A Simple Deterministic Reduction From Gomory-Hu Tree to Maxflow and Expander Decomposition

Maximilian Probst Gutenberg*
ETH Zurich
maximilian.probst@inf.ethz.ch

Weixuan Yuan*
ETH Zurich
weyuan@inf.ethz.ch

Abstract

Given an undirected graph G, a Gomory-Hu tree is a tree over the vertex set of G that preserves all pairwise mincuts in G. After various breakthroughs, the recent algorithm from [KPYY25] gives a simple reduction to the maxflow problem. For unweighted m-edge graphs, the reduction requires $\tilde{O}(m)$ randomized time and reduces to maxflow instances of total size $\tilde{O}(m)$. This is the first reduction that is tight up to polylogarithmic factors.

We de-randomize the above state-of-the-art result by presenting a simple deterministic reduction from Gomory-Hu tree to maxflow and expander decomposition. Our reduction again requires only $\tilde{O}(m)$ time and reduces to maxflow and expander decomposition on instances of size $\tilde{O}(m)$.

Our result implies that deterministic maxflow and expander decomposition with $\tilde{O}(m)$ time directly yield an $\tilde{O}(m)$ time deterministic algorithm for Gomory-Hu trees in unweighted graphs. Using state-of-the-art *randomized* expander decomposition algorithms, our reduction retains the guarantees of the reduction in [KPYY25] up to polylogarithmic factors.

1 Introduction

In 1961, Gomory and Hu proposed a pioneering result [GH61] that all pairwise mincuts of an undirected weighted graph $G = (V, E, w)$ can be preserved by a tree on V (later named the Gomory-Hu tree[1]). In the same paper, they proposed an algorithm that computes a Gomory-Hu tree via only $n - 1$ maxflow calls, remarkably improving over the trivial all-pairs maxflow algorithm that takes $\binom{n}{2} = \Theta(n^2)$ maxflow calls.

Fast Algorithms for Gomory-Hu trees. Since their seminal work, a large effort has been put into finding a better Gomory-Hu tree algorithm, yielding faster algorithms for various restricted graph classes. However, for general weighted graphs, Gomory and Hu's result remained state-of-the-art except for the improvements caused by better maxflow algorithms, until the remarkable and extensive progress made by the recent line of works [AKT21b, AKT21a, LPS21, Zha21, AKT22, AKL+22, ALPS23, AKL+25], which improve both randomized and deterministic Gomory-Hu tree algorithm to almost-linear time.[2] In particular, the randomized algorithm in [ALPS23] reduces

*The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation.

[1] A formal definition of a Gomory-Hu tree is given in Definition 4.1.

[2] By convention, we say near-linear time for a time cost of $O(m \cdot \text{polylog}(m))$ and denote it by $\tilde{O}(m)$; and we say almost-linear time for $O(m^{1+o(1)})$ and denote it by $\hat{O}(m)$.

Gomory-Hu tree to maxflow calls on instances with $\tilde{O}(n)$ vertices and $\tilde{O}(m)$ edges in total, plus an additional $n^{1+o(1)}$ time. [AKL+25] then de-randomizes [ALPS23], yielding a deterministic algorithm with a $m^{1+o(1)}$ total instance size and an additional $m^{1+o(1)}$ time. By plugging in the state-of-the-art almost-linear deterministic maxflow algorithm [VDBCP+23], both algorithms have an almost-linear runtime.

Despite these promising breakthroughs, all the above-mentioned almost-linear Gomory-Hu tree algorithms are still far from being practical. They critically rely on a highly complicated subroutine named *Single Source Mincuts (SSMC)*, which computes the edge connectivities of all vertices w.r.t. a given vertex. For both deterministic and randomized settings, solving SSMC requires sophisticated data structure designs, which usually takes up to 20 pages to define and analyze, making the algorithms harder to implement and to adapt to other computational models and settings.

Towards Simple and Practical Algorithms. Very recently, [KPYY25] bypasses SSMC and proposes a simple and likely practical Gomory-Hu tree algorithm for unweighted graphs. Their reduction is also the first to be tight up to polylogarithmic factors: the maxflow calls are on instances of total size $\tilde{O}(m)$, and it only requires $\tilde{O}(m)$ additional time.

However, the state-of-the-art **deterministic** Gomory-Hu tree algorithm proposed by [AKL+25] is still based on the complicated SSMC and involves various complicated ingredients, with a running time equivalent to maxflow calls on instances with total size $m^{1+o(1)}$ plus an additional $m^{1+o(1)}$ time.

It is thus natural to ask if we can obtain a faster **deterministic** reduction – ideally also one that is simpler and practical. We ask the following natural question:

Is there a **deterministic** reduction from Gomory-Hu tree to maxflow calls on $\tilde{O}(m)$ total instance size with an additional time $\tilde{O}(m)$?

In this work, we make significant progress towards answering the question above: for unweighted graphs, if there is a near-linear time deterministic algorithm to compute expander decompositions, then the answer is **yes**. In particular, we present a deterministic simple reduction from Gomory-Hu tree to maxflow and expander decomposition calls on instances with total size $\tilde{O}(m)$ with an additional $\tilde{O}(m)$ overhead. Both max-flow and expander decomposition are fundamental graph primitives that have undergone rapid progress in recent years; notably, it is widely conjectured that deterministic expander decomposition can be achieved in near-linear time. We give a brief introduction to maxflow and expander decompositions below.

Maxflow. The maxflow problem is one of the most central problems in graph algorithms. By the classical mincut-maxflow duality, any maxflow algorithm also gives a mincut. In a recent breakthrough, the first almost-linear-time algorithm for maxflow was given in [CKL+22], and since deterministic almost-linear time maxflow algorithms were given in [VDBCP+23, CKL+24, vdBCK+24]. All state-of-the-art algorithms for Gomory-Hu trees use these fast maxflow algorithms as an essential primitive.

Expander Decomposition. An expander decomposition algorithm decomposes a graph into well-connected vertex clusters such that inter-cluster edges only consist of a small fraction of all edges. We first give a formal definition of expander decomposition below.

Definition 1.1 (Expander Decomposition)**.** *A weighted graph $G = (V, E, w)$ a ϕ-expander if every cut $\emptyset \subsetneq S \subsetneq V$ expands, i.e. $w(E(S, V \setminus S)) \geq \phi \cdot \min\{\text{vol}(S), \text{vol}(V \setminus S)\}$. A ϕ-expander decomposition of graph G is a partition $X_1, X_2, \ldots, X_k$ of the vertex set V such that:*

1. *for every $1 \leq i \leq k$,* cluster *$G[X_i]$ is a ϕ-expander, and*
2. *the total number of* inter-cluster *edges, i.e. edges in G that are not contained in any cluster $G[X_i]$, is $\tilde{O}(\phi \sum_{e \in E} w(e))$*

For parameter $\phi > 0$, the current state-of-the art algorithm to compute a ϕ-expander decompositions requires $\tilde{O}(m)$ **randomized** time [LNPS23]. A **deterministic** algorithm to compute ϕ-expander decompositions with $m^{1+o(1)}$ runtime can be obtained by using the recent deterministic algorithm for computing such a decomposition for simple, unweighted graphs with $\phi = \tilde{O}(1)$ in [Chu23], as a cut player in the cut-matching game [KRV09, OSVV08, CGL+20], using deterministic exact maxflow [VDBCP+23, CKL+24] for the matching player implementation and expander pruning as suggested in [SW19] and various standard techniques. We describe in detail how to combine these methods to obtain the claimed guarantee in Section A. Despite the current time gap between the randomized and deterministic expander decomposition algorithms, we note that the expander decomposition problem remains an active area of research, and it is widely conjectured that a near-linear deterministic expander decomposition algorithm is achievable.

The standard expander decomposition defined in Definition 1.1 is not sufficient for our Gomory-Hu tree algorithm. In Section 3, we generalize the definition of expander decomposition (Definition 3.1) and present a simple reduction to standard expander decomposition calls on instances with total size $\tilde{O}(m)$. Our reduction might be of general interest.

