A Nearly Optimal All-Pairs Min-Cuts Algorithm in Simple Graphs

Jason Li
Simons Institute
UC Berkeley
jmli@cs.cmu.edu

Debmalya Panigrahi
Department of Computer Science
Duke University
debmalya@cs.duke.edu

Thatchaphol Saranurak
Computer Science & Engineering Division
University of Michigan Ann Arbor
thsa@umich.edu

***Abstract*—We give an $n^{2+o(1)}$-time algorithm for finding s-t min-cuts for all pairs of vertices s and t in a simple, undirected graph on n vertices. We do so by constructing a Gomory-Hu tree (or cut equivalent tree) in the same running time, thereby improving on the recent bound of $\tilde{O}(n^{2.5})$ by Abboud *et al.* (STOC 2021). Our running time is nearly optimal as a function of n.**

I. Introduction

An s-t mincut is a minimum (weight/cardinality) set of edges in a graph whose removal disconnects two vertices s, t. Finding s-t mincuts, and by duality the value of s-t maxflows, is a foundational question in graph algorithms. Naïvely, mincuts for all vertex pairs can be computed by running a maxflow algorithm separately for each vertex pair, thereby incurring $\Theta(n^2)$ maxflow calls on an n-vertex graph. In 1961, Gomory and Hu [10] gave a remarkable result where they constructed a cut equivalent tree (or Gomory-Hu tree, after its inventors) that captures an s-t mincut for every vertex pair s, t using just $n-1$ maxflow calls. By plugging in the current fastest maxflow algorithm [21], this gives an $\tilde{O}(mn + n^{5/2})$-time[1] algorithm for the all pairs min-cuts (APMC) problem on an n-vertex, m-edge graph. Improving on Gomory and Hu's 60-year old algorithm for the APMC problem on general, weighted graphs remains a major open question in graph algorithms.

For *unweighted* graphs however, we can do better. The first algorithm to do so was by Bhalgat *et al.* [6], who used Steiner mincuts to obtain a running time of $\tilde{O}(mn)$ in unweighted graphs. Karger and Levine [13] matched this bound using the same counting technique, but by a different algorithm based on randomized maxflow computations. In simple graphs, both these algorithms obtain a running time of $\tilde{O}(n^3)$ since $m = O(n^2)$. The first subcubic (in n) running time was recently obtained in a beautiful work by Abboud *et al.* [4], who achieved a running time of $\tilde{O}(n^{2.5})$ for simple graphs. They write: "Perhaps the most interesting open question is whether $\tilde{O}(m)$ time can be achieved, even in simple graphs and even assuming a linear-time maxflow algorithm." Interestingly, they also isolate why breaking the $n^{2.5}$ bound is challenging, and say: "...perhaps it will lead to the first conditional lower bound for computing a Gomory-Hu tree."

In this paper, we give an $n^{2+o(1)}$-time Gomory-Hu tree algorithm in simple graphs, thereby improving on the $\tilde{O}(n^{2.5})$ bound of Abboud *et al.* Our result is *unconditional* – specifically, we do not need to assume an $\tilde{O}(m)$-time maxflow algorithm. As a consequence, we also refute the possibility of a $n^{2.5}$ lower bound for the Gomory-Hu tree problem. Since there are $\binom{n}{2} = \Theta(n^2)$ vertex pairs, the running time of our algorithm is near-optimal for the all-pair mincuts problem. Even if one were to only construct a Gomory-Hu tree (and not report the mincut values explicitly for all vertex pairs), our algorithm is near-optimal as a function of n since m can be $\Theta(n^2)$.

Our main theorem is the following:

Theorem I.1. *There is an algorithm GHTree(G) that, given a simple n-vertex m-edge graph G, with high probability computes a Gomory-Hu tree of G in $n^{2+o(1)}$ time.*

Our techniques also yield a faster Gomory-Hu tree algorithm in sparse graphs. The previous record for sparse graphs is due to another recent algorithm of Abboud *et al.* [2] that takes $O(mc + \sum_{i=1}^{m/c} T(m, n, F_i))$ time, where $\sum_i F_i = O(m)$ and $T(m, n, F_i)$ is the time complexity for computing a maxflow of value at most F_i. (Here, c is a parameter that can be chosen by the algorithm designer to optimize the bound.) We improve this bound in the following theorem to $\tilde{O}(mc) + \frac{n^{1+o(1)}}{c} \cdot T(m, n)$ where $T(m, n)$ is the time complexity for computing a maxflow. For comparison, if we assume an $\tilde{O}(m)$-time maxflow algorithm, then the running time

The full version of this paper is [17]. DP supported in part by NSF awards CCF-1750140 (CAREER) and CCF-1955703, and ARO award W911NF2110230.

[1] $\tilde{O}(\cdot)$ suppresses poly-logarithmic factors.

improves from $\tilde{O}(m^{1.5})$ in [2] to $mn^{0.5+o(1)}$ in this paper. Using existing maxflow algorithms [14], [21], the bound is $\tilde{O}(m \cdot g(m,n))$ in [2] where $g(m,n) = \min(m^{1/2}n^{1/6}, m^{1/2}+n^{3/4})$, and improves to $\sqrt{mn} \cdot n^{o(1)} \cdot g(m,n)$ in this paper.

Theorem I.2. *There is an algorithm GHTreeSparse(G) that, given a simple n-vertex m-edge graph G, with high probability computes a Gomory-Hu tree of G in $\tilde{O}(mc) + \frac{n^{1+o(1)}}{c} \cdot T(m,n))$ time where $T(m,n)$ denotes the time complexity for computing a maximum flow on an n-vertex m-edge graph and c is a parameter that we can choose.*

Related Work: Gusfield [11] gave a Gomory-Hu tree algorithm that simplifies Gomory and Hu's algorithm, particularly from an implementation perspective, although it did not achieve an asymptotic improvement in the running time. If one allows a $(1+\epsilon)$ approximation, then faster algorithms are known; in fact, the problem can be solved using (effectively) polylog(n) maxflow calls [1], [16]. Finally, there is a robust literature on Gomory-Hu tree algorithms for special graph classes. This includes near-linear time algorithms for the class of planar graphs [8] and more generally, for surface-embedded graphs [7], as well as improved runtimes for graphs with bounded treewidth [5], [1]. For more discussion on the problem, the reader is referred to a survey in the Encyclopedia of Algorithms [20].

Concurrent to our work and independent of it, there were two other teams of researchers who achieved similar results to those presented in this paper. In a paper presented at the same conference as this paper (FOCS 2021), Abboud, Krauthgamer, and Trabelsi [3] obtained a Gomory-Hu tree algorithm for simple graphs that also runs in $O(n^{2+o(1)})$ time, thereby matching our result. In another independent work, Zhang [22] also obtained the same result, but under the additional assumption of a near-linear time max-flow algoithm. Without this assumption, the running time of Zhang's algorithm is $\tilde{O}(n^{17/8})$.

Organization: In Section II, we introduce the tools that we need for our Gomory-Hu tree algorithm. We then give the Gomory-Hu tree algorithm using these tools, and prove Theorem I.1 and Theorem I.2. In subsequent sections, we show how to implement each individual tool and establish their respective properties.

II. Gomory-Hu Tree Algorithm

We start this section by defining Gomory-Hu trees. It will be convenient to also define *partial* Gomory-Hu trees which will play an important role in our algorithm.

