Approximate Gomory–Hu Tree Is Faster Than $n − 1$ Max-Flows[*]

Jason Li
jmli@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Debmalya Panigrahi
debmalya@cs.duke.edu
Duke University
Durham, NC, USA

ABSTRACT

The Gomory-Hu tree or cut tree (Gomory and Hu, 1961) is a classic data structure for reporting $s − t$ mincuts (and by duality, the values of $s − t$ maxflows) for all pairs of vertices s and t in an undirected graph. Gomory and Hu showed that it can be computed using $n − 1$ exact maxflow computations. Surprisingly, this remains the best algorithm for Gomory-Hu trees more than 50 years later, *even for approximate mincuts*. In this paper, we break this longstanding barrier and give an algorithm for computing a $(1 + \epsilon)$-approximate Gomory-Hu tree using polylog(n) maxflow computations. Specifically, we obtain the runtime bounds we describe below.

We obtain a randomized (Monte Carlo) algorithm for undirected, weighted graphs that runs in $\tilde{O}(m + n^{3/2})$ time and returns a $(1+\epsilon)$-approximate Gomory-Hu tree algorithm whp. Previously, the best running time known was $\tilde{O}(n^{5/2})$, which is obtained by running Gomory and Hu's original algorithm on a cut sparsifier of the graph.

Next, we obtain a randomized (Monte Carlo) algorithm for undirected, unweighted graphs that runs in $m^{4/3+o(1)}$ time and returns a $(1 + \epsilon)$-approximate Gomory-Hu tree algorithm whp. This improves on our first result for sparse graphs, namely $m = o(n^{9/8})$. Previously, the best running time known for unweighted graphs was $\tilde{O}(mn)$ for an exact Gomory-Hu tree (Bhalgat *et al.*, STOC 2007); no better result was known if approximations are allowed.

As a consequence of our Gomory-Hu tree algorithms, we also solve the $(1 + \epsilon)$-approximate all pairs mincut (APMC) and single source mincut (SSMC) problems in the same time bounds. (These problems are simpler in that the goal is to only return the $s − t$ mincut values, and not the mincuts.) This improves on the recent algorithm for these problems in $\tilde{O}(n^2)$ time due to Abboud *et al.* (FOCS 2020).

CCS CONCEPTS

• **Theory of computation** → **Network flows**; **Graph algorithms analysis**.

KEYWORDS

graph algorithm, Gomory-Hu tree, graph connectivity

1 INTRODUCTION

The algorithmic study of cuts and flows is one of the pillars of combinatorial optimization. The foundations of this field were established in the celebrated work of Ford and Fulkerson in the mid-50s [8]. They studied the $s − t$ edge connectivity problem, namely finding a set of edges of minimum weight whose removal disconnects two vertices s from t in a graph (such a set of edges is called an $s − t$ mincut). They showed that the weight of an $s − t$ mincut equals the maximum flow between s and t in the graph, a duality that has underpinned much of the success in this field. Soon after their work, in a remarkable result, Gomory and Hu [11] showed that by using just $n − 1$ maxflows, they could construct a tree T on the vertices of an undirected graph G such that for every pair of vertices s and t, the $s − t$ edge connectivity in T was equal to that in G. In other words, the $\binom{n}{2}$ pairs of vertices had at most $n − 1$ different edge connectivities and they could be obtained using just $n − 1$ maxflow calls. Moreover, for all vertex pairs s and t, the bipartition of vertices in the $s − t$ mincut in tree T (note that this is just the bipartition created by removing the minimum weight edge on the unique $s − t$ path in T) was also an $s − t$ mincut in graph G. This data structure, called a *cut tree* or more appropriately a *Gomory-Hu tree* (abbreviated GH-tree) after its creators, has become a standard feature in algorithms textbooks, courses, and research since their work.

But, rather surprisingly, in spite of the remarkable successes in this field as a whole, the best algorithm for constructing a GH-tree remains the one given by Gomory and Hu almost six decades after their work. There have been alternatives suggested along the way, although none of them unconditionally improves on the original construction. Gusfield [12] gave an algorithm that also uses $n − 1$ maxflows, but on the original graph itself (the GH algorithm runs maxflows on contracted graphs as we will see later) to improve the performance of the algorithm in practice. Bhalgat *et al.* [5] (see also [13]) obtained an $\tilde{O}(mn)$ algorithm for this problem, but only for unweighted graphs. (Note that using the state of the art maxflow algorithms [17], the GH algorithm has a running time of $m^{4/3+o(1)}n$ for unweighted graphs, which is slower.) Karger and Levine [14] matched this running time using a randomized maxflow subroutine, also for unweighted graphs. Recently, Abboud *et al.* [2] improved this bound for *sparse* unweighted graphs to $\tilde{O}(m^{3/2}n^{1/6})$, thereby demonstrating that the $\tilde{O}(mn)$ is not tight, at least in certain edge

density regimes. Further improvements have been obtained in special cases: in particular, near-linear time algorithms are known for planar graphs [7] and surface-embedded graphs [6]. Experimental studies of GH tree algorithms have also been performed [10]. The reader is referred to a survey article on this topic for more background [18].

In spite of all the works described above, the status of the GH tree problem for general weighted graphs has remained unchanged for the last six decades. Namely, we know that a GH tree can be constructed using $n - 1$ maxflows, but can we do better? In fact, surprisingly, a faster GH tree algorithm is not known *even if one allowed approximations*, i.e., if the $s - t$ mincuts in the GH tree and those in the original graph could differ by a multiplicative factor. At first glance, this would appear surprising, since $\tilde{O}(m)$-time algorithms for $(1 + \epsilon)$-approximation of maxflows are known. (In contrast, obtaining an exact maxflow algorithm that runs in near-linear time remains one of the major open challenges in graph algorithms.) But, the difficulty in using these faster approximate maxflow algorithms in the GH tree problem is that the GH algorithm (and also Gusfield's algorithm) use recursive calls in a manner that approximation errors can build up across the different recursive layers of the algorithm. Approximation, however, does present some advantage, in that one can use standard graph sparsification techniques to reduce the number of edges to $\tilde{O}(n)$ (see, e.g., [4, 9]) and then apply the GH algorithm (with exact maxflow) on this sparse graph. This reduces the running time to $n - 1$ invocations of maxflow on $\tilde{O}(n)$-edge graphs, which has a total running time of $\tilde{O}(n^{5/2})$ using the current state of the art maxflow algorithm of Lee and Sidford [15]. But, fundamentally, even allowing approximations, we do not have a GH tree algorithm that beats the $O(n)$ maxflows benchmark set by the original GH algorithm.

But, there has been some exciting progress of late in this line of research. Very recently, in a beautiful paper, Abboud *et al.* [1] showed that the problem of finding all pairs edge connectivities (that a GH tree obtains) can be reduced to polylog(n) instances of the single source mincut problem (we call this the SSMC problem). Given a fixed source vertex s, the latter problem asks for the $s - t$ edge connectivity of s with every other vertex t. Their reduction is also robust to approximations because, crucially, the recursive depth of the reduction is only polylog(n) (as against the recursive depth of GH and Gusfield's algorithms, which can be $\Omega(n)$). So, in essence, they reduced the recursive depth of the algorithm in exchange for using a more powerful primitive, namely edge connectivity for $n-1$ pairs of vertices (one of the pair is common) rather than for just a single pair. The algorithm that they used to solve the single source edge connectivity problem is the obvious one: run $s - t$ maxflow for every vertex t. Naturally, this does not improve the running time for exact all pairs edge connectivity, since we are still running $n - 1$ maxflows. But, importantly, if approximations are allowed, we can now use the $\tilde{O}(m)$-time approximate maxflow algorithm rather than the exact one. Coupled with sparsification, this yields a running time bound of $\tilde{O}(n^2)$ improving on the previous bound of $\tilde{O}(n^{5/2})$.

However, while this improves the time complexity of approximate all pairs edge connectivity, the reduction framework of [1] does not support the construction of an approximate GH tree.

Namely, they give a data structure (called a *flow tree*) that returns the (approximate) edge connectivity of a vertex pair when queried, but does not return a mincut for that pair. Nevertheless, this result creates a range of possibilities, now that we have a technique for designing computation trees for all pairs edge connectivity that have small recursive depth. In this paper, we give the first approximation algorithm (our approximation factor is $1 + \epsilon$ for any $\epsilon > 0$) for GH tree that beats the running time of $n - 1$ maxflow calls. Namely, we show that a $(1 + \epsilon)$-approximate GH-tree can be constructed using polylog number of calls to an exact maxflow subroutine, plus $\tilde{O}(m)$ time outside these maxflow calls.

1.1 Our Results

To state our main result, we first formally define an approximate GH tree.

