Optimal Vertex Connectivity Oracles

Seth Pettie
pettie@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

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

Longhui Yin
ylh21@mails.tsinghua.edu.cn
Tsinghua University
Haidian, Beijing, China

ABSTRACT

A k-vertex connectivity oracle for undirected G is a data structure that, given $u, v \in V(G)$, reports $\min\{k, \kappa(u,v)\}$, where $\kappa(u,v)$ is the pairwise vertex connectivity between u, v. There are three main measures of efficiency: construction time, query time, and space. Prior work of Izsak and Nutov [Inf. Process. Lett. 2012] shows that a data structure of total size $O(kn \log n)$, which can even be encoded as a $O(k \log^3 n)$-bit labeling scheme, can answer vertex-connectivity queries in $O(k \log n)$ time. The construction time is polynomial, but unspecified.

In this paper we address the top three complexity measures.

The first is the space consumption. We prove that any k-vertex connectivity oracle requires $\Omega(kn)$ bits of space. This answers a long-standing question on the structural complexity of vertex connectivity, and gives a strong separation between the complexity of vertex- and edge-connectivity. Both Izsak and Nutov [Inf. Process. Lett. 2012] and the data structure we will present in this work match this lower bound up to polylogarithmic factors.

The second is the query time. We answer queries in $O(\log n)$ time, independent of k, improving on $\Omega(k \log n)$ time of Izsak and Nutov [Inf. Process. Lett. 2012]. The main idea is to build instances of SetIntersection data structures, with additional structure based on affine planes. This structure allows for optimum query time that is linear in the output size (This evades the general $k^{1/2-o(1)}$ and $k^{1-o(1)}$ lower bounds on SetIntersection from the 3SUM or OMv hypotheses, resp. Kopelowitz et al. [SODA 2016] and Henzinger et al. [STOC 2015].)

The third is the construction time. We build the data structure in time of roughly a max-flow computation on a unit-capacity graph, which is $m^{4/3+o(1)}$ using state-of-the-art algorithm by Tarun et al. [FOCS 2020]. Max-flow is a natural barrier for many problems that have an all-pairs-min-cut flavor. The main technical contribution here is a fast algorithm for computing a k-bounded version of a Gomory-Hu tree for element connectivity, a notion that generalizes edge and vertex connectivity.

CCS CONCEPTS

• **Theory of computation → Data structures design and analysis**; **Graph algorithms analysis**.

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3519945

KEYWORDS

graph connectivity, space lower bounds, data structures

ACM Reference Format:
Seth Pettie, Thatchaphol Saranurak, and Longhui Yin. 2022. Optimal Vertex Connectivity Oracles. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3519935.3519945

1 INTRODUCTION

Most measures of *graph connectivity* can be computed in polynomial time, and much of the recent work in graph *algorithms* aims at reducing these complexities to some natural barrier, either near-linear [26, 29, 30, 43, 44, 54, 60, 63], max-flow-time [49, 50], or matrix-multiplication-time [1, 13]. Graph connectivity has also been extensively studied from a *structural* perspective, where the aims are to understand the structure of some/all minimum cuts. This genre includes Gomory-Hu trees [33], the cactus representation [17], block-trees, SPQR-trees, and extensions to higher vertex connectivity [4, 15, 42, 58], and many others [6, 7, 18–23, 25, 31, 32, 35, 59].

In this paper we study the *data structural* approach to understanding graph connectivity, which incorporates elements of the *algorithmic* and *structural* camps, but goes further in that we want to be able to efficiently *query* the connectivity, e.g., either ask for the size of a min-cut or the min-cut itself.

