Friendly Cut Sparsifiers and Faster Gomory-Hu Trees

Amir Abboud[*]
Weizmann Institute of Science
Israel

Robert Krauthgamer[†]
Weizmann Institute of Science
Israel

Ohad Trabelsi[‡]
University of Michigan
USA

Abstract

We devise new cut sparsifiers that are related to the classical sparsification of Nagamochi and Ibaraki [Algorithmica, 1992], which is an algorithm that, given an unweighted graph G on n nodes and a parameter k, computes a subgraph with $O(nk)$ edges that preserves all cuts of value up to k. We put forward the notion of a *friendly cut sparsifier*, which is a minor of G that preserves all friendly cuts of value up to k, where a cut in G is called *friendly* if every node has more edges connecting it to its own side of the cut than to the other side. We present an algorithm that, given a simple graph G, computes in almost-linear time a friendly cut sparsifier with $\tilde{O}(n\sqrt{k})$ edges. Using similar techniques, we also show how, given in addition a terminal set T, one can compute in almost-linear time a *terminal sparsifier*, which preserves the minimum st-cut between every pair of terminals, with $\tilde{O}(n\sqrt{k} + |T|k)$ edges.

Plugging these sparsifiers into the recent $n^{2+o(1)}$-time algorithms for constructing a Gomory-Hu tree of simple graphs, along with a relatively simple procedure for handling the unfriendly minimum cuts, we improve the running time for moderately dense graphs (e.g., with $m = n^{1.75}$ edges). In particular, assuming a linear-time Max-Flow algorithm, the new state-of-the-art for Gomory-Hu tree is the minimum between our $(m + n^{1.75})^{1+o(1)}$ and the known $mn^{1/2+o(1)}$.

We further investigate the limits of this approach and the possibility of better sparsification. Under the hypothesis that an $\tilde{O}(n)$-edge sparsifier that preserves all friendly *minimum st-cuts* can be computed efficiently, our upper bound improves to $\tilde{O}(m + n^{1.5})$ which is the best possible without breaking the cubic barrier for constructing Gomory-Hu trees in non-simple graphs.

1 Introduction

This paper is on the rich and vibrant topic of graph sparsification, where the aim is to reduce the size of the graph, typically measured by the number of edges, while preserving the graph's properties as much as possible. This notion is appealing in computer science due to the gains in efficiency, across all metrics, both in theory and in practice, that come from working with smaller objects. In particular, it is a critical step inside state-of-the-art algorithms for many fundamental problems.

One of the most influential results on sparsification, due to Nagamochi and Ibaraki [NI92], essentially says that a set of w maximally spanning forests of a graph preserves all cuts with $\leq w$ edges. Its applications for speeding up algorithms are far-reaching and include problems like minimum cut [SW97], traveling salesman [JRR95], disjoint paths [KKR12], and most importantly for this paper, cut-equivalent (aka Gomory-Hu) trees.

THEOREM 1.1. (NAGAMOCHI-IBARAKI SPARSIFICATION [NI92]) *Given an unweighted graph G on n nodes and m edges, one can compute in $O(m)$-time a subgraph with $O(nw)$ edges that preserves all cuts of value $\leq w$.*

This sparsification tool is indispensable for the recent algorithms that break the longstanding cubic barrier for the classical Gomory-Hu tree problem. Gomory and Hu [GH61] discovered that every graph G has a cut-equivalent (weighted) tree T on the same set of nodes, such that for all $s, t \in V(G)$ the minimum s, t-cut in T is also the

minimum in G. Moreover, they showed that this tree can be computed using $n-1$ calls to a Max-Flow algorithm, and a major open question since then has been to determine the time complexity of computing such a tree. The quest for faster algorithms for Gomory-Hu tree is further motivated by the fact that it is a near-optimal data structure for answering minimum s, t-cut queries [AKT20b]. Only last year, a subcubic Gomory-Hu tree algorithm for unweighted (simple) graphs was discovered by Abboud, Krauthgamer, and Trabelsi [AKT21b]. And then in three independent follow-up papers, by those same authors [AKT21a], by Li, Panigrahi, and Saranurak [LPS21], and by Zhang [Zha21], the bound was brought down from $n^{2.5+o(1)}$ to $n^{2+o(1)}$, which is clearly almost-optimal for dense graphs.[1]

At a very high level, the way that Nagamochi-Ibaraki sparsification fits into the Gomory-Hu tree algorithms is as follows. These algorithms reduce the problem to $(n/w)^{1+o(1)}$ calls to a Max-Flow algorithm on graphs where only cuts of value in the range $[w, 2w]$ are of interest, for $w = 2^0, 2^1, \ldots, 2^{\log n}$. Instead of making these calls on the input graph, which might require $n/w \cdot \Omega(n^2) = \Omega(n^3)$ time, the algorithms operate on a Nagamochi-Ibaraki sparsifier that has only $O(nw)$ edges, resulting in a time bound of $(n/w)^{1+o(1)} \cdot (nw)^{1+o(1)} = n^{2+o(1)}$, if Max-Flow is solvable in almost-linear time (and it turns out that the existing algorithms, due to [vdBLL+21], are sufficient).

We aim to investigate the possibility of better sparsification methods for the Gomory-Hu tree problem and (hopefully) beyond. While the most outstanding open question in this context is whether subcubic time is possible for *weighted* graphs, the story for unweighted (simple) graphs remains unfinished: *What is the time complexity if the number of edges m is taken into account?* The state of the art is $\min\{n^{2+o(1)}, mn^{1/2+o(1)}\}$ [LPS21], and the only lower bound other than $\Omega(m)$ is the observation in [AKT21a] that $\Omega(n^{1.5})$ time is required, even when $m = n$, unless the cubic barrier can be broken for multigraphs (which seems to be the main roadblock towards weighted graphs). This leaves a gap of up to $\sqrt{n}$ and lets one hope for the conditionally-best-possible $O(m + n^{1.5})$ bound; can we achieve it with better sparsifiers?

Better sparsification? Alas, it is easy to see that $\Omega(nw)$ is a lower bound on any sparsifier that preserves all cuts with $\leq w$ edges. And even though we are only interested in *minimum s, t-cuts* of value $\leq w$ rather than *all* cuts, the $\Omega(nw)$ barrier remains. This would even be the case if the sparsifier is allowed to contract vertices (which makes sense if there is no cut of value $\leq w$ separating them) in addition to deleting edges.

DEFINITION 1.1. (SPARSIFIERS) *A sparsifier of G is a graph H obtained from it by deleting edges and contracting subsets of the nodes. A contracted graph is a special case of a sparsifier that is obtained only by contracting subsets of the nodes (and then removing any self-loops).[2] We say that H is a w-cut sparsifier of G if it preserves all cuts with $\leq w$ edges in G, meaning that none of their edges is deleted and nodes from different sides of such a cut are not contracted together.*

We usually refer to the *size* of a sparsifier as the number of edges in it, counting parallel edges separately. For many applications it makes sense to combine parallel edges into one weighted edge and count it only once, but this will not be used in this paper.

OBSERVATION 1.1. (OPTIMALITY OF NAGAMOCHI-IBARAKI) *For every $n \geq w \geq 1$, there exists a simple graph on n nodes where every w-cut sparsifier must have $\Omega(nw)$ edges.*

Proof. For simplicity, consider $w = n$ and take a clique on n nodes. (The proof generalizes for all w by taking n/w disjoint w-cliques.) The minimum s, t-cut for any pair s, t is $\{s\}$ or $\{t\}$ and it has $n - 1 < w$ edges, so they must all be preserved. Deleting any edge $\{u, v\}$ decreases the value of the minimum u, v-cut, and contracting it increases their minimum cut (to infinity). □

Fortunately, in many contexts, cliques and other structures that prevent us from beating the $O(nw)$ bound actually make the downstream problem easy. Conceptually, our message is that better sparsification may be possible if the goal is relaxed to preserving only the "hard" cuts. In the context of Gomory-Hu tree (and perhaps elsewhere), the hard cuts are friendly cuts, as discussed next, and our main point is that these can be preserved using $o(nw)$ edges.

---

[1] Zhang's algorithm has better running time $\tilde{O}(n^2)$, but requires a (hypothetical) $\tilde{O}(m)$-time Max-Flow algorithm.

[2] Recall that a *minor* is obtained by deleting edges and contracting *connected* subsets of nodes. This distinction is not relevant to our results, which extend immediately to minors. Indeed, the number of edges in our sparsifiers would not change if a contraction of a set S would be replaced by contracting every connected subset of S.

1.1 Friendly Cut Sparsifiers Given a cut, a node is called friendly if it has more neighbors to its own side than to the other side of the cut. The definition can be generalized so that α-friendly means that more than(or at least) an α-fraction of the neighbors are on the same side, for α that is not necessarily $1/2$. The cut is then called α-friendly if all nodes are α-friendly. Throughout this paper, we use $\alpha = 0.4$ for simplicity; our sparsification results will actually hold for any α but the application to Gomory-Hu tree algorithms will require a fixed $\alpha < 0.5$. Thus, for a cut to be unfriendly, there needs to be a node with $> 60\%$ of its edges going to the other side. Throughout, for a node v in a graph G we denote by $\deg_G(v)$ the total degree of v in G counting multiple edges accordingly, and omitting the subscript when it is clear from the context.

DEFINITION 1.2. (FRIENDLY AND UNFRIENDLY CUTS) *A cut S in a graph $G = (V, E)$ is called* unfriendly *if there exists a node $s \in S$ such that $|E(\{s\}, V \setminus S)| > 0.6\deg(s)$ or there exists a node $t \notin S$ such that $|E(\{t\}, S)| > 0.6\deg(t)$. Otherwise, the cut is called* friendly.

Questions involving friendly cuts are extensively studied in combinatorics (see [BL16, FKN$^+$21] and references therein). They actually arise in various contexts, including social learning, set theory, and statistical physics, and under different names, such as *internal partitions* [BL16], *satisfactory partitions* [GK00, BTV06, GMT20], and *q-cohesive sets* [Mor00].

Unfriendly cuts can be viewed as a generalization of the *trivial cuts* $(\{v\}, V \setminus \{v\})$, also known as *singleton-* or *degree-cuts*, that happen to be the minimum s, t-cuts in a clique. Observe that a trivial cut is not α-friendly for any $\alpha > 0$. So perhaps the lower bound of Observation 1.1 can be bypassed if one only wishes to preserve the friendly cuts?

DEFINITION 1.3. (FRIENDLY CUT SPARSIFIERS) *A friendly w-cut sparsifier of G is a graph H that preserves all the friendly cuts with $\leq w$ edges in G.*

Indeed, the main result of this paper is a *friendly cut sparsifier* giving a polynomial improvement over the Nagamochi-Ibaraki bound. The proof is in Section 2.

