every pair of vertices s,t. A recent line of work on fast algorithms for APMC has culminated with a reduction of APMC to polylog(n)-many max-flow computations. But unfortunately, no fast algorithms are currently known for exact max-flow in several standard models of computation, such as the cut-query model and the fully-dynamic model.

Our main technical contribution is a sparsifier that preserves all minimum s,t-cuts in an unweighted graph, and can be constructed using only approximate max-flow computations. We then use this sparsifier to devise new algorithms for APMC in unweighted graphs in several computational models: (i) a randomized algorithm that makes $\tilde{O}(n^{3/2})$ cut queries to the input graph; (ii) a deterministic fully-dynamic algorithm with $n^{3/2+o(1)}$ worst-case update time; and (iii) a randomized two-pass streaming algorithm with space requirement $\tilde{O}(n^{3/2})$. These results improve over the known bounds, even for (single pair) minimum s,t-cut in the respective models.

*The Harry Weinrebe Professorial Chair of Computer Science. Work partially supported by the Israel Science Foundation grant #1336/23.

1 Introduction

A powerful technique for speeding up the computation of graph algorithms is to employ sparsifiers, which are graphs that capture some property of the original graph while being significantly smaller. For example, Nagamochi and Ibaraki [NI92] showed that given a graph on n vertices and a parameter $k > 1$, one can construct a subgraph with $O(nk)$ edges that preserves all its cuts with at most k edges. This sparsifier is a key ingredient in several fast algorithms, from finding a global minimum cut [NI92], through constructing $(1 \pm \epsilon)$-cut sparsifiers [FHHP19], to solving the all-pairs minimum cut problem [LPS21, AKT21b, AKT21a, AKT22]. Another example are non-trivial minimum cut (NMC) sparsifiers, introduced by Kawarabayashi and Thorup [KT19], which preserve all non-trivial minimum cuts of a graph while reducing the number of vertices in the graph to $\tilde{O}(n/\delta)$, where δ is the minimum degree of the graph and $\tilde{O}(\cdot)$ hides polylogarithmic factors. This sparsifier has been the basis of many recent algorithms for finding a global minimum cut in several models of computation [KT19, HRW20, GNT20, AEG+22, GHN+23, KK25b, HKMR25, KK25c]. Additional examples include friendly-cut sparsifiers [AKT22], approximate cut sparsifiers [BK96], and many more variants that preserve other properties, like distances (called spanners), flows, spectrum, and resistances; or preserve such properties only for terminal vertices (called vertex sparsifiers).

We focus on the all-pairs minimum cut problem (APMC), where the input is an edge-weighted graph $G = (V, E, w)$, and the goal is to find for every pair $s, t \in V$ the minimum s,t-cut value, which is formally defined as

$$\lambda_G(s,t) \coloneqq \min\left\{\mathrm{cut}_G(S) : \ \{s\} \subseteq S \subseteq V \setminus \{t\}\right\},$$

where $\mathrm{cut}_G(S)$ denotes the total weight of edges crossing the cut $(S, V \setminus S)$. Throughout, we restrict attention to the case where the input G is unweighted, i.e., all edges have unit weight. Our method is based on efficiently constructing a sparsifier that preserves all these minimum cuts, and then solving it on the sparsifier. A key difference from prior work is that our approach adds new vertices to the graph, as formalized in the next definition.

Definition 1.1. *An all-pairs minimum cut sparsifier (APMC sparsifier) of a graph $G = (V, E)$ is a weighted graph $H = (U, E_H, w)$ where $U \supseteq V$, such that for every $s, t \in V$, there exists a minimum s,t-cut $S_H^{s,t}$ in H such that $S_H^{s,t} \cap V$ is a minimum s,t-cut in G of the same value, i.e.,*

$$\lambda_H(s,t) = \mathrm{cut}_H(S_H^{s,t}) = \mathrm{cut}_G(S_H^{s,t} \cap V) = \lambda_G(s,t).$$

A well-known example of an APMC sparsifier is the celebrated Gomory-Hu tree [GH61], which is a tree built on the same vertices ($U = V$ in our notation) using $n - 1$ calls to a minimum s,t-cut procedure. Besides the obvious sparsity of a tree ($n - 1$ edges), their algorithm achieves a quadratic improvement over the $\binom{n}{2}$ calls needed by a straightforward computation for all pairs. A recent line of work shows that a Gomory-Hu tree can be computed using only $\mathrm{polylog}(n)$ calls to a minimum s,t-cut procedure [AKT20, LPS21, AKT21b, AKT21a, AKT22, AKL+22, ALPS23, AKL+25, GKYY25] and minimal additional computation, hence the complexity of solving all-pairs minimum cuts is essentially equivalent to that of solving a single-pair minimum cut (known to be equivalent to solving maximum s,t-flow, and thus called a max-flow computation).

Unfortunately, computing exact minimum s,t-cut is currently still expensive in several models of computation including the cut-query, streaming and fully-dynamic models. Therefore, constructing a Gomory-Hu tree in these settings remains expensive as well. Our main contribution is a new construction of an APMC sparsifier that uses only *approximate* minimum s,t-cut, yielding efficient algorithms in models where finding approximate cuts is cheap, such as the cut-query and streaming

models. Furthermore, our APMC sparsifier is robust to changes in the input G and hence can be maintained efficiently in the fully-dynamic model. In fact, our sparsifier is so succinct that it yields algorithms for all-pairs minimum cut that are more efficient than current algorithms for single-pair minimum cut in the aforementioned settings.

1.1 Results

Cut-Query, Two-Party Communication and Submodular Minimization Our first result is an improved algorithm for all-pairs minimum cut in the cut-query model. It not only improves over the $\tilde{O}(n^{7/4})$ query complexity known for this problem [KK25a], but also the $\tilde{O}(n^{8/5})$ query complexity known for (single pair) minimum s,t-cut [JNS25].

Theorem 1.2. *There exists a randomized algorithm, that given cut-query access to an unweighted graph G on n vertices, solves the all-pairs minimum cut problem using $\tilde{O}(n^{3/2})$ cut queries and succeeds with probability $1 - 1/\operatorname{poly}(n)$.*

By an easy reduction, this result also implies the first (non-trivial) algorithm for all-pairs minimum cut in the two-party communication model. In this model, the edges of the input graph are partitioned between two players that wish to jointly compute some function of the graph. The goal is to design a communication protocol that allows the computation while minimizing the total number of bits exchanged between the two players. We note that our protocol matches the known bound for (single-pair) minimum s,t-cut [JNS25, GJ25].

Corollary 1.3. *There exists a randomized two-player communication protocol that computes all-pairs minimum cut problem of an unweighted graph with n vertices using $\tilde{O}(n^{3/2})$ bits of communication and succeeds with probability $1 - 1/\operatorname{poly}(n)$.*

Furthermore, recent work implies that one can compute a Gomory-Hu tree of a symmetric submodular function using $\operatorname{polylog}(n)$ calls to a submodular minimization procedure on functions with total ground-set size $\tilde{O}(n)$ [GKYY25].[1] It then follows easily that the query complexity APMC is at most $\operatorname{polylog}(n)$ times the query complexity of minimizing a submodular function (abbreviated SFM) on n elements.[2] For completeness, we state this formally in the next theorem and prove it in Appendix A. It basically rules out the natural approach of proving a query lower bound for SFM that is based on the hardness of APMC in the cut-query model and significantly exceeds $\Omega(n^{3/2})$, as that would contradict our cut-query algorithm above.

Theorem 1.4 (Query Complexity Version of [GKYY25])**.** *Let $Q(n)$ be the query complexity of minimizing a symmetric submodular function over a ground set of size n. Then, there exists an algorithm for constructing a Gomory-Hu tree of a symmetric submodular $f : 2^{[n]} \to \mathbb{R}_+$ using $Q(n) \cdot \operatorname{polylog}(n)$ queries.*

Fully-Dynamic Algorithms In the fully-dynamic setting, the input is a *dynamic graph* presented as a sequence of edge insertions and deletions over a fixed set of n vertices, and the goal is to maintain some function of the graph after each edge update. We present the first (non-trivial) fully-dynamic algorithm for APMC (i.e., for maintaining a Gomory-Hu tree). Furthermore, our algorithm is deterministic, and thus robust to adversarial updates. Even for (single-pair) minimum s,t-cut, all previous algorithms either return an approximate solution [ADK+16, GRST21, BvdBG+22] or are limited to instances where $\lambda_G(s,t) \leq \log^{o(1)} n$ [GI91, Fre97, HK99, JS21].

[1] The original work focuses on reducing the construction of Gomory-Hu trees for graphs to $\operatorname{polylog} n$ max-flow instances, we verify that their approach extends to all symmetric submodular functions.

[2] The algorithms solves many small SFM instances on contraction of the ground set, whose total size is $\tilde{O}(n)$.

Theorem 1.5. *There exists a deterministic fully-dynamic algorithm that maintains a Gomory-Hu tree for a dynamic unweighted graph on n vertices, using $n^{3/2+o(1)}$ worst-case update time.*

This theorem immediately implies an algorithm with $n^{3/2+o(1)}$ worst-case update time for maintaining edge connectivity in a dynamic graph. This is the first deterministic algorithm for the problem with $o(n^2)$ worst-case update time. Furthermore, when $m \geq n^{37/22} \approx n^{1.7}$, it matches, up to subpolynomial factors, the state-of-the-art deterministic amortized-time complexity of $\tilde{O}(\min(m^{1-1/12}, m^{11/13}n^{1/13}, n^{3/2}))$ [dVC25].

Corollary 1.6. *There exists a deterministic fully-dynamic algorithm that maintains the value of a global minimum cut of a dynamic unweighted graph on n vertices, using $n^{3/2+o(1)}$ worst-case update time.*

Streaming Algorithms We provide the first (non-trivial) algorithm for all-pairs minimum cut in dynamic streams, where the graph is presented as a sequence of edge insertions and deletions. Our algorithm uses $\tilde{O}(n^{3/2})$ space and makes two pass. Previously, the only known algorithm was for finding a (single-pair) minimum s,t-cut, and it uses $\tilde{O}(n^{5/3})$ space in two passes [RSW18]. No bounds are known even with other restrictions, e.g., insertion-only (no deletions) or another number of passes.

Theorem 1.7. *Given an unweighted graph G on n vertices presented as a dynamic stream, one can solve the all-pairs minimum cut problem using $\tilde{O}(n^{3/2})$ bits of storage in two passes. The algorithm is randomized and succeeds with probability $1 - 1/\operatorname{poly}(n)$.*

A summary of our results is presented in Table 1.

Computational Model	Complexity Measure	Our All-Pairs Result	Known Results
cut query	queries	$\tilde{O}(n^{3/2})$ Thm. 1.2	$\tilde{O}(n^{8/5})$, single pair [JNS25] $\tilde{O}(n^{7/4})$, all pairs [KK25a]
fully dynamic	worst-case update time	$n^{3/2+o(1)}$ Thm. 1.5	$\tilde{O}(m)$, single pair
dynamic streaming	storage, two passes	$\tilde{O}(n^{3/2})$ Thm. 1.7	$\tilde{O}(n^{5/3})$, single pair [RSW18] $\tilde{O}(m)$, all pairs
two-party communication	bits	$\tilde{O}(n^{3/2})$ Cor. 1.3	$\tilde{O}(n^{3/2})$, single pair [GJ25] $\tilde{O}(m)$, all pairs

Table 1: Comparison of our algorithms for all-pairs minimum-cut with existing algorithms for the same problem and for single pair minimum s,t-cut, across different computational models.

1.2 Related Work

All-Pairs Minimum Cut The all-pairs minimum cut problem has received significant attention in recent years, culminating in algorithms that make $O(\operatorname{polylog} n)$ calls to a minimum s,t-cut procedure and require in addition $\tilde{O}(m)$ time [HKP07, BHKP07, AKT20, LPS21, AKT21b, AKT21a, AKT22, AKL+22, ALPS23, AKL+25, GKYY25]. Over roughly the same time span, there has been significant progress on faster algorithms (in running time) for the minimum s,t-cut problem itself, and currently the best algorithm runs in $m^{1+o(1)}$ time [CKL+25].

Cut-Query Algorithms The study of cut-query algorithms was initiated in [RSW18], motivated by the relation between cut queries and submodular minimization. For the global minimum cut problem, known algorithms in unweighted graphs use $O(n)$ randomized cut queries [RSW18, AEG+22], matching the lower bound [GPRW20, LLSZ21], and $\tilde{O}(n)$ randomized cut queries suffice in weighted graphs [MN20]; in addition, known deterministic algorithms use $\tilde{O}(n^{5/3})$ queries in unweighted graphs [ASW25]. For minimum s,t-cut in unweighted graphs, known algorithms use $\tilde{O}(n^{8/5})$ cut queries [RSW18, ASW25, JNS25], leaving a gap as the only lower bound is $\Omega(n)$ queries. Recently, it was shown that APMC in unweighted graphs can be solved using $\tilde{O}(n^{7/4})$ cut queries [KK25a]. Other related work include an algorithm for approximating the maximum cut in weighted graphs using $\tilde{O}(n)$ cut queries [PRW24], and studying parallel efficiency by measuring the number of query rounds for both global and s,t minimum cuts in [KK25b].

Cuts in Dynamic Streams In the streaming model, exact computation of a global (or s,t) minimum cut requires $\Omega(n^2)$ space in one pass [Zel11].[3] Consequently, research has primarily focused on approximate solutions; one can obtain $(1\pm\epsilon)$-approximation using $\tilde{O}(\epsilon^{-2}n)$ space by employing cut sparsifiers, see e.g. [AGM12]. For global minimum cut in unweighted graphs this was later improved to $\tilde{O}(\epsilon^{-1}n)$ [DGL+25]. The study of exact solutions using more passes was motivated by an observation in [ACK19] that the approach of [RSW18] for global and s,t minimum cut in unweighted graphs can be implemented in two passes using $\tilde{O}(n)$ space and $\tilde{O}(n^{5/3})$ space, respectively; see also [AD21]. For weighted graphs, an $O(\log n)$-pass algorithm for global minimum cut using $\tilde{O}(n)$ space was presented in [MN20], and a smooth tradeoff of $\tilde{O}(rn^{1+1/r})$ space in $O(r)$ passes was later shown in [KK25b]. Finally, for finding a minimum s,t-cut in weighted graphs, there exists a lower bound of $\Omega(n^2/p^5)$ space for p-pass streaming algorithms [ACK19].

