Faster Cut-Equivalent Trees in Simple Graphs

Tianyi Zhang ✉ 🆔
Tel Aviv University, Israel

───── **Abstract** ─────

Let $G = (V, E)$ be an undirected connected simple graph on n vertices. A cut-equivalent tree of G is an edge-weighted tree on the same vertex set V, such that for any pair of vertices $s, t \in V$, the minimum (s, t)-cut in the tree is also a minimum (s, t)-cut in G, and these two cuts have the same cut value. In a recent paper [Abboud, Krauthgamer and Trabelsi, STOC 2021], the authors propose the first subcubic time algorithm for constructing a cut-equivalent tree. More specifically, their algorithm has [1]$\widetilde{O}(n^{2.5})$ running time. Later on, this running time was significantly improved to $n^{2+o(1)}$ by two independent works [Abboud, Krauthgamer and Trabelsi, FOCS 2021] and [Li, Panigrahi, Saranurak, FOCS 2021], and then to $(m + n^{1.9})^{1+o(1)}$ by [Abboud, Krauthgamer and Trabelsi, SODA 2022].

In this paper, we improve the running time to $\widetilde{O}(n^2)$ graphs if near-linear time max-flow algorithms exist, or $\widetilde{O}(n^{17/8})$ using the currently fastest max-flow algorithm. Although our algorithm is slower than previous works, the runtime bound becomes better by a sub-polynomial factor in dense simple graphs when assuming near-linear time max-flow algorithms.

1 Introduction

It is well known from Gomory and Hu [10] that any undirected graph can be compressed into a single tree while all pairwise minimum cuts are preserved exactly. More specifically, given any undirected graph $G = (V, E)$ on n vertices and m edges, there exists an edge weighted tree T on the same set of vertices V, such that: for any pair of vertices $s, t \in V$, the minimum (s, t)-cut in T is also a minimum (s, t)-cut in G, and their cut values are equal. Such trees are called Gomory-Hu trees or cut-equivalent trees. In the original paper [10], Gomory and Hu showed an algorithm that reduces the task of building a cut-equivalent tree to $n - 1$ max-flow instances. Gusfield [13] modified the original algorithm Gomory and Hu so that no graph contractions are needed when applying max-flow subroutines. So far, in weighted graphs, faster algorithms for building cut-equivalent trees were only byproducts of faster max-flow algorithms. In the recent decade, there has been a sequence of improvements on max-flows using the interior point method [16, 20, 19, 15, 6], and the current best running time is $\widetilde{O}(m + n^{1.5})$ by [6], so computing a cut-equivalent tree takes time $\widetilde{O}(mn + n^{2.5})$.
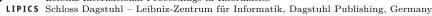
───────

[1] $\widetilde{O}$ hides poly-logarithmic factors.

When G is a simple graph, several improvements have been made over the years. Bhalgat et al. [14] designed an $\widetilde{O}(mn)$ time algorithm for cut-equivalent trees using a tree packing approach based on [9, 7]. Recent advances include an upper bound of $O(m^{3/2}n^{1/6})$ by Abboud, Krauthgamer and Trabelsi [2], and in a subsequent work [3] by the same set of authors, they proposed the first subcubic time algorithm that constructs cut-equivalent trees in simple graphs, and their running time is $n^{2.5+o(1)}$. Recently, by two independent works [4, 18], this running time was improved to $n^{2+o(1)}$ which is almost-optimal in dense graphs, and further to a subquadratic time $(m + n^{1.9})^{1+o(1)}$ by [5].

All of these upper bounds rely on the current fastest max-flow algorithm with runtime $\widetilde{O}(m + n^{1.5})$. However, even if we assume the existence of a $\widetilde{O}(m)$-time max-flow algorithm, the above algorithms still have $n^{2+o(1)}$ running time in dense graphs which contains an extra sub-polynomial factor.

1.1   Our results

Let $\mathsf{MF}(m_0, n_0)$ be the running time complexity of max-flow computation in unweighted multi-graphs with m_0 edges and n_0 vertices, and let $\mathsf{MF}(m_0) = \mathsf{MF}(m_0, m_0)$ for convenience.

The main result of this paper is a near-quadratic time algorithm assuming existence of quasi-linear time max-flow algorithms. For a detailed comparison with recent published works, please refer to the table below where conditional runtime refers to the assumption of near-linear time max-flow algorithms.

▶ **Hypothesis 1.** $\mathsf{MF}(m_0, n_0) = \widetilde{O}(m_0 + n_0)$.

▶ **Theorem 2.** *Let $G = (V, E)$ be a simple on n vertices. Under Hypothesis 1, there is a randomized algorithm that constructs a cut-equivalent tree of G in $\widetilde{O}(n^2)$ time with high probability. Using the current fastest max-flow algorithm [6], the running time becomes $\widetilde{O}(n^{17/8})$.*

reference	conditional runtime	unconditional runtime
[3]	$\widetilde{O}(n^{2.5})$	$n^{2.5+o(1)}$
[4]	$n^{2+o(1)}$	$n^{2+o(1)}$
[18]	$n^{2+o(1)}$	$n^{2+o(1)}$
[5]	$(m + n^{1.75})^{1+o(1)}$	$(m + n^{1.9})^{1+o(1)}$
new	$\widetilde{O}(n^2)$	$\widetilde{O}(n^{17/8})$

Comparison with subsequent works.   In a very recent but unpublished online preprint [1] (see also a note by [21]), significant progress has been made where an unconditional $\widetilde{O}(n^2)$ runtime has been achieved for cut-equivalent trees in general weighted graphs, which completely subsumes our result.

1.2   Technical overview

Our algorithm is largely based on the framework of [3]. In this subsection, we will discuss the running time bottlenecks of [3] and how to bypass them. For simplification, consider the following task. Let $\mathcal{T}$ be a partition tree which is an intermediate tree of the Gomory-Hu algorithm. Take an arbitrary node $N \subseteq V$ of $\mathcal{T}$ which represents a vertex subset of V. Let $G_{\mathcal{T}}[N] = (V_{\mathcal{T}}[N], E_{\mathcal{T}}[N])$ be the auxiliary graph obtained by contracting each component of $\mathcal{T} \setminus \{N\}$ into a single vertex in the original graph G.

Fix a pivot vertex $p \in N$, we want to find a sequence of vertices $v_1, v_2, \cdots, v_l \in N$, and compute a sequence of latest minimum cuts $(L_i, V_{\mathcal{T}}[N] \setminus L_i), 1 \le i \le l$ in $G_{\mathcal{T}}[N]$ for $(v_i, p), 1 \le i \le l$, where $v_i \in K_i, p \notin K_i$, such that:

(1) $l \ge \Omega(|N|)$.

(2) For each $1 \le i \le l$, $|L_i| \le |N|/2$.

If we cut all sides $L_i \cap N$ off of N and form tree nodes, then by the above two properties all tree nodes are vertex subsets of N of size at most $|N|/2$. So, if we can recursively repeat this procedure on smaller subsets, then it would produce a cut-equivalent tree in logarithmically many rounds.

For this task, the basic idea of [3] is to apply expander decompositions. Suppose the original graph G is decomposed into disjoint clusters $V = C_1 \cup C_2 \cup \cdots$ such that each $G[C_i]$ is a ϕ-expander, and the total number of inter-cluster edges is bounded by $\widetilde{O}(\phi n^2)$. For simplicity let us assume G is a roughly regular graph and each vertex $v \in V$ has degree $\deg_G(v) \in [n/2, n)$. For each v, suppose $(L_v, V \setminus L_v)$ is the latest min-cut for (v, p), and let C_v be the ϕ-expander of the expander decomposition that contains v. Vertices of N are divided into three types.

1. Vertices in clusters whose size is less than $n/4$, namely $|C_v| < n/4$.

2. Vertices in clusters whose size is at least $n/4$, namely $|C_v| \ge n/4$, plus that $|C_v \cap L_v| \le 10/\phi$. Note that there are only a constant number of such clusters.

3. $|C_v| \ge n/4$, plus that $|C_v \setminus L_v| \le 10/\phi$.

The first bottleneck

To compute L_v for type-1 vertices, we simply go over all such v's, and compute the max-flow from v to p in $G_{\mathcal{T}}[N]$. Since each type-1 vertex must contribute $n/2 - n/4 = n/4$ inter-cluster edges as the input graph G is simple, the total number of type-1 vertices does not exceed $\widetilde{O}(\phi n)$, summing over all tree nodes N of $\mathcal{T}$.

