APMF $<$ APSP? Gomory-Hu Tree for Unweighted Graphs in Almost-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

Ohad Trabelsi§
University of Michigan
Ann Arbor, USA
ohadt@umich.edu

***Abstract*—We design an $n^{2+o(1)}$-time algorithm that constructs a cut-equivalent (Gomory-Hu) tree of a simple graph on n nodes. This bound is almost-optimal in terms of n, and it improves on the recent $\tilde{O}(n^{2.5})$ bound by the authors (STOC 2021), which was the first to break the cubic barrier. Consequently, the All-Pairs Maximum-Flow (APMF) problem has time complexity $n^{2+o(1)}$, and for the first time in history, this problem can be solved faster than All-Pairs Shortest Paths (APSP). We further observe that an almost-linear time algorithm (in terms of the number of edges m) is not possible without first obtaining a subcubic algorithm for multigraphs.**

Finally, we derandomize our algorithm, obtaining the first subcubic deterministic algorithm for Gomory-Hu Tree in simple graphs, showing that randomness is not necessary for beating the $n-1$ times max-flow bound from 1961. The upper bound is $\tilde{O}(n^{2\frac{2}{3}})$ and it would improve to $n^{2+o(1)}$ if there is a deterministic single-pair maximum-flow algorithm that is almost-linear. The key novelty is in using a "dynamic pivot" technique instead of the randomized pivot selection that was central in recent works.

***Keywords*-Gomory-Hu; all-pairs max-flow; cut-equivalent tree; simple graphs**

I. Introduction

Connectivities (minimum-cut or maximum-flow) and distances (or shortest path) are perhaps the two most fundamental measures in graphs. Their computational complexity is a central object of study in algorithms and discrete optimization, and both have been extensively investigated in almost any setting of interest. Researchers often ponder the question: at a high-level, which of the two is harder?

The focus of this paper is on simple graphs (undirected, unweighted, no parallel edges or self-loops), denoting the input graph by G and its number of nodes by $n = |V(G)|$. For a single-pair $s, t \in V(G)$ both measures can be computed in $\tilde{O}(n^2)$ time, albeit using very different algorithms. For shortest path, Dijkstra's algorithm [15] from 1956 is sufficient, while either continuous optimization [38] or randomized contractions [24] are needed for maximum-flow. For the more demanding task of computing these measures for *all-pairs* in a given graph, the complexities appear to differ. Seidel's algorithm [36] solves all-pairs shortest paths in time $n^{\omega+o(1)}$, where $\omega \leq 2.37286$ is the fast matrix multiplication exponent [6]; whether faster algorithms are possible is one of the most well-known open questions in Algorithms. The key breakthrough for all-pairs maximum-flow came in 1961 with the fundamental discovery of Gomory and Hu [21] that any graph can be turned into a tree while preserving the minimum cuts for all pairs. Once the tree is obtained, all pairwise connectivities can be extracted in $\tilde{O}(n^2)$ time.

Theorem I.1 (Gomory and Hu [21])**.** *Every undirected graph G (even with edge weights) has an edge-weighted tree T on the same set of vertices $V(G)$ such that:*

- *for all pairs $s, t \in V(G)$ the minimum (s,t)-cut in T is also a minimum (s,t)-cut in G, and their values are the same.*

Such a tree is called a cut-equivalent tree*, aka* Gomory-Hu Tree*. Moreover, the tree can be constructed in the time of $n-1$ calls to a* Max-Flow *algorithm.*[1]

Such a strong structural result is not possible for shortest paths (e.g., because any compression to $\tilde{O}(n)$ bits must incur large error [37], [1]). Still, up until now, it has not lead to an algorithm solving the all-pairs maximum-flow problem (denoted All-Pairs Max-Flow) faster than all-pairs shortest path (All-Pairs Shortest-Paths). The original algorithm of Gomory and Hu for getting such a tree has time complexity $\Omega(n^3)$, and only very recently a subcubic $\tilde{O}(n^{2.5})$-time algorithm was found [5]. The immediate open question is whether the bound can be pushed all the way down to n^2, or perhaps the tools of fine-grained complexity could come in the way and establish a conditional lower bound.

* A full version of this paper can be found at arXiv:2106.02981

† Work partially supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.

‡ Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, and a Minerva Foundation grant.

§ Work partially done at Weizmann Institute of Science and partially supported by the NSF Grant CCF-1815316.

[1]The notation Max-Flow refers to the maximum (s,t)-flow problem, which clearly has the same value as minimum (s,t)-cut. In fact, we often need algorithms that find an optimal cut (not only its value), which is clearly different (and usually harder) than Global-Min-Cut.

Open Question I.2. *Can one construct a Gomory-Hu Tree of a simple graph G and solve all-pairs maximum-flow in $\tilde{O}(n^2)$-time?*

Positive answers were obtained recently [3], [28], but only for $(1+\varepsilon)$-approximate cuts. On the negative side, the Strong Exponential Time Hypothesis (SETH) gives an $n^{3-o(1)}$ lower bound for the harder setting of *directed graphs* [26] (see also [2] for a higher lower bound), but probably not for undirected graphs [4]. Recent work has identified a class of problems that are conjectured to have $n^{2.5}$ complexity [30] (including all-pairs shortest paths in directed graphs [12]); could our problem be one of them?

Main Result: For unweighted simple graphs, our main result resolves (up to $n^{o(1)}$ factors) the complexity of all-pairs maximum-flow and of the Gomory-Hu Tree problem in dense graphs, where $\Omega(n^2)$ is a lower bound due to the input size (and for the former problem also output size).

Theorem I.3. *There is a randomized algorithm, with success probability $1 - 1/\operatorname{poly}(n)$, that constructs a Gomory-Hu Tree of a simple graph G and solves All-Pairs Max-Flow in time $n^{2+o(1)}$.*

Note added in proof: The same theorem was obtained independently by Li, Panigrahi, and Saranurak in a paper published in the same conference [29]. The two algorithms use similar ingredients, but they are invoked and analyzed in a different way. Perhaps the most significant technical difference is that one of the main procedures is implemented in a simpler way in their paper compared to ours; we have added a remark (footnote 6) to Section I-B pointing out how to use their idea to simplify our algorithm as well. Moreover, a third independent paper with a slightly higher time bound of $\tilde{O}(n^{2+1/8})$ was announced by Zhang [40]; interestingly, the technique in this algorithm is further away from ours and assuming a near-linear time Max-Flow algorithm the running time becomes $\tilde{O}(n^2)$ which improves on both this paper and the one by Li *et al.* [29]. Finally, we remark that the new results in the rest of this section are exclusive to our paper and do not appear in the other papers.

We find Theorem I.3 surprising for multiple reasons. First, it shows that all n^2 answers can be computed in the same time, up to lower-order factors, as it takes to compute a single-pair maximum-flow. Second, for the first time in history, the time complexity of All-Pairs Max-Flow goes below that of All-Pairs Shortest-Paths. This might be counter-intuitive because of the apparent unruliness of flows compared to paths, e.g., the $n^{2+o(1)}$-time single-source algorithm was much more difficult to obtain. Perhaps the only indication for the plausibility of this outcome was given in our previous paper, where it was shown that truly subcubic time is possible for All-Pairs Max-Flow by only using combinatorial methods, whereas it is conjectured to be impossible for All-Pairs Shortest-Paths [5]. Third, the algorithm not only solves All-Pairs Max-Flow but also produces a Gomory-Hu Tree that, among other things, is a space-optimal data structure for answering minimum cut invocations in $\tilde{O}(1)$ time. While the Gomory-Hu algorithm reduces the problem to $n-1$ Max-Flow computations, our new algorithm can be viewed as a reduction to only $\tilde{O}(1)$ computations on graphs with $n^{2+o(1)}$ edges (but possibly more than n nodes).[2]

Gomory-Hu Tree in $\tilde{O}(m)$ Time?: Looking ahead, after achieving the optimal exponent for the number of nodes n, the next outstanding question is a bound in terms of the number of edges $m = |E(G)|$: Is there an almost-linear time algorithm? Such a bound is known for planar graphs [11], surface-embedded graphs [10], and bounded-treewidth graphs [7], [3]. It is also known in simple graphs if the algorithm is allowed to make nondeterministic guesses [4].

