Faster Algorithms for All-Pairs Bounded Min-Cuts

Amir Abboud
IBM Almaden Research Center, California, USA
amir.abboud@ibm.com

Loukas Georgiadis
University of Ioannina, Greece
loukas@cs.uoi.gr

Giuseppe F. Italiano
LUISS University, Rome, Italy
gitaliano@luiss.it

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

Nikos Parotsidis
University of Copenhagen, Denmark
nipa@di.ku.dk

Ohad Trabelsi
Weizmann Institute of Science, Israel
ohad.trabelsi@weizmann.ac.il

Przemysław Uznański
University of Wrocław, Poland
puznanski@cs.uni.wroc.pl

Daniel Wolleb-Graf
ETH Zürich, Switzerland
daniel.graf@inf.ethz.ch

Abstract

The All-Pairs Min-Cut problem (aka All-Pairs Max-Flow) asks to compute a minimum s-t cut (or just its value) for all pairs of vertices s, t. We study this problem in *directed graphs* with *unit* edge/vertex capacities (corresponding to edge/vertex connectivity). Our focus is on the *k-bounded* case, where the algorithm has to find all pairs with min-cut value less than k, and report only those. The most basic case $k = 1$ is the Transitive Closure (TC) problem, which can be solved in graphs with n vertices and m edges in time $O(mn)$ combinatorially, and in time $O(n^\omega)$ where $\omega < 2.38$ is the matrix-multiplication exponent. These time bounds are conjectured to be optimal.

We present new algorithms and conditional lower bounds that advance the frontier for larger k, as follows:

- A randomized algorithm for *vertex capacities* that runs in time $O((nk)^\omega)$. This is only a factor k^ω away from the TC bound, and nearly matches it for all $k = n^{o(1)}$.
- Two deterministic algorithms for *edge capacities* (which is more general) that work in DAGs and further reports a minimum cut for each pair. The first algorithm is combinatorial (does not involve matrix multiplication) and runs in time $O(2^{O(k^2)} \cdot mn)$. The second algorithm can be faster on dense DAGs and runs in time $O((k \log n)^{4^{k+o(k)}} \cdot n^\omega)$. Previously, Georgiadis et al. [ICALP 2017], could match the TC bound (up to $n^{o(1)}$ factors) only when $k = 2$, and now our two algorithms match it for all $k = o(\sqrt{\log n})$ and $k = o(\log \log n)$.
- The first super-cubic lower bound of $n^{\omega-1-o(1)}k^2$ time under the 4-Clique conjecture, which holds even in the simplest case of DAGs with unit vertex capacities. It improves on the previous (SETH-based) lower bounds even in the unbounded setting $k = n$. For combinatorial algorithms, our reduction implies an $n^{2-o(1)}k^2$ conditional lower bound. Thus, we identify new settings where the complexity of the problem is (conditionally) higher than that of TC.

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 7; pp. 7:1–7:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our three sets of results are obtained via different techniques. The first one adapts the network coding method of Cheung, Lau, and Leung [SICOMP 2013] to vertex-capacitated digraphs. The second set exploits new insights on the structure of latest cuts together with suitable algebraic tools. The lower bounds arise from a novel reduction of a different structure than the SETH-based constructions.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Network flows

Keywords and phrases All-pairs min-cut, k-reachability, network coding, Directed graphs, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.7

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at arXiv:1807.05803.

Funding *Robert Krauthgamer*: Work supported in part by ONR Award N00014-18-1-2364, Israel Science Foundation grant #1086/18, a Minerva Foundation grant, and a Google Faculty Research Award.
Nikos Parotsidis: The author is supported by Grant Number 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.
Ohad Trabelsi: Work partly done at IBM Almaden Research Center, USA.

Acknowledgements We thank Paweł Gawrychowski, Mohsen Ghaffari, Atri Rudra and Peter Widmayer for the valuable discussions on this problem.

1 Introduction

Connectivity-related problems are some of the most well-studied problems in graph theory and algorithms, and have been thoroughly investigated in the literature. Given a directed graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges,[1] perhaps the most fundamental such problem is to compute a *minimum s-t cut*, i.e., a set of edges E' of minimum-cardinality such that t is not reachable from s in $G \setminus E'$. This minimum s-t cut problem is well-known to be equivalent to maximum s-t flow, as they have the exact same value [17]. Currently, the fastest algorithms for this problem run in time $\tilde{O}(m\sqrt{n}\log^{O(1)} U)$ [24] and $\tilde{O}(m^{10/7}U^{1/7})$ (faster for sparse graphs) [26], where U is the maximum edge capacity (aka weight).[2]

The central problem of study in this paper is All-Pairs Min-Cut (also known as All-Pairs Max-Flow), where the input is a digraph $G = (V, E)$ and the goal is to compute the minimum s-t cut value for all $s, t \in V$. All our graphs will have *unit* edge/vertex capacities, in which case the value of the minimum s-t cut is just the maximum number of disjoint paths from s to t (aka edge/vertex connectivity), by [28]. We will consider a few variants: vertex capacities vs. edge capacities,[3] reporting only the value vs. the cut itself (a witness), or a general digraph vs. a directed acyclic graph (DAG). For all these variants, we will be interested in the *k-bounded* version (aka *bounded min-cuts*, hence the title of the paper) where the algorithm needs to find which minimum s-t cuts have value less than a given parameter k, and report

[1] We sometimes use arcs when referring to directed edges, or use nodes instead of vertices.
[2] The notation $\tilde{O}(\cdot)$ hides polylogarithmic factors.
[3] The folklore reduction where each vertex v is replaced by two vertices connected by an edge $v_{in} \to v_{out}$ shows that in all our problems, vertex capacities are no harder (and perhaps easier) than edge capacities. Notice that this is only true for directed graphs.

only those. Put differently, the goal is to compute, for every $s, t \in V$, the minimum between k and the actual minimum s-t cut value. Nonetheless, some of our results (the lower bounds) are of interest even without this restriction.

The time complexity of these problems should be compared against the fundamental special case that lies at their core – the Transitive Closure problem (aka All-Pairs Reachability), which is known to be time-equivalent to Boolean Matrix Multiplication, and in some sense, to Triangle Detection [34]. This is the case $k = 1$, and it can be solved in time $O(\min\{mn/\log n, n^\omega\})$, where $\omega < 2.38$ is the matrix-multiplication exponent [14, 23, 33, 11]; the latter term is asymptotically better for dense graphs, but it is not *combinatorial*.[4] This time bound is conjectured to be optimal for Transitive Closure, which can be viewed as a conditional lower bound for All-Pairs Min-Cut; but can we achieve this time bound algorithmically, or is All-Pairs Min-Cut a harder problem?

