All-Pairs Max-Flow is no Harder than Single-Pair Max-Flow: Gomory-Hu Trees in Almost-Linear Time

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

Jason Li
University of California, Berkeley
jmli@cs.cmu.edu

Debmalya Panigrahi
Duke University
debmalya@cs.duke.edu

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

***Abstract*—A Gomory-Hu tree (also called a cut tree) succinctly represents (s,t) min-cuts (and therefore, (s,t) max-flow values) of all pairs of vertices s,t in an undirected graph. In this paper, we give an $m^{1+o(1)}$-time algorithm for constructing a Gomory-Hu tree for a graph with m edges. This shows that the all-pairs max-flows problem has the same running time as the single-pair max-flow problem, up to a subpolynomial factor. Prior to our work, the best known Gomory-Hu tree algorithm was obtained in recent work by Abboud *et al.* (FOCS 2022) and requires $\tilde{O}(n^2)$ time for a graph with n vertices. Our result marks a natural culmination of over 60 years of research into the all-pairs max-flows problem that started with Gomory and Hu's pathbreaking result introducing the Gomory-Hu tree in 1961.**

***Index Terms*—Gomory-Hu trees, Graph algorithms, All-pairs maximum flows**

I. Introduction

Let $G=(V,E)$ be an undirected graph on n vertices and m edges with nonnegative edge weights $w(e)$ for $e \in E$. (We will assume throughout that the edge weights $w(e)$ are integers that are polynomially bounded in n.)[1] The *all-pairs minimum cuts* (APMC) problem asks for the values[2] of minimum (s,t)-cuts for all pairs of vertices $s,t \in V$. By the classical maxflow-mincut theorem, this is equivalent to the *all-pairs maximum flows* (APMF) problem that asks for the values[3] of the maximum (s,t)-flows. A naïve solution for these problems is obtained by running an (s,t)-maxflow algorithm separately for each pair of vertices $s,t \in V$. This incurs a running time equivalent to $\binom{n}{2}=\Theta(n^2)$ calls to any maxflow subroutine. In a remarkable result in 1961 [GH61], Gomory and Hu showed that one can do much better: the problem can be solved using just $n-1=\Theta(n)$ maxflow calls instead of $\Theta(n^2)$ calls. Their algorithm constructs a tree $T=(V,E_T)$ on the same set of vertices V such that for every pair of vertices $s,t \in V$, an (s,t)-mincut in the tree T is also an (s,t)-mincut in the graph G and these two cuts have the same value. This tree T is called a *cut tree* or a *Gomory-Hu tree* (GHTREE).

The 60 years that followed established the Gomory-Hu result as truly pioneering. In particular, the concept of a *sparse representation* that preserves interesting properties of a graph has developed into an entire field, and their novel use of *submodular minimization* in graphs is now ubiquitous. Notably, theirs was also the first non-trivial algorithm for *global min-cut*, a problem that subsequently became one of the most well-studied in the literature. Moreover, GHTREE algorithms are featured in standard textbooks in combinatorial optimization (e.g., [AMO93], [CCPS97], [Sch03]) and have found several diverse applications. Because of these reasons, and the fundamental nature of the APMC and APMF problems, substantial effort has gone into obtaining faster GHTREE algorithms. More than a dozen different algorithms were designed for the problem, achieving faster running times in restricted settings (see Table I and the survey [Pan16]). Much progress was made in the last few years in the special case of simple (unweighted) graphs [AKT21c], [AKT21a], [LPS21], [Zha22], [AKT22]. But, Gomory and Hu's result stood as the best algorithm for general weighted graphs until a recent result by Abboud *et al.* [AKL+22] broke this longstanding barrier and obtained a GHTREE algorithm that runs in $\tilde{O}(n^2)$ time. This raised the possibility of finally bridging the gap between the running times of finding a single (s,t)-maxflow (or (s,t)-mincut) and all-pairs maxflows (all-pairs mincuts), thereby bringing this long history of the study of GHTREE algorithms to a natural conclusion. In this paper, we achieve this goal by obtaining an almost optimal algorithm for the GHTREE

AA is supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science. This work is part of the project CONJEXITY that has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101078482). DP is supported by NSF Awards CCF-1750140 (CAREER) and CCF-1955703, and ARO Award W911NF2110230. TS is supported by NSF CAREER grant 2238138.

[1] The assumption of polynomially-bounded integer edge weights is not a crucial one. For general edge weights, we incur an additional factor of $\operatorname{poly}\log W$ in the running time, where W is the ratio of the maximum-to-minimum edge weight. In this extended abstract, for simplicity, we make the polynomially-bounded assumption.

[2] The *value* of a cut is the sum of weights of edges in the cut.

[3] The *value* of a flow is the net flow exiting the source, or equivalently, the net flow entering the sink.

problem that runs in $m^{1+o(1)}$ time on general weighted graphs.

Theorem I.1. *There is a randomized Monte Carlo algorithm for the* GHTREE *(and* APMF/APMC*) problem that runs in $m^{1+o(1)}$ time in general weighted graphs.*

This result also settles a basic question in data structures, where a problem is considered considered easy if it can be solved in (almost) linear time preprocessing and (almost) constant query time. Such solutions have long been known for the most basic problems such as connectivity and single-source shortest paths; while strong conditional lower bounds rule out such results for more complex problems such as reachability and (unrestricted) shortest paths. Up until this work it was unclear whether min-cut and max-flow should be classified as easy or hard when viewed through this lens (lower bounds had only been obtained in the dynamic setting [AW14], [HKNS15], [AVY15], [Dah16]).

