THEORY OF COMPUTING, Volume 17 (5), 2021, pp. 1–27
www.theoryofcomputing.org

New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs

Amir Abboud Robert Krauthgamer Ohad Trabelsi

Received December 19, 2019; Revised August 29, 2020; Published September 20, 2021

Abstract. We investigate the time-complexity of the All-Pairs Max-Flow problem: Given a graph with n nodes and m edges, compute for all pairs of nodes the maximum-flow value between them. If Max-Flow (the version with a given source-sink pair s,t) can be solved in time $T(m)$, then $O(n^2)\cdot T(m)$ is a trivial upper bound. But can we do better?

For directed graphs, recent results in fine-grained complexity suggest that this time bound is essentially optimal. In contrast, for undirected graphs with edge capacities, a seminal algorithm of Gomory and Hu (1961) runs in much faster time, $O(n)\cdot T(m)$. Under the plausible assumption that Max-Flow can be solved in near-linear time $m^{1+o(1)}$, this half-century old algorithm yields an $nm^{1+o(1)}$ bound. Several other algorithms have been designed through the years, including $\widetilde{O}(mn)$ time for unit-capacity edges (unconditionally), but none of them break the $O(mn)$ barrier. Meanwhile, no super-linear lower bound is known for undirected graphs.

We design the first hardness reductions for All-Pairs Max-Flow in undirected graphs, giving an essentially optimal lower bound for the *node-capacities* setting. For edge capacities, our efforts to prove similar lower bounds have failed, but we have discovered a surprising new algorithm that breaks the $O(mn)$ barrier for graphs with unit-capacity edges! Assuming $T(m)=m^{1+o(1)}$, our algorithm runs in time $m^{3/2+o(1)}$ and outputs a cut-equivalent tree

A conference version of this paper appeared in the Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA'20) [4].

ACM Classification: F.2.2, G.1.6

AMS Classification: 68W25

Key words and phrases: Gomory–Hu tree, conditional lower bounds, max-flow

 DOI: 10.4086/toc.2021.v017a005

(similarly to the Gomory–Hu algorithm). Even with current Max-Flow algorithms we improve the state of the art as long as $m = O(n^{5/3-\varepsilon})$. Finally, we explain the lack of lower bounds by proving a *non-reducibility* result. This result is based on a new near-linear time $\widetilde{O}(m)$ *nondeterministic* algorithm for constructing a cut-equivalent tree and may be of independent interest.

1 Introduction

In the maximum *st*-flow problem (abbreviated Max-Flow), the goal is to compute the maximum value of a feasible flow between a given pair of nodes s,t (sometimes called *terminals*) in an input graph.[1] Determining the time complexity of this problem is one of the most prominent open questions in fine-grained complexity and algorithms. The best running time known for directed (or undirected) graphs with n nodes, m edges, and largest integer capacity U is $\widetilde{O}(\min\{m^{10/7}U^{1/7},\ m^{11/8}U^{1/4}, m\sqrt{n}\log U\})$ [39, 38, 36], where $\widetilde{O}(f)$ hides logarithmic factors and stands for $O(f\log^{O(1)} f)$. To date, there is no $\Omega(m^{1+\varepsilon})$ lower bound for this problem, even when utilizing one of the popular conjectures of fine-grained complexity, such as the Strong Exponential-Time Hypothesis (SETH) of [31].[2] This gap is debated among experts, and a common belief is that such a lower bound is not possible, since a near-linear-time algorithm should exist. There is also a formal barrier for basing a lower bound for Max-Flow on SETH, as it would refute the so-called Nondeterministic SETH (NSETH) [16]. We will henceforth assume that Max-Flow can be solved in time $m^{1+o(1)}$, and investigate some of the most important questions that remain open under this favorable assumption. (None of our results require this assumption; it only serves to highlight their significance.)

Perhaps the most natural next step after the s,t version is the "all-pairs" version (abbreviated All-Pairs Max-Flow), where the goal is to solve Max-Flow for all pairs of nodes in the graph. This multi-terminal problem, dating back to 1960 [40, 18], is the main focus of our work:

What is the time complexity of computing Max-Flow between all pairs of nodes?

We will discuss a few natural settings, e. g., directed vs. undirected, or node capacities vs. edge capacities, in which the answer to this question may vary. A trivial strategy for solving this problem (in any setting) is to invoke a $T(m)$-time algorithm for the s,t version $O(n^2)$ times, giving a total time bound of $O(n^2)\cdot T(m)$, which is $n^2\cdot m^{1+o(1)}$ under our favorable assumption. But one would hope to do much better, as this all-pairs version arises in countless applications, such as a graph-clustering approach for image segmentation [47].

In undirected edge-capacitated graphs, a seminal paper of Gomory and Hu [26] showed in 1961 how to solve All-Pairs Max-Flow using only $n-1$ calls to a Max-Flow algorithm, rather than $O(n^2)$ calls, yielding an upper bound $O(n)\cdot T(m)$. (See also [28] for a different algorithm where all the $n-1$ calls can be executed on the original graph.) This time bound has improved over the years, following the improvements in algorithms for Max-Flow, and under our assumption it would ultimately be $n\cdot m^{1+o(1)}$.

[1] Throughout, we focus on computing the value of the flow (rather than an actual flow), which is equal to the value of the minimum *st*-cut by the famous max-flow/min-cut theorem [21].

[2] SETH asserts that for every fixed $\varepsilon > 0$ there is an integer $k \geq 3$, such that kSAT on n variables and m clauses cannot be solved in time $2^{(1-\varepsilon)n}m^{O(1)}$.

Even more surprisingly, Gomory and Hu showed that all the n^2 answers can be represented using a single tree, which can be constructed in the same time bound. Formally, a *cut-equivalent tree* to a graph G is an edge-capacitated tree T on the same set of nodes, with the property that for every pair of nodes s,t, every minimum st-cut in T yields a bipartition of the nodes which is a minimum st-cut in G, and of the same value as in T.[3] See also [25] for an experimental study, and the Encyclopedia of Algorithms [43] for more background. The only algorithm that constructs a cut-equivalent tree without making $\Omega(n)$ calls to a Max-Flow algorithm was designed by Bhalgat, Hariharan, Kavitha, and Panigrahi [13]. It runs in time $\widetilde{O}(mn)$ in unit-capacity graphs (or equivalently, if all edges have the same capacity), and utilizes a tree-packing approach that was developed in [19, 29], inspired by classical results of [22] and [20]. However, if Max-Flow can indeed be computed in near-linear time, then none of the later algorithms beat by a polynomial factor the time bound $n \cdot m^{1+o(1)}$ of Gomory and Hu's half-century old algorithm.

The time complexity of All-Pairs Max-Flow becomes higher in settings where Gomory and Hu's "tree structure" [26] does not hold. For instance, in node-capacitated graphs (where the flow is constrained at intermediate nodes,[4] rather than edges) flow-equivalent trees are impossible, since there could actually exist $\Omega(n^2)$ different maximum-flow values in a single graph [30] (see therein also an interesting exposition of certain false claims made earlier). Directed edges make the all-pairs problem even harder; in fact, in this case node capacities and edge capacities are equivalent, and thus this setting does not admit flow-equivalent trees, see [41, 32, 30]. In the last decade, different algorithms were proposed to beat the trivial $O(n^2) \cdot T(m)$ time bound in these harder cases. The known bound for unit-capacity graphs is $O(m^{\omega})$, due to Cheung, Lau, and Leung [17], where $\omega < 2.38$ is the matrix multiplication exponent. A related version, which is obviously no harder than All-Pairs Max-Flow, is to ask (among all pairs of nodes) only for flow values that are at most k, assuming unit node-capacities; for example, the case $k = 1$ is the transitive closure problem (reachability). For $k = 2$, an $\widetilde{O}(n^{\omega})$-time algorithm was shown in [24], and very recently a similar bound was achieved for all $k = O(1)$ [3]. The aforementioned papers [17, 24, 3] also present improved algorithms for acyclic graphs (DAGs). In addition, essentially optimal $\widetilde{O}(n^2)$ algorithms were found for All-Pairs Max-Flow in certain graph families, including small treewidth [11], planar graphs [35], and surface-embedded graphs [14].

The framework of fine-grained complexity has been applied to the all-pairs problem in a few recent papers, although its success has been limited to the directed case. Abboud, Vassilevska-Williams, and Yu [8] proved SETH-based lower bounds for some multi-terminal variants of Max-Flow, such as the single-source all-sinks version, but not all-pairs. Krauthgamer and Trabelsi [33] proved that All-Pairs Max-Flow cannot be solved in time $O(n^{3-\varepsilon})$, for any fixed $\varepsilon > 0$, unless SETH is false, even in the sparse regime $m = n^{1+o(1)}$. This holds also for unit-capacity graphs, and it essentially settles the complexity of the problem for directed sparse graphs, showing that the $O(n^2) \cdot T(m)$ upper bound is optimal if one assumes that $T(m) = m^{1+o(1)}$. Recently, Abboud et al. [3] proved a conditional lower bound that is even higher for dense graphs, showing that an $O(n^{\omega+1-\varepsilon})$-time algorithm would refute the 4-Clique conjecture.

[3]Notice that a minimum st-cut in T consists of a single edge that has minimum capacity along the unique st-path in T, and removing this edge disconnects T to two connected components. A *flow-equivalent tree* has the weaker property that for every pair of nodes s,t, the maximum st-flow value in T equals that in G. The key difference is that flow-equivalence maintains only the *values* of the flows (and thus also of the corresponding cuts).

[4]Granot and Hassin [27] considered a related but different notion of minimum st-cuts with node capacities, where the flow is constrained also by the capacities of the source and sink (in addition to the intermediate nodes), and so an equivalent tree exists and can be computed efficiently. This makes the problem much easier.

However, no non-trivial lower bound is known for undirected graphs.

1.1 The challenge of lower bounds in undirected graphs

Let us briefly explain the difficulty in obtaining lower bounds for undirected graphs. Consider the following folklore reduction from Boolean Matrix Multiplication (BMM) to All-Pairs Reachability in directed graphs (the aforementioned special case of All-Pairs Max-Flow with $k = 1$). In BMM the input is two $n \times n$ Boolean matrices P and Q, and the goal is to compute the product matrix R given by

$$R(a,c) := \bigvee_{b=1}^{n} \big(P(a,b) \wedge Q(b,c)\big), \qquad \forall a,c \in [n].$$

Computing R can be reduced to All-Pairs Reachability as follows. Construct a graph with three layers A,B,C with n nodes each, where the edges are directed $A \to B \to C$ and represent the two matrices: $a \in A$ is connected to $b \in B$ iff $P(a,b) = 1$; and $b \in B$ is connected to $c \in C$ iff $Q(b,c) = 1$. It is easy to see that $R(a,c) = 1$ iff node $a \in A$ can reach node $c \in C$ (via a two-hop path).

This simple reduction shows an $n^{\omega-o(1)}$ lower bound for All-Pairs Reachability in dense directed graphs assuming the BMM conjecture (see [7]), which states that any combinatorial algorithm (i. e., one that does not use fast matrix multiplication techniques) that solves BMM requires $n^{3-o(1)}$ time. Higher lower bounds can be proved by more involved reductions that utilize the extra power of flow over reachability, e. g., an $n^{3-o(1)}$ lower bound in sparse directed graphs assuming SETH [33]. Nevertheless, this simple reduction illustrates the main difficulty in adapting such reductions to undirected graphs.

Consider the same construction but with *undirected edges* (i. e., without the edge orientations). The main issue is that paths from A to C can now have more than two hops—they can crisscross between two adjacent layers before moving on to the next one. Indeed, it is easy to construct examples in which the product $R(a,c) = 0$ but there is a path from a to c (with more than two hops). Even if we try to use the extra power of flow, giving us information about the number of paths rather than just the existence of a path, it is still unclear how to distinguish flow that uses a two-hop path (YES case) from flow that uses only longer paths (NO case).

