Breaking the Cubic Barrier for All-Pairs Max-Flow: Gomory-Hu Tree in Nearly Quadratic Time

Amir Abboud
Weizmann Institute of Science
Rehovot, Israel
amir.abboud@weizmann.ac.il

Robert Krauthgamer
Weizmann Institute of Science
Rehovot, Israel
robert.krauthgamer@weizmann.ac.il

Jason Li
Simons Institute
University of California, Berkeley
Berkeley, USA
jmli@alumni.cmu.edu

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

Thatchaphol Saranurak
University of Michigan, Ann Arbor
Ann Arbor, USA
thsa@umich.edu

Ohad Trabelsi
Toyota Technological Institute at Chicago
Chicago, USA
ohadt@ttic.edu

***Abstract*—In 1961, Gomory and Hu showed that the All-Pairs Max-Flow problem of computing the max-flow between all $\binom{n}{2}$ pairs of vertices in an undirected graph can be solved using only $n-1$ calls to any (single-pair) max-flow algorithm. Even assuming a linear-time max-flow algorithm, this yields a running time of $O(mn)$, which is $O(n^3)$ when $m = \Theta(n^2)$. While subsequent work has improved this bound for various special graph classes, no subcubic-time algorithm has been obtained in the last 60 years for general graphs. We break this longstanding barrier by giving an $\tilde{O}(n^2)$-time algorithm on general, integer-weighted graphs. Combined with a popular complexity assumption, we establish a counter-intuitive separation: all-pairs max-flows are strictly *easier* to compute than all-pairs shortest-paths.**

Our algorithm produces a cut-equivalent tree, known as the Gomory-Hu tree, from which the max-flow value for any pair can be retrieved in near-constant time. For unweighted graphs, we refine our techniques further to produce a Gomory-Hu tree in the time of a poly-logarithmic number of calls to any max-flow algorithm. This shows an equivalence between the all-pairs and single-pair max-flow problems, and is optimal up to poly-logarithmic factors. Using the recently announced $m^{1+o(1)}$-time max-flow algorithm (Chen *et al.*, March 2022), our Gomory-Hu tree algorithm for unweighted graphs also runs in $m^{1+o(1)}$-time.

***Index Terms*—Gomory-Hu tree, graph algorithms, minimum cut, maximum flow**

I. Introduction

The *edge connectivity* of a pair of vertices s, t in an undirected graph is defined as the minimum weight of edges whose removal disconnects s and t in the graph. Such a set of edges is called an (s,t) mincut, and by duality, its value is equal to that of an (s,t) max-flow. Consequently, the edge connectivity of a vertex pair is obtained by running a max-flow algorithm, and by extension, the edge connectivity for *all* vertex pairs can be obtained by $\binom{n}{2} = \Theta(n^2)$ calls to a max-flow algorithm. (Throughout, n and m denote the number of vertices and edges in the input graph $G = (V, E, w)$, where $w : E \to \mathbb{Z}_+^0$ maps edges to non-negative integer weights. We denote the maximum edge weight by W.)

Definition 1 (The All-Pairs Max-Flow (APMF) Problem)**.** Given an undirected edge-weighted graph, return the edge connectivity of all pairs of vertices.

Remarkably, Gomory and Hu [40] showed in a seminal work in 1961 that one can do a lot better than this naïve algorithm. In particular, they introduced the notion of a *cut tree* (later called *Gomory-Hu tree*, which we abbreviate as GHTREE) to show that $n-1$ max-flow calls suffice for finding the edge connectivity of all vertex pairs.

Theorem 2 (Gomory-Hu (1961))**.** *For any undirected edge-weighted graph $G = (V, E)$, there is a cut tree (or* GHTREE*), which is defined as a tree $\mathcal{T}$ on the same set of vertices V such that for all pairs of vertices $s, t \in V$, the (s,t) mincut in $\mathcal{T}$ is also an (s,t) mincut in G and has the same cut value. Moreover, such a tree can be computed using $n-1$ max-flow calls.*[1]

Since their work, substantial effort has gone into obtaining better GHTREE algorithms, and faster algorithms are now known for many restricted graph classes, including unweighted graphs [7], [24], [48], simple graphs [6], [8], [9], [55], [72], planar graphs [26], surface-embedded graphs [25], bounded treewidth graphs [5], [16], and so on (see the survey [63]). Indeed, GHTREE algorithms are part of standard textbooks in combinatorial optimization (e.g., [12], [31], [67]) and have numerous applications in diverse areas such as networks [45], image processing [71], and optimization [62]. They have also

A full version of this paper is available at arXiv:2111.04958. A.A. is supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science. R.K. is supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, the Weizmann Data Science Research Center, and a Minerva Foundation grant. D.P. is supported in part by NSF Awards CCF-1750140 (CAREER) and CCF-1955703, and ARO Award W911NF2110230. O.T. is supported by the NSF Grant CCF-1815316, and by the NWO VICI grant 639.023.812, and his work is partially done at University of Michigan, Ann Arbor.

[1]These max-flow calls are on graphs that are contractions of G, and thus no larger than G.

2575-8454/22/$31.00 ©2022 IEEE
DOI 10.1109/FOCS54457.2022.00088

inspired entire research directions as the first example of a *sparse representation* of graph cuts, the first non-trivial *global min-cut* algorithm, the first use of *submodular minimization* in graphs, and so forth.

In spite of this attention, Gomory and Hu's 60-year-old algorithm has remained the state of the art for constructing a GHTREE in general, weighted graphs (or equivalently for APMF, due to known reductions [5], [55] showing that any APMF algorithm must essentially construct a GHTREE). Even if we assume an optimal $O(m)$-time max-flow algorithm, the Gomory-Hu algorithm takes $O(mn)$ time, which is $O(n^3)$ when $m = \Theta(n^2)$. Breaking this cubic barrier for the GHTREE problem has been one of the outstanding open questions in the graph algorithms literature.

In this paper, we break this longstanding barrier by giving a GHTREE algorithm that runs in $\tilde{O}(n^2)$-time for general, weighted graphs.

Theorem 3. *There is a randomized Monte Carlo algorithm for the* GHTREE *(and* APMF*) problems that runs in $\tilde{O}(n^2)$ time in general, weighted graphs.*

a) Remarks: 1. As noted earlier (and similar to state-of-the-art max-flow algorithms), we assume throughout the paper that edge weights are integers in the range $\{1, 2, \ldots, W\}$. Throughout, the notation $\tilde{O}(\cdot)$ hides poly-logarithmic factors in n and W.
2. Our result is unconditional, i.e., it does not need to assume a (near/almost) linear-time max-flow algorithm. We note that concurrent to our work, an almost-linear time max-flow algorithm has been announced [30]. Our improvement of the running time of GHTREE/APMF is independent of this result: even with this result, the best GHTREE/APMF bound was $m^{1+o(1)}n$ which is between $n^{2+o(1)}$ and $n^{3+o(1)}$ depending on the value of m, and we improve it to $\tilde{O}(n^2)$. Moreover, we stress that we do not need any recent advancement in max-flow algorithms for breaking the cubic barrier: even using the classic Goldberg-Rao max-flow algorithm [38] in our (combinatorial) algorithm solves GHTREE/APMF in subcubic time.

Our techniques also improve the bounds known for the GHTREE problem in *unweighted graphs*, and even for *simple graphs*, which are defined as unweighted graphs without parallel edges. Observe that the GHTREE problem in simple graphs is easier than in unweighted graphs, which in turn is easier than in general weighted graphs. For unweighted graphs, the best previous results were $\tilde{O}(mn)$ obtained by Bhalgat *et al.* [24] and by Karger and Levine [48], and an incomparable result that reduces the GHTREE problem to $O(\sqrt{m})$ max-flow calls [7]. There has recently been much interest and progress on GHTREE in simple graphs as well [6], [8], [9], [55], [72], with the current best running time being $(m + n^{1.9})^{1+o(1)}$.

We give a reduction of the GHTREE problem in unweighted graphs to $\mathrm{polylog}(n)$ calls of any max-flow algorithm. Note that this reduction is nearly optimal (i.e., up to the poly-log factor) since the all-pairs max-flow problem is at least as hard as finding a single-pair max-flow. Using the recent $m^{1+o(1)}$-time max-flow algorithm [30], this yields a running time of $m^{1+o(1)}$ for the GHTREE problem in unweighted graphs.

Theorem 4. *There is a randomized Monte Carlo algorithm for the* GHTREE *problem that runs in $m^{1+o(1)}$ time in unweighted graphs.*

b) APMF vs APSP: Our results deliver a surprising message to a fundamental question in graph algorithms: *What is easier to compute, shortest paths or max-flows?* Ignoring $n^{o(1)}$ factors, the single-pair versions are both solvable in linear-time and therefore equally easy; albeit, the shortest path algorithm [33] is classical, elementary, and fits on a single page, whereas the max-flow algorithm [30] is very recent, highly non-elementary, and requires more than a hundred pages to describe and analyze. This and nearly all other evidence (perhaps excluding the mere existence of the succinct Gumory Hu trees) had supported the consensus that max-flows are at least as hard as (if not strictly harder than) shortest paths, and perhaps this can be established by looking at the more general *all-pairs* versions: APMF and APSP (All-Pairs Shortest-Paths). Much effort had gone into proving this belief (APMF $\geq$ APSP) using the tools of fine-grained complexity [3], [5], [7], [10], [50], with limited success: it was shown that (under popular assumptions) APMF is strictly harder than APSP in *directed* graphs, but the more natural undirected setting remained open. The first doubts against the consensus were raised in the aforementioned $n^{2+o(1)}$ algorithms for APMF in simple (*unweighted*) graphs that go below the n^{ω} bound of APSP [68] (where $2 \leq \omega < 2.37286$ [14] denotes the fast matrix multiplication exponent). But if, as many experts believe, $\omega = 2+o(1)$ then the only conclusion is that APMF and APSP are equally easy in simple graphs. In general (*weighted*) graphs, however, one of the central conjectures of fine-grained complexity states that the cubic bound for APSP cannot be broken (even if $\omega = 2$). Under this "APSP Conjecture", Theorem 3 proves that APMF is *strictly easier* than APSP! Alternatively, if one still believes that APMF $\geq$ APSP, then our paper provides strong evidence against the APSP Conjecture and against the validity of the dozens of lower bounds that are based upon it (e.g., [2], [4], [11], [27], [65], [66], [70]) or upon stronger forms of it (e.g., [1], [18], [19], [32], [37]).