THEOREM 1.2. (FRIENDLY CUT SPARSIFIERS) *There is a randomized $m^{1+o(1)}$-time algorithm that, given a simple graph on n nodes and m edges and a parameter w, produces a friendly w-cut sparsifier with $m' = O(n\sqrt{w}\log^4 n)$ edges. There is also a deterministic $m^{1+o(1)}$-time algorithm achieving $m' = (n\sqrt{w})^{1+o(1)}$. Furthermore, the sparsifier is a contracted graph (and a minor).*

Technically, the result is very different from Nagamochi and Ibaraki's, and is based on an expander decomposition of the graph rather than on removing spanning forests. Perhaps the most similar result of this kind is by Kawarabayashi and Thorup [KT19], who study deterministic algorithms for edge connectivity. They show that all non-trivial minimum cuts in a simple graph of minimum degree δ can be preserved by a contracted graph on $\tilde{O}(m/\delta)$ edges. Our context is different because we do not have a lower bound on the minimum degree δ and we want to preserve cuts whose values is not necessarily close to the minimum; and for this reason we can only preserve cuts that are friendly rather than merely non-trivial.

Before proceeding to discuss the limits of friendly cuts sparsifiers in Section 1.3 let us motivate them further by presenting our main application.

1.2 Application: Faster Gomory-Hu tree The question whether the time bound $\min\{n^{2+o(1)}, mn^{1/2+o(1)}\}$ can be improved is particularly interesting from the perspective of *fine-grained complexity* when comparing constructing a Gomory-Hu tree to two other basic graph problems : triangle and 4-cycle detection. In the regime of moderately dense graphs where $m = n^{1.5}$, it is often conjectured that the quadratic barrier cannot be broken for these problems [YZ97, AW14, DG19] (and reporting triangles in subquadratic time is 3-SUM hard [Pat10, KPP16, BPWZ14]). An application of our friendly cut sparsifiers is a (conditional) separation: Gomory-Hu tree is subquadratic in this regime.

THEOREM 1.3. (FASTER GOMORY-HU TREE) *There is a randomized algorithm that constructs a Gomory-Hu Tree of a simple graph on n nodes and m edges in time $(m + n^{1.9})^{1+o(1)}$. Assuming an almost-linear time* Max-Flow *algorithm for undirected graphs with polynomially bounded edge weights, the running time becomes $(m + n^{1.75})^{1+o(1)}$.*

This theorem is achieved by taking the recent $n^{2+o(1)}$-time algorithms [LPS21, AKT21a, Zha21] and replacing the Nagamochi-Ibaraki w-cut sparsifier with our new friendly w-cut sparsifier from Theorem 1.2. Notably, expander-decompositions are now used *twice* in the state-of-the-art algorithms: once for sparsification and once for reducing the number of queries.

But why is it enough to preserve the friendly cuts? It is because there is an alternative, more efficient (and simpler) way to compute the unfriendly minimum s,t-cuts. This is stated in the next algorithm, which relies on the recent ISOLATING-CUTS procedure [LP20, AKT21b] and works even for weighted graphs. It solves a single-source version of Gomory-Hu tree, which is known to be essentially as hard as Gomory-Hu tree [AKT20b, LP21].

THEOREM 1.4. (SINGLE-SOURCE UNFRIENDLY MINIMUM-CUTS) *There is an algorithm that, given an undirected graph G on n nodes and m edges with polynomially bounded weights, and a source node $p \in V$, outputs for every $v \in V(G) \setminus \{p\}$ a p,v-cut S_v such that: if there is a minimum p,v-cut that is unfriendly then the output S_v is a minimum p,v-cut. The running time is $\tilde{O}(m)$ plus the time of $\mathrm{poly}\log n$ calls to a Max-Flow algorithm on $O(n)$-node $O(m)$-edge graphs.*

This algorithm for unfriendly cuts is given in Section 4. A technical overview of the issues that arise when plugging the friendly sparsifier and this algorithm into the recent Gomory-Hu tree algorithms is given in Section 5, and the actual details for proving Theorem 1.3 are in Section 5.

Could this approach lead us all the way to the conditionally optimal $m + n^{1.5}$ bound? At a high-level, the $n^{1.75}$ upper bound follows from the following calculation. Each of the n/w calls to Max-Flow in the recent algorithms [LPS21, AKT21a, Zha21] is performed on a sparsifier with $n\sqrt{w}$ edges, making the time bound $n^2/\sqrt{w}$. An alternative method (better for small w) is to use a w-partial tree [BHKP07] on the sparsifier, which has time bound $nw^{1.5}$. Thus the worst-case of $\min\{n^2/\sqrt{w}, nw^{1.5}\}$ over all possible w is $n^{1.75}$. So all we need is a friendly w-cut sparsifier on $\tilde{O}(n)$ edges, for all w, as that would lead to a bound $\min\{n^2/w, nw\} = O(n^{1.5})$. Is that possible?

1.3 The Limits of Friendly Sparsification Unfortunately, it is not hard to see that $n\sqrt{w}$ is also a lower bound for friendly w-cut sparsifiers.

OBSERVATION 1.2. (OPTIMALITY OF OUR BOUND) *For every $n \geq w \geq 1$, there exists a simple graph on n nodes such that every friendly w-cut sparsifier must have $\Omega(n\sqrt{w})$ edges.*

Proof. For simplicity, we consider only $w = n$ (the proof generalizes easily). Take a clique on n nodes and replace each node with a $10\sqrt{n}$-clique, connecting the original $n - 1$ edges incident at a node, to arbitrary nodes in its $10\sqrt{n}$-clique, but such that no new node gets more than $\sqrt{n}$ original edges. The new graph has $n^{1.5}$ nodes and all of the $\Omega(n^2)$ edges of the original clique must remain in every friendly w-cut sparsifier: Each trivial cut of the original clique corresponds to a cut with $n - 1$ edges that is friendly in the new graph, because each node has $10\sqrt{n} - 1$ edges to its side but only $\sqrt{n}$ edges across. □

However, the above proof leaves a bit of hope: the lower bound is because of friendly cuts that are not *minimum s,t-cuts* for any pair s,t. Indeed, all minimum s,t-cuts in this construction are trivial with $\leq 11\sqrt{n}$ edges. So a friendly *minimum s,t-cut* sparsifier could contract the entire graph.

DEFINITION 1.4. (FRIENDLY MINIMUM s,t-CUT SPARSIFIER) *We say that H is a* friendly minimum s,t-cut sparsifier *for G if for every pair s,t for which all the minimum s,t-cuts in G are friendly, at least one minimum s,t-cut is preserved in H.*

For the Gomory-Hu tree application, and perhaps others, this is all we need. Perhaps surprisingly, it turns out that the desired $\tilde{O}(n)$ bound can indeed be achieved, albeit we do not know how to achieve it in time that is faster than computing a Gomory-Hu tree. We prove the following theorem in Section 3, using a structural analysis of the Gomory-Hu tree when the graph is simple.

THEOREM 1.5. (NEAR-LINEAR UPPER BOUND) *Every simple graph on n nodes has a friendly minimum s,t-cut sparsifier with $O(n \log n)$ edges. Moreover, it can be computed in linear time from a Gomory-Hu tree of the graph.*

We have thus reached a computational *equivalence* between Gomory-Hu tree and friendly sparsification: faster algorithms exist for one if and only if they exist for the other.

THEOREM 1.6. (**COMPUTATIONAL EQUIVALENCE**) *Let* $T(n,m) = \Omega(m + n^{1.5})$. *An* $O(n \log n)$-*edge friendly minimum* s,t-*cut sparsifier of a simple graph can be computed in time* $T(n,m) \cdot n^{o(1)}$ *if and only if a GHT can be computed in time* $T(n,m) \cdot n^{o(1)}$.

In other words, the only remaining question in the context of Gomory-Hu tree algorithms for simple graphs (without breaking the cubic barrier for multigraphs) is whether an $\tilde{O}(n)$-edge friendly minimum s,t-cut sparsifier can be computed in $(m + n^{1.5})^{1+o(1)}$ time.

Finally, we ask whether the $\tilde{O}(n)$ upper bound can be obtained for (non-simple) multigraphs, and discover that there is an $\Omega(n^2)$ lower bound.

OBSERVATION 1.3. *For every* n, *there exists a multigraph on* n *nodes and* $O(n^2)$ *edges, for which every friendly minimum* s,t-*cut sparsifier has* $\Omega(n^2)$ *edges.*

Proof. Take a cycle on n nodes $v_1, \ldots, v_n$ where the weight of each edge (i.e. the number of parallel edges between the pair) is defined in the following alternating manner. Set $w(v_i, v_{i+1}) = \varepsilon n$ if i is even, and set $w(v_i, v_{i+1}) = n$ if i is odd, and finally set $w(v_1, v_n) = \varepsilon n - 1$. (The parameter ε can be chosen based on the friendliness parameter, in our case $\varepsilon = 0.4$ suffices.) The (only) minimum v_i, v_{i+1}-cut, when i is even, is the *friendly* cut $(\{v_1, \ldots, v_i\}, \{v_{i+1}, \ldots, v_n\})$. Any sparsifier must preserve all the edges of these $n/2$ cuts, whose total number of edges is $\geq n/2 \cdot \varepsilon n = \Omega(n^2)$. ☐

Therefore, a different notion of sparsification seems to be required for breaking the cubic barrier for weighted graphs. Perhaps *terminal sparsification*, discussed next.

1.4 Terminal Sparsification The techniques of this paper also lead to new *terminal* minimum s,t-cut sparsifiers.

DEFINITION 1.5. (**TERMINAL MINIMUM** s,t-**CUT** w-**SPARSIFIER**) *We say that* H *is a* terminal minimum s,t-cut w-sparsifier *of a graph* G *and terminal set* T *if it preserves all cuts with* $\leq w$ *edges that are a minimum* s,t-*cut for some pair of terminals* $s, t \in T$ *in* G.

An ideal analogue of Nagamochi-Ibaraki sparsification would be a terminal minimum s,t-cut sparsifier with $O(|T|w)$ edges. It is not hard to see that such a bound is existentially possible, even for multigraphs, and that it can be constructed from a Gomory-Hu tree.[3] We show that in almost-linear time we can get a bound that is worse by an additive $+n\sqrt{w}$ term; improving on the $O(nw)$ Nagamochi-Ibaraki bound. The proof is in Section 6.

THEOREM 1.7. *There is an* $m^{1+o(1)}$ *time algorithm that, given a simple graph on* n *nodes,* m *edges, and terminal set* T, *computes a terminal minimum* s,t-*cut* w-*sparsifier on* $(n\sqrt{w} + |T|w)^{1+o(1)}$ *edges.*

We remark that it would have been possible to achieve our $(n^{1.75} + m)^{1+o(1)}$ upper bound for Gomory-Hu tree using this terminal sparsifier rather than the friendly sparsifier, although in a less elegant way (in our view). Therefore, we expect the equivalence of Theorem 1.6 to be extendible to terminal sparsification with $\tilde{O}(|T|w)$ edges as well. But can the $O(nw)$ bound be beaten algorithmically for *multigraphs*?

1.5 Related Work We consider graph sparsification that preserves (certain) cut values. This topic has been extremely influential, and perhaps the first such result is the work of Gomory and Hu [GH61], because a Gomory-Hu tree is just a sparsifier for all minimum st-cuts in the graph. This line of research has generated several other influential notions, including for cut values up to a threshold [NI92], for directed rooted connectivity [Gab95], for minimum cuts between subsets of terminals [HKNR98], for global minimum cut [Kar99], and for all cuts up to some approximation [BK15].

---

[3]Take the terminal Gomory-Hu tree, which is a cut-equivalent tree on super-nodes where each super-node contains exactly one terminal. Contract each super-node. Sparsify the resulting $|T|$-node graph with Nagamochi-Ibaraki sparsification to get $O(|T|k)$ edges.