Open Question I.4. *Can one construct a Gomory-Hu Tree of a simple graph G in $m^{1+o(1)}$-time?*

One issue when studying this question is that even single-pair Max-Flow is not known to be in almost-linear time when $m = O(n^{1.5-\varepsilon})$. If our goal is to understand the complexity of constructing a Gomory-Hu Tree, it is natural (and common) to assume the following plausible hypothesis.

Hypothesis I.5. *Single-Pair Max-Flow on m-edge weighted graphs can be solved in time $m^{1+o(1)}$.*

This hypothesis does resolve Open Question I.4 positively, even for general (weighted) graphs, if we are only interested in a $(1+\varepsilon)$-approximate Gomory-Hu Tree [28], but not in the exact case. In the regime of sparse simple graphs, $m = O(n)$, even under this hypothesis, the fastest algorithm for Gomory-Hu Tree [4] has complexity $\tilde{O}(n^{1.5})$, leaving a gap of about $\sqrt{n}$. For general m, the state of the art would be $(\min(n^2, m^{1.5}))^{1+o(1)}$, due to Theorem I.3 and [4].

Our next observation is a reduction showing that improving the $n^{1.5}$-bound for simple graphs requires a breakthrough subcubic algorithm for Gomory-Hu Tree in more general settings. However, the recent advances in algorithms for Gomory-Hu Tree have been insufficient for breaking the cubic barrier even in unweighted non-simple graphs (*multigraphs*), which seem to be the main step towards weighted graphs; indeed, the known bound is $\tilde{O}(mn)$ [9], [24], which is $\tilde{O}(n^3)$ in the worst-case.

Hypothesis I.6. *No algorithm can construct a Gomory-Hu Tree of an unweighted multigraph with n nodes and $O(n^2)$ possibly parallel edges in time $O(n^{3-\varepsilon})$ for a fixed $\varepsilon > 0$.*

[2]However, viewing it this way is not sufficient for getting an almost-quadratic algorithm because the current Max-Flow algorithms are not linear-time for all edge densities. Our algorithm actually uses a subroutine due to [9] that cannot be reduced to Max-Flow computations.

Theorem I.7. *Assuming Hypothesis I.6, no algorithm can construct a Gomory-Hu Tree of a simple graph on n nodes and $m = O(n)$ edges in time $n^{1.5-\varepsilon}$ for a fixed $\varepsilon > 0$.*

Derandomization: Taking a step back, we ask whether the developments in recent years [9], [4], [3], [5] can give faster *deterministic* algorithms for Gomory-Hu Tree. In fact, no deterministic algorithm (in any density regime) faster than Gomory-Hu's deterministic $n \cdot MF(n, m) = \Omega(nm)$ algorithm is known. In fact, to our knowledge, the best upper bound in terms of n is $O(n^{3\frac{2}{3}})$ using Goldberg and Rao's deterministic Max-Flow algorithm [20]. The main difficulty is due to the fact that all these new algorithms solve the *single-source* minimum-cut problem in some clever way, and then use a *randomized pivot* technique, in which a uniformly random source serves as a pivot in a recursive process of logarithmic depth. The algorithm in Theorem I.3 (and in [5]) uses randomization also in other steps, such as expander decompositions and randomized hitting sets, but these can already be derandomized with existing methods.

A central contribution of our work is a new *dynamic pivot* technique that can (sometimes) derandomize the randomized-pivot technique at no extra cost.The main result in this context is the first subcubic algorithm for Gomory-Hu Tree.

Theorem I.8. *There is a deterministic algorithm that constructs a Gomory-Hu Tree of a simple graph and solves All-Pairs Max-Flow in time $\tilde{O}(n^{2\frac{2}{3}})$. The time bound improves to $n^{2+o(1)}$ assuming that single-pair Max-Flow can be computed in deterministic time $m^{1+o(1)}$ in m-edge weighted graphs.*

A. Prior Work

The literature on cut-equivalent (Gomory-Hu) trees is quite vast. They have found many important applications in several fields (see [5, Section 1.4]), and have been generalized in several ways. See the full version for more details and pointers.

Previous Algorithms for Gomory-Hu Tree: Over the years, the time complexity of constructing a Gomory-Hu Tree has decreased several times due to improvements in Max-Flow algorithms, but there have also been conceptually new algorithms. Gusfield [22] presented a modification of the Gomory–Hu algorithm in which all the $n - 1$ calls to Max-Flow are made on the original graph G (instead of on contracted graphs). Bhalgat, Hariharan, Kavitha, and Panigrahi [9] designed an $\tilde{O}(mn)$-time algorithm utilizing a tree packing approach [18], [16] that has also been used in other algorithms for cut-equivalent trees [14], [23], [4], [5]. In particular, they designed an $O(mk)$-time algorithm for constructing a k-partial Gomory-Hu Tree, which preserves the minimum cuts if their size is up to k (see [34] and the full version [8]). A simple high-degree/low-degree strategy is helpful in sparse graphs: only $\sqrt{m}$ nodes can have degree (and therefore outgoing flow) above $\sqrt{m}$, thus a $\sqrt{m}$-partial tree plus $\sqrt{m}$ Max-Flow queries are sufficient, which takes $O(m^{3/2})$ time if Max-Flow is solved in linear time. Using the current Max-Flow algorithms, this strategy results in the time bound $\tilde{O}(\min\{m^{3/2}n^{1/6}, \max\{mn^{3/4}, m^{3/2}\}\})$ [4]. Combined with [5], the state of the art before this work was $\tilde{O}(\min\{m^{3/2}n^{1/6}, mn^{3/4}, n^{3/2}m^{1/2}\})$ and now it is $\min(n^{2+o(1)}, \tilde{O}(m^{3/2}n^{1/6}), \tilde{O}(mn^{3/4}))$. Additionally, two recent algorithms accelerate the Gomory–Hu algorithm by making multiple steps at once to achieve $\tilde{O}(m)$ time, one requires nondeterminism [4] and the other requires a (currently non-existent) fast minimum-cut data structure [3].

In weighted graphs, the bound for Gomory-Hu Tree is still n times Max-Flow and therefore cubic using the very recent $\tilde{O}(m + n^{1.5})$ algorithm for Max-Flow [38]. If the graph is sparse enough, one can use the recent $\tilde{O}(m^{3/2-1/328})$ time algorithm [19]. If additionally the largest weight U is small, a series of algorithms exist [32], [31], [25] that run in time $\tilde{O}(\min\{m^{10/7}U^{1/7}, m^{11/8}U^{1/4}, m^{4/3}U^{1/3}\})$ and might give a better time bound. The main open question left by this work is whether the ideas behind the drastic speed ups for unweighted graphs will lead to faster algorithms for weighted graphs as well.

Due to space constraints, this version omits further discussion of prior work. See the full version for details and pointers about (i) practice-oriented algorithms; (ii) approximation algorithms for Gomory-Hu Tree; and (iii) the use of expander decomposition in recent algorithms for problems other than Gomory-Hu Tree.

B. Technical Overview

In this section we distill and highlight the novel technical ideas of this work, instead of providing an overview of the actual algorithms. The overview below is a considerable over-simplification, because the algorithms in both the work we improve upon [5] and (to a lesser extent) this paper combine intricately many ingredients, and they cannot be truly explained without many preliminaries that may only be known to experts. Nevertheless, the standard/advanced terminology and algorithms mentioned below without explanation can all be found in the Preliminaries (Section II, and Section 5.1 in the full version).