Definition 1.1 (Approximate Gomory-Hu tree). *Given a graph $G = (V, E)$, a $(1+\epsilon)$-approximate Gomory-Hu tree is a weighted tree T on V such that*

- *For all $s, t \in V$, consider the minimum-weight edge (u, v) on the unique $s-t$ path in T. Let U' be the vertices of the connected component of $T - (u, v)$ containing s. Then, the set $U' \subseteq V$ is a $(1 + \epsilon)$-approximate (s, t)-mincut, and its value is the weight of the (u, v) edge in T.*

We now state our main theorem that obtains a $(1+\epsilon)$-approximate GH tree for weighted graphs:

Theorem 1.2. *Let G be an undirected graph with non-negative edge weights. There is a randomized algorithm that w.h.p., outputs a $(1 + \epsilon)$-approximate Gomory-Hu tree and runs in $\tilde{O}(m)$ time plus calls to exact max-flow on instances with a total of $\tilde{O}(n\epsilon^{-1} \log^2 \Delta)$ vertices and $\tilde{O}(n\epsilon^{-1} \log^2 \Delta)$ edges, where Δ is the ratio of maximum to minimum edge weights. Assuming polynomially bounded edge weights and using the $\tilde{O}(m\sqrt{n})$ time max-flow algorithm of Lee and Sidford [15], the algorithm runs in $\tilde{O}(m + n^{3/2}\epsilon^{-2})$ time.*

For unweighted graphs, we obtain the following result, which gives a better running time for sparse graphs (if $m = o(n^{9/8})$):

Theorem 1.3. *Let G be an unweighted, undirected graph. There is a randomized algorithm that w.h.p., outputs a $(1 + \epsilon)$-approximate Gomory-Hu tree and runs in $\tilde{O}(m)$ time plus calls to exact max-flow on unweighted instances with a total of $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges. Using the $m^{4/3+o(1)}$-time max-flow algorithm for unweighted graphs of Liu and Sidford [17], the algorithm runs in $m^{4/3+o(1)}\epsilon^{-1}$ time.*

To the best of our knowledge, this is the first algorithm for (approximate) GH tree that goes beyond $n - 1$ maxflow calls in general weighted graphs. Our reduction to exact maxflow instances is "black box", i.e., any maxflow algorithm can be used; as a consequence, if one were to assume that eventually maxflow would be solved in $\tilde{O}(m)$-time as is often conjectured, then these theorems would automatically yield an $\tilde{O}(m)$-time algorithm for a $(1 + \epsilon)$-approximate GH tree.

Given these results, one might be tempted to replace the exact maxflow calls in our algorithm by approximate maxflow subroutines. Indeed, if this were possible, the running time of the overall

algorithm would be $\tilde{O}(m)$ without additional assumptions (i.e., without assuming a $\tilde{O}(m)$-time exact maxflow algorithm). Unfortunately, a key tool that we employ called the *isolating cuts lemma*, which was recently introduced by the authors for the deterministic mincut problem [16], requires the computation of exact maxflows; we are not aware of any approximation versions of this lemma. We leave the problem of obtaining a near-linear time approximate GH tree algorithm as an interesting open question (that is probably easier than an exact $\tilde{O}(m)$-time maxflow algorithm).

Abboud *et al.* [1] recently considered the APMC (also called *flow tree*) problem, which asks for the value of the $s - t$ mincut for all vertex pairs s, t but not a mincut itself.

Definition 1.4 (All-pairs min-cut (APMC)). *In the* all-pairs min-cut *(APMC) problem, the input is an undirected graph $G = (V, E)$ and we need to output a data structure that allows us to query the value of the (s, t)-mincut for each pair $s, t \in V$. In the $(1 + \epsilon)$-approximate APMC problem, the input is the same, and we need to output a $(1 + \epsilon)$-approximation to the value of the (s, v)-mincut for each $v \in V \setminus \{s\}$.*

Abboud *et al.* gave a framework that reduces the APMC problem to polylog(n) calls to the single source mincut (SSMC) problem.

Definition 1.5 (Single-source min-cut). *In the* single-source mincut *(SSMC) problem, the input is an undirected graph $G = (V, E)$ and a source vertex $s \in V$, and we need to output a (s, v)-mincut for each $v \in V \setminus \{s\}$. In the $(1 + \epsilon)$-approximate SSMC problem, the input is the same, and we need to output a $(1 + \epsilon)$-approximate (s, v)-mincut for each $v \in V \setminus \{s\}$.*

To solve the SSMC instances, Abboud *et al.* used $n - 1$ maxflows. Our work shows that the SSMC problem can be approximately solved using polylog(n) maxflows calls, and that an approximate GH tree can be recovered in the process. Our main tool is the following subroutine that we call the *Cut Threshold* (CT) problem, which may have further applications on its own:

Theorem 1.6 (Cut Threshold algorithm). *Let $G = (V, E)$ be a weighted, undirected graph, and let $s \in V$, and let $\lambda \geq 0$ be a parameter (the "cut threshold"). There is an algorithm that outputs whp all vertices $v \in V$ with $\text{mincut}(s, v) \leq \lambda$, and runs in $\tilde{O}(m)$ time plus polylog(n) calls to max-flow instances on $O(n)$-vertex, $O(m)$-edge graphs.*

We use this theorem to obtain an algorithm for approximately solving the SSMC problem that is faster than running approximate maxflows for all the $n - 1$ vertices separately:

Theorem 1.7. *Let G be a weighted, undirected graph, and let $s \in V$. There is an algorithm that outputs, for each vertex $v \in V \setminus \{s\}$, a $(1 + \epsilon)$-approximation of $\text{mincut}(s, v)$, and runs in $\tilde{O}(m \log \Delta)$ time plus polylog$(n) \cdot \log \Delta$ calls to max-flow on $O(n)$-vertex, $O(m)$-edge graphs, where Δ is the ratio of maximum to minimum edge weights.*

Finally, note that a (approximate) GH tree also solves the (approximate) APMC problem. But, we can also get an APMC algorithm by simply plugging in the SSMC algorithm in Theorem 1.7 to the reduction framework of Abboud *et al.* This improves the time complexity of the APMC problem from $\tilde{O}(mn)$ obtained by Abboud *et al.* [1] to $\tilde{O}(m + n^{3/2})$.

1.2 Our Techniques

To sketch our main ideas, let us first think of the CT problem (Theorem 1.6). Note that this theorem is already sufficient to obtain the improved the running times for the SSMC and APMC problems, although obtaining a $(1 + \epsilon)$-approximate GH tree needs additional ideas. To solve the CT problem, our main tool is the *isolating cuts lemma*, introduced by the authors recently for solving the deterministic mincut problem [16]. We first describe this tool.[1]

Definition 1.8 (Minimum isolating cuts). *Consider a weighted, undirected graph $G = (V, E)$ and a subset $R \subseteq V$ ($|R| \geq 2$). The minimum isolating cuts for R is a collection of sets $\{S_v : v \in R\}$ such that for each vertex $v \in R$, the set S_v is the side containing v of the minimal $(v, R \setminus v)$-mincut, i.e., for any set S satisfying $v \in S$ and $S \cap (R \setminus v) = \emptyset$, we have $w(\partial S) \leq w(\partial S_v)$, and moreover, if $w(\partial S) = w(\partial S_v)$ then $S_v \subseteq S$.*

Lemma 1.9 (Isolating Cuts Lemma [16]). *Fix a subset $R \subseteq V$ ($|R| \geq 2$). There is an algorithm that computes the minimum isolating cuts $\{S_v : v \in R\}$ for R using $O(\log |R|)$ calls to s–t max-flow on weighted graphs of $O(n)$ vertices and $O(m)$ edges, and takes $\tilde{O}(m)$ deterministic time outside of the max-flow calls. If the original graph G is unweighted, then the inputs to the max-flow calls are also unweighted. Moreover, the sets $\{S_v : v \in R\}$ are disjoint.*

The crucial aspect of the isolating cuts lemma is that the number of maxflow calls is $O(\log n)$ irrespective of the size of R. For the CT problem, define $Z = V \setminus \{s\}$; our goal is to invoke the isolating cuts lemma polylog(n) times and identify all vertices $v \in Z$ with $\text{mincut}(s, v) \leq \lambda$ whp. In fact, we will only describe an algorithm that identifies each vertex in Z satisfying this condition with probability $\Omega(1/\text{polylog}(n))$; removing these vertices from Z and repeating $O(\log n)$ times identifies all such vertices in Z whp. Fix a GH tree T of the graph rooted at s, and let (u, v) be an edge of weight $\leq \lambda$ in T where u is closer to s than v. Let T_v denote the subtree under v in T, and let n_v be the number of vertices in Z that appear in T_v. For any vertex $z \in T_v$, we have $\text{mincut}(s, z) \leq \lambda$. Now, suppose we sample a set of vertices from Z at rate $1/n_v$ and define this sample as R. Then, we invoke the isolating cuts lemma with the set R, after adding s to this set. Next, if the isolating cuts lemma returns cuts of value $\leq \lambda$, we mark the vertices in Z separated by those cuts from s as having $\text{mincut}(s, z) \leq \lambda$ and remove them from Z. Clearly, every marked vertex z indeed has $\min(s, z) \leq \lambda$. But, how many vertices do we end up marking? Let us focus on the subtree T_v. With constant probability, exactly one vertex from T_v is sampled in R, and with probability $\Omega(1/n_v)$, this sampled vertex is v itself. In that happens, the isolating cut lemma would return the $s - v$ mincut, namely the cut represented by the edge (u, v) in the GH tree. This allows us to mark all the n_v vertices that are in Z and appear in T_v. So, roughly speaking, we are able to mark at least n_v vertices with probability $1/n_v$ in this case. Of course, we do not know the value of n_v, but we try all sampling levels in inverse powers of 2. We formalize and refine this argument to show that we can indeed mark every vertex $z \in Z$ with $\text{mincut}(s, z) \leq \lambda$ with probability at least $\Omega(1/\log n)$ using this algorithm.

---

[1]We remark that the *minimal* condition was not present in [16], but the algorithm to find minimum isolating cuts from [16] can be trivially modified to output the minimal $(v, R \setminus v)$-mincuts, so we omit the algorithm and direct interested readers to [16].