A main technical novelty of this work is a trick to overcome this issue. The high-level idea is to design large gaps between the capacities of nodes in different layers in order to incentivize flow to move to the "next layer." Let us exhibit how this trick applies to the simple reduction above. Remove the edge orientations from our three-layer graph, and introduce node capacities, letting all nodes in B, the middle layer, have capacity $2n$, and all nodes in $A \cup C$, the other two layers, have capacity 1. Now, consider the maximum flow from $a \in A$ to $c \in C$. If $R(a,c) = 1$ then there is a two-hop path through some $b \in B$, which can carry $2n$ units of flow, hence the maximum-flow value is at least $2n$. On the other hand, if $R(a,c) = 0$ then every path from a to c must have at least four hops, and a maximum flow must be composed of such paths. Any such path must pass through at least one node in $A \cup C \setminus \{a,c\}$, whose capacity is only 1, hence the maximum flow is bounded by $|A \cup C \setminus \{a,c\}| = 2n-2$. This proves the same $n^{\omega-o(1)}$ lower bound as before, but now for undirected graphs with node capacities. We note that the argument for undirected graphs can be simplified a bit if we allow nodes of capacity 0. In Section 4 we utilize this trick in a more elaborate way to prove stronger lower bounds.

1.2 Our results

Our main negative result is the first (conditional) lower bound for All-Pairs Max-Flow that holds in undirected graphs. For sparse, node-capacitated graphs we are able to match the lower bound $n^{3-o(1)}$ that was previously known only for directed graphs [33], and it also matches the hypothetical upper bound $n^{3+o(1)}$.

Theorem 1.1. *Assuming SETH, no algorithm can solve All-Pairs Max-Flow in undirected graphs on n nodes and $O(n)$ edges with node capacities in $[n^2]$ in time $O(n^{3-\varepsilon})$ for some fixed $\varepsilon > 0$.*

Our lower bound holds even under assumptions that are weaker than SETH (see Section 4), as we reduce from the 3-Orthogonal-Vectors (3OV) problem. At a high level, it combines the trick described above for overcoming the challenge in undirected graphs, with the previous reduction of [33] from 3OV to the directed case. However, both of these ingredients have their own subtleties and fitting them together requires adapting and tweaking them very carefully.

Following our Theorem 1.1, the largest remaining gap in our understanding of All-Pairs Max-Flow concerns the most basic and fundamental setting: undirected graphs with edge capacities. What is the time complexity of computing a cut-equivalent tree? The upper bound has essentially been stuck at $n \cdot m^{1+o(1)}$ for more than half a century, while we cannot even rule out a near-linear $m^{1+o(1)}$ running time. To our great surprise, after a series of failed attempts at proving any lower bound, we have noticed a simple way to design a new algorithm for computing cut-equivalent trees for graphs with unit capacities, breaking the longstanding mn barrier!

Theorem 1.2. *There is an algorithm that, given an undirected graph G with n nodes and m edges with unit edge-capacities and parameter $1 \le d \le n$, constructs a cut-equivalent tree in time $\widetilde{O}(md + \Phi(m,n,d))$, where*

$$\Phi(m,n,d) = \max\left\{ \sum_{i=1}^{m/d} T(m,n,F_i) :\ F_1,\dots,F_{m/d} \ge 0, \sum_{i=1}^{m/d} F_i \le 2m \right\}$$

and $T(m,n,F)$ is the time bound for Max-Flow on instances whose flow value is at most a F.

Using the current bound on $T(m,n,F)$ we achieve running time $\widetilde{O}(m^{3/2}n^{1/6})$, and under the plausible hypothesis that $T(m,n) = m^{1+o(1)}$ our time bound becomes $m^{3/2+o(1)}$. In the regime of sparse graphs where $m = \widetilde{O}(n)$ the previous best algorithm of Bhalgat et al. [13] had running time $\widetilde{O}(n^2)$ whereas we achieve $\widetilde{O}(n^{5/3})$, or conditionally $n^{3/2+o(1)}$. In fact, we improve on their upper bound as long as $m = O(n^{5/3-\varepsilon})$. Clearly, this also leads to improved bounds for All-Pairs Max-Flow (with unit edge-capacities), for which the best strategy known is to compute the tree and then extract the answers in time $O(n^2)$.

Note added in proof. Subsequent work further improved the above bound. First, by employing the recent $\widetilde{O}(m+n^{1.5})$-time Max-Flow algorithm [15], our framework immediately gives a better running time $\widetilde{O}(\min\{m^{3/2}n^{1/6}, mn^{3/4}+m^{3/2}\})$. Second, new algorithms were developed for simple graphs: it has started with [6], and has culminated in running time $n^{2+o(1)}$ [5, 37], or a slightly faster $\widetilde{O}(n^2)$ *conditionally* on a linear-time Max-Flow algorithm [48], and an even better bound for sufficiently sparse graphs $\min\{m^{7/6}n^{1/2+o(1)}, mn^{1/2+o(1)} + m^{1/2}n^{5/4+o(1)}\}$ [37].

The main open question remains: Can we prove any super-linear lower bounds for the edge-capacitated case in undirected graphs? Is there an $m^{1+\varepsilon}$ lower bound under SETH for constructing a cut-equivalent tree? Perhaps surprisingly, we prove a strong barrier for the possibility of such a result.

We follow the non-reducibility framework of Carmosino et al. [16]. Intuitively, if problem A is conjectured to remain hard for nondeterministic algorithms while problem B is known to become significantly easier for such algorithms, then we should not expect a reduction from A to B to exist. Such a reduction would allow the nondeterministic speedups for problem B to carry over to A. To formalize this connection, Carmosino et al. introduce NSETH: the hypothesis that SETH holds against co-nondeterministic algorithms. NSETH is plausible because it is not clear how a powerful prover could convince a sub-2^n-time verifier that a given CNF formula is *not* satisfiable. Moreover, it is known that refuting NSETH would require new techniques since it would imply new circuit lower bounds. Then, Carmosino et al. exhibited nondeterministic (and co-nondeterministic) speedups for problems such as 3-SUM and Max-Flow (using LP duality), showing that a reduction from SAT to these problems would refute NSETH.

Our final result builds on Theorem 1.2 to design a near-linear time[5] *nondeterministic* algorithm for constructing a cut-equivalent tree. This algorithm can perform nondeterministic choices and in the end, outputs either a correct cut-equivalent tree or “don’t know” (i. e., aborts), however we are guaranteed that for every input graph there is at least one sequence of nondeterministic choices that leads to a correct output. Our result could have applications in computation-delegation settings and may be of interest in other contexts. In particular, since our nondeterministic witness can be constructed deterministically efficiently, namely, in polynomial but super-linear time, it provides a potentially interesting *certifying algorithm* [42, 9] (see [34] for a recent paper with a further discussion of the connections to fine-grained complexity). Our final non-reducibility result is as follows.

Theorem 1.3. *If for some $\varepsilon > 0$ there is a deterministic fine-grained reduction proving an $\Omega(m^{1+\varepsilon})$ lower bound under SETH for constructing a cut-equivalent tree of an undirected unit edge-capacitated graph on m edges, then NSETH is false.*

Our result (and this framework for non-reducibility) does not address the possibility of proving a SETH based lower bound with a randomized fine-grained reduction. This is because NSETH does not remain plausible when faced against randomization (see [16, 46]). That said, we are not aware of any examples where this barrier has been bypassed with randomization.

Roadmap. Our main algorithm is described in the Section 2. The nondeterministic algorithm and non-reducibility result are presented in Section 3. We then present our lower bounds in Section 4. The last section discusses open questions.

2 Algorithm for a cut-equivalent tree

The basic strategy in our algorithm for unit edge-capacities is to handle separately nodes whose connectivity (to other nodes) is high from those whose connectivity is low. The motivation comes from the simple

[5] We say that a time bound $T(n)$ is near-linear if it is bounded by $O(n \log^c n)$ for some constant $c > 0$.

observation that the degree of a node is an upper bound on the maximum flow from this node to any other node in the graph. Specifically, our algorithm has two stages. The first stage uses one method (of partial trees [29, 13]), to compute the parts of the tree that correspond to small connectivities, and the second stage uses another method (the classical Gomory–Hu algorithm [26]) to complete it to a cut-equivalent tree (see Figure 1). Let us briefly review these two methods.

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 tree; each tree node i is a *super-node*, which means that it is associated with a subset $V_i \subseteq V$; and these super-nodes form a partition $V = V_1 \sqcup \cdots \sqcup V_l$. 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 an auxiliary graph G' by merging nodes that lie in the same connected component of $\mathcal{T} \setminus \{i\}$ and invokes a Max-Flow algorithm to compute in this G' a minimum st-cut, denoted C'. (For example, if the current tree is a path on super-nodes $1,\dots,l$, then G' is obtained from G by merging $V_1 \cup \cdots \cup V_{i-1}$ into one node and $V_{i+1} \cup \cdots \cup V_l$ into another node.) The submodularity of cuts ensures that this cut is also a minimum st-cut in the original graph G, and it clearly induces a 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, a neighbor j is connected to the side containing V_j).

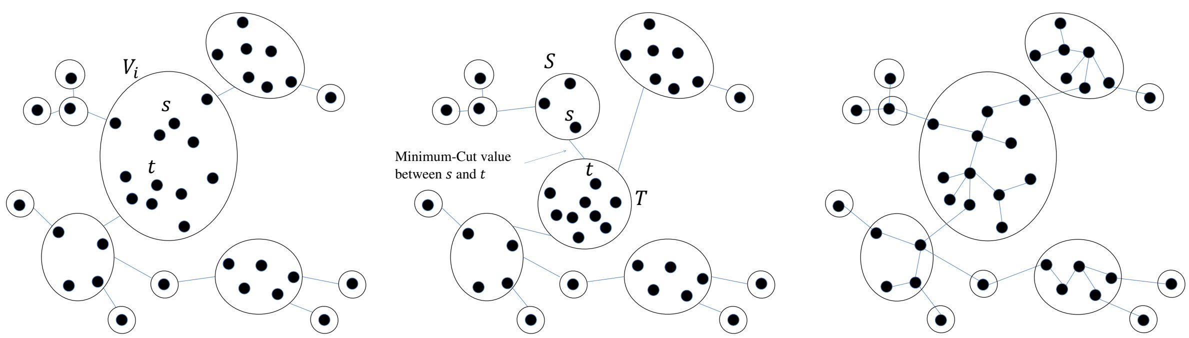

Figure 1: An illustration of the construction of $\mathcal{T}$. Left: $\mathcal{T}$ right before the partition of the super-node V_i. Middle: after the partitioning of V_i. Right: $\mathcal{T}$ as it unfolds after the Gomory–Hu algorithm finishes.

The algorithm performs these iterations until all super-nodes are singletons, and then $\mathcal{T}$ is a weighted tree with effectively the same node set as G. It can be shown [26] 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 $s \neq t \in V_i$.

Partial tree. A k-partial tree, formally defined below, can also be thought of as the result of contracting all edges of weight greater than k in a cut-equivalent tree of G. Such a tree can obviously be constructed using the Gomory–Hu algorithm, but as stated below (in Lemma 2.2), faster algorithms were designed in [29, 13], see also [43, Theorem 3]. We show below (in Lemma 2.3) that such a tree can be obtained also by a truncated execution of the Gomory–Hu algorithm, and finally we use this simple but crucial fact to prove our main theorem.