Cuts in Fully-Dynamic Graphs The study of global minimum cut in the fully-dynamic setting has a rich history: it was originally studied for small values of edge connectivity, namely $\lambda_G := \min_{s,t}\lambda_G(s,t) \leq \log^{o(1)} n$, see e.g. [EGIN97, HdLT01, KKM13, CGL+20, JST24]. For general λ_G, there exists a deterministic algorithm that maintains λ_G in worst-case update time $\tilde{O}(\min(\lambda_{\max}^{5.5}\sqrt{n}))$, where $\lambda_{\max}$ is the maximum edge connectivity of G throughout the graph updates [Tho07, GHN+23, dVC25]. Allowing randomization, there exist algorithms that maintain λ_G in worst-case update time $\tilde{O}(\min\{n,(n/\lambda_G^2)^2\})$ [GHN+23, HKMR25, KK25c]. Finally, one can achieve a $(1+o(1))$-approximation with worst-case update time $n^{o(1)}$ [TK00, Tho07, EHL25]. The study of minimum s,t-cut in the fully dynamic setting has received less attention, but features similar tradeoffs as global minimum cut: Existing algorithms either return an approximate solution [ADK+16, GRST21, BvdBG+22] or are limited to instances where $\lambda_G(s,t) \leq \log^{o(1)} n$ [GI91, Fre97, HK99, JS21].

1.3 Future Work

Lower Bounds for Submodular Function Minimization One of the main motivations for studying APMC in the cut-query model is its connection to submodular function minimization (SFM). Graph-cut functions are natural examples of submodular functions and have been extensively studied in the context of SFM. In particular, the best lower bounds known for SFM are based on graph-cut functions [GPRW20, LLSZ21] (there are slightly stronger lower bounds for deterministic algorithms that are based on different functions [CGJS23]). The following lemma shows that maximum-flow algorithms in the cut-query model have an inherent lower bound of $\Omega(n^{3/2})$ queries.

[3] These lower bounds do not apply in random order streams where better results are possible, e.g. [DGL+25].

It is based on a graph constructed in [KL98], where the maximum s,t-flow contains $\Omega(n^{3/2})$ edges, and has been observed previously [ASW25, JNS25] without a formal proof, hence we prove it for completeness in Appendix B.

Lemma 1.8. *Every randomized cut-query algorithm that recovers all edges of a maximum s,t-flow in an unweighted graph G and succeeds with high constant probability requires $\Omega(n^{3/2}/\log n)$ cut queries.*

Previous algorithms for minimum s,t-cut in the cut-query model [RSW18, ASW25, JNS25] are affected by this lower bound, e.g., they compute a maximum s,t-flow. In contrast, our algorithm for all-pairs minimum cut circumvents this lower bound by avoiding explicit recovery of any maximum s,t-flow. However, it still does not manage to break the $O(n^{3/2})$ barrier, which remains open. We conjecture that this is not possible and that every algorithm for minimum s,t-cut in the cut-query model requires $\tilde{\Omega}(n^{3/2})$ queries. Finally, Theorem 1.4 (which constructs a Gomory-Hu tree for a symmetric submodular function using polylog(n) calls to an SFM procedure on ground sets of size n) yields an exciting possible avenue for proving lower bounds for SFM.

Extension to Weighted Graphs The structural results that form the basis of our APMC sparsifier do *not* seem to extend to weighted graphs. In the case of streaming algorithms, there is an $\Omega(n^2/p^5)$ space lower bound for p-pass algorithms [ACK19], which rules out most practical algorithms. However, in the cut-query and fully-dynamic models, no such lower bounds are known, and in fact we are still lacking non-trivial algorithms for (single pair) minimum s,t-cut in weighted graphs; thus, either algorithms or lower bounds would be very interesting.

2 Technical Overview

Our work contains two main structural insights. The first one is that the minimum s,t-cuts of a graph G (for all pairs s,t) are encoded by the so-called friendly cuts of G and the vertex degrees in G. The concept of a friendly cut (its definition appears in Section 3 and is not needed here), emerges naturally in social learning, statistical physics and set theory under various names [Mor00, GK00, BTV06, BL16, GMT20, FKN+22]. Recently, this concept was used to show that every unweighted graph G can be contracted into a graph H_{fr} that preserves the value of every friendly cut in G and has a small number of edges [AKT22]. Such a graph H_{fr} is called a *friendly cut sparsifier* (see Definition 3.1). Throughout the paper we assume that the graph contains no vertices of degree 1 to simplify the proofs. This limitation can be easily removed in all our algorithmic results by handling such vertices separately.[4] Furthermore, we denote by $E(A,B)$ the set of edges with one endpoint in A and the other in B for $A,B \subseteq V$, when is a single vertex, $B=\{v\}$, we write $E(A,v)$ instead of $E(A,\{v\})$. Our first structural result is the following.

Theorem 2.1. *Let $G=(V,E)$ be an unweighted graph and let $\mathcal{A}_{s,t} \subseteq 2^V$ denote the set of minimum s,t-cuts in G for $s,t \in V$. Given a friendly cut sparsifier of G and the vertex degrees of G, one can recover for each pair $s,t \in V$ at least one cut $S \in \mathcal{A}_{s,t}$ along with its value.*

Remark 2.2. *Theorem 2.1 only guarantees recovering at least one minimum s,t-cut for each pair s,t. However, if one guarantees that no vertex of degree at most 2 in G is contracted during the construction of the friendly cut sparsifier, then one can recover the entire set $\mathcal{A}_{s,t}$ for each pair s,t. We omit the details for brevity.*

[4] We can also incorporate vertices of degree 1 into our cut encoding and APMC sparsifier (and thus handle them directly and not separately in our algorithmic results), however this requires additional technical details which we omit for clarity.

Friendly cut sparsifiers were previously used to design fast algorithms for APMC in [AKT22, KK25a]. Their approach divides the optimal cuts into friendly and unfriendly; friendly cuts are found efficiently by applying known algorithms on the sparsifier, whereas unfriendly cuts admit faster algorithms that find them directly in the original graph G. In contrast, we show that the friendly cut sparsifier already encodes all the minimum s,t-cuts without further access to G except for the vertex degrees. Unfortunately, recovering the minimum s,t-cuts from this encoding naively requires iterating over all the cuts of H_{fr}, which might take exponential time. While this may be acceptable in the streaming and cut-query settings, where one is mostly concerned with storage and query complexities, this approach is too inefficient for the fully-dynamic setting, and here comes our second structural contribution, which is to construct a small APMC sparsifier based on the encoding.

Our APMC sparsifier construction is based on a new *star transform* operation, that rearranges the vertices forming a contracted node of the friendly cut sparsifier (henceforth called a *super-vertex*) into a star graph; formally, it works as follows. Given a super-vertex $u \subseteq V$ in H_{fr}, split u back into its constituent vertices $\{v\}_{v\in u}$, and reconnect every edge incident to u back to its original endpoint $v \in u$. In addition, add a new *proxy vertex* u' and connect it to every vertex $v \in u$ with edge weight $w(v,u') \coloneqq |E(v, u \setminus v)|$, which is the degree of v in the induced subgraph $G[u]$. See Figure 1 for illustration. We show that performing this star transform to *all super-vertices* yields a graph H_{ap} that is an APMC sparsifier. We stress that all the information needed for the star transform is encoded in the friendly cut sparsifier and the vertex degrees of G, simply because $|E(v, u \setminus v)| = \deg(v) - |E(v, V \setminus u)|$ and the subtracted quantity counts edges that appear in H_{fr}, and can be recovered since we store the edges in H_{fr} along with their original endpoints in G.

The star transform bears superficial resemblance to the star-mesh transform from electrical network theory [Bed61, vLO73], which replaces a star with a weighted clique on the non-center vertices (of the star), with edges weights that preserves the resistance between each pair of non-center vertices. The Wye-Delta transform is a special case with 3 non-center vertices, that has been used in cut sparsification of planar graphs [KR14, GHP20, KK21]. However, there are several crucial differences. First, our star transform increases the number of vertices in the graph (recall it adds a proxy vertex), and thus it is more similar to the *inverse* of the mesh-star transform, which does not exist in general. Second, the star-mesh transform preserves cuts only in planar graphs, while our star transform is applicable to general graphs.

Theorem 2.3. *Let G be an unweighted graph on n vertices. Given access to a friendly cut sparsifier $H_{\text{fr}} = (V_{H_{\text{fr}}}, E_{H_{\text{fr}}})$ of G and the vertex degrees of G, one can construct an APMC sparsifier $H_{\text{ap}} = (U, E_{\text{ap}})$ of G with $|E_{\text{ap}}| \le |E_{H_{\text{fr}}}| + n$ in time $O(|E_{H_{\text{fr}}}| + n)$.*

One main advantage of our APMC sparsifier, in comparison to the Gomory-Hu tree, is that its construction does not require computing any *exact* minimum s,t-cut in G. Instead, it relies only on friendly cut sparsifiers, which themselves are based on expander decomposition, which in turn can be computed using *approximate* minimum s,t-cut. This suffices for the cut-query and streaming models, where we can leverage existing algorithms for expander decomposition [FKM23, CKMM24, KK25a]. For the fully-dynamic model, we provide a new guarantee and show that existing constructions of friendly cut sparsifiers are robust to edge updates, i.e., given a sequence of $\tilde{O}(\sqrt{n})$ edge updates to G one can apply the same updates to H_{fr} and obtain a friendly cut sparsifier of the updated graph. This allows us to build a new friendly cut sparsifier from scratch every $\tilde{O}(\sqrt{n})$ updates, and thus maintain an APMC sparsifier using $\tilde{O}(n^{3/2})$ worst-case update time.

The rest of the paper is organized as follows. In Section 3 we prove Theorems 2.1 and 2.3. We then use these structural results in Section 4 to design algorithms for the cut-query, fully-dynamic,

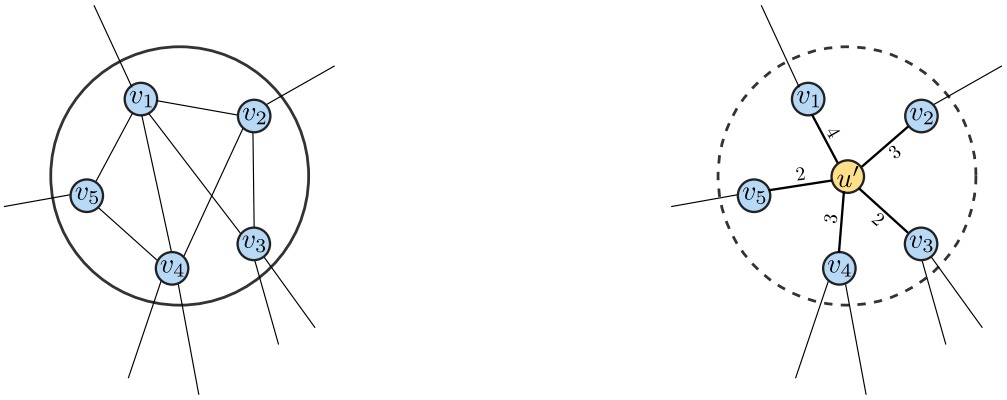

Figure 1: An illustration of the star-transform operation. The super-vertex $u = \{v_1, \ldots, v_5\}$ is split into its constituent vertices, adding a proxy vertex u' that is connected to every vertex $v_i \in u$ with edge weight equal to its degree in the induced graph $G[u]$. For example, v_5 has two neighbors in $G[u]$, hence it is connected to u' with an edge of weight 2.

and streaming models, proving Theorems 1.2, 1.5 and 1.7. Finally, we provide in Sections 5 and 6 more details for the construction of friendly cut sparsifiers in the streaming and fully-dynamic models.

3 Encoding All Minimum s,t-Cuts

In these section we prove Theorems 2.1 and 2.3, showing how to encode all the minimum s,t-cuts of a graph using a friendly cut sparsifier and the degree of every vertex, and how to construct an all-pairs minimum-cut (APMC) sparsifier using this encoding.

We begin by formally defining the notion of friendly cuts and friendly cut sparsifiers for a graph $G = (V, E)$. Given a cut $S \subseteq V$, let $\mathrm{cr}_S(v)$ be the number of edges incident to v that cross the cut $E(S, V \setminus S)$, and let us omit the subscript when clear from context. Define the *friendliness ratio* of a vertex $v \in V$ (with respect to S) as $1 - \mathrm{cr}(v)/\deg(v)$, where $\deg(v)$ is the degree of v in G. The cut S is called *α-friendly* if every vertex $v \in V$ has friendliness ratio $1 - \mathrm{cr}_S(v)/\deg(v) \geq \alpha$, and otherwise it is called *α-unfriendly*.

Throughout, a *contraction* of $G = (V, E)$ is a graph H obtained from G by contracting some subsets of vertices. A *super-vertex* is a vertex in H formed by contracting multiple vertices in G. Observe that contracting two vertices $u, v \in V$ is equivalent to adding an edge of infinite capacity between them. This view is useful because now H has the same vertex set as G, and furthermore, cuts in H are at least as large as those in G (when comparing the same $S \subset V$). All friendly cut sparsifiers throughout the paper are contraction-based.

Definition 3.1. *An* (α, w)-friendly cut sparsifier *of a graph G is a contraction H of G such that for every α-friendly cut $S \subseteq V$ of G with at most w edges we have* $\mathrm{cut}_H(S) = \mathrm{cut}_G(S)$.

Remark 3.2. *The statements of Theorems 2.1 and 2.3 do not specify the parameters of the friendly cut sparsifier, both theorems use a $(1/6, 2n)$-friendly cut sparsifier.*

The main idea in proving Theorem 2.1 (encoding all the minimum s,t-cuts using a friendly cut sparsifier) is provided in Lemma 3.4 below, which in turn relies on the following easy claim. We

say that a minimum s,t-cut $S \subseteq V$ with $s \in S$ is *minimal* if for all $S' \subset S$ with $s \in S$ we have $\mathrm{cut}_G(S') > \mathrm{cut}_G(S)$.