We now use the CT algorithm as a "sieve" to obtain an SSMC algorithm. We start with $\text{mincut}(s, v)$ for all vertices $v \in V \setminus \{s\}$ tentatively set to the maximum possible edge connectivity (call it $\lambda_{\max}$). Next, we run the CT algorithm with $\lambda = (1 - \epsilon)\lambda_{\max}$. The vertices v that are identified by this algorithm as having $\text{mincut}(s, v) \leq \lambda$ drop down to the next level of the hierarchy, while the remaining vertices v' are declared to have $\text{mincut}(s, v') \in ((1 - \epsilon)\lambda, \lambda]$. In the next level of the hierarchy, we again invoke the CT algorithm, but now with λ equal to $(1 - \epsilon)$ factor of the previous iteration. In this manner, we iteratively continue moving down the hierarchy, cutting the threshold λ by a factor of $(1 - \epsilon)$ in every step, until the connectivity of all vertices has been determined.

Finally, we come to the problem of obtaining an approximate GH tree. Gomory and Hu's original algorithm uses the following strategy: find an $s - t$ mincut for any pair of vertices s and t, and recurse on the two sides of the cut in separate subproblems where the other side of the cut is contracted to a single vertex. They used submodularity of cuts to show that contracting one side of an $s - t$ mincut does not change the connectivity between vertices on the other side. Moreover, they gave a procedure for combining the two GH trees returned by the recursive calls into a single GH tree at the end of the recursion. Ideally, we would like to use the same algorithm but replace an exact $s - t$ mincut with an approximate one. But now, the connectivities in the recursive subproblems are (additively) distorted by the approximation error of the $s - t$ mincut. This imposes two additional restrictions. **(a)** First, the values of the $s - t$ mincuts identified in the recursive algorithm must now be monotone non-decreasing with depth of the recursion so that the approximation error on a larger $s - t$ mincut doesn't get propagated to a smaller $s' - t'$ mincut further down in the recursion. **(b)** Second, the depth of recursion must now be $\text{polylog}(n)$ so that one can control the buildup of approximation error in the recursion by setting the error parameter in a single step to be $\epsilon/\text{polylog}(n)$. Unfortunately, neither of these conditions is met by Gomory and Hu's algorithm. For instance, the recursion depth can be $n - 1$ if each $s - t$ mincut is a degree cut. The order of $s - t$ mincut values in the recursion is also arbitrary and depends on the choice of s and t in each step (which itself is arbitrary).

Let us first consider condition **(a)**. Instead of finding the $s - t$ mincut for an arbitrary pair of terminal vertices s and t, suppose we found the Steiner mincut on the terminals, i.e., the cut of smallest value that splits the terminals. This would also suffice in terms of the framework since a Steiner mincut is also an $s - t$ mincut for some pair s, t. But, it brings additional advantages: namely, we get the monotonicity in cut values with recursive depth that we desire. At a high level, this is the idea that we implement: we use the CT algorithm (with some technical modifications) where we set the threshold λ to the value of the Steiner mincut, and identify a partitioning of the terminals where each subset of the partition represents a $(1 + \epsilon)$ approximation to the Steiner mincut.

But, how do we achieve condition **(b)**? Fixing the vertex s in the invocation of the SSMC algorithm, we can identify terminal vertices v that have $\text{mincut}(s, v) \in ((1 - \epsilon)\lambda, \lambda]$, where λ is the Steiner mincut. But, these approximate Steiner mincuts might be unbalanced in terms of the number of vertices on the two sides of the cut. To understand the problem, suppose there is a single Steiner mincut identified by the CT algorithm, and this cut is the degree cut

of s. Then, one subproblem contains all but one vertex in the next round of recursion; consequently, the recursive depth can be high. We overcome this difficulty in two steps. First, we ensure that the only "large" subproblem that we recurse on is the one that contains s. This can be ensured by sampling $O(\log n)$ different vertices as s, which boosts the probability that s is on the larger side of an unbalanced approximate Steiner mincut. This ensures that in the recursion tree, we can only have a large recursive depth along the path containing s. Next, we show that even though we are using an approximate method for determining mincuts, the approximation error only distorts the connectivities in the subproblems not containing s. This ensures that the approximation errors can build up only along paths in the recursion tree that have depth $O(\log n)$. Combining these two techniques, we obtain our overall algorithm for an approximate GH tree.

2 $(1 + \epsilon)$-APPROXIMATE SINGLE SOURCE MIN-CUT ALGORITHM

2.1 Preliminaries

We have already defined the SSMC problem, but for our analysis, we need some more definitions. In particular, we first define a Gomory-Hu Steiner tree and its approximation version.

Definition 2.1 (Gomory-Hu Steiner tree). *Given a graph $G = (V, E)$ and a set of terminals $U \subseteq V$, the Gomory-Hu Steiner tree is a weighted tree T on the vertices U, together with a function $f : V \to U$, such that*

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

Definition 2.2 (Approximate Gomory-Hu Steiner tree). *Given a graph $G = (V, E)$ and a set of terminals $U \subseteq V$, the $(1 + \epsilon)$-approximate Gomory-Hu Steiner tree is a weighted tree T on the vertices U, together with a function $f : V \to U$, such that*

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

It would also be useful in our analysis to use the notion of a *minimal* Gomory-Hu tree. We define this next.

Definition 2.3 (Rooted minimal Gomory-Hu Steiner tree). *Given a graph $G = (V, E)$ and a set of terminals $U \subseteq V$, a rooted minimal Gomory-Hu Steiner tree is a Gomory-Hu Steiner tree on U, rooted at some vertex $r \in U$, with the following additional property:*

(∗) *For all $t \in U \setminus \{r\}$, consider the minimum-weight edge (u, v) on the unique $r - t$ path in T; if there are multiple minimum weight edges, let (u, v) denote the one that is closest to t. Let U' be the vertices of the connected component of $T - (u, v)$ containing r. Then, $\partial_G f^{-1}(U') \subseteq V$ is a minimal (r, t)-mincut, and its value is $w_T(u, v)$.*

The following theorem establishes the existence of a rooted minimal Gomory-Hu Steiner tree rooted at any given vertex.

THEOREM 2.4. *For any graph $G = (V, E)$, terminals $U \subseteq V$, and root $r \in U$, there exists a rooted minimal Gomory-Hu Steiner tree rooted at r.*

PROOF. Let $\epsilon > 0$ be a small enough weight, and let G' be the graph G with an additional edge (r, v) of weight ϵ added for each $v \in V \setminus \{r\}$. (If the edge (r, v) already exists in G, then increase its weight by ϵ instead.) If $\epsilon > 0$ is small enough, then for all $t \in V \setminus \{r\}$ and $S \subseteq V$, if $\partial_{G'} S$ is an (r, t)-mincut in G', then $\partial_G S$ is an (r, t)-mincut in G.

Let (T', f) be a Gomory-Hu Steiner tree for G'. We claim that it is essentially a minimal Gomory-Hu Steiner tree for G, except that its edge weights need to be recomputed as mincuts in G and not G'. More formally, let T be the tree T' with the following edge re-weighting: for each edge (u, v) in T, take a connected component U' of $T - (u, v)$ and reset the edge weight of (u, v) to be $w(\partial_G f^{-1}(U'))$ and not $w(\partial_{G'} f^{-1}(U'))$. We now claim that (T, f) is a minimal Steiner Gomory-Hu tree for G.

We first show that (T, f) is a Gomory-Hu Steiner tree for G. Fix $s, t \in U$, let (u, v) be the minimum-weight edge on the $s - t$ path in T', and let U' be the vertices of the connected component of $T' - (u, v)$ containing s. Since (T', f) is a Gomory-Hu Steiner tree for G', we have that $\partial_{G'} f^{-1}(U')$ is an (s, t)-mincut in G'. If $\epsilon > 0$ is small enough, then by our argument from before, $\partial_G f^{-1}(U')$ is also an (s, t)-mincut in G. By our edge re-weighting of T, the edge (u, v) has the correct weight. Moreover, (u, v) is the minimum-weight edge on the $s - t$ path in T, since a smaller weight edge would contradict the fact that $\partial_G f^{-1}(U')$ is an (s, t)-mincut.

We now show the additional property $(*)$ that makes (T, f) a minimal Gomory-Hu Steiner tree. Fix $t \in U \setminus \{r\}$, and let (u, v) and U' be defined as in $(*)$, i.e., (u, v) is the minimum-weight edge (u, v) on the $r - t$ path that is closest to t, and U' is the vertices of the connected component of $T - (u, v)$ containing r. Since (T, f) is a Gomory-Hu Steiner tree for G, we have that $\partial_G f^{-1}(U')$ is an (r, t)-mincut of value $w_T(u, v)$. Suppose for contradiction that $\partial_G f^{-1}(U')$ is not a *minimal* (r, t)-mincut. Then, there exists $S \subsetneq f^{-1}(U')$ such that ∂S is also an (r, t)-mincut. By construction of G', $w(\partial_{G'} S) = w(\partial_G S) + |S|\epsilon$ and $w(\partial_{G'} f^{-1}(U')) = w(\partial_G f^{-1}(U')) + |f^{-1}(U')|\epsilon$. We have $w(\partial_G S) = w(\partial_G f^{-1}(U'))$ and $|S| < |f^{-1}(U')|$, so $w(\partial_{G'} S) < w(\partial_{G'} f^{-1}(U'))$. In other words, $f^{-1}(U')$ is not an (r, t)-mincut in G', contradicting the fact that (T', f) is a Gomory-Hu Steiner tree for G'. Therefore, property $(*)$ is satisfied, concluding the proof. □

2.2 Algorithms for the CT and SSMC Problems

As described earlier, the main tool in our SSMC algorithm is an algorithm for the Cut Threshold (CT) problem. We first describe a single step of the CUTTHRESHOLD algorithm (we call this CUT-THRESHOLDSTEP).

We remark that throughout this section, we will always set $z = \infty$, so the constraint $|S_v^i \cap U| \leq z$ in line 4 can be ignored. However, the variable z will play a role in the next section on computing a Gomory-Hu tree.