For type-2 vertices, using the isolating cut lemma devised in [3, 17], we can compute all the sides L_v in $\widetilde{O}(\mathsf{MF}(\mathsf{vol}_G(N))/\phi)$ time, which sum to $\widetilde{O}(\mathsf{MF}(n^2)/\phi)$ over all nodes N of $\mathcal{T}$. So, under Hypothesis 1, the total time cost of type-1 and type-2 vertices is $\widetilde{O}(\phi n^3 + n^2/\phi)$, which is always larger than $n^{2.5}$. So in their algorithm [3], parameter ϕ is equal to $1/\sqrt{n}$.

Our observation is that applying max-flow for each type-1 vertex is too costly. To overcome this bottleneck, we simply avoid computing cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ for both type-1 and type-2 vertices. If the total number of type-2&3 vertices is larger than the total number of type-1 vertices, then we can skip all type-1 vertices. However, if the number of type-1 vertices dominates in N, then the number of type-2&3 vertices is at most $\widetilde{O}(\phi n)$ over all such kind of N. In this case, the total degree $\mathsf{vol}_G(N)$ is at most $\widetilde{O}(\phi n^2)$, and therefore, when summing over all nodes N of $\mathcal{T}$, computing all type-1 vertices takes time at most $\widetilde{O}(\phi^2 n^3)$, instead of $\widetilde{O}(\phi n^3)$, and so the new balance would be $\widetilde{O}(\phi^2 n^3 + n^2/\phi)$. Therefore, if we choose $\phi = n^{-1/3}$, it becomes $n^{7/3}$ which is already better than $n^{2.5}$. In the final algorithm, we will classify expander sizes using $\log n$ many different thresholds, instead of just one threshold (which is n/4 here), and so in the end we can set $\phi = 1/\log^{O(1)} n$. In general cases where graph G has various vertex degrees, we need to apply boundary-linked expander decomposition from a recent work [11]; especially we need to make use of property (3) in Definition 4.2 of [11].

The second bottleneck

In the work [3], in order to compute latest min-cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ for type-3 vertices, they consider the laminar family formed by all sides L_v. If the laminar family has tree depth at most k, then their algorithm can compute cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ by applying $k + 10/\phi$ max-flows in $G_{\mathcal{T}}[N]$. To ensure that the depth is bounded by k, they need a first randomly refine node N into $|N|/k$ sub-nodes which takes $|N|/k$ Gomory-Hu steps. Hence, in total, it requires at least $k + |N|/k > \sqrt{|N|}$ max-flow invocations, which leads to a $n^{2.5}$ running time under Hypothesis 1.

To bypass this barrier, the observation is that the depth of the laminar family in each ϕ-expander is already small, so actually we do not need the help from the refinement step. More precisely, instead of looking at the entire laminar family formed by sets $\{L_v\}_{v \in N}$, we only look at the laminar family formed by sets $\{C_v \cap L_v\}_{v \in N}$ for each cluster C. It can be proved that the depth of this smaller laminar family is always bounded by $O(1/\phi)$. In the end, to compute latest min-cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ for all type-2&3 vertices, we will only use $O(1/\phi)$ max-flow instances in total.

2   Preliminaries

Let $G = (V, E)$ be an arbitrary simple graph on n vertices and m edges with unit-capacities. For any $v \in V$, let $\deg_G(v)$ be the number of its neighbors in V. For any subset $S \subseteq V$, define $\mathsf{vol}_G(S) = \sum_{v \in S} \deg_G(v)$, and let $\mathsf{out}_G(S)$ count the number of edges in $E \cap (S \times (V \setminus S))$, and define $G[S]$ to be the induced subgraph of S on G.

Introduced in [8], the latest minimum (s, t)-cut is a minimum (s, t)-cut such that the side containing s has minimum size as well. It is proved that latest minimum cuts are unique, and can be computed by any max-flow algorithm for (s, t).

Here are some basic facts about min-cuts.

▶ **Lemma 3** (Lemma 2.8 in [3])**.** *For any vertices $a, b, p \in V$, assume $(A, V \setminus A)$ and $(B, V \setminus B)$ are min-cuts for $(a, p), (b, p)$ respectively. If $b \in A$, then $(A \cup B, V \setminus (A \cup B))$ is a min-cut for (a, p) as well.*

▶ **Lemma 4** (Lemma 2.9 in [3])**.** *For any vertices $a, b, p \in V$, assume $(A, V \setminus A)$ and $(B, V \setminus B)$ are min-cuts for $(a, p), (b, p)$ respectively. If $a \notin B$ and $b \notin A$, then $(A \setminus B, V \setminus (A \setminus B))$ is a min-cut for (a, p).*

2.1   Cut-equivalent trees

A cut-equivalent tree is a tree $\mathcal{T}$ on V with weighted edges, such that for any pair $s, t \in V$, there is a minimum cut $(S, V \setminus S)$ in G such that it is also a minimum cut in $\mathcal{T}$ with the same cut value. Now let us turn to define some terminologies for cut-equivalent trees.

Partition trees

A partition tree $\mathcal{T}$ of graph G is a tree whose nodes $U_1, U_2, \cdots, U_l$ represent disjoint subsets of V such that $V = U_1 \cup U_2 \cup \cdots \cup U_l$. For each node U of $\mathcal{T}$, the auxiliary graph $G_{\mathcal{T}}[U] = (V_{\mathcal{T}}[U], E_{\mathcal{T}}[U])$ of U is built by contracting each component of $\mathcal{T} \setminus \{U\}$ into a single vertex in the original graph G.

Gomory-Hu algorithm

Gomory-Hu algorithm provides a flexible framework for constructing a cut-equivalent tree. The algorithm begins with a partition tree $\mathcal{T}$ which is the single node that subsumes the entire vertex set V, and creates more nodes iteratively by refining its nodes. In each iteration, the algorithm picks an arbitrary node that represent a non-singleton subset $U \subseteq V$, and selects two arbitrary vertex $s, t \in U$. Then, compute the minimum cut $(S, V_{\mathcal{T}}[U] \setminus S)$ between s, t in the auxiliary graph $G_{\mathcal{T}}[U]$. Finally, split node U into two nodes that correspond to subsets $S \cap U$ and $(V_{\mathcal{T}}[U] \setminus S) \cap U$ respectively, connected by an edge with weight equal to the value of the cut $(S, V_{\mathcal{T}}[U] \setminus S)$ in $G_{\mathcal{T}}[U]$. For each node W that was U's neighbor on $\mathcal{T}$, reconnect W to node $S \cap U$ if S contains the contracted node that subsumes W; otherwise reconnect W to node $(V_{\mathcal{T}}[U] \setminus S) \cap U$.

A tree is called **GH-equivalent**, if it is a partition tree that can be constructed during Gomory-Hu algorithm by certain choice of nodes U to split and pairs of vertices $s, t \in U$.

Refinement with respect to subsets

Consider a partition tree $\mathcal{T}$ which is GH-equivalent. Let U be one of $\mathcal{T}$'s node and let $R \subseteq U$ be a subset. A refinement of $\mathcal{T}$ with respect to R is to repeatedly execute a sequence of Gomory-Hu iterations by always picking two different vertices from $s, t \in R$ that are currently in the same node of $\mathcal{T}$ and refine $\mathcal{T}$ using a minimum (s, t)-cut. So after the refinement of $\mathcal{T}$ with respect to R, $\mathcal{T}$ is still GH-equivalent.

▶ **Lemma 5** ([12]). *For any node U of $\mathcal{T}$ and any subset $R \subseteq U$, a refinement of $\mathcal{T}$ with respect to R can be computed in time $\widetilde{O}(|R| \cdot \mathsf{MF}(\mathsf{vol}_G(U)))$.*

▶ **Lemma 6** (see definition of partial trees in [3]). *After the refinement on a GH-equivalent tree $\mathcal{T}$ with respect to R, for any $a, b \in R$, let $N_a \ni a$ and $N_b \ni b$ be nodes of $\mathcal{T}$. Then, the min-cut of (N_a, N_b) in $\mathcal{T}$ is a min-cut in G for (a, b) as well.*

k-partial trees

A k-partial tree $\mathcal{T}$ is a GH-equivalent partition tree such that all vertices $u \in V$ such that $\deg_G(u) \leq k$ are singletons of $\mathcal{T}$. The following lemma states that k-partial trees always exist and can be computed efficiently for small k's.

▶ **Lemma 7** ([14]). *There is an algorithm that, given an undirected graph with unit edge capacities and parameter k on n vertices, computes a k-partial tree in time $\min\{\widetilde{O}(nk^2), \widetilde{O}(mk)\}$.*

2.2 Expander decomposition

For any pair of disjoint sets $S, T \subseteq V$, let $E_G(S, T)$ be the set of edges between S, T in G. The conductance of a cut $(S, V \setminus S)$ is $\Phi_G(S) = |E_G(S, V \setminus S)| / \min\{\mathsf{vol}_G(S), \mathsf{vol}_G(V \setminus S)\}$, and the conductance of a graph G is defined as $\Phi_G = \min_S \Phi_G(S)$. A graph G is a ϕ-expander if $\Phi_G \geq \phi$.