Remark. Very recently, in independent work (see [ADK25]), an alternative $\tilde{O}(m)$ randomized generalized expander decomposition algorithm was proposed. While we show that a simple reduction to standard expander decompositions suffices, [ADK25] carefully generalizes the components of the standard algorithms to compute expander decompositions directly. While our result is much simpler, they achieve smaller polylogarithmic losses in the expansion vs. intercluster edges trade-off and in the runtime.

1.1 Our Contribution

We formally state our main result.

Theorem 1.2 (Deterministic GH tree)**.** *Given an m-edge, undirected, unweighted graph $G = (V, E)$, there is a deterministic algorithm that reduces Gomory-Hu tree to maxflow and expander decomposition calls on instances with total size $\tilde{O}(m)$ edges, with additional time $\tilde{O}(m)$.*[3]

Our reduction is optimal up to polylogarithmic factors, reducing to two of the most-used primitives in modern graph algorithms: maxflow and expander decomposition. It thereby vastly improves over the current state-of-the-art deterministic reduction in [AKL+25] both in overhead complexity and in algorithmic complexity! That is, even given essentially optimal primitives, the reduction [AKL+25] incurs subpolynomial loss while ours is tight up to polylogarithmic factors. Further, the

[3] All maxflow and expander decomposition calls are on undirected weighted graphs with nonnegative integer polynomially bounded weights.

algorithm is much simpler than the reduction in [AKL+25], as we bypass the *single-source mincut* subroutine, which requires a complicated dynamic data structure.

Combined with state-of-the-art randomized expander decomposition algorithms (for example [SW19]), our reduction recovers the guarantees obtained by the state-of-the-art randomized Gomory-Hu tree reduction from [KPYY25] up to polylogarithmic factors.

We also obtain a variant of the algorithm in Theorem 1.2 by incorporating techniques from [LP21], leading to a deterministic reduction from approximate Gomory-Hu tree to maxflow and expander decomposition for general weighted graphs.

Theorem 1.3 (Approximate GH Tree)**.** *Given an m-edge, undirected, weighted graph $G = (V, E, w)$, there is a deterministic algorithm that reduces an $(1+\epsilon)$-approximate Gomory-Hu tree of G to maxflow and expander decomposition calls on instances with total size $\tilde{O}(m/\epsilon^3)$ edges, with additional time $\tilde{O}(m/\epsilon^3)$.*

Again, our deterministic reduction matches the current best randomized approximate Gomory-Hu tree reduction proposed in Section 7 of [LNPS23] up to polylogarithmic factors (and slightly worse dependency in ϵ). We point out that [LNPS23] reduces to $(1-\epsilon)$-approximate maxflow for which a randomized $\tilde{O}(m/\epsilon)$ algorithm is known [She17], thus their reduction can be turned into an $\tilde{O}(m)$ time algorithm.

We organize the rest of the paper as follows. Section 2 lists necessary definitions and technical lemmas. In Section 3, we give a reduction from a generalized version of expander decomposition to the standard decomposition algorithm. The (approximate) Gomory-Hu tree algorithms and their analysis are presented in Section 4.

2 Preliminaries

Let $G = (V, E, w)$ be a weighted graph. We always assume weights are polynomially bounded positive integers. For vertex demand function $\mathbf{d} : V \to \mathbb{N}$ and $S \subseteq V$, let $\mathbf{d}(S) = \sum_{v \in S} \mathbf{d}(v)$. For $S \subseteq V$, we write $\partial S := E(S, \backslash V \setminus S)$ and $\mathrm{vol}(S) = \sum_{v \in S} w(\partial\{v\})$. The *conductance* of S is $\Phi_G(S) = \frac{w(\partial S)}{\min\{\mathrm{vol}(S), \mathrm{vol}(V \setminus S)\}}$; the conductance of S w.r.t. $\mathbf{d}$ is $\Phi_{G,\mathbf{d}}(S) = \frac{w(\partial S)}{\min\{\mathbf{d}(S), \mathbf{d}(V \setminus S)\}}$ We define the conductance of G (w.r.t. $\mathbf{d}$) as $\Phi_G := \min_{S \subseteq V} \Phi_G(S)$ ($\Phi_{G,\mathbf{d}} := \min_{S \subseteq V} \Phi_{G,\mathbf{d}}(S)$), where the minimum is taken over vertex sets S such that $\mathrm{vol}(S) > 0$ and $\mathrm{vol}(V \setminus S) > 0$ ($\mathbf{d}(S) > 0$ and $\mathbf{d}(V \setminus S) > 0$).

For two vertices $s, t \in G$, we denote the edge connectivity between s, t by $\lambda_G(s, t)$, where the subscript is often omitted. For a vertex set $U \subseteq V$, we write $\lambda(U) := \min_{s,t \in U} \lambda(s, t)$. For a positive integer τ, we call a set $C \subseteq V$ is τ-connected if for any $s, t \in C$, $\lambda(s, t) \geq \tau$. A τ-connected component is a maximal τ-connected set in V, and a τ-connected component w.r.t. U is the intersection of some τ-connected component and U.

Let $T_{maxflow}(m)$ and $T_{ED}(m)$ denote the time cost of maxflow and expander decomposition on a graph with m edges, respectively.

Isolating Cuts Lemma. The following Lemma is stated as given in [AKL+22] but was first introduced by [LP20] (an alternative proof is given in [AKT21b]). While all of these articles state the Lemma as a reduction to maximum flow on graphs of total size $\tilde{O}(m)$, we directly state the implied bound when using the deterministic near-linear time maximum flow algorithm given in [VDBCP+23].

Lemma 2.1 (Isolating Cuts Lemma). *There is an algorithm* ComputeIsolatingCuts *that takes as input a graph $G = (V, E, w)$ and a collection $\mathcal{U}$ of disjoint terminal sets $U_1, U_2, \ldots, U_h \subseteq V$. The algorithm then computes a collection of disjoint cuts $S_1, S_2, \ldots, S_h$ where for each $1 \leq i \leq h$, S_i is a vertex-minimal $(U_i, \cup_{j \neq i} U_j)$-mincut. The algorithm is deterministic and runs in time $m^{1+o(1)}$.*

Hit-And-Miss Theorems. We use the following hit-and-miss theorem that is a special case of a theorem from [CSP24].

Theorem 2.2 (see Definition 4 and Theorem 3.1 in [CSP24]). *For any positive integers N, a, b with $a \geq b$ and $b = O(1)$, there is an algorithm* ConstructHitAndMissFamily$([N], a, b)$ *that constructs a family $\mathcal{H}$ of* hit-and-miss *hash functions such that for any $A \subseteq [N]$ with $|A| \leq a$ and $B \subseteq [N] \setminus A$ with $|B| \leq b$, there is some function $h \in \mathcal{H}$ such that $h(x) = 0$ for all $x \in A$ and $h(y) = 1$ for all $y \in B$. The algorithm constructs $\mathcal{H}$ to be of size $(a \cdot \log N)^{O(b)}$ in deterministic time $N \cdot (a \cdot \log N)^{O(b)}$. Moreover, when $a = O(1)$, the size of $\mathcal{H}$ is $O(\log N)$.*

3 Expander Decomposition for General Demands: A Blackbox Reduction

In this section, we present a simple black-box reduction from more general expander decomposition with arbitrary demands to a standard expander decomposition as defined in Definition 1.1. Below we formally define the generalized expander decomposition.