Definition 2.1 (k-Partial Tree [29])**.** A *k-partial tree* of a graph $G=(V,E)$ is a weighted tree on $l \le |V|$ super-nodes constituting a partition $V = V_1 \sqcup \cdots \sqcup V_l$, with the following property: For every two nodes $s,t \in V$ whose minimum-cut value in G is at most k, s and t lie in different super-nodes $s \in S$ and $t \in T$, such that the minimum ST-cut in the tree defines a bipartition of V which is a minimum st-cut in G and has the same value.

Lemma 2.2 ([13])**.** *There is an algorithm that given an undirected graph with n nodes and m edges with unit edge-capacities and an integer $k \in [n]$, constructs a k-partial tree in time $\widetilde{O}(mk)$.*

Lemma 2.3. *Given a k-partial tree T_{low} of a graph $G=(V,E)$, there is a truncated execution of the Gomory–Hu algorithm that produces T_{low} (i. e., its auxiliary tree $\mathcal{T}$ becomes T_{low}).*

Proof. Consider an execution of the Gomory–Hu algorithm with the following choices. At each iteration, pick any two nodes $s,t \in V$ that lie in the same super-node i of the current tree $\mathcal{T}$ (hence they are feasible choice in a Gomory–Hu execution) *but furthermore* lie in different super-nodes of T_{low}, as long as such s,t exist. Then split super-node i of $\mathcal{T}$ using the minimum st-cut induced by T_{low} (rather than an arbitrary minimum st-cut). As this cut corresponds to an edge in T_{low}, it cannot split any super-node of T_{low}, which implies, by an inductive argument, that the super-nodes of T_{low} are subsets of the super-nodes of $\mathcal{T}$, and thus our chosen cut is a feasible choice for a Gomory–Hu execution. In order to claim that $\mathcal{T}$ will have the same tree structure as T_{low}, we also use the following simple inductive argument. Each pair A,B of super-nodes of $\mathcal{T}$ connected with an edge of capacity c has a pair $A' \subseteq A, B' \subseteq B$ of super-nodes of T_{low} with an edge of capacity c between A' and B' in T_{low}. Notice also that a pair s,t as required above can be chosen as long as $\mathcal{T}$ is not equal to T_{low}, and hence, together with the two inductive claims above, the Gomory–Hu execution continues until $\mathcal{T}$ becomes exactly T_{low}. □

We are now ready to prove our main theorem.

Proof of Theorem 1.2. Let $G=(V,E)$ be an input undirected graph with unit edge-capacities, and denote by V_{low} all the nodes in G whose degrees are at most the chosen parameter $d \in [n]$, and by $V_{\mathrm{high}} = V \setminus V_{\mathrm{low}}$ the nodes whose degrees are greater than d.

First use Lemma 2.2 to construct a d-partial tree T_{low}, and treat it as the auxiliary tree computed by a truncated execution of the Gomory–Hu algorithm. Then continue a Gomory–Hu execution (using this tree) to complete the construction of a cut-equivalent tree. Note that every node in V_{low} is in a singleton super-node of T_{low}, since its minimum cut value to any other node is at most d; thus a super-node V_i in T_{low} has more than one node if and only if it contains only nodes in V_{high}. Moreover, by the properties of T_{low}, two nodes have minimum-cut value greater than d if and only if they are in the same super-node V_i. Since by Lemma 2.3 there exists a truncated Gomory–Hu execution that produces T_{low}, a Gomory–Hu execution starting with T_{low} as the auxiliary tree will result in a cut-equivalent tree and the correctness follows. The running time bound follows as the first step of constructing T_{low} takes $\widetilde{O}(md)$ time, and the second step of the Gomory–Hu execution takes $|V_{\text{high}}|$ invocations of Max-Flow, that is running time $\sum_{i=1}^{m/d} T(m,n,F_i)$. Finally, we will use the following known claim regarding a bound on the total sum of capacities of cut-equivalent trees.

Claim 2.4 (From Lemma 4 in [13]). *Let G be a unit edge-capacity graph G and T its cut-equivalent tree. Then the sum of capacities over all edges of T is bounded by $2m$.*

Now, since every invocation of maximum st-flow with value F_i in our algorithm determines a unique edge with capacity F_i in the final cut-equivalent tree, and by Claim 2.4 the sum of the capacities over all the edges of the cut-equivalent tree satisfies $\sum_{i=1}^{m/d} F_i \leq 2m$, it holds that the total time spent on the m/d invocations of Max-Flow is bounded by $\Phi(m,n,d)$. Thus, the proof of Theorem 1.2 is concluded. □

We use the $T(m,n,F) = \widetilde{O}(m + m^{3/4}n^{1/4}F^{1/2})$ time algorithm by [44] to optimize our running time. By the concavity of $F^{1/2}$, the maximum of $\sum_{i=1}^{m/d} T(m,n,F_i)$ is attained when all $F_i = d$. By setting $d = \sqrt{m}n^{1/6}$ we get

$$\sum_{i=1}^{\sqrt{m}/n^{1/6}} (m + m^{3/4}n^{1/4}m^{1/4}n^{1/12}) = \sum_{i=1}^{\sqrt{m}/n^{1/6}} mn^{1/3} = m^{3/2}n^{1/6},$$

which is faster than the known $\widetilde{O}(mn)$-time algorithm of [13] whenever $m \in [n, n^{5/3}]$.

Finally, relying on a hypothetical $m^{1+o(1)}$-time algorithm for Max-Flow, we could set $d = \sqrt{m}$ to get a total running time of $m^{1+o(1)} \cdot m/\sqrt{m} + \widetilde{O}(m \cdot \sqrt{m}) \leq m^{3/2+o(1)}$, as claimed immediately after Theorem 1.2.

3 Near-linear nondeterministic algorithm for cut-equivalent tree

As no conditional lower bounds are known for the problem of constructing a cut-equivalent tree, one potentially promising approach is to design a reduction from SAT to prove that running time $n^{1+\delta-o(1)}$, for a fixed $\delta > 0$, is not possible assuming SETH. However, in this section we show that the existence of such a reduction (at least in the case of unit edge-capacities) would refute NSETH. This proves our Theorem 1.3.

Our main technical result in this section (Theorem 3.2) is a fast *nondeterministic* algorithm for constructing a cut-equivalent tree (the meaning of this notion will be formalized shortly). We then reach the conclusion about NSETH by following an argument first made in [16], however we have to rewrite their argument (rather than use their definitions and results directly), in order to adapt it from decision problems or functions (where each input has exactly one output) to total search problems, since every graph has at least one cut-equivalent tree (see Section 3.2).

Generally speaking, a search problem P is a binary relation, and we say that S is a solution to instance x iff $(x,S) \in P$. Let $\mathrm{SOL}(x) = \{S : (x,S) \in P\}$ denote the set of solutions for instance x. We say that P is a *total search problem*[6] if every instance x has at least one solution, i. e., $\mathrm{SOL}(x) \neq \emptyset$. Let $\perp$ be the "don't know" symbol and assume that $\perp \notin \mathrm{SOL}(x)$ for all x. For example, in our problem of constructing a cut-equivalent tree, x is a graph and $\mathrm{SOL}(x)$ is the set of all cut-equivalent trees for x.

[6]In some of the previous literature it is called a *total function*, although it is actually a relation rather than a function.

Definition 3.1 (Nondeterministic complexity of a total search problem)**.** We say that a total search problem P has nondeterministic time complexity $T(n)$ if there is a deterministic Turing Machine M such that for every instance x of P with size $|x| = n$:

1. $\forall g, \mathsf{T}_{\mathsf{HALT}}(M(x,g)) \leq T(n)$, i. e., the time complexity of M on input x with guess g is bounded by $T(n)$;
2. $\exists g, M(x,g) \in \mathrm{SOL}(x)$, i. e., at least one guess leads M to output a solution;
3. $\forall g, M(x,g) \in \{\perp\} \cup \mathrm{SOL}(X)$, i. e., every guess leads M to output either a solution or "don't know."

Note that the time bound in item 1 does not depend on g, which is useful for our purpose.

We can now state the main technical result of this section. We prove it in Section 3.1, and then use it in Section 3.2 to prove Theorem 1.3.

Theorem 3.2. *The nondeterministic complexity of constructing a cut-equivalent tree for an input graph with unit edge-capacities is $\widetilde{O}(m)$, where m is the number edges in the graph.*

This algorithm employs the Gomory–Hu algorithm in a very specific manner, where the vertices chosen at each iteration are "centroids" (see below). The same choice was previously used by Anari and Vazirani [10] in the context of parallel algorithms (for planar edge-capacitated graphs), to achieve a logarithmic recursion depth, which is key for parallel time. However, since our goal is different (we want near-linear total time) we have to worry about additional issues, besides the depth of the recursion. Many auxiliary graphs must be handled throughout the execution of the algorithm, and for each one we need to verify multiple minimum cuts. This is done by guessing cuts and flows, and the main challenge is to argue that the total size of all these objects (the auxiliary graphs, and the cuts and flows within them) is only $\widetilde{O}(m)$. Towards overcoming this challenge, we show a basic structural result about cut-equivalent trees (see Claim 3.9 below) which may have other applications. Prior to our work, it seemed unlikely that the Gomory–Hu approach could come close to near-linear time, even if $\mathsf{Max\text{-}Flow}$ could be computed in linear time, since a $\mathsf{Max\text{-}Flow}$ computation is executed many times in many auxiliary graphs. However, our analysis shows that the total size of all these auxiliary graphs can be near-linear (if the right vertices are chosen at each iteration), giving hope that this approach may still achieve the desired upper bound.

3.1 The nondeterministic algorithm

We now prove Theorem 3.2. Let $G = (V,E)$ be the input graph, and let $n = |V|$ and $m = |E|$.

Overview. At a high level, the nondeterministic algorithm first guesses nondeterministically a cut-equivalent tree $\mathcal{T}^*$, and then verifies it by a (nondeterministic) process that resembles an execution of the Gomory–Hu algorithm that produces $\mathcal{T}^*$. Similarly to the actual Gomory–Hu algorithm, our verification process is iterative and maintains a tree $\mathcal{T}$ of super-nodes, which means, as described in Section 2, that every tree node i is associated with $V_i \subseteq V$, and these super-nodes form a partition $V = V_1 \sqcup \cdots \sqcup V_l$. This tree $\mathcal{T}$ is initialized to have a single super-node corresponding to V and then modified at each iteration, hence we shall call it the *intermediate tree*. If all guesses work well, then eventually every super-node is a singleton and the tree $\mathcal{T}$ corresponds to $\mathcal{T}^*$. Otherwise (some step in the verification fails), the algorithm outputs $\perp$.

In a true Gomory–Hu execution, every iteration partitions some super-node into exactly two super-nodes connected by an edge (say $V_i = S \sqcup T$). In contrast, every iteration of our verification process partitions some super-node into multiple super-nodes that form a star topology, whose center is a singleton (say $V_i = \{w\} \sqcup V_{i,1} \sqcup \cdots \sqcup V_{i,d}$, where super-node $\{w\}$ has edges to all super-nodes $V_{i,1}, \ldots, V_{i,d}$). We call this an *expansion step* (see Figure 2), and the node in the center of the star (i. e., w) the *expanded node*. These expansion steps will be determined from the guess $\mathcal{T}^*$. For example, in the extreme case that $\mathcal{T}^*$ itself is a star, our verification process will take only one expansion step instead of $|V|-1$ Gomory–Hu steps.