Claim 3.3. *Let $S \subseteq V$ be a minimum s,t-cut in G that is α-unfriendly for some $\alpha \leq 1/6$. Then, exactly one vertex in G has friendliness ratio less than $1/6$, and it is either s or t.*

Proof. Every vertex besides s,t must have friendliness ratio at least $1/2$, as otherwise moving it to the other side of the cut would yield a cut with a smaller value, contradicting the optimality of S. The cut S is α-unfriendly for $\alpha \leq 1/6$, and thus at least one of s,t has friendliness ratio less than $1/6$. Assume towards contradiction this holds for both vertices, then together with the optimality of S we have

$$\mathrm{cr}_S(t) + \mathrm{cr}_S(s) > 5/6 \cdot [\deg(s) + \deg(t)] \geq 5/3 \cdot \mathrm{cut}(S).$$

However, at most one edge in $E(S, V \setminus S)$ can be incident to both s and t, and hence $\mathrm{cr}_S(t) + \mathrm{cr}_S(s) \leq \mathrm{cut}(S) + 1$. This yields a contradiction if $\mathrm{cut}(S) \geq 2$, which indeed holds because we assumed G has no vertices of degree 1 and thus $\mathrm{cut}(S) \geq \mathrm{cr}_S(s) > 5/6 \cdot \deg(s) \geq 5/3$. □

Lemma 3.4. *Let $S \subseteq V$ be a minimal minimum s,t-cut in G, and suppose $s \in S$ has friendliness ratio less than $1/6$. Then $X = S \setminus \{s\}$ is a $\frac{1}{6}$-friendly cut in G and furthermore $\mathrm{cut}_G(X) \leq 2\deg(s)$.*

Proof. We first show that every vertex $x \in X$ has $\deg(x) \geq 3$. Indeed, we assumed G has no vertices of degree 1, and x also cannot have degree 2, as otherwise moving it to the other side of S would violate minimality of S; hence, $\deg(x) \geq 3$.

We proceed to prove that every vertex in G has friendliness ratio at least $1/6$ with respect to the cut X. Consider first vertices $x \in X$. We know that $\mathrm{cr}_S(x)/\deg(x) \leq 1/2$, as otherwise moving x to the other side of the cut yields a cut with a smaller value, in contradiction to optimality of S. Furthermore, $|\mathrm{cr}_X(x) - \mathrm{cr}_S(x)| \leq 1$ as removing s from S can change the cut by at most one edge incident to x. Therefore, the friendliness ratio of x with respect to X is at least $1 - (\mathrm{cr}_S(x) + 1)/\deg(x) \geq 1 - 1/2 - 1/\deg(x) \geq 1/6$.

Next, consider vertices in $V \setminus X$. Vertices other than s,t have, by same argument as before, friendliness ratio at least $1/2 \geq 1/6$ in the cut S, which can only increase when moving s to the other side of the cut. The vertex t has, by Claim 3.3, friendliness ratio at least $1/6$ in the cut S, which can only increase when we move s to the other side of the cut. For the vertex s, its friendliness ratio in the cut S is less than $1/6$, and thus in the cut X it is $1 - \mathrm{cr}_X(s)/\deg(s) = 1 - (\deg(s) - \mathrm{cr}_S(s))/\deg(s) > 5/6$. This concludes the bound $1/6$ for all vertices in G.

Finally, the optimality of the cut S implies $\mathrm{cut}_G(X) \leq \mathrm{cut}_G(S) + \deg(s) \leq 2\deg(s)$. □

Proof of Theorem 2.1. Let $H_{\mathrm{fr}} = (V, E_{\mathrm{fr}}, w_{\mathrm{fr}})$ be a $(1/6, 2n)$-friendly cut sparsifier of G, and let $w_{\mathrm{fr}}(\cdot,\cdot)$ denote the total weight of corresponding edges in $E_{\mathrm{fr}}(\cdot,\cdot)$. To simplify notation, let

$$f_v(X) \coloneqq \mathrm{cut}_{H_{\mathrm{fr}}}(X) - 2w_{\mathrm{fr}}(X, v) + \deg(v), \tag{1}$$

for $v \in V$ and $X \subseteq V \setminus \{v\}$, and notice that it can be computed given H_{fr} and the vertex degrees in G. The algorithm to recover a minimum s,t-cut in G is as follows. Find a minimum s,t-cut in H_{fr} using any standard algorithm, and denote it by $S_{\mathrm{fr}} \subset V$. In addition, let

$$X^{(s)} = \arg \min_{X \subseteq V \setminus \{s,t\}} f_s(X),$$

and define $X^{(t)}$ similarly. Finally, return a set associated with the minimum of those three values, namely, one of $S_{\mathrm{fr}}, X^{(s)} \cup \{s\}, X^{(t)} \cup \{t\}$ together with its associated value.

We proceed to prove the correctness of the algorithm. We have $\text{cut}_G(A \cup B) = \text{cut}_G(A) + \text{cut}_G(B) - 2 \cdot |E(A, B)|$ whenever $A, B \subseteq V$ are disjoint, and thus the value of a cut $S = X \cup \{v\}$ where $X \subseteq V \setminus \{v\}$ can be written as

$$\text{cut}_G(S) = \text{cut}_G(X) - 2 \cdot |E(X, v)| + \deg(v). \tag{2}$$

Let us now show that $\text{cut}_G(X \cup \{v\}) \leq f_v(X)$ by comparing (2) with (1) term by term. Observe that $\text{cut}_G(X) \leq \text{cut}_{H_{\text{fr}}}(X)$ for all $X \subseteq V$, as H_{fr} is a contraction of G. In addition, $w_{\text{fr}}(X, v)$ has the same value as $|E(X, v)|$ up to infinite-weight edges in H_{fr} (arising from contractions). However, changing the weight of such edges does not affect $f_v(X)$, because if v is contracted with a vertex of X then the weight of the edge between them cancels out completely in $f_v(X)$ (it appears in all three terms). We conclude that $\text{cut}_G(X \cup \{v\}) \leq f_v(X)$; furthermore, equality holds if and only if no edge of $E(X, V \setminus X)$ is contracted in H_{fr}. In addition, the value of the cut S_{fr} in H_{fr} (which is a contraction of G) is clearly at least as large as a minimum s, t-cut value in G. Hence, the algorithm's output value is always at least as large as $\lambda_{s,t}(G)$.

We now argue that the algorithm output is actually $\lambda_{s,t}(G)$ (together with a minimum s, t-cut in G). If there exists a minimum s, t-cut that is 1/6-friendly, then it is preserved in H_{fr}, and thus the algorithm will find its value when computing S_{fr}. Otherwise, fix some 1/6-unfriendly minimal minimum s, t-cut $S \subset V$ in G, and assume without loss of generality that $s \in S$ has friendliness ratio less than 1/6. By Lemma 3.4, $X = S \setminus \{s\}$ is a 1/6-friendly cut with at most $2n$ edges. Therefore, the cut of X is preserved in H_{fr}, i.e., no edge in $E(X, V \setminus X)$ is contracted in H_{fr} and hence $f_s(X) = \text{cut}_G(X \cup \{s\}) = \lambda_{s,t}(G)$, and the algorithm will find this value when it evaluates $X^{(s)}$. □

Unfortunately, recovering a minimum s, t-cut using Theorem 2.1 requires iterating over all the cuts of H_{fr}, potentially taking exponential time as H_{fr} might have $\Omega(n)$ vertices. While this is acceptable in the cut-query and streaming models, where one is mostly concerned about the number of queries or space, it is excessive for the fully-dynamic setting where fast update times is crucial. We therefore propose to modify the friendly-cut sparsifier into an APMC sparsifier $H_{\text{ap}} = (V_{\text{ap}}, E_{\text{ap}}, w_{\text{ap}})$ of the input G, which does admit efficient minimum s, t-cut algorithms.

The APMC sparsifier is constructed by applying the star transform to every super-vertex in a $(1/6, 2n)$-friendly cut sparsifier H_{fr} of G. Let us outline why this process yields an APMC sparsifier. Fix $s, t \in V$, and recall that applying the star transform on u decomposes u into its constituent vertices, and adds a new proxy vertex u' connected to each vertex $v \in u$ with an edge of weight $|E(v, u \setminus v)|$. Consider some minimal minimum s, t-cut $S \subseteq V$ in G such that $s \in S$, and denote the super-vertex in H_{fr} that contains s by u. If S is 1/6-friendly then it is preserved in H_{fr}. Otherwise, by Lemma 3.4 we can write $S = X \cup \{s\}$, where X is a 1/6-friendly cut with at most $2n$ edges. Now, we have $\text{cut}_G(A \cup B) = \text{cut}_G(A) + \text{cut}_G(B) - 2 \cdot |E(A, B)|$ whenever $A, B \subseteq V$ are disjoint, and thus we can write $\text{cut}_G(S)$ as

$$\begin{aligned}
\text{cut}_G(X \cup \{s\}) &= \text{cut}_G(X) - 2 \cdot |E(X, s)| + \deg(s) \\
&= \text{cut}_G(X) - 2 \cdot |E(X, s)| + |E(s, u \setminus s)| + |E(s, X)| + |E(s, V \setminus (u \cup X))| \\
&= \text{cut}_G(X) - |E(X, s)| + |E(s, u \setminus s)| + |E(s, V \setminus (u \cup X))|.
\end{aligned} \tag{3}$$

Next, let us examine the corresponding cut $S = X \cup \{s\}$ in H_{ap}, but to be precise, we must assign each proxy vertices to some side of the cut. For a super-vertex $y \in H_{\text{fr}}$ such that $y \subseteq S$ or $y \subseteq V \setminus S$, we assign its proxy vertex y' to the same side as (all of) y; this way, none of the edges incident to y' cross the cut. Since X is 1/6-friendly, this handles all the super-vertices in H_{fr} except (possibly) u. We also handle u' by assigning it to the side opposite to s. Denote the corresponding cut value

by $\text{cut}_{H_{\text{ap}}}(X \cup \{s\})$, which has a slight abuse of notation because some proxy vertices are omitted. We can write it, similarly to (3) but replacing $E(s, u \setminus s)$ with the edge (s, u'), as

$$\text{cut}_{H_{\text{ap}}}(X \cup \{s\}) = \text{cut}_{H_{\text{ap}}}(X) - w_{\text{ap}}(X, s) + w_{\text{ap}}(s, u') + w_{\text{ap}}(s, V \setminus (u \cup X)),$$

where $w_{\text{ap}}(\cdot, \cdot)$ is the total weight of the edges in $E_{\text{ap}}(\cdot, \cdot)$. The cut X is preserved in H_{fr} since it is 1/6-friendly, and hence $\text{cut}_{H_{\text{ap}}}(X) = \text{cut}_G(X)$. Furthermore, $w_{\text{ap}}(X, s) = |E(X, s)|$ and $w_{\text{ap}}(s, V \setminus (u \cup X)) = |E(s, V \setminus (u \cup X))|$ because they correspond to edges outside the super-vertices of H_{fr}. Finally, by the definition of the star transform $w_{\text{ap}}(s, u') = |E(s, u \setminus s)|$. Altogether, $\text{cut}_{H_{\text{ap}}}(X \cup \{s\}) = \text{cut}_G(X \cup \{s\})$. We formalize this argument in Claim 3.5 below; the main difficulty is to show that H_{ap} does not have any "new" minimum s, t-cut, e.g., by assigning the proxy vertices differently. We then address the running time in Claim 3.6 below. Putting the claims together immediately proves Theorem 2.3.

Claim 3.5. *Let H_{fr} be a $(1/6, 2n)$-friendly cut sparsifier of an unweighted graph $G = (V, E)$ on n vertices. Then performing a star transform on every super-vertex in H_{fr} yields a graph $H_{\text{ap}} = (V_{\text{ap}}, E_{\text{ap}}, w_{\text{ap}})$ that is an APMC sparsifier of G with $|E_{\text{ap}}| \leq |E_{H_{\text{fr}}}| + n$.*

Claim 3.6. *Given a $(1/6, 2n)$-friendly cut sparsifier H_{fr} of an unknown unweighted graph $G = (V, E)$ on n vertices, and the degree of every vertex in G, one can perform a star transform on every super-vertex in H_{fr} in time $O(|E_H| + n)$.*

Proof of Claim 3.5. Fix a minimal minimum s, t-cut $S \subseteq V$ in G, let U be the set of super-vertices in H_{fr}, and let $U' = \{u' \mid u \in U\}$ be the set of their corresponding proxy vertices in H_{ap}. For every cut $C \subseteq V$ let $C_{\text{ap}} = A \cup C$ where $A = \arg\min_{T \subseteq U'} \text{cut}_{H_{\text{ap}}}(T \cup C)$ (breaking ties arbitrarily), which is the minimum cut corresponding to C in H_{ap}. We begin by showing that for any other s, t-cut $S' \subseteq V$ we have $\text{cut}_{H_{\text{ap}}}(S'_{\text{ap}}) \geq \text{cut}_G(S)$, and then consider S_{ap} itself and show that $\text{cut}_G(S) = \text{cut}_{H_{\text{ap}}}(S_{\text{ap}})$. Notice that we only need to consider cuts of the form C_{ap} for some $C \subseteq V$, since they minimize over all cuts in H_{ap} that assign the vertices of V to two sides in the same way (C and $V \setminus C$). Combining the two results, we obtain that S_{ap} is a minimum s, t-cut in H_{ap} and its value is equal to the minimum s, t-cut value in G. Therefore, H_{ap} is an APMC sparsifier of G.

Let $S' \subseteq V$ be some s, t-cut in G, U be the super-vertices of H_{fr}, and for each $u \in U$ denote $S'_u = (S' \cap u)$. We can write the value of the cut A in G as

$$\text{cut}_G(S') = \sum_{u \in U} |E(S_u, u \setminus S')| + |E(S_u, V \setminus (S' \cup u))|,$$

by partitioning the edges into those internal to super-vertices and those external to them. The term $|E(S'_u, V \setminus (S' \cup u))|$ is preserved in H_{fr} and in H_{ap} because it comprises of edges external to super-vertices.