Definition 3.1 (General Expander Decomposition). *We say that a weighted graph $G = (V, E, w)$ is a ϕ-expander* ***with respect to vertex demand vector*** $\mathbf{d} \in \mathbb{N}^V$ *if every cut $\emptyset \subsetneq S \subsetneq V$ expands, i.e. $w(E(S, V \setminus S)) \geq \phi \cdot \min\{\mathbf{d}(S), \mathbf{d}(V \setminus S)\}$.*

A (ϕ, q)-expander decomposition of G ***w.r.t. vertex demand vector*** $\mathbf{d} \in \mathbb{N}^V$ *is a partition $X_1, X_2, \ldots, X_k$ of the vertex set V such that:*

1. *for every $1 \leq i \leq k$,* cluster *$G[X_i]$ is a ϕ-expander w.r.t. $\mathbf{d}$, and*
2. *the total number of* intercluster *edges, i.e. edges in G that are not contained in any cluster $G[X_i]$, is upper bounded by $\tilde{O}(\phi \sum_v \mathbf{d}(v))$.*
3. *the flow problem, where each vertex $v \in X_i$ has $w(E(v, V \setminus X_i))$ units of source mass, $\tilde{O}(\phi \mathbf{d}(v))$ units of sink mass, and each edge has capacity $\tilde{O}(w(e))$, is feasible.*

We give the following simple reduction.

Lemma 3.2. *Given an m-edge, n-vertex weighted graph $G = (V, E, w)$ with vertex demand $\mathbf{d} : V \to \mathbb{N}$ bounded by poly(n) and a parameter $\phi > 0$, computing a ϕ-expander decomposition defined by Definition 3.1 w.r.t. $\mathbf{d}$ can be deterministically reduced to expander decomposition (defined in Definition 1.1) and maxflow calls on instances with total size $\tilde{O}(m)$ with an $\tilde{O}(m)$ overhead.*

Proof. We first show a polylog reduction from the first two properties in Definition 3.1 to standard expander decompositions. Let $D = \sum_{v \in V} \mathbf{d}(v)$ and $W = \sum_{e \in E} w(e)$. Let G' be the graph obtained by adding a self-loop with weight $W\mathbf{d}(v)/D$ to each vertex v in graph G, then $\mathrm{vol}_{G'}(v) = \mathrm{vol}_G(v) + \lceil W\mathbf{d}(v)/D \rceil$. Let q denote the polylog factor in the second point of Definition 1.1. Applying the standard expander decomposition to G' with parameter $\phi' = D\phi/W$, we obtain a partition

$V_1, \ldots, V_k$ of V such that $G'[V_i]$ is a ϕ'-expander w.r.t. $\mathrm{vol}_{G'}$ and the number of intercluster edges is no more than

$$q\phi' \sum_{v \in V} \mathrm{vol}_{G'}(v) \leq q \cdot (D\phi/W) \cdot (3W + n) \leq 4q\phi D.$$

By definition, for $i = 1, \ldots, k$ and any $S \subseteq V_i$ with $d(S), d(V_i \setminus S) \neq 0$, we have

$$\begin{aligned} \phi' = \frac{D\phi}{W} &\leq \frac{w(E(S, V_i \setminus S))}{\min\{\mathrm{vol}_{G'}(S), \mathrm{vol}_{G'}(V_i \setminus S)\}} \\ &\leq \frac{w(E(S, V_i \setminus S))}{\min\{W\mathbf{d}(S)/D, W\mathbf{d}(V_i \setminus S)/D\}} \\ &= \frac{W\Phi_{G[V_i],\mathbf{d}}(S)}{D}. \end{aligned}$$

It then follows $\Phi_{G[V_i],\mathbf{d}} \geq \phi$ and the total number of inter-cluster edges is upper bounded by $4q\phi D$. So $V_1, \ldots, V_k$ satisfies the first two properties in Definition 1.1 for $G, \mathbf{d}$ and $q' = 4q = \tilde{O}(1)$.

For the reduction of the last property in Definition 3.1, we refer the readers to Appendix A of [AKL+25]. The idea is to start from standard expander decomposition from [LS21] and then recursively trim vertices based on a flow problem similar to the one in [SW19]. □

Combined with state-of-the-art expander decomposition algorithms, our reduction yields the following results.

Corollary 3.3. *For a weighted undirected graph $G = (V, E, w)$ with vertex capacities $\mathbf{d} : V \to \mathbb{N}$ bounded by poly(n) and any parameter $\phi > 0$, there exists:*

(1) a deterministic algorithm that computes a ϕ-expander decomposition of G w.r.t. $\mathbf{d}$ in time $m^{1+o(1)}$;

(2) a randomized algorithm that computes a ϕ-expander decomposition of G w.r.t. $\mathbf{d}$ in time $\tilde{O}(m)$.

In the rest of the article, we denote the expander decomposition algorithms as ExpanderDecomp(G, d, ϕ) without explicitly mentioning which algorithm we adopt. We usually set $\mathbf{d} = \mathbf{1}_U$ for some $U \subseteq V$. In such case, we denote the algorithm by ExpanderDecomp(G, U, ϕ). For notational convenience, we denote by q the largest polylog factor hidden behind $\tilde{O}(\cdot)$ in Definition 3.1.

4 Gomory-Hu tree

In this section, we give a simple and deterministic reduction of (approximate) Gomory-Hu tree to expander decomposition and maxflow. For unweighted graphs, we match the rates of state-of-the-art Gomory-Hu tree algorithms for both deterministic and randomized cases, i.e., $\tilde{O}(1)$ maxflow calls if we adopt a $\tilde{O}(m)$ randomized expander decomposition algorithm, and $m^{1+o(1)}$ maxflow calls if we use a deterministic expander decomposition algorithm. For weighted graphs, one variant of our algorithm gives a deterministic reduction of $(1+\epsilon)$-approximate Gomory-Hu tree to expander decomposition and maxflow, leading to a near-linear time reduction to polylog maxflows if we adopt the near-linear time expander decomposition algorithm. We start by give a formal definition of U-Steiner Gomory-Hu tree for some terminal set $U \subseteq V$.

Definition 4.1. *Given a graph $G = (V, E, w)$ and a set of terminals $U \subseteq V$, the Gomory-Hu U-Steiner tree is a weighted tree T on the vertices U, together with a function $f : V \mapsto U$ such that:*

- *for all $s, t \in U$, consider any minimum-weight edge (u, v) on the unique st-path in T. Let U' be the vertices of the connected component of $T \setminus (u, v)$ containing s. Then, $f^{-1}(U') \subseteq V$ is an (s, t)-mincut, and its value is $w_T(u, v)$.*

In the above property, if $f^{-1}(U')$ is only an $(1+\epsilon)$-approximate (s, t)-mincut for some $\epsilon > 0$, we call (T, f) an $(1+\epsilon)$-approximate U-Steiner Gomory-Hu tree.

Our approach generally follows from the deterministic algorithm in [AKL+25], but we bypass the SSMC subroutine by a simple recursion at the price of only working for unweighted graphs. As some technical subroutines remain the same as [AKL+25], we refer interested readers to [AKL+25] for the correctness analysis of the subroutines and will not go into details here.

Following the standard approach, we aim to find a decomposition of (G, U) such that:

1. Gomoru-Hu trees on sub-instances can be correctly pieced together into a large GH tree;
2. the decomposition is somewhat balanced, so that the recursion depth is $m^{o(1)}$.

The first step is the same as [AKL+25]: we start by finding a threshold value τ^* such that the largest τ^*-connected component w.r.t. U contains at least $\frac{3}{4}|U|$ terminals while all (τ^*+1)-connected component contains fewer than $\frac{3}{4}|U|$ vertices. Let C^* denote the corresponding τ^*-connected component and let $C^*_U := C^* \cap U$.

Our major modification of [AKL+25]'s algorithm is based on the following observation (Claim 4.5): When $|C^*_U|$ is large enough (e.g. larger than $\frac{15}{16}|U|$, we can find a balanced decomposition of (G, U) into a collection of (v, r)-mincuts for an arbitrary $r \in C^*_U$. More importantly, we only need to look at (v, r)-mincuts with cut value exactly τ, hence avoiding SSMC. We note that such a decomposition into (v, r)-mincuts is standard ([AKL+22], [ALPS23],[AKL+25]) and we can easily combine the GH trees on sub-instances.