The naive strategy for solving All-Pairs Min-Cut is to execute a minimum s-t cut algorithm $O(n^2)$ times, with total running time $\tilde{O}(n^2 m^{10/7})$ [26] or $\tilde{O}(n^{2.5}m)$ [24]. For not-too-dense graphs, there is a faster randomized algorithm of Cheung, Lau, and Leung [13] that runs in time $O(m^\omega)$. For smaller k, some better bounds are known. First, observe that a minimum s-t cut can be found via k iterations of the Ford-Fulkerson algorithm [17] in time $O(km)$, which gives a total bound of $O(n^2mk)$. Another randomized algorithm of [13] runs in better time $O(mnk^{\omega-1})$ but it works only in DAGs. Notice that the latter bound matches the running time of Transitive Closure if the graphs are sparse enough. For the case $k = 2$, Georgiadis et al. [18] achieved the same running time as Transitive Closure up to sub-polynomial factor $n^{o(1)}$ in all settings, by devising two deterministic algorithms, whose running times are $\tilde{O}(mn)$ and $\tilde{O}(n^\omega)$.

Other than the lower bound from Transitive Closure, the main previously known result is from [21], which showed that under the Strong Exponential Time Hypothesis (SETH),[5] All-Pairs Min-Cut requires, up to sub-polynomial factors, time $\Omega(mn)$ in unit edge capacitated digraphs of any edge density, and even in the simpler case of (unit) vertex capacities and of DAGs. As a function of k their lower bound becomes $\Omega(n^{2-o(1)}k)$ [21]. Combining the two, we have a conditional lower bound of $(n^2k + n^\omega)^{1-o(1)}$.

Related Work. There are many other results related to the edge capacitated All-Pairs Min-Cut problem, let us mention a few. Other than DAGs, the problem has also been considered in the special cases of planar digraphs [6, 22], sparse digraphs and digraphs with bounded treewidth [6].

In *undirected* graphs, the problem was studied extensively following the seminal work of Gomory and Hu [19] in 1961, which introduced a representation of All-Pairs Min-Cuts via a weighted tree, commonly called a Gomory-Hu tree, and further showed how to compute it using $n - 1$ executions of maximum s-t flow. Bhalgat et al. [8] designed an algorithm that computes a Gomory-Hu tree in unit edge capacitated undirected graphs in $\tilde{O}(mn)$ time, and this upper bound was recently improved [4]. The case of bounded min-cuts (small k) in undirected graphs was studied by Hariharan et al. [20], motivated in part by applications in practical scenarios. The fastest running time for this problem is $\tilde{O}(mk)$ [31], achieved by combining results from [20] and [8]. On the negative side, there is an $n^{3-o(1)}$ lower bound for All-Pairs Min-Cut in sparse *capacitated* digraphs [21], and very recently, a similar lower bound was shown for *undirected* graphs with vertex capacities [4].

[4] Combinatorial is an informal term to describe algorithms that do not rely on fast matrix-multiplication algorithms, which are infamous for being impractical. See [1, 3] for further discussions.

[5] These lower bounds hold even under the weaker assumption that the 3-Orthogonal Vectors problem requires $n^{3-o(1)}$ time.

1.1 Our Contribution

The goal of this work is to reduce the gaps in our understanding of the All-Pairs Min-Cut problem (see Table 1 for a list of known and new results). In particular, we are motivated by three high-level questions. First, how large can k be while keeping the time complexity the same as Transitive Closure? Second, could the problem be solved in cubic time (or faster) in all settings? Currently no $\Omega(n^{3+\varepsilon})$ lower bound is known even in the hardest settings of the problem (capacitated, dense, general graphs). And third, can the actual cuts (witnesses) be reported in the same amount of time it takes to only report their values? Some of the previous techniques, such as those of [13], cannot do that.

New Algorithms. Our first result is a randomized algorithm that solves the k-bounded version of All-Pairs Min-Cut in a digraph with *unit vertex capacities* in time $O((nk)^\omega)$. This upper bound is only a factor k^ω away from that of Transitive Closure, and thus matches it up to polynomial factors for any $k = n^{o(1)}$. Moreover, any poly(n)-factor improvement over our upper bound would imply a breakthrough for Transitive Closure (and many other problems). Our algorithm builds on the network-coding method of [13], and in effect adapts this method to the easier setting of vertex capacities, to achieve a better running time than what is known for unit edge capacities. This algorithm is actually more general: Given a digraph $G = (V, E)$ with unit vertex capacities, two subsets $S, T \subseteq V$ and $k > 0$, it computes for all $s \in S, t \in T$ the minimum s-t cut value if this value is less than k, all in time $O((n + (|S| + |T|)k)^\omega + |S||T|k^\omega)$. Due to space constraints, we overview these results in Section 3.1, with full details in the full version.

Three weaknesses of this algorithm and the ones by Cheung et al. [13] are that they do not return the actual cuts, they are randomized, and they are not combinatorial. Our next set of algorithmic results deals with these issues. More specifically, we present two deterministic algorithms for DAGs with unit edge (or vertex) capacities that compute, for every $s, t \in V$, an actual minimum s-t cut if its value is less than k. The first algorithm is *combinatorial* (i.e., it does not involve matrix multiplication) and runs in time $O(2^{O(k^2)} \cdot mn)$. The second algorithm can be faster on dense DAGs and runs in time $O((k \log n)^{4^k + o(k)} \cdot n^\omega)$. These algorithms extend the results of Georgiadis et al. [18], which matched the running time of Transitive Closure up to $n^{o(1)}$ factors, from just $k = 2$ to any $k = o(\sqrt{\log n})$ (in the first case) and $k = o(\log \log n)$ (in the second case). We give an overview of these algorithms in Section 3.2, and the formal results are in the full version.