Suppose we are given an undirected graph G and wish to be able to answer pairwise edge- and vertex-connectivities up to some threshold k. Let $\lambda(u,v)$ and $\kappa(u,v)$ be the maximum number of edge-disjoint and internally vertex-disjoint paths, resp., between u to v. By Menger's theorem, these are equal to the minimum number of edges and vertices, resp., necessary to disconnect u and v.[1]

e-conn(u,v) : Return $\min\{\lambda(u,v), k\}$.
e-cut(u,v) : If $\lambda(u,v) < k$, return an edge-cut separating u, v with size $\lambda(u,v)$.
v-conn(u,v) : Return $\min\{\kappa(u,v), k\}$.
v-cut(u,v) : If $\kappa(u,v) < k$, return a vertex-cut separating u, v with size $\kappa(u,v)$.

The edge-connectivity problems are close to being solved. *Gomory-Hu tree* T [33] (aka *cut equivalent tree*) is an edge-weighted tree such that the bottleneck edge e between any u, v has weight $\lambda(u,v)$, and the partition defined by $T - \{e\}$ corresponds to a $\lambda(u,v)$-edge cut, which can be explicitly associated with e if we wish to also answer e-cut queries. Bottleneck queries can be answered in $O(1)$ time with $O(n \log n)$ preprocessing and $O(n)$ space, or $O(\alpha(n))$

[1] If $\{u,v\} \in E$, then $\kappa(u,v)$ represents the size of a *mixed cut* separating u, v consisting of $\{u,v\}$ and $\kappa(u,v) - 1$ other vertices.

time with $O(n)$ preprocessing and $O(n)$ space; see [9, 16, 57].[2] The time to compute the full Gomory-Hu tree is $\tilde{O}(m^{3/2}n^{1/6})$ [2] and there are faster $\tilde{O}(n^2)$-time algorithms on simple graphs [3, 52] or $\tilde{O}(m + nk^3)$-time algorithms [36] for representing connectivity up to k.

Thus, e-conn-queries can be answered in $O(1)$ time by a data structure occupying space $O(n)$ It is a long-standing open problem whether a similar result is possible for v-conn-queries. In 1979 Schnorr [61] proposed a cut-equivalent tree for *roundtrip* flow[3] in directed graphs, and in 1990 Gusfield and Naor [34] used Schnorr's result to build a Gomory-Hu-type tree for vertex connectivity. Benczur [5] found errors in Schnorr [61] and Gusfield and Naor [34], and proved more generally that there is no cut-equivalent tree for vertex connectivity. Benczur [5, pp. 505-506] suggested a way to get a *flow*-equivalent tree for roundtrip flow using a result of Cheng and Hu [12], which would yield a Gomory-Hu-type tree suitable for answering v-conn (but not v-cut) queries. This too, turned out to be incorrect. Hassin and Levin [37] gave an example of a vertex-capacitated graph (integer capacities in $[1, n^{O(1)}]$) that has $\Omega(n^2)$ distinct pairwise vertex connectivities, which cannot be captured by a Gomory-Hu-type tree representation.[4]

When the underlying graph has *unit capacity*, the counterexamples of [5, 37] do not rule out a representation of vertex-connectivity using, say, $\tilde{O}(1)$ trees, nor do they rule out some completely different $\tilde{O}(n)$-space structure for answering v-conn-queries, independent of k.

Most prior data structures supporting v-conn queries were actually *labeling schemes*.

I.e., a vertex *labeling* $\ell : V \to \{0,1\}^*$ is created such that a v-conn(u,v) query is answered by inspecting $\ell(u), \ell(v)$. Katz, Katz, Korman, and Peleg [46] initiated this line of research into labeling for connectivity. They proved that the maximum label length to answer v-conn queries is $\Omega(k \log(n/k^3))$ and $O(2^k \log n)$. To be specific, they give a class of graphs for which a $\Theta(1/k^2)$-fraction of the vertices require $\Omega(k \log(n/k^3))$-bit labels. However, this does not imply any non-trivial bound on the average/total label length. Their upper bound was subsequently improved to $O(k^3 \log n)$ [48] and then to $O(k^2 \log n)$ [39].[5] Using a different approach, Izsak and Nutov [40] gave an $O(k \log^3 n)$-bit labeling scheme for v-conn queries. A centralized version of the data structure takes $O(kn \log n)$ total space[6] and can be augmented to support v-cut queries with $O(k^2 n \log n)$ total space.