For any vertex subset $S \subseteq V$ and positive value $x > 0$, let $G[S]^x$ be the subgraph induced on S where we add $\lceil x \rceil$ self-loops to each vertex $v \in S$ for every boundary edge $(v, w), w \notin S$. As an example, in graph $G[S]^1$ the degrees of all vertices are the same as in the original graph G.

We need a strong expander decomposition algorithm from a recent work [11].

▶ **Definition 8** (boundary-linked expander decomposition [11]). *Let $G = (V, E)$ be a graph on n vertices and m edges, and $\alpha, \phi \in (0, 1)$ be parameters. An (α, ϕ)-expander decomposition of V consists of a partition $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ of V such that the following holds.*
(1) $\sum_{i=1}^{k} \mathsf{out}_G(C_i) \leq \log^4 n \cdot \phi m$.
(2) *For any i, $G[U_i]^{\alpha/\phi}$ is a ϕ-expander.*
(3) *For any i, $\mathsf{out}_G(C_i) \leq \log^7 n \cdot \phi \mathsf{vol}_G(C_i)$.*
In the original paper [11], the upper bounds are stated with $O(\cdot)$ notations that hide constant factors; here we simply raise the exponent of log-factors to simplify the notations.

▶ **Lemma 9** ([11]). *Given any unweighted graph $G = (V, E)$ on n vertices and m edges, for any $\alpha, \phi \in (0, 1)$ such that $\alpha \leq 1/\log^{\mathsf{c}} m$ where c is a certain constant, a (α, ϕ)-expander decomposition can be computed in $\widetilde{O}(m/\phi)$ with high probability.*

2.3   Isolating cuts

▶ **Lemma 10** (isolating cuts [3]). *Given an undirected edge-weighted graph $H = (X, F, \omega)$, a pivot vertex $p \in X$, and a set of terminal vertices $T \subseteq X$. For each $u \in T$, let $(K_u, X \setminus K_u)$ be the latest minimum (u, p)-cut for each $u \in T$. Then, in time $\widetilde{O}(\mathsf{MF}(|F|, |X|))$ we can compute $|T|$ disjoint sets $\{K'_u\}_{u \in T}$ such that for each $u \in T$, if $K_u \cap T = \{u\}$ then $K'_u = K_u$.*

3   Quadratic time Gomory-Hu tree under Hypothesis 1

3.1   The main algorithm

In this section we try to prove the first half of Theorem 2. Let $G = (V, E)$ denote the simple graph as our input data. Define some parameters: $\phi = \frac{1}{10 \log^{\mathsf{c}+10} n}$ is a global conductance parameter that is used to construct expander decompositions, and $r = 10 \log^5 n$ is a sampling parameter which is needed when choosing pivots; here c is the same constant as in Definition 8. Without loss of generality, assume $\sqrt{n}$ is an integral power of 2. Define a degree set $\mathcal{D} = \{\sqrt{n}, 2\sqrt{n}, 2^2 \sqrt{n}, \cdots, n\}$.

Preparation

Throughout the algorithm, $\mathcal{T}$ will be the cut-equivalent tree under construction, where each of $\mathcal{T}$'s node will represent a subset of vertices of V. As a preparation step, compute a (ϕ, ϕ)-expander decomposition on G and obtain a partitioning $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ of V. Categorize clusters in $\mathcal{C}$ according to their sizes: for each 2^i, define $\mathcal{C}_i$ to be the set of clusters whose sizes are within interval $[2^i, 2^{i+1})$.

At the beginning, initialize $\mathcal{T}$ to be a $\sqrt{n}$-partial tree by applying the algorithm from [14] that takes running time $\widetilde{O}(n^2)$.

Iteration

In each round, we will divide simultaneously all nodes of $\mathcal{T}$ which contains at least $20r$ vertices in V. In the end, the total number of rounds will be bounded by $\widetilde{O}(1)$. To describe our algorithm, let us focus on any single node $N \subseteq V$ of $\mathcal{T}$ whose size $|N|$ is at least $20r$. The first step is to refine the partition of N by a set of random pivots. More specifically, sample a pivot subset $R \subseteq N$ of size $10r$ by picking each vertex with probability proportional to its degree in G; more precisely, repeatedly sample for $10r$ times a vertex from N where each vertex $v \in N$ is selected with probability $\deg_G(v)/\mathsf{vol}_G(N)$.

Then, refine the node N of $\mathcal{T}$ by computing a partial tree with respect to R using Lemma 5, which further divides N into several subsets each containing a distinct vertex from R. After applying this pivot-sampling & refining step to each of the original node of $\mathcal{T}$, $\mathcal{T}$ has undergone one pass of partition, and now each node U of $\mathcal{T}$ is associated with a unique pivot vertex $p \in U \cap R$.

For the rest, let us focus on each node U of the current $\mathcal{T}$ as well as its pivot p, such that $U \subseteq N$ is a subdivision of the previous node N but $\mathsf{vol}_G(U) \geq 0.5\mathsf{vol}_G(N)$. For each $k \in \mathcal{D}$, define $U_k = \{u \in U \mid \deg_G(u) \in [k, 2k)\}$, so $U = \bigcup_{k \in \mathcal{D}} U_k$. Take a parameter $d \in \mathcal{D}$ such that $d|U_d|$ is maximized, namely $d \in \arg\max_{k \in \mathcal{D}} k|U_k|$. Therefore, $2d|U_d| \geq \mathsf{vol}_G(U)/\log n$. Next, for each index i, define $\mathsf{cnt}[i]$ to be the number of vertices from U_d that lie within clusters from $\mathcal{C}_i$. Take $s = 2^{i_U}$ such that $\mathsf{cnt}[i_U]$ is maximized.

For each cluster $C \in \mathcal{C}_{i_U}$ such that $C \cap U_d$ is nonempty, conduct the expander search routine described in the next subsection (Algorithm 2) to compute cuts $(K_u, V_\mathcal{T}[U] \setminus K_u)$ in the auxiliary graph $G_\mathcal{T}[U]$ for a set W_C of vertices $u \in W_C \subseteq C \cap U_d$ with respect to pivot p; so $u \in K_u, p \in V_\mathcal{T}[U] \setminus K_u$. It will be guaranteed that for each $u \in W_C$, we have $\mathsf{vol}_G(K_u \cap U) \leq 0.5\mathsf{vol}_G(N)$. In the end, define $W = \bigcup_{C \in \mathcal{C}_{i_U}} W_C$.

We will prove that all $(K_u, V_\mathcal{T}[U] \setminus K_u)$ are latest min-cuts for (u, p). Since all latest minimum cuts $(K_u, V_\mathcal{T}[U] \setminus K_u)$ are with respect to p, they should form a laminar family. Then, for each $u \in W$ such that K_u is maximal in the laminar family, split $K_u \cap U$ off the node U and create a new node for vertex set $K_u \cap U$. Since we always take maximal K_u's, all of these sets are disjoint in $V_\mathcal{T}[U]$, so the creation of new nodes on $\mathcal{T}$ is well-defined. Pseudo-code CondGomoryHu summarizes our algorithm.

◼ **Algorithm 1** CondGomoryHu($G = (V, E)$).

---

1 initialize a partition tree $\mathcal{T}$, as well as parameters ϕ, r;
2 **while** $\exists N \subseteq V$, U *a node of* $\mathcal{T}$, $|N| \geq 20r$ **do**
3   **for** *node* N *of* $\mathcal{T}$ *with* $|N| \geq 20r$ **do**
4   repeat for $10r$ times: each time we sample a vertex $u \in N$ with probability $\frac{\deg_G(u)}{\mathsf{vol}_G(N)}$, and let the sampled set be R;
5   call Lemma 5 on node N with respect to R;
6   **for** *node* $U \subseteq N$ *of* $\mathcal{T}$ *such that* $\mathsf{vol}_G(U) > 0.5\mathsf{vol}_G(N)$ **do**
7   take $d \in \mathcal{D}$ such that $d|U_d|$ is maximized;
8   take $s = 2^{i_U}$ such that $\mathsf{cnt}[i_U]$ is maximized;
9   **for** *each* $C \in \mathcal{C}_{i_U}$ **do**
10   run expander search on C within node U to compute a subset $W_C \subseteq C \cap U_d$, and the latest min-cuts $(K_u, V_\mathcal{T}[U] \setminus K_u)$ for each $u \in W_C$;
11   define $W = \bigcup_{C \in \mathcal{C}_{i_U}} W_C$;
12   for each $u \in W$ such that K_u is maximal, split $K_u \cap U$ off of U and create a new node on $\mathcal{T}$;
13 **for** *node* N *of* $\mathcal{T}$ *such that* $|N| < 20r$ **do**
14   repeatedly refine N using the generic Gomory-Hu steps until all nodes are singletons;
15 **return** $\mathcal{T}$ as a cut-equivalent tree;

3.2 Finding latest min-cuts in expanders

Our algorithm is similar to the one from [3]. The input to this procedure is a node $U \subseteq V$ of the current partition tree $\mathcal{T}$ under construction, together with parameters s, d defined previously, as well as an expander $C \in \mathcal{C}_{i_U}$ that intersects U_d. The output of this procedure will be a subset $W_C \subseteq C \cap U_d$, and their cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ for all $u \in W_C$ in the auxiliary graph $G_{\mathcal{T}}[U]$, with the extra property that $\mathsf{vol}_G(K_u \cap U) \leq 0.5\mathsf{vol}_G(N)$. In the end, we will show that with high probability, all these cuts are latest min-cuts in $G_{\mathcal{T}}[U]$.