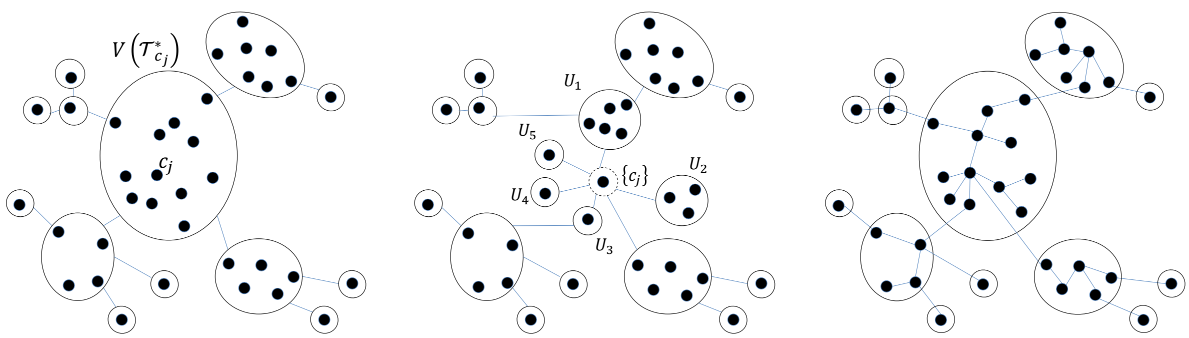

Figure 2: An illustration of the verification of a guessed tree $\mathcal{T}^*$. Left: the intermediate tree $\mathcal{T}$ right before an expansion step of the node c_j in the super-node $V(\mathcal{T}^*_{c_j})$. Middle: after the expansion step (of c_j, in the dashed circle) where $U_1, \ldots, U_4$ are c_j's neighbors in $\mathcal{T}^{(j+1)}$ such that $\bigcup_{i=1}^{4} U_i \cup \{c_j\} = V(\mathcal{T}^*_{c_j})$. Right: the guessed cut-equivalent tree $\mathcal{T}^*$.

To prove that our algorithm is correct, we will show that every expansion step corresponds to a valid sequence of steps in the Gomory–Hu algorithm. As the latter relies on minimum-cut computations in some auxiliary graph G', also our verification will need minimum-cut computations, which can be easily performed in nondeterministic linear time. However, this will not achieve overall running time $\widetilde{O}(m)$, because in some scenarios (e. g., in the above example where $\mathcal{T}^*$ is a star), most of the $|V|-1$ minimum-cut computations are performed on an auxiliary graph G' of size that is comparable to G, i. e., $\Omega(m)$. We overcome this obstacle using two ideas. First, we compute simultaneously all the minimum-cuts of the same expansion step in nondeterministic time that is *linear* in the size of G'. Second, we design a specific sequence of expansion steps such that the total size of *all auxiliary graphs* G' is $\widetilde{O}(m)$.

Detailed algorithm. The algorithm first guesses nondeterministically an edge-capacitated tree $\mathcal{T}^*$, and then verifies, as explained below, that it is a cut-equivalent tree. Here, verification means that upon the failure of any step, e. g., verifying some equality (say between the cut and flow values), the algorithm terminates with output $\perp$. (By the same reasoning, we may assume that all guesses are proper, e. g., a guessed tree is indeed a tree.) The verification process starts by picking a sequence of nodes $c_0, c_1, c_2, \ldots$

using the guess $\mathcal{T}^*$, as follows. Recall that a *centroid* of a tree is a node whose removal disconnects the tree into connected components (subtrees), each containing at most half the nodes in the tree. It is well-known that in every tree, a centroid exists and can be found in linear time. In a *recursive centroid decomposition* of a tree, one finds a centroid of the given tree, removes it and then repeats the process recursively in every connected component, until all remaining components are singletons (have size one). Our verification process computes this decomposition for the guess $\mathcal{T}^*$, which takes time $O(n \log n)$. For each recursion depth $i \geq 0$ (where clearly $i \leq \log n$), denote the set of centroids computed at depth i by $D_i \subset V$. For example, D_0 contains exactly one centroid, of the entire $\mathcal{T}^*$. Now let $c_0, c_1, c_2, \ldots$ be the centroids in this decomposition in order of increasing depth, i. e., starting with the one centroid $c_0 \in D_0$, followed by the centroids from D_1 (ordered arbitrarily), and so forth. Let $\mathcal{T}^*_{c_j}$ be the subtree of $\mathcal{T}^*$ in which the centroid c_j was computed; for example $\mathcal{T}^*_{c_0} = \mathcal{T}^*$.

Observation 3.3. For every two centroids from the same depth, $c_j \neq c_{j'} \in D_i$, the corresponding subtrees $\mathcal{T}^*_{c_j}$ and $\mathcal{T}^*_{c_{j'}}$ are node disjoint.

The verification process now initializes a tree $\mathcal{T}$, called the intermediate tree, to consist of a single super-node associated with V, and then performs on it expansion steps for nodes $c_0, c_1, c_2, \ldots$ (in this order) as explained below.

We now explain how to perform an expansion step for node c_j. Recall that c_j is a centroid of the subtree $\mathcal{T}^*_{c_j}$, therefore it defines a partition $V(\mathcal{T}^*_{c_j}) = \{c_j\} \sqcup U_1 \sqcup \cdots \sqcup U_d$, where $U_1, \ldots, U_d$ are the connected components after removing c_j. Notice that $d = \deg_{\mathcal{T}^*_{c_j}}(c_j) \leq \deg_{\mathcal{T}^*}(c_j)$, and that each U_k, $k \in [d]$, contains exactly one node $u_k \in U_k$ that is a neighbor of c_j in $\mathcal{T}^*_{c_j}$. The expansion step replaces the super-node $V(\mathcal{T}^*_{c_j})$ in $\mathcal{T}$ with $d+1$ super-nodes $\{c_j\}, U_1, \ldots, U_d$. (We slightly abuse notation and use a subset of nodes like $V(\mathcal{T}^*_{c_j})$ also to refer to the super-node in $\mathcal{T}$ associated with this subset.) These $d+1$ new super-nodes are connected by a star topology, where the singleton $\{c_j\}$ at the center and each newly-added edge $(\{c_j\}, U_k)$ is set to the same capacity as the edge (c_j, u_k) in the guess $\mathcal{T}^*$. In addition, every edge that was incident to super-node $V(\mathcal{T}^*_{c_j})$, say $(V(\mathcal{T}^*_{c_j}), W)$, is modified to an edge (U, W), where U is one of the new super-nodes $\{c_j\}, U_1, \ldots, U_d$, chosen according to the edge in $\mathcal{T}^*$ that was used to set a capacity for $(V(\mathcal{T}^*_{c_j}), W)$. (We will explain how the algorithm verifies the correctness of these edge weights shortly.)

It is easy to verify that the modifications to $\mathcal{T}$ (due to expansion steps) maintain the following property: Every super-node U in $\mathcal{T}$ induces a subtree of $\mathcal{T}^*$, i. e., the induced subgraph $\mathcal{T}^*[U]$ is connected. Moreover, eventually every super-node will be a singleton, and the intermediate tree will exactly match the guess $\mathcal{T}^*$. When we need disambiguation, we may use $\mathcal{T}^{(j)}$ to denote the tree's state before the expansion step for c_j. For example, $\mathcal{T}^{(0)}$ is the initial tree with a single super-node V.

Informally, the verification algorithm still has to check that the capacities of the newly-added tree edges correctly represent minimum-cut values. To this end, the algorithm now constructs an auxiliary graph G'_j just as in the Gomory–Hu algorithm (see Section 2). Specifically, G'_j is constructed by taking G, and then for each connected component of $\mathcal{T}^{(j)} \setminus \{V(\mathcal{T}^*_{c_j})\}$ (i. e., after removing super-node $V(\mathcal{T}^*_{c_j})$ from $\mathcal{T}^{(j)}$), merging the nodes in (all the super-nodes in) this component into a single node. Our analysis shows (in Claim 3.6) that for all $s,t \in V(\mathcal{T}^*_{c_j})$, every minimum st-cut in the auxiliary graph G'_j is also a minimum st-cut in G. In addition, all the auxiliary graphs of a single depth q can be constructed in near-linear time (Lemma 3.10).

Observe that each neighbor u_k of c_j in $\mathcal{T}^*_{c_j}$ defines a (c_j,u_k)-cut in the auxiliary graph G'_j, given by the two connected components of $\mathcal{T}^* \setminus \{(c_j,u_k)\}$. The algorithm evaluates for each u_k the capacity of this cut in G'_j, and verifies that it is equal to the capacity of the newly-added edge $(\{c_j\},U_k)$ (set to be the same as of edge (c_j,u_k) in $\mathcal{T}^*$). In fact, all these cuts evaluations are performed not sequentially but rather simultaneously for all $k \in [d]$, as follows. The key observation is that if we denote each aforementioned (c_j,u_k)-cut by $(V(G'_j) \setminus C'_k, C'_k)$, where $u_k \in C'_k$, then $\{c_j\}, C'_1, \dots, C'_d$ are disjoint subsets of $V(G'_j)$. One can clearly evaluate the capacity of all these d cuts in a single pass over the edges of G'_j, and since each edge contributes to at most two cuts (by the disjointness), this entire pass takes only linear time $O(|E(G'_j)|)$.

Next, to verify that each (c_j,u_k)-cut exhibited above, namely, each $(V(G'_j) \setminus C'_k, C'_k)$, is actually a minimum (c_j,u_k)-cut in G'_j, the algorithm finds a flow whose value is equal to the cut capacity. In order to perform this task simultaneously for all $k \in [d]$, our verification algorithm employs a known result about disjoint trees, as a witness for maximum-flow values in a graph with unit edge-capacities (strictly speaking, this witness provides lower bounds on maximum-flow values). In the following theorem, a *directed tree rooted at* r is a directed graph arising from an undirected tree all of whose edges are then directed away from r. This is equivalent to an arborescence (having exactly one path from r to every node), however we will not require that it spans all the graph nodes. In the following, $\mathsf{Max\text{-}Flow}_G(s,t)$ is the maximum st-flow value in a graph G.

Lemma 3.4. *Given an undirected multigraph $H = (V_H, E_H)$, a root node $r \in V_H$, and a function $\lambda : V_H \to [|E_H|]$, it is possible to nondeterministically verify in time $\widetilde{O}(|E_H|)$ that*

$$\forall v \in V_H \setminus \{r\}, \qquad \mathsf{Max\text{-}Flow}_H(r,v) \geq \lambda(v). \tag{3.1}$$

Here, nondeterministic verification means that if (3.1) holds then there exists a guess that leads to output "yes"; and if (3.1) does not hold then every guess leads to output "no."

Proof. We use the following theorem known from [12, Theorem 2.7], in its variation from [19] as the Tree Packing Theorem.

Theorem 3.5. *Let H_e be an Eulerian directed graph, and r_e be a node in H_e. Then there exist $\max_{v \neq r_e}\{\mathsf{Max\text{-}Flow}_{H_e}(r_e,v)\}$ edge-disjoint directed trees rooted at r_e, such that each node $v \in H_e$ appears in exactly $\mathsf{Max\text{-}Flow}_{H_e}(r_e,v)$ trees.*

Given the undirected multigraph H, first subdivide each edge into two edges with a new node in between them, then orient each edge in both directions[7] to obtain an Eulerian directed graph H_e. Observe that the minimum-cut values between pairs of original nodes in H_e are the same as in H. Now find all maximum-flow lower-bound values from r in H_e by guessing $|V_H|$ edge-disjoint trees and then counting occurrences of each node in those trees. By Theorem 3.5, these counts correspond to maximum-flow lower-bound values from r. And so if the guessed trees support the values given by λ, then answer "yes," and otherwise answer "no." Note that the conversion to directed Eulerian graph multiplied the amount of edges by 2, and so the running time is still near linear. □

[7]The subdivision and orientation are used to transform the undirected multigraph to a directed graph.