New Lower Bounds. Finally, we present conditional lower bounds for our problem, the k-bounded version of All-Pairs Min-Cut. As a result, we identify new settings where the problem is harder than Transitive Closure, and provide the first evidence that the problem cannot be solved in cubic time. Technically, the main novelty here is a reduction from the 4-Clique problem. It implies lower bounds that apply to the basic setting of DAGs with unit vertex capacities, and therefore immediately apply also to more general settings, such as edge capacities, capacitated inputs, and general digraphs, and they in fact improve over previous lower bounds [5, 21] in all these settings.[6] We prove the following theorem in Section 4.

▶ **Theorem 1.1.** *If for some fixed $\varepsilon > 0$ and any $k \in [n^{1/2}, n]$, the k-bounded version of All-Pairs Min-Cut can be solved on DAGs with unit vertex capacities in time $O((n^{\omega-1}k^2)^{1-\varepsilon})$, then 4-Clique can be solved in time $O(n^{\omega+1-\delta})$ for some $\delta = \delta(\varepsilon) > 0$.*

[6] It is unclear if our new reduction can be combined with the ideas in [4] to improve the lower bounds in the seemingly easier case of undirected graphs with vertex capacities.

Moreover, if for some fixed $\varepsilon > 0$ and any $k \in [n^{1/2}, n]$ that version of All-Pairs Min-Cut can be solved combinatorially in time $O((n^2k^2)^{1-\varepsilon})$, then 4-Clique can be solved combinatorially in time $O(n^{4-\delta})$ for some $\delta = \delta(\varepsilon) > 0$.

To appreciate the new bounds, consider first the case $k = n$, which is equivalent to not restricting k. The previous lower bound, under SETH, is $n^{3-o(1)}$ and ours is larger by a factor of $n^{\omega-2}$. For combinatorial algorithms, our lower bound is $n^{4-o(1)}$, which is essentially the largest possible lower bound one can prove without a major breakthrough in fine-grained complexity. This is because the naive algorithm for All-Pairs Min-Cuts is to invoke an algorithm for Max-Flow $O(n^2)$ times, hence a lower bound larger than $\Omega(n^4)$ for our problem would imply the first non-trivial lower bound for minimum s-t cut. The latter is perhaps the biggest open question in fine-grained complexity, and in fact many experts believe that near-linear time algorithms for minimum s-t cut do exist, and can even be considered "combinatorial" in the sense that they do not involve the infamous inefficiencies of fast matrix multiplication. If such algorithms for minimum s-t cut do exist, then our lower bound is tight.

Our lower bound shows that as k exceeds $n^{1/2-o(1)}$, the time complexity of k-bounded of All-Pairs Min-Cut exceeds that of Transitive Closure by polynomial factors. The lower bound is super-cubic whenever $k \geq n^{2-\omega/2+\varepsilon}$.

Table 1 Summary of new and known results. Unless mentioned otherwise, all upper and lower bounds hold both for unit edge capacities and for unit vertex capacitities.

Time		Input	Output	Reference
$O(mn), \tilde{O}(n^\omega)$	deterministic	digraphs	cuts, only $k = 2$	[18]
$O(n^2mk)$	deterministic	digraphs	cuts	[17]
$O(m^\omega)$	randomized	digraphs	cut values	[13]
$O(mnk^{\omega-1})$	randomized	DAGs	cut values	[13]
$O((nk)^\omega)$	randomized, vertex capacities	digraphs	cut values	full version
$2^{O(k^2)}mn$	deterministic	DAGs	cuts	full version
$(k \log n)^{4^k+o(k)} \cdot n^\omega$	deterministic	DAGs	cuts	full version
$(mn + n^\omega)^{1-o(1)}$	based on Transitive Closure	DAGs	cut values	
$n^{2-o(1)}k$	based on SETH	DAGs	cut values	[21]
$n^{\omega-1-o(1)}k^2$	based on 4-Clique	DAGs	cut values	Theorem 1.1

2 Preliminaries

We start with some terminology and well-known results on graphs and cuts. Next we will briefly introduce the main algebraic tools that will be used throughout the paper. We note that although we are interested in solving the k-bounded All-Pairs Min-Cut problem, where we wish to find the all-pairs min-cuts of size at most $k-1$, for the sake of using simpler notation we compute the min-cuts of size at most k (instead of less than k) solving this way the $(k+1)$-bounded All-Pairs Min-Cut problem.

Directed graphs. The input of our problem consists of an integer $k \geq 1$ and a *directed graph*, digraph for short, $G = (V, A)$ with $n := |V|$ *vertices* and $m := |A|$ *arcs*. Every arc $a = (u, v) \in A$ consists of a tail $u \in V$ and a head $v \in V$. By $G[S]$, we denote the subgraph of G *induced* by the set of vertices S, formally $G[S] = (S, A \cap (S \times S))$. By $N^+(v)$, we denote

the *out-neighborhood* of v consisting of all the heads of the arcs leaving v. We denote by $\text{outdeg}(v)$ the number of outgoing arcs from v. All our results extend to multi-digraphs, where each pair of vertices can be connected with multiple (*parallel*) arcs. For parallel arcs, we always refer to each arc individually, as if each arc had a unique identifier. So whenever we refer to a set of arcs, we refer to the set of their unique identifiers, i.e., without collapsing parallel arcs, like in a multi-set.

Flows and cuts. We follow the notation used by Ford and Fulkerson [17]. Let $G = (V, A)$ be a digraph, where each arc a has a nonnegative capacity $c(a)$. For a pair of vertices s and t, an s-t flow of G is a function f on A such that $0 \leq f(a) \leq c(a)$, and for every vertex $v \neq s, t$ the incoming flow is equal to outgoing flow, i.e., $\sum_{(u,v)\in A} f(u, v) = \sum_{(v,u)\in A} f(v, u)$. If G has vertex capacities as well, then f must also satisfy $\sum_{(u,v)\in A} f(u, v) \leq c(v)$ for every $v \neq s, t$, where $c(v)$ is the capacity of v. The value of the flow is defined as $|f| = \sum_{(s,v)\in A} f(s, v)$. We denote the existence of a path from s to t by $s \rightsquigarrow t$ and by $s \not\rightsquigarrow t$ the lack of such a path. Any set $M \subseteq A$ is an *s-t-cut* if $s \not\rightsquigarrow t$ in $G \setminus M$. M is a *minimal* s-t-cut if no proper subset of M is s-t-cut. For an s-t-cut M, we say that its *source side* is $S_M = \{x \mid s \rightsquigarrow x \text{ in } G \setminus M\}$ and its *target side* is $T_M = \{x \mid x \rightsquigarrow t \text{ in } G \setminus M\}$. We also refer to the source side and the target side as *s-reachable* and *t-reaching*, respectively. An s-t k-cut is a minimal cut of size k. A set $\mathcal{M}$ of s-t cuts of size at most k is called a set of s-t $\leq k$-cuts. We can define vertex cuts analogously.