To describe our algorithm, consider all vertices $u \in C \cap U_d$ and their latest minimum cuts $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ with respect to pivot $p \in U$. Let λ_u be the cut value of $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ in $G_{\mathcal{T}}[U]$. All of the sets $L_u \cap C \cap U_d$ should form a laminar family, which corresponds to a tree structure $\mathcal{T}_p^d[C]$, where each tree node M of $\mathcal{T}_p^d[C]$ packs a subset of $C \cap U_d$, such that for all $u \in M$ the set $L_u \cap C \cap U_d$ are the same. More specifically, $\mathcal{T}_p^d[C]$ is constructed as follows: first arrange the laminar family $\{L_u \cap C \cap U_d\}_{u \in C \cap U_d}$ as a tree, and then for each node $L_u \cap C \cap U_d$ on this tree, associate with this node the set $M = \{v \in C \cap U_d \mid L_v \cap C \cap U_d = L_u \cap C \cap U_d\} \subseteq L_u \cap C \cap U_d$.

We need to emphasize that our algorithm does not know $\mathcal{T}_p^d[C]$ at the beginning, but it will gradually explore part of $\mathcal{T}_p^d[C]$ during the process.

Preparation

Initialize $W_C = \emptyset$. Assume $|C \cap U_d| \geq 10/\phi^2$; otherwise we could simply reset $W_C = C \cap U_d$ and run $|C \cap U_d|$ instances of max-flow to compute all latest cuts. As a preparatory step, the algorithm repeatedly takes a random subset $T \subseteq C \cap U_d$ by selecting each vertex independently with probability ϕ. Then apply Lemma 10 on graph $G_{\mathcal{T}}[U]$ to compute isolating cuts of terminal vertices from T with respect to pivot p. This procedure goes on for $10 \log n/\phi$ iterations, and for each $u \in C \cap U_d$, let $(A_u^i, V_{\mathcal{T}}[U] \setminus A_u^i)$ be the isolating cut computed for u in the i-th iteration; if u was not selected by T in the i-th iteration, simply set $A_u^i = \{u\}$. Finally, let A_u be the set among $\{A_u^i\}_{1 \leq i \leq 10 \log n/\phi}$ such that the cut value of $(A_u, V_{\mathcal{T}}[U] \setminus A_u)$ in the auxiliary graph $G_{\mathcal{T}}[U]$ is minimized; to break ties, we select A_u^i that minimizes $|A_u^i|$. Let κ_u be the cut value of $(A_u, V_{\mathcal{T}}[U] \setminus A_u)$ for $u \in C \cap U_d$.

If one of $|A_u \cap C \cap U_d| > 2/\phi$, then the algorithm fails and aborts; we will prove that the failure probability is small.

Exploring $\mathcal{T}_p^d[C]$

A node M of $\mathcal{T}_p^d[C]$ is called **large** if $|C \cap U_d \setminus L_u| \leq 2/\phi$ for any $u \in M$, and if $|L_u \cap C \cap U_d| \leq 2/\phi$ it is called **small**. Similarly, a vertex $u \in C \cap U_d$ is called large, if $|C \cap U_d \setminus L_u| \leq 2/\phi$; otherwise if $|L_u \cap C \cap U_d| \leq 2/\phi$, it is called small.

▶ **Lemma 11.** *All large nodes on $\mathcal{T}_p^d[C]$ should lie on a single path ended at root.*

Proof. Suppose otherwise there exists two different large nodes M_1, M_2 of $\mathcal{T}_p^d[C]$ such that $M_1 \cap M_2 = \emptyset$. Take any $u_1 \in M_1, u_2 \in M_2$. Since M_1, M_2 are large nodes, we know $|C \cap U_d \setminus L_{u_1}| \leq 2/\phi$, $|C \cap U_d \setminus L_{u_2}| \leq 2/\phi$. As $C \cap U_d \cap L_{u_1}$ and $C \cap U_d \cap L_{u_2}$ are disjoint, we have $|C \cap U_d| \leq |C \cap U_d \setminus L_{u_1}| + |C \cap U_d \setminus L_{u_2}| \leq 4/\phi$, which contradicts that $|C \cap U_d| \geq 10/\phi^2$. ◀

The main idea of our algorithm is to find the lowest large node on $\mathcal{T}_p^d[C]$; to clarify a bit more, here "lowest" means farthest from root. Initialize a set $S \leftarrow C \cap U_d$ and maintain an ordering of vertices in S according to the cut value of κ_u, also initialize variable $Q \leftarrow \emptyset$.

Repeat the following procedure: take $u \in S \setminus Q$ such that κ_u is maximized. Apply max-flow in graph $G_{\mathcal{T}}[U]$ to compute the latest min-cut L_u for (u, p). Consider two possibilities.

- L_u is small. Then assign $S \leftarrow S \setminus (Q \cup L_u)$, and $Q \leftarrow \emptyset$.
- L_u is large. If $L_u \cap C \cap U_d = S$, then add u to Q; otherwise if $L_u \cap C \cap U_d \neq S$, reset $S \leftarrow L_u \cap C \cap U_d$ and $Q \leftarrow \{u\}$.

The repetition terminates if either (1) $|C \cap U_d \setminus S| > 2/\phi$ or (2) $|Q| > 2/\phi$. In the first case, assign $W_C \leftarrow \{u \in S \mid \mathsf{vol}_G(A_u \cap U) \leq 0.5\mathsf{vol}_G(N)\}, K_u \leftarrow A_u$ and terminate. Note that this notation $\mathsf{vol}_G(A_u \cap U)$ is well-defined, since all vertices in U are also vertices in V, not contracted vertices in $V_{\mathcal{T}}[U]$.

Now suppose we are in the second case. Let $L = L_v$ for an arbitrary $v \in Q$. We will prove afterwards that $(L, V_{\mathcal{T}}[U] \setminus L)$ is the latest min-cut corresponding to the lowest large node. Let κ be the cut value of $(L, V_{\mathcal{T}}[U] \setminus L)$, and define $B = \{u \in S \mid \kappa_u > \kappa\}$. Assign $W_C \leftarrow \{u \in S \setminus B \mid \mathsf{vol}_G(A_u \cap U) \leq 0.5\mathsf{vol}_G(N)\}, K_u \leftarrow A_u$ and for each $u \in W_C$.

After that, take an arbitrary $v \in Q$. If it satisfies that $\mathsf{vol}_G(L \cap U) \leq 0.5\mathsf{vol}_G(N)$, then update $W_C = W_C \cup \{v\}$ and $K_v \leftarrow L$. The whole exploration procedure is summarized as pseudo-code ExploreTree.

■ **Algorithm 2** ExploreTree(U, p, d, C).