Now, fix some super-vertex $u \in U$ and examine the term corresponding to the internal edges $|E(S'_u, u \setminus S')|$. In the induced graph $H_{\text{ap}}[u \cup \{u'\}]$, all edges are of the form (v, u') for $v \in u$. If $u' \in S'_{\text{ap}}$, the corresponding term is the weight of all the edges between u' and $u \setminus S'_u$, given by $w_{\text{ap}}(u \setminus S'_u, u') = \sum_{v \in u \setminus S'_u} |E(v, u \setminus v)| \geq |E(S'_u, u \setminus S')|$, where the equality is by the properties of the star transform. Similarly, if $u' \notin S'_{\text{ap}}$, the corresponding term is $w_{\text{ap}}(S'_u, u') = \sum_{v \in S'_u} |E(v, u \setminus v)| \geq |E(S'_u, u \setminus S')|$, where again the equality is by the properties of the star transform. Therefore, we find that $\text{cut}_{H_{\text{ap}}}(S'_{\text{ap}}) \geq \text{cut}_G(S') \geq \text{cut}_G(S)$, where the last inequality is by the minimality of S.

We now turn to prove that $\text{cut}_G(S) = \text{cut}_{H_{\text{ap}}}(S_{\text{ap}})$. If the cut S is 1/6-friendly then it is preserved in H_{fr} and hence also in H_{ap}. Otherwise, either s or t is the unique vertex with friendliness ratio $\leq 1/6$ by Claim 3.3, and assume without loss of generality that it is s and that $s \in S$. By

Lemma 3.4, $S = X \cup \{s\}$ where X is a 1/6-friendly cut with at most $2n$ edges. We argue that $\mathrm{cut}_{H_{\mathrm{ap}}}(X_{\mathrm{ap}} \cup \{s\}) = \mathrm{cut}_G(S)$ which yields the required result.

Examine the cut $X_{\mathrm{ap}} \cup \{s\}$ in H_{ap} and let u be the super-vertex in H_{fr} that contains s. Recall that similarly to (3),

$$\begin{aligned}\mathrm{cut}_G(X \cup \{s\}) &= \mathrm{cut}_G(X) - |E(s, X)| + |E(s, V \setminus (u \cup X))| + |E(s, u \setminus \{s\})| \\ \mathrm{cut}_{H_{\mathrm{ap}}}(X_{\mathrm{ap}} \cup \{s\}) &= \mathrm{cut}_{H_{\mathrm{ap}}}(X_{\mathrm{ap}}) - w_{\mathrm{ap}}(s, X) + w_{\mathrm{ap}}(s, V \setminus (u \cup X)) + w_{\mathrm{ap}}(s, u'),\end{aligned}$$

where in the last equation we used that the proxy vertex u' is on the opposite side of the cut from s. Observe that the proxy vertex u' is not in X_{ap} since a friendly cut cannot partition any super-vertex in H_{fr} and $s \notin X$. Hence, $u \cap X_{\mathrm{ap}} = \emptyset$ and therefore including u' in X_{ap} would increase its value, and we conclude that $u' \notin X_{\mathrm{ap}} \cup \{s\}$. We next prove that these two quantities above are equal, by showing that each term is equal to its corresponding term.

We begin by showing that $\mathrm{cut}_{H_{\mathrm{ap}}}(X_{\mathrm{ap}}) = \mathrm{cut}_G(X)$. Recall that X is a 1/6-friendly cut in G and $\mathrm{cut}_G(X) \le 2\deg(s) \le 2n$. Hence, no edge of $E(X, V \setminus X)$ is contracted in H_{fr} (and in H_{ap}). In addition, if a cut $A \subseteq V$ includes all the vertices inside a super-vertex u, then S'_{ap} must also include u', and we conclude $\mathrm{cut}_{H_{\mathrm{ap}}}(X_{\mathrm{ap}}) = \mathrm{cut}_G(X)$. Since $E(s, X)$ and $E(s, V \setminus u)$ are sets of edges external to super-vertices of H_{fr}, they are also preserved in H_{ap}. Finally, by the star transform operation we obtain $w_{\mathrm{ap}}(s, u') = |E(s, u \setminus s)|$, and $\mathrm{cut}_{H_{\mathrm{ap}}}(X \cup \{s\}) = \mathrm{cut}_G(X \cup \{s\})$ as required.

To conclude the proof we show the bound on the number of edges in H_{ap}. Every edge of E_{ap} that was not added during the star transform operation was already in $E_{H_{\mathrm{fr}}}$. Furthermore, for every super-vertex u in H_{fr} we add exactly $|u|$ new edges incident to the proxy vertex u'. Since $\sum_u |u| \le n$ we find that $|E_{\mathrm{ap}}| \le |E_{H_{\mathrm{fr}}}| + n$. □

Proof of Claim 3.6. For every super-vertex $u \in H_{\mathrm{fr}}$ and vertex $v \in u$ compute $|E(v, u)| = \deg(v) - E(v, V \setminus u)$, which altogether takes $O(|E_{H_{\mathrm{fr}}}|)$ time by iterating over each edge in H_{fr} once. Performing the star transform only requires knowing $|E(v, u \setminus v)| = |E(v, u)|$ for every $v \in u$. It takes $O(|E_{H_{\mathrm{fr}}}|)$ time to reconnect the edges to their original endpoints and an additional $O(n)$ time to add the edges incident to proxy vertices. □

4 Applications

In this section we leverage the structural results from Section 2 (proved in Section 3) to construct an APMC sparsifier of an input graph G, in order to prove our main theorems for the cut-query, streaming, and fully-dynamic models. As seen above, the main ingredient in constructing an APMC sparsifier is a friendly cut sparsifier. In the cut-query model, one can construct an (α, w)-friendly cut sparsifier using the following lemma from [KK25a], and putting it together with Theorem 2.3, we immediately obtain Theorem 1.2.

Lemma 4.1 (Lemma 1.7 of [KK25a])**.** *Given cut-query access to an unweighted graph G on n vertices and parameters α and w, one can recover all the edges (along with their endpoints in G) of a friendly (α, w)-cut sparsifier of G using $\tilde{O}(\alpha^{-1} n\sqrt{w})$ cut queries. The algorithm is randomized and succeeds with probability $1 - 1/\operatorname{poly}(n)$.*

Proof of Theorem 1.2. Begin by constructing a $(1/6, 2n)$-friendly cut sparsifier H_{fr} of G using Lemma 4.1 with $\alpha = 1/6$ and $w = 2n$. In addition, recover the degree of each vertex $v \in V$ by querying the cut $\{v\}$ and then apply Theorem 2.3 to construct an APMC sparsifier H_{ap} of G. Finally, build a Gomory-Hu tree of H_{ap} using some offline algorithm. The query complexity is dominated by the construction of H_{fr}, which takes $\tilde{O}(n^{3/2})$ cut queries by Lemma 4.1. □

Unfortunately, in the streaming and fully-dynamic models, efficient algorithms for constructing a friendly cut sparsifier are not known, and we provide the first such algorithms, based on the randomized construction of [AKT22].[5] The heart of the algorithm is to recursively apply an expander decomposition procedure on a contraction of the graph, shave some vertices from each expander and then contract the shaved expanders. We begin by defining the notion of expander decomposition.

Definition 4.2. *A graph G is a said to be a* ϕ-expander *if*

$$\forall S \subseteq V, \qquad \Phi_G(S) \coloneqq \frac{\mathrm{cut}_G(S)}{\min(vol(S), vol(V \setminus S))} \geq \phi,$$

where $vol(S) \coloneqq \sum_{v \in S} \deg(v)$ is called the volume of S. An $(\epsilon, \epsilon/\phi)$-expander decomposition *of a (multi)graph G is a partition of V into clusters $V = V_1 \cup \ldots \cup V_k$ such that the number of edges between clusters is at most $\epsilon \cdot |E|$, and every induced subgraph $G[V_i]$, for $i \in [k]$, is an (ϵ/ϕ)-expander.*

To fit our needs, we need to strengthen the friendly cut sparsifier construction of [AKT22]. We make a few modifications. The first one is to show that the construction succeeds even if we allow vertices to be shaved when they are within factor $\beta \in (0, 1)$ from the shaving conditions; this is critical in the streaming model, where one cannot store the entire graph and must rely on sparsifiers to approximate those conditions. The second modifications is to show that the construction is robust to deletions and insertions. The sparsifier H_{fr} is called *k-robust* if given a sequence of k' edge insertions $I = \{e_1, \ldots, e_{k'}\}$ and k'' edge deletions $D = \{e_1, \ldots, e_{k''}\}$ such that $k' + k'' \leq k$, the graph $(H_{\mathrm{fr}} \cup I) \setminus D$ is a $(2\alpha, w)$-friendly cut sparsifier of $(G \cup I) \setminus D$. We show that the construction of [AKT22] is in fact k-robust for $k = O(\sqrt{w})$. Finally, the proof of [AKT22] contains a flaw, which we fix in our construction. Our next lemma describes an algorithm for constructing a robust friendly cut sparsifier; its proof appears in Section 5, and the algorithm itself in Algorithm 1.

Lemma 4.3. *Suppose one has access to a procedure that, given a contracted multigraph and $\epsilon \in (0, 1)$, computes for it an $(\epsilon, \epsilon/\phi)$-expander decomposition. Then, given an unweighted graph G on n vertices, parameters $\alpha \in (0, 1/4), w \geq 1$ and slack $\beta \in (0, 1)$, one can compute the super-vertices of a $\sqrt{w}$-robust (α, w)-friendly cut sparsifier H_{fr} of G, that has $\tilde{O}((\beta\alpha)^{-1} n\phi\sqrt{w})$ edges. This algorithm makes* $\mathrm{polylog}(n)$ *calls to the expander decomposition procedure on contractions of G, and spends $O(m)$ time outside these calls.*[6]

Streaming Model. The main technical challenge in constructing a friendly cut sparsifier in the streaming model is to apply an expander decomposition recursively while making only a single pass over the data stream. Recent work [FKM23, CKMM24] has shown how to construct a single expander decomposition in a single pass. Using standard techniques, we extend these results to graphs G' that are contractions of the input graph G, and obtain the following generalization of [FKM23], whose proof appears in Section 6.

Theorem 4.4. *Given a dynamic stream of an unweighted graph G on n vertices, and parameter $\epsilon \in (0, 1)$, one can compute an $(\epsilon, \Omega(\epsilon/\log n))$-expander decomposition of a contraction of G using $\tilde{O}(\epsilon^{-2} n)$ space in one pass.*

[5] There exists also a deterministic construction with slightly worse guarantees.

[6] The slack parameter β applies in models where one cannot access G directly, but rather via cut sparsifiers. One example is the streaming model, where storing the entire graph is not feasible.

Using this theorem together with Lemma 4.3, we can construct a friendly cut sparsifier in the streaming model, as stated in the next lemma, whose proof appears in Section 6. The space requirement of this algorithm does not depend on w, because it only outputs the super-vertices of the friendly cut sparsifier (without its edges).

Lemma 4.5. *Given a graph G presented as a dynamic stream and parameters $\alpha \in (0, 1/2), w \geq 1$, on can output the super-vertices of an (α, w)-friendly cut sparsifier H_{fr} that has $|E_{H_{\text{fr}}}| = \tilde{O}(\alpha^{-1} n\sqrt{w})$ edges, using $\tilde{O}(\alpha^{-2} n)$ space in one pass.*

Proof of Theorem 1.7. In the first pass over the stream, find the super-vertices of a $(1/6, 2n)$-friendly cut sparsifier H_{fr} of G using Lemma 4.5. In the second pass, recover the edges of H_{fr} (along with original endpoints in G) and the degree of every vertex $v \in V$, and use them to construct an APMC sparsifier H_{ap} using Theorem 2.3. Finally, compute the minimum s, t-cuts in H_{ap} using some offline algorithm. The space complexity of the algorithm is dominated by storing the edges of H_{fr}, of which there are at most $\tilde{O}(n^{3/2})$ by Lemma 4.3. □

Fully Dynamic. In the fully dynamic model, we leverage the fact that our friendly cut sparsifier construction is robust to edge updates. This allows us to use a static algorithm to construct a $(\text{polylog}\, n, 2n)$-friendly cut sparsifier H of G, which is then valid for $\sqrt{n}$ edge updates, during which we can construct a new friendly cut sparsifier H'_{fr} in the background. Then, while the friendly cut sparsifier is valid every edge update requires $O(1)$ updates to the APMC sparsifier F. To obtain a deterministic algorithm, we employ a deterministic expander-decomposition procedure from [LS21].[7]

Theorem 4.6 (Corollary 2.5 of [LS21])**.** *There exists a deterministic algorithm that, given a weighted graph $G = (V, E, w)$ with m edges with weight bounded in $1 \leq w_e \leq U$ for all $e \in E$, and parameters $\epsilon \in (0, 1)$, $r \geq 1$, constructs an $(\epsilon, \epsilon/(\log^{O(r^4)} m \log U))$-expander decomposition of G in time $m(mU)^{O(1/r)} \log(mU)$.*

We employ Theorem 4.6 with $r = \log\log n$ and $U = O(n^2)$ to obtain an $(\epsilon, \epsilon/n^{o(1)})$-expander decomposition in time $m^{1+o(1)}$. Plugging this into Lemma 4.3 yields a deterministic algorithm for constructing a friendly cut sparsifier in time $m^{1+o(1)}$. Finally, to construct the Gomory-Hu tree of G we rebuild it from scratch on the APMC sparsifier after every update using the deterministic $m^{1+o(1)}$-time algorithm of [AKL+25].