More broadly, graph sparsification covers many other useful graph quantities, including directed reachability [AGU72], shortest-path distance [PS89, KNZ14, Gup01], resistance distance (i.e., effective resistance) [DKW15], potential energy of the Laplacian (called spectral sparsification) [ST11], and multicommodity flows (including routings) [RÖ2, Moi09]. There are some connections between these quantities, e.g., spectral sparsification always preserves all the resistances and also all the cuts.

The literature usually makes a distinction between edge sparsification (which decreases the number of edges by taking a subgraph, possibly with reweighted edges), and vertex sparsification (which decreases the number of vertices by merging or removing vertices, which in some cases produces a minor). In the latter context, the input graph usually comes with a set of terminals, whose properties (distances, cuts, etc.) must be preserved.

Graph sparsification (for cuts and in general) has many downstream applications. The original motivation for many of the abovementioned results is to speed up algorithms. Other uses include reducing storage and/or communication requirement in streaming and distributed settings, or to improve the approximation factor (to depend on the number of terminals). For a more empirical perspective, which addresses a range of objectives under the names graph summarization and graph coarsening, see the survey [LSDK18].

2 Friendly Sparsification

This section proves Theorem 1.2, the main sparsification result of this paper. The main workhorse of our construction is an efficient procedure that decomposes a graph into (node-disjoint) expanders, such that the number of edges between these expanders is small. We thus start with describing the relevant definitions and known algorithms in Section 2.1. We then present our deterministic algorithm, which is simpler, in Section 2.2. At a high level, its idea is very simple – compute an expander decomposition and then contract the nodes of each expander. The intuition is that it is safe to contract the nodes of an expander, because it should not be "split" by the low-weight cuts we are interested in. However, the actual procedure must be refined by "shaving" some nodes from each expander before contracting it. Moreover, we need a stronger version of the expander decomposition, that can handle demands.

Our randomized algorithm, which is faster and computes a slightly smaller sparsifier, is presented in Section 2.3. It is based on the same techniques but uses a more elementary version of expander decomposition (without demands). This limitation forces us to work in iterations that sparsify the graph gradually, but the advantage is that this version admits a (randomized) algorithm that is faster, replacing $n^{o(1)}$ terms with polylogarithmic factors.

2.1 Preliminaries: Expander Decomposition We mostly follow notations and definition from [SW19]. Let $G = (V, E)$ be an undirected graph with edge capacities. Define the *volume* of $C \subseteq V$ as $\mathrm{vol}_G(C) := \sum_{v \in C} \deg_G(v)$, where the subscripts referring to the graph are omitted if clear from the context. The *conductance* of a cut S in G is $\Phi_G(S) := \frac{\delta(S)}{\min(\mathrm{vol}_G(S), \mathrm{vol}_G(V \setminus S))}$. The *expansion* of a graph G is $\Phi_G := \min_{S \subset V} \Phi_G(S)$. If G is a singleton then $\Phi_G := 1$ by convention. Let $G[S]$ be the subgraph induced by $S \subset V$, and let $G\{S\}$ denote the induced subgraph $G[S]$ but with an added self-loop $e = (v, v)$ for each edge $e' = \{v, u\}$ where $v \in S, u \notin S$ (where each self-loop contributes 1 to the degree of a node), so that every node in S has the same degree as its degree in G. Observe that for all $S \subset V$, $\Phi_{G[S]} \geq \Phi_{G\{S\}}$, because the self-loops increase the volumes but not the values of cuts. We say that a graph G is a ϕ-*expander* if $\Phi_G \geq \phi$, and we call a partition $V = V_1 \sqcup \cdots \sqcup V_h$ a ϕ-expander-decomposition if $\min_i \Phi_{G[V_i]} \geq \phi$.

THEOREM 2.1. (THEOREM 1.2 IN [SW19]) *Given a graph $G = (V, E)$ of m edges and a parameter ϕ, one can compute with high probability a partition $V = V_1 \sqcup \cdots \sqcup V_h$ such that $\min_i \Phi_{G[V_i]} \geq \phi$ and $\sum_i \delta(V_i) = O(\phi m \log^3 m)$. In fact, the algorithm has a stronger guarantee that $\min_i \Phi_{G\{V_i\}} \geq \phi$. The running time of the algorithm is $O(m \log^4 m / \phi)$.*

For our deterministic upper bound we will need a deterministic version of Theorem 2.1, which exists, albeit with worse bounds [CGL+20]. If one is paying the extra bounds anyway, it is possible to introduce additional power to the decomposition by introducing *demands on the nodes* (to be used instead of the degrees when computing the volume) which greatly simplifies the proof that uses them (as a black box). Suppose that we are also given a *demand vector* $\mathbf{d} \in \mathbb{R}_{\geq 0}^V$, the graph $G = (V, E)$ is a $(\phi, \mathbf{d})$-*expander* if for all subsets $S \subseteq V$, $\Phi_G^{\mathbf{d}}(S) := \frac{\delta(S)}{\min(\mathbf{d}(S), \mathbf{d}(V \setminus S))} \geq \phi$. The following theorem statement from [LP20, Theorem III.8] gives

a deterministic algorithm. It is actually proved in [LS21, Corollary 2.5], by extending techniques from [CGL$^+$20].

THEOREM 2.2. ([LS21]) *Fix $\varepsilon > 0$ and any parameter $\phi > 0$. Given an edge-weighted, undirected graph $G = (V, E, w)$ and a demand vector $\mathbf{d} \in R_{\geq 0}^V$, there is a deterministic algorithm running in time $O(m^{1+\varepsilon})$ that computes a partition $V = V_1 \sqcup \cdots \sqcup V_h$ such that*

1. *For each $i \in [h]$, define a demand vector $\mathbf{d}_i \in \mathbb{R}_{\geq 0}^{V_i}$ given by $\mathbf{d}_i(v) = \mathbf{d}(v) + w(E(v, V \setminus V_i))$ for all $v \in V_i$. Then, the graph $G[V_i]$ is a $(\phi, \mathbf{d}_i)$-expander.*

2. *The total weight of inter-cluster edges is $w(E(V_1, \ldots, V_h)) = \sum_i w(E(V_i, V \setminus V_i)) \leq B \cdot \phi \mathbf{d}(V)$ for $B = (\log n)^{O(1/\varepsilon^4)}$.*

2.2 Deterministic Algorithm

Proof. [Proof of Theorem 1.2 (deterministic algorithm)] Given a simple graph $G = (V, E)$ and a parameter w, the algorithm first computes an expander decomposition of G into expanders $H_1, \ldots, H_\ell$, using Theorem 2.2 with parameter $\phi = 2^{-\log^{1/2} n} = n^{-o(1)}$ and demand function $\mathbf{d}(v) = \phi^{-1}\sqrt{w}$ for all $v \in V$. By setting the parameter $\varepsilon = (\log n)^{-1/9} = o(1)$, the running time is $m^{1+\varepsilon}$ and the outer-edges depend on $B = (\log n)^{O(1/\varepsilon^4)} \leq O(\phi^{-1}/\log n) \leq n^{o(1)}$. Then each expander H_i is a $(\phi, \mathbf{d}_i)$-expander where $\mathbf{d}_i(v) = \mathbf{d}(v) + |E(v, V \setminus H_i)|$ for all $v \in H_i$. And since the total demand is $\mathbf{d}(V) = n \cdot \phi^{-1}\sqrt{w}$, the total number of outer-edges is

$$m_0 := \sum_i |E(H_i, V \setminus H_i)| \leq B \cdot \phi \mathbf{d}(V) = Bn\sqrt{w} \leq n^{1+o(1)}\sqrt{w}.$$

Second, the algorithm computes for each expander H_i its *shaved expander* H_i', obtained by removing from H_i (simultaneously) all nodes $v \in H_i$ that satisfy at least one of these two conditions:

- it has a low degree $\deg_G(v) < 10\sqrt{w}$; or
- more than 10% of its degree goes outside of the expander, i.e., $|E(\{v\}, V \setminus H_i)| > 0.1 \deg_G(v)$.

Finally, the algorithm contracts every shaved expander H_i', and return the resulting contracted graph G'.

The running time is dominated by $m^{1+o(1)}$ the complexity of the expander decomposition procedure; the other operations take linear time. The following two claims conclude the proof.

CLAIM 2.1. *Let $S \subseteq V$ be a friendly cut in G that has weight $\delta(S) \leq w$. Then G' preserves this cut S (i.e., never contracts two nodes that are on different sides).*

Proof. Assume for contradiction that two nodes $x \in S, y \notin S$ are contracted into the same node in G', i.e., they belong to the same shaved expander H_i'. Consider the projection of the cut S on the expander H_i, given by $L := H_i \cap S$ and $R := H_i \cap (V \setminus S) = H_i \setminus L$, which are both non-empty because $x \in L$ and $y \in R$. Now since H_i is a $(\phi, \mathbf{d}_i)$-expander, we must have $\frac{|E(L,R)|}{\min\{\mathbf{d}_i(L), \mathbf{d}_i(R)\}} \geq \phi$. Clearly, $|E(L, R)| \leq |E(S, V \setminus S)| \leq w$, and thus

$$\min\{\mathbf{d}(L), \mathbf{d}(R)\} \leq \min\{\mathbf{d}_i(L), \mathbf{d}_i(R)\} \leq \frac{|E(L, R)|}{\phi} \leq \phi^{-1}w.$$

We will now show a lower bound on $\mathbf{d}(L)$. Since the cut S is friendly, $|E(\{x\}, S)| \geq 0.4 \deg_G(x)$. Moreover, since x was not shaved, we know that $\deg_G(x) \geq 10\sqrt{w}$ and that at most 10% of its degree goes outside of the expander, i.e., $|E(\{x\}, V \setminus H_i)| \leq 0.1 \deg_G(x)$. Combining these three inequalities,

$$|E(\{x\}, L)| = |E(\{x\}, S \cap H_i)| \geq (0.4 - 0.1) \deg_G(x) \geq 3\sqrt{w}.$$

Relying on the key property that the graph is simple, the number of *nodes* in this set L must be $|L| \geq 3\sqrt{w}$. By the definition of our demand function, all nodes $v \in V$ have $\mathbf{d}(v) = \phi^{-1}\sqrt{w}$, and therefore $\mathbf{d}(L) = \phi^{-1}\sqrt{w} \cdot |L| \geq 3\phi^{-1}w$. The argument for $\mathbf{d}(R)$ is symmetric, and we arrive at $\min\{\mathbf{d}(L), \mathbf{d}(R)\} \geq 3\phi^{-1}w$, in contradiction to the above. $\square$

CLAIM 2.2. *The number of edges in G' is $n^{1+o(1)}\sqrt{w}$.*

Proof. G' has three kinds of edges, all originating from G:

1. The outer-edges of the expander decomposition. Their number is $m_0 \leq Bn\sqrt{w} \leq n^{1+o(1)}\sqrt{w}$.

2. Edges incident to nodes that were shaved due to having low degree. Their number can be upper bounded by $n \cdot 10\sqrt{w} = O(n\sqrt{w})$.