Order of cuts. An s-t cut M is *later* (respectively *earlier*) than an s-t cut M' if and only if $T_M \subseteq T_{M'}$ (resp. $S_M \subseteq S_{M'}$), and we denote it $M \geq M'$ (resp. $M \leq M'$). Note that those relations are not necessarily complementary if the cuts are not minimal (see Figure 1 for an example).

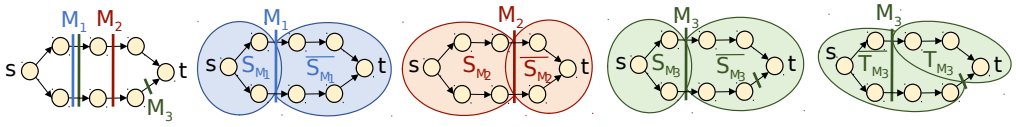

Figure 1 A digraph with three s-t-cuts M_1, M_2, M_3. While M_1 and M_2 are minimal, M_3 is not. Hence, the source side and target side differ only for M_3. This illustrates that the earlier and later orders might not be symmetric for non-minimal cuts. We have $M_3 < M_2$ yet $M_2 \not> M_3$ (and also $M_3 \leq M_2$ yet $M_2 \not\geq M_3$). Additionally, $M_1 \not< M_3$ yet $M_3 > M_1$ (yet both $M_1 \leq M_3$ and $M_3 \geq M_1$).

We make these inequalities strict (i.e., '$>$' or '$<$') whenever the inclusions are proper. We compare a cut M and an arc a by defining $a > M$ whenever both endpoints of a are in T_M. Additionally, $a \geq M$ includes the case where $a \in M$. Definitions of the relations '$\leq$' and '$<$' follow by symmetry. We refer to Figure 2 for illustrations.

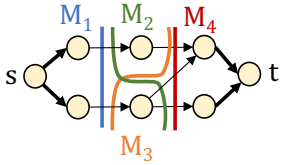

Figure 2 A digraph with several s-t cuts. Bold arcs represent parallel arcs which are too expensive to cut. M_1 is the earliest s-t min-cut and M_3 is the latest s-t min-cut. M_2 is later than M_1, but M_2 is not s-t-latest, as M_4 is later and not larger than M_2.

This partial order of cuts also allows us to define cuts that are extremal with respect to all other s-t cuts in the following sense:

▶ **Definition 2.1** (s-t-latest cuts [27]). *An s-t cut is s-t-*latest *(resp. s-t-*earliest*) if and only if there is no later (resp. earlier) s-t cut of smaller or equal size.*

Informally speaking, a cut is s-t-latest if we would have to cut through more arcs whenever we would like to cut off fewer vertices. This naturally extends the definition of an s-t-latest *min*-cut as used by Ford and Fulkerson [17, Section 5]. The notion of latest cuts has first been introduced by Marx [27] (under the name of *important* cuts) in the context of fixed-parameter tractable algorithms for multi(way) cut problems, but has been independently studied (or its equivalent formulations), [7, 29]. Since we need both earliest and latest cuts, we do not refer to latest cuts as important cuts. Additionally, we use the term *s-t-extremal* cuts to refer to the union of s-t-earliest and s-t-latest cuts.

3 Overview of Our Algorithmic Approach

3.1 Randomized Algorithms on General Graphs

We will now briefly recap the framework of Cheung et al. [13] as we will modify them later for our purposes. In this framework, edges are encoded as vectors, so that the vector of each edge $e = (u, v)$ is a randomized linear combination of the vectors correspond to edges incoming to u, the source of e. One can compute all these vectors for the whole graph, simultaneously, using some matrix manipulations. The bottleneck is that one has to invert a certain $m \times m$ matrix with an entry for each pair of edges. Just reading the matrix that is output by the inversion requires $\Omega(m^2)$ time, since most entries in the inverted matrix are expected to be nonzero even if the graph is sparse.

To overcome this barrier, while using the same framework, we define the encoding vectors on the nodes rather than the edges. We show that this is sufficient for the vertex-capacitated setting. Then, instead of inverting a large matrix, we need to compute the rank of certain submatrices which becomes the new bottleneck. When k is small enough, this turns out to lead to a significant speed up compared to the running time in [13].

3.2 Deterministic Algorithms with Witnesses on DAGs

Here we deal with the problem of computing certificates for the k-bounded All-Pairs Min-Cut problem. Our contribution here is twofold. We first prove some properties of the structure of the s-t-latest k-cuts and of the s-t-latest $\leq k$-cuts, which might be of independent interest. This gives us some crucial insights on the structure of the cuts, and allows us to develop an algorithmic framework which is used to solve the k-bounded All-Pairs Min-Cut problem. As a second contribution, we exploit our new algorithmic framework in two different ways, leading to two new algorithms which run in $O(mn^{1+o(1)})$ time for $k = o(\sqrt{\log n})$ and in $O(n^{\omega+o(1)})$ time for $k = o(\log \log n)$.

Let $G = (V, A)$ be a DAG. Consider some arbitrary pair of vertices s and t, and any s-t-cut M. For every intermediate vertex v, M must be either a s-v-cut, or a v-t-cut. The knowledge of all s-v and all v-t min-cuts does not allow us to convey enough information for computing an s-t min-cut of size at most k quickly, as illustrated in Figure 3.

However, we are able to compute an s-t min-cut by processing all the *s-v-earliest* cuts and all the *v-t-latest* cuts, of size at most k. We build our approach around this insight. We note that the characterization that we develop is particularly useful, as it has been shown that the number of all earliest/latest u-v $\leq k$-cuts can be upper bounded by $2^{O(k)}$, independently of the size of the graph.