The plan is to first motivate and explain the three new ideas that are the key to the new results. We hope that they can be appreciated on their own even if it is not possible to see exactly how they fit into the actual algorithm. Afterwards, we will discuss some of the complications that arise when using these ideas inside the algorithm.

1) The Simplest Hard Tree: Breaking the $n^{2.5}$ Barrier using Expander-Decomposition with Demands: The simplest hard case for constructing a Gomory-Hu Tree appears to be the following.[3] Given a graph G distinguish between the case where its only Gomory-Hu Tree is a star from the case where it is a c_ℓ-c_r tree; i.e. two stars connected by an edge between the two centers c_ℓ and c_r. In the first case, the graph has the property that the minimum u, v-cut for all nodes $u, v \in V$ is one of the two trivial cuts $\{u\}$ or $\{v\}$, and in the second case there is a single non-trivial minimum cut (C_ℓ, C_r), the cut between c_ℓ and c_r, while almost all others are trivial. This is a hard case even if we are given the node c_r in advance; it is helpful to think of it as the pivot or source from which we want to compute single-source cuts. The difficulty is in finding c_ℓ (if it exists at all). To make it more concrete (and still hard) suppose that both stars have $\Omega(n)$ nodes and the minimum c_ℓ, c_r-cut has value $w = \Theta(n)$ which in turn implies that both c_ℓ, c_r must have degree $\Omega(n)$. See Figure 1.

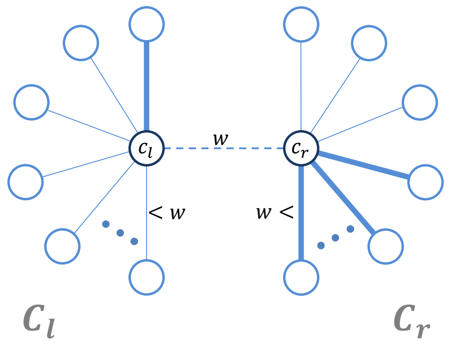

Figure 1: The cut-equivalent tree T of a hard case, where c_r is given and the goal is to identify c_ℓ, the only node with a non-trivial minimum cut to c_r. The weight of the cut (C_ℓ, C_r) is $\lambda_{c_l,c_r} = w$, illustrated by the dashed edge at the center; tree edges of weight $> w$ are thick, and tree edges of weight $< w$ are thin. The minimum cut between any pair of leaves u, v is trivial unless u is incident to a bold edge on the left and v is incident to a bold edge on the right.

It is instructive to observe that random sampling does not help, even if the cut we are searching for (the minimum c_ℓ, c_r-cut) is balanced. Even if we sample a node ℓ from c_ℓ's star and a node r from c_r's star (or c_r itself), their minimum ℓ, r-cut will not reveal the cut we are searching for; it could even be simply $\{\ell\}$ or $\{r\}$.[4] It seems that $\Omega(n)$ calls to a Max-Flow algorithm are required with this strategy. In fact, random sampling is one of the ways that help one see that c_ℓ-c_r is the "hardest tree": if the tree had a long path it would have been possible to shorten it with a randomized hitting set. Another method, the powerful ISOLATING-CUTS procedure [27], [5], receives a subset of nodes $C \subseteq V$ as input, and returns the minimum *isolating* cut for each node $v \in C$; meaning the minimum cut that separates v from all other nodes in $C \setminus \{v\}$. This method lets one resolve trees with many small subtrees, but it too fails against the c_ℓ-c_r tree where there is only one large subtree (see [5, Section 3] for more details). One important observation, that is the first step towards a solution is that c_ℓ is the highest degree node among the nodes in the left star C_ℓ;[5] but the right star could have many nodes with higher or lower degree than it.

Perhaps the main idea in the subcubic algorithm of [5] was to use an expander decomposition of G in order to locate the node c_ℓ using fewer or cheaper calls to a Max-Flow algorithm. The graph is decomposed into ϕ-expanders $H_1, \ldots, H_h$ with expansion parameter $\phi = 1/\sqrt{n}$ and the number of outer-edges (that leave an expander) is $\tilde{O}(|E|\phi) = \tilde{O}(n^{1.5})$. Since the cut we are searching for has only $O(n)$ edges, no expander can contain $\Omega(\sqrt{n})$ high degree nodes, say at least $n/10$, from each side of this cut, as that would violate the ϕ-expansion requirement, since it would induce inside the expander a cut of conductance $\frac{O(n)}{\Omega(\sqrt{n}\cdot n)} < \phi$ (see Section II-C for relevant definitions). Now, a key observation that crucially relies on the fact that G is a simple graph, is that only $\tilde{O}(\sqrt{n})$ nodes of high degree, can be in small expanders H_i, i.e., of size at most $n/100$; this is because such a high degree node in a small expander must have at least $n/10 - n/100 \geq \Omega(n)$ incident outer-edge, and there can only be $\tilde{O}(n^{1.5})$ outer-edges in the decomposition. On the other hand, there can only be $O(1)$ large (i.e., not small) expanders since there are only n nodes in the graph. Consequently, it is enough to search for c_ℓ inside the $O(1)$ large expanders, using the clever methods that follow, or among those $\tilde{O}(\sqrt{n})$ high-degree nodes using direct calls to a Max-Flow algorithm. To search in large expanders two methods are used: one is useful when there are few nodes from the left star in the expander, and one is useful when there are few nodes from the right star. (Recall that no expander can contain $\Omega(\sqrt{n})$ nodes from each star.)

In brief, the first procedure uses the fact that the expander contains c_ℓ plus only few nodes from C_ℓ in order to find a set of nodes S that isolates c_ℓ, in the sense that $C_\ell \cap S = \{c_\ell\}$, and then uses the ISOLATING-CUTS procedure. And the second procedure, which will be elaborated on in the next subsection, uses the fact that c_ℓ is the highest degree node in C_ℓ to conclude that it must be among the $O(\sqrt{n})$ highest-degree nodes in an expander that has $< \sqrt{n}$ nodes from C_r. The upshot is that $\tilde{O}(\sqrt{n})$ calls to a Max-Flow algorithm are enough to resolve any expander, and since this is only done for the $O(1)$ large expanders, the $n^{2.5}$ bound follows.

How can we beat this barrier of $\sqrt{n}$ calls to a Max-Flow

[3]It had appeared to be hard both for breaking the $n^{2.5}$ bound in simple graphs and, until today, for breaking the n^3 bound in multigraphs and weighted graphs. This is discussed in Section 3 in [5].

[4]This is a big difference from related problems that ask for the global connectivity of a graph. There, finding a node from each side is the end of the story.

[5]Suppose for contradiction that one of the leaves $v \in C_\ell$ has higher degree than c_ℓ. Then $\{c_\ell\}$ is a better cut than $\{v\}$ the minimum cut in the tree, violating the requirements of a Gomory-Hu Tree.

algorithm? It is natural to make the expansion parameter ϕ larger, e.g. $1/\log^{O(1)} n$ rather than $1/\sqrt{n}$, so that each expander is guaranteed to have fewer nodes from one of the sides, e.g., $\log^{O(1)} n$ rather than $\sqrt{n}$, and then it can be handled more efficiently. The issue is that the upper bound on the number of outer-edges $\tilde{O}(\phi|E|)$ becomes worse and we end up with $\Omega(n/\log^{O(1)} n)$ high-degree nodes in small expanders that must be checked. On the other hand, making ϕ smaller, e.g. $1/n$, makes the time of handling an expander correspond to $\Omega(n)$ Max-Flow calls. Thus, $\sqrt{n}$ seems like a natural sweet-spot.