Corollary I.2 (See [AKT20])**.** *There is a data structure that preprocesses a graph in randomized $m^{1+o(1)}$ time and can return the max-flow or min-cut value for any given pair of nodes in $\tilde{O}(1)$ time. Moreover, it can return the edges of a min-cut in amortized $\tilde{O}(1)$ time per edge.*

To prove Theorem I.1, our algorithm makes (s,t)-maxflow calls on graphs that add up to at most $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices. The running time bound of the overall algorithm then follows by using the recent breakthrough $m^{1+o(1)}$-time (s,t)-maxflow algorithm due to Chen *et al.* [CKL+22]. In fact, the runing time of the algorithm outside the (s,t)-maxflow calls is only an additive $\tilde{O}(m)+n^{1+o(1)}$ which means that in moderately-dense graphs we can use the (s,t)-maxflow algorithm due to van den Brand *et al.* [vdBLL+21] to get near-linear $\tilde{O}(m+n^{1.5})$ time, improving the result of [AKL+22] even in unweighted graphs.

Theorem I.3. *There is a randomized Monte Carlo algorithm for the* GHTREE *(and* APMF/APMC*) problem that runs in $\tilde{O}(m)+n^{1+o(1)}$ time in general weighted graphs and additionally makes a set of calls to any (s,t)-maxflow algorithm on graphs that cumulatively contain $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices.*

A. Related Work

Before the recent work of Abboud *et al.* [AKL+22], the time complexity of constructing a Gomory-Hu tree in general graphs improved over the years as a by-product of improvements in max-flow algorithms. The $\tilde{O}(n^2)$-time Monte Carlo algorithm of Abboud *et al.* [AKL+22] is fundamentally different from (and faster than) Gomory and Hu's algorithm. They also further improved the running time for unweighted graphs to $m^{1+o(1)}$. In this work, we match (and even improve) the unweighted running time of Abboud *et al.* but for general weighted graphs.

Besides the algorithms in Table I there has also been practice-oriented papers proposing faster algorithms for the problem [AIS+16], [Kol22b]; see [Kol22a] for a recent comparative study (and [GT01] for an older one). The applications of GHTREE include social network analysis [AIS+16], computer vision [WL93], telecommunications [Hu74], and even in algorithms for the b-matching problem [PR82], [AV20].

Proving a conditional lower bound for GHTREE has been a well-known open question in fine-grained complexity since the early days of this field. This was only accomplished for closely-related but harder variants, such as APMF in *directed* graphs [KT18], [AGI+19] or undirected graphs with vertex weights [AKT21b]. These settings are fundamentally harder since Gomory–Hu trees may not even exist [May62], [Jel63], [HL07]. In particular, SETH gives an $n^{3-o(1)}$ lower bound for weighted directed graphs even when $m=\tilde{O}(n)$ [KT18]; thus, in sparse graphs, our $m^{1+o(1)}$ algorithm provides a *quadratic* separation between the directed and undirected setting.

II. TECHNICAL OVERVIEW

We now introduce the main technical ideas in our work.

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 $\lambda(a,b)$. Let $G[S]$ denote the subgraph of G induced by S.

A. From Gomory-Hu Tree to Steiner Tree Packing

Recently, Abboud *et al.* [AKL+22] gave the first improvement over the classical algorithm of Gomory and Hu [GH61] for the GHTREE problem. Their algorithm takes $\tilde{O}(n^2)$ time, which improves on the Gomory-Hu bound of $n-1$ max-flow computations. At a high level, our algorithm also uses the same template as [AKL+22], so we describe this first.

a) Reduction to Single Source Problem.: The first step is to reduce the GHTREE problem to a restricted version of the single-source min-cuts problem. We define this latter problem first. 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 II.1 (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 $\lambda(U)\le\lambda(s,t)\le 1.1\lambda(U)$. The goal is to determine the value of $\lambda(s,t)$ for each terminal $t\in U\setminus\{s\}$.*

The following lemma from [AKL+22] states the reduction from GHTREE to the above problem.

Lemma II.2 (Reduction to Single-Source Terminal Mincuts, Lemma 6 in [AKL+22])**.** *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 II.1) on graphs with a total of $\tilde{O}(n)$*

Restriction	Running time	Reference
General	$(n-1)\cdot T(n,m)$	Gomory and Hu [GH61]
General	$(n-1)\cdot T(n,m)$	Gusfield [Gus90]
Bounded Treewidth*	$\tilde{O}(n)$	Arikati, Chaudhuri, and Zaroliagis [ACZ98]
Unweighted	$\tilde{O}(mn)$	Karger and Levine [KL15]
Unweighted	$\tilde{O}(mn)$	Bhalgat, Hariharan, Kavitha, and Panigrahi [BHKP07]
Planar	$\tilde{O}(n)$	Borradaile, Sankowski, and Wulff-Nilsen [BSW15]
Bounded Genus	$\tilde{O}(n)$	Borradaile, Eppstein, Nayyeri, and Wulff-Nilsen [BENW16]
Unweighted	$\tilde{O}(\sqrt{m})\cdot T(n,m)$	Abboud, Krauthgamer, and Trabelsi [AKT21b]
$(1+\epsilon)$-Approx*	$\tilde{O}(n^2)$	Abboud, Krauthgamer, and Trabelsi [AKT20]
Bounded Treewidth	$\tilde{O}(n)$	Abboud, Krauthgamer, and Trabelsi [AKT20]
Simple	$\tilde{O}(n^{2.5})$	Abboud, Krauthgamer, and Trabelsi [AKT21c]
$(1+\epsilon)$-Approx	$\text{polylog}(n)\cdot T(n,m)$	Li and Panigrahi [LP21]
Simple	$\tilde{O}(n^2)$	† Abboud, Krauthgamer, and Trabelsi [AKT21a]
Simple	$\tilde{O}(n^2)$	† Li, Panigrahi, and Saranurak [LPS21]
Simple	$\tilde{O}(n^{2\frac{1}{8}})$	† Zhang [Zha22]
Simple	$\tilde{O}(m+n^{1.9})$	Abboud, Krauthgamer, and Trabelsi [AKT22]
General	$\tilde{O}(n^2)$	# Abboud *et al.* [AKL+22]
Unweighted	$m^{1+o(1)}+\text{polylog}(n)\cdot T(n,m)$	Abboud *et al.* [AKL+22]
General	$\tilde{O}(n^2)$	# Zhang [Zha21]
General	$n^{1+o(1)}+\text{polylog}(n)\cdot T(n,m)$	**Theorem I.3** (this paper)