On the other hand, when $|C^*_U| < \frac{15}{16}|U|$, $U = C^*_U \cup (U \setminus C^*_U)$ is simply a balanced decomposition of U. However, C^* and $V \setminus C^*$ are apparently not a pairwise mincuts in G, so we cannot directly combine Gomory-Hu trees on them. To find a valid way to recursion, we first contract C^* into one terminal c and recurse on the contracted graph. By observing that all subtrees of c (in the returned GH tree) is naturally a collection of disjoint (v, r)-mincuts for any $r \in C^*_U$, we can thus perform a standard recursion on the rest of the graph by contracting all such subtrees.

All our subroutines (e.g., finding τ^* and C^*) are only direct applications of hit-and-miss family (Theorem 2.2), isolating cuts (Lemma 2.1), and expander decomposition (Lemma 3.2), leading to an implementable algorithm and simple time analysis. We note that all results in this section, except for Claim 4.9 work for weighted graphs.

4.1 τ-Connectivity and Large Component

We start with the following key subroutine, which constructs a hit and miss family on $A \subseteq V$ and applies isolating cuts to each set in the family. It captures all (v, r)-mincuts such that the v-side is

Algorithm 1: RemoveLeafStep(G, A, L)

$\mathcal{S} \leftarrow \emptyset$.
$\mathcal{H} \leftarrow$ ConstructHitAndMissFamily$(A, L, 2)$. /* See Theorem 2.2. */
foreach $h \in \mathcal{H}$ **do**
  $A_h \leftarrow \{a \in A \mid h(a) = 1\}$.
  $\{S_v\}_{v \in A_h} \leftarrow$ IsolatingCuts(G, A_h). /* See Lemma 2.1. */
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(S_v, v)\}_{v \in A_h}$.
return $\mathcal{S}$

small.

Lemma 4.2. *For a weighted undirected graph G, vertex set $A \subseteq V$, and parameter $\tau > 0$,* RemoveLeafStep(G, A, L) *returns a collection $\mathcal{S}$ in time $\tilde{O}(L^{O(1)} \cdot T_{maxflow}(m))$ such that*

- *for any $v, r \in A$ such that $m_{G,A,v,r} \leq L$, we have $(M_{G,v,r}, v) \in \mathcal{S}$.*

Proof. $|M_{G,v,r} \cap A| = m_{G,A,v,r} \leq L$ implies that there exists some $h \in \mathcal{H}$ such that $h(v) = h(r) = 1$ and $h(v) = 0$ for any $v \in M_{G,v,r} \cap A \setminus \{v\}$. It then follows IsolatingCuts(G, A_h) returns $(M_{G,v,r}, v)$ by minimality of isolating cuts algorithm.

For time bound, note that calling ConstructHitAndMissFamily takes time $L^{O(1)}$ and the size of $\mathcal{H}$ is also $L^{O(1)}$. It then follows from the time bound of isolating cuts algorithm. □

Given a pivot $r \in V$ and $\tau > 0$, the following algorithm finds all vertices in G whose connectivity w.r.t. r is at least τ.

Algorithm 2: CutThreshold(G, r, τ)

$\psi \leftarrow 1/(q \cdot 20 \log_2(n))$; $L \leftarrow 1000 \log_2^2(nW)/\psi$.
$A \leftarrow V$.
for $i = 1, 2, \ldots, O(\log^2 n)$ **do**
  $A' \leftarrow A$; $\mathcal{S} \leftarrow \emptyset$.
  for $k = 1, \ldots, \log_2 n$ **do**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{S \in$ RemoveLeafStep$(G, A', L) : r \notin S, w(\partial S) < \tau\}$.
    $\mathcal{X} \leftarrow$ ExpanderDecomp$(G, A', \psi \cdot \tau)$.
    foreach $X \in \mathcal{X}$ **do**
      Remove $\lceil |A' \cap X|/2 \rceil$ vertices from $A' \cap X$ from the set A'.
  $A \leftarrow A \setminus \cup_{S \in \mathcal{S}} S$
return A

Note that we can find $\lambda(U) := \min_{s,t \in U} \lambda(s, t)$ for $U \subseteq V$ by a binary search with CutThreshold, since $\lambda(U) = \min_{t \in U} \lambda(r, t)$ for any $r \in U$.

Claim 4.3. *For a weighted undirected graph G, vertex $r \in V$, and parameter $\tau > 0$,* CutThreshold(G, r, τ) *returns the set $\{v \in V : \lambda_G(r, v) < \tau\}$ in time $\tilde{O}(T_{ED}(m) + T_{maxflow}(m))$.*

Proof. The correctness follows from Claim 6.22 and Theorem 6.10 in [AKL+25]. The time bound follows from the fact that we call ExpanderDecomp and RemoveLeafStep for $O(\log^3 n)$ times. □

It then follows that $\mathcal{C}_{G,U,\tau}(r) = U \setminus$ CutThreshold(G, r, τ). The following algorithm is similar to Algorithm 9 in [AKL+25], which finds a small set of vertices that contains at least one vertex from

the large τ-connected component. We note that in [AKL+25], we apply SSMC to each vertex in the small set to find the large τ-connected component, which is replaced by simpler CutThreshold in this paper.

Algorithm 3: DetectLargeCC(G, U, τ)

$A \leftarrow U$; $\psi \leftarrow 1/(q \cdot 100 \log_2(n))$; $L \leftarrow \frac{100 \log_2(n)}{\psi}$.
while $|A| > 1/\psi$ **do**
  $\mathcal{S} \leftarrow$ RemoveLeafStep(G, A, L).
  $A' \leftarrow \emptyset$.
  foreach $S \in \mathcal{S}$ *where* $w(\partial S) < \tau$ *and* $|S \cap U| < \frac{3}{4}|U|$ **do**
    $A' \leftarrow A' \cup (A \cap S)$.
  `/* If few leaves were identified, it is safe to do a halving step */`
  if $|A'| < \frac{\psi}{100 \log_2(n)} \cdot |A|$ **then**
    $\mathcal{X} \leftarrow$ ExpanderDecomp$(G, A, \psi \cdot \tau)$.
    foreach $X \in \mathcal{X}$ **do**
      Remove $\lceil |A \cap X|/2 \rceil$ vertices in $A \cap X$ from the set A.
  else
    $A \leftarrow A \setminus A'$
if $\exists r \in A, |\mathcal{C}_{G,U,\tau}(r)| \geq \frac{3}{4}|U|$ **then**
  return $\mathcal{C}_{G,U,\tau}(r)$.
return $\emptyset$.

Theorem 4.4. *For a weighted undirected graph G and parameter $\tau > 0$,* DetectCC *returns empty set if $|\mathcal{C}_{G,U,\tau}(r)| < \frac{3}{4}|U|$ for any $r \in U$; otherwise it returns the only $\mathcal{C}_{G,U,\tau}(r)$ such that $|\mathcal{C}_{G,U,\tau}(r)| \geq \frac{3}{4}|U|$. The algorithm takes time $\tilde{O}(T_{ED}(m) + T_{maxflow}(m))$.*

Proof. The algorithm is the same as Algorithm 9 in [AKL+25] except for the way we verify whether a vertex is in a large τ-connected component. The correctness follows from Claim 6.16 in [AKL+25].

By Claim 6.17 in [AKL+25], the while loop in Algorithm 3 only has $\tilde{O}(1/\psi) = \tilde{O}(1)$ iterations and each iteration call ExpanderDecomp and RemoveLeafStep at most once. After the while loop. there are at most $1/\psi = \tilde{O}(1)$ vertices in A and we call CutThreshold $\log n$ times for each vertex in A. The time bound then follows from the time bound of CutThreshold and RemoveLeafStep. □

4.2 Gomory-Hu Tree Step

By a binary search with DetectLargeCC, we can find the threshold value τ^* w.h.p. such that τ^* is the largest positive number satisfying $\mathcal{C}_{G,U,\tau^*}(r) \geq \frac{3}{4}|U|$. Let C_U^* be the largest τ^*-connected component w.r.t. U.