Theorem 4.7. *Given an undirected, weighted graph $G = (V, E, w)$ with polynomially bounded weights, a Gomory-Hu tree of G can be constructed in $m^{1+o(1)}$ time.*

Proof of Theorem 1.5. The fully-dynamic algorithm is as follows. Throughout the updates, use the method explained further below to maintain a $(1/6, 2n)$-friendly cut sparsifier H_{fr} of the current graph G with $|E_{H_{\text{fr}}}| = n^{3/2+o(1)}$ and also the list of degrees in G. In addition, construct from it using Theorem 2.3 an APMC sparsifier H_{ap} of G with $O(|E_{H_{\text{fr}}}| + n) = n^{3/2+o(1)}$ edges. Finally, build for H_{ap} a Gomory-Hu tree using Theorem 4.7. The time for constructing the APMC sparsifier and the Gomory-Hu tree is thus $(O(|E_{H_{\text{fr}}}| + n))^{1+o(1)} = n^{3/2+o(1)}$ by the above bound on $|E_{H_{\text{fr}}}|$.

It remains to show how to maintain a $(1/6, 2n)$-friendly cut sparsifier H_{fr} using $n^{3/2+o(1)}$ worst-case update time. We divide the update sequence into epochs of $\sqrt{n}$ updates, and assume that at the beginning of each epoch we have a valid $(1/12, 2n)$-friendly cut sparsifier H_{fr} of the current graph

[7] In [AKT22] there is a different algorithm for deterministic construction of a friendly cut sparsifier. It leverages a stronger notion of expander decomposition, and employs a single expander decomposition instead a recursive one. To simplify our presentation, we use the same meta-algorithm for both the deterministic and randomized versions.

G. During the epoch, update the edges of H_{fr} according to edge updates to G, namely, whenever an edge is inserted/deleted into G, insert/delete that edge into H_{fr} (and update the degree list). Since the friendly cut sparsifier is $\sqrt{n}$-robust, it remains a $(1/6, 2n)$-friendly cut sparsifier after $\sqrt{n}$ edge updates. Therefore, throughout each epoch we can maintain H_{fr} using $O(1)$ time per update.

We now explain how to construct H_{fr} for the next epoch. For the first epoch, this is trivial because the graph G is empty and thus G itself is a friendly cut sparsifier. During the last $n^{1/2}/\log n$ updates of each epoch, construct in the background a new $(1/24, 2n)$-friendly cut sparsifier H'_{fr} of the current graph G. The total time for constructing H'_{fr} is $m^{1+o(1)} \leq n^{2+o(1)}$, where m is the number of edges in G at the beginning of the construction, by Lemma 4.3 and the complexity of the expander-decomposition procedure we use (Theorem 4.6 with $r = \log\log n$ and $U = O(n^2)$). Since this construction is spread over $\sqrt{n}/\log n$ updates, it takes $n^{2+o(1)}/(\sqrt{n}/\log n) = n^{3/2+o(1)}$ worst-case time per update. In addition, maintain the updates to G that occur during this construction in a buffer, and apply them to H'_{fr} once it is constructed, which takes $O(\sqrt{n}/\log n)$ time as each edge update takes $O(1)$ time. By the robustness property of H'_{fr}, it remains a $(1/12, 2n)$-friendly cut sparsifier of G after applying these updates. Overall, the algorithm uses $n^{3/2+o(1)}$ worst-case update time to maintain H_{fr}. Finally, our parameter choice implies $\phi = n^{o(1)}, \alpha = 1/6, w = 2n$ and $\beta = 1$, and thus by Lemma 4.3 we have $|E_{H_{\text{fr}}}| = n^{3/2+o(1)}$. □

5 Constructing a Friendly Cut Sparsifier

In this section we present a generic algorithm for constructing a friendly cut sparsifier, based on the work of [AKT22]. The algorithm is based on recursively applying an expander decomposition procedure on contractions of the graph, shaving some vertices from each expander and then contracting the shaved expanders. A vertex v is shaved from an expander if its degree is small compared to either the number of its incident edges exiting the expander or to a given threshold. We note that the original proof of the correctness of the algorithm had a flaw, which we fix later in the section. Furthermore, we strengthen the construction in two senses. The first, is that we allow for slack in the shaving condition, i.e. we do not contract some vertices with degree up to β^{-1} times as large as the threshold for some $\beta \in (0, 1)$. This is important in the streaming setting where one cannot directly access all graph edges, and consequently cannot verify contraction conditions exactly.

The second, is showing that the construction is $\alpha^{-1}\sqrt{w}$-robust, i.e. it continues to be a friendly cut sparsifier after $\alpha^{-1}\sqrt{w}$ edge insertions and deletions (and appropriately modifying the sparsifier). To do so we first prove that the construction also preserves all cuts $S \subseteq V$ such that every vertex $v \in S$ with degree at least $10\alpha^{-1}\sqrt{w}$ has friendliness ratio at most α (but vertices with smaller degree may have friendliness ratio less than α). This is stronger than the original condition, which required all vertices in S to have friendliness ratio at most α. The modified algorithm for constructing the friendly cut sparsifier is given in Algorithm 1.

The proof of Lemma 4.3 is by induction. For the base case we trivially have that $G_0 = G$ is a friendly cut sparsifier of G, and the following two claims complete the induction step. Both claims are based on analogous claims in [AKT22], however the proofs are slightly different due to the changes in the algorithm. Furthermore, our proof fixes the aforementioned flaw in the original proof of [AKT22]. Finally, the runtime bound follows as the expander decomposition is computed $O(\log(m/n)) = O(\log n)$ times, and the other steps in each iteration can be implemented in linear time in the number of edges.

Claim 5.1 (Correctness)**.** *Let $S \subseteq V$ be a cut with at most w edges such that every $v \in V$ with $\deg(v) \geq 10\alpha^{-1}\sqrt{w}$ has friendliness ratio at least α. Then for every iteration j of Algorithm 1, we have $\text{cut}_{G_j}(S) = \text{cut}_G(S)$.*

Algorithm 1 Friendly Cut Sparsifier Construction

Input: An unweighted graph $G = (V, E)$, parameters $w \geq 1, \alpha \in (0, 1/2), \beta \in (0, 1)$
Output: A robust (α, w)-friendly cut sparsifier
procedure FRIENDLY-CUT-SPARSIFIER(G, α, w)
  $\epsilon \leftarrow 0.01(\alpha\beta), j \leftarrow 1$
  $G_0 = (V_0, E_0) \leftarrow G$
  while $w_j \geq w$ **do**
    $w_j \leftarrow 4^{-j}(m/n)^2$
    add $(\epsilon/\phi)^{-1}\sqrt{w_j}$ self loops to every $v \in V_{j-1}$ ▷ ϕ is determined by the expander decomposition procedure
    $W_1, \ldots, W_k \leftarrow (\epsilon, \epsilon/\phi)$-expander decomposition of G_{j-1}
    for $i \in [k]$ **do**
      $R_i \leftarrow \left\{v \in W_i \mid d_{G_{j-1}}(v) < \max\{10\alpha^{-1}\sqrt{w_j}|v|, 4\alpha^{-1} \cdot |E_{j-1}(\{v\}, V_{j-1} \setminus W_i)|\}\right\}$
      $R'_i \leftarrow \left\{v \in W_i \mid \beta d_{G_{j-1}}(v) < \max\{10\alpha^{-1}\sqrt{w_j}|v|, 4\alpha^{-1} \cdot |E_{j-1}(\{v\}, V_{j-1} \setminus W_i)|\}\right\}$
      $R_i \leftarrow R_i \cup \rho$ where $\rho \subseteq R'_i$ ▷ arbitrary subset of R'_i
      $W'_i \leftarrow W_i \setminus R_i$
    $G_j = (V_j, E_j) \leftarrow$ contract in G_{j-1} each W'_i into a single vertex v_i ▷ keeping parallel edges and discarding self loops
    $j \leftarrow j + 1$
  return $G_{j-1}, \{W_i\}$.

Claim 5.2 (Size). *The number of edges in G_j when running Algorithm 1 with slack β and $\phi \geq \log n$ is at most $100\phi(\alpha\beta)^{-1}n\sqrt{w_j}$.*

Finally, the robustness of the sparsifier is due to the following lemma, whose proof is provided at the end of the section.

Lemma 5.3. *Let H_{fr} be the output of Algorithm 1 with $\alpha < 1/4$, $D = \{e_1, \ldots, e_a\}$ a sequence of a edge deletion and $I = \{e_1, \ldots, e_b\}$ a sequence of edge insertions such that $\sqrt{w} \geq a + b$. Then, the graph $(H_{\text{fr}} \cup I) \setminus D$ is a $(2\alpha, w)$-friendly cut sparsifier of $(G \cup I) \setminus D$.*

5.1 Proof of Correctness

In this section we give a corrected proof of Claim 2.3 from [AKT22], which states that the output of Algorithm 1 is a friendly cut sparsifier. Note that our proof is actually stronger as it shows that the sparsifier preserves even cuts that are not α-friendly, as long as all the vertices with low friendliness ratio have low degree.

We now describe the flaw in the original proof of [AKT22] and how we fix it. Fix a cut $S \subseteq V$ such that $\text{cut}_G(S) \leq w$ and some super-vertex $x \in S$ (recall that x is in the contracted graph G_{j-1}). Using this notation, the proof attempts to show that $|E_{j-1}(x, S)| \geq \alpha \deg_{G_{j-1}}(x)$; i.e. that the number edges incident to x that exit S is at least an α-fraction of the degree of x in G_{j-1}. To do so, the faulty proof uses the following bound

$$|E_{j-1}(x, S)| \geq \sum_{u \in x} |E(u, S)|, \tag{4}$$

which does not hold in general since some edges in $E(u, S)$ may be incident to other vertices in x, and thus are not counted in $|E_{j-1}(x, S)|$. We show that a slightly weaker bound still holds and is sufficient for constructing a friendly cut sparsifier.

Claim 5.4. *For every super-vertex $x \in V_{j-1}$ such that the degree of every $u \in x$ is at least $10\alpha^{-1}\sqrt{w}$, we have $|E_{j-1}(x,S)| \geq \frac{9\alpha}{10}\deg_{G_{j-1}}(x)$.*

Proof of Claim 5.4. We split the proof into two cases. First, when $d_{G_{j-1}}(x) \geq 2w$ we have that

$$\frac{|E_{j-1}(x,S)|}{\deg_{G_{j-1}}(x)} = \frac{\deg_{G_{j-1}}(x) - |E_{j-1}(x, V_{j-1} \setminus S)|}{\deg_{G_{j-1}}(x)} \geq 1 - \frac{w}{2w} = \frac{1}{2},$$

where the inequality is since $|E_{j-1}(x, V_{j-1} \setminus S)| \leq \mathrm{cut}_{G_{j-1}}(S) \leq w$ by the induction hypothesis. Since $\alpha \leq 1/2$ we obtain the desired bound. In the complementary case, we will show that $\deg_{G_{j-1}}(x) \geq (1 - \alpha^2/50) \cdot \mathrm{vol}(x)$. Intuitively, this means that most edges incident to vertices in x are external to the super-vertex rather than internal edges. This allows us to recover a bound similar to Equation (4). Formally,

$$\begin{aligned}
|E_{j-1}(x,S)| &= \sum_{u \in x} |E(u,S)| - |E(u,x)| = \deg_G(x) - (\mathrm{vol}(x) - \deg_{G_{j-1}}(x)) \\
&\geq \sum_{u \in x} \alpha \deg_G(u) - \frac{\alpha^2}{100}\mathrm{vol}(x) = (\alpha - \frac{\alpha^2}{50})\mathrm{vol}(x) \\
&\geq \left(\alpha - \frac{\alpha^2}{50}\right)\deg_{G_{j-1}}(x) \geq \frac{9\alpha}{10}\deg_{G_{j-1}}(x),
\end{aligned}$$

where the second equality is from $\sum_{u \in x} |E(u,x)| = \mathrm{vol}(x) - \deg_{G_{j-1}}(x)$, the first inequality uses $\sum_{u \in x} |E(u,S)| \geq \alpha \deg_G(u)$ by the friendliness ratio of every vertex with $\deg(v) \geq 10\alpha^{-1}\sqrt{n}$ and $\deg_{G_{j-1}}(x) \geq (1 - \alpha^2/50) \cdot \mathrm{vol}(x)$. The second inequality is since every vertex $u \in x$ has degree at least $10\alpha^{-1}\sqrt{w}$, and hence its friendliness ratio is at least α by the condition on S, and the last inequality is since $\alpha \leq 1/2$.

We conclude by showing that $\deg_{G_{j-1}}(x) \geq (1 - \alpha^2/100) \cdot \mathrm{vol}(x)$. Notice that $\deg_{G_{j-1}}(x) \geq \mathrm{vol}(x) - 2\binom{|x|}{2}$ since G is an unweighted graph. By line 11 of Algorithm 1, we have that $\deg(u) \geq 10\alpha^{-1}\sqrt{w}$ for every $u \in x$ and hence $\mathrm{vol}(x) \geq 10\alpha^{-1}\sqrt{w}|x|$. Furthermore, by our assumption above $2w \geq \deg_{G_{j-1}}(x) \geq 10\alpha^{-1}\sqrt{w}|x|$, where the last inequality holds as x is included in the shaved cluster W_i'. Moving sides we find $|x| \leq \alpha\sqrt{w}/5$. Finally, using the bounds on $\mathrm{vol}(x)$ and $|x|$,

$$\frac{\deg_{G_{j-1}}(x)}{\mathrm{vol}(x)} \geq 1 - \frac{2\binom{|x|}{2}}{\mathrm{vol}(x)} \geq 1 - \frac{|x|^2}{10\alpha^{-1}\sqrt{w}|x|} \geq 1 - \frac{\alpha\sqrt{w}/5}{10\alpha^{-1}\sqrt{w}} \geq 1 - \frac{\alpha^2}{50}. \qquad \square$$

Proof of Claim 5.1. By the induction hypothesis, we have $\mathrm{cut}_{G_{j-1}}(S) = \mathrm{cut}_G(S)$. When this assumption holds no vertex in S is contracted with any vertex in $V \setminus S$ in any previous iterations. Since this is the case, we will abuse the notation by treating S as a vertex subset of both G and G_{j-1}. Assume towards contradiction that two super vertices $x, y \in V_{j-1}$ such that $x \in S, y \notin S$ are contracted in iteration j.