TABLE I: Algorithms for constructing a data-structure that answers s,t-min-cut queries in $\tilde{O}(1)$ time (listed chronologically). Except for those marked with *, they all also produce a Gomory-Hu tree. $T(n,m)$ denotes the time to compute s,t-max-flow in an undirected graph, which using Chen *et al.* [CKL+22] is $m^{1+o(1)}$. The three results marked with † were obtained concurrently and independently of each other. The two results marked with # are not independent. The first version of [AKL+22] gave an $\tilde{O}(n^{2.875})$ upper bound; an additional observation in the second version brought the bound down to $\tilde{O}(n^2)$. The latter improvement was discovered independently by Zhang [Zha21].

vertices and $\tilde{O}(m)$ edges, and runs for $\tilde{O}(m)$ time outside of these calls.

b) Guide Trees.: Next, to solve single-source terminal mincuts (Problem II.1), [AKL+22] introduces the notion of *guide trees*, defined below. A tree T is called a U-Steiner tree if it is a subgraph that spans all terminals in U.

Definition II.3 (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.

Here, a guide tree is said to k-respect a cut if the following property holds (which traces back to the work of Karger [Kar00] on the global min-cut problem).

Definition II.4 (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 that is not necessarily a subgraph of G. 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.

The overall plan now comprises two steps. First, obtain a small number of guide trees with the property that for every $t \in U\setminus\{s\}$ satisfying the promise in Problem II.1 some (s,t)-mincut k-respects at least one of the guide trees, for some constant k. Second, design an algorithm that reveals, for every $t \in U \setminus \{s\}$, the value of the (s,t)-mincut if it k-respects a fixed guide tree. Clearly, put together, these two steps solve Problem II.1, and therefore, by Lemma II.2, give an algorithm for the GHTree problem.

For the second step, [AKL+22] already gives an algorithm that runs in nearly max-flow time.

Theorem II.5 (Single-Source Mincuts given a Guide Tree, Theorem 10 in [AKL+22])**.** *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 that finds, for each vertex $t \neq s$ in T, a value $\tilde{\lambda}(t) \geq \lambda(s,t)$ such that: if T is k-respecting an (s,t)-mincut then $\tilde{\lambda}(t)=\lambda(s,t)$ with high probability. The algorithm runs in time $\tilde{O}(T_{MF}(n,m))$ where $T_{MF}(n,m)$ is the time of single-pair max-flow on n-node m-edge graphs.*

This leaves the question of finding a small set of guide trees such that some (s,t)-mincut k-respects at least one of these trees, for every $t \in U\setminus\{s\}$ that satisfies the promise of Problem II.1.

c) Constructing Guide Trees: A New Algorithm.: The natural approach (that is analogous to Karger's work where he finds 2-respecting *spanning* trees [Kar00] as opposed to *Steiner* trees) is to pack a large number of U-Steiner trees in the input graph and then sample some of them. Using standard techniques, one can show that there exists a set of $\lambda(U)/2$ edge-disjoint U-Steiner trees in the input graph. For any (s,t)-cut there will be at least one edge crossing it; thus, on average, only ≤ 2.2 edges cross the (s,t)-mincut

whose value is assumed to be at most $1.1\lambda(U)$. The goal now becomes to obtain such a packing of U-Steiner trees. Finding an optimal packing is NP-hard, but an approximately optimal packing of $\lambda(U)/(4+\epsilon)$ trees can be computed efficiently (as we discuss next). Sampling $O(\log n)$ trees from this packing produces the desired set of guide trees with the k-respecting property for $k = 4$ *with high probability*.

The main contribution of this paper is a faster algorithm for Steiner tree packing, defined as follows. A subgraph H of G is said to be a *U-Steiner subgraph* if all the terminals are connected in H.[4] A *U-Steiner-subgraph packing* $\mathcal{P}$ is a collection of U-Steiner subgraphs $H_1, \dots, H_k$, where each subgraph H_i is assigned a value $\text{val}(H_i) > 0$. 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

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

The main theorem of this paper computes a U-Steiner-subgraph packing in $m^{1+o(1)}$ time. This strengthens Lemma 22 of [AKL+22] that could only compute such a packing in $\tilde{O}(n^2)$ time in weighted graphs. They also obtained an $m^{1+o(1)}$-time algorithm for unweighted graphs. For multiple reasons that we will see in this discussion, unweighted graphs are substantially easier in this context. Fundamentally, with edge weights, our packing is required to be *implicit* (in a sense that will be formalized in the next theorem) because in some cases the output must contain $\Omega(m)$ subgraphs of size $\Omega(m)$ each. This is not the case in [AKL+22] since in an unweighted graph, each edge can only be part of one subgraph in the packing, which means that the packing is of size m and can be explicitly maintained. Indeed, in the weighted case, the algorithm in [AKL+22] runs in $\tilde{O}(m^2)$ time, which gives the $\tilde{O}(n^2)$ bound when applied on a $(1+\epsilon)$-cut-sparsifier (e.g., [BK15]).

Denote by $\text{pack}(U)$ the maximum value of a feasible U-Steiner-subgraph packing in G. Our main theorem is the following.