Definition II.1 (Partial Gomory-Hu trees)**.** Let $G = (V, E)$ be a graph. A *partial Gomory-Hu tree* or simply a *partial tree* $(T, \mathcal{P})$ of G satisfies the following:

- T is a tree on $V(T) \subseteq V$ called a *terminal* set,
- $\mathcal{P}$ is a partition of V where each part $V_u \in \mathcal{P}$ contains exactly one terminal u,
- for any pair of terminals $u, v \in V(T)$, a u-v mincut (A_T, B_T) in T corresponds to a u-v mincut (A, B) in G where $A = \bigcup_{x \in A_T} V_x$ and $B = \bigcup_{y \in B_T} V_y$.

If $V(T) = V$, then T is a *Gomory-Hu tree* of G.

Terminology about Partial Trees: Let $X \subseteq V$ be a vertex set. We say that a partial tree $(T, \mathcal{P})$ *captures all mincuts separating X of size at most d* if, for every part $U \in \mathcal{P}$ and every pair of vertices $u, v \in U \cap X$, $\text{mincut}_G(u,v) > d$. When $X = V$, we say that $(T, \mathcal{P})$ *captures all mincuts of size at most d*. If all edges of T have weight at most d, then we say that $(T, \mathcal{P})$ *captures no mincut of size more than d*.

We say that $(T', \mathcal{P}')$ is a *refinement* of $(T, \mathcal{P})$ if $(T, \mathcal{P})$ can be obtained from $(T', \mathcal{P}')$ by contracting subtrees of T' and taking the union of the corresponding parts of $\mathcal{P}'$. In other words, a refinement adds edges while preserving the properties of a partial tree. The classic algorithm of Gomory and Hu [10] starts with a vacuous partial tree comprising a single node and refines it in a series of $n-1$ iterations, where each iteration adds a single edge to the tree. Our goal is to refine the partial tree faster by adding multiple edges in a single iteration.

Well-linked Decomposition: The key to defining a single iteration of our algorithm that refines a partial tree is the notion of a well-linked decomposition. We first define a *well-linked* set of vertices.

Definition II.2. We say that a vertex set X is (d, ϕ)-well-linked in a graph G if

- For each $v \in X$, $\deg_G(v) \geq d$, where $\deg_G(v)$ is the degree of vertex v in graph G, and
- For each partition (A, B) of X, $\frac{\text{mincut}_G(A,B)}{d \cdot \min\{|A|,|B|\}} \geq \phi$. Here, $\text{mincut}_G(A,B)$ is the smallest cut of G that has vertex subsets A and B on different sides of the cut.

The next lemma is an important technical contribution of our paper, and says that the set of high-degree vertices can be partitioned into a small number of well-linked sets. Actually, this is the only

place in this paper where we require that the input graph G is a simple graph.

Lemma II.3. *There is an algorithm* $\textsc{Partition}(G, d)$ *that, given a simple* n*-vertex* m*-edge graph* G *and a parameter* d*, returns with high probability a partition* $\{X_1, \ldots, X_k\}$ *of* $V_{\geq d} = \{v \mid \deg_G(v) \geq d\}$ *such that* $k = \tilde{O}(n/d)$ *and every set* X_i *is* (d, ϕ_{part})*-well-linked in* G*, where* $\phi_{\text{part}} = n^{-o(1)}$*. The algorithm* $\textsc{Partition}(G, d)$ *runs in* $m^{1+o(1)}$ *time.*

In a single iteration, our goal is to refine a partial tree that captures mincuts of size at most d to one that captures mincuts of size at most $2d$. For this, we would like to partition all the vertices in $V_{\geq d}$ using the above lemma, and repeatedly refine the partial tree so that it captures all mincuts of size at most $2d$ separating the terminal set that includes the vertices in the (d, ϕ)-well-linked set X_i. But, doing this on the input graph G would be too slow; instead, we use a sparse connectivity certificate that preserves all cuts of size at most $3d$. This suffices since in this iteration, we only seek to capture cuts of size at most $2d$.

Connectivity Certificate: We formally define connectivity certificates next.

Definition II.4. For any graph $G = (V, E)$, a *k-connectivity certificate* H of G is a subgraph of G that preserves all cuts in G of size $< k$, and ensures that all cuts in G of size $\geq k$ have size $\geq k$ in H as well. In other words, for any cut $(S, V \setminus S)$, we have $|E_H(S, V \setminus S)| \geq \min\{|E_G(S, V \setminus S)|, k\}$.

The next lemma, due to Nagamochi and Ibaraki [19], gives an efficient algorithm for obtaining a connectivity certificate.

Lemma II.5 ([19])**.** *There is an algorithm* $\textsc{Sparsify}(G, k)$ *that, given an* n*-vertex* m*-edge graph* G *and a parameter* k*, return a* k*-connectivity certificate* H *of* G *with at most* $\min\{m, nk\}$ *edges in* $O(m)$ *time.*

The Main Lemma: We are now ready to state our main lemma, which constitutes a refinement of the partial tree.

Lemma II.6. *There is an algorithm* $\textsc{Refine}(G, H, (T, \mathcal{P}), X, d, \phi)$ *that given*

- *graph* G *on* n *vertices and* m *edges, and a* $3d$*-connectivity certificate* H *of* G *with* $m' \leq \min\{m, 3nd\}$ *edges,*
- *a partial tree* $(T, \mathcal{P})$ *of* G *that captures all mincuts of size at most* d *and no mincut of size more than* $2d$*, and*
- *a set* X *that is* (d, ϕ)*-well-linked in* H*,*

in $\tilde{O}(\min\{m, nd\}/\phi)$ *time plus max-flow calls on several graph instances with* $\tilde{O}(n/\phi)$ *vertices and* $\tilde{O}(\min\{m, nd\}/\phi)$ *edges in total, returns with high probability a partial tree* $(T', \mathcal{P}')$ *of* G *where*

- $(T', \mathcal{P}')$ *is a refinement of* $(T, \mathcal{P})$*, and*
- $(T', \mathcal{P}')$ *captures all mincuts separating* $X \cup V(T)$ *of size at most* $2d$ *and no mincut of size more than* $2d$*.*

Crucially, when $3nd \leq m$, the running time in the above lemma does not depend on m, the number of edges in G. In other words, the algorithm does not even read in the entire graph G, instead operating on the $3d$-connectivity certificate H directly.

Small Connectivities: Recall that in a single iteration, Lemma II.3 produces $\tilde{O}(n/d)$ sets each of which is (d, ϕ_{part})-well-linked, and Lemma II.6 makes max-flow calls on graphs with $\tilde{O}(n/\phi_{\text{part}})$ vertices and $\tilde{O}(nd/\phi_{\text{part}})$ edges in total. The current fastest max flow algorithm gives the following runtime:

Theorem II.7 ([21])**.** *There is an algorithm that can find, with high probability, a maximum flow on a graph with* n *vertices and* m *edges in* $\tilde{O}(m + n^{1.5})$ *time.*

Using this algorithm, the runtime of the max flow calls in an iteration becomes $\tilde{O}(\frac{n}{d}) \cdot \tilde{O}(nd + n^{1.5}) \cdot n^{o(1)} = (n^2 + \frac{n^{2.5}}{d}) \cdot n^{o(1)}$ (recall that $\phi_{\text{part}} = n^{-o(1)}$ in Lemma II.3). While this suffices for $d \geq \sqrt{n}$, we need an additional trick to handle small connectivities, namely $d < \sqrt{n}$.