The labeling schemes [39, 40, 46, 48] focus on label-length, and do not discuss the construction time in detail, which is a large polynomial.

1.1 New Results

We resolve the long-standing question concerning the space complexity of representing vertex connectivity; cf. [5, 37]. In particular, we prove that any data structure answering v-conn queries requires space $\Omega(kn/\log n)$ (i.e., $\Omega(kn)$ *bits*), and that the lower bound extends to threshold queries (decide whether $\kappa(u,v) \le k$), equality queries (decide whether $\kappa(u,v) = k$), and approximate queries (distinguish $\kappa(u,v) < k - \sqrt{k}$ from $\kappa(u,v) > k + \sqrt{k}$). This implies that Izsak and Nutov's [40] centralized data structures are *optimal* to within a $\log^2 n$-factor, and that even the *average* length of their labeling scheme cannot be improved by more than a $\log^3 n$-factor; cf. [46]. It also implies a strong separation between the space complexity of storing all edge-connectivities ($O(n)$ space) and all vertex-connectivities ($\Omega(n^2/\log n)$ space when $k = n$).

Although the Izsak-Nutov structure is space-optimal, its query time of $O(k \log n)$ and polynomial construction time can be substantially improved. We design a version of this structure that allows for $O(\log n)$ query time independent of k. The key problem is to create random instances of SetIntersection that are still structured enough to answer intersection queries $S_i \cap S_j$ optimally, in time $O(|S_i \cap S_j|)$. Conditional lower bounds from 3SUM and OMv [38, 47] imply that this should be impossible for worst-case instances of SetIntersection.

Turning to construction time, we prove that the vertex connectivity oracle (or labeling scheme) can be constructed in time $T_{\text{uflow}}(m) \cdot \text{poly}(k, \log n)$, where $T_{\text{uflow}}(m)$ is the time for one max-flow computation on unit-capacity graphs with m edges. The main subproblem solved in the construction algorithm is computing a Gomory-Hu tree for *element connectivities* up to k.

1.2 Organization

Section 2 covers some preliminary concepts such as vertex connectivity, element connectivity, Gomory-Hu trees, and the (vertex-cut version of) the isolating cuts lemma of [49, 50].

Section 3 presents the lower bound on representations of vertex connectivity. Section 4 presents a space- and query-efficient vertex connectivity oracle based on Izsak and Nutov's [40] labeling scheme. In order to build this oracle efficiently, we need to compute Gomory-Hu trees that capture all element connectivities up to k. In Section 5 we show how to do this in $T_{\text{uflow}}(m)\text{poly}(k, \log n)$ time.

Section 6 concludes with some inspiring open problems.

2 PRELIMINARIES

2.1 Vertex Connectivity, Element Connectivity, and Gomory-Hu Trees

Define $\kappa_G(u,v)$ to be the vertex connectivity of u, v in $G = (V, E)$, i.e., the maximum number of internally vertex-disjoint paths between u and v. By Menger's theorem [53], this is the size of the

[2] These upper and lower bounds are in the comparison model. When G is unweighted, all min-cut values are integers in $[1, n)$, which can be sorted in linear time. In this case $O(n)$ preprocesssing for $O(1)$-time queries is possible.

[3] $\min\{f(u,v), f(v,u)\}$

[4] Hassin and Levin [37] pointed out that Benczur's proposal yields a Gomory-Hu-type tree representation for vertex "*separations*" in a capacitated graph $G = (V, E, c : V \to \mathbb{R}^+)$. The minimum separation of u, v is the minimum of $c(u), c(v)$, and the minimum $c(K)$ over all vertex cuts K disconnecting u, v.