A. Related Work

Algorithms: Before this work, the time complexity of constructing a Gomory-Hu tree in general graphs has improved over the years only due to improvements in max-flow algorithms. An alternative algorithm for the problem was discovered by Gusfield [41], where the $n-1$ max-flow queries are made on the original graph (instead of on contracted graphs). This algorithm has the same worst-case time complexity as Gomory-Hu's, but may perform better in practice [39]. Many faster algorithms are known for special graph classes or when allowing a $(1+\epsilon)$-approximation; see the full version of this paper for a summary. Moreover, a

few heuristic ideas for getting a subcubic complexity in social networks and web graphs have been investigated [13].

Hardness Results: The attempts at proving conditional lower bounds for All-Pairs Max-Flow have only succeeded in the harder settings of directed graphs [3], [50] or undirected graphs with vertex weights [7], where Gomory–Hu trees may not even exist [43], [46], [57]. In particular, SETH gives an $n^{3-o(1)}$ lower bound for weighted sparse directed graphs [50] and the 4-Clique conjecture gives an $n^{\omega+1-o(1)}$ lower bound for unweighted dense directed graphs [3].

Applications: Gomory-Hu trees have appeared in many application domains. We mention a few examples: in mathematical optimization for the b-matching problem [62] (and that have been used in a breakthrough NC algorithm for perfect matching in planar graphs [15]); in computer vision [71], leading to the *graph cuts* paradigm; in telecommunications [45] where there is interest in characterizing which graphs have a Gomory-Hu tree that is a *subgraph* [49], [61]. The question of how the Gomory-Hu tree changes with the graph has arisen in applications such as energy and finance and has also been investigated, e.g., [20], [21], [34], [42], [64].

B. Overview of Techniques

We now introduce the main technical ingredients used in our algorithm, and explain how to put them together to prove Theorem 3 and Theorem 4.

a) Notation: In this paper, a *graph* G is an undirected graph $G = (V, E, w)$ with edge weights $w(e) \in \{1, 2, \ldots, W\}$ for all $e \in E$. If $w(e) = 1$ for all $e \in E$, we say that G is unweighted. The total weight of an edge set $E' \subseteq E$ is defined as $w(E') = \sum_{e \in E'} w(e)$. For a cut $(S, V \setminus S)$, we also refer to a side S of this cut as a cut. The *value of cut* S is denoted $\delta(S) = w(E(S, V \setminus S))$. For any two vertices a, b, we say that S is an (a, b)-cut if $|S \cap \{a, b\}| = 1$. An (a, b)-*mincut* is an (a, b)-cut of minimum value, and we denote its value by $\lambda(a, b)$.

b) Reduction to Single-Source Minimum Cuts: The classic Gomory-Hu approach to solving APMF is to recursively solve (s, t) mincut problems on graphs obtained by contracting portions of the input graph. This leads to $n - 1$ max-flow calls on graphs that cumulatively have $O(mn)$ edges. Recent work [5] has shown that replacing (s, t) mincuts by a more powerful gadget of single-source mincuts reduces the cumulative size of the contracted graphs to only $\tilde{O}(m)$. But, how do we solve the single-source mincuts problem? Prior to our work, a subcubic algorithm was only known for simple graphs [6], [8], [9], [55], [72]. Unfortunately, if applied to non-simple graphs, these algorithms become incorrect, and not just inefficient.

Conceptually, our main contribution is to give an $\tilde{O}(n^2)$-time algorithm for the single source mincuts problem in general weighted graphs. For technical reasons, however, we will further restrict this problem in two ways: (1) the algorithm (for the single-source problem) only needs to return the values $\lambda(s, t)$ for some terminals $t \in U \setminus \{s\}$, and (2) the mincut values $\lambda(s, t)$ for the terminals $t \in U \setminus \{s\}$ are guaranteed to be within a 1.1-factor of each other.[2]

We now state a reduction from GHTREE to this restricted single-source problem. Let $U \subseteq V$ be a set of terminal vertices. The *U-Steiner connectivity/mincut* is $\lambda(U) = \min_{a,b \in U} \lambda(a, b)$. The restricted single-source problem is defined below.

Problem 5 (Single-Source Terminal Mincuts with Promise)**.** *The input is a graph $G = (V, E, w)$, a terminal set $U \subseteq V$ and a source terminal $s \in U$ with the promise that for all $t \in U \setminus \{s\}$, we have $\lambda(U) \leq \lambda(s, t) \leq 1.1\lambda(U)$. The goal is to determine the value of $\lambda(s, t)$ for each terminal $t \in U \setminus \{s\}$.*

The reduction has two high-level steps. First, we reduce the single-source terminal mincuts problem *without the promise that* $\lambda(s, t) \in [\lambda(U), 1.1\lambda(U)]$ (we define this formally in the full version of this paper) to the corresponding problem with the promise (i.e., Problem 5) by calling an approximate single-source mincuts algorithm of Li and Panigrahi [54]. Then, we use a reduction from Gomory-Hu tree to the single-source terminal mincuts without the promise that was presented by Li [51].[3] For completeness, we fully describe both steps of the reduction in the full version.

Lemma 6 (Reduction to Single-Source Terminal Mincuts)**.** *There is a randomized algorithm that computes a* GHTREE *of an input graph by making calls to max-flow and single-source terminal mincuts (with the promise, i.e., Problem 5) on 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.*

c) Guide Trees: The main challenge, thus, is to solve single-source terminal mincuts (Problem 5) faster than $n - 1$ max-flow calls. Let us step back and think of a simpler problem: the *global mincut* problem. In a beautiful paper, Karger [47] gave a two-step recipe for solving this problem by using the duality between cuts and tree packings. First, by packing a maximum set of edge-disjoint *spanning* trees in a graph and sampling one of them uniformly at random, the algorithm obtains a spanning tree that, with high probability, 2-*respects* the global mincut, meaning that only two edges from the tree cross the cut. Second, a simple linear-time dynamic program computes the minimum value cut that 2-respects the tree. Can we use this approach?

Clearly, we cannot hope to pack $\lambda(U)$ disjoint spanning trees since the global mincut value could be much less than $\lambda(U)$. But what about Steiner trees? A tree T is called a *U-Steiner tree* if it spans U, i.e., $U \subseteq V(T)$. When U is clear from the context, we write Steiner instead of U-Steiner.

First, we define the k-respecting property for Steiner trees.

Definition 7 (k-respecting)**.** Let $A \subseteq V$ be a cut in $G = (V, E, w)$. Let T be a tree on (some subset of) vertices in V.

[2]The value 1.1 is arbitrary and can be replaced by any suitably small constant greater than 1.

[3]The actual reduction is slightly stronger in the sense that it only requires a "verification" version of single-source terminal mincuts, but we omit that detail for simplicity.

We say that the tree T *k-respects* the cut A (and vice versa) if T contains at most k edges with exactly one endpoint in A.

Using this notion of k-respecting Steiner trees, we can now define a collection of guide trees that is analogous to a packing of spanning trees.

Definition 8 (Guide Trees)**.** For a graph G and set of terminals $U \subseteq V$ with a source $s \in U$, a collection of U-Steiner trees $T_1, \ldots, T_h$ is called a *k-respecting set of guide trees*, or in short *guide trees*, if for every $t \in U \setminus \{s\}$, at least one tree T_i k-respects some (s,t)-mincut in G.

Two questions immediately arise:

1) Can we actually obtain such k-respecting guide trees, for a small k (and h)?
2) Can guide trees be used to speed up the single-source mincuts algorithm?

The first question can be solved in a way that is conceptually (but not technically) similar to Karger's algorithm for global mincut. We first prove, using classical tools in graph theory (namely, Mader's splitting-off theorem [56], and Nash-Williams [60] and Tutte's [69] tree packing) that there exists a packing with $\lambda(U)/2$ edge-disjoint Steiner trees. Then, we use the width-independent Multiplicative Weights Update (MWU) framework [17], [35], [36] to pack a near-optimal number of Steiner trees using $\tilde{O}(m)$ calls to an (approximation) algorithm for the *minimum Steiner tree* problem. For the latter, we use Mehlhorn's 2-approximation algorithm [58] that runs in $\tilde{O}(m)$ time, giving a packing of $\lambda(U)/4$ Steiner trees in $\tilde{O}(m^2)$ time. To speed this up, we compute the packing in a $(1+\epsilon)$-cut-sparsifier of G (e.g., [22]), which effectively reduces m to $\tilde{O}(n)$ for this step. Overall, this gives an $\tilde{O}(n^2)$-time algorithm for constructing 4-respecting guide trees.

We note that our improved running time for unweighted graphs comes from replacing this algorithm for constructing guide trees by a more complicated algorithm. Specifically, we show that *all* of the $\tilde{O}(m)$ calls to (approximate) minimum Steiner tree during the MWU algorithm can be handled in a total of $m^{1+o(1)}$ time using a novel dynamic data structure that relies on (1) a non-trivial adaptation of Mehlhorn's reduction from minimum Steiner tree to approximate Single-Source Shortest Paths and (2) a recent decremental (dynamic) algorithm for the latter problem [23].[4] This achieves running time $m^{1+o(1)}$ compared with $\tilde{O}(n^2)$ for unweighted graphs.