Theorem II.6 (Packing U-Steiner subgraphs)**.** *For every fixed $\epsilon \in (0, 1/2)$, there is a randomized algorithm that, given a graph $G = (V, E, w)$ with m edges and a terminal set $U \subseteq V$, returns whp an* implicit *U-Steiner-subgraph packing $\mathcal{P}$ of value $\text{val}(\mathcal{P}) \ge \text{pack}(U)/(2+\epsilon)$ in $m^{1+o(1)}$ time. More precisely, if $H_1, \dots, H_k$ are the implicitly computed U-Steiner-subgraphs, then the algorithm returns the list $\text{val}(H_1), \dots, \text{val}(H_k)$ and a data structure which returns, given a query $i \in [k]$, the U-Steiner-subgraph H_i in $m^{1+o(1)}$ time.*

As mentioned previously, by standard techniques, we know that $\text{pack}(U) \ge \lambda(U)/2$. Therefore, to obtain our set of guide trees from Theorem II.6, we sample $O(\log n)$ U-Steiner subgraphs independently at random from the packing $\mathcal{P}$ where subgraph H is sampled with probability $\text{val}(H)/\text{val}(\mathcal{P})$. For each sampled subgraph H, we take any spanning tree on the vertices of H to obtain the set of guide trees. Finally, to improve the bound from $m^{1+o(1)}$ to $n^{1+o(1)}$, we observe (as in [AKL+22]) that the packing may as well be done on a sparsifier.

Lemma II.7 (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 $t \in U \setminus \{s\}$, $\lambda(U) \le \lambda(s,t) \le 1.1\lambda(U)$, computes a 4-respecting set of $O(\log n)$ guide trees. The algorithm runs in $n^{1+o(1)}$ time and succeeds with high probability.*

Combining Lemma II.7 with the above discussion yields Theorem I.3 and the new GHTREE algorithm.

B. Previous Work: Steiner Tree Packing via MWU

So far we have established that the time to solve GHTREE is $m^{1+o(1)}$ plus the time to solve the Steiner tree packing problem (up to a constant approximation factor). Approximate packing problems are well-studied (see e.g. [PST95]). The common way to solve them that is employed also in this paper (also in [AKL+22]) is by a Multiplicative Weights Update (MWU) algorithm, e.g., based on [GK07], [Fle00], [AHK12].

Basically, the idea is the following: in each iteration, we add a U-Steiner subgraph to the packing that (approximately) minimizes the sum of *edge lengths*. Here, the length of an edge e is an exponential function of the current *congestion* on the edge, which in turn is defined as the fraction of the edge weight $w(e)$ that has already been used up in the partial packing (recall the feasibility condition $\sum_{H\in\mathcal{P}:e\in E(H)} \text{val}(H) \le w(e)$). The algorithm stops when the length of the minimum Steiner subgraph goes above a certain threshold and one can prove that the resulting packing is approximately optimal. Each iteration thus involves computing a minimum Steiner subgraph and updating the length of its edges, and the efficiency of the algorithm is determined by the *number of iterations* and the *time per iteration*.

To prove their $\tilde{O}(m^2)$ bound, Abboud *et al.* [AKL+22] use Mehlhorn's 2-approximation algorithm [Meh88] in each iteration to solve the minimum Steiner subgraph problem in $\tilde{O}(m)$ time, and bound the total number of iterations by $\tilde{O}(m)$. The latter is because the length of the minimum-length edge in the subgraph can only increase $\tilde{O}(1)$ times.

Their $m^{1+o(1)}$ time algorithm for *unweighted* graphs is much more involved and serves as the inspiration for this work. The underlying observation (as in many other works, including the recent $m^{1+o(1)}$ max-flow algorithm [CKL+22]) is that the input to the minimum Steiner subgraph problem does not change much from one iteration to another. Thus, a dynamic data structure could potentially handle all $\tilde{O}(m)$ iterations in $m^{o(1)}$ amortized time per length-update. Indeed, [AKL+22] successfully designs such an efficient dynamic algorithm for the minimum Steiner subgraph problem (discussed later). The

[4] The switch from Steiner trees to Steiner subgraphs is a technicality; for intuitive purposes, it is sufficient for the reader to consider a Steiner tree packing.

$m^{1+o(1)}$ bound now follows from the observation that the total number of length-updates in *unweighted* graphs is $\tilde{O}(m)$.

In weighted graphs, however, this approach fundamentally does not work because the number of edge-length updates is already $\Omega(n^2)$. This is for the same reason that the total number of edges in an explicit subgraph packing can be $\Omega(n^2)$, even when $m = O(n)$. This is sometimes referred to as the *flow decomposition barrier* (see e.g. [GR98], [Mad10], [BGS21]).

C. Part I: Steiner Tree Packing via Randomized MWU

The main conceptual contribution of this work is that we step away from standard MWU (that insists on an explicit packing and thus cannot go beyond $\Omega(n^2)$) and apply the recently introduced *randomized* MWU [BGS21] instead. At a very high level, Bernstein *et al.* [BGS21] employed this technique to pack s,t-paths in a graph (leading to an approximate max-flow algorithm) and we generalize it significantly in order to pack *Steiner trees*.

The main idea in randomized MWU is to only update the edge lengths of a randomly chosen subset of edges in every iteration, and show that the resulting random edge lengths stay sufficiently close to what the edge lengths would be in a deterministic (standard) MWU algorithm. Importantly, the probability that an edge is included in the subset is *not* uniform; it depends on the weight of the edge in such a way that the length of any edge is updated only $\tilde{O}(1)$ times in expectation. We give the precise details of this procedure in the full version but for now let us focus on the most significant technical obstacle towards realizing it.

a) Threshold Queries.: In standard MWU, one had to compute a minimum Steiner subgraph in each iteration (and then update the length of its edges). Now, we need to implicitly compute a Steiner subgraph and then explicitly access a randomly chosen subset of its edges (and update their lengths), *in time proportional to the number of accessed edges and not to the size of the entire subgraph*.