As mentioned in the overview, either we can balancedly decompose U into (v, r)-mincuts with cut value τ, or $U = C_U^* \cup (U \setminus C_U^*)$ is a balanced decomposition. We formalize the observation in the following claim.

Claim 4.5. *Let $r \in C_U^*$ be an arbitrary vertex and $B = \{v \in C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(r) : m_{G,U,v,r} \leq \frac{15}{16}|U|\}$. We have either $|B| \geq \frac{1}{8}|U|$ or $|C_U^*| < \frac{15}{16}|U|$.*

Proof of Claim 4.5. Suppose $|C_U^*| \geq \frac{15}{16}|U|$, we will show $|B| \geq \frac{1}{8}|U|$. Denote the complement of B in $C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(r)$ by D. If $D = \emptyset$, we have

$$|B| = |C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(r)| \geq \frac{15}{16}|U| - \frac{3}{4}|U| \geq \frac{1}{8}|U|.$$

We may then assume $D \neq \emptyset$. By submodularity and vertex-minimality of the cuts, for any two vertices $x, y \in D$, we have either $M_{G,x,r} \subseteq M_{G,y,r}$, $M_{G,x,r} \subseteq M_{G,y,r}$, or $M_{G,x,r} \cap M_{G,y,r} = \emptyset$. However, the last case can never happen since $|M_{G,U,x,r}|, |M_{G,U,y,r}| \geq \frac{15}{16}|U|$. So the collection $\{M_{G,x,r} : x \in D\}$ is nesting. Let $M_{G,v,r}$ be a minimal set in the collection and let $v' = c_{G,U,r}(v)$, then $M_{G,v,r} = f_{G,U,r}^{-1}(T_{G,U,r}[v'])$. We claim $M_{G,v,r} \cap C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(v') \subseteq B$, hence

$$\begin{aligned}|B| &\geq |M_{G,v,r} \cap C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(v')| \geq |M_{G,v,r} \cap U| - |U \setminus C_U^*| - |C_{G,U,\tau+1}(v')| \\ &\geq \frac{15}{16}|U| - \frac{1}{16}|U| - \frac{3}{4}|U| = \frac{1}{8}|U|.\end{aligned}$$

It remains to prove $M_{G,v,r} \cap C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(v') \subseteq B$. Let $x \in M_{G,v,r} \cap C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(v')$. It follows from the minimality of $M_{G,x,r}$ that $x \in M_{G,x,r} \cap U \subseteq M_{G,v,r} \cap U = T_{G,U,r}[v']$. Since $x \notin \mathcal{C}_{G,U,\tau+1}(v')$, we have $\lambda_G(x, v) = \tau$, so there exists an edge e with weight τ on the path connecting x, v' in $T_{G,U,r}$. Cutting this edge and taking the x-side yields a subtree of $T_{G,U,r}[v']$, whose preimage under $f_{G,U,r}$ is a (x, v')-mincut with weight τ. Let W denote the x-side of this cut. Since v' is the root of the tree $T_{G,U,r}[v']$, the edge e cannot be on the path $T_{G,U,r}[v, r]$, so $r \in V \setminus W$ is on the same side as v'. Since $x \in C_U^* \setminus \mathcal{C}_{G,U,\tau+1}(r)$, we have $\lambda_G(x, r) = \tau$. Therefore, W is also an (x, r)-mincut. By minimality of $M_{G,x,r}$, we have

$$M_{G,x,r} \cap U \subseteq W \cap U \subsetneq M_{G,v,r} \cap U.$$

Recall that $M_{G,v,r}$ is the minimal set in $\{M_{G,y,r} : y \in D\}$, we consequently have $x \notin D$, i.e., $x \in B$. □

The following algorithm repeats Algorithm 11 in [AKL+25] for $O(\log^2 n)$ times and decomposes U into balanced mincuts when $C_U^* \geq \frac{15}{16}|U|$.

Algorithm 4: DECOMP(G, C, r, τ)

$\psi \leftarrow 1/(q \cdot 20\log_2(n))$; $L \leftarrow 1000\log_2^2(nW)/\psi$; $\mathcal{S} \leftarrow \emptyset$.
$A \leftarrow C$.
for $i = 1, 2, \ldots, O(\log^2 n)$ **do**
  $A' \leftarrow A$.
  for $k = 1, \ldots, \log_2 n$ **do**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{S \in$ REMOVELEAFSTEP$(G, A', L) : r \notin S, w(\partial S) = \tau, |S \cap U| \leq \frac{15}{16}|U|\}$.
    $\mathcal{X} \leftarrow$ EXPANDERDECOMP$(G, A, \psi \cdot \tau)$.
    foreach $X \in \mathcal{X}$ **do**
      Remove $\lceil |A' \cap X|/2 \rceil$ vertices from $A' \cap X$ from the set A'.
  $A \leftarrow A \setminus \cup_{S \in \mathcal{S}} S$
return $\mathcal{S}_{\max} := \{(S_v, v) \in \mathcal{S} : S_v$ is maximal in $\mathcal{S}\}$.

Claim 4.6. *For a weighted undirected graph $G = (V, E, w)$,* DECOMP(G, C, r, τ) *returns in time $\tilde{O}(T_{ED}(m) + T_{maxflow}(m))$ a collection $\mathcal{S}_{\max}$ such that sets in $\mathcal{S}_{\max}$ are disjoint and their union contains B.*

Proof. The correctness follows from Claim 6.22 and Theorem 6.10 in [AKL+25]. The time bound follows from the fact that we call ExpanderDecomp and RemoveLeafStep for $O(\log^3 n)$ times. □

4.3 Exact Gomory-Hu Tree

Combining algorithms for the two cases in Claim 4.5 yields the final Gomory-Hu tree algorithm.

Algorithm 5: GHTree(G, U)

if $|U| = 1$ **then**
  return (U, f), where f maps each vertex to the unique vertex in U.
Let $\tau^* \in \mathbb{N}$ be the largest $\tau \in \mathbb{N}$ such that the largest τ-connected component w.r.t. U contains more than $\frac{1}{2}|U|$ vertices. Let C^* be the corresponding τ^*-connected component and let $C^*_U \leftarrow C^* \cap U$.
if $|C^*_U| \geq \frac{15}{16}|U|$ **then**
  $r \leftarrow$ an arbitrary vertex in C^*_U.
  Call Decomp(G, C^*_U, r, τ^*) to obtain $\mathcal{S}_{\max}$.
  foreach $S_v \in \mathcal{S}_{\max}$ **do**
    $G_v \leftarrow G/(V \setminus S_v)$, $U \leftarrow U \cap S_v$; denote the new vertex by x_v.
    $(T_v, f_v) \leftarrow$ GHTree (G_v, U_v).
  $G_{\text{large}} \leftarrow G/\mathcal{S}$, $U \leftarrow U \setminus (\cup_v S_v)$; denote the vertex obtained by contracting S_v by y_v.
  $(T_{\text{large}}, f_{\text{large}}) \leftarrow$ GHTree $(G_{\text{large}}, U_{\text{large}})$.
  return CombineTypeI$((T_{\text{large}}, f_{\text{large}}), \{(T_v, f_v) : v \in R\}, \tau^*)$
else
  $G_{\text{small}} \leftarrow G/C^*$, denote the new vertex by c, $U_{\text{small}} \leftarrow (U \setminus C^*_U) \cup \{c\}$.
  $(T_{\text{small}}, f_{\text{small}}) \leftarrow$ GHTree$(G_{\text{small}}, U_{\text{small}})$.
  Root T_{small} at c and let R denote the children of c.
  $\mathcal{S} \leftarrow \{f^{-1}_{\text{small}}(T_{\text{small}}[v]) : v \in R\}$.
  $G_{\text{large}} \leftarrow G/\mathcal{S}$, $U \leftarrow C^*_U$; denote the vertex obtained by contracting S_v by y_v.
  $(T_{\text{large}}, f_{\text{large}}) \leftarrow$ GHTree $(G_{\text{large}}, U_{\text{large}})$.
  return CombineTypeII$((T_{\text{large}}, f_{\text{large}}), (T_{\text{small}}, f_{\text{small}}), c)$