[5] The labeling schemes of Katz et al. [46], Korman [48], and Hsu and Lu [39] differentiate between $\kappa(u,v) \ge k_0$ and $\kappa(u,v) < k_0$ using labels of size $O(2^{k_0} \log n)$, $O(k_0^2 \log n)$, and $O(k_0 \log n)$, respectively. They can be used to answer v-conn queries by concatenating labels for all $k_0 = 1, 2, \ldots, k$, thereby introducing an $O(k)$-factor overhead in [39, 48].

[6] Following convention, the space of centralized data structures is measured in $O(\log n)$-bit words and labeling schemes are measured in bits.

Table 1: A history of vertex-connectivity oracles. By convention, space for centralized data structures is measured in $O(\log n)$-bit words and space for labeling schemes is measured in bits. With a couple exceptions, all data structures and lower bounds are for v-conn$(u,v) = \min\{\kappa(u,v), k\}$ queries, where k is an arbitrary parameter. The constructions based on block trees, SPQR trees [4], and [42] only work for $k \in \{2,3,4\}$, and Pettie and Yin [58] only works when $k = \kappa(G)+1$ where $\kappa(G)$ is the global minimum vertex connectivity of G. Nutov's [56] data structure ($\star$) determines whether $\kappa(u,v) \le k$ and if so, returns a pointer to a cut of size at most k in $O(1)$ time. The $\Omega(k\log(n/k^3))$ lower bound of Katz et al. [46] is for the worst-case (longest) label length; it implies nothing on average length. The new $\Omega(nk)$-bit lower bound is for the total size of the data structure, and implies an $\Omega(k)$-bit lower bound on the average length of any labeling scheme.

Citation		Space (Words)	Labeling (Bits)	Query Time	Construction Time
Block-tree	$k=2$	$O(n)$		$O(1)$	$O(m)$
Block-tree + SPQR-tree [4]	$k=3$	$O(n)$		$O(1)$	$O(m)$
Block-tree + SPQR-tree + [42]	$k=4$	$O(n)$		$O(1)$	$O(m+n\alpha(n))$
[46]		$O(n2^k)$	$O(2^k \log n)$	$O(2^k)$	poly(n)
			$\Omega(k\log(n/k^3))$	*any*	*any*
[48]		$O(nk^3)$	$O(k^3\log n)$	$O(k\log k)$	poly(n)
[39]		$O(nk^2)$	$O(k^2\log n)$	$O(\log k)$	poly(n)
[40]		$O(nk\log n)$	$O(k\log^3 n)$	$O(k\log n)$	poly(n)
[1]		$O(n^2)$	$O(n\log n)$	$O(1)$	$O(nk)^\omega$
[58]	$k=\kappa(G)+1$	$O(n\kappa(G))$	$O(\kappa(G)\log n)$	$O(1)$	$\tilde{O}(m+n\cdot\text{poly}(\kappa(G)))$
[56]	($\star$)	$O(nk^2+n^2)$		$O(1)$	poly(n)
New		$O(nk\log n)$	$O(k\log^3 n)$	$O(\log n)$	$T_{\text{uflow}}(m)\cdot\text{poly}(k,\log n)$
		$\Omega(nk/\log n)$	$\Omega(k)$ avg.	*any*	*any*

smallest mixed cut $C \subset (E \cup (V-\{u,v\}))$ whose removal disconnects u,v.[7] *Element connectivity* is a useful generalization of vertex- and edge-connectivity [10, 11, 14, 24, 27, 41]. Let $U \subseteq V$ be a set of terminals. Then for $u,v \in U$, $\kappa'_{G,U}(u,v)$ is the maximum number of paths between u and v that are E-disjoint and $(V-U)$-disjoint.[8] By duality, this is equivalently the size of the smallest mixed cut $C \subset (E \cup (V-U))$ whose removal disconnects u,v. Observe that when $U=V$, $\kappa'_{G,U}(u,v) = \lambda_G(u,v)$ is the same as edge-connectivity, and when $U=\{u,v\}$, $\kappa'_{G,U}(u,v) = \kappa_G(u,v)$ captures vertex connectivity. More generally we have Lemma 2.1, which follows directly from the definitions.