We summarize the construction of guide trees in the next theorem, which we prove in Section III. (The new dynamic data structure that is used in the improvement for unweighted graphs is deferred to the full version of the paper.)

Theorem 9 (Constructing Guide Trees)**.** *There is a randomized algorithm that, given a graph $G = (V, E, w)$, a terminal set $U \subseteq V$ and a source terminal $s \in U$, with the guarantee that for all $t \in U \setminus \{s\}$, $\lambda(U) \leq \lambda(s,t) \leq 1.1\lambda(U)$, computes a 4-respecting set of $O(\log n)$ guide trees. The algorithm takes $\tilde{O}(n^2)$ time on weighted graphs (i.e., when $w(e) \in \{1, 2, \ldots, W\}$ for all $e \in E$) and $m^{1+o(1)}$ time on unweighted graphs (i.e., when $w(e) = 1$ for all $e \in E$).*

But, how do guide trees help? In the case of global mincuts, the tree is spanning, hence every k tree edges define a partition of V, and also a cut in G. Therefore, once the k-respecting property has been achieved, finding the best k-respecting cut is a search over at most n^k cuts for any given tree, and can be done using dynamic programming for small k [47]. In contrast, specifying the k tree-edges that are cut leaves an exponential number of possibilities when T is a Steiner tree based on which side of the cut the vertices not in T appear on. In fact, in the extreme case where the Steiner tree is a single edge between two terminals s and t, computing the 1-respecting mincut is as hard as computing (s,t)-mincut.

We devise a recursive strategy to solve the problem of obtaining k-respecting (s,t)-mincuts. First, we root the tree T at a centroid, and recursively solve the problem on each subtree, finding a new vertex s if necessary. By the choice of the centroid, each subtree contains half as many vertices, so the recursion depth is logarithmic. We show that each recursive call preserves the k-respecting property for (s,t)-mincuts for vertices t in the targeted subtree. However, in general, this is too expensive since the entire graph G is being used in each recursive call, and there can be many subtrees (and a correspondingly large number of recursive calls). Nevertheless, we show that this strategy can be made efficient *when all the cut edges are in the same subtree* by an application of the Isolating Cuts Lemma from [8], [53] and suitably contracting the graph in each recursive call.

This leaves us with the case where the cut edges are spread across multiple subtrees. Here, we use a different recursive strategy. Consider the subtrees rooted at the children of the centroid. We randomly remove a subset of these subtrees, and recursively solve the problem on the remaining tree with a smaller value of k. (We ensure that the subtree containing s, if it exists, is never removed.) Note that this effectively turns our challenge in working with Steiner trees vis-à-vis spanning trees into an advantage; if we were working on spanning trees, sampling and removing subtrees would have violated the spanning property. This strategy works directly when there exists at least one cut edge in a subtree other than those containing s and t; then, with constant probability, we remove this subtree but not the ones containing s, t to reduce k by at least 1. The more tricky situation is if the cut edges are only in the subtrees of s and t; this requires a more intricate procedure involving a careful relabeling of the source vertex s using a Cut Threshold Lemma from [54].

The algorithm is presented in detail in Section II, and we state here its guarantees.

Theorem 10 (Single-Source Mincuts given a Guide Tree)**.** *Let $G = (V, E, w)$ be a weighted graph, let T be a tree defined on (some subset of) vertices in V, and let s be a vertex in T. For any fixed integer $k \geq 2$, there is a Monte-Carlo algorithm*

[4]While decremental algorithms for approximate single-source shortest paths have been known since [44], the algorithm of [23] is the first to work against *adaptive* adversaries, which is required for the MWU framework. In particular, their algorithm is deterministic.

that finds, for each vertex $t \neq s$ in T, a value $\tilde{\lambda}(t) \geq \lambda(s,t)$ such that $\tilde{\lambda}(t) = \lambda(s,t)$ if T is k-respecting an (s,t)-mincut. The algorithm takes $m^{1+o(1)}$ time.

d) Remarks: The algorithm in Theorem 10 calls max-flow on instances of maximum number of m edges and n vertices and total number of $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices, and spends $\tilde{O}(m)$ time outside these calls. The number of logarithmic factors hidden in the $\tilde{O}(\cdot)$ depends on k. Note that the running time of the algorithm is $m^{1+o(1)}$ even when G is a weighted graph.

e) Putting it all together: Proof of Theorem 3 and Theorem 4: The three ingredients above suffice to prove our main theorems. By Lemma 6, it suffices to solve the single-source mincut problem (Problem 5). Given an instance of Problem 5 on a graph G with terminal set U, we use Theorem 9 to obtain a 4-respecting set of $O(\log n)$ guide trees. We call the algorithm in Theorem 10 for each of the $O(\log n)$ trees separately and keep, for each $t \in U \setminus \{s\}$, the minimum $\tilde{\lambda}(s,t)$ found over all the $O(\log n)$ trees.

The running time of the final algorithm equals that of max-flow calls on graphs with at most $O(m)$ edges and $O(n)$ vertices each, and total number of $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices. In addition, the algorithm takes $\tilde{O}(n^2)$ time outside of these calls (in Theorem 9); in unweighted graphs, the additional time is only $m^{1+o(1)}$.

II. Single-Source Mincuts Given a Guide Tree

In this section, we present our single-source mincut algorithm (SSMC) given a guide tree, which proves Theorem 10.

Before describing the algorithm, we state two tools we will need. The first is the *Isolating-Cuts* procedure introduced by Li and Panigrahi [53] and independently by Abboud, Krauthgamer, and Trabelsi [8]. (Within a short time span, this has found multiple interesting applications [6], [9], [28], [29], [52], [54], [55], [59], [72].)

Recall that for a vertex set $S \subseteq V$, $\delta(S)$ denotes the total weight of edges with exactly one endpoint in S (i.e., the value of the cut $(S, V \setminus S)$). For any two disjoint vertex sets $A, B \subseteq V$, we say that S is an (A,B)-*cut* if $A \subseteq S$ and $B \cap S = \emptyset$ or $B \subseteq S$ and $A \cap S = \emptyset$. In other words, the cut S "separates" the vertex sets A and B. We say that S is an (A,B)-*mincut* if it is an (A,B)-cut of minimum value, and let $\lambda(A,B)$ denote the value of an (A,B)-mincut. As described earlier, if A and B are singleton sets, say $A = \{a\}$ and $B = \{b\}$, then we use the shortcut (a,b)-mincut to denote an (A,B)-mincut, and use $\lambda(a,b)$ to denote the value of an (a,b)-mincut.

We now state the isolating cuts lemma from [8], [53]:

Lemma 11 (Isolating Cuts Lemma: Theorem 2.2 in [53], also follows from Lemma 3.4 in [8])**.** *There is an algorithm that, given a graph $G = (V,E,w)$ and a collection $\mathcal{U}$ of disjoint terminal sets $U_1, \ldots, U_h \subseteq V$, computes a $(U_i, \cup_{j \neq i} U_j)$-mincut for every $U_i \in \mathcal{U}$. The algorithm calls max-flow on graphs that cumulatively contain $O(m \log h)$ edges and $O(n \log h)$ vertices, and spends $\tilde{O}(m)$ time outside these calls.*

a) Remark: The isolating cuts lemma stated above slightly generalizes the corresponding statement from [8], [53]. In the previous versions, each of the sets $U_1, U_2, \ldots, U_h$ is a distinct singleton vertex in V. The generalization to disjoint sets of vertices is trivial because we can contract each set U_i for $i \in [h]$ and then apply the original isolating cuts lemma to this contracted graph to obtain Lemma 11.

We call each $(U_i, \cup_{j \neq i} U_j)$-mincut S_i a *minimum isolating cut* because it "isolates" U_i from the rest of the terminal sets, using a cut of minimum size. The advantage of this lemma is that it essentially only costs $O(\log h)$ max-flow calls, which is an exponential improvement over the naïve strategy of running h max-flow calls, one for each U_i.

The next tool is the *Cut-Threshold* procedure of Li and Panigrahi, which has been used earlier in the approximate Gomory-Hu tree problem [54] and in edge connectivity augmentation and splitting off algorithms [28].

Lemma 12 (Cut-Threshold Lemma: Theorem 1.6 in [54])**.** *There is a randomized, Monte-Carlo algorithm that, given a graph $G = (V,E,w)$, a vertex $s \in V$, and a threshold $\overline{\lambda}$, computes all vertices $v \in V$ with $\lambda(s,v) \leq \overline{\lambda}$ (recall that $\lambda(s,v)$ is the size of an (s,v)-mincut). The algorithm calls max-flow on graphs that cumulatively contain $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices, and spends $\tilde{O}(m)$ time outside these calls.*

We use the Cut-Threshold lemma to obtain the following lemma, which is an important component of our final algorithm. At a high level, we simply binary search on the value $\lambda_{\max}$; we leave the details to the full version of the paper.

Lemma 13. *For any subset $U \subseteq V$ of vertices and a vertex $s \notin U$, there is a randomized, Monte-Carlo algorithm that computes $\lambda_{\max} = \max\{\lambda(s,t) : t \in U\}$ as well as all vertices $t \in U$ attaining this maximum, i.e., the vertex set $\arg\max_{t \in U}\{\lambda(s,t)\}$. The algorithm calls max-flow on graphs that cumulatively contain $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices, and spends $\tilde{O}(m)$ time outside these calls.*

b) The SSMC *Algorithm:* Having introduced the main tools, we are now ready to present our SSMC algorithm (see Figure 1). The input to the algorithm is a graph $G = (V,E,w)$ containing a specified vertex s, a (guide) tree T containing s, and a positive integer k. The algorithm is a recursive algorithm, and although the guide tree initially only contains vertices in V, there will be additional vertices (not in V) that are introduced into the guide tree in subsequent recursive calls. To distinguish between these two types of vertices, we define $R(T)$ as the subset of vertices of T that are in V, and call these *real* vertices. We call the vertices of T that are not in V *fake* vertices.