The new observation is that even though $\phi = 1/\log^{O(1)} n$ may not allow us to directly find c_ℓ (since it may be among the $\Omega(n/\log^{O(1)} n)$ high-degree nodes in small expanders), it does efficiently reduce the set of candidates for being c_ℓ from $\Omega(n)$ to about $n/\log^{O(1)} n$. If this candidate elimination can be repeated, we will quickly, after $\tilde{O}(1)$ times, reach a set of only $\tilde{O}(1)$ candidates that can be checked directly. To make this repetition possible and to avoid pointlessly eliminating the same set of candidates each time, we utilize a more powerful expander-decomposition with *vertex demands*, that lets us set to 0 the demand of nodes we have already handled. These demands control much better how the vertices of a single expander are split between the two stars, because we can count only nodes that have non-zero demand and are not yet handled. A similar strategy was used by Li and Panigrahy [27] for their deterministic global min-cut algorithm (in weighted graphs), a related but very different problem. The two algorithms share a certain high-level strategy, even though almost all the details are different. These two striking applications of expander-decomposition with vertex demands indicate that we are likely to see additional applications of this powerful technology.

2) From One Hard Cut to Single-Source Cuts using New Structural Results: After resolving the hard case of the previous section, where there was only one "candidate" c_ℓ with a difficult-to-find cut, we move on to the more general setting. Suppose we are given a source (or pivot) node p, such as c_r in the previous example, and want to compute the minimum cut to all other nodes $v \in V$. For concreteness and simplicity, assume that we are only interested in cuts of value between w and $2w$, where $w = \Theta(n)$. It might seem that the strategy of the previous section simply works because c_ℓ could have been any high-degree node: we take all nodes of degree $\geq w$ to be candidates and the $\tilde{O}(1)$ calls to Max-Flow are magically supposed to find the minimum p, v-cut for all of them.

Alas, from a closer look we see that the fact that c_ℓ is the highest degree node in its cut C_ℓ is crucially used in the second of the two methods described above, and this will not be the case for all high-degree nodes; from now on we will call this second method Procedure LEFTY because it handles expanders with few nodes from the right side.[6] The strategy taken in this paper and also (in a much less efficient way) in [5] is to talk about *estimates* instead of degrees. The estimate $c'(v)$ of a node v is initially the degree of v, and it is reduced whenever a p, v-cut of smaller value is found. Eventually, when a node is "done" the estimate becomes equal to the value of the minimum p, v-cut (that is always upper bounded by the degree of v). In Procedure LEFTY in [5], instead of taking the $O(\phi^{-1})$ highest degree nodes from an expander, we actually take the highest estimate nodes. This lets us compute correct cuts not only for nodes that have the highest degree among their cut-members, but also those whose current estimate is highest. If this (the entire algorithm) is repeated enough times, letting the estimates improve after each repetition, it is guaranteed that all the minimum cuts will be found.[7]

But how many times should the algorithm be repeated? In [5] it is argued that $\tilde{O}(1)$ repetitions are sufficient because the algorithm has the budget to make $\sqrt{n}$ additional invocations each time Procedure LEFTY is called and because the depth of the Gomory-Hu Tree can be upper bounded by $\sqrt{n}$ by taking a random hitting set of $\sqrt{n}$ nodes. Unfortunately, both of these things are not possible if we want $n^{2+o(1)}$ time. First, we cannot spend $\sqrt{n}$ additional invocations and second, the depth reduction itself already has $\Omega(n^{2.5})$ running time.

The approach taken in this paper is different, much more efficient, and exploits the fact that the graph is simple in a deeper way. We do not repeat the algorithm at all, but we augment Procedure LEFTY so that it does not stop handling an expander until the node with highest estimate in the expander is done. The number of calls to Max-Flow can no longer be upper bounded by $O(\phi^{-1}) = \tilde{O}(1)$ and, in principle, some expanders might incur a much larger number of calls. But we can provide an upper bound on the total number of calls using the fact that each additional call was a result of a previously undone node becoming done, meaning that a new minimum cut was found. In fact, the new cut is also not "easy" in the sense that it contains at least two high-degree nodes; this important point will not be discussed in this overview (except briefly in Section I-B4 Item 4). Then, using a new bound on a certain notion of depth of a Gomory-Hu Tree of a simple graph, we can upper bound the number of such cuts by $O(n/w)$ which is a negligible overhead for

[6]Note added in proof: In the independent paper of Li *et al.* [29] this issue does not arise because they simply use the first of the two procedures (that we call Procedure RIGHTY) in a symmetric set-up to solve the second case as well. The same can be done in our algorithm too, which would resolve the issue discussed in this part of the overview more easily, and lead to an overall simpler algorithm and an analysis that does not require the new structural results discussed here (though they might still have independent interest).

[7]To get intuition, observe that if a node v is the only node in its cut C_v that is not done, then it must have a higher estimate than all other nodes in C_v. The same cannot be said about the degree.

our algorithm. This bound is proved in the full version using combinatorial arguments of the following flavor: Suppose there is a set of k minimum cuts of value between w and $2w$ that nested (each cut is a subset of the previous one). If most of them have "private sets" of $\Omega(w)$ nodes that do not belong to the other cuts, then $k = O(n/w)$, as desired. Otherwise, there is a sequence of 10 cuts with only $o(w)$ private nodes each; now, because the graph is simple and each of these cuts must contain a node of degree $\geq w$, we get that each of these cuts has $(1 - o(1))w$ edges leaving the entire set of 10 cuts. But then there are $(10 - o(1))w$ edges leaving the largest of these cuts, contradicting the fact that its value is $\leq 2w$. This structural result follows from basic combinatorics, but when used properly (e.g., one has to bypass complications with easy cuts), it can lead to significant algorithmic savings.

3) Derandomization using a Dynamic Pivot: Until now, we have described how to find the minimum cuts from a single pivot p to all other nodes $v \in V$. Let use remark that essentially all techniques needed for our $n^{2+o(1)}$ time single-source minimum cuts algorithm can be derandomized with standard tools.[8] To fully construct a Gomory-Hu Tree, this algorithm is used recursively in a manner that is similar to the Gomory-Hu algorithm. There is an intermediate tree T that starts from a single super-node containing all of V, and gets refined throughout the recursion until eventually each node is in its own super-node. A refinement of the tree T is performed by dividing a super-node into smaller super-nodes, based on information about some minimum cuts. The Gomory-Hu algorithm divides the super-node into two each time, while our algorithm (since it has all the minimum cuts from a single pivot) may divide it into many super-nodes at once.[9] For purposes of efficiency, we want to make the recursion depth logarithmic. This is guaranteed if the cuts from the pivot happen to divide the graph in a balanced way; but unfortunately, it could be the case that the minimum cut from the pivot p to all other nodes is $(\{p\}, V \setminus \{p\})$ and we have learned very little. A popular way to circumvent this issue ([9], [3], [5]), used also in our randomized algorithm, is to pick the pivot at random and argue that with high probability, many of the cuts will be balanced. But it is not clear how a deterministic algorithm can avoid depth $\Omega(n)$ due to a sequence of bad pivots.

One approach is to use the new structural results to argue that, in a simple graph, if we always pick nodes of degree $\geq w$ as pivots, the number of calls to the single-source algorithm will be $O(n/w)$. This would be an improvement over $\Omega(n)$ calls, but it will not lead to an $n^{2+o(1)}$ bound (even if Max-Flow can be solved in linear time).

The approach we take is both more powerful and conceptually simpler. We modify the single-source algorithm so that it is guaranteed to return balanced cuts: if this is not possible because the pivot is bad, then the single-source algorithm may change the pivot into a better one. A good pivot is always guaranteed to exist, e.g. the centroid p^* of the Gomory-Hu Tree has the property that for all $v \in V$, the minimum p^*, v-cut has $\leq n/2$ nodes in the side of v.