Let $D = D^0 \cup D^1 \cup \cdots \cup D^{\lfloor \lg |U| \rfloor}$ be the union of the sets output by the algorithm. Let D^* be all vertices $v \in U \setminus s$ for which there exists an (s, v)-cut of weight at most W whose side containing v has at most z vertices in U.

Algorithm 1 CUTTHRESHOLDSTEP$(G = (V, E), s, U, W, z)$

1: Initialize $R^0 \leftarrow U$ and $D \leftarrow \emptyset$
2: **for** i from 0 to $\lfloor \lg |U| \rfloor$ **do**
3:     Compute minimum isolating cuts $\{S_v^i : v \in R^i\}$ on inputs G and R^i
4:     Let D^i be the union of $S_v^i \cap U$ over all $v \in R^i \setminus \{s\}$ satisfying $w(\partial S_v^i) \leq W$ and $|S_v^i \cap U| \leq z$
5:     $R^{i+1} \leftarrow$ subsample of R^i where each vertex in $R^i \setminus \{s\}$ is sampled independently with probability $1/2$, and s is sampled with probability 1
6: **return** $D^0 \cup D^1 \cup \cdots \cup D^{\lfloor \lg |U| \rfloor}$
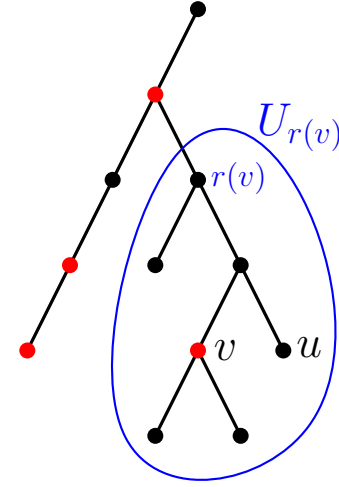
Figure 1: Let $i = \lfloor \lg n_{r(v)} \rfloor = \lfloor \lg 7 \rfloor = 2$, and let the red vertices be those sampled in R^2. Vertex v is active and hits u because v is the only vertex in $U_{r(v)}$ that is red.

Lemma 2.5. $D \subseteq D^*$ *and* $\mathbb{E}[|D|] = \Omega(|D^*|/\log |U|)$.

PROOF. We first prove that $D \subseteq D^*$. Each vertex $u \in D$ belongs to some S_v^i satisfying $w(\partial S_v^i) \leq W$ and $|S_v^i \cap U| \leq z$. In particular, ∂S_v^i is an (s, u)-cut with weight at most W whose side S_v^i containing u has at most z vertices in U, so $u \in D^*$.

It remains to prove that $\mathbb{E}[|D|] \geq \Omega(|D^*|/\log |U|)$. Consider a rooted minimal Steiner Gomory-Hu tree T of G on terminals U rooted at s, which exists by Theorem 2.4. For each vertex $v \in U \setminus \{s\}$, let $r(v)$ be defined as the child vertex of the lowest weight edge on the path from v to s in T. If there are multiple lowest weight edges, choose the one with the maximum depth.

For each vertex $v \in D^*$, consider the subtree rooted at v, define $U_v \subseteq D^*$ to be the vertices in the subtree, and define n_v as the number of vertices in the subtree. We say that a vertex $v \in D^*$ is *active* if $v \in R^i$ for $i = \lfloor \lg n_{r(v)} \rfloor$. In addition, if $U_{r(v)} \cap R^i = \{v\}$, then we say that v *hits* all of the vertices in $U_{r(v)}$ (including itself); see Figure 1. In particular, in order for v to hit any other vertex, it must be active. For completeness, we say that any vertex in $U \setminus D^*$ is not active and does not hit any vertex.

To prove that $\mathbb{E}[|D|] \geq \Omega(|D^*|/\log |U|)$, we will show that

(a) each vertex u that is hit is in D,

(b) the total number of pairs (u, v) for which $v \in D^*$ hits u is at least $c|D^*|$ in expectation for some small enough constant $c > 0$, and

(c) with probability at least $1 - \frac{c}{2|U|^2}$ (for the constant $c > 0$ in (b)), each vertex u is hit by at most $O(\log |U|)$ vertices $v \in D^*$.

For (a), consider the path from u to the root s in T, and take any vertex $v \in D^*$ on the path that is active (possibly u itself). Such a vertex must exist since u is hit by some vertex. By definition, for $i = \lfloor \lg n_{r(v)} \rfloor$, we have $U_{r(v)} \cap R^i = \{v\}$, so $\partial f^{-1}(U_{r(v)})$ is a $(v, R^i \setminus \{v\})$-cut. By the definition of $r(v)$, we have that $\partial f^{-1}(U_{r(v)})$ is a (v, s)-mincut. On the other hand, we have that ∂S_v^i is a $(v, R^i \setminus v)$-mincut, so in particular, it is a (v, s)-cut. It follows that $\partial f^{-1}(U_{r(v)})$ and ∂S_v^i are both (v, s)-mincuts and $(v, R^i \setminus v)$-mincuts, and $w(\partial S_v^i) = \text{mincut}(s, v) \leq W$. Since T is a minimal Gomory-Hu Steiner tree, we must have $f^{-1}(U_{r(v)}) \subseteq S_v^i$. Since S_v^i is the minimal $(v, R^i \setminus \{v\})$-mincut, it is also the minimal (v, s)-mincut, so $S_v^i \subseteq f^{-1}(U_{r(v)})$. It follows that $f^{-1}(U_{r(v)}) = S_v^i$. Since $f^{-1}(U_{r(v)})$ is the minimal (v, s)-mincut and $v \in D^*$, we must have $|f^{-1}(U_{r(v)}) \cap U| \leq z$, so in particular, $|S_v^i \cap U| = |f^{-1}(U_{r(v)}) \cap U| \leq z$. Therefore, the vertex v satisfies all the conditions of line 4. Moreover, since $u \in U_{r(v)} \subseteq f^{-1}(U_{r(v)}) = S_v^i$, vertex u is added to D in the set $S_v^i \cap U$.

For (b), for $i = \lfloor \lg n_{r(v)} \rfloor$, we have $v \in R^i$ with probability exactly $1/2^i = \Theta(1/n_{r(v)})$, and with probability $\Omega(1)$, no other vertex in $U_{r(v)}$ joins R^i. Therefore, v is active with probability $\Omega(1/n_{r(v)})$. Conditioned on v being active, it hits exactly $n_{r(v)}$ many vertices. It follows that v hits $\Omega(1)$ vertices in expectation.

For (c), the number of vertices v that hit vertex u is at most the number of active vertices v for which $r(v)$ is on the path from u to s in T. Label these vertices $u = v_1, v_2, \ldots, v_\ell = s$, ordered by increasing distance from u to $r(v_i)$ in T. Each vertex $v_j \in D^*$ is active with probability $\Theta(1/n_{r(v_j)})$, which is at most $\Theta(1/j)$ since $v_1, \ldots, v_j \in U_{r(v_j)}$. Each vertex $v_j \notin D^*$ is never active. Therefore, the expected number of active vertices on the path from u to s is at most $\sum_{j=1}^{\ell} \Theta(1/j) = \Theta(\ln \ell) \leq \Theta(\ln |U|)$. A standard Chernoff bound shows that with probability at least $1 - \frac{c}{2|U|^3}$ for any constant $c > 0$, the number of active vertices on the path is indeed $O(\ln |U|)$, where the $O(\cdot)$ hides the dependency on c. Taking a union bound over all $u \in U$, the probability that this is true for all vertices is at least $1 - \frac{c}{2|U|^2}$.

Finally, we show why properties (a) to (c) imply $\mathbb{E}[|D|] \geq \Omega(|D^*|/\log |U|)$. In the event that property (c) fails, the total number of pairs (u, v) for which v hits u can be trivially upper bounded by $|U|^2$. Since this occurs with probability at most $\frac{c}{2|U|^2}$, the total contribution to the expectation $c|D^*|$ in property (b) is at most $c/2$. Therefore, the contribution to the expectation in the event that property (c) succeeds is at least $c|D^*| - c/2 \geq (c/2)|D^*|$. In this case, since each vertex is hit at most $O(\log |U|)$ times, there are at least $\Omega(|D^*|/\log |U|)$ vertices hit in expectation, all of which are included in D by property (a). □

We now use iterate Algorithm CutThresholdStep to obtain the CutThreshold algorithm:

Algorithm 2 CutThreshold$(G = (V, E), s, W)$

1: Initialize $U \leftarrow V$ and $D_{\text{total}} \leftarrow \emptyset$
2: **for** $O(\log^2 n)$ iterations **do**
3:     Let D be the union of the sets output by CutThresholdStep(G, s, U, W, ∞)
4:     Update $D_{\text{total}} \leftarrow D_{\text{total}} \cup D$ and $U \leftarrow U \setminus D$
5: **return** D_{total}

Corollary 2.6. *W.h.p., the output D_{total} of CutThreshold is exactly all vertices $v \in U \setminus \{s\}$ for which the (s, v)-mincut has weight at most W.*

PROOF. By Lemma 2.5, $|U \cap D^*|$ decreases by $\Omega(|D^*|/\log n)$ in expectation. After $O(\log^2 n)$ iterations, we have $\mathbb{E}[|U \cap D^*|] \leq 1/\text{poly}(n)$, so w.h.p., $U \cap D^* = \emptyset$. Each vertex in D^* that is removed from U is added to D_{total}, and no vertices in $U \setminus D^*$ are added to D_{total}, so w.h.p., the algorithm returns the correct set D^*. □

In other words, CutThreshold is an algorithm that fulfills Theorem 1.6, restated below.