The verification algorithm then applies Lemma 3.4 to G'_j with c_j as the root, and verifies in time $\widetilde{O}(|E(G'_j)|)$ that the maximum-flow from c_j to each u_k is at least the capacity of the (c_j, u_k)-cut exhibited above (in turn verified to be equal to the capacity of edge (c_j, u_k) in $\mathcal{T}^*$).

Correctness. We begin by claiming that if the guessed tree $\mathcal{T}^*$ is a correct cut-equivalent tree of G, then our algorithm outputs $\mathcal{T}^*$; we discuss the complement case afterwards. Since $\mathcal{T}^*$ is a cut-equivalent tree, every verification step of an expansion will not fail and so the algorithm will not terminate prematurely, and output $\mathcal{T}^*$, as required.

Next, we show that if $\mathcal{T}^*$ is not a cut equivalent tree, then our algorithm will not succeed. This is proved mainly by the claim below, that an intermediate tree attained by expansion steps can be attained also by a sequence of Gomory–Hu steps.

Claim 3.6. *If there is a sequence of Gomory–Hu steps simulating expansions attaining $\mathcal{T}^{(j)}$, and another expansion step is being done to attain $\mathcal{T}^{(j+1)}$, then there is a sequence of Gomory–Hu steps simulating this last step too.*

Proof. Assume there is a truncated execution of the Gomory–Hu algorithm that produces $\mathcal{T}^{(j)}$. We describe a sequence of Gomory–Hu algorithm's steps starting with $\mathcal{T}^{(j)}$ that produces $\mathcal{T}^{(j+1)}$. Recall that $U_1, \dots, U_d$ are $\{c_j\}$'s neighbors in $\mathcal{T}^{(j+1)}$ such that $\bigcup_{i=1}^{d} U_i \cup \{c_j\} = V(\mathcal{T}^*_{c_j})$, and $u_1, \dots, u_d$ are the nodes by which the capacities of the edges $(\{c_j\}, U_k)$, $k \in [d]$, were chosen.

The Gomory–Hu steps are as follows, where we denote by $\mathcal{T}$ the intermediate tree along the execution. Starting with $\mathcal{T} = \mathcal{T}^{(j)}$, for $k = 1, \dots, d$, the Gomory–Hu execution picks the pair c_j, u_k from the super-node containing it in $\mathcal{T}$ as the pair s, t in the Gomory–Hu algorithm description (see the description in Section 2), and the given minimum-cut value between them is asserted. Then, for the partitioning of this super-node in $\mathcal{T}$, the execution picks the minimum-cut between c_j, u_k as in $\mathcal{T}^*$ (which is a minimum cut also in the corresponding auxiliary graph) and modifies the intermediate tree accordingly. Note that the last expansion step was assumed to be successful (i. e., verified correctly), thus all the cuts chosen for the partitioning are minimum-cuts. □

Now, assume for the contrary that $\mathcal{T}^*$ is not a cut-equivalent tree of G and our algorithm still produces it. As a consequence of Claim 3.6, there is a sequence of Gomory–Hu steps attaining $\mathcal{T}^*$, contradicting the proof of correctness of the Gomory–Hu algorithm (which cannot produce $\mathcal{T}^*$). Thus, it is impossible that our algorithm finishes and produces $\mathcal{T}^*$, and so in one of the minimum-cut verifications after an expansion step, the cut witness inspired from $\mathcal{T}^*$ would not be correct, or there would not be a set of directed trees to testify that the corresponding cuts are minimal. This completes the proof of correctness.

Running time. Observe that the running time of a single expansion step, i. e., verifying its corresponding minimum cuts by evaluating cuts and flows, is near-linear in the size of the auxiliary graph. Thus, we only have to show that the total size of all the auxiliary graphs (over all the expansions) is near-linear in m. We prove in Lemma 3.7 below an $O(m)$ bound for a single depth q, and since the depth of the decomposition is $O(\log n)$, we immediately conclude in Corollary 3.8 that the total size of all auxiliary graphs over all depths is $\widetilde{O}(m)$.

Lemma 3.7. *Let $D_q = \{c_{j_1},\dots,c_{j_2}\}$ contain the centroids at depth q. Then the total size of $G'_{j_1},\dots,G'_{j_2}$ is at most $O(m)$.*

Corollary 3.8. *The total size of all auxiliary graphs (over all depths) is $\widetilde{O}(m)$.*

Proof of Lemma 3.7. Let us count for each edge $uv \in E(G)$ in how many auxiliary graphs of depth q it appears. This quantity turns out to be at most $2+(\mathrm{dist}_{\mathcal{T}}(u,v)-1)$, where $\mathrm{dist}_{\mathcal{T}}(u,v)$ is the hop-distance, i. e., the minimum number of edges (ignoring weights or capacities) in a path between u and v in the tree $\mathcal{T}$. The summand 2 comes from edges uv such that either u or v belong to $V(\mathcal{T}_{c_j})$ for some auxiliary graph G'_j. Clearly, every such edge is in at most two auxiliary graphs at depth q, because there is at most one index $j' \in D_q$ where $u \in V(\mathcal{T}_{c_{j'}})$ and at most one index $j'' \in D_q$ where $v \in V(\mathcal{T}_{c_{j''}})$. The summand $\mathrm{dist}_{\mathcal{T}}(u,v)-1$ bounds the other appearances of edge uv, i. e., when neither u nor v belongs to some $V(\mathcal{T}_{c_j})$, and is proved in the claim below. While our graph has unit capacities, the claim holds for general capacities.

Claim 3.9. *For every cut-equivalent tree $\mathcal{T}$ of a graph G with edge capacities $c_G : E \to \mathbb{R}_+$,*

$$\sum_{uv\in E(G)} c_G(u,v)\cdot \mathrm{dist}_{\mathcal{T}}(u,v) \le 2 \sum_{uv\in E(G)} c_G(u,v).$$

Proof. We first show that $\sum_{uv\in E_G} c_G(u,v)\cdot \mathrm{dist}_{\mathcal{T}}(u,v) = \sum_{e\in E_{\mathcal{T}}} c_{\mathcal{T}}(e)$, where $c_{\mathcal{T}}(\cdot)$ denotes edge capacity in $\mathcal{T}$. Recalling that $\mathcal{T}$ is a cut-equivalent tree, each $c_{\mathcal{T}}(e)$ is the value of a certain cut in G, hence we can evaluate the right-hand side differently, by summing over the graph edges $uv \in E(G)$ and counting for each edge in how many such cuts it appears. Moreover, the count for each graph edge $uv \in E(G)$ is exactly $\mathrm{dist}_{\mathcal{T}}(u,v)$ contributions of $c_G(u,v)$, giving altogether the left-hand side.

Second, we show that $\sum_{uv\in E(\mathcal{T})} c_{\mathcal{T}}(u,v) \le 2\sum_{uv\in E(G)} c_G(u,v)$ (see also Lemma 4 in [13]). To see this, observe that $c_{\mathcal{T}}(u,v) \le \min\{\deg_{c_G}(u), \deg_{c_G}(v)\}$ where $\deg_{c_G}(u)$ is the total capacity of edges incident to u. Now fix a root vertex in $\mathcal{T}$, and bound each tree edge by $c_{\mathcal{T}}(u,v) \le \deg_{c_G}(v)$, where v is the child of u (i. e., farther from the root) in $\mathcal{T}$. Summing this bound over all the tree edges and observing that the corresponding vertices v are all distinct proves the desired inequality, and the claim follows. ☐

To complete the proof of Lemma 3.7, recall that by Observation 3.3 the super-nodes $V(\mathcal{T}_{c_{j_1}}),\dots,$ $V(\mathcal{T}_{c_{j_2}})$ of the same depth q are pairwise disjoint. Thus, an edge uv appears in at most $dist_{\mathcal{T}^*}(u,v)-1$ auxiliary graphs of depth q, which totals to $O(m)$ for all the edges in this depth according to the unit edge-capacity special case of the above Claim 3.9. This concludes Lemma 3.7. ☐

Next, we bound the time it takes to construct all the auxiliary graphs.

Lemma 3.10. *The total time it takes to construct the auxiliary graphs for all the expansions in the centroid decomposition is $\widetilde{O}(m)$.*

Proof. Let c_j be a node that is expanded at some depth $q \ge 1$, and let $c_{j,1},\dots,c_{j,d}$ be the expanded nodes in $U_1,\dots,U_d$, respectively at depth $q+1$ (or just $\perp$ for singletons).

Note that $G'_{j,1},\dots,G'_{j,d}$ (whichever exist) can all be constructed in total time that is linear in the size of G'_j.

Thus, the total time it takes to construct the auxiliary graphs at a single depth q is linear in the size of the auxiliary graphs in the parent depth. Since the auxiliary graph at depth 0 (i. e., the entire graph) can trivially be constructed in $O(m)$ time and is linear in the input size, it follows by Corollary 3.8 that the total construction time of the auxiliary graphs for all depths takes at most $\widetilde{O}(m)$ time. □

3.2 Reduction from a decision problem to a total search problem

Let us start with the formal statement of NSETH.

Hypothesis 3.11 (Nondeterministic Strong Exponential-Time Hypothesis (NSETH))**.** For every $\varepsilon > 0$ there exists $k = k(\varepsilon)$ such that k-TAUT (the language of all k-DNF formulas that are tautologies) is not in $\mathsf{NTIME}(2^{n(1-\varepsilon)})$.

Note that deciding if a k-DNF formula is a tautology is equivalent to deciding if a k-CNF formula is satisfiable, thus the above hypothesis could be stated also using k-CNF appropriately. Next, we define (deterministic) fine-grained reductions from a decision problem to a total search problem. Note that these are Turing reductions.

Definition 3.12 (Fine-Grained Reduction from a Decision Problem to a total search problem)**.** Let L be a language and P be a total search problem, and let $T_L(\cdot)$ and $T_P(\cdot)$ be time bounds. We say that (L, T_L) admits a fine-grained reduction to (P, T_P) if for all $\varepsilon > 0$ there is a $\gamma > 0$ and a deterministic Turing machine M^P (with an access to an oracle that generates a solution to every instance of P) such that:

1. M^P decides L correctly on all inputs when given a correct oracle for P.
2. Let $\widetilde{Q}(M^P, x)$ denote the set of oracle queries made by M^P on input x of length n. Then the query lengths obey the bound

$$\forall x, \qquad \mathsf{T}_{\mathsf{HALT}}(M^P, |x|) + \sum_{q \in \widetilde{Q}(M,x)} (T_P(|q|))^{1-\varepsilon} \leq (T_L(n))^{1-\gamma},$$

where $\mathsf{T}_{\mathsf{HALT}}(M^P, |x|)$ is the maximal time M^P runs on inputs of size $|x|$.

We are now ready to prove the non-reducibility result under NSETH for total search problems with small nondeterministic complexity. The proof arguments are similar to those of Carmosino et al. [16].

Theorem 3.13. *Suppose P is a total search problem with nondeterministic time complexity $T(m)$. If for some $\delta > 0$ there is a deterministic fine-grained reduction from k-SAT with time-bound 2^n to P with time bound $T(m)^{1+\delta}$, i. e., from $(k\text{-SAT}, 2^n)$ to $(P, T(m)^{1+\delta})$, then NSETH is false.*

Proof. We will use the assumption of the theorem to describe a nondeterministic algorithm for k-TAUT that refutes NSETH. Let ϕ be an instance of k-TAUT, and note that $\phi \in k$-TAUT iff $\neg\phi \notin k$-SAT. Our nondeterministic algorithm A first computes the CNF formula $\neg\phi$, then simulates the assumed reduction M_1 from k-SAT to P on $\neg\phi$, and eventually outputs the negation of the simulation's answer, or *Reject* if the simulation returns $\perp$.