To be more concrete, each edge e now has a *steadiness* value $\sigma(e)$ that determines the probability of choosing the edge in the random sample in any iteration. The steadiness of an edge is unrelated to its length and remains fixed throughout the algorithm; it is only a function of its weight. In each iteration we consider the minimum *length* Steiner subgraph S and ask for the set of all edges $\sigma_{\leq j}(S)$ in S whose steadiness is below a randomly chosen threshold j. The goal is to compute this set in $|\sigma_{\leq j}(S)| \cdot m^{o(1)}$ time, in an amortized sense across all iterations.

The main technical result of this paper is a new dynamic algorithm for the minimum U-Steiner subgraph problem with the following features: (1) it supports decremental updates in the form of increase of edge lengths in $m^{o(1)}$ amortized time per update, (2) it is deterministic and therefore works in the adaptive adversary model (this is crucial for any form of MWU), and (3) it supports the aforementioned *threshold queries* that are critical for *randomized* MWU. We state the main theorem summarizing the properties of the data structure.

Theorem II.8 (Informal)**.** *There is a deterministic, decremental data-structure that, in $n^{o(1)}$ update time, implicitly maintains a $(2+\epsilon)$-approximate minimum length U-Steiner subgraph S at all times. At any time, given a steadiness index j, the data structure must return all edges of S whose steadiness is at most j, denoted by the set $\sigma_{\leq j}(S)$, in time proportional to the output size.*

b) From Minimum Steiner Subgraph to Shortest-Paths.: The starting point of our new dynamic algorithm is the algorithm of [AKL+22] that satisfies all the requirements except the ability to answer threshold queries. Let us overview it and explain why threshold queries are so challenging.

It is well-known that a 2-approximate minimum Steiner tree S of a graph G can be computed by taking the MST of a *helper graph* H in which each pair of terminals is connected by an edge of weight equal to their shortest path distance in G; the MST is expanded into a Steiner tree by taking the paths in G that correspond to the edges in the MST of H. Mehlhorn [Meh88] gives an $\tilde{O}(m)$ algorithm by observing that it suffices to find an MST of a different helper graph that can be computed via single-source shortest paths as opposed to (the more time-consuming) all-pairs shortest paths. Now suppose that G is undergoing a decremental sequence of edge updates; the natural approach is to (1) maintain the helper graph H via single-source shortest paths computations, (2) maintain an MST of H, and (3) maintain a Steiner subgraph S by expanding the MST of H. Fortunately, very efficient deterministic dynamic algorithms for MST [HdLT01] and single-source shortest-paths [BGS21] already exist, and with certain subtle modifications to Mehlhorn's algorithm (such as changing the definition of the helper graph) this approach can be turned into an efficient dynamic algorithm [AKL+22].

The main issue, however, is how to expand an MST of H into a Steiner subgraph of G (item (3) in the previous paragraph) in $m^{o(1)}$ *output-sensitive* time, i.e. only spend $x \cdot m^{o(1)}$ time if the output has size x. At a high level, this is difficult because each edge $e \in E(H)$ in $MST(H)$ corresponds to a shortest path $\pi(e) \subseteq E(G)$ in G that was computed by the dynamic single-source shortest-paths algorithm; a single edge $e \in E(G)$ in G could appear in a large number of such paths $\pi(e_1), \ldots, \pi(e_p)$ corresponding to different edges $e_1, \ldots, e_p \in MST(H)$ in the MST where $p = n^{\Omega(1)}$. Thus, even if the dynamic shortest paths algorithm can return the path in constant time per edge, a single edge in the Steiner subgraph we output may incur an overhead of $n^{\Omega(1)}$ time.

In [AKL+22], this issue is resolved by a careful expansion process that uses the ability of the shortest path algorithm to return a short prefix of the path. Adapting their approach to our setting where we need to support threshold queries, fails for a subtle but fatal reason: the Steiner subgraph that results in their expansion would depend on the (randomly chosen) steadiness threshold j. This breaks the analysis of the randomized MWU framework where we insist that there is an implicit but *fixed* Steiner subgraph in each iteration

that is *independent* of the threshold we choose. Given this fundamental bottleneck to using the previous ideas, we take a completely different approach in this paper.

Our idea for resolving the main issue of expanding the MST into a Steiner subgraph is to let the dynamic shortest paths algorithm do it for us. Given the MST, we know the set of pairs (s, y) that we are going to query for shortest path, where s is the fixed source and y is in a set of *targets* $Y \subseteq V(G)$. So we would like a decremental algorithm that can (implicitly) maintain a *single-source subset spanner*, i.e. a shortest path "tree" rooted at s and "only spanning" the targets Y, in the sense that its total size is proportional to the distances to the targets in Y, with the ability to answer threshold queries efficiently. The technical core of the paper is the implementation of such a single-source multiple-target algorithm, which can be of independent interest.

Theorem II.9 (Informal)**.** *There is a deterministic, decremental data structure that, in $n^{o(1)}$ update time, explicitly maintains the following:*

- *$(1+\epsilon)$-approximate distances estimates for all vertices in V from a fixed source vertex s, and a shortest paths tree realizing these estimates.*

In addition, it can maintain a target set $Y \subseteq V$, with $n^{o(1)}$ update time per insertion or deletion, with the following guarantees:

- *The data structure implicitly maintains a subgraph H spanning $Y \cup \{s\}$ whose total length is at most $(1+\epsilon)\sum_{v\in Y} \mathrm{dist}(s, v)$.*
- *Given a steadiness index j, the data structure returns $\sigma_{\le j}(H)$ in $n^{o(1)}$ output-sensitive time.*

This completes the overview of our new GHTREE algorithm, whose details appear in the full version.