THEOREM 1.6 (Cut Threshold algorithm). *Let $G = (V, E)$ be a weighted, undirected graph, and let $s \in V$, and let $\lambda \geq 0$ be a parameter (the "cut threshold"). There is an algorithm that outputs whp all vertices $v \in V$ with $\text{mincut}(s, v) \leq \lambda$, and runs in $\tilde{O}(m)$ time plus $\text{polylog}(n)$ calls to max-flow instances on $O(n)$-vertex, $O(m)$-edge graphs.*

Finally, we use the CutThreshold algorithm to design our SSMC algorithm:

---

Algorithm 3 ApproxSSMC$(G = (V, E), s, \epsilon)$

1: Initialize bounds: $w_{\min} \leftarrow$ minimum weight of an edge in G, and $w_{\max} \leftarrow$ maximum weight of an edge
2: **for** all integers $j \geq 0$ s.t. $(1 + \epsilon)^j w_{\min} \in [w_{\min}, (1 + \epsilon)n w_{\max}]$ **do**
3:     $W_j \leftarrow (1 + \epsilon)^j w_{\min}$
4:     $D_j \leftarrow$ CutThreshold(G, s, W)
5: For each vertex $v \in V$, take the largest D_i containing v, and set $\tilde{\lambda}(v) \leftarrow W_i$
6: **return** $\tilde{\lambda} : V \rightarrow \mathbb{R}$

Lemma 2.7. *W.h.p., the output $\tilde{\lambda}$ of ApproxSSMC satisfies $\text{mincut}(s, v) \leq \tilde{\lambda}(v) \leq (1 + \epsilon)\text{mincut}(s, v)$.*

PROOF. For all $v \in V \setminus \{s\}$, we have $w_{\min} \leq \text{mincut}(s, v) \leq w(\partial(\{s\})) \leq n w_{\max}$, so there is an integer j with $W_j \in [\text{mincut}(s, v), (1 + \epsilon)\text{mincut}(s, v)]$. The lemma follows from Corollary 2.6 applied to this j. □

We have therefore proved Theorem 1.7, restated below.

THEOREM 1.7. *Let G be a weighted, undirected graph, and let $s \in V$. There is an algorithm that outputs, for each vertex $v \in V \setminus \{s\}$, a $(1 + \epsilon)$-approximation of $\text{mincut}(s, v)$, and runs in $\tilde{O}(m \log \Delta)$ time plus $\text{polylog}(n) \cdot \log \Delta$ calls to max-flow on $O(n)$-vertex, $O(m)$-edge graphs, where Δ is the ratio of maximum to minimum edge weights.*
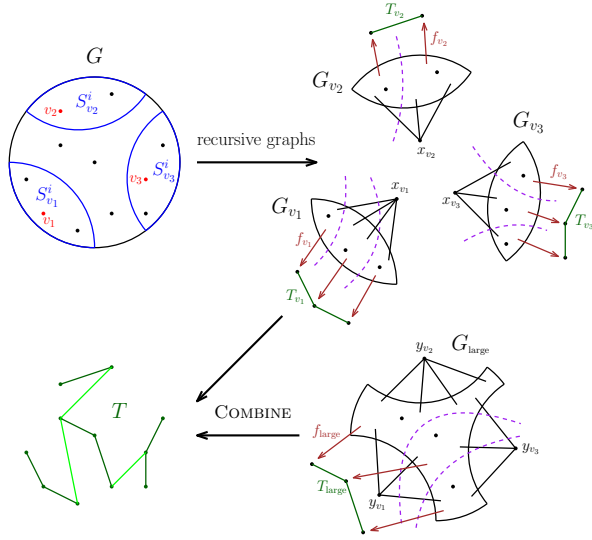
Figure 2: Recursive construction of G_{large} and G_v for $v \in R^i_{\text{small}}$. Here, $R^i_{\text{small}} = \{v_1, v_2, v_3\}$, denoted by red vertices on the top left. The dotted blue curves on the right mark the boundaries of the regions $f_{v_i}^{-1}(u) : u \in U_{v_i}$ and $f_{v_{\text{large}}}^{-1}(u) : u \in U_{\text{large}}$. The light green edges on the bottom left are the edges $(f_{v_i}(x_{v_i}), f_{\text{large}}(y_{v_i}))$ added on line 1 of Combine.

3 APPROXIMATE GOMORY-HU STEINER TREE

3.1 Unweighted Graphs

Let $\epsilon > 0$ be a fixed parameter throughout the recursive algorithm. We present our approximate Steiner Gomory-Hu tree algorithm in ApproxSteinerGHTree below. See Figure 2 for a visual guide to the algorithm.

At a high level, the algorithm applies divide-and-conquer by cutting the graph along sets S_v^i computed by CutThreshold-Step, applying recursion to each piece, and stitching the recursive Gomory-Hu trees together in the same way as the standard recursive Gomory-Hu tree construction. To avoid complications, we only select sets S_v^i from a *single* level $i \in \{0, 1, 2 \ldots, \lfloor \lg |U| \rfloor\}$, which are guaranteed to be vertex-disjoint. Furthermore, instead of selecting all sets $\{S_v^i : v \in R^i\}$, we only select those for which $|S_v^i \cap U| \le |U|/2$; this allows us to bound the recursion depth. By choosing the source $s \in U$ at *random*, we guarantee that in expectation, we do not exclude too many sets S_v^i. The chosen sets partition the graph into disjoint sets of vertices (including the set of vertices outside of any chosen set S_v^i). We split the graph along this partition a similar way to the standard Gomory-Hu tree construction: for each set in the partition, contract all other vertices into a single vertex and recursively compute the Steiner Gomory-Hu tree of the contracted graph. This gives us a collection of Gomory-Hu Steiner trees, which we then stitch together into a single Gomory-Hu Steiner tree in the standard way.

Algorithm 4 ApproxSteinerGHTree$(G = (V, E), U)$

1: $\lambda \leftarrow$ global Steiner mincut of G with terminals U
2: $s \leftarrow$ uniformly random vertex in U
3: Call CutThresholdStep$(G, s, U, (1 + \epsilon)\lambda, |U|/2)$, and let R^j and $S_v^j : v \in R^j$ $(0 \le j \le \lg |U|)$ be the intermediate variables in the algorithm
4: Let $i \in \{0, 1, \ldots, \lfloor \lg |U| \rfloor\}$ be the iteration maximizing $\left| \bigcup_{v \in R^i} (S_v^i \cap U) \right|$

5: **for** each $v \in R^i$ **do**　　▷ Construct recursive graphs and apply recursion
6: 　　Let G_v be the graph G with vertices $V \setminus S_v^i$ contracted to a single vertex x_v　　　　　　▷ S_v^i are disjoint
7: 　　Let $U_v \leftarrow S_v^i \cap U$
8: 　　$(T_v, f_v) \leftarrow$ ApproxSteinerGHTree(G_v, U_v)
9: Let G_{large} be the graph G with (disjoint) vertex sets S_v^i contracted to single vertices y_v for all $v \in R^i$
10: Let $U_{\text{large}} \leftarrow U \setminus \bigcup_{v \in R^i} (S_v^i \cap U)$
11: $(T_{\text{large}}, f_{\text{large}}) \leftarrow$ ApproxSteinerGHTree$(G_{\text{large}}, U_{\text{large}})$

12: Combine $(T_{\text{large}}, f_{\text{large}})$ and $\{(T_v, f_v) : v \in R^i\}$ into (T, f) according to Combine
13: **return** (T, f)

Algorithm 5 Combine$((T_{\text{large}}, f_{\text{large}}), \{(T_v, f_v) : v \in R^i\})$

1: Construct T by starting with the disjoint union $T_{\text{large}} \cup \bigcup_{v \in R^i} T_v$ and, for each $v \in R^i$, adding an edge between $f_v(x_v) \in U_v$ and $f_{\text{large}}(y_v) \in U_{\text{large}}$ of weight $w(\partial_G S_v^i)$
2: Construct $f : V \to U$ by $f(v') = f_{\text{large}}(v')$ if $v' \in U_{\text{large}}$ and $f(v') = f_v(v')$ if $v' \in U_v$ for some $v \in R^i$
3: **return** (T, f)

3.2 Approximation

Since the approximation factors can potentially add up down the recursion tree, we need to bound the depth of the recursive algorithm. Here, there are two types of recursion: the recursive calls (G_v, U_v), and the single call $(G_{\text{large}}, U_{\text{large}})$. Taking a branch down (G_v, U_v) is easy: since $|U_v| \le |U|/2$, the algorithm can travel down such a branch at most $\lg |U|$ times. The difficult part is in bounding the number of branches down $(G_{\text{large}}, U_{\text{large}})$. It turns out that after polylog(n) consecutive branches down $(G_{\text{large}}, U_{\text{large}})$, the Steiner mincut increases by factor $(1 + \epsilon)$, w.h.p.; we elaborate on this insight in Section 3.3, which concerns the running time. Since the Steiner mincut can never decrease down any recursive branch, it can increase by factor $(1 + \epsilon)$ at most ϵ^{-1}polylog$(n) \log \Delta$ times. Thus, we have a bound of ϵ^{-1}polylog$(n) \log \Delta$ on the recursion depth, w.h.p.