The next theorem, due to Hariharan *et al.* [12] and Bhalgat *et al.* [6], gives a fast algorithm for computing a partial tree that captures all small cuts:

Theorem II.8 ([12], [6])**.** *There is an algorithm* $\textsc{SmallConn}(G, d)$ *that, given a simple* n*-vertex* m*-edge graph* G *and a parameter* d*, returns with high probability a partial tree* $(T, \mathcal{P})$ *that captures all cuts of size at most* d *in* $\tilde{O}(\min\{md, m + nd^2\})$ *time.*

If we set $d = \sqrt{n}$, then this theorem gives a partial tree that captures all cuts of size at most $\sqrt{n}$ in $\tilde{O}(n^2)$ time. We initialize our algorithm with this partial tree, and then run the iterative refinement process described above for $d = \sqrt{n}, 2\sqrt{n}, \ldots, n/2, n$ to obtain the Gomory-Hu tree. We formally describe this algorithm below and prove its correctness and runtime bounds.

The Gomory-Hu Tree Algorithm: The algorithm is given in Algorithm 1. We first establish correctness of the algorithm. The next

Algorithm 1: GHTREE(G)

1) Initialize $(T, \mathcal{P}) \leftarrow$ SMALLCONN(G, c) where c is a parameter we can choose.
2) For $d = c, 2c, \ldots, n/2, n$
   a) $H \leftarrow$ SPARSIFY($G, 3d$)
   b) $\{X_1, \ldots, X_{\tilde{O}(n/d)}\} \leftarrow$ PARTITION(H, d)
   c) For each X_i, $(T, \mathcal{P}) \leftarrow$ REFINE($G, H, (T, \mathcal{P}), X_i, d, \phi_{\text{part}}$)
3) Return T

property formalizes the progress made by the algorithm in a single iteration of the for loop.

Lemma II.9. *At the beginning of each for-loop iteration of Algorithm 1, if $(T, \mathcal{P})$ is a partial tree of G that captures all mincuts of size at most d and no mincut of size more than d, then at the end of the iteration, $(T, \mathcal{P})$ captures all mincuts of size at most $2d$ and no mincut of size more than $2d$.*

Proof: First, observe that the input to REFINE($\cdot$) is valid: (1) H is a $3d$-connectivity certificate of G containing $\leq \min\{m, 3nd\}$ edges by Lemma II.5, (2) $(T, \mathcal{P})$ is a partial tree of G that captures all mincuts of size at most d and no mincut of size more than $2d$ by assumption, and (3) X is (d, ϕ_{part})-well-linked in H by Lemma II.3.

By the second property in Lemma II.6, $(T, \mathcal{P})$ captures no mincut of size more than $2d$. It remains to show that at the end of the iteration, $(T, \mathcal{P})$ captures all mincuts of size at most $2d$. For mincuts of size at most d, this follows from the assumption. Consider an s-t mincut of size more than d but at most $2d$. Since $s, t \in V_{\geq d}$ in G, it follows that $s, t \in V_{\geq d}$ in H as well. Thus, $s, t \in \cup_i X_i$ produced by PARTITION(H, d). There are two cases. If $s, t \in X_i$ for some i, then Lemma II.6 ensures that the s-t mincut is captured by $(T, \mathcal{P})$ after REFINE($G, H, (T, \mathcal{P}), X_i, d, \phi_{\text{part}}$). If $s \in X_i$, $t \in X_j$ where $i < j$ (wlog), then, when we call REFINE($G, H, (T, \mathcal{P}), X_j, d, \phi_{\text{part}}$), we have $s \in V(T)$ and $t \in X_j$. Again, by Lemma II.6, the s-t mincut is captured by $(T, \mathcal{P})$ after the call to REFINE. ∎

The following is a simple corollary of the above lemma.

Lemma II.10. *Algorithm 1 computes a Gomory-Hu tree T.*

Proof: First, note that $(T, \mathcal{P}) \leftarrow$ SMALLCONN(G, c) captures all mincuts in G of size at most c by Theorem II.8. Therefore, by Lemma II.9, at the end of each iteration of the for loop, Algorithm 1 captures all mincuts of size at most $2d$. As a consequence, at the end of the final loop, Algorithm 1 captures all mincuts of size at most n. Therefore, T is indeed a Gomory-Hu tree. ∎

We now establish the running time of Algorithm 1.

Lemma II.11. *By choosing $c = \sqrt{n}$, Algorithm 1 takes $n^{2+o(1)}$ time.*

Proof: SMALLCONN(G, c) takes $\tilde{O}(m + nc^2) = \tilde{O}(n^2)$ time by Theorem II.8. For each of the $O(\log n)$ iterations, SPARSIFY($G, 3d$) takes $O(m)$ time (by Lemma II.5) and PARTITION(G, d) takes $m^{1+o(1)} = n^{2+o(1)}$ time (by Lemma II.3). Since H has $O(nd)$ edges and X_i is (d, ϕ_{part})-well-linked, REFINE($G, H, (T, \mathcal{P}), X_i, d, \phi_{\text{part}}$) takes $(\min\{m, nd\} + n^{1.5}) \cdot n^{o(1)} \leq (nd + n^{1.5}) \cdot n^{o(1)}$ time by Lemma II.6 and Theorem II.7.[2] Since there are at most $\tilde{O}(n/d)$ well-linked sets X_i, the total time spent on REFINE is $(n^2 + n^{2.5}/d) \cdot n^{o(1)} = n^{2+o(1)}$ since $d \geq c = \sqrt{n}$. The lemma follows by summing the time over all iterations. ∎

By analyzing the time differently, we obtain the following.

Lemma II.12. *For any parameter c, Algorithm 1 takes $\tilde{O}(mc) + \frac{n^{1+o(1)}}{c} \cdot T(m, n)$ time where $T(m, n)$ denotes the time complexity for computing a maximum flow on an n-vertex m-edge graph.*

Proof: SMALLCONN(G, c) takes $\tilde{O}(mc)$ time. For each of the $O(\log n)$ iterations, SPARSIFY($G, 3d$) takes $O(m)$ time (by Lemma II.5) and PARTITION(G, d) takes $m^{1+o(1)} = m \cdot n^{o(1)}$ time (by Lemma II.3). Also, REFINE($\cdot$) takes $(\min\{m, nd\} + T(m, n)) \cdot n^{o(1)} \leq (m + T(m, n)) \cdot n^{o(1)} \leq T(m, n) \cdot n^{o(1)}$ time by Lemma II.6.[3] Since there are at most $\tilde{O}(n/d) = \tilde{O}(n/c)$ well-linked sets X_i, the total time spent on REFINE is $\frac{n^{1+o(1)}}{c} \cdot T(m, n)$. The lemma follows by summing the time over all iterations. ∎

To conclude, observe that Theorem I.1 follows from Lemma II.10 and Lemma II.11. Similarly, Theorem I.2 follows from Lemma II.10 and Lemma II.12.

[2]Note that since the running time is convex and each graph has at most nd edges and n vertices, the worst case is when there are $n^{o(1)}$ maxflow calls on graphs with nd edges and n vertices.

[3]Note that since the running time is convex and each graph has at most m edges and n vertices, the worst case is when there are $n^{o(1)}$ maxflow calls on graphs with m edges and n vertices.