If x, y are contracted then they must be in the same shaved cluster W_i'. Let $L = W_i \cap S$ and $R = W_i \setminus S$. Note that both are sets not empty because $x \in L, y \in R$. Since W_i is an ϵ/ϕ-expander we have

$$\min\{\mathrm{vol}(L), \mathrm{vol}(R)\} \leq \frac{|E_{j-1}(L,R)|}{\epsilon/\phi} \leq \frac{w\phi}{\epsilon}, \tag{5}$$

where the last inequality follows since $|E_{j-1}(L,R)| \leq \mathrm{cut}_{G_{j-1}}(S) = \mathrm{cut}_G(S) \leq w$ by the induction hypothesis. To proceed, we lower bound $\min\{\mathrm{vol}(L), \mathrm{vol}(R)\}$ to show a contradiction. We focus on lower bounding $\mathrm{vol}(L)$, and the argument for $\mathrm{vol}(R)$ is symmetric. Notice that since x was

not shaved, then $|E_{j-1}(x, V_{j-1} \setminus W_i)| \leq \alpha \deg_{G_{j-1}}(x)/4$ by line 11 of Algorithm 1. Furthermore, every vertex $u \in x$ has degree at least $10\alpha^{-1}\sqrt{w_j}$ since it was contracted in some prior iteration. Therefore, by Claim 5.4 we have $|E_{j-1}(x, S)| \geq \frac{9\alpha}{10} \deg_{G_{j-1}}(x)$, Combining these two bounds, we find that

$$|E_{j-1}(x, L)| \geq \left(\frac{9}{10}\alpha - \frac{\alpha}{4}\right) \deg_{G_{j-1}}(x) \geq \frac{13}{2}\sqrt{w_j}|x|,$$

where the first inequality is since L is S limited to W_i and the last inequality follows since x was not shaved, and thus $\deg_{G_{j-1}}(x) \geq 10\alpha^{-1}\sqrt{w_j}|x|$ by line 11 of Algorithm 1. Now observe that $|E_{j-1}(x, L)| \leq |x| \cdot |L|$ since G is an unweighted graph. These two bounds imply that $|L| \geq \frac{13}{2}\sqrt{w_j}$. Recalling that $(\epsilon/\phi)^{-1}\sqrt{w_j}$ self loops are added to every vertex in G_{j-1} at the beginning of iteration j, we have that $\mathrm{vol}(L) \geq \frac{13}{2}\frac{1}{\epsilon/\phi}w_j$. Thus, (recalling that a symmetric argument holds for $\mathrm{vol}(R)$) we arrive at a contradiction to Equation (5) since $\min\{\mathrm{vol}(L), \mathrm{vol}(R)\} \geq \frac{13}{2}\frac{1}{\epsilon/\phi}w_j > \frac{w\phi}{\epsilon}$. □

5.2 Proof of Size Guarantee

In this section we prove Claim 5.2, which is similar to Claim 2.4 in [AKT22]. The main difference between the proofs is the addition of the slack parameter β in the contraction criteria.

Proof of Claim 5.2. Notice that G' has three types of edges,

1. The outer edges of the expander decomposition, of which there are at most $m'_j \coloneqq \epsilon(|E_{j-1}| + n \cdot (\epsilon/\phi)^{-1}\sqrt{w_j})$, where the second term is since $(\epsilon/\phi)^{-1}\sqrt{w_j}$ self loops are added to every vertex in G_{j-1} prior to the expander decomposition.

2. The edges adjacent to vertices shaved because of low degree, i.e. those with $\beta \deg(v) < 10\alpha^{-1}\sqrt{w_j}$, of which there are at most $10(\alpha\beta)^{-1}n\sqrt{w_j}$.

3. Edges that are incident to vertices that were shaved because at least $\alpha \deg(v)/4$ of their edges went outside their expander. For every $v \in V$ let $d_{out}(v)$ be the cardinality of the edge set $E(\{v\}, V \setminus W)$, where W is the cluster to which v belongs. Furthermore, let $X \subseteq V$ be the set of the aforementioned shaved vertices, observe that the total number of inter-cluster edges is at most $O(\epsilon(|E_{j-1}| + n \cdot (\epsilon/\phi)^{-1}\sqrt{w_{j-1}})) = \sum_{v \in V} d_{out}(v)$ as explained above. Since for every $x \in X$ we have $\beta^{-1}d_{out}(x) \geq \alpha^{-1} \deg(x)/4$, we find that the number of edges incident to X is at most $4(\alpha\beta)^{-1}m'_j$.

Summing up the three types of edges, we find that the total number of edges in G_j is at most

$$\begin{aligned}
|E_j| &\leq (4(\alpha\beta)^{-1} + 1)m'_j + (\alpha\beta)^{-1}n\sqrt{w_j} \\
&= (4(\alpha\beta)^{-1} + 1) \cdot \epsilon\left(|E_{j-1}| + n \cdot (\epsilon/\phi)^{-1}\sqrt{w_j}\right) + 10(\alpha\beta)^{-1}n\sqrt{w_j} \\
&\leq 8(\alpha\beta)^{-1} \cdot \epsilon|E_{j-1}| + 16\phi(\alpha\beta)^{-1}n\sqrt{w_j},
\end{aligned}$$

where the last inequality is by $\alpha \leq 1/2$ and $\phi \geq \log n$. Substituting $|E_{j-1}| \leq 100\phi(\alpha\beta)^{-1}n\sqrt{w_{j-1}}$ by the induction hypothesis, $\epsilon = 0.01\alpha\beta$, and $w_{j-1} = 4w_j$ we find that

$$|E_j| \leq 16\phi(\alpha\beta)^{-1}n\sqrt{w_j} + 16\phi(\alpha\beta)^{-1}n\sqrt{w_j} \leq 100\phi(\alpha\beta)^{-1}n\sqrt{w_j}$$ □

5.3 Robustness to Edge Updates

Proof of Lemma 5.3. Denote $G' = (G \cup I) \setminus D$ and $H'_{\text{fr}} = (H_{\text{fr}} \cup I) \setminus D$ and let $S \subseteq V$ be a 2α-friendly cut in G' with at most w edges. Denote the edge set of G' by E' We will show that $\text{cut}_{H'_{\text{fr}}}(S) = \text{cut}_{G'}(S)$, notice that this is equivalent to showing that no edge in $E'(S, V \setminus S)$ is contracted in H'_{fr}. We prove this by showing that S satisfies the conditions of Claim 5.1, i.e. every vertex $v \in V$ with $\deg_G(v) \geq 10\alpha^{-1}\sqrt{w}$ has friendliness ratio at least α in G (in regard to S). By Claim 5.1, this implies that no edge in $E(S, V \setminus S)$ is contracted in H_{fr}, and therefore no edge in $E'(S, V \setminus S)$ is contracted in H'_{fr} (since updates only add or remove edges, without changing the contraction structure).

Assume towards contradiction that there exists a vertex $v \in V$ with $\deg_G(v) \geq 10\alpha^{-1}\sqrt{n}$ and friendliness ratio less than α in G but friendliness ratio more than 2α in G'. We now examine how the friendliness ratio of v changes from G to G'. Notice that the friendliness ratio of v changes the most if all the edge updates are incident to v. Furthermore, we show that the friendliness ratio changes the most if all edge updates are insertions. After a single edge insertion the friendliness ratio of v in G' is at most $1 - \frac{\text{cr}(v)}{\deg(v)+1}$. Similarly, after a single edge deletion the friendliness ratio of v in G' is at most $1 - \frac{\text{cr}(v)-1}{\deg(v)-1}$. Now notice that,

$$1 - \frac{\text{cr}(v)}{\deg(v)+1} > 1 - \frac{\text{cr}(v)-1}{\deg(v)-1},$$

follows if $2\,\text{cr}(v) - \deg(v) > 1$; this holds since the friendliness ratio of v in G' is always less than $1/2$ as $\alpha < 1/4$ and $k < \deg(v)/2$. Therefore, it remains to analyze the case when all edge updates are insertions. The friendliness ratio of v in G' after k edge insertions is at most

$$1 - \frac{\text{cr}(v)}{\deg(v)+k} < 1 - \frac{(1-\alpha)\deg(v)}{\deg(v)+\alpha\deg(v)} < 2\alpha,$$

where the first inequality is since the friendliness ratio of v in G is less than α, and since $k \leq \sqrt{w} \leq \alpha\deg(v)$ by our assumption on the degree of v. Therefore, v has friendliness ratio smaller than 2α in G', and we arrive at a contradiction. □

6 Friendly Cut Sparsifiers in Dynamic Streams

In this section we provide an algorithm for finding the super-vertices of an (α, w)-friendly cut sparsifier in the dynamic streaming model, proving Lemma 4.5. The construction is an implementation of Algorithm 1 in the dynamic streaming model.

Recall that the main ingredient in Algorithm 1 is performing an expander decomposition on a contraction of the input graph at each iteration of the main loop. The expander decomposition procedure we leverage is a variant of the algorithm of [FKM23]. The main technical tool used in the expander decomposition is a (δ, ϵ) power cut sparsifier.

Definition 6.1. *A (δ, ϵ)-cut sparsifier for a (multi)graph $G = (V, E_G)$ is a graph $H = (V, E_H, w_H)$ such that,*

$$\forall S \subseteq V, \qquad (1-\delta)\cdot w_G(E(S,\overline{S})) - \epsilon \cdot vol(S) \leq w_H(S,\overline{S}) \leq (1+\delta)\cdot w_G(E(S,\overline{S})) + \epsilon \cdot vol(S).$$

A distribution $\mathcal{D}$ over graphs H is called a (δ, ϵ, p)-power cut sparsifier distribution for G, if for every partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V into disjoint sets, with probability at least $1 - p$ over the

choice of $H \sim \mathcal{D}$, for all $C \in \mathcal{C}$ we have that $H\{C\}$ is a (δ, ϵ)-cut sparsifier of $G\{C\}$, where $G\{C\}$ is the induced graph on C after adding self loops to every $v \in C$ such that it has the same degree in $G\{C\}$ as in G. A sample H from $\mathcal{D}$ is called a (δ, ϵ, p)-power cut sparsifier of G.

One of the main technical results of [FKM23] is that one can sample from a $(\delta, \epsilon, n^{-C})$-power cut sparsifier distribution in a dynamic stream using $\tilde{O}(n/(\delta\epsilon))$ space. Combining this with an expander decomposition tailored to power cut sparsifiers, they obtain the following result.

Theorem 6.2 (Theorem 1 of [FKM23])**.** *Given $K = O(\epsilon^{-1} \operatorname{polylog} n)$ $(1/(5 \log n), \epsilon)$-power cut sparsifiers of a graph G on n vertices for some $\epsilon \geq 0$, one can find an $(\epsilon, \epsilon/\log n)$-expander decomposition of G.*

In order to perform expander decomposition recursively, we need to sample from a $(\delta, \epsilon, n^{-C})$-power cut sparsifier distribution for a contraction of G, which is only defined after the stream ends. The following lemma, whose proof is provided in Section 6.1, gives an algorithm for this. We note that an earlier version of [FKM23] contained a similar result, however it was removed in the most up-to-date version.[8]

Lemma 6.3. *Given an unweighted graph G on n vertices presented as a dynamic stream, and a contraction G' of G, there exists an algorithm using $\tilde{O}(n/(\delta\epsilon))$ space that outputs with high probability a sample from a $(\delta, \epsilon, n^{-C})$-power cut sparsifier distribution of G' for any constant $C > 0$.*

The proof of Lemma 4.5 also needs a cut sparsifier of G to estimate vertex degrees.

Definition 6.4. *A* $(1 \pm \epsilon)$-cut sparsifier *a graph G is a weighted subgraph H that satisfies*

$$\forall S \subseteq V, \qquad (1-\epsilon) \cdot \operatorname{cut}_G(S) \leq \operatorname{cut}_H(S) \leq (1+\epsilon) \cdot \operatorname{cut}_G(S).$$

Theorem 6.5 (Theorem 3.3 of [AGM12])**.** *Given a graph G presented as a dynamic stream, there exists an algorithm that constructs a $(1 \pm \epsilon)$-cut sparsifier of G using $\tilde{O}(n/\epsilon^2)$ space in one pass.*

Proof of Lemma 4.5. Throughout the proof we use $\beta = 1/2$ as the slack parameter in Algorithm 1 and fix $\epsilon = 0.01(\alpha\beta) \cdot \log^{-3} n$. The proof follows by implementing Algorithm 1 in the dynamic streaming model using Theorem 6.2 and Lemma 6.3. Begin by constructing $O((K \cdot \log n) + \log n)$ copies of the data structure of Lemma 6.3 during the stream with $\delta = 1/(5 \log n)$ and ϵ as above, where $K = \Theta(\epsilon^{-1} \operatorname{polylog} n)$. This is because K power cut sparsifiers are needed for each level of the expander decomposition, and there are $O(\log n)$ levels of recursion. In parallel, maintain a quality $(1 \pm 1/100)$-cut sparsifier H_c of G using Theorem 6.5. Therefore, the overall space complexity is $\tilde{O}(n/\epsilon^{-2}) = \tilde{O}(n/\alpha^{-2})$.

We now explain how to implement the main loop of Algorithm 1 with slack parameter $\beta = 1/2$. Fix some iteration j of the main loop. Begin by adding $(\epsilon/\phi(n))^{-1}\sqrt{w_j}$ self loops to every $v \in V_{j-1}$ in every power cut sparsifier for this level. Note that this can be implemented by simulating the insertion of these self loops at the end of the stream. Then, find a $(\epsilon, \epsilon/\log n)$-expander decomposition $W_1, \ldots, W_k$ of G_{j-1} by using Theorem 6.2.