We extend the definition of k-respecting (i.e., Definition 7) to fake vertices as follows:

Definition 14 (Generalized k-respecting)**.** Let $A \subseteq V$ be a cut in $G = (V,E,w)$. Let T be a tree on (some subset of) vertices in V as well as additional vertices not in V. We say that T k-*respects* cut A (and vice versa) if there exists a set

F_A of fake vertices such that T contains at most k edges with exactly one endpoint in $A \cup F_A$; we say that such edges are *cut* by $A \cup F_A$.

We also note that even if all the vertices in T are real vertices, T may not be a subgraph of G.

Recall that our goal is to obtain a value $\tilde{\lambda}(t) \geq \lambda(s,t)$ for every terminal $t \in U \setminus \{s\}$ such that if an (s,t)-mincut k-respects T, then $\tilde{\lambda}(t) = \lambda(s,t)$. We will actually compute $\tilde{\lambda}(t)$ for every real vertex $t \in R(T) \setminus \{s\}$; clearly, this suffices since the input Steiner tree (i.e., at the top level of the recursion) spans all the vertices in U.

The algorithm maintains estimates $\tilde{\lambda}(t)$ of the mincut values $\lambda(s,t)$ for all $t \in R(T) \setminus \{s\}$. The values $\tilde{\lambda}(t)$ are initialized to ∞, and whenever we compute an (s,t)-cut in the graph, we "update" $\tilde{\lambda}(t)$ by replacing $\tilde{\lambda}(t)$ with the value of the (s,t)-cut if it is lower. Formally, we define $\textsc{Update}(t,x) : \tilde{\lambda}(t) \leftarrow \min(\tilde{\lambda}(t), x)$.

We describe the algorithm below. The reader should use the illustration in Figure 1 as a visual description of each step of the algorithm.

1) First, we describe a base case. If $|R(T)|$ is less than some fixed constant, then we simply compute the (s,t)-mincut in G separately for each $t \in R(T) \setminus \{s\}$ using $|R(T)| - 1 = O(1)$ max-flow calls, and run $\textsc{Update}(t, \lambda(s,t))$.
   From now on, assume that $|R(T)|$ is larger than some (large enough) constant.[5]
2) Let c be a *centroid* of the tree T, defined in the following manner: c is a (possibly fake) vertex in T such that if we root T at c, then each subtree rooted at a child of c has at most $|R(T)|/2$ real vertices.[6]
   If $c \in R(T)$ and $s \neq c$, then compute an (s,c)-mincut in G (whose value is denoted λ) using a max-flow call and run $\textsc{Update}(c, \lambda(s,c))$.
3) Root T at c and let $u_1, \ldots, u_\ell$ be the children of c. For each $i \in [\ell]$, let T_i be the subtree rooted at u_i. Recall that $R(T_i)$ denotes the set of real vertices in the respective subtrees T_i for $i \in [\ell]$. (For technical reasons, we ignore subtrees T_i that do not contain any real vertex.) Use Lemma 11 to compute minimum isolating cuts in G with the following terminal sets: (1) $U_i = R(T_i)$ for $i \in [\ell]$. (2) If $c \in R(T)$, then we add an additional set $U_{\ell+1} = \{c\}$. Note that $\cup_i U_i = R(T)$ irrespective of whether $c \in R(T)$ or not.
   Let $S_i \subseteq V$ be the $(U_i, R(T) \setminus U_i)$-mincut in G obtained from Lemma 11. We ignore $S_{\ell+1}$ (if it exists) and proceed with the remaining sets S_i for $i \in [\ell]$ in the next step.
4) For each $i \in [\ell]$, define G_i as the graph G with $V \setminus S_i$ contracted to a single vertex. Now, there are two cases. In the first case, we have $s \in V \setminus S_i$. Then, the contracted vertex for $V \setminus S_i$ is labeled the new s in graph G_i. Correspondingly, define T_i' as the tree T_i with an added edge (s, u_i) (recall that u_i is the root of T_i). In the second case, we have $s \in S_i$. Then, assign a new label c_i to the contracted vertex for $V \setminus S_i$ in G_i. In this case, define T_i' as the tree T_i with an added edge (c_i, u_i), and keep the identity of vertex s unchanged since it is in T_i. (Note that if $s = c$, the only difference is that the second case does not happen for any $i \in [\ell]$.)
   In both cases above, make recursive calls (G_i, T_i', k) for all $i \in [\ell]$. Call $\textsc{Update}(t, \lambda'(s,t))$ for all $t \in R(T_i) \setminus \{s\}$ where the recursive call returns the value $\lambda'(s,t)$ for the variable $\tilde{\lambda}(t)$. Furthermore, if $s \in S_i$, call $\textsc{Update}(t, \lambda'(s,c_i))$ for all $t \in R(T) \setminus R(T_i)$ where the recursive call returns the value $\lambda'(s,c_i)$ for the variable $\tilde{\lambda}(c_i)$.
   If $k = 1$, then we terminate the algorithm at this point, so from now on, assume that $k > 1$.
5) Sample each subtree T_i independently with probability $1/2$ except the subtree containing s (if it exists), which is sampled with probability 1. (If $c = s$, then there is no subtree containing s, and all subtrees are sampled with probability $1/2$.) Let $T^{(5)}$ be the tree T with all (vertices of) non-sampled subtrees deleted. Recursively call $(G, T^{(5)}, k-1)$ and update $\tilde{\lambda}(t)$ for all $t \in R(T^{(5)})$. (Note that $R(T^{(5)})$ denotes the set of real vertices in tree $T^{(5)}$. Moreover, by the sampling procedure, s is always in $R(T^{(5)})$ and hence, the recursion is valid.) Repeat this step for $O(\log n)$ independent sampling trials.
6) Execute this step only if $s \neq c$, and let T_s be the subtree from step (3) containing s. Using Lemma 13, compute the value $\lambda_{\max} = \max\{\lambda(s,t) : t \in R(T) \setminus R(T_s)\}$, as well as all vertices $t \in R(T) \setminus R(T_s)$ attaining this maximum. Update $\tilde{\lambda}(t) = \lambda_{\max}$ for all such t, and arbitrarily select one such t to be labeled s'. Let $T^{(6)}$ be the tree T with (the vertices of) subtree T_s removed. Recursively call $(G, T^{(6)}, k-1)$ where s' is treated as the new s, and update $\tilde{\lambda}(t)$ for all $t \in R(T^{(6)})$.

[5]For example, the constant 10 is more than enough.

[6]A centroid always exists by the following simple argument: take the (real or fake) vertex of T of maximum depth whose subtree rooted at T has at least $|R(T)|/2$ real vertices. By construction, this vertex is a centroid of T, and it can be found in time linear in the number of vertices in the tree using a simple dynamic program.

A. Correctness

First, we use a standard (uncrossing) property of mincuts. For completeness, we prove the lemma below in the full version of the paper.

Lemma 15. *Let $G = (V, E, w)$ be a weighted, undirected graph with vertex subset $U \subseteq V$. For any subsets $\emptyset \subsetneq X \subseteq X' \subsetneq U$ and an $(X', U \setminus X')$-mincut $A' \subseteq V$ of G, there is an $(X, U \setminus X)$-mincut $A \subseteq V$ of G satisfying $A \subseteq A'$.*

Now, we proceed to establish correctness of the SSMC algorithm. Note that $\tilde{\lambda}(t)$ starts with the value ∞, and every time we run $\textsc{Update}(t,x)$, we have that x is the value of some (s,t)-cut in G. Naturally, this would suggest that our estimate $\tilde{\lambda}(t)$ is always an upper bound on the true value

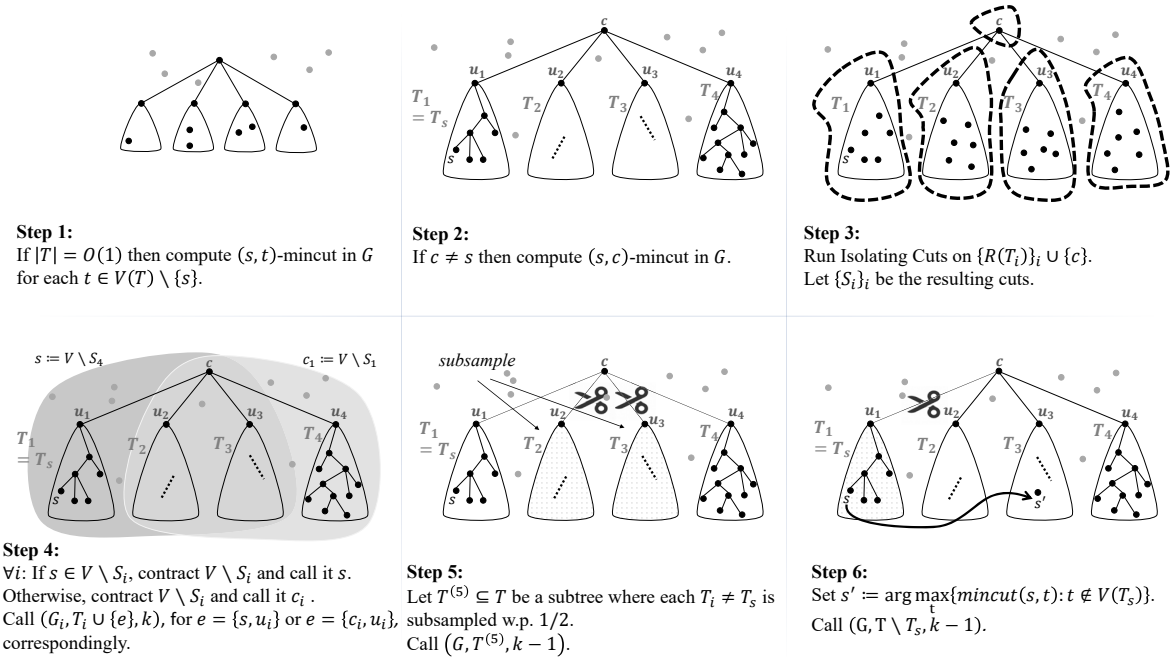


Fig. 1: An illustration of the steps inside a recursive iteration of the SSMC algorithm. We assume that the centroid c has four children in T and that all tree vertices are real; in particular $c \in R(T)$, which simplifies some of the steps. Graph vertices that are not spanned by T are represented by gray dots. The gray areas in Step 4 refer to contracted subsets, and the scissors symbol in Steps 5 and 6 means we remove the subtree.