III. Refinement with Well-linked Set

Our goal in this section is to prove the main lemma (Lemma II.6). Let us first recall the setting of the lemma. We have a graph $G = (V, E)$ with n vertices and m edges and a $3d$-connectivity certificate H of G containing $m' \leq \min\{m, 3nd\}$ edges. Let $(T, \mathcal{P})$ be a partial tree of G that captures all mincuts of size at most d and no mincut of size more than $2d$. Let X be a (d, ϕ)-well-linked set in H. For each terminal $u_i \in V(T)$ and its corresponding part $V_i \in \mathcal{P}$, let $X_i = V_i \cap X$.

Now, we define the *sparsified auxiliary graph* H_i. For each connected component C in $T \setminus \{u_i\}$, let $V_C = \bigcup_{u \in V(C)} V_u$ where each $V_u \in \mathcal{P}$. The graph H_i is obtained from H by contracting V_C into one vertex u_C for every component C in $T \setminus \{u_i\}$. Let n'_i and m'_i denote the number of vertices and edges in H_i respectively. (H_i is unweighted but not necessarily a simple graph.) Below, we bound the total size of H_i over all i. The bound on $\sum_i m'_i$ crucially exploits the fact that the graph is unweighted.

Proposition III.1. *$\sum_{u_i \in V(T)} n'_i \leq 3n$ and $\sum_{u_i \in V(T)} m'_i \leq \min\{3m, 5nd\}$.*

Proof: Observe that $n'_i = |V_i| + \deg_T(u_i)$. So $\sum_{u_i \in V(T)} n'_i = n + 2|V(T)| \leq 3n$. Next, we bound $\sum_{u_i \in V(T)} m'_i$. For any vertex $x \in V$, let $\mathrm{rep}(x) \in V(T)$ be the unique terminal such that $x \in V_{\mathrm{rep}(x)}$. Consider each edge $(x, y) \in E(H)$. Let $P_{xy} = (\mathrm{rep}(x), \ldots, \mathrm{rep}(y)) \subseteq V(T)$ be the unique path in T between $\mathrm{rep}(x)$ and $\mathrm{rep}(y)$. (Possibly x and y are in the same part of $\mathcal{P}$ and so $\mathrm{rep}(x) = \mathrm{rep}(y)$.) The crucial observation is that an edge (x, y) appears in H_i if and only if the terminal u_i is in P_{xy} (otherwise, x are y are contracted into one vertex in H_i). That is, the contribution of (x, y) to $\sum_{u_i \in V(T)} m'_i$ is exactly $|V_T(P_{xy})| = 1 + |E_T(P_{xy})|$. Summing over all edges $e \in E(H)$, this implies that

$$\sum_{u_i \in V(T)} m'_i \leq |E(H)| + \sum_{(x,y) \in E(H)} |E_T(P_{xy})|.$$

Recall that $|E(H)| = m' \leq \min\{m, 3nd\}$. The last important observation is that $\sum_{(x,y) \in E(H)} |E_T(P_{xy})|$ is exactly the total weight of edges in T. This is because each $(x, y) \in E(H)$ contributes exactly one unit of weight to each tree-edge in $E_T(P_{xy})$. The total weight of edges in T is at most $\min\{2m, 2nd\}$. To see this, observe that it is at most $(|V(T)| - 1) \cdot 2d \leq 2nd$, because T has no edge with weight more than $2d$. Also, it is at most $\sum_{u_i \in V(T)} \deg_G(u_i) \leq 2m$ because each tree edge $(u_i, u_j) \in E(T)$ has weight $\mathrm{mincut}_G(u_i, u_j) \leq \min\{\deg_G(u_i), \deg_G(u_j)\}$. This implies the bound $\sum_{u_i \in V(T)} m'_i \leq \min\{3m, 5nd\}$ as claimed. ∎

The key step for proving Lemma II.6 is captured by the following lemma.

Lemma III.2. *Given H_i, X_i, and $(T, \mathcal{P})$, there is an algorithm that takes $\tilde{O}(m'_i/\phi)$ time and additionally makes max-flow calls on several graphs with $\tilde{O}(n'_i/\phi)$ vertices and $\tilde{O}(m'_i/\phi)$ edges in total, and then returns a partial tree $(T'_i, \mathcal{P}'_i)$ **of G** such that*

- *$(T'_i, \mathcal{P}'_i)$ is a refinement of $(T, \mathcal{P})$, and*
- *$(T'_i, \mathcal{P}'_i)$ captures all mincuts separating $X_i \cup V(T)$ of size at most $2d$ and no mincut of size more than $2d$.*

Before proving Lemma III.2, we show that it implies Lemma II.6. (See Figure 1 for illustration.)

Proof of Lemma II.6. : We apply Lemma III.2 for all i simultaneously and obtain $(T'_i, \mathcal{P}'_i)$ each of which refines $(T, \mathcal{P})$ in exactly one part $V_i \in \mathcal{P}$. Let $(T', \mathcal{P}')$ be the refinement of $(T, \mathcal{P})$ such that $(T', \mathcal{P}')$ refines the part $V_i \in \mathcal{P}$ according to $(T'_i, \mathcal{P}'_i)$ for every i. Note that $(T', \mathcal{P}')$ can be computed in $O(n)$ time. Clearly, $(T', \mathcal{P}')$ captures no mincut of size more than $2d$ (i.e., T' has no edge of weight more than $2d$) because none of T'_i does.

It remains to prove that $(T', \mathcal{P}')$ captures all mincuts separating $V(T) \cup \left(\bigcup_i X_i\right) = V(T) \cup X$ of size at most $2d$. That is, there is no pair $x, y \in V(T) \cup X$ where $\mathrm{mincut}_G(x, y) \leq 2d$ and $x, y \in \mathcal{P}'$ are in the same part. This is true because, if x and y are from a different part of $\mathcal{P}$, then they are still from a different part in $\mathcal{P}'$ as $\mathcal{P}'$ is a refinement of $\mathcal{P}$. Otherwise, if x and y are from the same part of $\mathcal{P}$, say $V_i \in \mathcal{P}$, then $x, y \in X_i \cup V(T)$ and so Lemma III.2 guarantees that they must be separated by $\mathcal{P}'_i$ and hence in $\mathcal{P}'$. This concludes the correctness of Lemma II.6.

Next, we analyze the running time. The total running time is $\sum_i \tilde{O}(m'_i/\phi) = \tilde{O}(\min\{m, nd\}/\phi)$ by Proposition III.1 and Lemma III.2. Finally, the graphs that the algorithm makes max-flow calls on contain in total at most $\sum_i \tilde{O}(n'_i/\phi) = \tilde{O}(n/\phi)$ vertices and $\sum_i \tilde{O}(m'_i/\phi) = \tilde{O}(\min\{m, nd\}/\phi)$ edges by Proposition III.1. This completes the proof.

Proof of Lemma III.2. : For the remaining part of this section, we prove Lemma III.2. There are two main ingredients.