D. Part II: Decremental Single-Source Multiple-Target Shortest Paths with Threshold-Queries

The more conceptual ideas overviewed above let us reduce the GHTREE problem to designing a (variant of a) dynamic shortest paths algorithm. What remains is an implementation of the dynamic data structure in Theorem II.9. Our starting point is the decremental algorithm for single-source shortest-paths *without threshold-queries* by Bernstein, Gutenberg, and Saranurak [BGS21]. This is a rather involved algorithm that builds on several breakthroughs in dynamic shortest path algorithms (e.g. [HKN14], [CS21a]). Fortunately, we do not need to change their algorithm; we only augment it with additional data structures that handle the threshold queries. Still, one has to design such an augmentation for each one of its components and with new ideas in each case.

a) High-level description and comparison with [BGS21].: The authors of [BGS21] have already achieved the desired augmentation for supporting threshold queries, in the special case of single-source *single-target* shortest path. That is, their dynamic algorithm has one target $y \in V$ and a threshold j, and returns the set of edges with steadiness below j in the shortest s, y-path (that is maintained by the base algorithm without threshold queries). Our setting is a strict generalization and comes with significantly more challenges: our dynamic algorithm maintains a subset of targets $Y \subseteq V$ and a threshold j and must handle all targets "simultaneously" (without returning each edge more than $n^{o(1)}$ times). For this reason, our algorithm uses essentially all ideas in the data structures of [BGS21] for threshold queries, but introduces several additional ideas on top. Let us begin with an overview of their algorithm, then discuss some of the challenges that arise in the multi-target setting; finally, we will present one new ingredient that we find particularly interesting: *decremental expander routing*.

The decremental shortest paths algorithm of [BGS21] is based on the classical *Even-Shiloach* trees [SE81], whose running time depends on the depth of the tree. Thus, it is important to maintain such trees on *emulators* with small diameter. These emulators "compress" the distances in the graph by encoding subsets of vertices into single new vertices called *cores*. The algorithm has three components that call each other recursively (on smaller distance scales):

1) For each core, we maintain an approximate shortest paths tree, up to some small diameter, rooted at the core. This is called APXTREE in the full version of the paper.
2) An efficient algorithm for connecting any pair of nodes *within* a core by a short path. This is called CORETREE in the full version of the paper.
3) A way to cover the graph with cores (and balls around them) without much overlap.

The last component is structural and does not need to be changed in our setting. The other two, however, must be augmented. In [BGS21] these algorithms know a specific source-target pair whose shortest path they might need to report. To support this, they employ several tricks, e.g. artificially including the edge with minimal steadiness on the path (without increasing its length by too much): this has the advantage that it is easy to determine if there are any edges on the path with steadiness below a given threshold j by only monitoring the edge with minimal steadiness. Our setting is more difficult because we have multiple targets and the tricks employed by [BGS21] no longer work. Next, we describe a more conceptually interesting challenge related to finding short paths in expanders that arises when augmenting the second item above.

b) Decremental Expander Routing.: At the heart of [BGS21], and perhaps the single most important idea for making the above approach work, is to make sure that each core is approximately an *expander graph*, and that it remains so throughout the sequence of (decremental) updates using a certain *expander pruning* subroutine. Roughly speaking, expanders are important for threshold queries and more generally for data structures that can report paths in output-sensitive time, because we can find a short path between any pair of nodes in the expander. The algorithm of [BGS21] uses such a *short path oracle for decremental expanders* due to Chuzhoy and Saranurak [CS21b]. In our new context, however, we would like to exploit expanders even further. We will need

to specify a *set of pairs* and get a short path *between every pair* in a way that no edge is used too many times in the set of paths. That is, we need to be able to find an *embedding* of any (constant degree) graph inside a decremental expander.

Definition II.10 (Embedding)**.** Let W, D be two graphs with $V(D) \subseteq V(W)$. A set $\mathcal{P}$ of paths in W is called an *embedding* of D into W if, for every edge $e = (u, v) \in E(D)$, there is a path $\text{path}(u, v) \in \mathcal{P}$ such that $\text{path}(u, v)$ is a u–v path in W. The *dilation* of the embedding $\mathcal{P}$ is l if the length of every path in $\mathcal{P}$ is at most l, and the *congestion* of the embedding is η if every edge of W participates in at most η paths in $\mathcal{P}$.

Indeed, we generalize the proof of [CS21b] from the single-pair setting to the multi-pair setting. Our new *decremental expander routing* is of independent interest and we believe it is likely to have further applications in dynamic graph algorithms. Let us now state it more formally and then discuss how it fits in the larger context of expander routing subroutines.

Recall the following notation on expander graphs. For any graph $G = (V, E)$ and a vertex set $S \subseteq V$, the volume of S is $\text{vol}_G(S) = \sum_{u \in S} \deg_G(u)$ the sum of degree of vertices in S. We say that G is a *φ-expander* if, for any vertex set $S \subset V$, we have that $\frac{|E_G(S, V \setminus S)|}{\min\{\text{vol}_G(S), \text{vol}_G(V \setminus S)\}} \geq \varphi$. In words, for any cut in G, at least a φ-fraction of edges incident to the smaller side of the cut are crossing the cut.

Roughly speaking, our new subroutine does the following. Given an expander W that undergoes edge deletions, we maintain a subgraph X of W and a subgraph X' of X, such that: (1) even the smaller subgraph X' is large (contains half the nodes of W), and (2) we can efficiently route any given demand graph on X' inside X. Specifically, let $D = (X', E_D)$ be a graph with bounded maximum degree where each edge of D represents a demand pair. The algorithm can return an embedding of D into $W[X]$ with congestion and dilation at most $n^{o(1)}$ in $|E_D| n^{o(1)}$ time.