In more detail, we start from the highest degree node as the pivot p and proceed with the single-source algorithm as usual. Whenever a cut C_q for node $q \in V$ is obtained, either via a call to Max-Flow or via the ISOLATING-CUTS procedure (Lemma II.9), we check to see if the cut is good in the sense that the side of q has $\leq n/2$ nodes. (We work with so-called latest cuts, see the full version and [17]) so that if a good cut exists, it will be found.) If the cut is not good, the algorithm runs a *pivot-change protocol* that makes q the new pivot. In this protocol, the estimates and cuts of all nodes $v \in V$ are updated; fortunately, due to a triangle-inequality-like property of cuts,[10] it can all be done in $O(1)$ time per node $v \in V$, which is negligible compared to the time for solving the Max-Flow leading to that pivot-change. And that is it; the algorithm can proceed as if q was the pivot all along.

4) Further Complications: Finally, let us mention the complications that arise when turning the above ideas into a full algorithm for constructing a Gomory-Hu Tree. Happily, some of the complications in our previous work [5] were alleviated due to the recent discovery of a Max-Flow algorithm that runs in near-linear time when $m = \Omega(n^{1.5})$ [38].

1) The first reason that our actual algorithms are more complicated than what has been described is that the single-source algorithm must work with auxiliary graphs of the input graph G, and these are obtained by contracting certain nodes (as in the Gomory-Hu framework). While the input graph is simple, the auxiliary graphs could have parallel edges, and so the same arguments cannot be applied in a black-box manner. Still, we apply them by going back and forth (in both the algorithm and analysis) between the nodes of the auxiliary graph and of the input graph. While this does not introduce substantial technical difficulties, the notation is heavier and it takes some time to get used to.
2) When searching for cuts of value $w = o(n)$ the number of calls to a Max-Flow algorithm is higher. Typically, our algorithms make $\tilde{O}(n/w)$ calls to a Max-Flow algorithm. Fortunately, as w gets smaller, each call becomes cheaper because it can be applied on a Nagamochi-Ibaraki sparsifier [33] of the graph that only has $O(nw)$ edges and preserves all cuts of value

[8]The only exception is the k-partial tree that is only needed as long as the upper bound for Max-Flow is not almost-linear. For the purposes of this overview, it is not needed.

[9]This is a crucial point, as discussed in our previous papers [4], [3]

[10]It is well-known that $\min(\lambda_{x,y}, \lambda_{y,z}) \leq \lambda_{x,z}$ for all nodes x, y, z, where $\lambda_{u,v}$ denotes the minimum u, v-cut value.

$\leq w$. So if we had an almost-linear time Max-Flow algorithm, all the calls together would overall take $(nw)^{1+o(1)} \cdot n/w = n^{2+o(1)}$ time. But using the current Max-Flow algorithms each call takes $\tilde{O}(nw + n^{1.5})$, which is larger when $w = o(\sqrt{n})$, making the overall time super-quadratic. To circumvent this issue, we start with a preliminary step that computes a $\sqrt{n}$-partial tree of the graph [9] in $\tilde{O}(n^2)$ (randomized) time, and afterwards all cuts we are interested in will have $w > \sqrt{n}$.

3) In our randomized algorithm, where we use the standard randomized pivot selection to go from a single-source algorithm to a full tree, it is convenient to assume that all minimum cuts are unique. Otherwise the Max-Flow algorithm might adversarially return bad (unbalanced) cuts with respect to whatever pivot we choose. This can be achieved by adding a small perturbation to each edge, which guarantees that ties are broken in some consistent manner and does not ruin the arguments that need the graph to be unweighted because we can still analyze the underlying simple graph. There are other ways to circumvent this issue; for example, by dropping the randomized pivot strategy altogether and using our dynamic pivot technique instead. Still, we have chosen to present this method because it may be the easiest to follow for readers who are familiar with prior work.

4) Finally, there is a step in our single-source algorithm that might seem unnecessary at first. We invoke the Isolating-Cuts procedure to compute all easy cuts (recall these contain only one node of degree $\geq w$). We call them easy because they are indeed easy to compute using a single call to Max-Flow and without any expansion-based arguments. The need for this step is in order to bound the number of new cuts obtained in Procedure Lefty later on in the algorithm using our new structural results. This result can only bound the number of cuts that are not easy. Therefore, omitting this step might make Procedure Lefty prohibitively expensive.

Note added in proof.
new paragraph should be here

II. Preliminaries

A. Gomory-Hu's Algorithm and Partial Trees

First, we give some general definitions that are often used by algorithms for Gomory-Hu trees.

Partition Trees.: A *partition tree* T of a graph $G = (V, E)$ is a tree whose nodes $1, \ldots, l$ are *super-nodes*, which means that each node i is associated with a subset $V_i \subseteq V$; and these super-nodes form a disjoint partition $V = V_1 \sqcup \cdots \sqcup V_l$. We will assume that the edges of a partition tree are weighted in the natural way: each edge in T represents a vertex bipartition in G and we can define its weight to be the value of this cut in G. An *auxiliary graph* G_i is constructed from G by merging nodes that lie in the same connected component of $T \setminus \{i\}$. These merged nodes (representing multiple nodes in G) will be called *contracted node*. For example, if the partition tree is a path on super-nodes $1, \ldots, l$, then G_i is obtained from G by merging $V_1 \cup \cdots \cup V_{i-1}$ into one contracted node and $V_{i+1} \cup \cdots \cup V_l$ into another contracted node. We will use the notations $n'_i := |V_i|$, $m'_i := |E(G_i)|$, and $n_i := |V(G_i)|$. Note that $n'_i \leq n_i$ since $V(G_i)$ contains V_i as well as some other contracted nodes, with $n'_i = n_i$ only if the tree T has a single super-node. The following is a brief description of the classical Gomory-Hu algorithm [21] (see Figure 2).

The Gomory-Hu algorithm.: This algorithm constructs a cut-equivalent tree $\mathcal{T}$ in iterations. Initially, $\mathcal{T}$ is a single node associated with V (the node set of G), and the execution maintains the invariant that $\mathcal{T}$ is a partition tree of V. At each iteration, the algorithm picks arbitrarily two graph nodes s, t that lie in the same tree super-node i, i.e., $s, t \in V_i$. The algorithm then constructs from G the auxiliary graph G_i and invokes a Max-Flow algorithm to compute in this G_i a minimum st-cut, denoted C'.

The submodularity of cuts ensures that this cut is also a minimum st-cut in the original graph G, and it clearly induces a disjoint partition $V_i = S \sqcup T$ with $s \in S$ and $t \in T$. The algorithm then modifies $\mathcal{T}$ by splitting super-node i into two super-nodes, one associated with S and one with T, that are connected by an edge whose weight is the value of the cut C', and further connecting each neighbor of i in $\mathcal{T}$ to either S or T (viewed as super-nodes), depending on its side in the minimum st-cut C' (more precisely, neighbor j is connected to the side containing V_j).

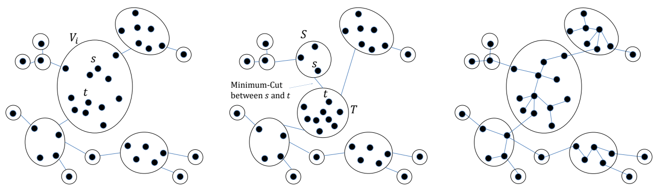

Figure 2: An illustration of the construction of $\mathcal{T}$. Left: $\mathcal{T}$ right before the partition of the super-node V_i. Middle: after the partitioning of V_i. Right: $\mathcal{T}$ as it unfolds after the Gomory-Hu algorithm finishes.

The algorithm performs these iterations until all super-nodes are singletons, which happens after $n - 1$ iteration. Then, $\mathcal{T}$ is a weighted tree with effectively the same node set as G. It can be shown [21] that for every $s, t \in V$, the minimum st-cut in $\mathcal{T}$, viewed as a bipartition of V, is also a minimum st-cut in G, and of the same cut value. We stress that this property holds regardless of the choice made at each step of two nodes $s \neq t \in V_i$.