Let M_2 be the Turing Machine showing that P has nondeterministic time complexity $T(m)$. Whenever the reduction M_1 produces a query to P, our algorithm A executes M_2 on this query with some guess string g. Let g_i be the guess string used for the i^{th} query to P made by M_1. If any of the executions of M_2 throughout the simulation outputs $\perp$, then A stops and outputs *Reject*. Otherwise (all executions output valid answers), the simulation continues until M_1 terminates. At this point, the output of M_1 must be correct, and our algorithm A outputs the opposite answer.

Let us argue about the correctness of our algorithm. First, it only outputs *Accept* if the guesses and all answers to the P-queries were correct and then M_1 rejected, meaning that $\neg\phi \notin k$-SAT i. e., $\phi \in k$-TAUT. Second, for every yes-instance $\phi \in k$-TAUT there is at least one sequence of guesses $g_1, g_2, \dots$ that makes A output *Accept*, due to the correctness of the reduction M_1 and the fact that M_2 nondeterministically computes P correctly. Finally, the running time of A can be upper bounded by

$$\mathsf{T}_{\mathsf{HALT}}(M_1) + \sum_{q \in \widetilde{Q}(M_1,x)} T(|q|) \le \mathsf{T}_{\mathsf{HALT}}(M_1) + \sum_{q \in \widetilde{Q}(M_1,x)} T(|q|)^{(1+\delta)(1-\varepsilon)} \le (2^n)^{1-\gamma}$$

for $0 < \varepsilon < \delta$ where the last inequality is due to the reduction from k-SAT to P, $\mathsf{T}_{\mathsf{HALT}}(M_1)$ is the time of operations done by M_1, $\widetilde{Q}(M_1,x)$ is the queries made by M_1 to the P-oracle on an input x, and the last inequality follows for some $\gamma(\varepsilon) > 0$ because M_1 is a correct fine-grained reduction. Thus, A refutes NSETH. $\square$

Since the construction of a cut-equivalent tree is a total search problem, and by Theorem 1.2 its nondeterministic complexity is $\widetilde{O}(m)$, applying Theorem 3.13 implies that any deterministic reduction from SETH to the construction of a cut-equivalent tree that implies a lower bound of $\Omega(m^{1+\delta})$, for some $\delta > 0$, would refute NSETH, concluding Theorem 1.3.

4 Conditional lower bound for All-Pairs Max-Flow

In this section we prove a conditional lower bound for All-Pairs Max-Flow in undirected graphs with node capacities. Our construction is inspired by the one in [33], which was designed for directed graphs with edge capacities, but we adapt it to undirected graphs with node capacities using our new trick described in the introduction. In fact, readers familiar with the reduction in [33] may notice that we had to tweak it a little, making it simpler in certain ways but more complicated in others. This was necessary in order to apply our new trick to it.

The starting point for our reduction is the 3OV problem.

Definition 4.1 (3OV)**.** Given three sets $U_1, U_2, U_3 \subseteq \{0,1\}^d$ containing n binary vectors each, over dimension d, decide if there is a triple (α, β, γ) of vectors in $U_1 \times U_2 \times U_3$, whose dot product is 0. That is, a triple for which for all $i \in [d]$ at least one of $\alpha[i], \beta[i], \gamma[i]$ is equal to 0.

An adaptation of the reduction by Williams [45] shows that 3OV cannot be solved in $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$ and $d = \omega(\log n)$, unless SETH is false (see [1]). For us, it suffices to assume the milder conjecture that 3OV cannot be solved in $O(n^{3-\varepsilon})$ time when $d = n^{\delta}$, for all $\varepsilon, \delta > 0$. Refuting this conjecture would have important implications beyond refuting SETH [23, 2], e. g., it would refute the Weighted Clique Conjecture.

The high-level structure of the reduction is the following. Create three layers of nodes that correspond to the three sets of vectors, with additional two layers in between them that correspond to the coordinates. These additional layers help keep the number of edges small by avoiding direct edges between pairs of vectors. Among other things, we utilize the trick described in the introduction and set the capacity of the nodes in the leftmost and rightmost sides to be 1, while making the other capacities much larger. This way a flow would not gain too much from crisscrossing through these nodes. Formally, we prove the following.

Lemma 4.2. *3OV over vector sets of size n and dimension d can be reduced to All-Pairs Max-Flow in undirected graphs with $\Theta(n \cdot d)$ nodes, $\Theta(n \cdot d)$ edges, and node capacities in $[2n^2 d]$.*

Since, as explained earlier, the 3OV conjecture is a consequence of SETH, Theorem 1.1 immediately follows from Lemma 4.2, which is proved below.

Proof of Lemma 4.2. Given a 3OV instance F we construct a graph G with maximum flow size between some pair (among a certain set of pairs) bounded by a certain amount if and only if F is a yes instance. For simplicity, we first provide a construction that has some of the edges directed (only where we will specifically mention that), and then we show how to avoid these directions. In addition, some of the edges will be capacitated as well. However, the amount of such edges is small enough so that subdividing them with appropriate capacitated nodes will work too without a significant change to the size of the graph constructed.

An intermediate construction with few directed edges. To simplify the exposition, we start with a construction of a graph G' in which most of the edges are undirected, but some are still directed (see Figure 3).

Our final graph G will be very similar to G'. It will have the same nodes and edges except that all edges will be undirected and the capacities on the nodes will be a little different.

We construct the graph G' on N nodes $V_1 \cup V_2 \cup V_3 \cup A \cup B \cup \{v_B\}$. The layer V_1 contains a node α of capacity 1 for every vector $\alpha \in U_1$. V_2 contains $d+1$ nodes for every vector $\beta \in U_2$: d nodes of capacity 1 denoted by β_i for every $i \in [d]$, plus a node denoted by β' of capacity $d-1$. V_3 contains a node γ of capacity 1 for every vector γ in U_3. The intermediate layer A contains $2d$ nodes: two nodes C_i^0 and C_i^1 of capacity n for every coordinate $i \in [d]$. The other intermediate layer B contains a node C_i of capacity n for every coordinate $i \in [d]$. Finally, the auxiliary node v_B has capacity $n(d-1)$. With a slight abuse of notation, we will use the following symbols in the following ways: α will be either a node in V_1 or a vector in U_1; β will be a vector in U_2; γ will be either a node in V_3 or a vector in U_3; and C_i will be either a node in B or a coordinate in $[d]$. The usage will be clear from context.

The edges of the network will be defined as follows. First, we describe the edges that depend on the given 3OV instance.

- For every α and $i \in [d]$, we add a *directed* edge from α to C_i^0 if $\alpha[i] = 0$, and a directed edge from α to C_i^1 if $\alpha[i] = 1$.
- For every β, we add an (undirected) edge from β_i to C_i if $\beta[i] = 1$.
- For every γ and $i \in [d]$, we add an (undirected) edge from C_i to γ if $\gamma[i] = 1$.

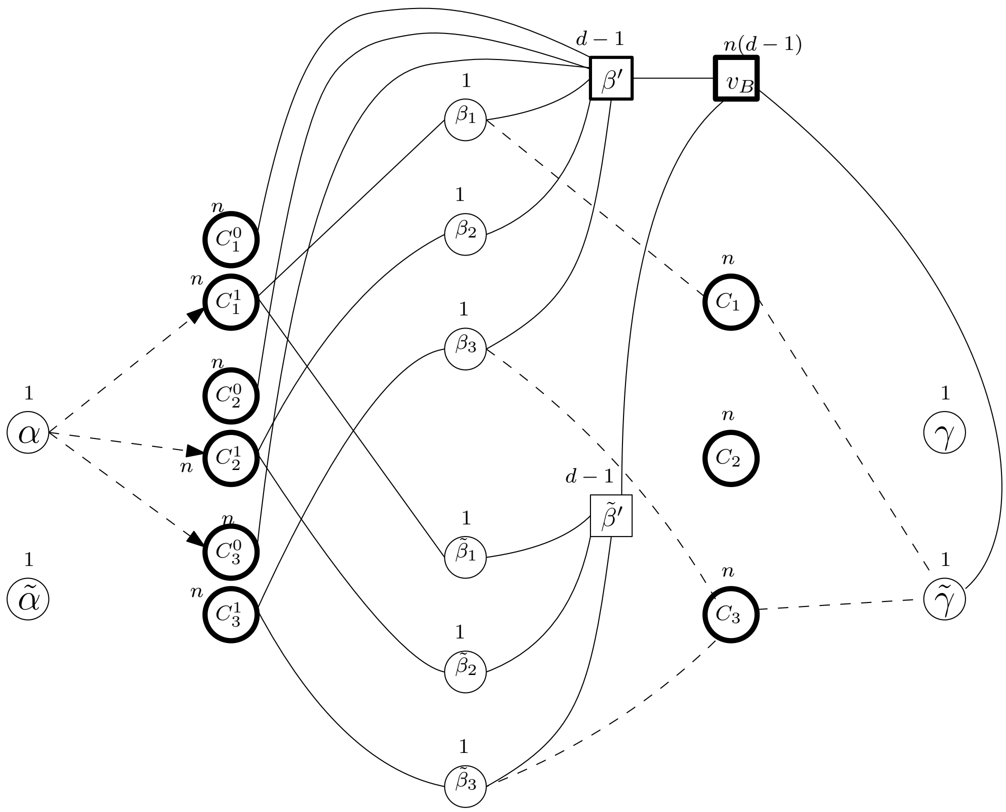

Figure 3: An illustration of part of the reduction. Here, U_1, U_2, and U_3 have two vectors each; α and $\widetilde{\alpha}$ in U_1, β and $\widetilde{\beta}$ in U_2, γ and $\widetilde{\gamma}$ in U_3. Bolder nodes correspond to nodes of higher capacity, and dashed edges are conditional on the input instance. For simplicity, we omit the edges not relevant to α and $\widetilde{\gamma}$, and also the edges from nodes in $\{C_i^0\}_{i\in[3]}$ to nodes in $\{\beta', \widetilde{\beta}'\}$. In this illustration, $\alpha = 110$, $\beta = 101$, $\widetilde{\beta} = 001$, and $\widetilde{\gamma} = 101$. Note that the triple α, $\widetilde{\beta}$, and $\widetilde{\gamma}$ has an inner product 0, and indeed the maximum flow from α to $\widetilde{\gamma}$ is $2 \cdot 3 - 1 = 5$.

Moreover, there will be some (undirected) edges that are independent of the vectors. For every β, we have an edge of capacity 1 from C_i^0 to β', and an edge of capacity 1 from C_i^1 to β_i. Also, for every β, we have an edge from β_i to β', and an edge from β' to v_B. Finally, for every γ, we have an edge from v_B to $\gamma \in V_3$. (Unless specified otherwise, these edges have no capacity constraints.)

The graph built has $N = n + 2d + n \cdot d + n + 1 + d + n = \Theta(nd)$ nodes, at most $O(nd)$ edges, all of its capacities are in $[N]$, and its construction time is $O(nd)$.

The following two claims prove the correctness of this intermediate reduction.

Claim 4.3. *If every triple of vectors in (U_1, U_2, U_3) has inner product at least 1, then for all pairs $\alpha \in V_1, \gamma \in V_3$ the maximum-flow in G' is at least $n \cdot d$.*

Proof. Assume that every triple of vectors in (U_1, U_2, U_3) has inner product at least 1, and fix some α

and γ. We will explain how to send $n \cdot d$ units of flow from α to γ in G'. By the assumption, for every β there exist an $i \in [d]$ such that $\alpha[i] = \beta[i] = \gamma[i] = 1$, and denote this index by i_β. Each i_β induces a path $(\alpha \to C^1_{i_\beta} \to \beta_{i_\beta} \to C_{i_\beta} \to \gamma)$ from α to γ, and so we pass a single unit of flow through every one of them, in what we call the first phase. Note that so far, the flow sums up to n, and we carry on with describing the second phase of flow through nodes of the form β'.