---

1 prepare A_u and κ_u for all $u \in C \cap U_d$;

2 initialize $S \leftarrow C \cap U_d, Q \leftarrow \emptyset$;

3 **while** $\max\{|C \cap U_d \setminus S|, |Q|\} \leq 2/\phi$ **do**

4   take $u \in \arg\max_{v \in S \setminus Q}\{\kappa_v\}$;

5   apply max-flow to compute L_u;

6   **if** u *is small* **then**

7 $S \leftarrow S \setminus (Q \cup L_u)$, and $Q \leftarrow \emptyset$;

8   **else**

9   **if** $L_u \cap C \cap U_d = S$ **then**

10 $Q \leftarrow Q \cup \{u\}$;

11   **else**

12 $S \leftarrow L_u \cap C \cap U_d, Q \leftarrow \{u\}$;

13 **if** $|C \cap U_d \setminus S| > 2/\phi$ **then**

14   **return** $W_C \leftarrow \{u \in S \mid \mathsf{vol}_G(A_u \cap U) \leq 0.5\mathsf{vol}_G(N)\}, K_u \leftarrow A_u, \forall u \in W_C$;

15 **else**

16   define $B = \{u \in S \mid \kappa_u > \kappa\}$ where κ is the cut value of $(S, V_{\mathcal{T}}[U] \setminus S)$;

17 $W_C \leftarrow \{u \in S \setminus B \mid \mathsf{vol}_G(A_u \cap U) \leq 0.5\mathsf{vol}_G(N)\}, K_u \leftarrow A_u, \forall u \in W_C$;

18   draw an arbitrary vertex $v \in Q$ and set $L = L_v$;

19   **if** $\mathsf{vol}_G(L \cap U) \leq 0.5\mathsf{vol}_G(N)$ **then**

20   assign $W_C \leftarrow W_C \cup \{v\}$ and $K_v \leftarrow L$;

21   **return** W_C;

3.3   Proof of correctness

First we prove a basic property of isolating cuts, which is also used in [3].

▶ **Lemma 12** ([3]). *For each $u \in C \cap U_d$, either $|L_u \cap C \cap U_d| \le 2/\phi$ or $|C \cap U_d \setminus L_u| \le 2/\phi$; namely each vertex is either large or small. Furthermore, with high probability, when u is small, $A_u = L_u$.*

Proof. Since $\deg_G(u) < 2d$, the cut value of $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is smaller than $2d$. Unpack all contracted vertices of $V_{\mathcal{T}}[U]$, and let $L'_u \subseteq V$ be the set of vertices belonging to L_u or contracted in L_u. Therefore, since $G_{\mathcal{T}}[U]$ is a contracted graph of G, the cut value of $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is equal to the cut value of $(L'_u, V \setminus L'_u)$.

Suppose otherwise that $|L'_u \cap C \cap U_d| > 2/\phi$ and $|C \cap U_d \setminus L'_u| > 2/\phi$. Then, by property (2) of the (ϕ, ϕ)-expander decomposition, $G[C]^1$ is a ϕ-expander, and so the number of edges between $L'_u \cap C$ and $C \setminus L'_u$ is at least $\phi \cdot \min\{\mathsf{vol}_G(L'_u \cap C), \mathsf{vol}_G(C \setminus L'_u)\} > 2d$, which is a contradiction.

Let us turn to the second half of the statement. Suppose u is small, and so $|L_u \cap C \cap U_d| \le 2/\phi$. Then, since T selects each vertex in $C \cap U_d$ with probability ϕ, with probability $\phi \cdot (1 - \phi)^l > \phi/8$, $T \cap L_u \cap C = \{u\}$ is a singleton. In this case, by Lemma 10, $A_u^i = L_u$. As T is sampled for $10 \log n/\phi$ times, with high probability $A_u = L_u$. ◀

Here is a basic fact regarding large vertices.

▶ **Lemma 13.** *For any large vertex u, $\lambda_u < \kappa_u$.*

Proof. If $\lambda_u = \kappa_u$, then the latest min-cut should be contained in A_u, which contains at most $2/\phi$ vertices from $C \cap U_d$, and so u cannot be large. ◀

Next we analyze the behavior of the while-loop in ExploreTree.

▶ **Lemma 14.** *If $Q \ne \emptyset$, then for each $u \in Q$, $L_u \cap C \cap U_d = S$.*

Proof. Each time S is updated, either Q adds a vertex u on line-10 such that $L_u \cap C \cap U_d = S$, or Q is updated to $\{u\}$ on line-12. So the equality always holds. ◀

▶ **Lemma 15.** *At the beginning of any iteration of the while-loop, $\forall v \in S$, if v is large, then we have $L_v \cap C \cap U_d \subseteq S$.*

Proof. We prove this statement by induction on the number of iterations. Initially, this holds as $S = C \cap U_d$. For any intermediate iteration, consider two cases.

▪ u is small. We claim that before updating S, for all large vertices $v \in S \setminus (Q \cup L_u)$, L_v and $L_u \cup Q$ are disjoint; if this can be proved, then we conclude $L_v \cap C \cap U_d \subseteq S \setminus (Q \cup L_u)$, as $L_v \cap C \cap U_d \subseteq S$ holds before.

Suppose that $L_v \cap L_u \ne \emptyset$. Then as all latest minimum cuts form a laminar family and that $v \notin L_u$, it must be $L_u \subseteq L_v$. As v is large, $L_v \cap C \cap U_d$ contains more vertices than $A_v \cap C \cap U_d$, and so by Lemma 13, we have $\lambda_v < \kappa_v$. Now, by line-4, since κ_u is the largest among all vertices in $S \setminus Q$, $\kappa_v \le \kappa_u$. Finally, using Lemma 12, we know $\kappa_u = \lambda_u$ as u is small. Concatenating all the inequalities we have:

$$\lambda_v < \kappa_v \le \kappa_u = \lambda_u$$

which contradicts the fact that $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is a min-cut for (u, p) as $L_u \subseteq L_v$.

Now suppose that $L_v \cap Q \ne \emptyset$, say $w \in L_v \cap Q$. Then by Lemma 14, $v \in S \subseteq L_w$, and so both w, v are in $L_v \cap L_w$, which means $L_v = L_w$, and so $L_v \cap L_u = L_w \cap L_u = L_u \ne \emptyset$, which is a contradiction as discussed just before.

u is large. In this case, the algorithm would reassign $S \leftarrow L_u \cap C \cap U_d$. Then, for all $v \in S$, as $(L_v, V_{\mathcal{T}}[U] \setminus L_v)$ is the latest minimum cut, it must be $L_v \subseteq L_u$, irrespective of whether v is large or not. ◄

Next we prove that when the while-loop ends, either all vertices in S are small, or S corresponds to the cut of the lowest large node on $\mathcal{T}_p^d[C]$.

▶ **Lemma 16.** *If $|C \cap U_d \setminus S| > 2/\phi$, then all vertices in S are small.*

Proof. Consider any vertex $u \in S$. If u is large, then by Lemma 15, $L_u \cap C \cap U_d \subseteq S$, and so $|C \cap U_d \setminus L_u| \geq |C \cap U_d \setminus S| > 2/\phi$, which contradicts the definition of being large. ◄

▶ **Lemma 17.** *After the while-loop ends, if $|C \cap U_d \setminus S| \leq 2/\phi$ and $|Q| > 2/\phi$, then $(L, V_{\mathcal{T}}[U] \setminus L)$ is the latest min-cut of the lowest large node on $\mathcal{T}_p^d[C]$. Moreover, $B = \{u \in S \mid \kappa_u > \kappa\}$ is the set of all vertices u such that $L_u \cap C \cap U_d = S$, and consequently all $L_u, \forall u \in B$ are equal.*

Proof. As the while-loop ends with $|Q| > 2/\phi$, the last iteration must have ended on line-10. Therefore, $(L, V_{\mathcal{T}}[U] \setminus L)$ is the latest min-cut of some $u \in Q$. Suppose otherwise $(L, V_{\mathcal{T}}[U] \setminus L)$ is not the latest min-cut of the lowest large node on the imaginary tree $\mathcal{T}_p^d[C]$. Then, there exists a large vertex $v \in L \cap C \cap U_d$ such that $L_v \cap C \cap U_d \subsetneq S$ but $|C \cap U_d \setminus L_v| \leq 2/\phi$. As $|Q| > 2/\phi$, there must exist $w \in L_v \cap Q$. By Lemma 14, $v \in L \cap C \cap U_d = S = L_w \cap C \cap U_d$, so both v, w are in $L_v \cap L_w$, and consequently $L_v = L_w$, $L_v \cap C \cap U_d = S$, contradiction.

Now let us turn to the second half of the statement. Consider any $u \in B$. $(A_u, V_{\mathcal{T}}[U] \setminus A_u)$ cannot be a min-cut as $\kappa_u > \kappa$. By Lemma 12, u must be a large vertex. On the one hand, by Lemma 15, $L_u \cap C \cap U_d \subseteq S$, and on the other hand, $L_u \cap C \cap U_d$ cannot be strictly smaller than S as S is the lowest already. Hence $L_u \cap C \cap U_d = S$.

For any $u \notin B$, by definition $\kappa_u \leq \kappa$. If u is large, then $\lambda_u < \kappa_u \leq \kappa$, so $L_u \cap C \cap U_d \subsetneq S$, which also contradicts that S corresponds to the lowest large node on $\mathcal{T}_p^d[C]$. ◄

Finally, we prove that all cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ output by the algorithm are latest min-cuts with high probability.