3. Edges incident to nodes that were shaved due to having more than 10% of their degree going outside their expander. We upper bound the number of these edges by $20m_0$ as follows. Let $X \subseteq V_{j-1}$ be the set of these shaved nodes, and let $d_{out}(v)$ be the number of edges from v to nodes outside its expander. Observe that $m_0 = 1/2 \cdot \sum_{v \in V} d_{out}(v)$, and by definition, every $v \in X$ satisfies $d_{out}(v) > 0.1 \deg_G(v)$. Altogether,

$$\sum_{v \in X} \deg_G(v) \leq \sum_{v \in X} 10 \cdot d_{out}(v) \leq 10 \sum_{v \in V} d_{out}(v) = 20m_0.$$

Altogether, the total number of edges is $O(m_0 + n\sqrt{w}) \leq n^{1+o(1)}\sqrt{w}$, proving the claim. □

This completes the proof of the deterministic algorithm in Theorem 1.2. □

The more efficient bound with polylogarithmic factors instead of $n^{o(1)}$, using randomized expander-decomposition algorithms, is given next.

2.3 Randomized Algorithm Here, we prove the improved bound in Theorem 1.2 that uses a randomized expander-decomposition algorithm. The arguments are similar to those in the deterministic algorithm section above, but they are applied in a recursive manner.

Proof. [Proof of Theorem 1.2 (randomized algorithm)] Given a simple graph $G = (V, E)$ and a parameter w, the algorithm proceeds in iterations, where each iteration $j = 0, 1, 2, \ldots$ produces a friendly w_j-cut sparsifier G_j of G with $m_j \leq Kn\sqrt{w_j}$ edges, for $w_j = 4^{-j}(m/n)^2$ and $K = \log^{O(1)} n$ to be determined later. The iterations continue as long as $w_j \geq w$, hence the last iteration j satisfies $w_j < 4w$ and its sparsifier G_j is reported as the final sparsifier. Iteration 0 produces $G_0 = G$ itself which is trivially a sparsifier.

Each iteration $j \geq 1$ uses the sparsifier $G_{j-1} = (V_{j-1}, E_{j-1})$ produced in the previous iteration, as follows. The first step (of iteration j) uses Theorem 2.1 with parameter $\phi = 0.01/B$, where $B = O(\log^3 n)$ denotes the factor in its out-edges gurantee. It is used to decompose the graph G_{j-1} into expanders $H_1, \ldots, H_\ell$, but only after adding $\phi^{-1}\sqrt{w_j}$ self-loops to each node in G_{j-1}, to increase its volume (but not counting them in the degree).

Then each expander H_i is a ϕ-expander, and the total number of outer-edges is

$$m'_j := \sum_{i=1}^{\ell} |E_{G_{j-1}[V']}(H_i, V' \setminus H_i)| \leq B \cdot \phi(m_{j-1} + n\phi^{-1}\sqrt{w_j}).$$

The second step (of iteration j) is to compute for each expander H_i its *shaved expander* H'_i, obtained by removing from H_i (simultaneously) every node $v \in H_i$ that satisfies at least one of these two conditions:

- it has a low degree $\deg_{G_{j-1}}(v) < 10\sqrt{w_j} \cdot \text{size}_G(v)$; or
- more than 10% of its degree goes outside of the expander, i.e., $|E_{G_{j-1}}(\{v\}, V_{j-1} \setminus H_i)| > 0.1 \deg_{G_{j-1}}(v)$.

The third and final step (of iteration j) produces the sparsifier G_j by contracting in the sparsifier G_{j-1} every shaved expander H'_i, and removing self-loops (but keeping parallel edges). The resulting G_j is a contracted graph of G, because a node in a shaved expander is itself a contraction of nodes from V.

The running time of each iteration j is dominated by the complexity of the expander decomposition procedure, which is $O(m_{j-1} \cdot B\phi^{-1} \log n) = O(m_{j-1}B^2 \log n)$; the other operations take linear time. The number of iterations is $O(\log \frac{(m/n)^2}{w}) = O(\log n)$, and in fact $\sum_j m_{j-1} = O(m)$, and therefore the overall running time is $\tilde{O}(m)$.

The correctness is proved in the following two claims.

CLAIM 2.3. *Let $S \subseteq V$ be a friendly cut in G that has weight $\delta_G(S) \leq w$. Then every G_j preserves this cut S (i.e., never contracts two nodes that are on different sides).*

Proof. We prove this by induction on j. The case $j = 0$ holds trivially, because $G_0 = G$. Consider then $j \geq 1$, and observe that $\delta_G(S) \leq w \leq w_j$. By the induction hypothesis, S is preserved in G_{j-1}; hence, every node in V_{j-1} is the contraction of nodes either only from S or nodes only from $V \setminus S$. With a slight abuse of notation, we can thus think of S as a subset of V_{j-1}, and thus also a cut in G_{j-1} with $\delta_G(S) \leq w$ edges. Assume for contradiction that two nodes $x \in S, y \notin S$ are contracted into the same node in G_j. This must happen because some shaved expander H_i' contracts them together, more precisely, x is in a contracted node $\bar{x} \in H_i'$, and y is in a contracted node $\bar{y} \in H_i'$. Consider now the projection of the cut S on the expander H_i, given by

$$L := H_i \cap S \quad \text{and} \quad R := H_i \setminus S = H_i \setminus L,$$

which are both non-empty because $\bar{x} \in L$ and $\bar{y} \in R$. Now since H_i is a ϕ-expander, we must have $\frac{|E_{G_{j-1}}(L,R)|}{\min\{\text{vol}(L),\text{vol}(R)\}} \geq \phi$. Clearly, $|E_{G_{j-1}}(L,R)| \leq |\delta_G(S)| \leq w$, and thus

$$\min\{\text{vol}(L), \text{vol}(R)\} \leq \frac{|E_{G_{j-1}}(L,R)|}{\phi} \leq \phi^{-1} w.$$

We now show a lower bound on $\text{vol}(L)$, and a symmetric argument applies to $\text{vol}(R)$. Since the cut S is friendly, $|E_G(\{x\}, S)| \geq 0.4 \deg_G(x)$. This inequality in fact holds for every node $x' \in V$ in the same contracted node $\bar{x}$, and summing all these inequalities we get

$$|E_{G_{j-1}}(\{\bar{x}\}, S)| = \sum_{x' \in \bar{x}} |E_G(\{x'\}, S)| \geq \sum_{x' \in \bar{x}} 0.4 \deg_G(x') = 0.4 \deg_{G_{j-1}}(\bar{x}).$$

In addition, since $\bar{x}$ was not shaved, we know that at most 10% of its degree goes outside of the expander, i.e.,

$$|E_{G_{j-1}}(\{\bar{x}\}, V_{j-1} \setminus H_i)| \leq 0.1 \deg_{G_{j-1}}(\bar{x}),$$

and that its degree is

$$\deg_{G_{j-1}}(\bar{x}) \geq 10\sqrt{w_j} \cdot \text{size}_G(\bar{x}).$$

Combining the last three inequalities and recalling that $L = S \cap H_i$,

$$|E_{G_{j-1}}(\{\bar{x}\}, L)| \geq (0.4 - 0.1) \deg_{G_{j-1}}(x) \geq 3\sqrt{w_j} \cdot \text{size}_G(\bar{x}).$$

Relying on the key property that G_{j-1} is obtained from a simple G (by contractions), we also have $|E_{G_{j-1}}(\{\bar{y}\}, L)| \leq \text{size}_G(\bar{x}) \cdot |L|$, and altogether $|L| \geq 3\sqrt{w_j}$. Moreover, all nodes $v \in L$ have, because of the added self-loops, $\text{vol}(v) \geq \phi^{-1}\sqrt{w_j}$, and therefore $\text{vol}(L) \geq |L| \cdot \phi^{-1}\sqrt{w_j} \geq 3\phi^{-1}w_j$. By a symmetric argument for $\text{vol}(R)$, we arrive at the contradiction that $\min\{\text{vol}(L), \text{vol}(R)\} \geq 3\phi^{-1}w$. $\square$

CLAIM 2.4. *The number of edges in G_j is at most $Kn\sqrt{w_j}$.*

Proof. Each sparsifier G_j has three kinds of edges, all originating from G_{j-1}:

1. The outer-edges of the expander decomposition. Their number is $m_j' \leq B \cdot \phi(m_{j-1} + n\phi^{-1}\sqrt{w_j})$.

2. Edges incident to nodes that are shaved due to having low degree. Their number can be upper bounded by $\sum_{v \in V_{j-1}} 10\sqrt{w_j} \cdot \text{size}_G(\bar{y}) \leq 10n\sqrt{w_j}$.

3. Edges incident to nodes that are shaved since more than 10% of their degree goes outside their expander. We upper bound the number of these edges by $20m_j'$ as follows. Let $X \subseteq V_{j-1}$ be the set of these shaved nodes, and let $d_{out}(v)$ be the number of edges (in G_{j-1}) from $v \in V_{j-1}$ to nodes outside its expander. Observe that $m_j' = 1/2 \cdot \sum_{v \in V_{j-1}} d_{out}(v)$, and by definition every $v \in X$ satisfies $d_{out}(v) > 0.1 \deg_{G_{j-1}}(v)$. Altogether,

$$\sum_{v \in X} \deg_{G_{j-1}}(v) \leq \sum_{v \in X} 10 \cdot d_{out}(v) \leq 10 \sum_{v \in V} d_{out}(v) = 20m_j'.$$

Altogether, the total number of edges in G_j is

$$m_j \leq 21m'_j + 10n\sqrt{w_j} \leq 21B\phi m_{j-1} + (21B + 10)n\sqrt{w_j}.$$

Using the induction hypothesis and plugging our parameters choice $\phi = 0.01/B$ and $K = 100B$, we get $B\phi m_{j-1} \leq 0.01Kn\sqrt{w_{j-1}} = 0.02Kn\sqrt{w_{j-1}}$ and $(21B + 10) \leq 0.5K$, and the claim follows. □

This completes the proof of the randomized algorithm in Theorem 1.2, because the for the final G_j is a friendly w_j-cut sparsifier for $w \leq w_j < 4w$. □

3 An $O(n \log n)$-Size Sparsifier for Friendly Minimum Cuts

In this section we show that in order to preserve friendly *minimum* cuts, the total number of edges in the most succinct sparsifier is always at most $\tilde{O}(n)$, proving Theorem 1.5 and the reduction from sparsification to Gomory-Hu tree in the equivalence of Theorem 1.6. Unlike the upper bounds of the previous section, we do not know of a fast algorithm for computing these more efficient sparsifiers, unless we are given a Gomory-Hu tree of the graph. Recall that a *friendly minimum s,t-cut sparsifier* (Definition 1.4 in Section 1) is only required to preserve a minimum s,t-cut for pairs that do not have any unfriendly minimum s,t-cut.

Proof. [Proof of Theorem 1.5] Let T be a Gomory-Hu tree of G. Call an edge of T that correspond to an unfriendly minimum cut an *unfriendly edge*. Define the sparsifier H to be G after contracting the nodes of each connected component in T comprised only of unfriendly edges. The contracted graph H satisfies the requirement for a friendly minimum s,t-cut sparsifier because for any pair u,v, such that the only minimum u,v-cut is friendly, at least one minimum u,v-cut (the one in T) has survived the contractions.