This depth bound alone is not enough for the following reason: if the approximation factor increase by $(1 + \epsilon)$ along each recursive branch, then the total approximation becomes $(1+\epsilon)^{\epsilon^{-1}\text{polylog}(n) \log \Delta}$, which is no good because the $(1 + \epsilon)$ and ϵ^{-1} cancel each other. Here, our key insight is that actually, the approximation factor does not distort *at all* down $(G_{\text{large}}, U_{\text{large}})$. It may increase by factor

$(1 + \epsilon)$ down any (G_v, U_v), but this can only happen $\lg |U|$ times, giving us an approximation factor of $(1 + \epsilon)^{\lg |U|}$, which is fine because we can always retroactively replace ϵ with $\Theta(\epsilon/\lg |U|)$ to obtain the desired $(1 + \epsilon)$.

The lemma below formalizes our insight that approximation factors are preserved down the branch $(G_{\text{large}}, U_{\text{large}})$.

Lemma 3.1. *For any distinct vertices $p, q \in U_{\text{large}}$, we have* $\text{mincut}_{G_{\text{large}}}(p, q) = \text{mincut}_G(p, q)$.

PROOF. Since G_{large} is a contraction of G, we have $\text{mincut}_{G_{\text{large}}}(p, q) \geq \text{mincut}_G(p, q)$. To show the reverse inequality, fix any (p, q)-mincut in G, and let S be one side of the mincut. We show that for each $v \in R^i$, either $S_v^i \subseteq S$ or $S_v^i \subseteq V \setminus S$. Assuming this, the cut ∂S stays intact when the sets S_v^i are contracted to form G_{large}, so $\text{mincut}_{G_{\text{large}}}(p, q) \leq w(\partial S) = \text{mincut}_G(p, q)$.

Consider any $v \in R^i$, and suppose first that $v \in S$. Then, $S_v^i \cap S$ is still a $(v, R^i \setminus v)$-cut, and $S_v^i \cup S$ is still a (p, q)-cut. By the submodularity of cuts,

$$w(\partial_G S_v^i) + w(\partial_G S) \geq w(\partial_G(S_v^i \cup S)) + w(\partial_G(S_v^i \cap S)).$$

In particular, $S_v^i \cap S$ must be a minimum $(v, R^i \setminus v)$-cut. Since S_v^i is the minimal $(v, R^i \setminus v)$-mincut, it follows that $S_v^i \cap S = S_v^i$, or equivalently, $S_v^i \subseteq S$.

Suppose now that $v \notin S$. In this case, we can swap p and q, and swap S and $V \setminus S$, and repeat the above argument to get $S_v^i \subseteq V \setminus S$. □

Similarly, the lemma below says that approximation factors distort by at most $(1 + \epsilon)$ down a (G_v, U_v) branch.

Lemma 3.2. *For any $v \in R^i$ and any distinct vertices $p, q \in U_v$, we have* $\text{mincut}_G(p, q) \leq \text{mincut}_{G_v}(p, q) \leq (1 + \epsilon)\text{mincut}_G(p, q)$.

PROOF. The lower bound $\text{mincut}_G(p, q) \leq \text{mincut}_{G_v}(p, q)$ holds because G_v is a contraction of G, so we focus on the upper bound. Fix any (p, q)-mincut in G, and let S be the side of the mincut not containing s (recall that $s \in U$ and $s \notin S_v^i$). Since $S_v^i \cup S$ is a (p, s)-cut (it is also a (q, s)-cut), it is in particular a Steiner cut for terminals U, so $w(S_v^i \cup S) \geq \lambda$. Also, $w(S_v^i) \leq (1 + \epsilon)\lambda$ by the choice of the threshold $(1 + \epsilon)\lambda$ (line 3). Together with the submodularity of cuts, we obtain

$$(1 + \epsilon)\lambda + w(\partial_G S) \geq w(\partial_G S_v^i) + w(\partial_G S)$$
$$\geq w(\partial_G(S_v^i \cup S)) + w(\partial_G(S_v^i \cap S))$$
$$\geq \lambda + w(\partial_G(S_v^i \cap S)).$$

The set $S_v^i \cap S$ stays intact under the contraction from G to G_v, so $w(\partial_{G_v}(S_v^i \cap S)) = w(\partial_G(S_v^i \cap S))$. Therefore,

$$\text{mincut}_{G_v}(p, q) \leq w(\partial_{G_v}(S_v^i \cap S))$$
$$= w(\partial_G(S_v^i \cap S))$$
$$\leq w(\partial_G S) + \epsilon\lambda$$
$$\leq \text{mincut}_G(p, q) + \epsilon\,\text{mincut}_G(p, q),$$

as promised. □

Finally, the lemma below determines our final approximation factor.

Lemma 3.3. APPROXSTEINERGHTREE$(G = (V, E), U)$ *outputs a* $(1 + \epsilon)^{\lg |U|}$*-approximate Gomory-Hu Steiner tree.*

PROOF. We apply induction on $|U|$. Since $|U_v| \leq |U|/2$ for all $v \in R^i$, by induction, the recursive outputs (T_v, f_v) are Gomory-Hu Steiner trees with approximation $(1 + \epsilon)^{\lg |U_v|} \leq (1 + \epsilon)^{\lg |U| - 1}$. By definition, this means that for all $s, t \in U_v$ and the minimum-weight edge (u, u') on the s–t path in T_v, letting $U_v' \subseteq U_v$ be the vertices of the connected component of $T_v - (u, u')$ containing s, we have that $f_v^{-1}(U_v')$ is a $(1 + \epsilon)^{\lg |U| - 1}$-approximate (s, t)-mincut in G_v with value is $w_T(u, u')$. Define $U' \subseteq U$ as the vertices of the connected component of $T - (u, u')$ containing s. By construction of (T, f) (lines 1 and 2), the set $f^{-1}(U')$ is simply $f_v^{-1}(U_v')$ with the vertex x_v replaced by $V \setminus S_v^i$ in the case that $x_v \in f^{-1}(U')$. Since G_v is simply G with all vertices $V \setminus S_v^i$ contracted to x_v, we conclude that $w_{G_v}(\partial f_v^{-1}(U_v')) = w_G(\partial f^{-1}(U'))$. By Lemma 3.2, the values $\text{mincut}_G(s, t)$ and $\text{mincut}_{G_v}(s, t)$ are within factor $(1 + \epsilon)$ of each other, so $w_G(\partial f^{-1}(U'))$ approximates the (s, t)-mincut in G to a factor $(1 + \epsilon) \cdot (1 + \epsilon)^{\lg |U| - 1} = (1 + \epsilon)^{\lg |U|}$. In other words, the Gomory-Hu Steiner tree condition for (T, f) is satisfied for all $s, t \in U_v$ for some $v \in R^i$.

By induction, the recursive output $(T_{\text{large}}, f_{\text{large}})$ is a Gomory-Hu Steiner tree with approximation $(1 + \epsilon)^{\lg |U_{\text{large}}|} \leq (1 + \epsilon)^{\lg |U|}$. Again, consider $s, t \in U_{\text{large}}$ and the minimum-weight edge (u, u') on the s–t path in T_{large}, and let $U_{\text{large}}' \subseteq U_{\text{large}}$ be the vertices of the connected component of $T_{\text{large}} - (u, u')$ containing s. Define $U' \subseteq U$ as the vertices of the connected component of $T - (u, u')$ containing s. By a similar argument, we have $w_{G_{\text{large}}}(\partial f_{\text{large}}^{-1}(U_{\text{large}}')) = w_G(\partial f^{-1}(U'))$. By Lemma 3.1, we also have $\text{mincut}_G(s, t) = \text{mincut}_{G_{\text{large}}}(s, t)$, so $w_G(\partial f^{-1}(U'))$ is a $(1 + \epsilon)^{\lg |U|}$-approximate (s, t)-mincut in G, fulfilling the Gomory-Hu Steiner tree condition for (T, f) in the case $s, t \in U_{\text{large}}$.

There are two remaining cases: $s \in U_v$ and $t \in U_{v'}$ for distinct $v, v' \in R^i$, and $s \in U_v$ and $t \in U_{\text{large}}$; we treat both cases simultaneously. Since G has Steiner mincut λ, each of the contracted graphs G_{large} and G_v has Steiner mincut at least λ. By induction, every edge in T_v and T_{large} or $T_{v'}$ (depending on case) has weight at least $(1 + \epsilon)^{-\lg |U|}\lambda$. By construction, the s–t path in T has at least one edge of the form $(f_v(x_v), f_{\text{large}}(y_v))$, added on line 1; this edge has weight $w(\partial_G S_v^i) \leq (1 + \epsilon)\lambda$. Therefore, the minimum-weight edge on the s–t path in T has weight at least $(1 + \epsilon)^{-\lg |U|}\lambda$ and at most $(1 + \epsilon)\lambda$; in particular, it is a $(1 + \epsilon)^{\lg |U|}$-approximation of $\text{mincut}_G(s, t)$. If the edge is of the form $(f_v(x_v), f_{\text{large}}(y_v))$, then by construction, the relevant set $f^{-1}(U')$ is exactly S_v^i, which is a $(1 + \epsilon)$-approximate (s, t)-mincut in G. If the edge is in T_{large} or T_v or $T_{v'}$, then we can apply the same arguments used previously. □

3.3 Running Time Bound

In order for a recursive algorithm to be efficient, it must make substantial progress on each of its recursive calls, which can then be used to bound its depth. For each recursive call (G_v, U_v, ϵ), we have $|U_v| \leq |U|/2$ by construction, so we can set our measure of progress to be $|U|$, the number of terminals, which halves upon each recursive call. However, progress on $(G_{\text{large}}, U_{\text{large}}, \epsilon)$ is unclear; in particular, it is possible for $|U_{\text{large}}|$ to be very close to $|U|$

with probability 1. For G_{large}, we define the following alternative measure of progress. Let $P(G, U, W)$ be the set of unordered pairs of distinct vertices whose mincut is at most W:

$$P(G, U, W) = \left\{ \{u, v\} \in \binom{U}{2} : \text{mincut}_G(u, v) \leq W \right\}.$$

In particular, we will consider its size $|P(G, U, W)|$, and show the following expected reduction:

Lemma 3.4. *For any $W \leq (1 + \epsilon)\lambda$, over the random selection of s and the randomness in* CutThresholdStep, *we have*

$$\mathbb{E}[|P(G_{\text{large}}, U_{\text{large}}, W)|] \leq \left(1 - \Omega\left(\frac{1}{\log^2 n} \right) \right) |P(G, U, W)|.$$

Before we prove Lemma 3.4, we show how it implies progress on the recursive call for G_{large}.