Algorithm 6: CombineTypeI$((T_{\text{large}}, f_{\text{large}}), \{(T_v, f_v) : v \in R\}, \tau)$

Let T be the disjoint union of T_{large} and all T_v.
for $v \in R$ **do**
  Add an edge between $f_v(x_v)$ and $f_{\text{large}}(y_v)$ with weight τ.
  $S_v \leftarrow f_v^{-1}(T_v)$
  for $x \in S_v$ **do**
    $f(x) \leftarrow f_v(x)$.
for $x \in V \setminus \bigcup_{v \in R} S_v$ **do**
  $f(x) \leftarrow f_{\text{large}}(x)$.
return (T, f)

Algorithm 7: CombineTypeII($(T_{\text{large}}, f_{\text{large}}), (T_{\text{small}}, f_{\text{small}}), c \in T_{\text{small}}$)

Let T be the disjoint union of T_{large} and $T_{\text{small}} \setminus \{c\}$.
Root T_{small} at c and let R denote the children of c.
for $v \in R$ **do**
  Add an edge between v and $f_{\text{large}}(y_v)$ with weight $w_{T_{\text{small}}}(v, c)$.
  $S_v \leftarrow f_v^{-1}(T_{\text{small}}[v])$
  for $x \in S_v$ **do**
    $f(x) \leftarrow f_{\text{small}}(x)$.
for $x \in V \setminus \bigcup_{v \in R} S_v$ **do**
  $f(x) \leftarrow f_{\text{large}}(x)$.

Claim 4.7. *For a weighted undirected graph $G = (V, E, w)$ $\text{GHTree}(G, U)$ outputs a Gomory-Hu U-Steiner tree.*

Proof. We will show by induction on the size of U since each recursive step strictly reduces the size of U. The base case $|U| = 1$ is trivial. For the induction step, we start by noting that the size of $|U|$ always decreases when the recursion depth increases. Let U be the current terminal set with more than one vertex. When $|C_U^*| \geq \frac{15}{16}|U|$, we have $U_{large} \leq \frac{7}{8}|U|$ and $|U_v| \leq \frac{15}{16}|U|$ for each $S_v \in \mathcal{S}_{\max}$. When $|C_U^*| < \frac{15}{16}|U|$, we have $|U_{large}| = |C_U^*| < \frac{15}{16}|U|$ and $|U_{small}| \leq \frac{1}{4}|U| + 1 < \frac{2}{3}|U|$.

It remains to show our two algorithms for combining Gomory-Hu Steiner trees are correct. We have two kinds of recursions in Algorithm 5. The first kind of recursion and combination is exactly the same as the algorithm in Appendix A of [AKL+22]. (Note that we only changed the way for finding the mincuts $\{S_v\}$.) So we omit it here. For the second kind of recursion, since (T_{small}, f) is a U_{small}-Steiner Gomory-Hu tree on G_{small}, we have $\{S_v\}_{v \in R} = \{f_v^{-1}(T_{small}[v])\}_{v \in R}$ are disjoint (c, v)-mincuts in G_{small}. Recall that c is the contraction of the τ-connected component C^*, it then follows that $\lambda_G(x, v) < \tau$ for any $x \in C_U^*$ and $v \in R$. So any (x, v)-mincut cannot cross C^*, hence S_v is also a (x, v)-mincut in G. Fix any $x \in C_U^*$, we then have that $\{S_v\}_{v \in R}$ is a collection of (x, v)-mincuts in G. Therefore, CombineTypeII is equivalent to CombingTypeI, except that the weights in the combined tree may be distinct, which do not affect the correctness of the Gomory-Hu property. □

Claim 4.8. *The recursion depth of $\text{GHTree}(G, U)$ is $O(\log n)$*

Proof. We have proven in the proof of Claim 4.7 that the size of $|U|$ shrinks by a constant factor when recursing. □

Although the correctness and depth results work for weighted graphs. The number of edges may blow up when we recurse if the initial graph is weighted. However, we can bound the total number of edges throughtout the recursions by analyzing the terminals' degree in the initial graph G when G is unweighted.

Claim 4.9. *For an unweighted undirected graph $G = (V, E)$ and terminals $U \subseteq V$, $\text{GHTree}(G, V)$ takes time $\tilde{O}(T_{ED}(m) + T_{maxflow}(m))$.*

Proof. It suffices to bound the total number of vertices and edges across all instances by $\tilde{O}(n)$ and $\tilde{O}(m)$, respectively. Let $\mathcal{I}_i$ denote the set of all instances in recursion level i, where $\mathcal{I}_0 = \{(G, U)\}$. Let u_i, n_i, m_i denote the total number of terminals (i.e., vertices in U), vertices, and edges respectively, where $u_0 = |U|, n_0 = n$, and $m_0 = m$.

For analysis, consider an arbitrary non-leaf instance (G', U') in the recursion tree.

We start by bounding u_i. Let $u(G')$ denote the number of terminals in G' and let $L(G')$ denote the number of terminals across all leaves of the subtree rooted at G'. We then have $L(G') = \sum_{G''\text{is a child of } G'} L(G'')$ and $u(G') = \sum_{G''\text{is a child of } G'} u(G'') - I_{|C'\cap U'|<\frac{15}{16}|U'|}$, since the number of terminals increases by 1 by G'_{small} only when we perform the second type of recursion. Note that L and u coincide at the leaves; we can then conclude that $L(G) \le 2u(G) - 1$. So $u_i \le 2u_0 - 1$ for each i by induction.

For n_i, we have $2|\mathcal{S}_{\max}|$ and $|\mathcal{S}| + 1$ new vertices in two types of recursions, respectively. Since each set $S_v \in \mathcal{S}_{\max}$ (or $\mathcal{S}$) correspond to a vertex in U, we have $n_{i+1} \le n_i + 2u_i$, hence $n_i \le n_0 + 2iu_0$ and the total number of vertices is $\tilde{O}(n)$.

To bound m_i, we divide the edges into two types. We call the contraction of $V \setminus S_v$ in the first recursion and the contraction of C^* in the second recursion *parent node*. For any set containing a parent node, we also call its contraction a parent node. Therefore, any instance in $\mathcal{I}_i$ contains at most i parent nodes. So the total number of edges incident to some parent node in recursion level i is bounded by in_i. It then follows that the total number of such edges is $\tilde{O}(n)$.

It remains to bound the number of edges not incident to any parent node (we call such edges *bad edges*). We denote the number of such edges in recursion level i by m'_i. We define the following potential

$$\Phi_i = \sum_{(G',U')\in\mathcal{I}_i} \mathbf{deg}_G(U' \cap U) \cdot \log_{16/15} |U'|.$$