Next, we show that $|E(H)| \leq O(n \log n)$; observe that $|E(H)|$ is at most the total weight of friendly edges in T. Indeed, each edge in H (where parallel edges are counted separately) contributes to the weight of at least one friendly edge in the Gomory-Hu tree. Root T by an arbitrary node r. We first show that the total weight of friendly edges on any path in T from a node to an ancestor of it on the tree is bounded, up to constant factors, by the total number of nodes of the tree that descend from this path (that is, below it in the rooted tree). This lets us "charge" a path of weight w to $\Omega(w)$ descendants. Then, we will decompose T into paths (via a light/heavy decomposition) such that a node is a descendant of $O(\log n)$ paths, giving an upper bound of $O(n \log n)$ on the total weight.

Let P be any path in T, and let e_1 be a friendly edge on this path, and consider the subpath of P that contains the next 100 friendly edges $e_2, \ldots, e_{101}$ below e_1. Our goal is to bound $w(e_1)$, up to constant factors, by the total number of nodes that are *private* descendants of this subpath, in a sense that will become clear. Suppose that there indeed are 100 friendly edges in P below e_1; otherwise, we will handle e_1 by a different, simpler argument. In between each pair of friendly edges e_i, e_{i+1} there could be a subpath of unfriendly edges. Denote the endpoints of the friendly edges by $e_1 = (v'_1, v_1), \ldots, e_{101} = (v'_{101}, v_{101})$ and note that v_i is equal to v'_{i+1} if and only if there are no unfriendly edges between e_i and e_{i+1}. Define the private set of node v_i, denoted private(v_i), to be the descendants of all nodes on the subpath of P that are between v_i and v'_{i+1}, except for the descendants of v_{i+1}.[4] This partitions the descendants of our subpath so that each descendant is assigned to one of the nodes v_i. The key claim is the following; it is reminiscent of [AKT21a, Claim 3.3] but uses the friendliness of cuts rather than their w-largeness and non-easiness.

CLAIM 3.1. *There exists an $1 \leq i \leq 100$ such that* private(v_i) $\geq w(e_1)/1000$.

Proof. Denote $w := w(e_1)$ and suppose for contradiction that private(v_i) $< w/1000$ for all $i \in [100]$. Since e_1 is friendly, we know that v_1 must send $> 0.4 \deg(v_i) \geq 0.4w$ edges to its descendants in the tree. By our assumption, the number of descendants of the nodes between v_1 and v'_{101} is $< 100 \cdot w/1000 = w/10$, and since our graph G is simple, only $w/10$ edges can go to these nodes. Therefore, there are $> 0.3w$ edges between v_1 and the descendants of v_{101}, which implies that $w(e_i) > 0.3w$ for all $i \in [101]$. The same argument above can be repeated for each e_i where $i \in [100]$ since they are all friendly edges, and we get that for all $i \in [100]$ there are at least $0.3w \cdot 0.4 - w/100 = 0.11w$ edges between the node v_i and the descendants of v_{101}. Consequently, $w(e_{101}) \geq 100 \cdot 0.11w = 11w$. Now, looking up instead of down, and using the fact that e_{11} is also friendly, we

---

[4] We assume that a node is a descendant of itself.

conclude that there are at least $0.4 \cdot 11 \cdot w = 4.4w$ edges between v'_{101} and all nodes above v'_{101} in the tree. But since there are only $< w/100$ nodes above v'_{101} that are not also above v_1, we conclude that there are at least $4.4w - 0.01w > w$ edges between v'_{101} and the nodes above v_1, meaning that $w(e_1)$ must be $> w$, a contradiction. □

Therefore, if each edge e_1 charges its weight to the private nodes on this subpath (letting each node pay ≤ 1000 times), the total charge received by any node (over all edges of P) is at most $100 \cdot 1000 = O(1)$; because each node is in the subpath of at most 100 friendly edges.

For the corner case where there are only < 100 friendly edges in P below $e_1 = (v'_1, v_1)$ we can simply charge $w(e_1)$ to all descendants of v_1. Since e_1 is friendly, there are at least $0.4w(e_1)$ edges between v_1 and its descendants, and due to the simplicity of G, the number of descendants must also be $\geq 0.4w(e_1)$; therefore, each node receives a charge of $\leq 1/0.4$. And since a node may only be charged < 100 times in this way, the total additional charge for each node is $O(1)$.

Finally, we use the heavy-light path-decomposition of T (see [ST83]). In this decomposition of the rooted tree T, each node u_h picks the edge $u_h u_l$ neighboring it from below if $|V(T_{u_l})| > |V(T_{u_h})|/2$, and makes it a *heavy* edge (the rest are light edges). As explained above, the weights on each path of heavy edges can be bounded by the total amount of nodes descending from it. Any leaf-root path can contain at most $\log n$ light edges, and so also at most $\log n$ heavy paths. Finally, each light edge is treated as its own path, and so each node can be charged at most $2 \log n$ times (number of heavy paths and light edges in the path above it to the root), concluding the proof.   □

To obtain the reduction of Theorem 1.6 observe that the sparsifier H in the proof can be obtained from the Gomory-Hu tree T in $O(m)$ time.

4   Single-Source Unfriendly Cuts in $\tilde{O}(m)$ Time

The main result of this section is an algorithm for computing all minimum p, v-cuts for a single source p that are *unfriendly*. The main tool that we use is the ISOLATING-CUTS procedure discovered independently by Li and Panigrahi [LP20] (for global minimum cut algorithm in weighted graphs) and by Abboud, Krauthgamer, and Trabelsi [AKT21b] (for the first subcubic Gomory-Hu tree algorithm for simple graphs), and within a short time span has found several interesting applications [LP21, CQ21, MN21, LNP+21, LPS21, AKT21a, Zha21]. In particular, it was used by Li and Panigrahi [LP21] along with randomized sampling to solve the $(1+\varepsilon)$-approximate single-source minimum cuts problem using $\tilde{O}(1)$ queries to Max-Flow.

DEFINITION 4.1. (MINIMUM ISOLATING CUTS) *Consider a weighted, undirected graph $G = (V, E)$ and a subset $R \subseteq V$ ($|R| \geq 2$). The minimum isolating cuts for R is a collection of sets $\{S_v : v \in R\}$ such that for each node $v \in R$, the set S_v is the side containing v of a minimum cut separating $\{v\}$ and $R \setminus \{v\}$, i.e. for any set S satisfying $v \in S$ and $S \cap (R \setminus \{v\}) = \emptyset$, we have $\delta(S_v) \leq \delta(S)$.*

LEMMA 4.1. (THE ISOLATING-CUTS PROCEDURE ([LP20, AKT21B])) *There is an algorithm that, given an undirected graph $G = (V, E, c)$ on n nodes and m edges of total weight $c(E)$ and a subset $R \subseteq V$, computes the minimum isolating cuts $\{S_v : v \in R\}$ for R in G using $O(\log |R|)$ calls to Max-Flow on graphs with $O(n)$ nodes, $O(m)$ edges, and $O(c(E))$ total weight, and takes $\tilde{O}(m)$ deterministic time outside of these calls.*

THEOREM 4.1. (APPROXIMATE SINGLE-SOURCE MIN-CUTS ([LP21])) *Let G be an undirected graph on n nodes and m edges, with polynomially bounded weights, and let $p \in V$. There is an algorithm that outputs, for each node $v \in V \setminus \{p\}$, a $(1 + \varepsilon)$-approximation of Min-Cut(p, v), and runs in $\tilde{O}(m)$ time plus $\text{poly} \log n$ calls to Max-Flow on $O(n)$-nodes, $O(m)$-edge graphs.*

The following two similar lemmas about unfriendly cuts are the keys for utilizing the ISOLATING-CUTS procedure (with the help of an approximation algorithm) to compute the exact minimum cuts. The idea is that if v's minimum p, v-cut is unfriendly, e.g. due to v having too many edges to p's side (observe that no other node x in v's side could have more than half of its edges to p's or else the minimum p, v-cut is better off by putting x on the other side), then all other nodes u in the cut must have a minimum p, u-cut whose value is smaller than v's cut by a constant factor. This happens because moving v to the other side gives a p, u-cut of much smaller value.

Therefore, a $(1 + \varepsilon)$-approximation to the maximum-flow values to p allows us to distinguish between v and its cut members u, and thus to only pick v as terminal out of the nodes in the cut when using the ISOLATING-CUTS procedure, which isolates v.

LEMMA 4.2. *If the minimum p, v-cut $S, v \in S$ satisfies $|E(\{v\}, V \setminus S)| > 0.6 \cdot \deg(v)$, then for all $v' \in S \setminus \{s\}$ $\lambda_{p,v'} \le 0.8 \cdot \lambda_{p,v}$.*

Proof. Our goal is to show that the cut $S \setminus \{v\}$ has value $\le 0.8 \lambda_{p,v}$, which establishes the upper bound on $\lambda_{p,u}$ for all $u \in S \setminus \{v\}$. First, notice that $\lambda_{p,v} = \delta(S) \le \deg(v)$ because the degree is always an upper bound on the minimum cut. Since $\delta(S) = |E(\{v\}, V \setminus S)| + |E(S \setminus \{v\}, V \setminus S)|$ and $|E(\{v\}, V \setminus S)| > 0.6 \cdot \deg(v)$ we know that $|E(S \setminus \{v\}, V \setminus S)| < \lambda_{p,v} - 0.4 \cdot \deg(v)$. Moreover, $|E(S \setminus \{v\}, \{v\})| < 0.4 \cdot \deg(v)$ because $> 0.6 \deg(v)$ of v's edges leave S. Thus, $\delta(S \setminus \{v\}) = |E(S \setminus \{v\}, \{v\})| + |E(S \setminus \{v\}, V \setminus S)| < \lambda_{p,v} - 0.6 \cdot \deg(v) + 0.4 \cdot \deg(v) < \lambda_{p,v} - 0.2 \cdot \deg(v) \le 0.8 \cdot \lambda_{p,v}$. □

LEMMA 4.3. *If the minimum p, v-cut $S, v \in S$ satisfies $|E(\{p\}, S)| > 0.6 \cdot \deg(p)$, then for all $v' \in (V \setminus S) \setminus \{p\}$ $\lambda_{p,v'} \le 0.8 \cdot \lambda_{p,v}$.*

Proof. The goal is to show that the cut $(V \setminus S) \setminus \{p\}$ has value $\le 0.8 \lambda_{p,v}$; the proof is similar to the one above. First, $\lambda_{p,v} \le \deg(p)$. Then, since $\delta(S) = |E(\{p\}, S)| + |E((V \setminus S) \setminus \{p\}, S)|$ and $|E(\{p\}, S)| > 0.6 \cdot \deg(p)$ we know that $|E((V \setminus S) \setminus \{p\}, S)| < \lambda_{p,v} - 0.6 \cdot \deg(p)$. Moreover, $|E(\{p\}, (V \setminus S) \setminus \{p\})| < 0.4 \cdot \deg(p)$, implying that $\delta((V \setminus S) \setminus \{p\}) = |E(\{p\}, (V \setminus S) \setminus \{p\})| + |E((V \setminus S) \setminus \{p\}, S)| < 0.4 \cdot \deg(p) + \lambda_{p,v} - 0.6 \cdot \deg(p) < \lambda_{p,v} - 0.2 \cdot \deg(p) \le 0.8 \cdot \lambda_{p,v}$. □

We are ready to prove the main result of this section.