$\lambda(s,t)$. However, this is not immediately clear because the vertex s may be relabeled in a recursive call from step (6). The lemma below shows that this relabeling is not an issue. We defer the proof to the full version of the paper.

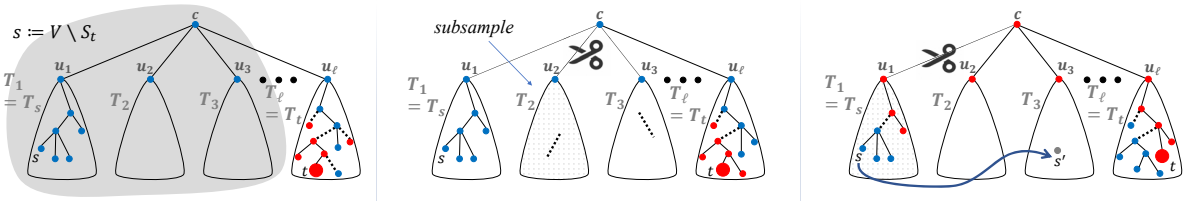
Lemma 16 (Upper bound)**.** *For any instance $(G = (V,E,w), T, k)$ and a vertex $t \in R(T)$, the output value $\tilde{\lambda}(t)$ is at least $\lambda(s,t)$.*

The lemma above establishes the condition $\tilde{\lambda}(t) \geq \lambda(s,t)$ of Theorem 10. It remains to show equality when T is k-respecting an (s,t)-mincut, which we prove below.

Lemma 17 (Equality)**.** *Consider an instance $(G = (V,E,w), T, k)$ and a vertex $t \in R(T)$ such that there is an (s,t)-mincut in G that k-respects T. Then, the value $\tilde{\lambda}(t)$ computed by the algorithm equals $\lambda(s,t)$ w.h.p.*

Proof. Consider an (s,t)-mincut C in G that k-respects T. First, if the centroid c is the vertex t, then the mincut computation in Step (2) correctly recovers $\lambda(s,t)$. Otherwise, let T_t be the subtree containing t. We have a few cases based on the locations of the edges in T that cross the cut C, which we call the *cut* edges. Note that there is at least one cut edge along the (s,t) path in T, and it is incident to (the vertices of) either T_t or the subtree T_s containing s. (If $c = s$ and there is no subtree T_s containing s, then at least one cut edge must be incident on some vertex in T_t.)

The first case (Case 1 in Figure 2) is that all the cut edges are incident to the vertices of a single subtree T_j, which must be either T_t or T_s (if the latter exists). Then, there is a side $A \in \{C, V \setminus C\}$ of the (s,t)-mincut C whose vertices in $R(T)$ are all in $R(T_j)$; in other words, $A \cap R(T) = A \cap R(T_j)$. Note that A is an $(A \cap R(T_j), R(T) \setminus (A \cap R(T_j)))$-mincut since if there were a smaller such cut, then that would also be a smaller (s,t)-cut, which contradicts that A is an (s,t)-mincut. Also, by construction, S_j is a $(R(T_j), R(T) \setminus R(T_j))$-mincut. We now apply Lemma 15 on parameters $U = R(T)$, $A = A$, $X = A \cap R(T_j)$, $A' = S_j$, and $X' = R(T_j)$. The lemma implies that there is an $(A \cap R(T_j), R(T) \setminus (A \cap R(T_j)))$-mincut $\tilde{A} \subseteq S_j$, and this cut survives in the contracted graph G_j. Since $\tilde{A}$ is an (s,t)-cut of the same value as A, we conclude that $\tilde{A}$ is also an (s,t)-mincut. Finally, we argue that $\tilde{A}$ also k-respects the tree T_j' in the recursive instance. By definition, since A k-respects T, there exists a set F_A of fake vertices such that T contains at most k edges cut by $A \cup F_A$. Since A and $\tilde{A}$ agree on vertices in $R(T_j)$, tree T also contains at most k edges cut by $\tilde{A} \cup F_A$ (it is the exact same set of edges). Define $F_{\tilde{A}} = F_A \cap V(T_j)$, and from $\tilde{A} \cap R(T) \subseteq R(T_j)$, we observe that tree T_j contains at most k edges cut by $\tilde{A} \cap F_{\tilde{A}}$ (it is all edges from before, restricted to tree T_j). Furthermore, the new edge (s, u_j) or (c_j, u_j) added to T_j' is cut by $\tilde{A} \cup F_{\tilde{A}}$ if and only if the edge (c, u_j) of T is cut by $\tilde{A} \cup F_A$. It follows that at most k edges of T_j' are cut by $\tilde{A} \cup F_{\tilde{A}}$. Thus, the lemma



Case 1: All cut-edges are in T_t:
- Call (G_t, T_t', k); step **4**.

$$\sum_i f(m_i, n_i, t/2, k)$$

Case 2: There exists a cut-edge outside of $T_s \cup T_t$:
- Subsample each $T' \neq T_s$ w.p. $1/2$ and repeat $O(\log n)$ times; step **5**.

$$O(\log n) \cdot f(m, n, t, k-1)$$

Case 3: There exists a cut-edge in T_s:
- Set $s' \coloneqq \arg\max_t \{mincut(s,t) : t \notin V(T_s)\}$ and remove T_s; step **6**.

$$\tilde{O}(m) + f(m, n, t, k-1)$$

Fig. 2: An illustration of the different cases, which part of our algorithm deals with them, and the corresponding running time. Here, vertices in the side of s are depicted by blue dots, vertices in the side of t by red dots, and cut edges by dashed lines. The gray area refers to a contracted subset, and the scissors symbol means we remove the subtree. Observe that whenever the latter happens, we get rid of at least one cut edge.

statement is satisfied on recursive call (G_j, T_j', k) of Step (4), and the algorithm recovers $\lambda(s,t)$ w.h.p.

In the rest of the proof, we assume that the edges of T cut by $A \cup F_A$ are incident to (the vertices of) at least two subtrees. Suppose first (Case 2 in Figure 2) that a cut edge is incident to some subtree T_j that is not T_t or T_s (or only T_t, if $s = c$ and T_s does not exist). In each independent trial of Step (5), we sample T_t but not T_j with constant probability. In this case, since T_j is discarded in the construction of $T^{(5)}$, the (s,t)-mincut C $(k-1)$-respects the resulting tree $T^{(5)}$. Over $O(\log n)$ independent trials, this happens w.h.p., and the algorithm correctly recovers $\lambda(s,t)$ w.h.p.

We are left with the case (Case 3 in Figure 2) that all edges of T cut by $A \cup F_A$ are incident to subtrees T_t and T_s. Note that T_s must exist since if $s = c$ and Case 2 does not happen, we would be in Case 1. Furthermore, $T_s \neq T_t$, because otherwise, we would either be in Case 1 (if all cut edges are incident on $T_t = T_s$) or in Case 2 (if there is at least one cut edge incident on some $T_j \neq T_t = T_s$).

Since $T_s \neq T_t$, we have $t \notin R(T_s)$, i.e., $t \in R(T) \setminus R(T_s)$. If $\lambda(s,t) = \lambda_{\max}$ (where $\lambda_{\max}$ is as defined in Step (6)), then Step (6) sets $\lambda(s,t) = \lambda_{\max}$ correctly. Otherwise, we must have $\lambda(s,t) < \lambda_{\max}$. In this case, we claim that the vertex s' (that has the property $\lambda(s,s') = \lambda_{\max}$ in Step (6) of the algorithm) satisfies $\lambda(s',t) = \lambda(s,t)$. To prove this claim, we first observe that s' must appear on the s-side of the (s,t)-mincut C. Otherwise, if s' is on the t-side, then C is an (s,s')-cut of value $\lambda(s,t) < \lambda_{\max}$, contradicting the guarantee $\lambda(s,s') = \lambda_{\max}$. It follows that $\lambda(s',t) \leq \lambda(s,t)$. Next, observe that s must appear on the s'-side of the (s',t)-mincut C'. Otherwise, if s is on the t-side, then C' is an (s,s')-cut of value $\lambda(s',t) \leq \lambda(s,t) < \lambda_{\max}$, contradicting the guarantee $\lambda(s,s') = \lambda_{\max}$. It follows that $\lambda(s,t) \leq \lambda(s',t)$, which proves the claim $\lambda(s,t) = \lambda(s',t)$.

Consider again the (s,t)-mincut C. Since s' is on the s-side of the (s,t)-mincut C, if we swap the locations of s and s' in T, then C still k-respects the modified tree, and the edges of the tree that cross the cut are the same (except that s and s' swap places on the edges). In particular, the subtree T_s with s replaced by s' has at least one cut edge. By removing this modified subtree T_s, we arrive at the tree $T^{(6)}$ in Step (6), and the (s,t)-mincut C must $(k-1)$-respect $T^{(6)}$. So, the recursive call $(G, T^{(6)}, k-1)$ recovers $\lambda(s',t)$ w.h.p., which equals $\lambda(s,t)$ by the claim above.