We claim that for every β, an additional amount of $(d-1)$ units can pass through β', which would add up to a total flow of $n(d-1)+n = nd$, concluding the proof. Indeed, for every β, we send flow in the following way. For every $i \in [d] \setminus i_\beta$, if $\alpha[i] = 1$ then we send a single unit through $(\alpha \to C^1_i \to \beta_i \to \beta' \to v_B \to \gamma)$, and otherwise we send a unit of flow through $(\alpha \to C^0_i \to \beta' \to v_B \to \gamma)$.

Since we defined the flow in paths, we only need to show that the capacity constraints are satisfied. Nodes of the form C_i are only used in the first phase, and the flow through them equals n in total, and so their flow is within the capacity. The node v_B is only used in the second phase and has $n(d-1)$ units of flow passing through it, just as its capacity. For every β and $i = i_\beta$, we pass in the first phase a single unit of flow through β_i. For every β and $i \neq i_\beta$, we transfer in the second phase a unit of flow through β_i if and only if $\alpha[i] = 1$, thus it is bounded. For every β', we pass in the second phase exactly $(d-1)$ units of flow through β', preserving its capacity. For every $C^j_i \in N(\alpha)$, where for a node x we denote by $N(x)$ the set of nodes adjacent to x, with $i \in [d]$ and $j \in \{0,1\}$, we pass a total of n units of flow to nodes in V_2, one unit on each edge, thus the capacities are preserved, concluding the proof. □

Claim 4.4. *If there is a triple of vectors $(\alpha_\Phi, \beta_\Phi, \gamma_\Phi) \in (U_1, U_2, U_3)$ whose inner product is 0, then the maximum-flow in G' from $\alpha_\Phi \in V_1$ to $\gamma_\Phi \in V_3$ is at most $nd-1$.*

Proof. Assume for contradiction that there exists such a flow of value at least nd, and denote it by f. Let $f = \{p_1, \ldots, p_{|f|}\}$ be a description of f as a (multi-)set of paths of single units of flow. By our construction, the total capacity of all nodes in $N(\alpha_\Phi)$ sums up to nd exactly. Therefore, f must have all of the nodes in $N(\alpha_\Phi)$ saturated.

Consider a node $C^j_i \in N(\alpha_\Phi)$ for some $i \in [d]$ and $j \in \{0,1\}$. Note that C^j_i is saturated in f while its capacity is n and it has exactly n edges adjacent to it (excluding the edges incoming from V_1) of capacity 1 each. Therefore, we get that every node in $N(C^j_i) \setminus V_1$ must receive a single unit of flow from C^j_i in f. Hence, every β-cloud, which we define as all the nodes that are associated with a β, must have exactly d flow paths in f for which it is the first β-cloud that they pass through. We call this a *first passing* of a path through a β-cloud. In particular, for every β and for every $i \in [d]$ such that $\alpha_\Phi[i] = 1$ there must be a path $p_{\beta,i}$ in f whose prefix is $(\alpha_\Phi, C^1_i, \beta_i, \ldots)$.

Our main claim is that the β_Φ-cloud can only have up to $d-1$ flow paths that are first passing through it. Clearly, if there are more, then at least one of them does not pass through ${\beta_\Phi}'$ (whose capacity is only $d-1$), so name this path p'. We will argue that this path must be in conflict with one of the $p_{\beta,i}$ paths described above.

For some $i \in [d]$ the prefix of p' must be $(\alpha_\Phi, C^1_i, \beta_{\Phi_i}, C_i, \ldots)$, since this is the only way it can avoid the node ${\beta_\Phi}'$. This can only happen for an $i \in [d]$ for which $\alpha[i] = \beta[i] = 1$, or else those edges will not exist in G. But since $(\alpha_\Phi, \beta_\Phi, \gamma_\Phi)$ is a triple whose inner product is 0, it must be that $\gamma_\Phi[i] = 0$ and so the edge $\{C_i, \gamma\}$ is not in the graph. Hence, after C_i this path can only go to a node $\widetilde{\beta}_i$ for some $\widetilde{\beta}$, and the (longer) prefix of p' must be $(\alpha_\Phi, C^1_i, \beta_i, C_i, \widetilde{\beta}_i, \ldots)$. Note that this is the same index i, and we know that $\alpha_\Phi[i] = 1$. Therefore, by the above, we know that there is another path $p_{\widetilde{\beta},i}$ in f that has $\widetilde{\beta}_i$ as the third node on the path. (That is, there is already a path that is first-passing through $\widetilde{\beta}_i$.) This is a contradiction to the fact that the capacity of $\widetilde{\beta}_i$ is 1. $\square$

The final construction. The main issue with avoiding the directions on the edges between nodes in V_1 and A, is that additional α's might participate in the flow as well, potentially allowing one additional unit of flow to pass through. As described in the introduction, the solution is to multiply the capacities of all nodes that are not in $V_1 \cup V_3$ by $2n$. This is how we get our final graph G from G'. In the following we show how this modification concludes the proof of Lemma 4.2.

Claim 4.5. *If every triple of vectors in (U_1, U_2, U_3) has inner product at least 1, then for all pairs $\alpha \in V_1, \gamma \in V_3$ the maximum-flow in G is at least $2n^2d$.*

Proof. Since the flow that was defined in Claim 4.3 does not touch nodes in $V_1 \cup V_3$, considering the same flow in G but multiplied by $2n$, we get a new flow that is of size $nd \cdot (2n)$, concluding the proof. $\square$

Claim 4.6. *If there is a triple of vectors $(\alpha_\Phi, \beta_\Phi, \gamma_\Phi) \in (U_1, U_2, U_3)$ whose inner product is 0, then the maximum-flow in G from $\alpha_\Phi \in V_1$ to $\gamma_\Phi \in V_3$ is at most $2n^2d - 1$.*

Proof. Let f be the maximum flow from α_Φ to γ_Φ in G. The paths in f can be divided into two kinds: paths that pass through nodes in $(V_1 \cup V_3) \setminus \{\alpha_\Phi, \gamma_\Phi\}$, and paths that do not. The total contribution of paths of the first kind can be upper bounded by the size of $(V_1 \cup V_3) \setminus \{\alpha_\Phi, \gamma_\Phi\}$, which is $2n - 2$, since the capacity of all nodes in this set is 1. On the other hand, paths from the second kind must obey the directions of the directed edges in G' and can therefore be used in G', except that in G their multiplicity (the amount of flow we push through them) can be larger by a factor of $2n$. Therefore, we can upper bound the total contribution of paths of the second kind by $2n$ times the maximum flow in G', which is $(nd - 1)(2n)$. Thus, the overall flow is at most $(nd - 1)(2n) + 2n - 2 = 2n^2d - 2$, which proves Claim 4.6. $\square$

Since we showed a gap of at least one unit of flow between the yes and the no instances, the proof of Lemma 4.2 is concluded. $\square$

5 Open problems

Many gaps and open questions around the complexity of maximum flow remain after this work. We highlight a few for which our intuitions may have changed following our discoveries.

- Can we break the $O(mn)$ barrier also when the graphs have arbitrary (polynomial) capacities? Our result gives hope that this may be possible.
- Can we reduce the directed case to the undirected, node-capacitated case? Because of our lower bound, it is likely that both of these cases will end up having the same time complexity, and so such a reduction may be possible.

  Note added in proof. Ongoing work by the current authors indeed provides such a reduction but only for the sparse case.

- Can we generalize the nondeterministic algorithm to arbitrary edge capacities? Notice that one obstacle for achieving that goal is finding lower bounds witnesses for flows from a certain source to other nodes.

- Can we prove any conditional lower bound for All-Pairs Max-Flow in undirected graphs with edge capacities? This is obviously the most important and intriguing open question in this context. Our new deterministic and nondeterministic upper bounds make this task more challenging than previously thought.

6 Acknowledgments

We would like to thank Arturs Backurs for asking about the nondeterministic complexity of the problems, Marvin Kunnemann for pointing out the connection to certifying algorithms, and Richard Peng for helpful comments on the different known upper bounds for Max-Flow.

References

[1] AMIR ABBOUD, ARTURS BACKURS, AND VIRGINIA VASSILEVSKA WILLIAMS: Tight hardness results for LCS and other sequence similarity measures. In *Proc. 56th FOCS*, pp. 59–78. IEEE Comp. Soc., 2015. [doi:10.1109/FOCS.2015.14] 17

[2] AMIR ABBOUD, KARL BRINGMANN, HOLGER DELL, AND JESPER NEDERLOF: More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proc. 50th STOC*, pp. 253–266. ACM Press, 2018. [doi:10.1145/3188745.3188938] 17

[3] AMIR ABBOUD, LOUKAS GEORGIADIS, GIUSEPPE F. ITALIANO, ROBERT KRAUTHGAMER, NIKOS PAROTSIDIS, OHAD TRABELSI, PRZEMYSŁAW UZNAŃSKI, AND DANIEL WOLLEB-GRAF: Faster algorithms for all-pairs bounded min-cuts. In *Proc. 46th Internat. Colloq. on Automata, Languages, and Programming (ICALP'19)*, volume 132, pp. 7:1–7:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. [doi:10.4230/LIPIcs.ICALP.2019.7] 3

[4] AMIR ABBOUD, ROBERT KRAUTHGAMER, AND OHAD TRABELSI: New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In *Proc. 31st Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'20)*, pp. 48–61. SIAM, 2020. [doi:10.1137/1.9781611975994.4, arXiv:1901.01412] 1

[5] AMIR ABBOUD, ROBERT KRAUTHGAMER, AND OHAD TRABELSI: APMF < APSP? Gomory–Hu tree for unweighted graphs in almost-quadratic time, 2021. Accepted to FOCS 2021. [arXiv:2106.02981] 5

[6] AMIR ABBOUD, ROBERT KRAUTHGAMER, AND OHAD TRABELSI: Subcubic algorithms for Gomory–Hu tree in unweighted graphs. In *Proc. 53rd STOC*, pp. 1725–1737. ACM Press, 2021. [doi:10.1145/3406325.3451073] 5

[7] AMIR ABBOUD AND VIRGINIA VASSILEVSKA WILLIAMS: Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th FOCS*, pp. 434–443. IEEE Comp. Soc., 2014. [doi:10.1109/FOCS.2014.53] 4

[8] AMIR ABBOUD, VIRGINIA VASSILEVSKA WILLIAMS, AND HUACHENG YU: Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. Preliminary version in STOC'15. [doi:doi:10.1137/15M1050987] 3

[9] EYAD ALKASSAR, SASCHA BÖHME, KURT MEHLHORN, AND CHRISTINE RIZKALLAH: A framework for the verification of certifying computations. *J. Autom. Reason.*, 52(3):241–273, 2014. Preliminary version in CAV'11. [doi:10.1007/s10817-013-9289-2] 6

[10] NIMA ANARI AND VIJAY V. VAZIRANI: Planar graph perfect matching is in NC. *J. ACM*, 67(4):21:1–21:34, 2020. Preliminary version in FOCS'18. [doi:10.1145/3397504] 10

[11] SRINIVASA RAO ARIKATI, SHIVA CHAUDHURI, AND CHRISTOS D. ZAROLIAGIS: All-pairs min-cut in sparse networks. *J. Algorithms*, 29(1):82–110, 1998. [doi:10.1006/jagm.1998.0961] 3