First, we show that the problem of creating a partial tree on a set of terminals can be reduced to finding single source connectivity on the terminals. This step closely mirrors [16]: while they focus on the *approximate* Gomory-Hu tree problem, their

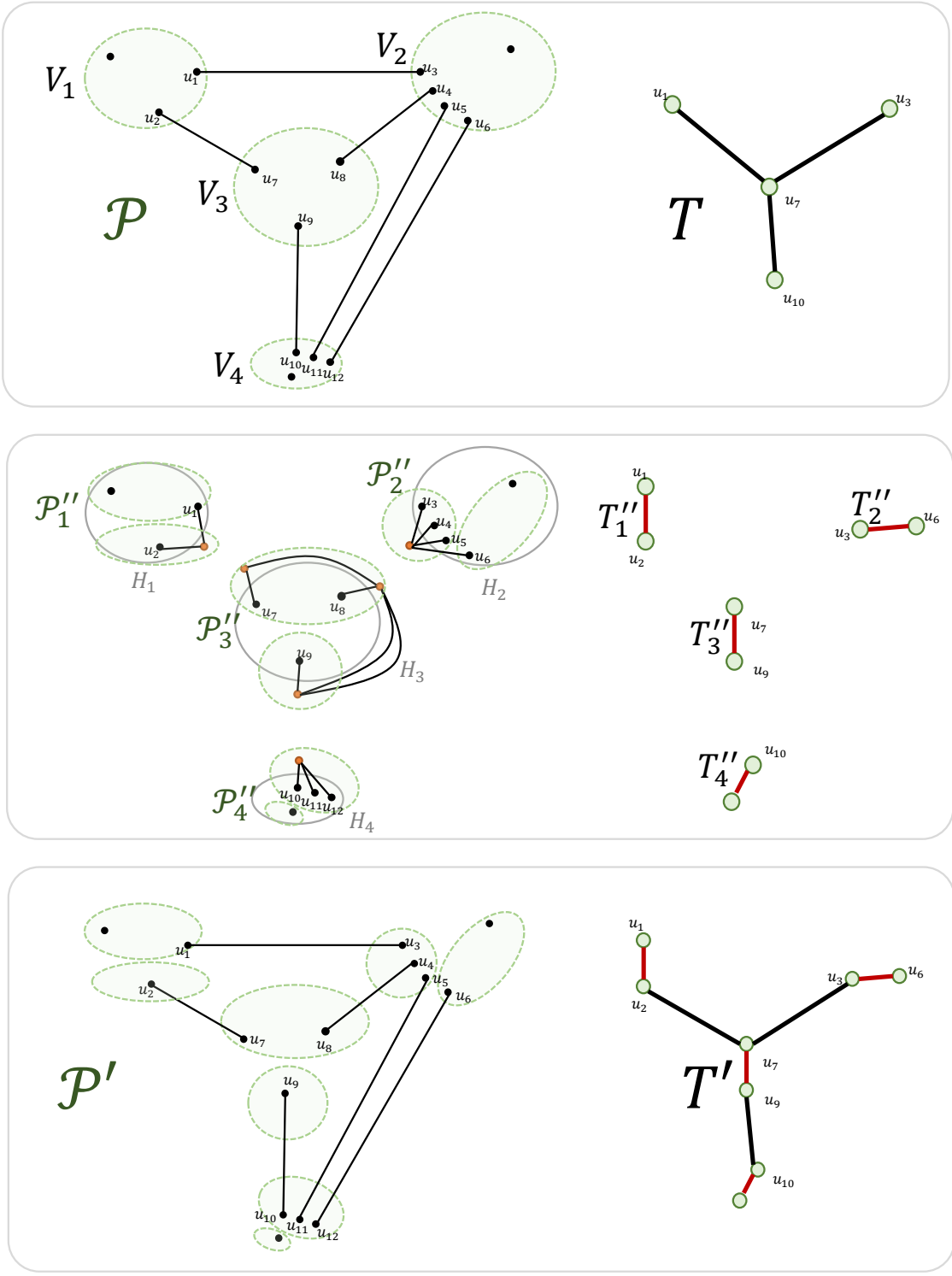

Figure 1. Refining $(T, \mathcal{P})$ to $(T', \mathcal{P}')$. The algorithm for Lemma III.2 computes partial trees $(T_i'', \mathcal{P}_i'')$ of every sparsified auxiliary graph H_i. This is illustrated in the second box in the figure above. The algorithm in Lemma III.2 computes a partial tree $(T_i', \mathcal{P}_i')$ of G which is a refinement of $(T', \mathcal{P})$ in (not shown in the figure above). Then, in the proof of Lemma II.6, we "combine" these refinement on each part V_i and we obtain the refined partial tree $(T', \mathcal{P}')$ of $(T, \mathcal{P})$.

techniques translate over to the exact case. (See details in full version [17].)

Lemma III.3. *Let $G = (V, E)$ be an n-vertex m-edge unweighted (respectively, weighted) graph with a terminal set $X \subseteq V$, and let $k \geq 0$ be a real number. Suppose we have an oracle that, given a terminal $p \in X$, returns $\min\{\text{mincut}_G(p, v), k\}$ for all other terminals $v \in X$. Then, there is an algorithm that computes with high probability a partial tree $(T, \mathcal{P})$ of G where $V(T) \subseteq X$ that captures all mincuts separating X of size at most k and no mincuts of size more than k. It makes calls to the oracle and max-flow on unweighted (respectively, weighted) graphs with a total of $\tilde{O}(n)$ vertices and $\tilde{O}(m)$ edges, and runs for $\tilde{O}(m)$ time outside of these calls.*

Note that it is crucial for us that the reduction above works even when the oracle only returns $\min\{\text{mincut}_G(p, v), k\}$ and not $\text{mincut}_G(p, v)$. The next lemma exactly implements this oracle:

Lemma III.4. *Let $G = (V, E)$ be an n-vertex m-edge graph. Let X be a (d, ϕ)-well-linked set in G. Let $p \in X$ be any fixed vertex in X. Then, there is an algorithm that computes $\min\{\text{mincut}_G(p, v), 2d\}$ for all other $v \in X$ in $O(\frac{m \log n}{\phi})$ time plus $\frac{\text{polylog}(n)}{\phi}$ max-flow calls each on a graph with $O(n)$ vertices and $O(m)$ edges.*

We will prove Lemma III.4 in Section III-A. First, we show how to apply both lemmas above to prove Lemma III.2. We start with a simple observation.

Proposition III.5. *X_i is (d, ϕ)-well-linked in H_i.*

Proof: As X is (d, ϕ)-well-linked in H, any subset $X_i \subseteq X$ is also (d, ϕ)-well-linked in H. Now, observe that the property that a vertex set is (d, ϕ)-well-linked is preserved under graph contraction. As H_i is a contracted graph of H, the proposition follows. ∎

By setting parameters $G \leftarrow H_i$ and $X \leftarrow X_i$ as the inputs of Lemma III.4, we obtain the required oracle for Lemma III.3 when $k = 2d$. By applying Lemma III.3, we obtain a partial tree $(T_i'', \mathcal{P}_i'')$ of H_i where $V(T_i'') \subseteq X$, that captures all mincuts separating X_i of size at most $2d$ and no mincuts of size more than $2d$. This steps takes $\tilde{O}(m_i'/\phi)$ time and makes max-flow calls on several graphs with $\tilde{O}(n_i'/\phi)$ vertices and $\tilde{O}(m_i'/\phi)$ edges in total.