A GH-Equivalent Partition Tree is a partition tree that can be obtained by a truncated execution of the Gomory-Hu algorithm, in the sense that there is a sequence of choices for the pairs $s \neq t \in V_i$ that can lead to such a tree. A basic property of partition trees and auxiliary graphs is the following.

Lemma II.1 ([21]). *Let T be a GH-Equivalent Partition Tree of a graph $G = (V, E)$ and let G' be an auxiliary graph of G on original nodes $V' \subseteq V$. For all $u, v \in V'$ it holds that Max-Flow$_G(u, v) =$ Max-Flow$_{G'}(u, v)$.*

The following simple lemma describes the flexibility in designing cut-equivalent tree algorithms based on the Gomory-Hu framework, where the proof is deferred to the full version.

Lemma II.2. *Given a GH-Equivalent Partition Tree T' of an input graph G, and a cut-equivalent tree T_i of the auxiliary graph G_i for each super-node V_i in T', one can construct a full cut-equivalent tree T of G in linear time.*

A simple corollary is the following natural lemma that we will use, and is proved in the full version.

Lemma II.3. *Let G be a graph and suppose that $S_1, \dots, S_k$ are non-crossing minimum cuts between a single source p and k targets. Then there exists a Gomory-Hu Tree of G in which all of these k cuts are minimum cuts between p and the respective targets.*

1) k-Partial Trees: A k-partial tree, formally defined below, can also be thought of as taking a cut-equivalent tree of G and contracting all edges of weight greater than k. Such a tree can obviously be constructed using the Gomory-Hu algorithm, but as stated below (in Lemma II.5), faster algorithms were designed in [23], [9], see also [34, Theorem 3]. It is known that such a tree is a GH-Equivalent Partition Tree, see [4, Lemma 2.3].

Definition II.4 (k-Partial Tree [23]). *A k-partial tree of a graph $G = (V, E)$ is a partition tree of G with the following property: For every two nodes $x, y \in V$ whose minimum x, y-cut value in G is at most k, nodes x and y lie in different super-nodes $x \in X$ and $y \in Y$, such that the minimum X, Y-cut in the tree defines a bipartition of V which is a minimum x, y-cut in G and the two cuts have the same value.*

Lemma II.5 ([9]). *There is an algorithm that given an undirected graph having n nodes, m edges, and unit edge-capacities together with an integer $k \in [n]$, constructs a k-partial tree in time $\min\{\tilde{O}(nk^2), \tilde{O}(mk)\}$.*

B. Nagamochi-Ibaraki Sparsification

We use the sparsification method by Nagamochi and Ibaraki [33]. We will need the following simple lemma, showing that sparsification and edge-perturbation can be combined in a convenient way, and proved in the full version.

Lemma II.6 (The Nagamochi-Ibaraki sparsifier of a perturbed graph). *Given an unweighted multigraph graph G on n nodes and m (possibly parallel) edges, an integer $w \geq 1$, and a (weighted) graph $\tilde{G}$ that is a perturbed version of G with unique minimum cuts, one can compute in $O(m)$ time a perturbed sparsifier G_w on $O(nw)$ edges such that:*

- *any cut of weight $< w$ in $\tilde{G}$ has exactly the same weight in G_w, and*
- *any cut of weight $\geq w$ in $\tilde{G}$ still has weight $\geq w$ in G_w.*

Consequently, if we restrict our attention to the cuts of weight $< w$, the sparsifier still has unique minimum cuts.

C. Expander Decomposition

We mostly follow notations and definition from [35]. Let $G = (V, E)$ be an undirected graph with edge capacities. Define the *volume* of $C \subseteq V$ as $\mathrm{vol}_G(C) := \sum_{v \in C} \deg_G(v)$, where the subscripts referring to the graph are omitted if clear from the context, and where for a node v in a graph G we denote by $deg_G(v)$ the total degree of v in G counting multiple edges accordingly. The *conductance* of a cut S in G is $\Phi_G(S) := \frac{\delta(S)}{\min(\mathrm{vol}_G(S), \mathrm{vol}_G(V \setminus S))}$. The *expansion* of a graph G is $\Phi_G := \min_{S \subset V} \Phi_G(S)$. If G is a singleton then $\Phi_G := 1$ by convention. Let $G[S]$ be the subgraph induced by $S \subset V$, and let $G\{S\}$ denote the induced subgraph $G[S]$ but with an added self-loop $e = (v, v)$ for each edge $e' = (v, u)$ where $v \in S, u \notin S$ (where each self-loop contributes 1 to the degree of a node), so that every node in S has the same degree as its degree in G. Observe that for all $S \subset V$, $\Phi_{G[S]} \geq \Phi_{G\{S\}}$, because the self-loops increase the volumes but not the values of cuts. We say that a graph G is a *ϕ-expander* if $\Phi_G \geq \phi$, and we call a partition $V = V_1 \sqcup \dots \sqcup V_h$ a ϕ-expander-decomposition if $\min_i \Phi_{G[V_i]} \geq \phi$.

Theorem II.7 (Theorem 1.2 in [35]). *Given a graph $G = (V, E)$ of m edges and a parameter ϕ, one can compute with high probability a partition $V = V_1 \sqcup \dots \sqcup V_h$ such that $\min_i \Phi_{G[V_i]} \geq \phi$ and $\sum_i \delta(V_i) = O(\phi m \log^3 m)$. In fact, the algorithm has a stronger guarantee that $\min_i \Phi_{G\{V_i\}} \geq \phi$. The running time of the algorithm is $O(m \log^4 m / \phi)$.*

We will need the following strengthening of Theorem II.7, where the input contains vertex demands (to be used instead of vertex degrees). Given in addition a *demand vector* $\mathbf{d} \in \mathbb{R}^V_{\geq 0}$, the graph $G = (V, E)$ is a *$(\phi, \mathbf{d})$-expander* if for all subsets $S \subseteq V$, $\Phi^{\mathbf{d}}_G(S) := \frac{\delta(S)}{\min(\mathbf{d}(S), \mathbf{d}(V \setminus S))} \geq \phi$. The following theorem statement is taken from [27], but its proof is actually a variation of a result from [13], and gives a deterministic algorithm. We are not aware of a faster algorithm, not even a randomized one based for instance on [35].

Theorem II.8 (Theorem $III.8$ in [27]). *Fix $\varepsilon > 0$ and any parameter $\phi > 0$. Given an edge-weighted, undirected graph $G = (V, E, w)$ and a demand vector $\mathbf{d} \in R^V_{\geq 0}$, there is a deterministic algorithm running in time $O(m^{1+\varepsilon})$ that computes a partition $V = V_1 \sqcup \cdots \sqcup V_h$ such that*

1) *For each $i \in [h]$, define a demand vector $\mathbf{d}_i \in \mathbb{R}^{V_i}_{\geq 0}$ given by $\mathbf{d}_i(v) = \mathbf{d}(v) + w(E(v, V \setminus V_i))$ for all $v \in V_i$. Then, the graph $G[V_i]$ is a $(\phi, \mathbf{d}_i)$-expander.*
2) *The total weight of inter-cluster edges is $w(E(V_1, \ldots, V_h)) = \sum_i w(E(V_i, V \setminus V_i)) \leq B \cdot \phi \mathbf{d}(V)$ for $B = (\log n)^{O(1/\varepsilon^4)}$.*

D. The Isolating Cuts Procedure

A key tool that will be used in both our deterministic and randomized algorithms is a simple yet powerful procedure that can compute many minimum cuts all at once with $\tilde{O}(1)$ calls to a Max-Flow algorithm. More specifically, if a subset of the nodes $C \subseteq V$ is given, the procedure returns the minimum *isolating* cut for each node $v \in C$; meaning the minimum cut that separates v from all other nodes in $C \setminus \{v\}$. This is very useful when we can pick a set C that contains only one node from each minimum p, v-cut C_v for many nodes v; then, all of these cuts are guaranteed to be found.