Lemma 2.1. *Fix an undirected graph $G=(V,E)$ and a terminal set $U \subseteq V$. For any $u,v \in U$, $\kappa'_{G,U}(u,v) \ge \kappa_G(u,v)$ with equality if and only if there exists a mixed $\kappa_G(u,v)$-cut C disconnecting u,v with $C \cap U = \emptyset$.*

To shorten our notations, we add a definition:

Definition 2.2. *A terminal set $U \subseteq V$ **captures** u,v if there exists a mixed $\kappa_G(u,v)$-cut $C \subseteq V \cup E$ such that*

$$U \cap (\{u,v\} \cup C) = \{u,v\}.$$

Lemma 2.1 is then rephrased as: $\kappa'_{G,U}(u,v) \ge \kappa_U(u,v)$ with equality if and only if U *captures* u,v.

If C is a mixed cut, the connected components of $G-C$ are called *sides* of C. Note that minimum edge-cuts have two sides but minimum vertex-cuts may have an unbounded number of sides. For any $A \subset V$, ∂A is the set of vertices adjacent to A in G, but not in A. Thus, ∂A is a vertex cut disconnecting A from $V-(A\cup\partial A)$.

It is well known that Gomory-Hu trees exist for element connectivity; see [11, 62]. We use the following general definition:

Definition 2.3. *A k-Gomory-Hu tree for element connectivity w.r.t. graph G and terminal set U is a triple (T,f,C) satisfying*

- *(Flow equivalency) $T=(V_T,E_T,w: E_T \to [1,k))$ is a weighted tree and $f: U \to V_T$ satisfies that, if $f(u)=f(v)$ then $\kappa'_{G,U}(u,v) \ge k$; and otherwise, $\kappa'_{G,U}(u,v) = \min_{e\in T(f(u),f(v))} w(e)$, where $T(x,y)$ is the unique T-path between x and y.*
- *(Cut equivalency) $C: E_T \to 2^{E\cup(V-U)}$. For any edge $e \in T(f(u),f(v)), C(e)$ is a $w(e)$-cut disconnecting u,v in G. (Each of the two connected components in $T-e$ represents the union of some subset of the sides of $C(e)$.)*

Note that f is unnecessary if $k=\infty$ as in this case $V_T=U$, and that C is unnecessary if we are only interested in $\min\{\kappa'_{G,U}(u,v),k\}$-queries. This definition can be extended to $(1+\epsilon)$-approximations by relaxing the first criterion (flow equivalency) to $\kappa'_{G,U}(u,v) \le \min_{e\in T(f(u),f(v))} w(e) \le (1+\epsilon)\kappa'_{G,U}(u,v)$, with trivial f ($V_T=U$ and f maps trivially), and the second criterion similarly.

2.2 Isolating Cut Lemma and Max-Flows

Li and Panigrahi's [50] *isolating cuts* lemma finds, for a terminal set I, the minimum edge-cut separating $u \in I$ from $I-\{u\}$ in time proportional to $O(\log|I|)$ max-flows. It was generalized to vertex connectivity in [49] and here we generalize this lemma slightly further to be able to handle *element* connectivity.

[7] Sometimes only "pure" vertex cuts $C \subset V-\{u,v\}$ are considered and $\kappa_G(u,v)$ is left undefined or artificially defined to be $|V|-1$ whenever u,v are adjacent in G. This definition fails to distinguish highly connected and poorly connected pairs of vertices that happen to be adjacent. Inserting a vertex into every edge can be a trivial reduction that transforms "mixed cuts" into "pure cuts", but this increases the number of vertices to $\Omega(m)$, so only used when the time bound is based to m. This explains why we define $\kappa_G(u,v)$ in this way.