This concludes all cases, and hence the proof of Lemma 17. □

B. Running Time

Lemma 18 (Running time)**.** *For any fixed integer $k \geq 1$, the algorithm calls max-flow on instances of at most n vertices and m edges each, and a total of $\tilde{O}(n)$ vertices and $\tilde{O}(m)$ edges. Moreover, these max-flow calls dominate the running time.*

We first bound the total number of vertices across all recursive instances, then use the same technique to also bound the total number of edges.

We use the following notation for any recursive call: $r = |R(T)|$ and n represents the number of vertices in G including contracted vertices, i.e., vertices resulting from the contraction on Step (4) of any previous instance. (Since the original instance has no contracted vertex, the initial value of n is just the number of vertices in the input graph.) The function $f(n,r,k)$ represents an upper bound on the total number of vertices among all max-flow calls that occur, starting from a single input instance with parameters n, r, k (and including max-flows called inside recursive instances).

Fix an instance with parameters n, r, k. For each i, let n_i represent the number of vertices in G_i, and let $r_i = |R(T_i)|$. Now observe that

1) $\sum_{i=1}^{\ell}(n_i - 1) = n - 1$ since the sets S_i are disjoint by the guarantee of Lemma 11, and
2) $r_i \leq r/2$ for each $i \in [\ell]$ by the fact that c is a centroid.

We now consider the individual steps of the recursive SSMC algorithm.

1) The algorithm calls a single max-flow in step (2), and then in step (3), it calls Lemma 11, which in turn calls max-flows on a total of $O(n \log \ell)$ vertices. In total, this is $O(n \log \ell)$ vertices among the max-flow calls.
2) In step (4), the algorithm makes recursive calls on trees T'_i containing $r_i + 1$ real vertices each, and graphs G_i containing n_i vertices each, so the total number of vertices in the max-flow calls in the recursion is at most $\sum_{i \in [\ell]} f(n_i, r_i + 1, k)$.
3) In step (5), the algorithm makes $O(\log n)$ independent calls to an instance where k decreases by 1. So, this step contributes at most $O(\log n) \cdot f(n, r, k-1)$.
4) In step (6), the algorithm calls Lemma 13, which in turn calls max-flows on a total of $\tilde{O}(n)$ vertices, followed by a recursive call on an instance where k decreases by 1. In total, this step contributes at most $\tilde{O}(n) + f(n, r, k-1)$.

We may assume that $f(n, r, k)$ is monotone non-decreasing in all three parameters, which gives us the recursive formula

$$f(n,r,k) \leq \underbrace{O(n \log \ell)}_{\text{steps (2),(3)}} + \underbrace{\sum_{i \in [\ell]} f(n_i, r_i + 1, k)}_{\text{step (4)}} + \underbrace{O(\log n) \cdot f(n,r,k-1)}_{\text{step (5), only for } k>1} + \underbrace{\tilde{O}(n) + f(n,r,k-1)}_{\text{step (6), only for } k>1}.$$

We now claim that $f(n, r, k)$ solves to $\tilde{O}(n)$ for any constant k, where the number of polylog terms depends on k. For $k = 1$, the recursive formula $f(n, t, 1)$ solves to $O(n \log^2 t)$. This is because $r_i + 1 \leq r/2 + 1 \leq \frac{2}{3} r$ for all $i \in [\ell]$ limits the recursive depth to $O(\log t)$.[7] And, since $\sum_{i=1}^{\ell} (n_i - 1) = n - 1$, the sum of $f(\cdot)$ in any recursive level is $O(n \log t)$. For larger k, note that if we assume that $f(n, r, k-1) \leq \tilde{O}(n)$, then we also obtain $f(n, r, k) \leq \tilde{O}(n)$, where the $\tilde{O}(\cdot)$ hides more logarithmic factors. The claim then follows by induction on k. (Note that the polylogarithmic dependency on k is $f(n, r, k) = n(\log n)^{O(k)}$.)

We now bound the total number of edges. We use the following notation in any recursive call: as earlier, $r = |R(T)|$ and n represents the number of vertices in G including contracted vertices. In addition, m' represents n plus the number of edges in G *not incident to a contracted vertex*. (Since the original instance has no contracted vertex, the initial value of m' is just the number of vertices plus the number of edges in the input graph.) The function $g(m', n, r, k)$ represents an upper bound on $f(n, r, k)$ plus the total number of edges not incident to contracted vertices among all max-flow calls that occur, starting from a single input instance with parameters m', n, r, k (and including max-flows called inside recursive instances). This then implies a bound on the total number of edges over all max-flow calls, including those incident to contracted vertices, by the following argument. Each recursive instance has at most $O(\log |R(T)|)$ contracted vertices, since each contraction in Step (4) decreases $|R(T)|$ by a constant factor. So the total number of edges incident to contracted vertices is at most the total number of vertices times $O(\log |R(T)|)$, which is at most $f(n, r, k) \cdot O(\log n) \leq g(m', n, r, k) \cdot O(\log n)$. So from now on, we only focus on edges not incident to contracted vertices.

Fix an instance with parameters m', n, r, k. For each i, let n_i represent the number of vertices in G_i, let m'_i represent the number of edges in G_i not incident to a contracted vertex, and let $r_i = |R(T_i)|$. Once again, observe that $\sum_{i=1}^{\ell} (n_i - 1) = n - 1$ and $r_i \leq r/2$ for each $i \in [\ell]$. This time, we also have $\sum_{i=1}^{\ell} m'_i \leq m'$ by the following explanation: Lemma 11 guarantees that the vertex sets S_i are disjoint, and the edges of each G_i not incident to a contracted vertex have both endpoints in S_i, and are therefore disjoint over all i. We may assume that $g(m, n, r, k)$ is monotone non-decreasing in all four parameters, which gives us the recursive formula

$$g(m',n,r,k) \leq \underbrace{O((m+n) \log \ell)}_{\text{steps (2),(3)}} + \underbrace{\sum_{i \in [\ell]} g(m_i, n_i, r_i, k)}_{\text{step (4)}} + \underbrace{O(\log n) \cdot g(m,n,r,k-1)}_{\text{step (5), only for } k>1} + \underbrace{\tilde{O}(m) + g(m,n,r,k-1)}_{\text{step (6), only for } k>1}.$$

Similar to the solution for $f(n, r, k)$, we now have that $g(m, n, r, k)$ solves to $\tilde{O}(m + n)$ for any constant k by the same inductive argument. (Once again, the polylogarithmic dependency on k is $f(n, r, k) = (m + n)(\log n)^{O(k)}$.)

Since the graph never increases in size throughout the recursion, each max-flow call is on a graph with at most as many vertices and edges as the original input graph. Finally, we claim that the max-flow calls dominate the running time of the algorithm. In particular, finding the centroid of T on step (2) can be done in time in the size of the tree (see the footnote at step (2)), which is dominated by the single max-flow call on the same step. This finishes the proof of Lemma 18.

III. Constructing Guide Trees

In this section, we show how to obtain guide trees that prove Theorem 9. Our algorithm is based on the notion of a *Steiner subgraph packing*, as described next.

Definition 19. Let $G = (V, E, w)$ be an undirected edge-weighted graph with a set of terminals $U \subseteq V$. A subgraph H of G is said to be a *U-Steiner subgraph* (or simply a Steiner subgraph if the terminal set U is unambiguous from the context) if all the terminals are connected in H. In this case, we also call H a *terminal-spanning* subgraph of G.

Definition 20. A *U-Steiner-subgraph packing* $\mathcal{P}$ is a collection of U-Steiner subgraphs $H_1, \ldots, H_k$, where each subgraph

[7] Here, we have used the assumption that r is larger than some constant, e.g. 10.

H_i is assigned a value $\text{val}(H_i) > 0$. If all $\text{val}(H_i)$ are integral, we say that $\mathcal{P}$ is an *integral packing*. Throughout, a packing is assumed to be *fractional* (which means that it does not have to be integral), unless specified otherwise. The *value* of the packing $\mathcal{P}$ is the total value of all its Steiner subgraphs, denoted $\text{val}(\mathcal{P}) = \sum_{H\in\mathcal{P}} \text{val}(H)$. We say that $\mathcal{P}$ is *feasible* if

$$\forall e \in E, \quad \sum_{H\in\mathcal{P}: e\in H} \text{val}(H) \leq w(e).$$

To understand this definition, think of $w(e)$ as the "capacity" of e; then, this condition means that the total value of all Steiner subgraphs $H \in \mathcal{P}$ that "use" edge e does not exceed its capacity $w(e)$. A *Steiner-tree packing* is a packing $\mathcal{P}$ where each subgraph $H \in \mathcal{P}$ is a tree.

Denote by $\text{pack}(U)$ the maximum value of a feasible U-Steiner-subgraph packing in G. The next two lemmas show a close connection between Steiner-subgraph packing $\text{pack}(U)$ and U-Steiner mincut $\lambda(U)$, and that the former problem admits a $(2+\epsilon)$-approximation algorithm.

Lemma 21. *For every graph G with terminal set U, we have $\lambda(U)/2 \leq \text{pack}(U) \leq \lambda(U)$.*

Lemma 22. *There is a deterministic algorithm that, given $\epsilon \in (0, 1/2)$, a graph $G = (V, E, w)$, and a terminal set $U \subseteq V$, returns a U-Steiner-subgraph packing $\mathcal{P}$ of value $\text{val}(\mathcal{P}) \geq \text{pack}(U)/(2+\epsilon)$ in $\tilde{O}(m^2/\epsilon^2)$ time, and in the case of unweighted G in $m^{1+o(1)}/\epsilon^2$ time.*

We defer Lemmas 21 and 22 to the full version of the paper. Assuming these lemmas, we immediately obtain the following.