We will prove that $\Phi_{i+1} + m'_{i+1} \le \Phi_i + m'_i$. Note that $\Phi_0 \le 2m \log n$, it then follows that $m'_i \le m'_0 + \Phi_0 = \tilde{O}(m)$. Consider instance (G', U') at recursion level i. Let τ', C' be the threshold cut value and corresponding τ'-connected component in the call $\textsc{GHTree}(G', U')$. When $|C' \cap U'| \ge \frac{15}{16}|U'|$, the number of bad edges does not increase when we recurse on (G', U'), but the contribution of (G', U') to the potential decreases since any sub-instance of (G', U') has less terminals. We may then assume $|C' \cap U'| < \frac{15}{16}|U'|$. Edges in G'_{small} are either incident to the parent node c' or edges in $G'[V' \setminus C']$. Edges in G'_{large} are either edges in $G'[C']$, or incident to some contracted node $\{S_v\}$. It then suffices to charge the last type of edges to the potential decrease when recursing on (G', U'). Consider $v \in R$ that is not a parent node, and let s_v denote the contraction of S_v. Since $w_{G'}(\partial S_v) < \tau'$, the number of edges incident to s_v in G'_{large} is less than τ'. So the total number of edges incident to some s_v (but not incident to any parent node) in G'_{large} is less than $\tau'|U' \cap U|$ (note that any new terminal created by the algorithm is a parent node by definition). On the other hand, the potential decrease caused by recursion on (G', U') is

$$\begin{aligned}
&\mathbf{deg}_G(U' \cap U) \cdot \log_{16/15} |U'| \\
&\quad - \mathbf{deg}_G(U'_{small} \cap U) \cdot \log_{16/15} |U'_{small}| - \mathbf{deg}_G(U'_{large} \cap U) \cdot \log_{16/15} |U'_{large}| \\
\ge &\mathbf{deg}_G(U' \cap U) \cdot \log_{16/15} |U'| - (\mathbf{deg}_G(U'_{small} \cap U) + \mathbf{deg}_G(U'_{large} \cap U)) \cdot (\log_{16/15} |U'| - 1) \\
= &\mathbf{deg}_G(U' \cap U) \ge \tau' |U' \cap U|
\end{aligned}$$

where the first inequality follows from $|U'_{small}|, |U'_{large}| \le \frac{15}{16}|U'|$ and the last one follows from the fact that U' is a τ'-connected component. □

4.4 Approximate Gomory-Hu Tree

In this section, we give an approximate GH tree algorithm for weighted graphs and prove Theorem 1.3. Let $\epsilon > 0$ be a parameter.

Algorithm 8: ApproximateGHTree(G, U, ϵ)

if $|U| = 1$ **then**
  Return (U, f), where f maps each vertex to the unique vertex in U.
Let $\tau^* \in \mathbb{N}$ be the largest $\tau \in \mathbb{N}$ such that the largest τ-connected component w.r.t. U contains more than $\frac{1}{2}|U|$ vertices. Let C^* be the corresponding τ^*-connected component and let $C^*_U \leftarrow C^* \cap U$. Let r be an arbitrary vertex in C^*_U.
if $|C^*_U| \geq \frac{15}{16}|U|$ **then**
  $\tau \leftarrow \tau^*$, $\mathcal{S} \leftarrow$ Decomp(G, C^*_U, r, τ).
else
  Find $\lambda(U)$ by binary search with CutThreshold.
  $\tau \leftarrow \max\{\tau^*, (1+\epsilon)^{\lceil \log_{1+\epsilon} \lambda(U) \rceil} + 1\}$.
  Call CutThreshold(G, r, τ) and let $\mathcal{S}$ denote the collection $\{S \in \text{RemoveLeafStep}(G, A', L) : r \notin S, w(\partial S) < \tau\}$ returned by the inner for-loop with the largest total vertex size.
foreach $S_v \in \mathcal{S}$ **do**
  $G_v \leftarrow G/(V \setminus S_v)$, $U \leftarrow U \cap S_v$; denote the new vertex by x_v.
  $(T_v, f_v) \leftarrow \text{GHTree}\,(G_v, U_v)$.
$G_{\text{large}} \leftarrow G/\mathcal{S}$, $U \leftarrow U \setminus D$; denote the vertex obtained by contracting S_v by y_v.
$(T_{\text{large}}, f_{\text{large}}) \leftarrow \text{GHTree}\,(G_{\text{large}}, U_{\text{large}})$.
return CombineTypeI$((T_{\text{large}}, f_{\text{large}}), \{(T_v, f_v) : v \in R\}, \tau)$.

The correctenss analysis follows from [LP21]'s framework.

Lemma 4.10. *For any distinct vertices $x, y \in U_{large}$, we have $\lambda_{G_{large}}(x, y) = \lambda_G(x, y)$. For any distinct vertices $x, y \in U_v$, we have $\lambda_G(x, y) \leq \lambda_{G_v}(x, y) \leq (1+\epsilon)\lambda_G(x, y)$.*

Proof. When $|C^*_U| \geq \frac{15}{16}|U|$, each $S_v \in \mathcal{S}$ is exactly a (v, r)-mincut in G. A standard submodularity arguments give us $\lambda_G(x, y) = \lambda_{G_{large}}(x, y)$ for $x, y \in U_{large}$ and $\lambda_G(x, y) = \lambda_{G_v}(x, y)$ for $x, y \in U_v$.

When $|C^*_U| \geq \frac{15}{16}|U|$, we remark that $\mathcal{S}$ is returned by applying IsolatingCuts to r and a set of vertices $R \subseteq U$, and each set $S_v \in \mathcal{S}$ satisfies $\lambda_G(U) \leq w_G(\partial S_v) \leq (1+\epsilon)\lambda_G(U)$, which is the same case as Algorithm 4 in [LP21]. We then refer to the proof of Lemma 3.1 and Lemma 3.2 in [LP21]. □

Lemma 4.11 (c.f. Lemma 3.3 in [LP21])**.** ApproximateGHTree(G, U, ϵ) *outputs a $(1+\epsilon)^{\log_{16/15}|U|}$-approximate Gomory-Hu U-Steiner tree, where d is the largest recursion depth.*

Proof. By Lemma 4.10, recursions from (G, U) to (G_{large}, U_{large}) do not increase the error. Note that any recursion from (G, U) to some (G_v, U_v) reduces the size of U by a factor at least 15/16. The same induction argument as Claim 3.3 in [LP21] then shows that the accumulative error is at most $(1+\epsilon)^{\log_{16/15}|U|}$. □

For time analysis, we first bound the largest depth of Algorithm 8.

Claim 4.12. *The recursion depth of* ApproximateGHTree(G, U, ϵ) *is $\tilde{O}(1/\epsilon)$.*

Proof. Suppose CutThreshold calls IsolatingCuts at most $N = \tilde{O}(q^{O(1)})$ times. Let (G', U') be some instance and let (G'', U'') be its descendent in the recursion tree with $\text{depth}(G'', U'') = \text{depth}(G', U') + N \log n$. We claim that either $\lceil \log_{1+\epsilon} \lambda_{G''}(U'') \rceil \geq \lceil \log_{1+\epsilon} \lambda_{G'}(U') \rceil + 1$ or $|U''| \leq (1 - \frac{1}{16N})|U'|$. Thus, the depth is at most $N^{O(1)} \cdot \log_{1+\epsilon}(mW) = \tilde{O}(1/\epsilon)$.

We then prove the above claim. By Claim 4.5, if we execute Line 8 in some recursion between (G', U') and (G'', U''), we have $|U''| \leq \frac{15}{16}|U'|$. We may then assume all recursions between (G', U') and (G'', U'') execute the else part. Suppose we have $\tau = \tau^*$ in one such recursion. With a slight abuse of notation, let U denote the terminal set before this recursion. Then CutThreshold cuts off at least $\frac{1}{16}$-fraction of $|U|$ by Claim 4.5 and Claim 4.3. As $\mathcal{S}$ is the collection returned by the call of IsolatingCuts with most total vertex size, we have $|U_{large}| \leq (1 - \frac{1}{16N})|U|$ and $|U_v| \leq \frac{1}{4}|U|$, where U is the terminal set before this recursion.