[8] Namely, these paths does not intersect at edges in E, nor vertices in $(V-U)$.

Lemma 2.4 (Isolating cuts, vertex version with forbidden terminals). *Let $I \cup F$ be an independent set in graph $G = (V, E)$, and $I \cap F = \emptyset$. We want to find, for each $v \in I$, a set $S_v \subset V$ such that (i) $S_v \cap I = \{v\}$ and $(\partial S_v) \cap F = \emptyset$, (ii) it minimizes $|\partial S_v|$, (iii) subject to (ii), it minimizes $|S_v|$. Then $\{S_v\}_{v \in I}$ can be computed with $\log |I|$ calls to max-flow on graphs with $O(m)$ edges, $O(m)$ vertices, and capacities in $\{1, \infty\}$. If we do not care to minimize $|\partial S_v|$ if it is larger than k, then the max-flow instances can be unit-capacity multigraphs with $O(m + k|F|)$ edges and vertices.*

Proof. (sketch) The proof follows [49, Lemma 4.2]. The authors called max-flow instances to compute the isolating cuts. Our lemma adds a new requirement, that $\partial S_v \cap F = \emptyset$. This can be effected by replacing $v \in F$ in the flow networks with two vertices and an edge $v_{\text{in}} \to v_{\text{out}}$ with capacity ∞. (Or in the unit-capacity case, with $k+1$ parallel edges from v_{in} to v_{out}.) This ensures vertices in F never appear in the cuts ∂S_v we are interested in. □

3 SPACE LOWER BOUND ON VERTEX CONNECTIVITY ORACLES

Informal strategy and intuitions. We use an error-correcting-code-type argument to derive an $\Omega(n^2)$-bit lower bound in the case that $k = n$. By taking the product of n/k copies of this construction on $\Theta(k)$-vertex graphs, we derive the $\Omega(kn)$-bit lower bound for general $k \in [1, n]$. The idea is to show the existence of a *codebook* $\mathcal{T}$ of $n \times n$ Boolean matrices with the following property. Each $T \in \mathcal{T}$ can be represented by a certain graph $G[T]$ on $O(n)$ vertices. Supposing there exists a b-bit vertex connectivity oracle for this graph, we query $\kappa_{G[T]}(u, v)$ for all pairs u, v. From these values we reconstruct a different Boolean matrix $\tilde{T} \neq T$, which is within the decoding radius[9] of $\mathcal{T}$, and therefore lets us deduce the identity of the original matrix T. In other words, it must be that $b \geq \log_2 |\mathcal{T}|$. A key technical idea is to show that each $T \in \mathcal{T}$ in the codebook can, in a certain sense, be *approximately* factored as the product of two rectangular Boolean matrices, with addition and multiplication over $\mathbb{Z}$.

The specifics of the construction are detailed in Theorem 3.1.

Theorem 3.1. *There exists a constant $c \geq 5$ and a subset $\mathcal{T} \subseteq \{0,1\}^{n \times n}$ of boolean matrices having the following properties.*