Corollary 23. *There is a deterministic algorithm that, given $\epsilon \in (0, 1/2)$ and a graph $G = (V, E, w)$ with m edges and terminal set $U \subseteq V$, returns a U-Steiner subgraph packing $\mathcal{P}$ of value $\text{val}(\mathcal{P}) \geq \lambda(U)/(4+\epsilon)$ in $\tilde{O}(m^2/\epsilon^2)$ time, and in the case of unweighted G in $m^{1+o(1)}/\epsilon^2$ time.*

A. Algorithm for constructing guide trees.

Given Corollary 23, we can now prove Theorem 9.

Proof of Theorem 9. Fix $\epsilon_0 = 1/60$ (or another sufficiently small $\epsilon_0 > 0$). The construction of guide trees is described in Algorithm 1. To analyze this algorithm, let $\mathcal{P}$ be the packing computed in line 5. Consider $t \in U \setminus \{s\}$, and let $(S_t, V \setminus S_t)$ be a minimum s,t-cut in G. Denote by $w(S_t, V \setminus S_t)$ the total edge-weight of this cut in G, and by $w'(S_t, V \setminus S_t)$ the total edge-weight of the cut in G' between these same vertices.

Consider first an unweighted input G. Then, the computation in line 5 is applied to $G' = G$. By combining Corollary 23 and the promise in the single source terminal mincuts problem (Problem 5) that $\lambda_G(s,t) \leq 1.1\lambda(U)$, [8] we get that

$$\text{val}(\mathcal{P}) \geq \frac{\lambda(U)}{4+\epsilon_0} \geq \frac{\lambda_G(s,t)}{1.1(4+\epsilon_0)} = \frac{w'(S_t, V\setminus S_t)}{1.1(4+\epsilon_0)} . \quad (1)$$

[8] For a graph G', $\lambda_{G'}(s,t)$ denotes the value of an (s,t)-mincut in G', and recall that $\lambda(U)$ is the value of a U-Steiner mincut in G.

Algorithm 1: An algorithm for constructing guide trees

input : Undirected graph $G = (V, E, w)$ (weighted or unweighted) and terminal set $U \subseteq V$
output: A collection of guide trees for U

1 **if** *G is weighted then* **then**
2 **compute** for it a $(1 \pm \epsilon_0)$-cut-sparsifier G' using [22] and $\epsilon_0 = 1/60$, thus G' has $m = \tilde{O}(n/\epsilon_0^2)$ edges
3 **else**
4 **let** $G' \leftarrow G$
5 **compute** a packing $\mathcal{P}$ for G' by applying Corollary 23
6 **sample** $300 \ln n$ subgraphs from $\mathcal{P}$, each drawn independently from the distribution $\{\text{val}(H)/\text{val}(\mathcal{P})\}_{H\in\mathcal{P}}$
7 **compute** any Steiner tree of each sampled subgraph, and report these trees

If the input graph G is weighted, then the bound in (1) applies to the cut-sparsifier G' of G, and we get that

$$\text{val}(\mathcal{P}) \geq \frac{\lambda_{G'}(s,t)}{1.1(4+\epsilon_0)} \geq \frac{(1-\epsilon_0)\cdot w(S_t, V\setminus S_t)}{1.1(4+\epsilon_0)} \quad (2)$$

$$\geq \frac{(1-\epsilon_0)\cdot w'(S_t, V\setminus S_t)}{1.1(4+\epsilon_0)(1+\epsilon_0)} \geq \frac{w'(S_t, V\setminus S_t)}{1.1(4+30\epsilon_0)} . \quad (3)$$

We remark that now the packing $\mathcal{P}$ contains subgraphs of the sparsifier G' and not of G, but it will not pose any issue.

In both cases we have the weaker inequality

$$\text{val}(\mathcal{P}) \geq \frac{w'(S_t, V\setminus S_t)}{1.1(4+30\epsilon_0)} . \quad (4)$$

Let E'_t be the set of edges in the cut $(S_t, V \setminus S_t)$ in G' (depending on the case, G' is either G or the sparsifier). Let $\mathcal{P}_{\leq 4} \subseteq \mathcal{P}$ be the subset of all Steiner subgraphs $H \in \mathcal{P}$ whose intersection with E'_t is at most 4 edges, and let F_t be the event that no subgraph from $\mathcal{P}_{\leq 4}$ is sampled in line 6. Then

$$\Pr[F_t] = (1 - \text{val}(\mathcal{P}_{\leq 4})/\text{val}(\mathcal{P}))^{300\ln n}$$
$$\leq n^{-300\cdot \text{val}(\mathcal{P}_{\leq 4})/\text{val}(\mathcal{P})}. \quad (5)$$

Similarly to Karger's paper [47], define x_H to be one less than the number of edges of H that crosses the cut E'_t, and observe that x_H is always a non-negative integer (because U is connected in H). Since $\mathcal{P}$ is a packing, every edge of E'_t appears in at most one subgraph of $\mathcal{P}$, and consequently,

$$\sum_{H\in\mathcal{P}} \text{val}(H)(x_H+1) \leq \sum_{e\in E'_t} w(e) = w'(S_t, V\setminus S_t)$$
$$\implies \sum_{H\in\mathcal{P}} \text{val}(H) x_H \leq w'(S_t, V\setminus S_t) - \sum_{H\in\mathcal{P}} \text{val}(H)$$
$$= w'(S_t, V\setminus S_t) - \text{val}(\mathcal{P}).$$

Observe that for a random $\bar{H} \in \mathcal{P}$ drawn as in line 6,

$$\mathbb{E}_{\bar{H}}[x_{\bar{H}}] = \sum_{H \in \mathcal{P}} x_H \cdot \frac{\mathrm{val}(H)}{\mathrm{val}(\mathcal{P})} \leq \frac{w'(S_t, V \setminus S_t)}{\mathrm{val}(\mathcal{P})} - 1 \leq 1.1(4 + 30\epsilon_0) - 1,$$

where the last inequality is by (4). By Markov's inequality,

$$\Pr_{\bar{H}}[x_{\bar{H}} \geq 4] \leq \frac{1.1(4 + 30\epsilon_0) - 1}{4} \leq 0.99.$$

Observe that $\mathrm{val}(\mathcal{P}_{\leq 4})/\mathrm{val}(\mathcal{P}) = \Pr_{\bar{H}}[x_{\bar{H}} < 4] \geq 0.01$, and so by plugging into (5) we get that $\Pr[F_t] \leq 1/n^3$. Finally, by a union bound we have that with high probability, for every $t \in U \setminus \{s\}$, at least one of the subgraphs that are sampled in line 6 of the algorithm 4-respects the cut E'_t,[9] and thus at least one of the trees reported by the algorithm 4-respects E'_t. Furthermore, since the cut E'_t in G' has the exact same bipartition of V as the (s,t)-mincut in G, the reported tree mentioned above 4-respects also the (s,t)-mincut in G (recall that Definition 7 refers to a cut as a bipartition of V).

Finally, computing a Steiner tree of a Steiner subgraph in Line 7 only takes linear time, and so the running time is dominated by line 5 and thus it is bounded by $\tilde{O}(n^2/\epsilon^2)$ for weighted graphs and by $m^{1+o(1)}/\epsilon^2$ for unweighted graphs, and by fixing a small $\epsilon > 0$, we can write these as $\tilde{O}(n^2)$ and $m^{1+o(1)}$, respectively. This concludes the proof of Theorem 9. □

IV. Conclusion

In this paper, we broke the longstanding cubic barrier for the GHTREE and APMF problems in general, weighted graphs. All previous improvements since 1961, were either corollaries of speed-ups in (single-pair) max-flow algorithms, or were limited to special graph classes. Assuming the APSP Conjecture, a cornerstone of fine-grained complexity, our result disproves the belief that computing max-flows is at least as hard as computing shortest-paths.

Our algorithm has a running time of $\tilde{O}(n^2)$ which is nearly-optimal for APMF if all $\binom{n}{2}$ max-flow values must be returned. For GHTREE in unweighted graphs, our techniques yield an improved running time of $m^{1+o(1)}$. In fact, a succinct representation of all-pairs max-flows can be produced in the time of $\tilde{O}(1)$ calls to a (single-pair) max-flow algorithm.

It is an interesting open question as to whether our techniques can be extended to obtain an $m^{1+o(1)}$-time GHTREE algorithm for weighted graphs as well. In particular, the most significant challenge is to design a new dynamic Steiner tree subroutine that can be used for packing Steiner trees in weighted graphs in almost-linear time.

[9] Strictly speaking, Definition 7 defines 4-respecting only relative to a tree T, but the same wording extends immediately to any graph T (not necessarily a tree).

References

[1] Abboud, A., Cohen-Addad, V., and Klein, P. N. New hardness results for planar graph problems in p and an algorithm for sparsest cut. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020* (2020), K. Makarychev, Y. Makarychev, M. Tulsiani, G. Kamath, and J. Chuzhoy, Eds., ACM, pp. 996–1009.

[2] Abboud, A., and Dahlgaard, S. Popular conjectures as a barrier for dynamic planar graph algorithms. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA* (2016), I. Dinur, Ed., IEEE Computer Society, pp. 477–486.

[3] Abboud, A., Georgiadis, L., Italiano, G. F., Krauthgamer, R., Parotsidis, N., Trabelsi, O., Uznanski, P., and Wolleb-Graf, D. Faster Algorithms for All-Pairs Bounded Min-Cuts. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)* (2019), vol. 132, pp. 7:1–7:15.

[4] Abboud, A., Grandoni, F., and Williams, V. V. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015* (2015), P. Indyk, Ed., SIAM, pp. 1681–1697.

[5] Abboud, A., Krauthgamer, R., and Trabelsi, O. Cut-equivalent trees are optimal for min-cut queries. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020* (2020), pp. 105–118.