This lemma was introduced by Li and Panigrahi [27] for their deterministic global minimum cut, and was later independently rediscovered in [5] (using a different proof and a slightly different statement where there is a pivot) and used in a Gomory-Hu Tree algorithm. Independently, Li and Panighrahy [28] also used it for (an approximate) Gomory-Hu Tree.

Lemma II.9 (The ISOLATING-CUTS Procedure ([27]. See also [5])). *Given an undirected graph $G = (V, E, c)$ on n nodes and m edges, a* pivot *node $p \in V$, and a set of* connected *vertices $C \subseteq V$, let $(C_v, V \setminus C_v)$ where $v \in C_v, p \in V \setminus C_v$ be the latest minimum (p, v)-cut for each $v \in C$. One can deterministically compute $|C|$ disjoint sets $\{C'_v \subset V\}_{v \in C}$ such that*

$$\forall v \in C, \quad \text{if } C_v \cap C = \{v\} \text{ then } C'_v = C_v$$

in time $O(MF(n, m, c(E)) \cdot \log n)$, that is, if the minimum (p, v)-cut isolates v, then it is in the collection output by the algorithm.

III. GOMORY-HU TREE IN NEAR-QUADRATIC TIME

Recall from Section II-A that an auxiliary graph G' is obtained from a graph $G = (V, E)$ and from a GH-Equivalent Partition Tree T by keeping the nodes $V' \subseteq V$ from one super-node of T and contracting certain subsets of the other nodes as prescribed by T, introducing other nodes $V(G') \setminus V'$ into G'. The connectivity of any pair of original graph nodes $u, v \in V'$ in G and G' is the same (Lemma II.1). To simplify the exposition, we will use the following definition of connectivity of an auxiliary graph; note that the requirement is only for the original graph nodes V' and not for the other nodes in G'.

Definition III.1 (Connectivity of an Auxiliary Graph). *We say that an auxiliary graph G' has connectivity k if for every pair of original graph nodes $u, v \in V'$, the minimum (u, v)-cut in G' (and thus also in G) has value at least k.*

A. Single-Source in Near-Quadratic Time

Theorem III.2. *Given a simple graph $G = (V, E)$ on $N = |V|$ nodes with a designated pivot $p \in V$, an auxiliary graph G' on the graph nodes $V' \subseteq V(G)$ with $n = |V(G')|, m = |E(G')|$ and connectivity $\geq \sqrt{N}$, and a perturbed version $\tilde{G}$ of G' with unique minimum cuts, one can compute the minimum (p, v)-cut in $\tilde{G}$ for all nodes $v \in V'$ in total time $m^{1+o(1)} + Nn^{1+o(1)}$.*

In the rest of this section we prove Theorem III.2. Denote the value of the minimum u, v-cut in $\tilde{G}$ by $\lambda_{u,v} :=$ Max-Flow$_{\tilde{G}}(u, v)$, and fix a sufficiently large constant $\gamma \geq 1$ (it will determine the overall success probability).

The Algorithm: First, initialize an estimate $c'(v) := \deg_{\tilde{G}}(v)$ for all $v \in V'$ along with a witness cut $C_v = \{v\}$. Now for each $j = \lfloor \log \sqrt{N} \rfloor, \ldots, \lceil \log N \rceil$, let $w := 2^j$ and execute the following process. We shall refer to this process as stage w, and it will compute correct estimates $c'(v) = \lambda_{p,v}$ (and witness cuts) for all $v \in V'$ such that $w \leq \lambda_{p,v} < 2w$.

Stage w::

1) Compute a Nagamochi-Ibaraki sparsifier G_w (by Lemma II.6) for the perturbed auxiliary graph $\tilde{G}$, so that this G_w has $O(nw)$ edges, all cuts of value $< 2w$ are preserved, and all cuts of value at least $2w$ still have value at least $2w$.
2) Construct the set $V'_{\geq w} := \{v \in V' \mid \deg_G(v) \geq w\}$ and call the ISOLATING-CUTS procedure (see Lemma II.9) on $G_w, p, V'_{\geq w}$ to get a p, v-cut for all $v \in V'_{\geq w}$ and update the estimates $c'(v)$ and C_v if the new cut has smaller value (and if its value is not $\geq 2w$). This step computes correctly all cuts that contain only one node from $V'_{\geq w}$; such cuts will be called easy below.[11]
3) Initialize a set $C = \{v \in V' \mid c'(v) > w\}$; its nodes will be called *candidates*.
4) While $|C| > \log n$ do:
   - Compute an expander decomposition $(H_1, \ldots, H_\ell)$ of G' (see Theorem II.8) with parameters $\varepsilon := (\log n)^{-1/9} = o(1)$ and $\phi := 2^{-\log^{1/2} n} = n^{-o(1)}$, and demand function

$$d(v) := \begin{cases} w, & \text{if } v \in C \\ 0, & \text{otherwise.} \end{cases} \tag{III.1}$$

[11]Alas, if the true minimum cut is not easy, a returned cut containing one node from S may be incorrect, and the algorithm will not know it.

- Define the *size* of an expander H, denoted $\text{size}_G(H)$, to be the total number of nodes from G that appear in (the possibly contracted nodes) of $V(H) \subseteq V(G')$.[12]
- For each expander H_i with $\text{size}_G(H_i) \geq w/2$ do:
  a) Execute Procedure RIGHTY (in Algorithm 1).
  b) Execute Procedure LEFTY (in Algorithm 2).
  c) Remove all nodes in H_i from C.

5) Now that $|C| \leq \log n$, compute $\lambda_{p,v}$ for all v in C by using Max-Flow invocations in G_w.

Algorithm 1: Procedure RIGHTY

```
1  repeat $2e\gamma\phi^{-1}\ln N$ times
2      $S_{\geq w} \leftarrow$ randomly include each $v \in V(H_i) \cap C$
         independently with probability $\phi$
3      call the ISOLATING-CUTS procedure
         (Lemma II.9) on $G_w, p, S_{\geq w}$ to get a $p, v$-cut
         $S'_v$ and a corresponding value $\delta(S'_v)$ for all
         $v \in S_{\geq w}$
4      foreach $v \in S_{\geq w}$ such that
         $\delta(S'_v) < \min\{c'(v), 2w\}$ do
5          $c'(v) \leftarrow \delta(S'_v)$
6          set $S'_v$ as the witness cut for $v$
```

Algorithm 2: Procedure LEFTY

```
1  Initialize a heap containing the nodes of $V(H_i) \cap C$
     keyed by their estimate $c'(v)$
2  repeat $\alpha = 3\phi^{-1}$ times
3      if heap is empty then
4          halt
5      $v^* \leftarrow$ node with maximum key $c'(v)$ extracted
         from the heap
6      invoke Max-Flow to obtain the minimum
         $p, v^*$-cut in $G_w$, call it $S^*$ and its value $\lambda_{p,v^*}$
7      if $c'(v^*) > \lambda_{p,v^*}$ then
8          $\alpha$++ // increment the number of
               repetitions
9      foreach $v \in S^*$ such that $c'(v) > \lambda_{p,v^*}$ do
10         $c'(v) \leftarrow \lambda_{p,v^*}$
11         set $S^*$ as the witness cut for $v$
12     $C \leftarrow C \setminus \{v^*\}$
```

Running-Time Analysis: We first bound the number of repetitions in Step 4, and then bound the time required for each repetition, where the proofs are postponed to the full version.