Corollary 3.5. *Let λ_0 be the global Steiner mincut of G. W.h.p., after $\Omega(\log^3 n)$ recursive calls along G_{large} (replacing $G \leftarrow G_{\text{large}}$ each time), the global Steiner mincut of G is at least $(1 + \epsilon)\lambda_0$ (where λ_0 is still the global Steiner mincut of the initial graph).*

Proof. Let $W = (1 + \epsilon)\lambda_0$. Initially, we trivially have $|P(G, U, W)| \leq \binom{|U|}{2}$. The global Steiner mincut can only increase in the recursive calls, since G_{large} is always a contraction of G, so we always have $W \leq (1 + \epsilon)\lambda$ for the current global Steiner mincut λ. By Lemma 3.4, the value $|P(G, U, W)|$ drops by factor $1 - \Omega(\frac{1}{\log^2 n})$ in expectation on each recursive call, so after $\Omega(\log^3 n)$ calls, we have

$$\mathbb{E}[|P(G, U, W)|] \leq \binom{|U|}{2} \cdot \left(1 - \Omega\left(\frac{1}{\log^2 n} \right) \right)^{\Omega(\log^3 n)} \leq \frac{1}{\text{poly}(n)}.$$

In other words, w.h.p., we have $|P(G, U, W)| = 0$ at the end, or equivalently, the Steiner mincut of G is at least $(1 + \epsilon)\lambda$. □

Combining both recursive measures of progress together, we obtain the following bound on the recursion depth:

Lemma 3.6. *Let $w_{\min}$ and $w_{\max}$ be the minimum weight and maximum weight of any edge in G. W.h.p., the depth of the recursion tree of* ApproxSteinerGHTree *is $O(\epsilon^{-1} \log^3 n \log(n\Delta))$.*

Proof. For any $\Theta(\log^3 n)$ successive recursive calls down the recursion tree, either one call was on a graph G_v, or $\Theta(\log^3 n)$ of them were on the graph G_{large}. In the former case, $|U|$ drops by half, so it can happen $O(\log n)$ times total. In the latter case, by Corollary 3.5, the global Steiner mincut increases by factor $(1 + \epsilon)$. Let $w_{\min}$ and $w_{\max}$ be the minimum and maximum weights in G, so that $\Delta = w_{\max}/w_{\min}$. Note that for any recursive instance (G', U') and any $s, t \in U'$, we have $w_{\min} \leq \text{mincut}_{G'}(s, t) \leq w(\partial(\{s\})) \leq nw_{\max}$, so the global Steiner mincut of (G', U') is always in the range $[w_{\min}, nw_{\max}]$. It follows that calling G_{large} can happen $O(\epsilon^{-1} \log(nw_{\max}/w_{\min}))$ times, hence the bound. □

We state the next theorem for *unweighted* graphs only. For weighted graphs, there is no nice bound on the number of new edges created throughout the algorithm, and therefore no easy bound on the overall running time. In the next section, we introduce a graph sparsification step to handle this issue.

Lemma 3.7. *For an* unweighted *graph $G = (V, E)$, and terminals $U \subseteq V$,* ApproxSteinerGHTree(G, V, ϵ) *takes time $\tilde{O}(m\epsilon^{-1})$ plus calls to max-flow on instances with a total of $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges.*

Proof. For a given recursion level, consider the instances $\{(G_i, U_i, W_i)\}$ across that level. By construction, the terminals U_i partition U. Moreover, the total number of vertices over all G_i is at most $n + 2(|U| - 1) = O(n)$ since each branch creates 2 new vertices and there are at most $|U| - 1$ branches. The total number of new edges created is at most the sum of weights of the edges in the final $(1 + \epsilon)$-approximate Gomory-Hu Steiner tree. For an unweighted graph, this is $O(m)$ by the following well-known argument. Root the Gomory-Hu Steiner tree T at any vertex $r \in U$; for any $v \in U \setminus r$ with parent u, the cut $\partial\{v\}$ in G is a (u, v)-cut of value $\deg(v)$, so $w_T(u, v) \leq \deg(v)$. Overall, the sum of the edge weights in T is at most $\sum_{v \in U} \deg(v) \leq 2m$.

Therefore, there are $O(n)$ vertices and $O(m)$ edges in each recursion level. By Lemma 3.6, there are $O(\epsilon^{-1} \log^4 n)$ levels (since $\Delta = 1$ for an unweighted graph), for a total of $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges. In particular, the instances to the max-flow calls have $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges in total. □

Combining Lemmas 3.3 and 3.7 and resetting $\epsilon \leftarrow \Theta(\epsilon/\log n)$, we obtain Theorem 1.3, restated below.

Theorem 1.3. *Let G be an unweighted, undirected graph. There is a randomized algorithm that w.h.p., outputs a $(1 + \epsilon)$-approximate Gomory-Hu tree and runs in $\tilde{O}(m)$ time plus calls to exact max-flow on unweighted instances with a total of $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges. Using the $m^{4/3+o(1)}$-time max-flow algorithm for unweighted graphs of Liu and Sidford [17], the algorithm runs in $m^{4/3+o(1)}\epsilon^{-1}$ time.*

Finally, we prove Lemma 3.4, restated below.

Lemma 3.4. *For any $W \leq (1 + \epsilon)\lambda$, over the random selection of s and the randomness in* CutThresholdStep, *we have*

$$\mathbb{E}[|P(G_{\text{large}}, U_{\text{large}}, W)|] \leq \left(1 - \Omega\left(\frac{1}{\log^2 n} \right) \right) |P(G, U, W)|.$$

Proof. Let D^* be all vertices $v \in U \setminus s$ for which there exists an (s, v)-cut of weight at most W whose side containing v has at most $|U|/2$ vertices in U. Define $D = \bigcup_{j=0}^{\lfloor \lg |U| \rfloor} \bigcup_{v \in R^i} (S_v^i \cap U)$. Let $P_{\text{ordered}}(G, U, W)$ be the set of ordered pairs $(u, v) : u, v \in V$ for which there exists an (u, v)-mincut of weight at most W with at most $|U|/2$ vertices in U on the side $S(u, v) \subseteq V$ containing u. We now state and prove the following four properties:

(a) For all $u, v \in U$, $\{u, v\} \in P(G, U, W)$ if and only if either $(u, v) \in P_{\text{ordered}}(G, U, W)$ or $(v, u) \in P_{\text{ordered}}(G, U, W)$ (or both).

(b) For each pair $(u, v) \in P_{\text{ordered}}(G, U, W)$, we have $u \in D^*$ with probability at least $1/2$,

(c) For each $u \in D^*$, there are at least $|U|/2$ vertices $v \in U$ for which $(u, v) \in P_{\text{ordered}}(G, U, W)$.

(d) Over the randomness in CutThresholdStep$(G, U, (1+\epsilon)\lambda)$, $\mathbb{E}[|D|] \geq \Omega(|D^*|/\log |U|)$.

Property (a) follows by definition. Property (b) follows from the fact that $u \in D^*$ whenever $s \notin S(u, v)$, which happens with probability at least $1/2$. Property (c) follows because any vertex $v \in U \setminus S(u, v)$ satisfies $(u, v) \in P_{\text{ordered}}(G, U, W)$, of which there are at least $|U|/2$. Property (d) follows from Lemma 2.5 applied on CutThresholdStep$(G, U, W, |U|/2)$, and then observing that even though we actually call CutThresholdStep$(G, U, (1 + \epsilon)\lambda, |U|/2)$, the set D can only get larger if the weight parameter is increased from W to $(1 + \epsilon)\lambda$.