The formal statement is described below and the proof is in the full version of the paper.

Lemma II.11 (Decremental Expander Routing)**.** *Let $C_{\text{EMBED}} > 0$ be a large enough constant, and let constant $C_{\text{ROUTE}} > 0$ depend on C_{EMBED}. Define $\gamma_{\text{EMBED}} = 2^{C_{\text{EMBED}} \log^{3/4} n}$ and $h_{\text{ROUTE}} = 2^{C_{\text{ROUTE}} \log^{7/8} n}$. There is an algorithm* PRUNEROUTER(W) *with the following guarantee.*

- ***(input):** Given an unweighted multi-graph $W = (V, E)$ with maximum degree $O(\log n)$ that is initially a γ_{EMBED}-expander which undergoes at most $|V(W)|/h_{\text{ROUTE}}$ edge deletions,*
- ***(maintain):** the algorithm maintains two decremental sets $X \subseteq V$ and $X' \subseteq X$ using $O(m h_{\text{ROUTE}})$ total update time such that*
  - *$W[X]$ is a $\Omega(\gamma_{\text{EMBED}})$-expander at any point of time,*
  - *$\text{vol}_W(V \setminus X) \leq O(i/\gamma_{\text{EMBED}})$ after i updates, and*
  - *$|X'| \geq |V|/2$.*
- ***(query):** At any time, given a demand graph $D = (X', E_D)$ with maximum degree $O(1)$, the algorithm returns an embedding of D into $W[X]$ with congestion and dilation at most h_{ROUTE} in $O(|E_D| h_{\text{ROUTE}})$ time.*

Efficient algorithms for routing in expanders lie at the core of several recent breakthroughs in graph algorithms. Previous work has studied the following three versions.

1) Static routing: Given an expander W and given a demand graph D, route D inside W in near-linear time. This is the weakest notion and already has celebrated applications; namely, it is a key subroutine in the almost-linear time max-flow algorithm [CKL+22].
2) Static routing queries: Preprocess an expander W such that given any demand graph D we can route it in W in time that is near-linear in the size of D. This clearly strengthens static routing and is the non-dynamic version of our result with $X' = X = V(W)$. It has been used extensively in distributed algorithms [GKS17], [CS20], for example, for listing subgraphs [CPSZ21], [CHCGL21], [CHLV22].
3) Decremental *single-pair* routing: Maintain an expander W under edge deletions in near-linear time, and given a demand *pair* s, t return an s, t-path in near-constant time. This version strengthens static routing as well via simple greedy process. The aforementioned result by [CS21b] has accomplished this version and quickly found several applications to shortest paths [BGS20], [Chu21], [BGS21], [CKL+22].

The latter two versions strengthen static routing in two different ways. Our algorithm achieves the best of both worlds: it allows for arbitrary demand graph queries and it also supports decremental updates to the expander. We are convinced that it will have more applications.

E. Conclusion: Our Main Contributions

We conclude this overview by summarizing the three main contributions of this work:

- The introduction of the *randomized* MWU framework into the GHTREE problem in order to compute a near-optimal packing of Steiner trees in a graph. Previously it was only used to compute a packing of paths (in the context of single-pair max-flow) [BGS21].
- Executing numerous technical modifications to previous work in order to apply our approach. In particular, (1) when replacing MWU with randomized MWU inside the Steiner tree packing algorithm of [AKL+22], the minimum Steiner subgraph gadget (that relies on a reduction to single-source multiple-target shortest-paths) has to be augmented to support *threshold-queries*, and (2) when adapting the single-pair shortest-paths algorithm of [BGS21] that supports threshold-queries into the single-source multiple-target setting, the entire algorithm has to be revisited and generalized.
- In order to execute the latter, we have designed a new *decremental expander routing* algorithm that is likely to find more applications in dynamic graph algorithms.

III. Future Work

Our work brings the line of research started by the pioneering work of Gomory and Hu to its logical conclusion by showing that the all-pairs minimum cuts or maximum flow problems (APMC and APMF) are equivalent, up to subpolynomial terms in the running time, to their single-pair versions. Nevertheless, several interesting directions of research remain open.

First, there is the question of deterministic algorithms. Almost all of the recent literature that makes progress on APMF relies heavily on the use of randomization. The best deterministic algorithm for this problem, or indeed for simpler problems such as partitioning the vertices of a graph into k-connected components, remains the 60-year old algorithm of Gomory and Hu. Can we design fast deterministic algorithms for APMF?

Second is the question of generalization. There are several directions one can hope to generalize undirected graphs to. Perhaps the most natural is to directed graphs. Here, the nonexistence of Gomory-Hu trees is well-known [Ben95], but there are other data structures (e.g., [CH91]) that capture (symmetric) all-pairs MC. How fast can we compute such a data structure? One can ask similar questions for other generalizations of undirected graph connectivity – e.g., vertex connectivity, element connectivity, hypergraph connectivity, etc. An $(m^{1+o(1)})$-time construction of the Gomory-Hu tree for element connectivity is very interesting as it would also imply a near optimal algorithm for computing all-pair vertex connectivity in $m^{2+o(1)}$ time which is tight with a conditional lower bound [HLSW22].

Finally, there is the question of aesthetics. The reductions of Gomory and Hu and Abboud, Krauthgamer, and Trabelsi from the all-pairs problem to n single-pair computations or $O(\log n)$ single-source computations are short, combinatorially elegant proofs. In contrast, the reduction to $n^{o(1)}$ single-pair computations given in the current paper is highly technical and requires several heavy hammers and over 50 pages to describe. An intriguing question is whether we can replicate the elegance of these prior works in an almost-linear-time algorithm for APMF.

References

[ACZ98] Srinivasa Rao Arikati, Shiva Chaudhuri, and Christos D. Zaroliagis. All-pairs min-cut in sparse networks. *J. Algorithms*, 29(1):82–110, 1998.