We are not quite done as we need a partial tree of G (not of H_i) with all properties required by Lemma III.2, but the remaining steps are quite easy. Suppose the vertex u_i, which was the unique terminal in V_i in the partial tree $(T, \mathcal{P})$, is now in part $V_{u_i} \subseteq V(H)$ of the partition $\mathcal{P}_i''$. Moreover, let $x_{u_i} \in X_i \cap V(T_i'')$ denote the unique terminal of part $V_{u_i} \in \mathcal{P}_i''$. The algorithm just checks if $\text{mincut}_{H_i}(u_i, x_{u_i}) \leq 2d$ by using a single max-flow call on H_i. If so, we further refine $(T_i'', \mathcal{P}_i'')$ according to the mincut separating u_i and x_{u_i}. If not, then we let u_i replace x_{u_i} as a unique terminal of part $V_{u_i} \in \mathcal{P}_i''$. At this point, $(T_i'', \mathcal{P}_i'')$ is a partial tree of H_i that captures all mincuts separating $X_i \cup \{u_i\}$ of size at most $2d$ and no mincuts of size more than $2d$. Finally, we refine the part V_i of $(T, \mathcal{P})$ according to $(T_i'', \mathcal{P}_i'')$ and obtain a partial tree $(T_i', \mathcal{P}_i')$ of G as desired. The reason this is correct is because $(T_i'', \mathcal{P}_i'')$ captures only mincuts of size at most $2d$ but H preserves exactly *all* cuts of G of size at most $3d$. The running times in these final steps are subsumed by the previous steps. This completes the proof of Lemma III.2.

A. Single-source Mincut Values for Well-linked Sets: Proof of Lemma III.4

We recall the setting of Lemma III.4. We have an n-vertex m-edge graph G and a (d, ϕ)-well-linked set X in G. Let $p \in X$ be any fixed vertex in X. The goal is to compute $\min\{\text{mincut}_G(p, v), 2d\}$ for all other $v \in X \setminus \{p\}$.

Now, we need to introduce some notation. We say that a cut (A, B) in G is an (S, T)-cut if $S \subseteq A$ and $T \subseteq B$. Moreover, (A, B) is an (S, T)-mincut if, additionally, $|E_G(A, B)| = \text{mincut}_G(S, T)$. We say that (A, B) is *the (unique) S-minimal* (S, T)-mincut if, for any (S, T)-mincut (A', B'), we have $S \subseteq A \subseteq A'$. A key tool in proving Lemma III.4 is the following *Isolating Cuts Lemma* of Li and Panigrahi [15], which was discovered independently by Abboud, Krauthgamer, and Trabelsi [4].

Lemma III.6 (Isolating Cut Lemma [15], [4])**.** *There is an algorithm that, given an undirected graph $G = (V, E)$ on n vertices and m edges and a terminal set $T \subseteq V$, finds the t-minimal $(t, T \setminus \{t\})$-mincut for every $t \in T$ in $O(m \log n)$ time plus $O(\log n)$ maxflow calls each on a graph with $O(n)$ vertices and $O(m)$ edges.*

Fix any $x \in X$ where $\text{mincut}_G(p, x) \leq 2d$. Let (A, B) be any (p, x)-mincut where $p \in A$ and $x \in B$. We have three observations. The first crucial observation says that (A, B) must be "unbalanced" w.r.t. X.

Proposition III.7. $\min(|A \cap X|, |B \cap X|) \leq \frac{2}{\phi}$.

Proof: By the well-linkedness of X, we have $d\phi \cdot \min(|A \cap X|, |B \cap X|) \leq |E(A, B)|$. On the other hand, we have $|E(A, B)| \leq \text{mincut}_G(p, x) \leq 2d$. The bound follows by combining the two inequalities. ∎

Let S be an i.i.d. sample of X with rate $\phi/2$. Let $T = S \cup \{p\}$. The second observation roughly says that, with probability $\Omega(\phi) \geq 1/n^{o(1)}$, one side of (A, B) contains only one vertex from T.

Proposition III.8. *With probability at least $\phi/(2e)$, either*

- *$A \cap T = \{p\}$ and $x \in T$, or*
- *$B \cap T = \{x\}$ and $p \in T$.*

Proof: By Proposition III.7, either $|A \cap X| \leq 2/\phi$ or $|B \cap X| \leq 2/\phi$. If $|A \cap X| \leq 2/\phi$, then we have

$$\begin{aligned}&\Pr[A \cap T = \{p\} \text{ and } x \in T]\\ =&\Pr[(A \setminus \{p\}) \cap S = \emptyset] \cdot \Pr[x \in S]\\ =&\left(1 - \frac{\phi}{2}\right)^{|A \cap X| - 1} \cdot \frac{\phi}{2} \geq \frac{1}{e} \cdot \frac{\phi}{2}.\end{aligned}$$

If $|B \cap X| \leq 2/\phi$, then we have

$$\begin{aligned}&\Pr[B \cap T = \{x\} \text{ and } p \in T]\\ =&\Pr[(B \setminus \{x\}) \cap S = \emptyset] \cdot \Pr[x \in S]\\ =&\left(1 - \frac{\phi}{2}\right)^{|B \cap X| - 1} \cdot \frac{\phi}{2} \geq \frac{1}{e} \cdot \frac{\phi}{2}.\end{aligned}$$

■

The last observation says that given that the event in Proposition III.8 happens, then either the $(p, T \setminus \{p\})$-mincut or the $(x, T \setminus \{x\})$-mincut is a (p, x)-mincut. This will be useful for us because the Isolating Cut Lemma can compute the $(p, T \setminus \{p\})$-mincut and the $(x, T \setminus \{x\})$-mincut quickly.

Proposition III.9. *We have the following:*

1) *If $A \cap T = \{p\}$ and $x \in T$, then any $(p, T \setminus \{p\})$-mincut is a (p, x)-mincut.*
2) *If $B \cap T = \{x\}$ and $p \in T$, then any $(x, T \setminus \{x\})$-mincut is a (p, x)-mincut.*

Proof: (1): As $A \cap T = \{p\}$, (A, B) is a $(p, T \setminus \{p\})$-cut and so $\text{mincut}(p, T \setminus \{p\}) \leq |E(A, B)| = \text{mincut}(p, x)$. Since $x \in T$, any $(p, T \setminus \{p\})$-cut is a (p, x)-cut. Therefore, a $(p, T \setminus \{p\})$-mincut is a (p, x)-cut of size at most $\text{mincut}(p, x)$. So it is a (p, x)-mincut.

(2): The proof is symmetric. As $B \cap T = \{x\}$, (B, A) is a $(x, T \setminus \{x\})$-cut and so $\text{mincut}(x, T \setminus \{x\}) \leq |E(A, B)| = \text{mincut}(p, x)$. Since $p \in T$, any $(x, T \setminus \{x\})$-cut is a (p, x)-cut. Therefore, a $(x, T \setminus \{x\})$-mincut is a (p, x)-cut of size at most $\text{mincut}(p, x)$. So it is a (p, x)-mincut. ■

The above observations directly suggest an algorithm stated in Algorithm 2. Below, we prove its correctness in Lemma III.10 and bound the running time in Lemma III.11.

Algorithm 2: SINGLESOURCEMINCUT(G, X, d, ϕ, p)

1) Initialize $\texttt{val}[x] = 2d$ for all $x \in X \setminus \{p\}$.
2) Repeat $c \cdot \frac{\ln n}{\phi}$ times (for a large enough constant c)
   a) Sample S from X i.i.d. at rate $\frac{\phi}{2}$.
   b) Call the Isolating Cuts Lemma (Lemma III.6) on terminal set $T = S \cup \{p\}$ and obtain a $(t, T \setminus \{t\})$-mincut C_t of size $\delta(C_t)$ for every $t \in T$.
   c) For each $x \in S \setminus \{p\}$, do the following:
      i) If C_x is a (p, x)-cut (i.e., $p \notin C_x$), then $\texttt{val}[x] \leftarrow \min\{\texttt{val}[x], \delta(C_x)\}$.
      ii) If C_p is a (p, x)-cut (i.e., $x \notin C_p$), then $\texttt{val}[x] \leftarrow \min\{\texttt{val}[x], \delta(C_p)\}$.
3) Return $\texttt{val}[\cdot]$.