The main challenge is finding the shaved clusters W_i'. Let $\widetilde{\deg}(u) \coloneqq \operatorname{cut}_{H_c}(u)$. We now present the modified shaving conditions that use $\widetilde{\deg}(u)$ to estimate the degree of u and a power cut sparsifier to estimate the number of edges between u and $V_{j-1} \setminus W_i'$. For each modified criteria we

[8] The construction was part of a proof for constructing a spanner in the streaming setting that contained an error, and thus was removed. However, the source of the error was elsewhere, and the expander decomposition for a contraction is still valid.

first show that it successfuly shaves every vertex satisfying the original criteria, and then show that any additional vertex is shaved has slack 1/2. The first condition is to shave every super vertex $u \in V_{j-1}$ that satisfies

$$\widetilde{\deg}(u)/(1+1/100) < 10\alpha^{-1}\sqrt{w_j}|u|$$

Notice that by the guarantee of the cut sparsifier, if $\deg_{G_{j-1}}(u) < 10\alpha^{-1}\sqrt{w_j}|u|$ then

$$\frac{\widetilde{\deg}(u)}{1+1/100} < \frac{(1+1/100)d_{G_{j-1}}(u)}{1+1/100} = d_{G_{j-1}}(u) < 10\alpha^{-1}\sqrt{w_j}|u|,$$

and thus the algorithm shaves u. Furthermore, no vertex of degree at least $10\alpha^{-1}\sqrt{w_j}|v|/(1-1/100)$ is shaved. Therefore, this implements the degree condition in line 11 with slack $1/(1+1/100) > 1/2$.

For the second condition, we need to estimate the number of edges between u and $V_{j-1} \setminus W_i$. Notice that $E_{G_{j-1}}(u, V_{j-1} \setminus W_i)$ corresponds to the cut $S = \{u\}$ in the induced graph $G[(V_j \setminus W_i) \cup \{u\}]$, and thus it can be estimated using a power cut sparsifier. There are $O(n)$ such induced graphs, and since they are all fixed in advance (before opening the sketch), taking a union bound we find that a single sample from the power cut distribution is sufficient to approximate cuts within all of them with probability at least $1 - n^{-C+1}$. The second condition (the out-degree condition) is to shave every super vertex $u \in V_{j-1}$ is modified to be

$$\rho \cdot \widetilde{\deg}(u) < 4\alpha^{-1} w_{H_{\mathrm{p}}}(E_{G_{j-1}}(u, V_{j-1} \setminus W_i)), \tag{6}$$

where H_{p} is a (δ, ϵ)-cut sparsifier for $G_{j-1}\{u \cup V_{j-1} \setminus W_i\}$ and $\rho \in (0,1)$ is some constant to be determined later. We begin by showing that if $4\alpha^{-1}|E_{G_{j-1}}(u, V_{j-1} \setminus W_i)| > \deg(u)$ then this condition holds. By the properties of the (δ, ϵ)-cut sparsifier

$$\begin{aligned}
4\alpha^{-1} w_{H_{\mathrm{p}}}(E_{G_{j-1}}(u, V_{j-1} \setminus W_i)) &\geq 4\alpha^{-1}\left((1-\delta)\cdot|E_{G_{j-1}}(u, V_{j-1} \setminus W_i)| - \epsilon \cdot \deg(u)\right) \\
&\geq 4\alpha^{-1}\left(\left(1 - \frac{1}{5\log n}\right)|E_{G_{j-1}}(u, V_{j-1} \setminus W_i)| - \frac{0.01\alpha\beta}{\log^3 n}\deg(u)\right) \\
&> \left(1 - \frac{1}{5\log n} - \frac{0.04}{\log^3 n}\right)\deg(u) \\
&\geq \left(1 - \frac{1}{5\log n} - \frac{0.04}{\log^3 n}\right)\left(1 + \frac{1}{100}\right)^{-1}\widetilde{\deg}(u),
\end{aligned}$$

where the last inequality follows from the assumption on u. Therefore, choosing $\rho = 3/4$ we have that Equation (6) holds for any large enough n. We now show that if $8\alpha^{-1}|E_{G_{j-1}}(u, V_{j-1} \setminus W_i)| \leq \deg(u)$ then the condition does not hold. Notice that in this case

$$\begin{aligned}
4\alpha^{-1} w_H(E_{G_{j-1}}(u, V_{j-1} \setminus W_i)) &\leq 4\alpha^{-1}\left(1 + \frac{1}{5\log n}\right)|E_{G_{j-1}}(u, V_{j-1} \setminus W_i)| + \frac{0.04}{\log^3 n}\deg(u) \\
&\leq 4\alpha^{-1}\left(1 + \frac{1}{5\log n}\right)\frac{\alpha}{8}\deg(u) + \frac{0.04}{\log^3 n}\deg(u) \\
&\leq \left(\frac{1}{2} + \frac{1}{10\log n} + \frac{0.04}{\log^3 n}\right)\deg(u) \\
&\leq \left(\frac{1}{2} + \frac{1}{10\log n} + \frac{0.04}{\log^3 n}\right)\left(1 - \frac{1}{100}\right)^{-1}\widetilde{\deg}(u) \leq \frac{3}{4}\widetilde{\deg}(u),
\end{aligned}$$

where the second inequality is by the assumption on u and the last inequality is true for large enough n. Therefore, we can implement line 11 with slack 1/2. □

6.1 Power Cut Sparsifiers with Contraction

In this section we show how to construct a power cut sparsifier for a contraction of a graph G given in a dynamic stream. The construction is a modification of the algorithm of [FKM23]. Note that while the result of [FKM23] is stated for simple graphs, it is explicitly mentioned that this construction works for unweighted multigraphs as well. Furthermore, an earlier version of [FKM23] contained a similar result to Lemma 6.3, which was removed in the most up-to-date version. The construction was part of a proof for constructing a spanner in the streaming setting that contained an error, and thus was removed. However, the source of the error was elsewhere, and the expander decomposition for a contraction is still valid.

We begin by presenting the algorithm of [FKM23] for constructing a (δ, ϵ)-power cut sparsifier in a dynamic stream, and then explain how to modify it to handle contractions. To construct the sparsifier, sample each edge $(u, v) \in E$ independently with probability $p = \gamma(1/d(u) + 1/d(v))$, where $\gamma = O(\frac{\log^2 n}{\epsilon\delta})$. To implement this algorithm in a dynamic stream, initialize $\log n$ uniform hash functions $h_i : \binom{V}{2} \to \{0, 1\}$, which take an edge and return whether it is sampled in the i-th level.[9] These hash functions induce $\log n$ graphs, $G_i = (V, E_i)$ where an edge e is included E_i if $h_j(e) = 1$ for all $j \leq i$. Since it is impossible to store all the edges of G_i, the algorithm utilizes the following sparse-recovery procedure.

Lemma 6.6 (Folklore, e.g. Theorem 2.2 of [AGM12])**.** *There exists a linear sketch that recovers a k-sparse vector $x \in \mathbb{R}^n$ from a dynamic stream with success probability $1-p$ using $O(k \log n \log 1/p)$ space.*

For each $v \in V$, the algorithm maintains a sketch s_v^i of the vector $x_v^i \in \mathbb{R}^{\binom{V}{2}}$ where $x_v^i(e) = 1$ if e is incident to v in G_i, and 0 otherwise. For every vertex $v \in V$, let $i^*(v) = \lfloor \log_2(d(v))/(2\gamma) \rfloor$. It is straightforward to see that with high probability each vertex $v \in V$ has degree at most $8 \log n$ in $G_{i^*(v)}$. Therefore opening the sketch $s_v^{i^*(v)}$ of $x_v^{i^*(v)}$ recovers all edges incident to v in $G_{i^*(v)}$. Furthermore, these edges were sampled with probability at least $\gamma/d(v)$. The sparsifier is then formed by taking the union of all edges sampled for each $v \in V$, and giving each edge $e = (u, v)$ weight $2^{\min\{i^*(u), i^*(v)\}}$. One of the main theorems of [FKM23] is that this algorithm yields an (δ, ϵ)-power cut sparsifier with high probability using $\tilde{O}(n/(\epsilon\delta))$ space. We now show how to modify this algorithm to handle contractions.

Proof of Lemma 6.3. To construct the power cut sparsifier we now need an additional cut sparsifier H_c of G with quality $\epsilon = 1/100$. This can be constructed using $\tilde{O}(n)$ storage in one pass of streaming using Theorem 6.5.

We now describe the modified algorithm. Begin by assigning an arbitrary ordering on the vertices of G. During the stream perform the same algorithm, except when updating the sketch s_v^i of x_v^i, for every edge $e = (u, v)$ insertion set $x_v^i(e) = 1$ for the higher index vertex and $x_v^i(e) = -1$ for the lower index vertex. Then, at the end of the stream, once the algorithm is given the contraction V' of V, let $s_u^i = \sum_{v \in u} s_v^i$ for every $u \in V'$ and $i \in [\log n]$. To estimate the degree of $u \in V'$, the algorithm uses the cut sparsifier H_c to estimate $\text{cut}_G(u)$ which is equal to the degree. Then, let $i^*(u) = \lfloor \log_2(d(u))/(2\gamma) \rfloor$ and recover the edges in the sketch $s_u^{i^*(u)}$. The rest of the algorithm is identical to the original and the correctness analysis follows similarly. □

[9] Note that naively storing each such function requires $O(n^2)$ space, but one can use a pseudorandom generator to reduce the space to $\tilde{O}(n)$.

References

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 265–276. ACM, 2019. doi:10.1145/3313276.3316361.

[AD21] Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In *4th Symposium on Simplicity in Algorithms, SOSA 2021*, pages 172–180. SIAM, 2021. doi:10.1137/1.9781611976496.19.

[ADK+16] Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pages 335–344. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.44.

[AEG+22] Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 507–518. IEEE, 2022. doi:10.1109/FOCS54457.2022.00055.

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.

[AKL+22] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-Hu tree in nearly quadratic time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 884–895. IEEE, 2022. doi:10.1109/FOCS54457.2022.00088.

[AKL+25] Amir Abboud, Rasmus Kyng, Jason Li, Debmalya Panigrahi, Maximilian Probst Gutenberg, Thatchaphol Saranurak, Weixuan Yuan, and Wuwei Yuan. Deterministic almost-linear-time gomory-hu trees. *CoRR*, abs/2507.20354, 2025. doi:10.48550/ARXIV.2507.20354.

[AKT20] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 105–118. IEEE, 2020. doi:10.1109/FOCS46700.2020.00019.

[AKT21a] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. APMF<APSP? Gomory-Hu Tree for Unweighted Graphs in Almost-Quadratic Time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1135–1146. IEEE, 2021. doi:10.1109/FOCS52979.2021.00112.

[AKT21b] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory-hu tree in unweighted graphs. In *STOC '21*, pages 1725–1737. ACM, 2021. doi:10.1145/3406325.3451073.

[AKT22] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster gomory-hu trees. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 3630–3649. SIAM, 2022. doi:10.1137/1.9781611977073.143.

[ALPS23] Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. All-pairs max-flow is no harder than single-pair max-flow: Gomory-Hu trees in almost-linear time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 2204–2212. IEEE, 2023. doi:10.1109/FOCS57990.2023.00137.

[ASW25] Aditya Anand, Thatchaphol Saranurak, and Yunfan Wang. Deterministic edge connectivity and max flow using subquadratic cut queries. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 124–142. SIAM, 2025. doi:10.1137/1.9781611978322.4.

[Bed61] S Bedrosian. Converse of the star-mesh transformation. *IRE Transactions on Circuit Theory*, 8(4):491–493, 1961.

[BHKP07] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In *39th Annual ACM Symposium on Theory of Computing*, STOC'07, pages 605–614, 2007. doi:10.1145/1250790.1250879.

[BK96] András A. Benczúr and David R. Karger. 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*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.

[BL16] Amir Ban and Nati Linial. Internal partitions of regular graphs. *J. Graph Theory*, 83(1):5–18, 2016. doi:10.1002/JGT.21909.

[BTV06] Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. The satisfactory partition problem. *Discret. Appl. Math.*, 154(8):1236–1245, 2006. doi:10.1016/J.DAM.2005.10.014.

[BvdBG+22] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.20.

[CGJS23] Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Parallel submodular function minimization. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/d8a7f2f7e346410e8ac7b39d9ff28c4a-Abstract-Conference.html.

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1158–1167. IEEE, 2020. doi:10.1109/FOCS46700.2020.00111.

[CKL+25] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *J. ACM*, 72(3):19:1–19:103, 2025. doi:10.1145/3728631.

[CKMM24] Yu Chen, Michael Kapralov, Mikhail Makarov, and Davide Mazzali. On the streaming complexity of expander decomposition. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*, volume 297 of *LIPIcs*, pages 46:1–46:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.46.

[DGL+25] Matthew Ding, Alexandro Garces, Jason Li, Honghao Lin, Jelani Nelson, Vihan Shah, and David P. Woodruff. Space complexity of minimum cut problems in single-pass streams. In *16th Innovations in Theoretical Computer Science Conference, ITCS 2025*, volume 325 of *LIPIcs*, pages 43:1–43:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ITCS.2025.43.

[dVC25] Tijn de Vos and Aleksander B. G. Christiansen. Tree-packing revisited: Faster fully dynamic min-cut and arboricity. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 700–749. SIAM, 2025. doi:10.1137/1.9781611978322.21.

[EGIN97] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. doi:10.1145/265910.265914.

[EHL25] Antoine El-Hayek, Monika Henzinger, and Jason Li. Fully dynamic approximate minimum cut in subpolynomial time per operation. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 750–784. SIAM, 2025. doi:10.1137/1.9781611978322.22.

[FHHP19] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM J. Comput.*, 48(4):1196–1223, 2019. doi:10.1137/16M1091666.

[FKM23] Arnold Filtser, Michael Kapralov, and Mikhail Makarov. Expander decomposition in dynamic streams. In *14th Innovations in Theoretical Computer Science Conference, ITCS 2023*, volume 251 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.50.

[FKN+22] Asaf Ferber, Matthew Kwan, Bhargav Narayanan, Ashwin Sah, and Mehtaab Sawhney. Friendly bisections of random graphs. *Communications of the American Mathematical Society*, 2(10):380–416, 2022.

[Fre97] Greg N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM J. Comput.*, 26(2):484–538, 1997. doi:10.1137/S0097539792226825.

[GH61] Ralph E. Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[GHN+23] Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. Fully dynamic exact edge connectivity in sublinear time. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 70–86. SIAM, 2023. doi:10.1137/1.9781611977554.CH3.