[12] JØRGEN BANG-JENSEN, ANDRÁS FRANK, AND BILL JACKSON: Preserving and increasing local edge-connectivity in mixed graphs. *SIAM J. Discr. Math.*, 8(2):155–178, 1995. [doi:10.1137/S0036142993226983] 13

[13] ANAND BHALGAT, RAMESH HARIHARAN, TELIKEPALLI KAVITHA, AND DEBMALYA PANIGRAHI: An $\widetilde{O}(mn)$ Gomory–Hu tree construction algorithm for unweighted graphs. In *Proc. 39th STOC*, pp. 605–614. ACM Press, 2007. [doi:10.1145/1250790.1250879] 3, 5, 7, 8, 9, 15

[14] GLENCORA BORRADAILE, DAVID EPPSTEIN, AMIR NAYYERI, AND CHRISTIAN WULFF-NILSEN: All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *Proc. 32nd Internat. Symp. Comput. Geom. (SoCG'16)*, pp. 22:1–22:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. [doi:10.4230/LIPIcs.SoCG.2016.22, arXiv:1411.7055] 3

[15] 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 *Proc. 53rd STOC*, pp. 859–869. ACM Press, 2021. [doi:10.1145/3406325.3451108] 5

[16] MARCO L. CARMOSINO, JIAWEI GAO, RUSSELL IMPAGLIAZZO, IVAN MIHAJLIN, RAMAMOHAN PATURI, AND STEFAN SCHNEIDER: Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proc. 7th Innovations in Theoret. Comp. Sci. conf. (ITCS'16)*, pp. 261–270. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. [doi:10.1145/2840728.2840746] 2, 6, 9, 16

[17] HO YEE CHEUNG, LAP CHI LAU, AND KAI MAN LEUNG: Graph connectivities, network coding, and expander graphs. *SIAM J. Comput.*, 42(3):733–751, 2013. [doi:10.1137/110844970] 3

[18] ROBERT T. CHIEN: Synthesis of a communication net. *IBM J. Res. Dev.*, 4(3):311–320, 1960. [doi:10.1147/rd.43.0311] 2

[19] RICHARD COLE AND RAMESH HARIHARAN: A fast algorithm for computing Steiner edge connectivity. In *Proc. 35th STOC*, pp. 167–176. ACM Press, 2003. [doi:10.1145/780542.780568] 3, 13

[20] JACK EDMONDS: Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and Their Appl., Proc. Conf. Calgary 1969*, pp. 69–87. Gordon and Breach, 1970. [doi:10.1007/3-540-36478-1_2] 3

[21] LESTER R. FORD AND DELBERT R. FULKERSON: Maximal flow through a network. *Canad. J. Math.*, 8(3):399–404, 1956. [doi:10.4153/CJM-1956-045-5] 2

[22] HAROLD N. GABOW: A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. System Sci.*, 50(2):259–273, 1995. Preliminary version in STOC'91. [doi:10.1006/jcss.1995.1022] 3

[23] JIAWEI GAO, RUSSELL IMPAGLIAZZO, ANTONINA KOLOKOLOVA, AND R. RYAN WILLIAMS: Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Trans. Algorithms*, 15(2):1–35, 2019. Preliminary version in SODA'17. [doi:10.1145/3196275] 17

[24] LOUKAS GEORGIADIS, DANIEL GRAF, GIUSEPPE F. ITALIANO, NIKOS PAROTSIDIS, AND PRZEMYSŁAW UZNAŃSKI: All-pairs 2-reachability in $O(n^{\omega} \log n)$ time. In *Proc. 44th Internat. Colloq. on Automata, Languages, and Programming (ICALP'17)*, volume 80, pp. 74:1–74:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. [doi:10.4230/LIPIcs.ICALP.2017.74] 3

[25] ANDREW V. GOLDBERG AND KOSTAS TSIOUTSIOULIKLIS: Cut tree algorithms: an experimental study. *J. Algorithms*, 38(1):51–83, 2001. [doi:10.1006/jagm.2000.1136] 3

[26] RALPH E. GOMORY AND TE CHIANG HU: Multi-terminal network flows. *J. SIAM*, 9(4):551–570, 1961. [doi:10.1137/0109047] 2, 3, 7

[27] FRIEDA GRANOT AND REFAEL HASSIN: Multi-terminal maximum flows in node-capacitated networks. *Discr. Appl. Math.*, 13(2-3):157–163, 1986. [doi:10.1016/0166-218X(86)90079-X] 3

[28] DAN GUSFIELD: Very simple methods for all pairs network flow analysis. *SIAM J. Comput.*, 19(1):143–155, 1990. [doi:10.1137/0219009] 2

[29] RAMESH HARIHARAN, TELIKEPALLI KAVITHA, AND DEBMALYA PANIGRAHI: Efficient algorithms for computing all low $s-t$ edge connectivities and related problems. In *Proc. 18th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'07)*, pp. 127–136. SIAM, 2007. ACM DL. 3, 7, 8

[30] REFAEL HASSIN AND ASAF LEVIN: Flow trees for vertex-capacitated networks. *Discr. Appl. Math.*, 155(4):572–578, 2007. [doi:10.1016/j.dam.2006.08.012] 3

[31] RUSSELL IMPAGLIAZZO AND RAMAMOHAN PATURI: On the complexity of k-SAT. *J. Comput. System Sci.*, 62(2):367–375, 2001. [doi:10.1006/jcss.2000.1727] 2

[32] FREDERICK JELINEK: On the maximum number of different entries in the terminal capacity matrix of oriented communication nets. *IRE Trans. Circuit Theory*, 10(2):307–308, 1963. [doi:10.1109/TCT.1963.1082149] 3

[33] ROBERT KRAUTHGAMER AND OHAD TRABELSI: Conditional lower bounds for all-pairs max-flow. *ACM Trans. Algorithms*, 14(4):42:1–42:15, 2018. [doi:10.1145/3212510] 3, 4, 5, 17

[34] MARVIN KÜNNEMANN: On nondeterministic derandomization of Freivalds' algorithm: Consequences, avenues and algorithmic progress. In *Proc. 26th Eur. Symp. Algorithms (ESA'18)*, pp. 56:1–56:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. [doi:10.4230/LIPIcs.ESA.2018.56] 6

[35] JAKUB LACKI, YAHAV NUSSBAUM, PIOTR SANKOWSKI, AND CHRISTIAN WULFF-NILSEN: Single source – all sinks max flows in planar digraphs. In *Proc. 53rd FOCS*, pp. 599–608. IEEE Comp. Soc., 2012. [doi:10.1109/FOCS.2012.66] 3

[36] YIN TAT LEE AND AARON SIDFORD: Path finding methods for linear programming: Solving linear programs in $\widetilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *Proc. 55th FOCS*, pp. 424–433. IEEE Comp. Soc., 2014. [doi:10.1109/FOCS.2014.52] 2

[37] JASON LI, DEBMALYA PANIGRAHI, AND THATCHAPHOL SARANURAK: A nearly optimal all-pairs min-cuts algorithm in simple graphs, 2021. Accepted to FOCS 2021. [arXiv:2106.02233] 5

[38] YANG P. LIU AND AARON SIDFORD: Faster energy maximization for faster maximum flow. In *Proc. 52nd STOC*, pp. 803–814. ACM Press, 2020. [doi:10.1145/3357713.3384247] 2

[39] ALEKSANDER MADRY: Computing maximum flow with augmenting electrical flows. In *Proc. 57th FOCS*, pp. 593–602. IEEE Comp. Soc., 2016. [doi:10.1109/FOCS.2016.70] 2

[40] WATARU MAYEDA: Terminal and branch capacity matrices of a communication net. *IRE Trans. Circuit Theory*, 7(3):261–269, 1960. [doi:10.1109/TCT.1960.1086673] 2

[41] WATARU MAYEDA: On oriented communication nets. *IRE Trans. Circuit Theory*, 9(3):261–267, 1962. [doi:10.1109/TCT.1962.1086912] 3

[42] ROSS M. MCCONNELL, KURT MEHLHORN, STEFAN NÄHER, AND PASCAL SCHWEITZER: Certifying algorithms. *Computer Sci. Review*, 5(2):119–161, 2011. [doi:10.1016/j.cosrev.2010.09.009] 6

[43] DEBMALYA PANIGRAHI: Gomory–Hu trees. In MING-YANG KAO, editor, *Encyclopedia of Algorithms*, pp. 858–861. Springer, 2016. [doi:10.1007/978-1-4939-2864-4_168] 3, 8

[44] AARON SIDFORD AND KEVIN TIAN: Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow. In *Proc. 59th FOCS*, pp. 922–933. IEEE Comp. Soc., 2018. [doi:10.1109/FOCS.2018.00091] 9

[45] R. RYAN WILLIAMS: A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoret. Comput. Sci.*, 348(2–3):357–365, 2005. [doi:10.1016/j.tcs.2005.09.023] 17

[46] R. RYAN WILLIAMS: Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation. In *Proc. 31st Comput. Complexity Conf. (CCC'16)*, pp. 2:1–2:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. [doi:10.4230/LIPIcs.CCC.2016.2, arXiv:1601.04743] 6

[47] ZHENYU WU AND RICHARD LEAHY: An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *ACM Trans. Pattern Anal. Machine Intell.*, 15(11):1101–1113, 1993. [doi:10.1109/34.244673] 2

[48] TIANYI ZHANG: Faster cut-equivalent trees in simple graphs, 2021. [arXiv:2106.03305] 5

AUTHORS

Amir Abboud
Department of Computer Science and Applied Mathematics
Senior Scientist
The Weizmann Institute of Science
Rehovot, Israel
amir.abboud@weizmann.ac.il
`http://weizmann.ac.il/math/AmirAbboud/home`

Robert Krauthgamer
Professor
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science
Rehovot, Israel
robert.krauthgamer@weizmann.ac.il
`http://www.wisdom.weizmann.ac.il/~robi/`

Ohad Trabelsi
Postdoc
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, USA
ohadt@umich.edu
`http://sites.google.com/view/ohadtrabelsi/`

ABOUT THE AUTHORS

AMIR ABBOUD received his Ph. D. at Stanford University in 2017 under Virginia Vassilevska Williams. He was subsequently a Research Staff Member at the theory group in the IBM Almaden Research Center. Since 2021, he has been a faculty member at the Weizmann Institute of Science. Amir mainly works in the emerging field of fine-grained complexity and algorithms, with a special interest in combinatorial problems on graphs and strings. His favorite sport is soccer, and on August 14, 2020 his favorite team was FC Barcelona.

ROBERT KRAUTHGAMER (called "Robi" by his friends and colleagues) received his Ph. D. at the Weizmann Institute of Science in 2001 under Uriel Feige. He was subsequently a postdoc in Berkeley's theory group, and then a Research Staff Member at the theory group in the IBM Almaden Research Center. Since 2007, he has been a faculty member at the Weizmann Institute of Science. Robi's main research area is the design of algorithms for problems involving combinatorial optimization, finite metric spaces, high-dimensional geometry, data analysis, and related areas. His favorite sport since youth has been swimming. Once he swam across the Sea of Galilee in a 10 km competitive race, and was the last one to arrive at the finish line.

OHAD TRABELSI received his Ph. D. at the Weizmann Institute of Science under Robert Krauthgamer. He was subsequently a postdoc in the same group. Since then, he has been a postdoc at the Department of Electrical Engineering and Computer Science at the University of Michigan. Ohad's research focuses on the design of algorithms, and their conditional lower bounds via fine-grained reductions. In particular, he likes to study problems of combinatorial nature such as Maximum Flow, Hamiltonian Path, and various other graph problems. His hobbies include playing soccer, video games, and the recorder, the non-flute musical instrument that according to many, is also more pleasing to the ear.