With properties (a) to (d) in hand, we now finish the proof of Lemma 3.4. Consider the iteration i maximizing the size of $D^i :=\bigcup_{v \in R^i}(S_v^i \cap U)$ (line 4), so that $|D^i| \geq |D|/(\lfloor \lg |U| \rfloor + 1)$. For any vertex $u \in D^i$, all pairs $(u, v) \in P_{\text{ordered}}(G, U, W)$ (over all $v \in U$) disappear from $P_{\text{ordered}}(G, U, W)$, which is at least $|U|/2$ many by (c). In other words,

$$|P_{\text{ordered}}(G, U, W) \setminus P_{\text{ordered}}(G_{\text{large}}, U_{\text{large}}, W)|$$
$$\geq \frac{|U|}{2}|D^i|$$
$$\geq \Omega\left(\frac{|U| \cdot |D|}{\log |U|}\right).$$

Taking expectations and applying (d),

$$\mathbb{E}[|P_{\text{ordered}}(G, U, W) \setminus P_{\text{ordered}}(G_{\text{large}}, U_{\text{large}}, W)|]$$
$$\geq \Omega\left(\frac{|U| \cdot \mathbb{E}[|D|]}{\log |U|}\right)$$
$$\geq \Omega\left(\frac{|U| \cdot |D^*|}{\log^2 |U|}\right).$$

Moreover,

$$|U| \cdot |D^*| \geq \mathbb{E}\left[|\{(u, v) : u \in D^*\}|\right] \geq \frac{1}{2}|P_{\text{ordered}}(G, U, W)|,$$

where the second inequality follows by (b). Putting everything together, we obtain

$$\mathbb{E}[|P_{\text{ordered}}(G, U, W) \setminus P_{\text{ordered}}(G_{\text{large}}, U_{\text{large}}, W)|]$$
$$\geq \Omega\left(\frac{|P_{\text{ordered}}(G, U, W)|}{\log |U|}\right).$$

Finally, applying (a) gives

$$\mathbb{E}[|P(G, U, W) \setminus P(G_{\text{large}}, U_{\text{large}}, W)|] \geq \Omega\left(\frac{|P(G, U, W)|}{\log |U|}\right).$$

Finally, we have $P(G_{\text{large}}, U_{\text{large}}, W) \subseteq P(G, U, W)$ since the (u, v)-mincut for $u, v \in U_{\text{large}}$ can only increase in G_{large} due to G_{large} being a contraction of G (in fact it says the same by Lemma 3.1). Therefore,

$$|P(G, U, W)| - |P(G_{\text{large}}, U_{\text{large}}, W)|$$
$$= |P(G, U, W) \setminus P(G_{\text{large}}, U_{\text{large}}, W)|,$$

and combining with the bound on $\mathbb{E}[|P(G, U, W) \setminus P(G_{\text{large}}, U_{\text{large}}, W)|]$ concludes the proof. □

3.4 Weighted Graphs

For weighted graphs, we cannot easily bound the total size of the recursive instances. Instead, to keep the sizes of the instances small, we sparsify the recursive instances to have roughly the same number of edges and vertices. By the proof of Lemma 3.7, the total number of vertices over all instances in a given recursion level is at most $n + 2(|U| - 1) = O(n)$. Therefore, if each such instance is sparsified, the total number of edges becomes $\tilde{O}(n)$, and the algorithm is efficient.

It turns out we only need to re-sparsify the graph in two cases: when we branch down to a graph G_v (and not G_{large}), and when the mincut λ increases by a constant factor, say 2. The former can happen at most $O(\log n)$ times down any recursion branch, since $|U|$ decreases by a factor 2 each time, and the latter occurs $O(\log(n\Delta))$ times down any branch. Each time, we sparsify up to factor $1 + \Theta(\epsilon/\log(n\Delta))$, so that the total error along any branch is $1 + \Theta(\epsilon)$.

We now formalize our arguments. We begin with the specification routine due to Benczur and Karger [3].

THEOREM 3.8. *Given a weighted, undirected graph G, and parameters $\epsilon, \delta > 0$, there is a randomized algorithm that with probability at least $1 - \delta$ outputs a $(1 + \epsilon)$-approximate sparsifier of G with $O(n\epsilon^{-2}\log(n/\delta))$ edges.*

We now derive approximation and running time bounds.

THEOREM 3.9. *Suppose that the recursive algorithm ApproxSteinerGHTree sparsifies the input in the following three cases, using Theorem 3.8 with the same parameter ϵ and the parameter $\delta = 1/\text{poly}(n)$:*

(1) *The instance was the original input, or*
(2) *The instance was obtained from calling (G_v, U_v), or*
(3) *The instance was obtained from calling $(G_{\text{large}}, U_{\text{large}})$, and the Steiner mincut increased by a factor of at least 2 since the last sparsification.*

Then w.h.p., the algorithm outputs a $(1 + \epsilon)^{O(\log(n\Delta))}$-approximate Gomory-Hu Steiner tree and takes $\tilde{O}(m)$ time plus calls to maxflow on instances with a total of $\tilde{O}(n\epsilon^{-1}\log \Delta)$ vertices and $\tilde{O}(n\epsilon^{-1}\log \Delta)$ edges.

PROOF. We first argue about the approximation factor. Along any branch of the recursion tree, there is at most one sparsification step of type (1), at most $O(\log n)$ sparsification steps of type (2), and at most $O(\log(n\Delta))$ sparsification steps of type (3). Each sparsification distorts the pairwise mincuts by a $(1 + \epsilon)$ factor, so the total distortion is $(1 + \epsilon)^{O(\log(n\Delta))}$.

Next, we consider the running time. The recursion tree can be broken into chains of recursive G_{large} calls, so that each chain begins with either the original instance or some intermediate G_v call, which is sparsified by either (1) or (2). Fix a chain, and let n' be the number of vertices at the start of the chain, so that the number of edges is $O(n' \log n)$. Within each chain, the number of vertices can only decrease down the chain. After each sparsification, many sparsifications of type (2), and between two consecutive sparsifications, the number of edges can only decrease down the chain since the graph can only contract. It follows that each instance in the chain has at most n' vertices and $O(n'\epsilon^{-2}\log n)$ edges. By Lemma 3.6, each chain has length $O(\epsilon^{-1}\log^3 n \log(n\Delta))$, so the total number of vertices and edges in the chain is $\tilde{O}(n'\epsilon^{-3}\log \Delta)$. Imagine charging these vertices and edges to the n' vertices at the root of the chain. In other words, to bound the total number of edges in the recursion tree, it suffices to bound the total number of vertices in the original instance and in intermediate G_v calls.

In the recursion tree, there are n original vertices and at most $2(|U|-1)$ new vertices, since each branch creates 2 new vertices and there are at most $|U|-1$ branches. Each vertex joins $O(\log n)$ many G_v calls, since every time a vertex joins one, the number of terminals drops by half; note that a vertex is never duplicated in the recursion tree. It follows that there are $O(n \log n)$ many vertices in intermediate G_v calls, along with the n vertices in the original instance. Hence, from our charging scheme, we conclude that there are a total of $\tilde{O}(n\epsilon^{-3} \log \Delta)$ vertices and edges in the recursion tree. In particular, the instances to the max-flow calls have $\tilde{O}(n\epsilon^{-3} \log \Delta)$ vertices and edges in total. $\qquad\square$

Resetting $\epsilon \leftarrow \Theta(\epsilon/\log(n\Delta))$, we have thus proved Theorem 1.2, restated below.

Theorem 1.2. *Let G be an undirected graph with non-negative edge weights. There is a randomized algorithm that w.h.p., outputs a $(1+\epsilon)$-approximate Gomory-Hu tree and runs in $\tilde{O}(m)$ time plus calls to exact max-flow on instances with a total of $\tilde{O}(n\epsilon^{-1} \log^2 \Delta)$ vertices and $\tilde{O}(n\epsilon^{-1} \log^2 \Delta)$ edges, where Δ is the ratio of maximum to minimum edge weights. Assuming polynomially bounded edge weights and using the $\tilde{O}(m\sqrt{n})$ time max-flow algorithm of Lee and Sidford [15], the algorithm runs in $\tilde{O}(m + n^{3/2}\epsilon^{-2})$ time.*

REFERENCES

[1] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. 2020. Cut-Equivalent Trees are Optimal for Min-Cut Queries. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society.

[2] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. 2020. New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla (Ed.). SIAM, 48–61.

[3] András A Benczúr and David R Karger. 2015. Approximate s-t Min-Cuts in $\tilde{O}(n^2)$ time. 44, 2 (2015), 290–319.

[4] András A. Benczúr and David R. Karger. 2015. Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs. *SIAM J. Comput.* 44, 2 (2015), 290–319.

[5] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. [n.d.]. Fast edge splitting and Edmonds' arborescence construction for unweighted graphs. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, Shang-Hua Teng (Ed.). 455–464.

[6] Glencora Borradaile, David Eppstein, Amir Nayyeri, and Christian Wulff-Nilsen. 2016. All-Pairs Minimum Cuts in Near-Linear Time for Surface-Embedded Graphs. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA (LIPIcs, Vol. 51)*, Sándor P. Fekete and Anna Lubiw (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1–22:16.

[7] Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. 2015. Min st-Cut Oracle for Planar Graphs with Near-Linear Preprocessing Time. *ACM Trans. Algorithms* 11, 3 (2015), 16:1–16:29.

[8] L. R. Ford and D. R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics* 8 (1956), 399–404.

[9] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. 2019. A General Framework for Graph Sparsification. *SIAM J. Comput.* 48, 4 (2019), 1196–1223.

[10] Andrew V. Goldberg and Kostas Tsioutsiouliklis. 2001. Cut Tree Algorithms: An Experimental Study. *J. Algorithms* 38, 1 (2001), 51–83.

[11] R. E. Gomory and T. C. Hu. 1961. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.* 9, 4 (1961), 551–570.

[12] Dan Gusfield. 1990. Very Simple Methods for All Pairs Network Flow Analysis. *SIAM J. Comput.* 19, 1 (1990), 143–155.

[13] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. 2007. 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*, Nikhil Bansal, Kirk Pruhs, and Clifford Stein (Eds.). SIAM, 127–136.

[14] David R. Karger and Matthew S. Levine. 2015. Fast Augmenting Paths by Random Sampling from Residual Graphs. *SIAM J. Comput.* 44, 2 (2015), 320–339.

[15] Yin Tat Lee and Aaron Sidford. 2014. Path Finding Methods for Linear Programming: Solving Linear Programs in Õ(vrank) Iterations and Faster Algorithms for Maximum Flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 424–433.

[16] Jason Li and Debmalya Panigrahi. 2020. Deterministic Min-cut in Polylogarithmic Max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society.

[17] Yang P Liu and Aaron Sidford. 2020. Faster Divergence Maximization for Faster Maximum Flow. *arXiv preprint arXiv:2003.08929* (2020).

[18] Debmalya Panigrahi. 2016. Gomory-Hu Trees. In *Encyclopedia of Algorithms*. 858–861.