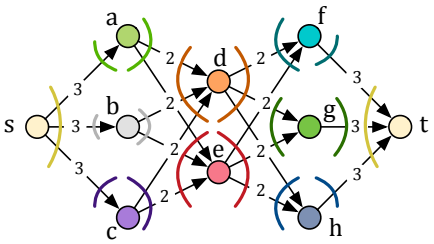

Figure 3 A digraph where each arc appears in at least one s-v or one v-t min-cut. The numbers on the arcs denote the number of parallel arcs. Note that neither of the two s-t min-cuts of size 9 (marked in yellow) are contained within the union of any two s-v or v-t min-cuts. Thus, finding all those min-cuts and trying to combine them in pairs in a divide-and-conquer-style approach is not sufficient to find an s-t min-cut.

For a more precise formulation on how to recover a min-cut (or extremal $\leq k$-cuts) from cuts to and from intermediate vertices, consider the following. Let A_1, A_2 be an *arc split*, that is a partition of the arc set A with the property that any path in G consists of a (possibly empty) sequence of arcs from A_1 followed by a (possibly empty) sequence of arcs from A_2. Assume that for each vertex v we know all the s-v-earliest $\leq k$-cuts in $G_1 = (V, A_1)$ and all the v-t-latest $\leq k$-cuts in $G_2 = (V, A_2)$. We show that a set of arcs M that contains as a subset one s-v-earliest $\leq k$-cut in G_1, or one v-t-latest $\leq k$-cut in G_2 for every v, is a s-t-cut. Moreover, we show that all the s-t-cuts of arcs with the above property include all the s-t-latest $\leq k$-cuts. Hence, in order to identify all s-t-latest $\leq k$ cuts, it is sufficient to identify all sets M with that property. We next describe how we use these structural properties to compute all s-t-extremal $\leq k$-cuts.

We formulate the following combinatorial problem over families of sets, which is independent of graphs and cuts, that we can use to compute all s-t-extremal $\leq k$-cuts. The input to our problem is c families of sets $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_c$, where each family $\mathcal{F}_i$ consists of at most K sets, and each set $F \in \mathcal{F}_i$ contains at most k elements from a universe U. The goal is to compute all minimal subsets $F^* \subset U, |F^*| \leq k$, for which there exists a set $F \in \mathcal{F}_i$ such that $F \subseteq F^*$, for all $1 \leq i \leq c$. We refer to this problem as Witness Superset. To create an instance (s, t, A_1, A_2) of the Witness Superset problem, we set $c = |V|$ and $\mathcal{F}_v$ to be all s-v-earliest $\leq k$-cuts in G_1 and all v-t-latest $\leq k$-cuts in G_2. Informally speaking, the solution to the instance (s, t, A_1, A_2) of the Witness Superset problem picks all sets of arcs that cover at least one earliest or one latest cut for every vertex. In a post-processing step, we filter the solution to the Witness Superset problem on the instance (s, t, A_1, A_2) in order to extract all the s-t-latest $\leq k$-cuts. We follow an analogous process to compute all the s-t-earliest $\leq k$-cuts.

Algorithmic framework. We next define a common algorithmic framework for solving the k-bounded All-Pairs Min-Cut problem, as follows. We pick a partition of the vertices V_1, V_2, such that there is no arc in $V_2 \times V_1$. Such a partition can be trivially computed from a topological order of the input DAG. Let $A_1, A_2, A_{1,2}$ be the sets of arcs in $G[V_1]$, in $G[V_2]$, and $A \cap (V_1 \times V_2)$.

- First, we recursively solve the problem in $G[V_1]$ and in $G[V_2]$. The recursion returns without doing any work whenever the graph is a singleton vertex.

- Second, for each pair of vertices (s,t), such that $s \in V_1$ has an outgoing arc from $A_{1,2}$ and $t \in V_2$, we solve the instance $(s,t,A_{1,2},A_2)$ of Witness Superset. Notice that the only non-empty earliest cuts in $(V,A_{1,2})$ for the pair (x,y) are the arcs $(x,y) \in A_{1,2}$.
- Finally, for each pair of vertices (s,t), such that $s \in V_1, t \in V_2$, we solve the instance $(s,t,A_1,A_{1,2} \cup A_2)$ of Witness Superset.

The Witness Superset problem can be solved naively as follows. Let $\mathcal{F}_v$ be the set of all s-v-earliest $\leq k$-cuts and all v-t-latest $\leq k$-cuts. Assume we have $\mathcal{F}_{v_1}, \mathcal{F}_{v_2}, \ldots, \mathcal{F}_{v_c}$, for all vertices $v_1, v_2, \ldots, v_c$ that are both reachable from s in $(V,A_{1,2})$ and that reach t in (V,A_2). Each of these sets contains $2^{O(k)}$ cuts. We can identify all sets M of arcs that contain at least one cut from each $\mathcal{F}_i$, in time $O(k \cdot (2^{O(k)})^c)$. This yields an algorithm with super-polynomial running time. However, we speed up this naive procedure by applying some judicious pruning, achieving a better running time of $O(c \cdot 2^{O(k^2)} \cdot \text{poly}(k))$, which is polynomial for $k = o(\sqrt{\log n})$. In the following, we sketch the two algorithms that we develop for solving efficiently the k-bounded All-Pairs Min-Cut problem.

Iterative division. For the first algorithm, we process the vertices in reverse topological order. When processing a vertex v, we define $V_1 = \{v\}$ and V_2 to be the set of vertices that appear after v in the topological order. Notice that V_1 has a trivial structure, and we already know all s-t-latest $\leq k$-cuts in $G[V_2]$. In this case, we present an algorithm for solving the instance $(v,t,A_{1,2},A_2)$ of the Witness Superset problem in time $O(2^{O(k^2)} \cdot c \cdot \text{poly}(k))$, where $c = |A_{1,2}|$ is the number of arcs leaving v. We invoke this algorithm for each v-w pair such that $w \in V_2$. For $k = o(\sqrt{\log n})$ this gives an algorithm that runs in time $O(\text{outdeg}(v) \cdot n^{1+o(1)})$ for processing v, and $O(mn^{1+o(1)})$ in total.

Recursive division. For the second algorithm, we recursively partition the set of vertices evenly into sets V_1 and V_2 at each level of the recursion. We first recursively solve the problem in $G[V_1]$ and in $G[V_2]$. Second, we solve the instances $(s,t,A_{1,2},A_2)$ and $(s,t,A_1,A_{1,2} \cup A_2)$ of Witness Superset for all pairs of vertices from $V_1 \times V_2$. Notice that the number of vertices that are both reachable from s in (V,A_1) and reach t in $(V,A_{1,2} \cup A_2)$ can be as high as $O(n)$. This implies that even constructing all $\Theta(n^2)$ instances of the Witness Superset problem, for all s,t, takes $\Omega(n^3)$ time. To overcome this barrier, we take advantage of the power of fast matrix multiplications by applying it into suitably defined matrices of binary codes (codewords). At a very high-level, this approach was used by Fischer and Meyer [16] in their $O(n^\omega)$ time algorithm for transitive closure in DAGs – there the binary codes where of size 1 indicating whether there exists an arc between two vertices.