[GHP20] Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. *SIAM J. Discret. Math.*, 34(1):130–162, 2020. doi:10.1137/17M1163153.

[GI91] Zvi Galil and Giuseppe F. Italiano. Fully dynamic algorithms for edge-connectivity problems (extended abstract). In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 317–327. ACM, 1991. doi:10.1145/103418.103454.

[GJ25] Hossein Gholizadeh and Yonggang Jiang. A subquadratic two-party communication protocol for minimum cost flow. *CoRR*, abs/2510.03427, 2025. URL: https://arxiv.org/abs/2510.03427, arXiv:2510.03427.

[GK00] Michael U. Gerber and Daniel Kobler. Algorithmic approach to the satisfactory graph partitioning problem. *Eur. J. Oper. Res.*, 125(2):283–291, 2000. doi:10.1016/S0377-2217(99)00459-2.

[GKYY25] Maximilian Probst Gutenberg, Rasmus Kyng, Weixuan Yuan, and Wuwei Yuan. A simple and fast reduction from Gomory-Hu trees to polylog maxflows. *CoRR*, abs/2509.02520, 2025. Accepted to SODA 2026. arXiv:2509.02520, doi:10.48550/ARXIV.2509.02520.

[GMT20] Ajinkya Gaikwad, Soumen Maity, and Shuvam Kant Tripathi. The satisfactory partition problem. *CoRR*, abs/2007.14339, 2020. URL: https://arxiv.org/abs/2007.14339, arXiv:2007.14339.

[GNT20] Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1260–1279. SIAM, 2020.

[GPRW20] Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S. Matthew Weinberg. New query lower bounds for submodular function minimization. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 64:1–64:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ITCS.2020.64.

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2212–2228. SIAM, 2021. doi:10.1137/1.9781611976465.132.

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

[HK99] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.

[HKMR25] Monika Henzinger, Evangelos Kosinas, Robin Münk, and Harald Räcke. Efficient contractions of dynamic graphs - with applications. In *33rd Annual European Symposium on Algorithms, ESA 2025*, volume 351 of *LIPIcs*, pages 36:1–36:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ESA.2025.36.

[HKP07] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient algorithms for computing all low $s-t$ edge connectivities and related problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 127–136. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283398.

[HRW20] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM J. Comput.*, 49(1):1–36, 2020. doi:10.1137/18M1180335.

[Iva25] Paata Ivanisvili. A local sufficient condition for the superadditivity that is strictly more general than the convexity. MathOverflow, 2025. URL:https://mathoverflow.net/q/490528 (version: 2025-04-04), AUTHOR PROFILE:https://mathoverflow.net/users/50901/paata-ivanisvili.

[JNS25] Yonggang Jiang, Danupon Nanongkai, and Pachara Sawettamalya. Minimum s–t cuts with fewer cut queries. *CoRR*, abs/2510.18274, 2025. Accepted to SODA 2026. URL: https://arxiv.org/abs/2510.18274, arXiv:2510.18274.

[JS21] Wenyu Jin and Xiaorui Sun. Fully dynamic $s-t$ edge connectivity in subpolynomial time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 861–872. IEEE, 2021. doi:10.1109/FOCS52979.2021.00088.

[JST24] Wenyu Jin, Xiaorui Sun, and Mikkel Thorup. Fully dynamic min-cut of superconstant size in subpolynomial time. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 2999–3026. SIAM, 2024. doi:10.1137/1.9781611977912.107.

[KK21] Lior Kalman and Robert Krauthgamer. Flow metrics on graphs. *CoRR*, abs/2112.06916, 2021. URL: https://arxiv.org/abs/2112.06916, arXiv:2112.06916.

[KK25a] Yotam Kenneth-Mordoch and Robert Krauthgamer. All-pairs minimum cut using $\tilde{O}(n^{7/4})$ cut queries. *CoRR*, abs/2510.16741, 2025. Accepted to SODA 2026. URL: https://arxiv.org/abs/2510.16741, arXiv:2510.16741.

[KK25b] Yotam Kenneth-Mordoch and Robert Krauthgamer. Cut-query algorithms with few rounds. In *33rd Annual European Symposium on Algorithms, ESA 2025*, volume 351 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ESA.2025.100.

[KK25c] Yotam Kenneth-Mordoch and Robert Krauthgamer. Simple algorithms for fully dynamic edge connectivity. *CoRR*, abs/2508.07783, 2025. arXiv:2508.07783, doi:10.48550/ARXIV.2508.07783.

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1131–1142. SIAM, 2013. doi:10.1137/1.9781611973105.81.

[KL98] David R. Karger and Matthew S. Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 69–78. ACM, 1998. doi:10.1145/276698.276714.

[KR14] Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014. doi:10.1016/J.IPL.2014.02.011.

[KT19] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.

[LLSZ21] Troy Lee, Tongyang Li, Miklos Santha, and Shengyu Zhang. On the cut dimension of a graph. In *36th Computational Complexity Conference, CCC 2021*, volume 200 of *LIPIcs*, pages 15:1–15:35. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CCC.2021.15.

[LPS21] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1124–1134. IEEE, 2021. doi:10.1109/FOCS52979.2021.00111.

[LS21] Jason Li and Thatchaphol Saranurak. Deterministic weighted expander decomposition in almost-linear time. *CoRR*, abs/2106.01567, 2021. URL: https://arxiv.org/abs/2106.01567, arXiv:2106.01567.

[MN20] Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.

[MN21] Sagnik Mukhopadhyay and Danupon Nanongkai. A note on isolating cut lemma for submodular function minimization. *CoRR*, abs/2103.15724, 2021. URL: https://arxiv.org/abs/2103.15724.

[Mor00] Stephen Morris. Contagion. *The Review of Economic Studies*, 67(1):57–78, 2000.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

[PRW24] Orestis Plevrakis, Seyoon Ragavan, and S. Matthew Weinberg. On the cut-query complexity of approximating max-cut. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*, volume 297 of *LIPIcs*, pages 115:1–115:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.115.

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018*, 2018. doi:10.4230/LIPICS.ITCS.2018.39.

[Tho07] Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. doi:10.1007/S00493-007-0045-2.

[TK00] Mikkel Thorup and David R. Karger. Dynamic graph algorithms with applications. In *7th Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, volume 1851 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2000. doi:10.1007/3-540-44985-X_1.

[vLO73] M. van Lier and R. H. J. M. Otten. Planarization by transformation. *IEEE Transactions on Circuit Theory*, 20(2):169–171, 1973. doi:10.1109/TCT.1973.1083633.

[Zel11] Mariano Zelke. Intractability of min- and max-cut in streaming graphs. *Inf. Process. Lett.*, 111(3):145–150, 2011. doi:10.1016/J.IPL.2010.10.017.

A Gomory-Hu Trees for Symmetric Submodular Functions

In this section we prove Theorem 1.4. We begin by restating the theorem for ease of reference.

Theorem 1.4 (Query Complexity Version of [GKYY25]). *Let $Q(n)$ be the query complexity of minimizing a symmetric submodular function over a ground set of size n. Then, there exists an algorithm for constructing a Gomory-Hu tree of a symmetric submodular $f : 2^{[n]} \to \mathbb{R}_+$ using $Q(n) \cdot \text{polylog}(n)$ queries.*

The proof is based on the reduction of [GKYY25] from constructing a Gomory-Hu tree of a graph to $\text{polylog}(n)$ calls to max-flow computations on graphs of total size $\tilde{O}(n)$.

Proof. The correctness of the algorithm follows from the fact that g is symmetric submodular, as is explicitly shown in [GKYY25]. Therefore, it remains to analyze the query complexity of the algorithm. The algorithm is based on partitioning the set of terminals, and then recursing on each part. The following claim shows that the total number of elements in all the recursive calls is only $O(n \log^2 n)$. We note that the claim is stated for weighted graphs, but the same proof applies to symmetric submodular functions as well.

Claim A.1 (Claim 5.3 of [GKYY25]). *A call on a function g of ground set size n to the Gomory-Hu tree algorithm, creates recursive calls on functions of total ground set size $O(n \log^2 n)$.*

In each recursive call, the algorithm makes either $O(\log^2 n)$ calls to an SFM procedure and $O(\text{polylog}\, n)$ calls to an isolating-cuts procedure. Outside these procedures the algorithm does not make any additional queries to g. The following theorem implements the isolating-cuts procedure using $O(\log n)$ calls to an SFM procedure.

Theorem A.2 (Theorem 2.4 of [MN21]). *Let g be a symmetric submodular function over a ground set V of size n, let $R \subseteq V$ be a set of terminals, and $\mathcal{A}$ be an SFM procedure. Then, there exists an isolating-cuts procedure for g and R that makes $O(\log |R|)$ calls to $\mathcal{A}$ with ground set size $O(n)$ and an additional $|R|$ calls to $\mathcal{A}$ on ground sets $U_1, \ldots, U_{|R|}$ where $\sum_{i=1}^{|R|} |U_i| \le n$.*

We now show that the function $Q(n)$ is superadditive, i.e., for every $n_1, \ldots, n_k$ such that $\sum_i n_i \le n$ we have $\sum_i Q(n_i) \le Q(n)$. To do so, we will use the following fact, whose proof is based on [Iva25] and is included at the end of the section for completeness.

Fact A.3. *Let $f : \mathbb{R} \to \mathbb{R}_+$ be a function such that $f(x)/x$ is non-decreasing, then f is superadditive.*

Notice that the query complexity of SFM is at least $\Omega(n)$ [GPRW20, LLSZ21] and hence it satisfies the conditions of the fact. Therefore, denoting the number of vertices in the j-th partition of the i-th recursive call by $n_{i,j}$, the total query complexity of the algorithm is

$$\sum_i \sum_j Q(n_{i,j}) \le \sum_i Q(n) \cdot \text{polylog}\, n \le Q(n) \cdot \text{polylog}\, n,$$

where the first inequality is since $n_{i,j} \le n$ and $\sum_j n_{i,j} \le n \cdot \text{polylog}\, n$, and the last inequality is by Claim A.1. □

Proof of Fact A.3. The proof is based on [Iva25] and is included for completeness. $f : \mathbb{R} \to \mathbb{R}_+$ be a function such that $f(x)/x$ is non-decreasing. Notice that $f(x) \le x f(x+y)/(x+y)$ for every $x, y > 0$ and similarly $f(y) \le y f(x+y)/(x+y)$. Therefore,

$$f(x) + f(y) \le \frac{(x+y) f(x+y)}{x+y} = f(x+y)$$ □

B Max-Flow Cut-Query Lower Bound

In this section we prove Lemma 1.8, proving a lower bound on the number of cut queries needed to recover every edge in some maximum s,t-flow in an unweighted graph G.

Proof. The lower bound is based on a graph family closely related to a hard example provided in [KL98]. We define a family $\mathcal{G}$ of unweighted graphs on $n+2$ vertices with two designated vertices s,t. The family is defined as follows. Initialize an empty graph G on $n+2$ vertices. Let $n' \coloneqq 8\cdot\lfloor n/8 \rfloor$ and ignore the excess $n-n'$ vertices for the rest of the proof. In addition, let φ be the closest integer to n' such that $n'/\sqrt{\varphi}$ is integral and φ is divisible by 8; note that $\varphi = \Theta(n)$. The graph is G is obtained by picking $n'/4$ arbitrary vertices $s_1,\ldots,s_{n'/4}$, and adding the edges $\{s,s_i\}_{i=1}^{n'/4}$ and similarly picking $n'/4$ different vertices $t_1,\ldots,t_{n'/4}$ and adding the edges $\{t,t_i\}_{i=1}^{n'/4}$. Then, partition the remaining $n'/2$ vertices into layers $L_1,\ldots,L_{n'/\sqrt{\varphi}}$, where each layer L_i contains $\sqrt{\varphi}/2$ vertices. Connect each layer L_i to layer L_{i+1} by sampling uniformly a random set of cardinality $\varphi/8$ from $L_i \times L_{i+1}$. Finally, add a complete bipartite graph between the sets $\{s_1,\ldots,s_{n'/4}\}$ and L_1, and $\{t_1,\ldots,t_{n'/4}\}$ and $L_{n'/\sqrt{\varphi}}$.

It is straightforward to see that the maximum s,t-flow in G is $\varphi/8$, since each layer L_i is connected to the next layer L_{i+1} by $\varphi/8$ edges. Furthermore, every edge between each consecutive layers L_i, L_{i+1} participates in the maximum flow. Therefore, to recover every edge in the maximum flow, we must recover all the edges between each pair of layers L_i, L_{i+1}. Notice that the number of bits needed to encode the edges between two consecutive layers is $\log_2 \binom{\varphi/4}{\varphi/8} \le \varphi$. Since there are $n/\sqrt{\varphi}$ such layers, the total number of bits needed to encode all the edges in the maximum flow is $n'\sqrt{\varphi} \ge n^{3/2}/64$.

We now use the above construction to prove a lower bound on the number of cut queries needed to recover all the edges in the maximum flow. Denote the success probability of the algorithm by $p = \Pr_{G\sim\mathcal{G}}[A = G]$ and recall that p is a constant. It is clear that recovering every edge of the maximum flow is equivalent to recovering the graph G. To prove the lower bound on the number of cut queries needed to recover G, we will use an information-theoretic argument. By Yao's minimax principle, it suffices to prove a lower bound for deterministic algorithms on a random graph G sampled from the above distribution. Let $A \in \mathcal{G}$ be the output of the deterministic cut-query algorithm after making a sequence R of q cut queries on a graph G uniformly sampled from $\mathcal{G}$. It is clear that $H(G') \le H(R) \le 2q\log n$ since each cut query returns a value in the range $[0,n^2]$. Therefore,

$$H(G \mid A) = H(G,A) - H(A) \ge H(G) - H(A) \ge n^{3/2}/64 - 2q\log n.$$

On the other hand, by Fano's inequality we can upper bound $H(G \mid A)$ as follows,

$$H(G \mid A) \le H(\text{error}) + \Pr[\text{error}]\cdot\log_2|\mathcal{G}| \le 1 + (1-p)\cdot n^{3/2}/64.$$

Combining the two we bounds we find that $q \ge \Omega(pn^{3/2}/\log n)$. □