It remains to consider the case $\tau < \tau^*$ throughout the recursions from (G', U') to (G'', U''). We only need to consider the case where each intermediate recursion is from (G, U) to (G_{large}, U_{large}), since we always have $|U_v| \leq \frac{1}{4}|U|$. Assume $\lceil \log_{1+\epsilon} \lambda_{G''}(U'') \rceil = \lceil \log_{1+\epsilon} \lambda_{G'}(U') \rceil$, we then have $\tau = (1+\epsilon)^{\lceil \log_{1+\epsilon} \lambda_{G'}(U') \rceil} + 1$ throughout the recursions. Let $A' = \{v \in U' : \lambda_{G'}(v, r) < \tau\}$ for some $r \in C'$ (note that A' is independent with the choice of r since C' is a τ^*-connected component). By Claim 4.3, calling CutThreshold cuts off A', so $\mathcal{S}$ covers at least $|A'|/N$ vertices and the size of A' decreases by a factor $(1 - 1/N)$ after each recursion. Therefore, $N \log n$ recursions eliminate A' and we have $\lambda_{G''}(U'') \geq \tau = (1+\epsilon)^{\lceil \log_{1+\epsilon} \lambda_{G'}(U') \rceil} + 1$, leading to a contradiction with $\lceil \log_{1+\epsilon} \lambda_{G''}(U'') \rceil = \lceil \log_{1+\epsilon} \lambda_{G'}(U') \rceil$. □

Proof of Theorem 1.3. We may assume $\epsilon \leq 1$. Let $\epsilon' = \epsilon/(2 \log_{16/15} |U|)$, we then have $(1 + \epsilon')^{\log_{16/15} |U|} \leq 1+\epsilon$. By Lemma 4.11, ApproximateGHTree(G, U, ϵ') returns an $(1+\epsilon)$-approximate U-Steiner GH tree.

For time analysis, it suffices to bound the total number of vertices and edges throughout the algorithm. Using the notation in the proof of Claim 4.9, we have

$$u_i = |U|, n_{i+1} \leq n_i + 2u_i \leq 3in.$$

For edge numbers, note that the recursion in ApproximateGHTree is the same as the first type recursion in GHTree, so all edges in a given level can be charged to the original edges in G or edges incident to some parent node. It then follows that $m_i \leq m + in_i \leq m + 3i^2 n$.

As the recursion depth of ApproximateGHTree is $\tilde{O}(1/\epsilon)$, it then follows that the total number of vertices and edges are $\tilde{O}(n/\epsilon^2)$ and $\tilde{O}(m/\epsilon^3)$ respectively. □

5 References

[ADK25] Daniel Agassy, Dani Dorfman, and Haim Kaplan. Expander decomposition for non-uniform vertex measures. *arXiv preprint arXiv:2510.23913*, 2025. 3

[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 *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 884–895. IEEE, 2022. 1, 4, 7, 12

[AKL+25] Amir Abboud, Rasmus Kyng, Jason Li, Debmalya Panigrahi, Maximilian Probst, Thatchaphol Saranurak, Weixuan Yuan, and Wuwei Yuan. Deterministic almost-linear-time gomory-hu trees. In *Accepted by 2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, 2025. 1, 2, 3, 4, 6, 7, 8, 9, 10, 11

[AKT21a] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. APMF $<$ apsp? gomory-hu tree for unweighted graphs in almost-quadratic time. In *Foundations of Computer Science (FOCS), 2021 IEEE 62nd Annual Symposium on*, 2021. 1

[AKT21b] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory-hu tree in unweighted graphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1725–1737. ACM, 2021. 1, 4

[AKT22] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster gomory-hu trees. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3630–3649. SIAM, 2022. 1

[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 *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2204–2212. IEEE, 2023. 1, 2, 7

[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 *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020. 3

[Chu23] Julia Chuzhoy. A distanced matching game, decremental apsp in expanders, and faster deterministic algorithms for graph cut problems. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2122–2213. SIAM, 2023. 3, 18

[CKL+22] 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. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022. 2

[CKL+24] Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, *s-t* shortest path, and minimum-cost flow. *STOC'24*, 2024. 2, 3

[CSP24] Karthik C. S. and Merav Parter. Deterministic replacement path covering. *ACM Transactions on Algorithms*, 20(4):1–35, 2024. 5

[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. 1

[KPYY25] Rasmus Kyng, Maximilian Probst, Weixuan Yuan, and Wuwei Yuan. A simple and fast reduction from gomory-hu trees to polylog maxflows. In *Accepted by 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025. 1, 2, 4

[KRV09] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):1–15, 2009. 3

[LNPS23] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Near-linear time approximations for cut problems via fair cuts. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 240–275. SIAM, 2023. 3, 4

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 85–92. IEEE, 2020. 4

[LP21] Jason Li and Debmalya Panigrahi. Approximate gomory–hu tree is faster than n–1 max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1738–1748, 2021. 4, 14

[LPS21] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. In *Foundations of Computer Science (FOCS), 2021 IEEE 62nd Annual Symposium on*, 2021. 1

[LS21] Jason Li and Thatchaphol Saranurak. Deterministic weighted expander decomposition in almost-linear time. *arXiv preprint arXiv:2106.01567*, 2021. 6

[OSVV08] Lorenzo Orecchia, Leonard J Schulman, Umesh V Vazirani, and Nisheeth K Vishnoi. On partitioning graphs via single commodity flows. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 461–470, 2008. 3, 18

[She17] Jonah Sherman. Area-convexity, l-infty regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017. 4

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2635. SIAM, 2019. 3, 4, 6, 18

[vdBCK+24] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-linear time algorithms for decremental graphs: Min-cost flow and more via duality. *to appear at FOCS'24*, 2024. 2, 18

[VDBCP+23] Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023. 2, 3, 4, 18, 19

[Zha21] Tianyi Zhang. Gomory-hu trees in quadratic time. *arXiv preprint arXiv:2112.01042*, 2021. 1

A Deterministic Expander Decomposition for Weighted Graphs

In [Chu23], Theorem 2.8, a deterministic algorithm was given that, for a simple, unweighted graph G with m edges, computes a cut (A, B) in G with $|E_G(A, B)| \leq \psi(\log n)^{O(1)} \cdot deg(G)$ such that:

- either $deg_G(A), deg_G(B) \geq deg(G)/3$, or
- $deg_G(A) \geq \frac{2}{3} deg(G)$ and $G[A]$ is ψ-expander.

in time $m^{1+o(1)}/\psi$.

In [vdBCK+24], a sparse cut preserving reduction from weighted graphs to unweighted graphs was given. While this reduction produces multi-graphs with self-loops, we can remove the former by taking the subdivision graph, and the latter by replacing a vertex with x self-loops by a sparse expander over x vertices. Together, this yields that we can extend the algorithm above to weighted graphs, however, at a loss of polylogarithmic factors in the size of $|E_G(A; B)|$ and a runtime increase by a factor $\tilde{O}(1/\psi)$.

Consider next the cut-matching game for weighted graphs. While previously, the cut-matching game was mainly presented for unweighted graphs, it extends seamlessly. Extending the statement of [OSVV08], we obtain an $\tilde{O}(\alpha)$-approximate balanced ϕ-sparsest cut procedure for weighted graph $G = (V, E, w)$ and $\phi \in (0, 1)$ by implementing $O(\log n)$ rounds, where in each round, we have to do implement the following steps:

- **Cut Player:** In the cut step, our algorithm is given a weighted graph W of size at most $\tilde{O}(m)$ and has to output an α-approximate balanced ψ-sparsest cut for some $\psi = \tilde{\Omega}(1/\alpha)$.
- **Flow Player:** In the matching step, we are given a demand $\mathbf{d}$ over the vertices and have to either give a flow in G that routes the demands (for weights w), or outputs a violating cut.

We can implement the cut player using our extension of the algorithm by [Chu23], above. We can implement the matching player via exact deterministic max flow [VDBCP+23], which runs in deterministic time $m^{1+o(1)}$. The runtime of this procedure can be bounded by $\tilde{O}(1) \cdot (m^{1+o(1)}/\psi^2 + m^{1+o(1)} = m^{1+o(1)}$ since $\psi = \tilde{\Omega}(1/\alpha)$ and $\alpha = \tilde{O}(1)$.

Finally, we obtain expander decompositions by using the above primitive in the framework of [SW19] and by using their insight that, besides invocations of the algorithm, only a single

additional call to exact maxflow yields a $(\phi, \tilde{O}(1))$-expander decomposition. Since we can again use [VDBCP+23] for the maxflow procedure, this yields a deterministic algorithm that runs in $m^{1+o(1)}$ total time.