▶ **Lemma 18.** *All cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ output by the algorithm are latest min-cuts with high probability.*

Proof. If the algorithm terminates on line-14, then by Lemma 16, all vertices in W_C are small. So by Lemma 12, $L_u = A_u = K_u, \forall u \in W_C$. Otherwise, if the algorithm terminates on line-21, then by Lemma 17, all vertices in W_C are small. Hence, by Lemma 12, $L_u = A_u = K_u, \forall u \in W_C \setminus B$; also, for any $u \in W_C \cap B$, we have $L_u = L = K_u$. ◄

3.4 Running time analysis

First we analyze the running time of each call of expander search.

▶ **Lemma 19.** *The total running time of the expander search in graph $G_{\mathcal{T}}[U]$ is bounded by $\widetilde{O}(\mathsf{MF}(\mathsf{vol}_G(U), |V_{\mathcal{T}}[U]|)/\phi)$.*

Proof. During the preparation step, each invocation of Lemma 10 induces a set of max-flow instances whose total size is bounded by $\widetilde{O}(|E_{\mathcal{T}}[U]|) = \widetilde{O}(\mathsf{vol}_G(U))$. Since it is repeated for $O(\log n/\phi)$ times, the total time is at most $\widetilde{O}(\mathsf{MF}(\mathsf{vol}_G(U), |V_{\mathcal{T}}[U]|)/\phi)$.

Next, let us analyze the cost of ExploreTree.

▷ **Claim 20.** After each iteration of the while-loop, the value of $|C \cap U_d \setminus S| + |Q|$ always increases by at least 1, so the total number of max-flow instances during the loop is bounded by $O(1/\phi)$.

Proof of claim. If an iteration ends on line-10, Q increases by one while S does not change. If an iteration of the while-loop ends on line-7, then on the one hand, by Lemma 14 we have $Q \subseteq S$; on the other hand, by the pseudo-code, $u \notin Q$ before updating S, Q. Hence, after line-7, $|C \cap U_d \setminus S| + |Q|$ increases by at least 1.

   If an iteration ends on line-12, we claim that before updating S, Q, we have $L_u \cap Q = \emptyset$. In fact, by Lemma 14, for any $w \in Q$, $L_w \cap C \cap U_d = S$. By Lemma 15, as $L_u \cap C \cap U_d \neq S$, it must be $L_u \cap C \cap U_d \subsetneq S = L_w \cap C \cap U_d$. Hence, $w \notin L_u$. As w is arbitrary, we know $Q \cap L_u = \emptyset$. Therefore, after updating $S \leftarrow L_u \cap C \cap U_d$, $|C \cap U_d \setminus S|$ has increased by $|Q|$. Notice that after updating Q, $|Q| = 1$. So $|C \cap U_d \setminus S| + |Q|$ has increased by one.   ◁

   Since each while-loop conducts one max-flow in graph $G_{\mathcal{T}}[U]$, by the above claim, the total cost of the while-loop involves max-flow instances of total size $\widetilde{O}(\mathsf{vol}_G(U)/\phi)$, and the reduction time is dominated by the same amount. After the while-loop, the running time is linear in the size of output, so it is not the bottleneck.   ◀

   Next we analyze the running time during refinement of U.

▶ **Lemma 21.** *The total running time of cutting vertices (the for-loop on line-6 of CondGomoryHu) from U takes total time of $\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(2d \cdot \mathsf{cnt}[i_U] \log^2 n, |V_{\mathcal{T}}[U]|))$.*

Proof. On the one hand, the number of clusters in $\mathcal{C}_{i_U}$ is at most n/s since each cluster has size at least s. So, by Lemma 19, the total time of expander search is $\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(\mathsf{vol}_G(U)))$. By maximality of $d|U_d|$ and $\mathsf{cnt}[i_U]$, we have that:

$$\mathsf{vol}_G(U) \leq 2d|U_d| \log n \leq 2d \cdot \mathsf{cnt}[i_U] \log^2 n$$

   Since the number of edges in $G_{\mathcal{T}}[U]$ is $\mathsf{vol}_G(U)$, the overall time complexity would be $\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(2d \cdot \mathsf{cnt}[i_U] \log^2 n, |V_{\mathcal{T}}[U]|))$.   ◀

   To bound the total time across all different nodes of $\mathcal{T}$ that correspond to the same choice of (s, d), we need the following lemma.

▶ **Lemma 22.** *In any single iteration of the while-loop on line-2 of CondGomoryHu, over all different nodes U of $\mathcal{T}$ that correspond to the same choice of (s, d), we have $\sum_U \mathsf{cnt}[i_U] \leq 4sn/d$.*

Proof. If $s \geq d/4$, then since all such nodes U are packing disjoint subsets of vertices of V, $\sum_U \mathsf{cnt}[i_U] \leq n \leq 4sn/d$. So next we only consider the case where $s < d/4$.

   When $s < d/4$, we can upper bound the total number of vertices in clusters in $\mathcal{C}_{i_U}$ whose degrees in G are within the interval $[d, 2d]$. Take any cluster $C \in \mathcal{C}_{i_U}$ and any vertex $u \in C \cap U_d$. Since $|C| < 2s = d/2$ and G is a simple graph, at least $d/2$ of u's neighbors in G are outside of C. So the crossing edges contributed by u is at least $d/2$. By property (3) of the Definition 8, the total number of crossing edges should be bounded as:

$$\sum_{C \in \mathcal{C}_{i_U}} \mathsf{out}_G(C) \leq \log^7 n \cdot \phi \cdot \sum_{C \in \mathcal{C}_{i_U}} \mathsf{vol}_G(C) \leq \log^7 n \cdot \phi \cdot \sum_{C \in \mathcal{C}_{i_U}} (4s^2 + \mathsf{out}_G(C))$$

$$\leq 4\log^7 n \cdot \phi sn + \log^7 n \cdot \phi \sum_{C \in \mathcal{C}_{i_U}} \mathsf{out}_G(C)$$

As $\phi = \frac{1}{10\log^{c+10}n}$, we have $\sum_{C\in\mathcal{C}_{i_U}}\mathsf{out}_G(C) \leq 8\log^7 n \cdot \phi sn < sn$. As each vertex in $C \cap U_d$ contributes $d/2$ to the above summation, the total number of vertices from U_d in $\mathcal{C}_{i_U}$ is bounded by $2sn/d$. ◄

Combining the above two lemmas gives the following corollary.

▶ **Corollary 23.** *Under Hypothesis 1, the time of dividing all nodes of $\mathcal{T}$ for a single iteration of the while-loop on line-2 in* CondGomoryHu *is bounded by* $\widetilde{O}(n^2)$.

The next thing would be analyzing the total number of rounds of the while-loop. Similar to [3], we first need to prove that with high probability, the number of $u \in C \cup U_d$ such that $\mathsf{vol}_G(K_u) > 0.5\mathsf{vol}_G(U)$ is roughly at most $|U_d|/r$.

▶ **Lemma 24.** *With high probability over the choice of $R \subseteq N$, the total number of $u \in U_d$ such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$ is at most $\frac{4|U_d|\log^2 n}{r}$.*

Proof. The proof is similar to the one in [3]. To avoid confusion, let $\mathcal{T}^{\mathrm{old}}$ be the version of $\mathcal{T}$ before refining with respect to R, and let $\mathcal{T}$ refer to the tree after refinement. For each pair of vertices in the super node $x, q \in N$, define $(\Gamma_x^q, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus \Gamma_x^q)$ to be the latest minimum cut of (x, q) in $G_{\mathcal{T}^{\mathrm{old}}}[N]$. Define M_x^q to be the set of all vertices $y \in N$ such that $x \in \Gamma_y^q$. Basic concentration inequalities show that for any q, x, if $\mathsf{vol}_G(M_x^q) \geq \frac{\log n}{r}\mathsf{vol}_G(N)$, then with high probability, $M_x^q \cap R \neq \emptyset$.

The following claim is a crucial relationship between Γ_u^p and L_u.

▷ Claim 25 (Observation 4.3 in [3]). $\Gamma_u^p \cap U = L_u \cap U$.

Proof of claim. As $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is the latest minimum cut in $G_{\mathcal{T}}[U]$ which is a contracted graph of $G_{\mathcal{T}^{\mathrm{old}}}[N]$ following standard Gomory-Hu steps, $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is a min-cut for (u, p) in $G_{\mathcal{T}^{\mathrm{old}}}[N]$. Since $(\Gamma_u^p, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus \Gamma_u^p)$ is the latest minimum cut in $G_{\mathcal{T}^{\mathrm{old}}}[N]$, we have $\Gamma_u^p \cap U \subseteq L_u \cap U$. Next we only focus on the other direction.