Claim III.3. *Every repetition of Step 4 reduces $|C|$ by at least factor 2. Consequently, this step is repeated at most $O(\log n)$ times (at a single stage w).*

Claim III.4. *Each repetition in Step 4 takes time $O(m^{1+\varepsilon}) = m^{1+o(1)}$ plus the time for $O(\phi^{-1}N/w) = N/w \cdot n^{o(1)}$ Max-Flow invocations in a graph with n nodes and $O(nw)$ edges.*

Combining the above two claims we establish that the running time of stage w of the algorithm is bounded by $O(m^{1+\varepsilon}\log n) \leq m^{1+o(1)}$ plus the time for $O(\phi^{-1}N/w \cdot \log n) \leq N/w \cdot n^{o(1)}$ invocations to a Max-Flow algorithm on n nodes and $O(nw)$ edges. Since every stage has $w \geq \Omega(n^{1/2})$, employing the recent $\tilde{O}(m+n^{1.5})$-time Max-Flow algorithm [38], would solve each of these Max-Flow invocations in time $\tilde{O}(nw + n^{1.5}) \leq \tilde{O}(nw)$, which adds up over the entire stage to $\tilde{O}(\phi^{-1}N/w \cdot \log n \cdot nw) = \tilde{O}(Nn \cdot \phi^{-1}\log n) \leq Nn^{1+o(1)}$. The running time of all steps other than 4 is dominated by $O(\log N)$ Max-Flow invocations in G_w, which itself is negligible compared to the above. We have $O(\log N)$ stages, and their overall time bound is $O(\log N) \cdot \tilde{O}(m^{1+\varepsilon}\log n + Nn \cdot \phi^{-1}\log n) \leq \tilde{O}(m^{1+\varepsilon} + Nn\phi^{-1})$.

To conclude the running time analysis, it remains to show Claim III.5 (proved in the full version) that was used to bound the number of additional invocations. Recall that a minimum (p, v)-cut S_v is called easy if S_v contains ≤ 1 nodes of degree $\geq w$.

Claim III.5. *Let G be a simple graph on N nodes, let G' be an auxiliary graph of G with respect to $V' \subset V(G)$ that includes a pivot p, and suppose $\tilde{G}$ is a perturbed version of G' with unique minimum cuts. Then the number of different minimum (p, v)-cuts, over all $v \in V'$, that have value $\geq w$ and are not easy, is at most $O(N/w)$.*

Correctness: We say that a node v is done if $c'(v) = \lambda_{p,v}$, and our goal is to show that at the conclusion of the algorithm all nodes are done. The argument is by induction on the stages. For the base case, note that in the first stage, where $w = \sqrt{N}$, all nodes $v \in V'$ with $\lambda_{p,v} < w$ are done at the beginning of the stage. This holds trivially because, by our assumption that the connectivity of the auxiliary graph G' is $\geq \sqrt{N}$, there cannot be any node $v \in V'$ with $\lambda_{p,v} < w$.

The following claim concludes the proof of correctness, and is proved in the full version.

Claim III.6. *Let $w = 2^j$ for some $j \in \{\lfloor \log \sqrt{N} \rfloor, \ldots, \lceil \log N \rceil\}$ and consider the corresponding stage. Suppose that all nodes $v \in V'$ with $\lambda_{p,v} < w$ are done at the beginning of the stage. Then, with probability at least $1 - 1/N^{2\gamma}$, all nodes with $\lambda_{p,v} < 2w$ are done at*

[12]For example, if H contains two nodes from $V' \subseteq V(G)$ and two contracted nodes from $V(G') \setminus V'$ each of which obtained from ten nodes from G, then $\text{size}_G(H) = 22$.

the end of the stage.

IV. Conclusion

The main result of this paper is an $n^{2+o(1)}$ time algorithm for constructing the Gomory-Hu Tree of an unweighted graph, leading to an almost-optimal algorithm for All-Pairs Max-Flow in unweighted graphs. Even assuming an almost-linear time Max-Flow algorithm (as in Hypothesis I.5) the previously known algorithms [5] could not go below $n^{2.5}$. The improvement is achieved by utilizing a stronger primitive than previous work: an expander decomposition with *vertex demands*. Fitting this tool into the Gomory-Hu framework involves several technical challenges, and a new structural understanding of the Gomory-Hu Tree of a simple graph. To reach the $n^{2+o(1)}$ bound, the algorithm exploits the simplicity of the graph in a second way compared to the $n^{2.5}$ algorithm, and therefore it may be even harder to adapt for weighted graphs. The main open question in the field has become even more outstanding.

Open Question 1. *Can the Gomory-Hu Tree of a weighted graph be constructed in subcubic time?*

The first step towards this question would be to achieve a subcubic algorithm for unweighted *multigraphs* on $O(n^2)$ edges; i.e. to refute Hypothesis I.6. Interestingly, due to Theorem I.7, this is required before an $m^{1+o(1)}$ algorithm can be achieved in simple graphs.

We also introduce a new derandomization technique that replaces the randomized pivot selection of recent works with a dynamic pivot. It gives hope that whatever bounds are achieved with randomized algorithms, in the context of Gomory-Hu Tree algorithms, can also be derandomized.

Finally, an interesting take-away message from this paper is that All-Pairs Max-Flow suddenly appears to be an easier problem than All-Pairs Shortest-Paths. At least in unweighted graphs, and at least with current algorithms, its upper bound is $n^{2+o(1)}$ compared with the $n^{\omega+o(1)}$ for All-Pairs Shortest-Paths. The relative easiness is not only theoretical: the methods used by our All-Pairs Max-Flow algorithm are much less infamous than the fast matrix multiplication that goes into the All-Pairs Shortest-Paths algorithm. However, formally proving a separation between the two is difficult, as it requires proving an $\Omega(n^{2+\varepsilon})$ lower bound for matrix multiplication, and may not even be possible because ω is often conjectured to be 2. Perhaps the more promising direction is by resolving Open Question 1 with an $O(n^{3-\varepsilon})$ time algorithm for All-Pairs Max-Flow in weighted graphs. This would establish a separation, assuming the popular conjecture that All-Pairs Shortest-Paths in weighted graphs requires $n^{3-o(1)}$ time [39], making Open Question 1 even more consequential.

Acknowledgements: We are grateful to Thatchaphol Saranurak for helpful clarifications regarding the literature on randomized versus deterministic expander decompositions.

References

[1] Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):1–20, 2017.

[2] 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.

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

[4] 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.

[5] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for Gomory–Hu tree in unweighted graphs. In *53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC'21, page 1725–1737. ACM, 2021.

[6] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 522–539, 2021.

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

[8] Anand Bhalgat, Richard Cole, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient algorithms for Steiner edge connectivity computationand Gomory-Hu tree construction for unweighted graphs. http://hariharan-ramesh.com/papers/gohu.pdf, 2008. Unpublished full version of [9].

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

[10] 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.

[11] 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.

[12] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Algorithms, reductions and equivalences for small weight variants of all-pairs shortest paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198 of *LIPIcs*, pages 47:1–47:21, 2021.

[13] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167, 2020.

[14] Richard Cole and Ramesh Hariharan. A fast algorithm for computing steiner edge connectivity. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 167–176. ACM, 2003.

[15] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[16] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications*, pages 69–87, 1970.

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

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

[19] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than Goldberg-Rao. *CoRR*, 2021.

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

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

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

[23] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient algorithms for computing all low $s-t$ edge connectivities and related problems. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 127–136. SIAM, 2007.

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

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

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

[27] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 85–92, 2020.

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

[29] 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.

[30] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS*, volume 151 of *LIPIcs*, pages 53:1–53:18, 2020.

[31] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 803–814, 2020.

[32] Aleksander Mądry. Computing maximum flow with augmenting electrical flows. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '16, pages 593–602. IEEE Computer Society, 2016.

[33] Hiroshi Nagamochi and Toshihide Ibaraki. Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs. *Algorithmica*, 7:583–596, 1992.

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

[35] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2616–2635, 2019.

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

[37] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

[38] 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'21, pages 859–869, 2021.

[39] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

[40] Tianyi Zhang. Faster cut-equivalent trees in simple graphs. *arXiv preprint arXiv:2106.03305*, 2021.