Algebraic framework. In order to use coordinate-wise boolean matrix multiplication with the entries of the matrices being codewords we first encode all s-t-earliest and all s-t-latest $\leq k$-cuts using binary codes. The bitwise boolean multiplication of such matrices with binary codes in its entries allows a serial combination of both s-v cuts and v-t cuts based on AND operations, and thus allows us to construct a solution based on the OR operation of pairwise AND operations. We show that *superimposed codes* are suitable in our case, i.e., binary codes where sets are represented as bitwise-OR of codewords of objects, and small sets are guaranteed to be encoded uniquely. Superimposed codes provide a unique representation for sets of k elements from a universe of size $\text{poly}(n)$ with codewords of length $\text{poly}(k \log n)$. In this setting, the union of sets translates naturally to bitwise-OR of their codewords.

Tensor product of codes. To achieve our bounds, we compose several identical superimposed codes into a new binary code, so that encoding *set families* with it enables us to solve the corresponding instances of Witness Superset. Our composition has the cost of an exponential increase in the length of the code. Let $\mathcal{F} = F_1, \ldots, F_c$ be the set family that we wish to encode, and let $S_1, \ldots, S_c$ be their superimposed codes in the form of vectors. We construct a c-dimensional array M where $M[i_1, \ldots, i_c] = 1$ iff $S_j[i_j] = 1$, for each $1 \leq j \leq c$. In other words, the resulting code is the tensor product of all superimposed codes. This construction creates enough redundancy so that enough information on the structure of the set families is preserved. Furthermore, we can extract the encoded information from the bitwise-OR of several codewords. The resulting code is of length $O((k \log n)^{O(K)})$, where K is the upperbound on the allowed number of sets in each encoded set family. In our case $K \approx 4^k$, which results to only a logarithmic dependency on n at the price of a doubly-exponential dependency on k, thus making the problem tractable for small values of k.

From slices to Witness Superset. Finally, we show how the Witness Superset can be solved using tensor product of superimposed codes. Consider the notion of cutting the code of dimension K with an axis-parallel hyperplane of dimension $K - 1$. We call this resulting shorter codeword a *slice* of the original codeword. A slice of a tensor product is a tensor product of one dimension less, or an empty set, and a slice of a bitwise-OR of tensor products is as well a bitwise-OR of tensor products (of one dimension less). Thus, taking a slice of the bitwise-OR of the encoding of families of sets is equivalent to removing a particular set from some families and to dropping some other families completely and then encoding these remaining, reduced families. Thus, we can design a non-deterministic algorithm, which at each step of the recursion picks k slices, one slice for each element of the solution we want to output, and then recurses on the bitwise-OR of those slices, reducing the dimension by one in the process. This is always possible, since each element that belongs to a particular solution of Witness Superset satisfies one of the following: it either has a witnessing slice and thus it is preserved in the solution to the recursive call; or it is dense enough in the input so that it is a member of each solution and we can detect this situation from scanning the diagonal of the input codeword. This described nondeterministic approach is then made deterministic by simply considering every possible choice of k slices at each of the K steps of the recursion. This does not increase substantially the complexity of the decoding procedure, since $O(((K \cdot \text{poly}(k \log n))^k)^K)$ for $K \approx 4^k$ is still only doubly-exponential in k.

4 Reducing 4-Clique to All-Pairs Min-Cut

In this section we prove Theorem 1.1 by showing new reductions from the 4-Clique problem to k-bounded All-Pairs Min-Cut with unit vertex capacities. These reductions yield conditional lower bounds that are much higher than previous ones, which are based on SETH, in addition to always producing DAGs. Throughout this section, we will often use the term nodes for vertices.

▶ **Definition 4.1** (The 4-Clique Problem). *Given a 4-partite graph G, where $V(G) = A \cup B \cup C \cup D$ with $|A| = |B| = |C| = |D| = n$, decide whether there are four nodes $a \in A$, $b \in B$, $c \in C$, $d \in D$ that form a clique.*

This problem is equivalent to the standard formulation of 4-Clique (without the restriction to 4-partite graphs). The currently known running times are $O(n^{\omega+1})$ using matrix multiplication [15], and $O(n^4/\operatorname{polylog} n)$ combinatorially [35]. The k-Clique Conjecture

[3] hypothesizes that current clique algorithms are optimal. Usually when the k-Clique Conjecture is used, it is enough to assume that the current algorithms are optimal for every k that is a multiple of 3, where the known running times are $O(n^{\omega k/3})$ [30] and $O(n^k/\operatorname{polylog} n)$ combinatorially [32], see e.g. [2, 3, 10, 12, 25]. However, we will need the stronger assumption that one cannot improve the current algorithms for $k = 4$ by any polynomial factor. This stronger form was previously used by Bringmann, Grønlund, and Larsen [9].

4.1 Reduction to the Unbounded Case

We start with a reduction to the unbounded case (equivalent to $k = n$), that is, we reduce to All-Pairs Min-Cut with unit node capacities (abbreviated APMVC, for All-Pairs Minimum Vertex-Cut). Later (in Section 4.1) we will enhance the construction in order to bound k.

▶ **Lemma 4.2.** *Suppose APMVC on n-node DAGs with unit node capacities can be solved in time $T(n)$. Then 4-Clique on n-node graphs can be solved in time $O(T(n) + MM(n))$, where $MM(n)$ is the time to multiply two matrices from $\{0,1\}^{n\times n}$.*

To illustrate the usage of this lemma, observe that an $O(n^{3.99})$-time combinatorial algorithm for APMVC would imply a combinatorial algorithm with similar running time for 4-Clique.

Proof. Given a 4-partite graph G as input for the 4-Clique problem, the graph H is constructed as follows. The node set of H is the same as G, and we abuse notation and refer also to $V(H)$ as if it is partitioned into A,B,C, and D. Thinking of A as the set of sources and D as the set of sinks, the proof will focus on the number of node-disjoint paths from nodes $a \in A$ to nodes $d \in D$. The edges of H are defined in a more special way, see also Figure 4 for illustration.