Proof. [Proof of Theorem 1.4] The algorithm is as follows. Let $\varepsilon = \delta = 0.01$.

1. Use the algorithm of Theorem 4.1 to get an estimate $c'(v)$ for all nodes $v \in V \setminus \{p\}$ such that $\lambda_{p,v} \le c'(v) \le (1 + \varepsilon) \cdot \lambda_{p,v}$.

2. For each $i \in \{0, \ldots, \log_{(1+\delta)} n\}$ compute the set $T_i = \{v \in V \mid c'(v) \ge (1 + \delta)^i\} \cup \{p\}$ of nodes whose estimate is at least $(1 + \delta)^i$.

3. For each of the $O(\log n)$ sets T_i make a call to the ISOLATING-CUTS procedure of Lemma 4.1 on G where the set of terminals is T_i. For each $v \in T_i$ let S_v be the returned isolating cut of v. Update the estimate $c'(v)$ to be the minimum between $c'(v)$, the value of S_v, and the value of S_p (the returned isolating cut for p).

4. Return the estimates of all nodes.

The running time of the algorithm is the time of the approximate single-source algorithm plus the time for $O(\log n)$ calls to the ISOLATING-CUTS procedure.

For the correctness, let v be any node $v \in V$ such that the minimum v, p-cut C_v is unfriendly and it has value $w := \lambda_{p,v}$. We will show that $c'(v) = w$ by the end of the algorithm. Let i be such that $(1+\delta)^i \le w \le (1+\delta)^{i+1}$. There are two cases:

- C_v is unfriendly because $|E(\{v\}, V \setminus C_v)| > 0.6 \cdot \deg(v)$: In this case, Lemma 4.2 implies that for all $u \in C_v \setminus \{v\}$ we have $\lambda_{p,u} < 0.8 \cdot w$ and therefore $c'(u) < 0.8 \cdot (1 + \varepsilon) \cdot w = 0.9w < w/(1 + \delta) \le (1+\delta)^i$, implying that $u \notin T_i$. Thus, $T_i \cap C_v = \{v\}$ and the minimum isolating cut for v with respect to T_i has value at most $\delta(C_v) = w$, which means that the ISOLATING-CUTS procedure will return a minimum p, v-cut (the cut S_v).

- C_v is unfriendly because $|E(\{p\}, C_v)| > 0.6 \cdot \deg(p)$: In this case, Lemma 4.3 implies that for all $u \in (V \setminus C_v) \setminus \{p\}$ we have $\lambda_{p,u} < 0.8 \cdot w$ and therefore $c'(u) < (1+\delta)^i$ and $u \notin T_i$. Thus, $(V \setminus C_v) \cap T_i = \{p\}$ and thus the minimum isolating cut for p with respect to T_i has value at most $\delta(C_v) = w$, which means that the ISOLATING-CUTS procedure will return a minimum p, v-cut (the cut S_p).

□

5 The Gomory-Hu Tree Algorithm

In this section we explain how to embed our new friendly cut sparsifiers (and the complementary algorithm for unfriendly cuts) into the recent algorithms for Gomory-Hu tree. We first present an overview of our methods, following by the technical details required to prove Theorem 1.3 and the reduction from Gomory-Hu tree to friendly minimum cut sparsifiers in Theorem 1.6.

5.1 Technical Overview. It is likely that each of the recent $n^{2+o(1)}$ algorithms for Gomory-Hu tree [LPS21, AKT21a, Zha21] can be sped up by using our sparsifier instead of Nagamochi-Ibaraki. However, as we attempt to explain next, due to the subtle combination of several ingredients in each of these algorithms, it has to be done in just the right way. The order in which the different ingredients is invoked is different in each of the three algorithms; we have found the presentation of Abboud, Krauthgamer, and Trabelsi [AKT21a] to be the easiest to modify. Let us now explain the issues that arise in this process.

All of these algorithms follow the paradigm of the Gomory-Hu algorithm where there is an intermediate tree T that gets iteratively refined until it becomes the full Gomory-Hu tree. A refinement step splits one of the super-nodes V_i of T into two or more super-nodes, using minimum s, t-cuts that are computed in the *auxiliary graph* G_i of the super-node V_i. The most expensive (and interesting) part of all three algorithms is to compute single-source minimum cuts from a pivot $p \in V_i$ to the other nodes in V_i. Importantly, even though there are $\Omega(n)$ refinement steps throughout the algorithm, the total size of all auxiliary can be upper bounded by $O(m \log n)$. Therefore, when refining a super-node on n_i nodes and m_i edges, it is important to only spend time proportional to n_i and m_i, not n and m.

Our friendly sparsification result of Theorem 1.2 and the algorithm of Theorem 1.4 are easy to adapt when solving the single-source problem in $(m + n^{1.75})^{1+o(1)}$ time (assuming linear-time Max-Flow). However, the sparsification result only works for *simple* graphs, and although our input graph is simple, its auxiliary graphs are not. This is the source of all troubles. We need to be able to solve the single-source problem in G_i while only paying $(m_i + n_i^{1.75})^{1+o(1)}$ time, not $(m + n^{1.75})^{1+o(1)}$, but this seems impossible because we can only sparsify the auxiliary graph directly down to $n\sqrt{w}$ edges, not $n_i\sqrt{w}$. This was not an issue with Nagamochi-Ibaraki sparsification because it can sparsify the auxiliary graph down to $O(n_i w)$ edges.

One approach for resolving this issue (that we were unable to make work) is to first compute a friendly w-cut sparsifier H_w, for all $w \in \{2^i\}_{i=0}^{\log n}$, then compute the Gomory-Hu tree of each H_w, and then use them to answer queries efficiently when computing the Gomory-Hu tree of G. However this approach is not efficient enough because it is unclear how to compute the Gomory-Hu tree of H_w faster than using the $(m\sqrt{n})^{1+o(1)}$ bound for Gomory-Hu tree [LPS21] which gives $n\sqrt{w} \cdot \sqrt{n} = \Omega(n^2)$. We need to be able to sparsify each auxiliary graph inside the Gomory-Hu tree algorithm.

The approach that does work for us is to sparsify each auxiliary graph *using the same original sparsifier of the input graph*. We first compute a friendly w-cut sparsifier H_w of the simple graph G, for all $w \in \{2^i\}_{i=0}^{\log n}$, and then whenever we have an auxiliary graph G_i we compute the "projection" of each sparsifier on the auxiliary graph. The resulting *sparsified auxiliary graphs* (defined in Section 5.2) may have $m_{w,i} > n_i\sqrt{w}$ edges each, but we can prove that the total number of edges in all of these sparsified auxiliary graphs is upper bounded (up to log factors) by the number of edges in the sparsifier $\sum_i m_{w,i} = n\sqrt{w}$, for each w.

5.2 Preliminaries Since the result of this section is obtained by modifying specific theorems in [AKT21a], we will follow their notation and framework. The reader is referred to [AKT21a, Section 2.1] for the necessary preliminaries on the Gomory-Hu tree algorithmic paradigm, intermediate trees, partition trees, GH-equivalent partition trees, auxiliary graphs, and k-partial trees. On top of these definitions, we will use the following notion of sparsified capacitated auxiliary graph that will be used in our modified analysis (similar to previous used notions [AKT20a, Section 3.2] and [LPS21, Section 3]).

Capacitated Auxiliary Graphs Recall that an auxiliary graph (in the Gomory-Hu algorithm) of a super-node V_i in a GH-Equivalent Partition Tree T of a graph is obtained by contracting all connected components in $T \setminus V_i$. A CAG is constructed the same way as an auxiliary graph, but with parallel edges merged into a single edge that is weighted by their total capacity.

Sparsified CAGs Suppose that T is a GH-Equivalent Partition Tree of a simple graph G and let V' be one of its super-nodes and G' be the corresponding auxiliary graph. Let H be a sparsifier of G (that may contract subsets of the nodes). The sparsified CAG H' of G' is obtained from H by contracting each connected component

of $T \setminus \{V'\}$, i.e. in the same way that G' was obtained from G. Observe that if two nodes x, y of G were contracted in the sparsifier H even though they belong to two different connected components X and Y of $T \setminus \{V'\}$, then they will cause the two components X and Y to be contracted together in the sparsified CAG H'. The next lemma shows that this (rather dramatic) consistency enforcement makes sense.

LEMMA 5.1. *Let H be a friendly w-sparsifier of G and let G' be an auxiliary graph of G obtained from super-node V' in a* **GH-Equivalent Partition Tree** *T of G. Then the sparsified* **CAG** *H' preserves all friendly minimum s, t-cuts of value up to w in G' for any pair $s, t \in V'$.*

Proof. Suppose that (S, T) is a minimum s, t-cut in G' (and therefore also in G) of value $\leq w$, and moreover suppose that it is friendly. The friendly w-sparsifier H must preserve (S, T) and therefore only contracts nodes from within S and from within T. This implies that none of the contractions performed on H in order to obtain the sparsified auxiliary graph H' may contract any node from S with any node from T. Therefore, (S, T) is also preserved in H'. $\square$

Next, we bound the total running time in the recursive algorithm. In order to be more general, we phrase it for a **Partition Tree**. Clearly, any **GH-Equivalent Partition Tree** is also a **Partition Tree**, and so we are left with proving the following lemma.

LEMMA 5.2. *Let H be a sparsifier of G and let T be any* **GH-Equivalent Partition Tree** *of G with super-nodes $V_1, \ldots, V_k$ and let H_i be the sparsified* **CAG** *corresponding to V_i. Then, $\sum_{i=1}^{k} |E(H_i)| = O(|E(H)|)$ and $\sum_{i=1}^{k} |V(H_i)| = O(|V(G)|)$.*[5]

To prove the edges part of Lemma 5.2, one might first think to apply Lemma 3.12 [AKT20a] that is used to bound the total sizes of all CAGs of a **GH-Equivalent Partition Tree** T for an input graph G. However, the sparsified CAGs in our context might not be consistent with the tree, leading to contractions that make this direction problematic. Our strategy in overcoming this issue is to construct a new sparsifier H' from H with precisely the same number of edges, such that the CAG and the sparsified CAG of each super-node $V_i \in T$ for H' are the same. This way, we could directly apply Lemma 3.12.

Proof. [Proof of Lemma 5.2] First, in order to bound the total number of nodes in all CAGs $\sum_{i=1}^{k} V(H_i)$, we sum in a way similar to Lemma 4.1 in [AKT21b]. Observe that also here, each sparsified CAG H_i has at most $|V_i| + \deg_T(V_i)$ nodes, which sums up to $O(|V(G)|)$, as claimed.

Second, for the edge bound, construct a graph H' from H by partitioning each node u of H into the parts intersecting each super-node of T, connecting each edge previously connected to u to an arbitrary new part of u. Observe that H' has the same number of edges as H' (but maybe a higher number of nodes), and that no node of H' intersects with more than one super-nodes of T. By Lemma 3.12 in [AKT20a], the total number of edges in all CAGs satisfies $\sum_{i=1}^{k} |E(H'_i)| \leq |E(H')|$ which equals $|E(H)|$.