Let $W_1, W_2, \cdots, W_l \subseteq V_{\mathcal{T}^{\mathrm{old}}}[N]$ be all contracted vertices of $V_{\mathcal{T}}[U]$ which are crossed by Γ_u^p; in other words, $\Gamma_u^p \cap W_i \neq \emptyset$ and $W_i \setminus \Gamma_u^p \neq \emptyset$ for all $1 \leq i \leq l$. Since N is refined using pivots from R, according to Lemma 6, we know that for each i there exists a pivot $q_i \in R \cap W_i$ such that $(W_i, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus W_i)$ is a minimum cut for (q_i, p); in fact, $q_i \in R \cap W_i$ are in the neighboring nodes of U in $\mathcal{T}$.

We claim that $q_i \in \Gamma_u^p$; otherwise if $q_i \notin \Gamma_u^p$, as $u \notin W_i$, by Lemma 4, the cut $(X, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus X)$ where $X = \Gamma_u^p \setminus W_i$ is also a minimum cut for (u, p), which contradicts that $(\Gamma_u^p, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus \Gamma_u^p)$ is the latest min-cut.

Construct a new cut $(Y, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus Y)$ where $Y = \Gamma_u^p \cup \bigcup_{i=1}^l W_i$. On the one hand, as $q_i \in \Gamma_u^p, \forall i$, by repeatedly applying Lemma 3 we know $(Y, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus Y)$ is a minimum cut for (u, p) as well; On the other hand, Y does not cross any contracted nodes in $G_{\mathcal{T}}[U]$, so $(Y, V_{\mathcal{T}}[U] \setminus Y)$ is a valid cut in $G_{\mathcal{T}}[U]$ as well. As $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is the latest cut in $G_{\mathcal{T}}[U]$, we know $L_u \cap U \subseteq Y \cap U = \Gamma_u^p \cap U$. This concludes our proof. ◁

Consider the set of all $u \in U_d$ such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$; let them be $u_1, u_2, \cdots, u_l$. By the above claim, it must be $\mathsf{vol}_G(\Gamma_{u_i}^p \cap N) \geq \mathsf{vol}_G(\Gamma_{u_i}^p \cap U) = \mathsf{vol}_G(L_{u_i} \cap U) > 0.5\mathsf{vol}_G(N)$ as well. Therefore, any two sets $\Gamma_{u_i}^p, \Gamma_{u_j}^p$ must intersect. Since $(\Gamma_{u_i}^p, V_{\mathcal{T}^{\mathrm{old}}}[N] \setminus \Gamma_{u_i}^p)$ are latest cuts with respect to the same pivot p, they should form a total order, say $\Gamma_{u_1}^p \subseteq \Gamma_{u_2}^p \subseteq \cdots \Gamma_{u_l}^p$, and so by definition $u_2, u_3, \cdots, u_l \in M_{u_1}^p$.

▷ Claim 26. $M_{u_1}^p \cap R = \emptyset$.

Proof of claim. If $\exists w \in M^p_{u_1} \cap R$, then by definition, $u \in \Gamma^p_w$. As $(\Gamma^p_w, V_{\mathcal{T}^{\text{old}}}[N] \setminus \Gamma^p_w)$ is the latest min-cut for (w, p) in $G_{\mathcal{T}^{\text{old}}}[N]$, any min-cut for (w, p) in $G_{\mathcal{T}^{\text{old}}}[N]$ should contain u on the same side as w. By Lemma 6, u should belong to the part which contains w after the refinement with respect to R, which makes a contradiction as u stays with p in the same part. ◁

By the above lemma, we know $\mathsf{vol}_G(M^p_{u_1}) < \frac{\log n}{r}\mathsf{vol}_G(N)$, and hence we have:

$$d(l-1) \leq \mathsf{vol}_G(M^p_{u_1}) < \frac{\log n}{r}\mathsf{vol}_G(N) \leq \frac{2\log n}{r}\mathsf{vol}_G(U) \leq \frac{4\log^2 n}{r}d|U_d|$$

So $l \leq \frac{4|U_d|\log^2 n}{r}$. ◀

Finally we need to bound the total number of rounds in the while-loop. Call a cluster $C \in \mathcal{C}_{i_U}$ **bad**, if the total number of vertices $u \in C \cap U_d$ such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$ is more than $0.1|C \cap U_d|$; otherwise it is called **good**.

▶ **Lemma 27.** *Consider any invocation of ExploreTree with input parameters U, p, d, C. Suppose cluster C is good, then $|W_C| > 0.8|C \cap U_d|$.*

Proof. First consider the case where ExploreTree terminated on line-14. The while-loop must have terminated on line-7. Then as u is small, $|S| \geq |C \cap U_d| - 2/\phi - |Q| - |C \cap L_u \cap U_d| \geq |C \cap U_d| - 6/\phi$. Therefore, $|W_C| \geq |C \cap U_d| - 6/\phi - 0.1|C \cap U_d| > 0.8|C \cap U_d|$.

Now suppose ExploreTree terminated on line-21. In this case, $C \cap U_d \setminus W_C$ only includes vertices in $C \cap U_d \setminus S$, plus vertices $v \in C \cap U_d$ such that $\mathsf{vol}_G(L_v \cap U) > 0.5\mathsf{vol}_G(N)$. Since C is good, we know $|W_C| \geq |C \cap U_d| - 2/\phi - 0.1|C \cap U_d| > 0.8|C \cap U_d|$. ◀

▶ **Lemma 28.** $\sum_{C \in \mathcal{C}_{i_U} \text{ is bad}} |C \cap U_d| \leq \frac{40|U_d|\log^2 n}{r}$.

Proof. By Lemma 24, the total number of vertices u such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$ is at most $\frac{4|U_d|\log^2 n}{r}$. By definition of badness, we have $\sum_{C \in \mathcal{C}_{i_U} \text{ is bad}} |C \cap U_d| \leq \frac{40|U_d|\log^2 n}{r}$. ◀

▶ **Lemma 29.** *For each node U such that $\mathsf{vol}_G(U) > 0.5\mathsf{vol}_G(N)$, and for each set $K_u = L_u$ which is cut off by our algorithm, we have $\mathsf{vol}_G(K_u \cap U) \leq 0.5\mathsf{vol}_G(N)$. Furthermore, let P be the rest of U after cutting all K_u's. Then $\mathsf{vol}_G(P) \leq (1 - \frac{1}{2\log^2 n})\mathsf{vol}_G(U)$.*

Proof. The first half of the claim is automatically guaranteed by the algorithm. Let us only consider the second half.

By Lemma 27, the total volume that has been cut off from U is at least

$$\sum_{C \in \mathcal{C}_{i_U} \text{ is good}} d|W_C| \geq \sum_{C \in \mathcal{C}_{i_U} \text{ is good}} 0.8d|C \cap U_d| \geq \frac{0.8d}{\log n}|U_d| - 0.8d\sum_{C \in \mathcal{C}_{i_U} \text{ is bad}} |C \cap U_d|$$

$$\geq \frac{0.8d}{\log n}|U_d| - \frac{32d\log^2 n}{r}|U_d| \geq \frac{0.8}{\log^2 n}\mathsf{vol}_G(U) - \frac{32}{\log^3 n}\mathsf{vol}_G(U)$$

$$\geq \frac{1}{2\log^2 n}\mathsf{vol}_G(U)$$

Hence, the volume of $\mathsf{vol}_G(P)$ is reduced by a factor of at most $1 - \frac{1}{2\log^2 n}$. ◀

By Lemma 29, after each round of the while-loop, for each node $U \subseteq N$, either we already have $\mathsf{vol}_G(U) \leq 0.5\mathsf{vol}_G(N)$ after the refinement with respect to random set R, or U is further divided into sub-nodes whose volume are at most $\max\{0.5\mathsf{vol}_G(N), (1 - \frac{1}{2\log^2 n})\mathsf{vol}_G(U)\}$. Therefore the number of rounds is at most $\log^3 n$. So the total running time should be $\widetilde{O}(n^2)$ as well under Hypothesis 1.

4 Unconditional cut-equivalent trees

4.1 The main algorithm

In this section we will prove the second half of Theorem 2 using existing max-flow algorithms. The algorithm is mostly the same as the previous algorithm conditioning on Hypothesis 1, and the extra work is to deal with the additive $n^{1.5}$ term that appears in the running time of max-flow algorithm from [6]. Similar to the previous algorithm, we will also use the same set of parameters ϕ, r, and use degree set $\mathcal{D} = \{\sqrt{n}, 2\sqrt{n}, 2^2\sqrt{n}, \cdots, n\}$.