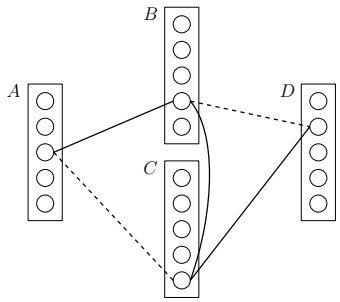

Figure 4 An illustration of H in the reduction. Solid lines between nodes represent the existence of an edge in the input graph G, and dashed lines represent the lack thereof.

- (A to B) For every $a \in A, b \in B$ such that $\{a, b\} \in E(G)$, add to $E(H)$ a directed edge (a, b).
- (B to C) For every $b \in B, c \in C$ such that $\{b, c\} \in E(G)$, add to $E(H)$ a directed edge (b, c).
- (C to D) For every $c \in C, d \in D$ such that $\{c, d\} \in E(G)$, add to $E(H)$ a directed edge (c, d).

The definition of the edges of H will continue shortly. So far, edges in H correspond to edges in G, and there is a (directed) path $a \to b \to c \to d$ if and only if the three (undirected) edges $\{a, b\}, \{b, c\}, \{c, d\}$ exist in G. In the rest of the construction, our goal is to make this 3-hop path contribute to the final $a \to d$ flow *if and only if* (a, b, c, d) is a 4-clique in G (i.e.,

all six edges exist, not only those three). Towards this end, additional edges are introduced, that make this 3-hop path useless in case $\{a, c\}$ or $\{b, d\}$ are not also edges in G. This allows "checking" for five of the six edges in the clique, rather than just three. The sixth edge is easy to "check".

- (A to C) For every $a \in A, c \in C$ such that $\{a, c\} \notin E(G)$, add to $E(H)$ a directed edge (a, c).
- (B to D) For every $b \in B, d \in D$ such that $\{b, d\} \notin E(G)$ in G, add to $E(H)$ a directed edge (b, d).

This completes the construction of H. Note that these additional edges imply that there is a path $a \to b \to d$ in H iff $\{a, b\} \in E(G)$ and $\{b, d\} \notin E(G)$, and similarly, there is a path $a \to c \to d$ in H iff $\{a, c\} \notin E(G)$ and $\{c, d\} \in E(G)$. Let us introduce notations to capture these paths. For nodes $a \in A, d \in D$ denote:

$$B'_{a,d} = \{b \in B \mid \{a, b\} \in E(G) \text{ and } \{b, d\} \notin E(G)\},$$
$$C'_{a,d} = \{c \in C \mid \{a, c\} \notin E(G) \text{ and } \{c, d\} \in E(G)\}.$$

We now argue that if an APMVC algorithm is run on H, enough information is received to be able to solve 4-Clique on G by spending only an additional post-processing stage of $O(n^3)$ time.

▷ Claim 4.3. Let $a \in A, d \in D$ be nodes with $\{a, d\} \in E(G)$. If the edge $\{a, d\}$ does not participate in a 4-clique in G, then the node connectivity from a to d in H, denoted $NC(a, d)$, is exactly

$$NC(a, d) = |B'_{a,d}| + |C'_{a,d}|,$$

and otherwise $NC(a, d)$ is strictly larger.

Proof of Claim 4.3. We start by observing that all paths from a to d in H have either two or three hops.

Assume now that there is a 4-clique (a, b^*, c^*, d) in G, and let us exhibit a set P of node-disjoint paths from a to d of size $|B'_{a,d}| + |C'_{a,d}| + 1$. For all nodes $b \in B'_{a,d}$, add to P the 2-hop path $a \to b \to d$. For all nodes $c \in C'_{a,d}$, add to P the 2-hop path $a \to c \to d$. So far, all these paths are clearly node-disjoint. Then, add the 3-hop path $a \to b^* \to c^* \to d$ to P. This path is node-disjoint from the rest because $b^* \notin B'_{a,d}$ (because $\{b^*, d\} \in E(G)$) and $c^* \notin C'_{a,d}$ (because $\{a, c^*\} \in E(G)$).

Next, assume that no nodes $b \in B, c \in C$ complete a 4-clique with a, d. Then for every set P of node-disjoint paths from a to d, there is a set P' of 2-hop node-disjoint paths from a to d that has the same size. To see this, let $a \to b \to c \to d$ be some 3-hop path in P. Since (a, b, c, d) is not a 4-clique in G and $\{a, d\}, \{a, b\}, \{b, c\}, \{c, d\}$ are edges in G, we conclude that either $\{a, c\} \notin E(G)$ or $\{b, d\} \notin E(G)$. If $\{a, c\} \notin E(G)$ then $a \to c$ is an edge in H and the 3-hop path can be replaced with the 2-hop path $a \to c \to d$ (by skipping b) and one is remained with a set of node-disjoint paths of the same size. Similarly, if $\{b, d\} \notin E(G)$ then $b \to d$ is an edge in H and the 3-hop path can be replaced with the 2-hop path $a \to b \to d$. This can be done for all 3-hop paths and result in P'. Finally, note that the number of 2-hop paths from a to d is exactly $|B'_{a,d}| + |C'_{a,d}|$, and this completes the proof of Claim 4.3. ◁

Computing the estimates. To complete the reduction, observe that the values $|B'_{a,d}|+|C'_{a,d}|$ can be computed for all pairs $a \in A, d \in D$ using two matrix multiplications. To compute the $|B'_{a,d}|$ values, multiply the two matrices M, M' which have entries from $\{0,1\}$, with $M_{a,b} = 1$ iff $\{a,b\} \in E(G) \cap A \times B$ and $M'_{b,d} = 1$ iff $\{b,d\} \notin E(G) \cap B \times D$. Observe that $|B'_{a,d}|$ is exactly $(M \cdot M')_{a,d}$. To compute $|C'_{a,d}|$, multiply M, M' over $\{0,1\}$ where $M_{a,c} = 1$ iff $\{a,c\} \notin E(G) \cap A \times C$ and $M'_{c,d} = 1$ iff $\{c,d\} \in E(G) \cap C \times D$.