Lemma III.10. *Algorithm 2 computes, with high probability, $\texttt{val}[x] = \min\{2d, \text{mincut}(p, x)\}$ for all $x \in X \setminus \{p\}$.*

Proof: Note that $\texttt{val}[x] \leq 2d$ from initialization. So we only need to show that if $\text{mincut}(p, x) \leq 2d$, then $\texttt{val}[x] = \text{mincut}(p, x)$ whp. On one hand, $\texttt{val}[x] \geq \text{mincut}(p, x)$ because whenever $\texttt{val}[x]$ is decreased, it is assigned the size of some (p, x)-cut (which is either C_x or C_p). On the other hand, we claim $\texttt{val}[x] \leq \text{mincut}(p, x)$ whp. To see this, observe that, with probability at least $1 - (1 - \phi/(2e))^{c \cdot \frac{\ln n}{\phi}} \geq 1 - 1/n^{10}$, that there exists an iteration in Algorithm 2 where the event in Proposition III.8 happens. That is, $A \cap T = \{p\}$ and $x \in T$, or $B \cap T = \{x\}$ and $p \in T$. Given this, by Proposition III.9, either a $(x, T \setminus \{x\})$-mincut C_x or a $(p, T \setminus \{p\})$-mincut C_p is a (p, x)-mincut and so the algorithm sets $\texttt{val}[x] \leq \text{mincut}(p, x)$. ■

Lemma III.11. *Algorithm 2 takes $O\left(\frac{m \log^2 n}{\phi}\right)$ time plus $O\left(\frac{\log^2 n}{\phi}\right)$ max-flow calls each on a graph with $O(n)$ vertices and $O(m)$ edges.*

Proof: Note that $O\left(\frac{\log n}{\phi}\right)$ invocations of Lemma III.6 takes $O\left(\frac{\log^2 n}{\phi}\right)$ max-flow calls each on a graph with $O(n)$ vertices and $O(m)$ edges plus $O\left(\frac{m \log^2 n}{\phi}\right)$ time. Additionally, for each invocation of Lemma III.6, we update $\texttt{val}[\cdot]$ in $O(n)$ time for a total of $O\left(\frac{n \log n}{\phi}\right)$ time. ■

By Lemma III.10 and Lemma III.11, this completes the proof of Lemma III.4.

IV. Well-linked Partitioning

The goal of this section is to prove Lemma II.3. We start with some notation. For disjoint vertex subsets $V_1, \ldots, V_\ell \subseteq V$, define $E_G(V_1, \ldots, V_\ell)$ as the set of edges $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$ for some $i \neq j$. For a vector $\mathbf{d} \in \mathbb{R}^V$ of entries on the vertices, define $\mathbf{d}(v)$ as the entry of v in $\mathbf{d}$, and for a subset $U \subseteq V$, define $\mathbf{d}(U) := \sum_{v \in U} \mathbf{d}(v)$. We now introduce the concept of an expander "weighted" by *demands* on the vertices.

Definition IV.1 ($(\phi, \mathbf{d})$-expander)**.** Consider a weighted, undirected graph $G = (V, E)$ with edge weights w and a vector $\mathbf{d} \in \mathbb{R}^V_{\geq 0}$ of non-negative "demands" on the vertices. The graph G is a $(\phi, \mathbf{d})$-*expander* if for all subsets $S \subseteq V$,

$$\frac{|E_G(S, V \setminus S)|}{\min\{\mathbf{d}(S), \mathbf{d}(V \setminus S)\}} \geq \phi.$$

We now state the algorithm of [18] that computes our desired expander decomposition, which generalizes the result from [9].

Theorem IV.2 ($(\phi, \mathbf{d})$-expander decomposition algorithm [18])**.** *Fix any $\epsilon > 0$ and any parameter $\phi > 0$. Given a weighted, undirected graph $G = (V, E)$ with edge weights w and a non-negative demand vector $\mathbf{d} \in \mathbb{R}^V_{\geq 0}$ on the vertices, there is a deterministic algorithm running in $m^{1+\epsilon}(\lg n)^{O(1/\epsilon^2)}$ time that partitions V into subsets $V_1, \ldots, V_\ell$ such that*

1) *For each $i \in [\ell]$, define the demands $\mathbf{d}_i \in \mathbb{R}^{V_i}_{\geq 0}$ as $\mathbf{d}$ restricted to the vertices in V_i. Then, the graph $G[V_i]$ is a $(\phi, \mathbf{d}_i)$-expander.*
2) *The total number $|E_G(V_1, \ldots, V_\ell)|$ of inter-cluster edges is $B\phi \cdot \mathbf{d}(V)$ where $B = (\lg n)^{O(1/\epsilon^4)}$.*

Given Theorem IV.2, we can apply it to obtain the desired well-linked sets using the following key lemma:

Lemma IV.3. *There is an algorithm that, given any subset $U \subseteq V_{\geq d} = \{v \mid \deg_G(v) \geq d\}$, outputs disjoint subsets $X_1, \ldots, X_k$ of U such that $k \leq 2n/d$, every set X_i is (d, ϕ)-well-linked in G for $\phi = n^{-o(1)}$, and $|\cup_i X_i| \geq |U|/2$. This algorithm runs in $m^{1+o(1)}$ time.*

Proof: Apply Theorem IV.2 with $\phi = \frac{1}{8B}$ (recall that $B = (\lg n)^{O(1/\epsilon^4)}$), and the following demands: $\mathbf{d}(v) = d$ for all $v \in U$ and $\mathbf{d}(v) = 0$ for all $v \notin U$. (We will set the value of ϵ later.) We obtain a partition $V_1, \ldots, V_\ell$ of V with $|E_G(V_1, \ldots, V_\ell)| \leq B\phi \cdot \mathbf{d}(V) = \mathbf{d}(V)/8 = d \cdot |U|/8$. For each $i \in [\ell]$ and vertex $v \in U \cap V_i$, assign v the value $x(v) = \frac{|E_G(V_i, V \setminus V_i)|}{|U \cap V_i|}$, so that $\sum_{v \in U} x(v) = 2|E_G(V_1, \ldots, V_\ell)| \leq d \cdot |U|/4$. If we select a vertex $v \in U$ uniformly at random, then the expected value of $x(v)$ is at most $d/4$; so, by Markov's inequality, we have $x(v) \leq d/2$ with probability at least $1/2$. Let $U' \subseteq U$ be all vertices $v \in U$ with $x(v) \leq d/2$; it follows that $|U'| \geq |U|/2$. For each subset V_i, the value of $x(v)$ is identical for all vertices in $U \cap V_i$. Hence, either $U \cap V_i$ is contained in U' or is disjoint from it; without loss of generality, let $V_1, \ldots, V_k$ be the sets that are contained in U' for some $k \leq \ell$. We now set $X_i = U \cap V_i$ for all $i \in [k]$.