[AGI+19] Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster Algorithms for All-Pairs Bounded Min-Cuts. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 7:1–7:15, 2019.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[AIS+16] Takuya Akiba, Yoichi Iwata, Yosuke Sameshima, Naoto Mizuno, and Yosuke Yano. Cut tree construction from massive graphs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 775–780. IEEE, 2016.

[AKL+22] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2022.

[AKT20] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 105–118, 2020.

[AKT21a] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. APMF $<$ APSP? Gomory-Hu tree for unweighted graphs in almost-quadratic time. *Accepted to FOCS'21*, 2021. arXiv:2106.02981.

[AKT21b] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. *Theory of Computing*, 17(5):1–27, 2021.

[AKT21c] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory–hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1725–1737, 2021.

[AKT22] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster Gomory-Hu trees. *Accepted to SODA'22*, 2022. arXiv:2110.15891.

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[AV20] Nima Anari and Vijay V Vazirani. Planar graph perfect matching is in NC. *Journal of the ACM*, 67(4):1–34, 2020.

[AVY15] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. of 47th STOC*, pages 41–50, 2015.

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 434–443, 2014.

[Ben95] András A. Benczúr. Counterexamples for directed and node capacitated cut-trees. *SIAM J. Comput.*, 24(3):505–510, 1995.

[BENW16] Glencora Borradaile, David Eppstein, Amir Nayyeri, and Christian Wulff-Nilsen. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *32nd International Symposium on Computational Geometry*, volume 51 of *SoCG '16*, pages 22:1–22:16, 2016.

[BGS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134. IEEE, 2020.

[BGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1000–1008. IEEE, 2021.

[BHKP07] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In *39th Annual ACM Symposium on Theory of Computing*, STOC'07, pages 605–614, 2007.

[BK15] András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015.

[BSW15] Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3), 2015.

[CCPS97] W.J. Cook, W.H. Cunningham, W.R Pulleybank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1997.

[CH91] Chung-Kuan Cheng and T. C. Hu. Ancestor tree for arbitrary multi-terminal cut functions. *Ann. Oper. Res.*, 33(3):199–213, 1991.

[CHCGL21] Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. Tight distributed listing of cliques. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2878–2891. SIAM, 2021.

[CHLV22] Keren Censor-Hillel, Dean Leitersdorf, and David Vulakh. Deterministic near-optimal distributed listing of cliques. In

Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing, pages 271–280, 2022.

[Chu21] Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 626–639, 2021.

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022.

[CPSZ21] Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *Journal of the ACM (JACM)*, 68(3):1–36, 2021.

[CS20] Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 377–388. IEEE, 2020.

[CS21a] Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2478–2496. SIAM, 2021.

[CS21b] Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2478–2496. SIAM, 2021.

[Dah16] Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[Fle00] Lisa K Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

[GH61] Ralph E. Gomory and Te C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9:551–570, 1961.

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[GKS17] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed mst and routing in almost mixing time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2017.

[GR98] Andrew V Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998.

[GT01] Andrew V. Goldberg and Kostas Tsioutsiouliklis. Cut tree algorithms: an experimental study. *Journal of Algorithms*, 38(1):51–83, 2001.

[Gus90] Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

[HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 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*, pages 146–155. IEEE, 2014.

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. of the 47th STOC*, pages 21–30. ACM, 2015.

[HL07] Refael Hassin and Asaf Levin. Flow trees for vertex-capacitated networks. *Discrete Appl. Math.*, 155(4):572–578, 2007.

[HLSW22] Zhiyi Huang, Yaowei Long, Thatchaphol Saranurak, and Benyu Wang. Tight conditional lower bounds for vertex connectivity problems. *arXiv preprint arXiv:2212.00359*, 2022.

[Hu74] Te C. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974.

[Jel63] F. Jelinek. On the maximum number of different entries in the terminal capacity matrix of oriented communication nets. *IEEE Transactions on Circuit Theory*, 10(2):307–308, 1963.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000.

[KL15] David R. Karger and Matthew S. Levine. Fast augmenting paths by random sampling from residual graphs. *SIAM J. Comput.*, 44(2):320–339, 2015.

[Kol22a] Vladimir Kolmogorov. A computational study of gomory-hu tree construction algorithms. *arXiv preprint arXiv:2204.10169 v3*, 2022.

[Kol22b] Vladimir Kolmogorov. Orderedcuts: A new approach for computing gomory-hu tree. *arXiv preprint arXiv:2208.02000*, 2022.

[KT18] Robert Krauthgamer and Ohad Trabelsi. Conditional lower bounds for all-pairs max-flow. *ACM Trans. Algorithms*, 14(4):42:1–42:15, 2018.

[LP21] Jason Li and Debmalya Panigrahi. Approximate Gomory-Hu tree is faster than $n-1$ max-flows. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1738–1748. ACM, 2021.

[LPS21] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *Accepted to FOCS'21*, 2021. arXiv:2106.02233.

[Mad10] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 121–130, 2010.

[May62] W. Mayeda. On oriented communication nets. *IRE Transactions on Circuit Theory*, 9(3):261–267, 1962.

[Meh88] Kurt Mehlhorn. A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[Pan16] Debmalya Panigrahi. Gomory-Hu trees. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 858–861. Springer New York, 2016.

[PR82] Manfred W Padberg and M Ram Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[PST95] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20(2):257–301, 1995.

[Sch03] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

[SE81] Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

[vdBLL+21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances. In *53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 859–869. ACM, 2021.

[WL93] Zhenyu Wu and Richard Leahy. 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):1101–1113, 1993.

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

[Zha22] Tianyi Zhang. Faster Cut-Equivalent Trees in Simple Graphs. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 109:1–109:18, Dagstuhl, Germany, 2022.