After having these estimates and computing APMVC on H, it can be decided whether G contains a 4-clique in $O(n^2)$ time as follows. Go through all edges $\{a,d\} \in E(G) \cap A \times D$ and decide whether the edge participates in a 4-clique by comparing $|B'_{a,d}| + |C'_{a,d}|$ to the node connectivity $NC(a,d)$ in H. By the above claim, an edge $\{a,d\}$ with $NC(a,d) > |B'_{a,d}| + |C'_{a,d}|$ is found if and only if there is a 4-clique in G. The total running time is $O(T(n) + MM(n))$, which completes the proof of Lemma 4.2. ◀

4.2 Reduction to the *k*-Bounded Case

Next, we exploit a certain versatility of the reduction and adapt it to ask only about min-cut values (aka node connectivities) that are smaller than k. In other words, we will reduce to the k-bounded version of All-Pairs Min-Cut with unit node capacities (abbreviated kAPMVC, for k-bounded All-Pairs Minimum Vertex-Cut). Our lower bound improves on the $\Omega(n^\omega)$ conjectured lower bound for Transitive Closure as long as $k = \omega(n^{1/2})$.

▶ **Lemma 4.4.** *Suppose kAPMVC on n-node DAGs with unit node capacities can be solved in time $T(n,k)$. Then 4-Clique on n-node graphs can be solved in time $O(\frac{n^2}{k^2} \cdot T(n,k) + MM(n))$, where $MM(n)$ is the time to multiply two matrices from $\{0,1\}^{n \times n}$.*

Due to space constraints, the proof appears in the full version.

Proof of Theorem 1.1. Assume there is an algorithm that solves kAPMVC in time $O((n^{\omega-1}k^2)^{1-\varepsilon})$. Then by Lemma 4.4 there is an algorithm that solves 4-Clique in time $= O(\frac{n^2}{k^2} \cdot (n^{\omega-1}k^2)^{1-\varepsilon} + MM(n)) \leq O(n^{\omega+1-\varepsilon'})$, for some $\varepsilon' > 0$. The bound for combinatorial algorithms is achieved similarly. ◀

References

1. A. Abboud and V. V. Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *FOCS*, pages 434–443, October 2014. doi:10.1109/FOCS.2014.53.
2. Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-Grained Complexity of Analyzing Compressed Data: Quantifying Improvements over Decompress-and-Solve. In *FOCS*, pages 192–203, 2017. doi:10.1109/FOCS.2017.12.
3. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *FOCS*, pages 98–117, 2015. doi:10.1109/FOCS.2015.16.
4. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs. *CoRR*, abs/1901.01412, 2019. arXiv:1901.01412.
5. 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. doi:10.1137/15M1050987.
6. Srinivasa R Arikati, Shiva Chaudhuri, and Christos D Zaroliagis. All-Pairs Min-Cut in Sparse Networks. *Journal of Algorithms*, 29(1):82–110, 1998. doi:10.1006/jagm.1998.0961.
7. Attila Bernáth and Gyula Pap. Blocking unions of arborescences. *CoRR*, abs/1507.00868, 2015. arXiv:1507.00868.

8 Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. An $\tilde{O}(mn)$ Gomory-Hu Tree Construction Algorithm for Unweighted Graphs. In *STOC*, pages 605–614, 2007. doi:10.1145/1250790.1250879.

9 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A Dichotomy for Regular Expression Membership Testing. In *FOCS*, pages 307–318, 2017. doi:10.1109/FOCS.2017.36.

10 Karl Bringmann and Philip Wellnitz. Clique-Based Lower Bounds for Parsing Tree-Adjoining Grammars. In *CPM*, pages 12:1–12:14, 2017. doi:10.4230/LIPIcs.CPM.2017.12.

11 Timothy M. Chan. All-Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time. *Algorithmica*, 50(2):236–243, 2008. doi:10.1007/s00453-007-9062-1.

12 Yi-Jun Chang. Conditional Lower Bound for RNA Folding Problem. *CoRR*, abs/1511.04731, 2015. arXiv:1511.04731.

13 Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. Graph Connectivities, Network Coding, and Expander Graphs. *SIAM Journal on Computing*, 42(3):733–751, 2013. doi:10.1137/110844970.

14 D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

15 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

16 M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *SWAT*, pages 129–131. IEEE, 1971. doi:10.1109/SWAT.1971.4.

17 Lester Randolph Ford, Jr. and Delbert Ray Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

18 Loukas Georgiadis, Daniel Graf, Giuseppe F. Italiano, Nikos Parotsidis, and Przemysław Uznański. All-Pairs 2-Reachability in $O(n^\omega \log n)$ Time. In *ICALP*, volume 80, pages 74:1–74:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.74.

19 R. E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. doi:10.1137/0109047.

20 Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient Algorithms for Computing All Low *s*-*t* Edge Connectivities and Related Problems. In *SODA*, pages 127–136, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283398.

21 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.

22 Jakub Łącki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Single Source – All Sinks Max Flows in Planar Digraphs. In *FOCS*, pages 599–608. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.66.

23 F. Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *ISSAC*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

24 Y. T. Lee and A. Sidford. Path Finding Methods for Linear Programming: Solving Linear Programs in $\widetilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow. In *FOCS*, pages 424–433, 2014. doi:10.1109/FOCS.2014.52.

25 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *SODA*, pages 1236–1252, 2018.

26 A. Mądry. Computing Maximum Flow with Augmenting Electrical Flows. In *FOCS*, pages 593–602, 2016. doi:10.1109/FOCS.2016.70.

27 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

28 K. Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

29 Hiroshi Nagamochi and Yoko Kamidoi. Minimum cost subpartitions in graphs. *Inf. Process. Lett.*, 102(2-3):79–84, 2007. doi:10.1016/j.ipl.2006.11.011.

30 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

31 Debmalya Panigrahi. Gomory-Hu trees. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 858–861. Springer, 2016. doi:10.1007/978-1-4939-2864-4.

32 Virginia Vassilevska. Efficient algorithms for clique problems. *Inf. Process. Lett.*, 109(4):254–257, 2009. doi:10.1016/j.ipl.2008.10.014.

33 V. Vassilevska Williams. Multiplying Matrices Faster Than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012. doi:10.1145/2213977.2214056.

34 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

35 Huacheng Yu. An improved combinatorial algorithm for Boolean matrix multiplication. *Inf. Comput.*, 261:240–247, 2018. doi:10.1016/j.ic.2018.02.006.