We first show that each set X_i is (d, ϕ)-well-linked. Since $X_i \subseteq U \subseteq V_{\geq d}$, we have $\deg_G(v) \geq d$ for all $v \in X_i$. Now consider a partition (A, B) of X_i. For any subset $S \subseteq V_i$ that contains A and is disjoint from B, we have $|E_G(S, V_i \setminus S)| \geq \phi \cdot \min\{\mathbf{d}(S), \mathbf{d}(V \setminus S)\} = d\phi \cdot \min\{|A|, |B|\}$, where the inequality holds by definition of $(\phi, \mathbf{d})$-expander. It follows that $\text{mincut}_G(A, B) \geq d\phi \cdot \min\{|A|, |B|\}$, and hence, X_i is (d, ϕ)-well-linked.

We now show that $|V_i| \geq d/2$ for all $i \in [k]$; since the V_i are disjoint, this would imply that $k \leq 2n/d$. Recall that $X_i = U \cap V_i$, so that $|E_G(V_i, V \setminus V_i)| = \sum_{v \in X_i} x(v) \leq |X_i| \cdot d/2$. By averaging, there exists $v \in X_i$ with $|E_G(v, V \setminus V_i)| \leq d/2$. Since $\deg_G(v) \geq d$, at least $d/2$ edges incident to v must have their other endpoint inside V_i. Since G is simple, the endpoints must be distinct, so $|V_i| \geq d/2$, as promised.

Finally, we fix the value of $\epsilon = (\lg n)^{-1/5}$. Then,

$$\phi = \frac{1}{8B} = \frac{1}{8(\lg n)^{O(1/\epsilon^4)}} = \frac{1}{8(\lg n)^{O((\lg n)^{4/5})}} = \frac{1}{n^{o(1)}}.$$

The running time is $m^{1+\epsilon} \cdot (\lg n)^{O(1/\epsilon^2)} = m^{1+(\lg n)^{-1/5}} \cdot (\lg n)^{O(\lg n)^{2/5}} = m^{1+o(1)}$. ∎

We now prove Lemma II.3 using Lemma IV.3. Begin with $U = V_{\geq d}$ and repeatedly apply Lemma IV.3 to obtain disjoint $X_1, \ldots, X_k \subseteq U$, and then reassign U to be $U \setminus \bigcup_{i \in [k]} X_i$ for the next iteration; stop when $|U| = 1$. Since the size of U halves at each iteration, the number of iterations is at most $\lceil \log_2 n \rceil$. We thus obtain $\lceil \log_2 n \rceil \cdot 2n/d$ sets, each of which is (d, ϕ)-well-linked in G, where $\phi = n^{-o(1)}$.

V. Conclusion

In this paper, we gave an $n^{2+o(1)}$-time algorithm for constructing a Gomory-Hu tree in a simple, undirected graph thereby solving the All Pairs Minimum Cuts problem in the same running time. Generalizing this result to weighted graphs, thereby improving on Gomory and Hu's 60-year old algorithm that uses $n - 1$ maxflow calls would

be a breakthrough result. An intermediate goal would be to show this for unweighted multigraphs, i.e., allowing parallel edges but not edge weights. The $\tilde{O}(mn)$-time Gomory-Hu tree algorithms of Bhalgat *et al.* [6] and of Karger and Levine [13] apply to these graphs, but not to general weighted graphs, suggesting that this intermediate class might be easier for the APMC problem than general weighted graphs. Obtaining subcubic (in n) running times for the APMC problem in unweighted (but not necessarily simple) graphs remains an interesting open question.

A different question concerns the optimality of the result presented in this paper. As we discussed, our result is nearly optimal if mincut values have to be explicitly reported for all vertex pairs. Even if that is not required, our algorithm is nearly optimal if the input graph is dense, i.e., if $m = \Theta(n^2)$. So, that leaves graphs containing $o(n^2)$ edges under the condition that we do not need explicit reporting of mincut values for all vertex pairs. Ideally, for such graphs, one would like to design a near-linear time algorithm, i.e., a running time of $m^{1+o(1)}$. But, that is not known even for a single s-t mincut, i.e. for the maxflow problem. A more immediate goal is to construct a Gomory-Hu tree via a subpolynomial (or polylogarithmic) number of maxflow calls. Indeed, this was recently achieved at the cost of obtaining an *approximate* Gomory-Hu tree instead of an exact one [16]. For the exact problem, the current paper gives a reduction, but to polylogarithmic calls of the single source mincut problem rather than the s-t mincut problem.[4] Clearly, the former is a more powerful oracle, and hence the reduction is easier. Improving this reduction to the s-t mincut problem, or equivalently removing the approximation in the result of [16], remains an interesting open question.

References

[1] A. Abboud, R. Krauthgamer, and O. Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 105–118. IEEE, 2020. 2, 10

[2] A. Abboud, R. Krauthgamer, and O. Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 48–61. SIAM, 2020. 1, 2

[3] A. Abboud, R. Krauthgamer, and O. Trabelsi. APMF < apsp? gomory-hu tree for unweighted graphs in almost-quadratic time. *CoRR*, abs/2106.02981, 2021. 2

[4] A. Abboud, R. Krauthgamer, and O. Trabelsi. Subcubic algorithms for Gomory-Hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing*, 2021. 1, 7

[5] S. R. Arikati, S. Chaudhuri, and C. D. Zaroliagis. All-pairs min-cut in sparse networks. *J. Algorithms*, 29(1):82–110, 1998. 2

[6] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. An Õ(mn) Gomory-Hu tree construction algorithm for unweighted graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 605–614, 2007. 1, 3, 10

[7] G. Borradaile, D. Eppstein, A. Nayyeri, and C. Wulff-Nilsen. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In S. P. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, volume 51 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. 2

[8] G. Borradaile, P. Sankowski, and C. Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 601–610. IEEE Computer Society, 2010. 2

[9] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1158–1167, 2020. 9

[10] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. 1, 2

[11] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM J. Comput.*, 19(1):143–155, 1990. 2

[12] R. Hariharan, T. Kavitha, and D. Panigrahi. Efficient algorithms for computing all low s-t edge connectivities and related problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 127–136, 2007. 3

[4] [1] also give a similar reduction, although they require the oracle to actually report mincuts while we only require the mincut values.

[13] D. R. Karger and M. S. Levine. Fast augmenting paths by random sampling from residual graphs. *SIAM J. Comput.*, 44(2):320–339, 2015. 1, 10

[14] T. Kathuria, Y. P. Liu, and A. Sidford. Unit capacity maxflow in almost $o(mˆ{4/3})$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 119–130. IEEE, 2020. 2

[15] J. Li and D. Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society, 2020. 7

[16] J. Li and D. Panigrahi. Approximate Gomory-Hu tree is faster than $n-1$ max-flows. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing*, 2021. 2, 5, 10

[17] J. Li, D. Panigrahi, and T. Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *CoRR*, abs/2106.02233, 2021. 1, 7

[18] J. Li and T. Saranurak. Deterministic weighted expander decomposition in almost-linear time, 2021. arXiv:2106.01567. 9

[19] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. 3

[20] D. Panigrahi. Gomory-Hu trees. In *Encyclopedia of Algorithms*, pages 858–861. 2016. 2

[21] J. van den Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. 2021. arXiv:2101.05719. 1, 2, 3

[22] T. Zhang. Faster cut-equivalent trees in simple graphs. *CoRR*, abs/2106.03305, 2021. 2