It remains to bound the total number of edges in each sparsified CAG H_i of H by its corresponding number in the CAG H'_i of H'. This is correct because each edge in H_i can be identified with an edge in H'_i, as follows. Let $e = uw$ be an edge in H that contributes to H_i. Let u', w' be the new parts of u, w, respectively, such that the edge uw in H became $u'w'$ in H'. The only way the edge $u'w' \in H'$ would not contribute one to the total number of edges in H'_i is if both u' and w' belong to the same subtree in T neighboring V_i. However, this would imply that u, w were merged together in H_i, in contradiction to the edge u, w showing in H_i. $\square$

5.3 The Single-Source Algorithm This is the main algorithmic result that uses all the ingredients in order to solve the single-source problem in an auxiliary graph. It augments and improves Theorem 4.2 in [AKT21a] using the new sparsifiers and by using an additional procedure for the unfriendly cuts.

THEOREM 5.1. *Suppose there is an algorithm solving* **Max-Flow** *on undirected graphs with m edges of polynomially bounded weight in $m^{1+o(1)}$ time. Given a simple graph $G = (V, E)$ on $N = |V|$ nodes with a designated pivot*

---

[5]Just like Lemma 3.12 [AKT20a] and Lemma 4.1 [AKT21b] that are used in the proof, this lemma holds even if T is any **Partition Tree** that is not necessarily a **GH-Equivalent Partition Tree**.

$p \in V$, an auxiliary graph G' on the graph nodes $V' \subseteq V(G)$ with $n = |V(G')|, m = |E(G')|$, a 2w-friendly-sparsified *CAG* H_w of G' with n_w nodes and m_w edges for each $w \in \{2^i\}_{i=0}^{\log N}$, and a perturbed version $\tilde{G}$ of G' with unique minimum cuts, one can compute the minimum (p, v)-cut in $\tilde{G}$ for all nodes $v \in V'$ in total time

$$m^{1+o(1)} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \min\{\tilde{O}(m_w w), \frac{N^{1+o(1)}}{w} \cdot m_w\}.$$

Using the existing *Max-Flow* algorithms, the time bound is

$$\left(m + n^{1.5} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \min\left\{m_w w, \frac{N}{w} \cdot (m_w + n_w^{1.5})\right\}\right)^{1+o(1)}.$$

Proof. The algorithm that was used to prove Theorem 4.2 in [AKT21a] suffices, if we make the following three changes.

- In the algorithm of [AKT21a] there is an estimate $c'(v)$ for each node $v \in V$ that is always an upper bound on the minimum p, v-cut value $\lambda_{p,v}$ and each estimate is witnessed by a p, v-cut of the same value. Throughout the algorithm, when cuts are found, the estimates decrease until, in the end, they are guaranteed to equal the minimal values. In [AKT21a] the estimates were initialized using the trivial cuts $(\{v\}, V \setminus \{v\})$.

  In this paper, instead, we use the algorithm of Theorem 1.4 to compute the minimum p, v-cuts that are unfriendly for all $v \in V$ and use them as the initial estimates $c'(v)$ and witness cuts C_v. After this initialization, the algorithm will only use *Max-Flow* calls on friendly sparsifiers but not on G and therefore it will only be guaranteed to find minimum friendly cuts. But since whenever we find a cut we only use it if it is better than the estimate, and the optimal cuts that are unfriendly are at hand from the beginning, the rest of the algorithm can operate as if the sparsifiers preserved all cuts of value $\leq 2w$ (as was actually the case when the Nagamochi-Ibaraki sparsifier G_w was used in [AKT21a]). The additional running time for this stage is $m^{1+o(1)}$ assuming an almost-linear time *Max-Flow* algorithm, and is $\tilde{O}(m + n^{1.5})$ with existing algorithms.

- The second change is that we use the $2w$-friendly-sparsified auxiliary graph H_w instead of the Nagamochi-Ibaraki sparsifier G_w, in every stage w for all $w \in \{2^i\}_{i=0}^{\log N}$. As discussed above, this does not affect the correctness. One subtlety is that the algorithm is also working with a perturbed version $\tilde{G}$ of G (for rather technical reasons[6]) and so, just like in Lemma 2.7 in [AKT21a] the sparsifier can have the same perturbation as G by simply adding the same amount $\varepsilon(e)$ to each edge e in H that was added to it in G. As a result, the running time bound for each query or each call to the ISOLATING-CUTS procedure improves from $\tilde{O}(nw)$ to $\tilde{O}(m_w)$, and the upper bound on the time of stage w improves from $N^{1+o(1)}/w \cdot T(n, nw)$ to $N^{1+o(1)}/w \cdot T(n_w, m_w)$, where $T(n, m)$ is *Max-Flow* time.

- Finally, in [AKT21a] the single-source algorithm assumes that the auxiliary graph has connectivity $\geq \sqrt{N}$ and therefore it can avoid any stage with $w < \sqrt{N}$. This was useful because the existing *Max-Flow* algorithms have running time $\tilde{O}(m + n^{1.5})$ rather than $m^{1+o(1)}$ and so when they had $m = O(nw)$ from Nagamochi-Ibaraki and $w > \sqrt{n}$ they could assume that the extra $n^{1.5}$ is negligible. And the way they enforce that the theorem is always used (inside their full Gomory-Hu tree algorithm) with auxiliary graphs with connectivity at least $\sqrt{N}$ is by having a preliminary stage where they compute a $\sqrt{N}$-partial tree [BHKP07].

  Here, we do not assume a lower bound on the connectivity of the auxiliary graph and therefore have to handle stages where w is very small. We handle them by computing a w-partial tree [BHKP07] on the sparsifier H_w, giving an upper bound of $\tilde{O}(m_w w)$ on the time of stage w which could be better than the $N/w \cdot T(n_w, m_w)$ bound, where $T(n, m)$ is *Max-Flow* time.

---

[6]The purpose was to enforce unique minimum cuts. An alternative approach is to work with minimal (also known as latest) cuts from the source p.

To summarize, the correctness of the algorithm is preserved after making these changes, and the time complexity improves. The time for stage w becomes $\min\{\tilde{O}(m_w w), N^{1+o(1)} T(n_w, m_w)/w\}$ and the time bounds in the theorem are obtained by adding the $m^{1+o(1)}$ cost of the expander decompositions (that are done in exactly the same way as in [AKT21a], i.e. on G') and the $\tilde{O}(T(n,m))$ cost for the unfriendly single-source cuts algorithm. $\square$

5.4 The Full Gomory-Hu Tree Algorithm We are now ready to prove the main theorem of this paper, Theorem 1.3 (as well as the reduction from Gomory-Hu tree to sparsification in Theorem 1.6).

The algorithm uses a randomized pivot strategy to compute the Gomory-Hu tree recursively using calls to the single-source algorithm of Theorem 5.1. It is very similar to the algorithm in Section 4.2 in [AKT21a] with the following two differences:

- The step (Step 1) that computes a $\sqrt{N}$-partial tree is removed, since our single-source algorithm no longer needs the lower bound on the connectivity of an auxiliary graph.

- Instead, the first step of the algorithm (the new Step 1) computes a $2w$-friendly-sparsifier H_w for each $w \in \{2^i\}_{i=0}^{\log N}$ using Theorem 1.2.

- When using the single-source algorithm (in Step 2(c)) we use the algorithm of the new Theorem 5.1 rather than Theorem 4.2 of [AKT21a]. And thus in Step 2(b) when computing the auxiliary graph G_i and its perturbed version, the algorithm also prepares the $2w$-friendly-sparsified auxiliary graph $H_{w,i}$ of G_i for each $w \in \{2^i\}_{i=0}^{\log N}$.

The proof of correctness remains unchanged. The running time analysis is a bit more subtle and it uses Lemma 5.2. We still have a recursive algorithm with $O(\log N)$ levels and the goal is to bound the total time of a single level. Suppose that the GH-Equivalent Partition Tree that corresponds to a level has k super-nodes $V_1, \ldots, V_k$. Denote the number of nodes and edges in the auxiliary graph G_i of super-node V_i by n_i and m_i, and denote the number of nodes and edges in the $2w$-friendly-sparsified auxiliary graph $H_{w,i}$ of G_i by $n_{w,i}$ and $m_{w,i}$. Then, by Theorem 5.1, and assuming an almost-linear time Max-Flow algorithm, the total time for the level is:

$$\sum_{i=1}^{k} \left(m_i^{1+o(1)} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \min\{\tilde{O}(m_{w,i} w), \frac{N^{1+o(1)}}{w} \cdot m_{w,i}\} \right).$$

By Lemma 3.12 in [AKT20a] we have that $\sum_{i=1}^{k} m_i = O(M)$ and by Lemma 5.2 we have that $\sum_{i=1}^{k} m_{w,i} = O(|E(H_w)|) = \tilde{O}(N\sqrt{w})$. Therefore, the total time of a level can be upper bounded by:

$$M^{1+o(1)} + \sum_{i=1}^{k} \sum_{w \in \{2^i\}_{i=0}^{\log N}} m_{w,i} \cdot \min\{w, \frac{N}{w}\} \cdot N^{o(1)}$$

$$\leq M^{1+o(1)} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} N\sqrt{w} \cdot \min\{w, \frac{N}{w}\} \cdot N^{o(1)}.$$

Since $\min\{w^{1.5}, N/w^{0.5}\} \leq N^{0.75}$ for all $w \leq N$, with $w = \sqrt{N}$ achieving the maximum, we obtain the $M^{1+o(1)} + N^{1.75+o(1)}$ upper bound. To prove the reduction from Gomory-Hu tree to fast sparsification in Theorem 1.6 observe that if the edge bound on the sparsifiers was $\tilde{O}(N)$ instead of $\tilde{O}(N\sqrt{w})$ then the above expression would result in an $M^{1+o(1)} + N^{1.5+o(1)}$ upper bound. Finally, using the existing Max-Flow algorithms the upper bound on a level becomes:

$$\sum_{i=1}^{k} \left(\left(m_i + n_i^{1.5} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \min\left\{ m_{w,i} w, \frac{N}{w} \cdot (m_{w,i} + n_{w,i}^{1.5}) \right\} \right)^{1+o(1)} \right).$$

By Lemma 4.1 in [AKT21b] we have that $\sum_{i=1}^{k} n_i = O(N)$ and by Lemma 5.2 we have that $\sum_{i=1}^{k} n_{w,i} = O(N)$, thus we can upper bound the above by:

$$\left(M + N^{1.5} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \sum_{i=1}^{k} \min \left\{ m_{w,i} w, \ \frac{N}{w} \cdot (m_{w,i} + n_{w,i}^{1.5}) \right\} \right)^{1+o(1)}$$

$$\leq \left(M + N^{1.5} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \min \left\{ (\sum_{i=1}^{k} m_{w,i}) w, \ \frac{N}{w} \cdot \sum_{i=1}^{k} (m_{w,i} + n_{w,i}^{1.5}) \right\} \right)^{1+o(1)},$$

because $\sum_i \min\{x_i, y_i\} \leq \min\{\sum_i x_i, \sum_i y_i\}$, which now gives:

$$\leq \left(M + N^{1.5} + \sum_{w \in \{2^i\}_{i=0}^{\log N}} \min \left\{ N\sqrt{w} \cdot w, \ \frac{N}{w} \cdot (N\sqrt{w} + N^{1.5}) \right\} \right)^{1+o(1)},$$

and since $\min\{Nw^{1.5}, N^2/w^{0.5} + N^{2.5}/w\} \leq N^{1.9}$ for all $w \leq N$, with $w = N^{3/5}$ achieving the maximum, the upper bound becomes $M^{1+o(1)} + N^{1.9+o(1)}$.