Preparation

Throughout the algorithm, $\mathcal{T}$ will be the cut-equivalent tree under construction, where each of $\mathcal{T}$'s node will represent a subset of vertices of V. As a preparation step, compute a (ϕ, ϕ)-expander decomposition on G and obtain a partitioning $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ of V. Categorize clusters in $\mathcal{C}$ according to their sizes: for each 2^i, define $\mathcal{C}_i$ to be the set of clusters whose sizes are within interval $[2^i, 2^{i+1})$.

Iteration

In each round, the algorithm tries to simultaneously subdivide all nodes of $\mathcal{T}$ which contains at least $20r$ vertices in V. Following the same procedure as in the previous algorithm, for each node N of $\mathcal{T}$, further refine N into a set of smaller sub-nodes. Then, for each such sub-node U, define variables d, U_d and $s = 2^{i_U}$ accordingly. If (1) $d \geq n^{3/4}$ or (2) $s > n^{3/4}/\sqrt{d}$, we would continue to the do same as in algorithm CondGomoryHu which invokes the expander search procedure.

The unconditional algorithm diverges from the conditional algorithm in Theorem 2 from here if $d < n^{3/4}$ and $s \leq n^{3/4}/\sqrt{d}$. Intuitively, when s is relatively small, the number of expanders whose sizes are roughly s would be large, and so expander searches would be costly because of the additive term $n^{1.5}$ in the running time of computing max-flow. What we would do is to directly apply Lemma 7 on graph $G_\mathcal{T}[U]$ to isolate all vertices in U_d on the tree $\mathcal{T}$ once and for all. The pseudo-code is summarized as GomoryHu.

4.2 Running time analysis

▶ **Lemma 30.** *Each round of the while-loop in GomoryHu takes time $\widetilde{O}(n^{17/8})$.*

Proof. Let us study an arbitrary iteration. Suppose the condition on line-6 holds, namely $d \geq n^{3/4}$ or $s > n^{3/4}/\sqrt{d}$. Then in this case we would do exactly the same as in the conditional algorithm, and the only difference we are invoking the max-flow algorithm from [6]. According to Lemma 21, we could upper bound the running time as

$$\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(2d \cdot \mathsf{cnt}[i_U]\log^2 n, |V_\mathcal{T}[U]|)) = \widetilde{O}(\frac{nd}{s}\mathsf{cnt}[i_U] + \frac{n}{s} \cdot |V_\mathcal{T}[U]|^{1.5})$$

for each node U. Since all tree nodes U are disjoint vertex subsets of V, $\sum_{U \in \mathcal{T}} |V_\mathcal{T}[U]|^{1.5} \leq n^{1.5}$. Therefore, by Lemma 22, this sums to $\widetilde{O}(n^2 + \frac{n^{2.5}}{s})$.

■ **Algorithm 3** GomoryHu($G = (V, E)$).

---

1 initialize a partition tree $\mathcal{T}$, as well as parameters ϕ, r;

2 while $\exists N \subseteq V$, U *a node of* $\mathcal{T}$, $|N| \geq 20r$ **do**

3   **for** *node N of $\mathcal{T}$ with $|N| \geq 20r$* **do**

4   repeat for $10r$ times: each time we sample a vertex $u \in N$ with probability $\frac{\deg_G(u)}{\text{vol}_G(N)}$, and let the sampled set be R;

5   call Lemma 5 on node N with respect to R;

6   **for** *node $U \subseteq N$ of $\mathcal{T}$ such that $\text{vol}_G(U) > 0.5\text{vol}_G(N)$* **do**

7   take d such that $d|U_d|$ is maximized;

8   take $s = 2^{i_U}$ such that $\text{cnt}[i_U]$ is maximized;

9   **if** $d \geq n^{3/4}$ *or* $s > n^{3/4}/\sqrt{d}$ **then**

10   **for** *each $C \in \mathcal{C}_{i_U}$* **do**

11   run expander search on C within node U to compute a subset $W_C \subseteq C \cap U_d$, and the latest min-cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ for each $u \in W_C$;

12   define $W = \bigcup_{C \in \mathcal{C}_{i_U}} W_C$;

13   for each $u \in W$ such that K_u is maximal, split $K_u \cap U$ off of U and create a new node on $\mathcal{T}$;

14   **else**

15   apply Lemma 7 on the auxiliary graph $G_{\mathcal{T}}[U]$ with input parameter $k = 2d$, so that all vertices in U_d become singletons in $\mathcal{T}$;

16 for *node U of $\mathcal{T}$ such that $|U| < 20r$* **do**

17   repeatedly refine U using the generic Gomory-Hu steps until all nodes are singletons;

18 return $\mathcal{T}$ as a cut-equivalent tree;

We first claim that $s \geq \sqrt{2d}$. In fact, by maximality of $\text{cnt}[i_U]$, there exists at least one cluster $C \in \mathcal{C}_{i_U}$ that intersects U_d. Take any $u \in C \cap U_d$. Then since G is a simple graph, more than $d - s$ neighbors of u are outside of C, thus $\text{out}_G(C) > d - s$. By property (3) of Definition 8, we have:

$$\text{out}_G(C) \leq \log^7 n \cdot \phi\text{vol}_G(C) \leq \log^7 n \cdot \phi(4s^2 + \text{out}_G(C))$$

As $\phi = \frac{1}{10\log^{c+10}}$, we have $\text{out}_G(C) \leq 0.4s^2 + 0.1\text{out}_G(C)$, and so $\text{out}_G(C) < 0.5s^2$. As $\text{out}_G(C) > d - s$, we have $s > \sqrt{2d}$.

When $d \geq n^{3/4}$, as $s \geq \sqrt{2d} > n^{3/8}$ we have $\widetilde{O}(n^2 + \frac{n^{2.5}}{s}) = \widetilde{O}(n^{17/8})$. If $d < n^{3/4}$ and $s > n^{3/4}/\sqrt{d}$, then we also bound the total running time as $\widetilde{O}(n^2 + \frac{n}{s\phi} \cdot n^{1.5}) = \widetilde{O}(n^{17/8})$.

Now suppose the condition on line-6 does not hold, then $d < n^{3/4}$ and $s \leq n^{3/4}/\sqrt{d}$. In this case, similar to Lemma 22, we can prove that the total volume $\text{vol}_G(U)$ over all different U's is bounded by $\widetilde{O}(ns)$. So applying Lemma 7 in this round takes time at most $\widetilde{O}(nsd) = \widetilde{O}(n^{17/8})$. ◀

▶ **Lemma 31.** *The total number of rounds of the while-loop in* GomoryHu *is bounded by* $O(\log^3 n)$.

Proof. If each round of the while-loop, if $d \geq n^{3/4}$ or $s > n^{3/4}/\sqrt{d}$ for node U, then according to the proof of Lemma 29, the volume of each subdivision is bounded by $\max\{0.5\mathsf{vol}_G(N), (1 - \frac{1}{2\log^2 n})\mathsf{vol}_G(U)\}$. If $d < n^{3/4}$ and $s \leq n^{3/4}/\sqrt{d}$, then all vertices in U_d become singletons on $\mathcal{T}$; also, and for the same reason, all subdivision of U should be at most $(1 - \frac{1}{2\log^2 n})\mathsf{vol}_G(U)$. Therefore, the number of while-loop iterations is at most $O(\log^3 n)$. ◄

───── **References** ─────

1   Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Gomory-hu tree in subcubic time. *arXiv preprint*, 2021. `arXiv:2111.04958`.

2   Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 48–61. SIAM, 2020.

3   Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory-hu tree in unweighted graphs. *arXiv preprint*, 2020. `arXiv:2012.10281`.

4   Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Apmf< apsp? gomory-hu tree for unweighted graphs in almost-quadratic time. *arXiv preprint*, 2021. `arXiv:2106.02981`.

5   Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster gomory-hu trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3630–3649. SIAM, 2022.

6   Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. *arXiv preprint*, 2021. `arXiv:2101.05719`.

7   Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 11–26. Springer, 2003.

8   Harold N Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 812–821. IEEE Computer Society, 1991.

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

10 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.

11 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021.

12 Frieda Granot and Refael Hassin. Multi-terminal maximum flows in node-capacitated networks. *Discrete applied mathematics*, 13(2-3):157–163, 1986.

13 Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.

14 Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An $o(mn)$ gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 605–614, 2007.

15 Tarun Kathuria, Yang P Liu, and Aaron Sidford. Unit capacity maxflow in almost $m^{4/3}$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130. IEEE, 2020.

16 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\widetilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

17 Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *Annual Symposium on Foundations of Computer Science*, 2020.

18 Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *arXiv preprint*, 2021. `arXiv:2106.02233`.

19 Yang P Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020.

20 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

21 Tianyi Zhang. Gomory-hu trees in quadratic time. *arXiv preprint*, 2021. `arXiv:2112.01042`.