[6] Abboud, A., Krauthgamer, R., and Trabelsi, O. APMF < APSP? Gomory-Hu tree for unweighted graphs in almost-quadratic time. *Accepted to FOCS'21* (2021). arXiv:2106.02981.

[7] Abboud, A., Krauthgamer, R., and Trabelsi, O. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. *Theory of Computing 17*, 5 (2021), 1–27.

[8] Abboud, A., Krauthgamer, R., and Trabelsi, O. Subcubic algorithms for gomory–hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2021), pp. 1725–1737.

[9] Abboud, A., Krauthgamer, R., and Trabelsi, O. Friendly cut sparsifiers and faster Gomory-Hu trees. *Accepted to SODA'22* (2022). arXiv:2110.15891.

[10] Abboud, A., Vassilevska Williams, V., and Yu, H. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. of 47th STOC* (2015), pp. 41–50.

[11] Abboud, A., and Williams, V. V. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014* (2014), pp. 434–443.

[12] Ahuja, R., Magnanti, T., and Orlin, J. *Network Flows*. Prentice Hall, 1993.

[13] Akiba, T., Iwata, Y., Sameshima, Y., Mizuno, N., and Yano, Y. Cut tree construction from massive graphs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), IEEE, pp. 775–780.

[14] Alman, J., and Williams, V. V. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021* (2021), pp. 522–539.

[15] Anari, N., and Vazirani, V. V. Planar graph perfect matching is in NC. *Journal of the ACM 67*, 4 (2020), 1–34.

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

[17] Arora, S., Hazan, E., and Kale, S. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing 8*, 1 (2012), 121–164.

[18] Backurs, A., Dikkala, N., and Tzamos, C. Tight hardness results for maximum weight rectangles. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy* (2016), I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, Eds., vol. 55 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 81:1–81:13.

[19] Backurs, A., and Tzamos, C. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In *International Conference on Machine Learning* (2017), PMLR, pp. 311–321.

[20] Barth, D., Berthomé, P., Diallo, M., and Ferreira, A. Revisiting parametric multi-terminal problems: Maximum flows, minimum cuts and cut-tree computations. *Discrete Optimization 3*, 3 (2006), 195–205.

[21] Baswana, S., Gupta, S., and Knollmann, T. Mincut sensitivity data structures for the insertion of an edge. In *28th Annual European Symposium on Algorithms (ESA 2020)* (2020).

[22] Benczúr, A. A., and Karger, D. R. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput. 44*, 2 (2015), 290–319.

[23] BERNSTEIN, A., GUTENBERG, M. P., AND SARANURAK, T. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)* (2022), IEEE, pp. 1000–1008.

[24] BHALGAT, A., HARIHARAN, R., KAVITHA, T., AND PANIGRAHI, D. An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In *39th Annual ACM Symposium on Theory of Computing* (2007), STOC'07, pp. 605–614.

[25] BORRADAILE, G., EPPSTEIN, D., NAYYERI, A., AND WULFF-NILSEN, C. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *32nd International Symposium on Computational Geometry* (2016), vol. 51 of *SoCG '16*, pp. 22:1–22:16.

[26] BORRADAILE, G., SANKOWSKI, P., AND WULFF-NILSEN, C. Min st-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms 11*, 3 (2015).

[27] BRINGMANN, K., GAWRYCHOWSKI, P., MOZES, S., AND WEIMANN, O. Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). *ACM Transactions on Algorithms (TALG) 16*, 4 (2020), 1–22.

[28] CEN, R., LI, J., AND PANIGRAHI, D. Augmenting edge connectivity via isolating cuts. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022* (2022).

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

[30] CHEN, L., KYNG, R., LIU, Y. P., PENG, R., GUTENBERG, M. P., AND SACHDEVA, S. Maximum flow and minimum-cost flow in almost-linear time. *CoRR abs/2203.00671* (2022).

[31] COOK, W., CUNNINGHAM, W., PULLEYBANK, W., AND SCHRIJVER, A. *Combinatorial Optimization*. Wiley, 1997.

[32] CYGAN, M., MUCHA, M., WEGRZYCKI, K., AND WŁODARCZYK, M. On problems equivalent to (min,+)-convolution. *ACM Transactions on Algorithms (TALG) 15*, 1 (2019), 1–25.

[33] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[34] ELMAGHRABY, S. E. Sensitivity analysis of multiterminal flow networks. *Operations Research 12*, 5 (1964), 680–688.

[35] FLEISCHER, L. K. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics 13*, 4 (2000), 505–520.

[36] GARG, N., AND KÖNEMANN, J. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing 37*, 2 (2007), 630–652.

[37] GAWRYCHOWSKI, P., MOZES, S., AND WEIMANN, O. Planar negative k-cycle. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021* (2021), D. Marx, Ed., SIAM, pp. 2717–2724.

[38] GOLDBERG, A. V., AND RAO, S. Beyond the flow decomposition barrier. *J. ACM 45*, 5 (1998), 783–797.

[39] GOLDBERG, A. V., AND TSIOUTSIOULIKLIS, K. Cut tree algorithms: an experimental study. *Journal of Algorithms 38*, 1 (2001), 51–83.

[40] GOMORY, R. E., AND HU, T. C. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics 9* (1961), 551–570.

[41] GUSFIELD, D. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing 19*, 1 (1990), 143–155.

[42] HARTMANN, T., AND WAGNER, D. Dynamic Gomory–Hu tree construction–fast and simple. *arXiv preprint arXiv:1310.0178* (2013).

[43] HASSIN, R., AND LEVIN, A. Flow trees for vertex-capacitated networks. *Discrete Appl. Math. 155*, 4 (2007), 572–578.

[44] HENZINGER, M., KRINNINGER, S., AND NANONGKAI, D. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science* (2014), IEEE, pp. 146–155.

[45] HU, T. C. Optimum communication spanning trees. *SIAM Journal on Computing 3*, 3 (1974), 188–195.

[46] JELINEK, F. On the maximum number of different entries in the terminal capacity matrix of oriented communication nets. *IEEE Transactions on Circuit Theory 10*, 2 (1963), 307–308.

[47] KARGER, D. R. Minimum cuts in near-linear time. *Journal of the ACM 47*, 1 (2000), 46–76.

[48] KARGER, D. R., AND LEVINE, M. S. Fast augmenting paths by random sampling from residual graphs. *SIAM J. Comput. 44*, 2 (2015), 320–339.

[49] KORTE, B., AND VYGEN, J. *Combinatorial optimization*, vol. 2. Springer, 2012.

[50] KRAUTHGAMER, R., AND TRABELSI, O. Conditional lower bounds for all-pairs max-flow. *ACM Trans. Algorithms 14*, 4 (2018), 42:1–42:15.

[51] LI, J. *Preconditioning and Locality in Algorithm Design*. PhD thesis, Carnegie Mellon University, 2021.

[52] LI, J., NANONGKAI, D., PANIGRAHI, D., SARANURAK, T., AND YINGCHAREONTHAWORNCHAI, S. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2021), pp. 317–329.

[53] LI, J., AND PANIGRAHI, D. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020* (2020), pp. 85–92.

[54] LI, J., AND PANIGRAHI, D. Approximate Gomory-Hu tree is faster than $n-1$ max-flows. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2021), ACM, pp. 1738–1748.

[55] LI, J., PANIGRAHI, D., AND SARANURAK, T. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *Accepted to FOCS'21* (2021). arXiv:2106.02233.

[56] MADER, W. A reduction method for edge-connectivity in graphs. In *Advances in Graph Theory*, B. Bollobás, Ed., vol. 3 of *Annals of Discrete Mathematics*. Elsevier, 1978, pp. 145–164.

[57] MAYEDA, W. On oriented communication nets. *IRE Transactions on Circuit Theory 9*, 3 (1962), 261–267.

[58] MEHLHORN, K. A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters 27*, 3 (1988), 125–128.

[59] MUKHOPADHYAY, S., AND NANONGKAI, D. A note on isolating cut lemma for submodular function minimization. *arXiv preprint arXiv:2103.15724* (2021).

[60] NASH-WILLIAMS, C. S. A. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society s1-36*, 1 (01 1961), 445–450.

[61] NAVES, G., AND SHEPHERD, F. B. When do Gomory-Hu subtrees exist? *CoRR* (2018).

[62] PADBERG, M. W., AND RAO, M. R. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research 7*, 1 (1982), 67–80.

[63] PANIGRAHI, D. Gomory-Hu trees. In *Encyclopedia of Algorithms*, M.-Y. Kao, Ed. Springer New York, 2016, pp. 858–861.

[64] PICARD, J.-C., AND QUEYRANNE, M. On the structure of all minimum cuts in a network and applications. In *Combinatorial Optimization II*. Springer, 1980, pp. 8–16.

[65] RODITTY, L., AND ZWICK, U. On dynamic shortest paths problems. In *European Symposium on Algorithms* (2004), Springer, pp. 580–591.

[66] SAHA, B. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015* (2015), V. Guruswami, Ed., IEEE Computer Society, pp. 118–135.

[67] SCHRIJVER, A. *Combinatorial Optimization*. Springer, 2003.

[68] SEIDEL, R. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences 51*, 3 (1995), 400–403.

[69] TUTTE, W. T. On the Problem of Decomposing a Graph into n Connected Factors. *Journal of the London Mathematical Society s1-36*, 1 (01 1961), 221–230.

[70] WILLIAMS, V. V., AND WILLIAMS, R. R. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM 65*, 5 (2018), 27:1–27:38.

[71] WU, Z., AND LEAHY, R. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE transactions on pattern analysis and machine intelligence 15*, 11 (1993), 1101–1113.

[72] ZHANG, T. Faster Cut-Equivalent Trees in Simple Graphs. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)* (Dagstuhl, Germany, 2022), vol. 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 109:1–109:18.