6 Terminal Sparsification

In this section we use the proof of Theorem 1.2 in Section 2.2, our deterministic algorithm for friendly sparsification using an expander decomposition with node-demands, in order to obtain *terminal sparsifiers* (Definition 1.5 in Section 1.4). Our goal is to prove Theorem 1.7.

Proof. [Proof of Theorem 1.7] Let us only explain the differences from the proof of Theorem 1.2 in Section 2.2. The demand function is defined a bit differently: we still set $\mathbf{d}(v) = \phi^{-1}\sqrt{w}$ for all non-terminals $v \in V \setminus T$, but the demand of terminals is larger, we set $\mathbf{d}(v) = 3\phi^{-1}w$ for all $v \in T$. This changes the total demand to $\mathbf{d}(V) = n \cdot \phi^{-1}\sqrt{w} + 3|T| \cdot \phi^{-1}w$, and the total number of outer-edges to

$$m_0 := \sum_i |E(H_i, V \setminus H_i)| \leq B \cdot \phi \mathbf{d}(V) = Bn\sqrt{w} + 3B|T|w \leq n^{o(1)} \cdot (n\sqrt{w} + |T|w).$$

This causes the upper bound of Claim 2.2 on the number of edges in the sparsifier to change to $n^{o(1)} \cdot (n\sqrt{w} + |T|w)$, simply by plugging in the new value of m_0.

The only interesting change is in the correctness proof of Claim 2.1, which becomes the following.

CLAIM 6.1. *Let $S \subseteq V$ be a minimum s,t-cut in G where $s,t \in T$ that has weight $\delta(S) \leq w$. Then G' preserves this cut S (i.e., never contracts two nodes that are on different sides).*

The proof is similar to that of Claim 2.1, but the argument for lower bounding $\mathbf{d}(L)$ is different (and does not rely on the friendliness of S anymore). There are two cases: either $x \in T$ or $x \notin T$. If $x \in T$ we are in an easy case because $\mathbf{d}(L) \geq \mathbf{d}(x) = 3\phi^{-1}w$. If $x \notin T$ then, since S is a minimum s,t-cut for some pair of nodes where $x \neq s, x \neq t$ then $|E(\{x\}, S)| \geq 0.5 \deg_G(x) \geq 0.4 \deg_G(x)$ or else moving x to the other side gives a better s,t-cut. From this, the proof can be carried on in the same way. □

7 Conclusions

In this paper we have put forward the notion of *friendly cut sparsification* and proved that all friendly cuts with $\leq w$ edges in a simple graph can be preserved with a contracted graph on only $\tilde{O}(n\sqrt{w})$ edges. This edge bound is tight up to log factors, and moreover, the sparsifier can be computed algorithmically in near-linear time. Plugging this sparsification result instead of Nagamochi-Ibaraki's $O(nw)$-edge sparsifiers into the recent algorithms for Gomory-Hu tree (along with a new subroutine for single-source *unfriendly* minimum cuts) leads to the new state-of-the-art bound of $\min\{m + n^{1.75}, m\sqrt{n}\} \cdot n^{o(1)}$ for computing a Gomory-Hu tree of a simple graph.

Furthermore, we have shown that an $O(n \log n)$ edge bound is possible (for all w) if we only want to preserve friendly *minimum* s, t-cuts in a simple graph, and that such a sparsifier can be computed in $(m + n^{1.5})^{1+o(1)}$ time *if and only if* a Gomory-Hu tree can be computed in that time. Since $\Omega(m + n^{1.5})$ is also a conditional lower bound for Gomory-Hu tree in simple graphs, unless the cubic barrier for multigraphs can be broken, this shows that better sparsification is the way towards tight bounds. In a sense, the only remaining question for Gomory-Hu tree in simple graphs is that of computing better friendly sparsifiers.

It is plausible that the "dynamic pivot" derandomization technique of Abboud, Krauthgamer, and Trabelsi [AKT21a] can also be combined with our sparsifiers, leading to a deterministic $(m + n^{1.75})^{1+o(1)}$ upper bound, assuming deterministic $m^{1+o(1)}$-time Max-Flow. And it is also plausible that the technique of Zhang [Zha21] for avoiding the $n^{o(1)}$ factors from using expander decomposition with node-demands could be incorporated to give a randomized $\tilde{O}(m + n^{1.75})$ algorithm, assuming $\tilde{O}(m)$-time Max-Flow.

A side corollary of our results is a subcubic $n^{2.5+o(1)}$ algorihtm for single-source minimum-cuts in simple gaphs (and therefore also Gomory-Hu tree), that is considerably simpler than existing methods [AKT21b, AKT21a, LPS21, Zha21]: execute the classical Gomory-Hu $n \cdot$ Max-Flow(m) algorithm on the friendly n-cut sparsifier of Theorem 1.2 with $m = \tilde{O}(n^{1.5})$ and use the algorithm of Theorem 1.4 for the unfriendly cuts.

References

[AGU72] A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. `doi:10.1137/0201008`.

[AKT20a] A. Abboud, R. Krauthgamer, and O. Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS'20, pages 105–118, 2020. `doi:10.1109/FOCS46700.2020.00019`.

[AKT20b] A. Abboud, R. Krauthgamer, and O. Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'20, page 48–61, 2020. `doi:10.1137/1.9781611975994.4`.

[AKT21a] A. Abboud, R. Krauthgamer, and O. Trabelsi. APMF < APSP? Gomory-Hu tree for unweighted graphs in almost-quadratic time. *Accepted to FOCS'21*, 2021.

[AKT21b] A. Abboud, R. Krauthgamer, and O. Trabelsi. Subcubic algorithms for Gomory–Hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1725–1737, 2021. `doi:10.1145/3406325.3451073`.

[AW14] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science*, FOCS'14, pages 434–443, 2014. `doi:10.1109/FOCS.2014.53`.

[BHKP07] A. Bhalgat, R. Hariharan, T. Kavitha, and D. 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`.

[BK15] A. A. Benczúr and D. R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. `doi:10.1137/070705970`.

[BL16] A. Ban and N. Linial. Internal partitions of regular graphs. *Journal of Graph Theory*, 83(1):5–18, 2016.

[BPWZ14] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick. Listing triangles. In *41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014. `doi:10.1007/978-3-662-43948-7\_19`.

[BTV06] C. Bazgan, Z. Tuza, and D. Vanderpooten. The satisfactory partition problem. *Discrete Applied Mathematics*, 154(8):1236–1245, 2006.

[CGL+20] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167, 2020. `doi:10.1109/FOCS46700.2020.00111`.

[CQ21] C. Chekuri and K. Quanrud. Isolating cuts,(bi-)submodularity, and faster algorithms for connectivity. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[DG19] B. Dudek and P. Gawrychowski. Computing quartet distance is equivalent to counting 4-cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC'19, pages 733–743. ACM, 2019. `doi:10.1145/3313276.3316390`.

[DKW15] M. Dinitz, R. Krauthgamer, and T. Wagner. Towards resistance sparsifiers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz*

International Proceedings in Informatics (LIPIcs), pages 738–755. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. `doi:10.4230/LIPIcs.APPROX-RANDOM.2015.738`.

[FKN$^+$21] A. Ferber, M. Kwan, B. Narayanan, A. Sah, and M. Sawhney. Friendly bisections of random graphs. *arXiv preprint arXiv:2105.13337*, 2021.

[Gab95] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

[GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9:551–570, 1961. Available from: `http://www.jstor.org/stable/2098881`.

[GK00] M. U. Gerber and D. Kobler. Algorithmic approach to the satisfactory graph partitioning problem. *European Journal of Operational Research*, 125(2):283–291, 2000.

[GMT20] A. Gaikwad, S. Maity, and S. K. Tripathi. The satisfactory partition problem. *arXiv preprint arXiv:2007.14339*, 2020.

[Gup01] A. Gupta. Steiner points in tree metrics don't (really) help. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, pages 220–227, 2001. Available from: `http://dl.acm.org/citation.cfm?id=365411.365448`.

[HKNR98] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57:366–375, 1998. `doi:10.1006/jcss.1998.1592`.

[JRR95] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. *Handbooks in Operations Research and Management Science*, 7:225–330, 1995.

[Kar99] D. R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999. `doi:10.1287/moor.24.2.383`.

[KKR12] K.-i. Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.

[KNZ14] R. Krauthgamer, H. Nguyen, and T. Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014. `doi:10.1137/120888843`.

[KPP16] T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.

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

[LNP$^+$21] J. Li, D. Nanongkai, D. Panigrahi, T. Saranurak, and S. Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 317–329, 2021.

[LP20] J. Li and D. Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS'20, pages 85–92, 2020. `doi:10.1109/FOCS46700.2020.00017`.

[LP21] J. Li and D. Panigrahi. Approximate Gomory-Hu tree is faster than $n-1$ max-flows. In *53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC'21, pages 1738–1748. ACM, 2021. `doi:10.1145/3406325.3451112`.

[LPS21] J. Li, D. Panigrahi, and T. Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *Accepted to FOCS'21*, 2021.

[LS21] J. Li and T. Saranurak. Deterministic weighted expander decomposition in almost-linear time. *CoRR*, abs/2106.01567, 2021. `arXiv:2106.01567`.

[LSDK18] Y. Liu, T. Safavi, A. Dighe, and D. Koutra. Graph summarization methods and applications: A survey. *ACM Comput. Surv.*, 51(3), 2018. `doi:10.1145/3186727`.

[MN21] S. Mukhopadhyay and D. Nanongkai. A note on isolating cut lemma for submodular function minimization. *arXiv preprint arXiv:2103.15724*, 2021.

[Moi09] A. Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'09, page 3–12. IEEE Computer Society, 2009. `doi:10.1109/FOCS.2009.28`.

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

[NI92] H. Nagamochi and T. Ibaraki. Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs. *Algorithmica*, 7:583–596, 1992. `doi:10.1007/BF01758778`.

[Pat10] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610, 2010.

[PS89] D. Peleg and A. A. Schäffer. Graph spanners. *J. Graph Theory*, 13(1):99–116, 1989. `doi:10.1002/jgt.3190130114`.

[Rö02] H. Räcke. Minimizing congestion in general networks. In *43rd Symposium on Foundations of Computer Science*, pages 43–52. IEEE Computer Society, 2002. `doi:10.1109/SFCS.2002.1181881`.

[ST83] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. `doi:10.1016/0022-0000(83)90006-5`.

[ST11] D. A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

`doi:10.1137/08074489X`.

[SW97] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[SW19] T. Saranurak and D. Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'19, pages 2616–2635, 2019. `doi:10.1137/1.9781611975482.162`.

[vdBLL⁺21] J. van den Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances. In *53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC'21, page 859–869. ACM, 2021. `doi:10.1145/3406325.3451108`.

[YZ97] R. Yuster and U. Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997.

[Zha21] T. Zhang. Faster cut-equivalent trees in simple graphs. *arXiv preprint arXiv:2106.03305*, 2021.