1. *(Code Properties) $|\mathcal{T}| = 2^{n^2/3}$, each row of each $T \in \mathcal{T}$ has Hamming weight exactly $n/2$, and every two $T, T' \in \mathcal{T}$ has Hamming distance at least $n^2/3$.*
2. *(Matrix Decomposition) For every $T \in \mathcal{T}$, there exists $A \in \{0,1\}^{n \times cn}, B \in \{0,1\}^{cn \times n}$ such that $C = AB$ encodes T in the following way. Let $\hat{T} \in \{0,1\}^{n \times n}$ be such that*
$$\hat{T}(i,j) = \begin{cases} 0 & \text{if } C(i,j) < 2n \\ 1 & \text{if } C(i,j) \geq 2n \end{cases}$$
Then $T, \hat{T}$ have Hamming distance less than $n^2/6$. Moreover, $C(i,j) \leq 2.1n$ for all i, j.
3. *(Vertex Connectivity) Let A, B be the matrices corresponding to T from Part 2. Let $G[T] = (X \cup Y \cup Z, E)$ be an undirected tripartite graph with $|X| = |Z| = n$ and $|Y| = cn$, where $E \cap (X \times Y)$ encodes A and $E \cap (Y \times Z)$ encodes B.*
Let $\tilde{T}$ be such that
$$\tilde{T}(i,j) = \begin{cases} 0 & \text{if } \kappa(x_i, z_j) < 4n - 2 \\ 1 & \text{if } \kappa(x_i, z_j) \geq 4n - 2 \end{cases}$$
Then $\hat{T} = \tilde{T}$ so $T, \tilde{T}$ have Hamming distance less than $n^2/6$.

Proof. For Part 1, a simple probabilistic construction shows the existence of $\mathcal{T}$. Pick a random codebook $\mathcal{T}_0$ containing $(1 + o(1))2^{n^2/3}$ matrices and discard a negligible fraction of codewords that are within distance $(2/5)n^2$ of another codeword. Obtain $\mathcal{T}$ from $\mathcal{T}_0$ as follows. Take each $T_0 \in \mathcal{T}_0$ and see if it is within Hamming distance $O(n^{3/2})$ of a matrix T, all of whose rows have Hamming weight $n/2$. If so, include $T \in \mathcal{T}$ as a representative of T_0. Only a negligible fraction of $\mathcal{T}_0$ matrices fail to satisfy this property. Thus, $\mathcal{T}$ has the requisite size and each pair of matrices in $\mathcal{T}$ has Hamming distance at least $(2/5)n^2 - O(n^{3/2}) > n^2/3$.

Turning to Part 2, the construction of A, B is probabilistic. We pick B uniformly at random such that each row has Hamming weight $n/2$, then choose A depending on B. We call (i, k) *eligible* if the vector $B(k, \cdot)$ has an unusually high agreement with $T(i, \cdot)$, in particular, if
$$|\{j : B(k,j) = T(i,j)\}| \geq n/2 + \sqrt{n}.$$
For each row i, we set $A(i, k) = 1$ for the first $4n$ values of k for which (i, k) is eligible. Since the probability of (i, k) being eligible is some constant p, it is possible to build A with high probability if c is sufficiently large, say $4p^{-1} + 1$.

Suppose $T(i, j) = 1$ and consider the random variable $C(i, j)$. By the definition of eligibility and the fact that rows of B and T have Hamming weight $n/2$,
$$\Pr(B(k,j) = 1 \mid (i,k) \text{ eligible}) \geq \frac{1/2(n/2 + \sqrt{n})}{n/2} = \frac{1}{2} + \frac{1}{\sqrt{n}}$$
Thus $C(i, j)$ is a random variable that dominates $\text{Binom}(4n, 1/2 + 1/\sqrt{n})$, hence $\mathbb{E}(C(i,j)) \geq 2n + 4\sqrt{n}$. The case when $T(i, j) = 0$ is symmetric, in which case $\text{Binom}(4n, 1/2 - 1/\sqrt{n})$ dominates $C(i, j)$. By a Chernoff bound, the probability that $\tilde{T}(i, j) \neq T(i, j)$ is the probability that $C(i, j)$ deviates from its expectation by at least $4\sqrt{n}$, which is at most $\exp(-2(4\sqrt{n})^2/4n) = \exp(-8)$. The Hamming distance between $\hat{T}$ and T is therefore at most $n^2 \exp(-8) < n^2/6$ in expectation.

Lastly, by a Chernoff bound, the probability that $C(i, j)$ deviates from $2n$ by a constant factor is exponentially small. In particular, we have $C(i, j) \leq 2.1n$ for all i, j with high probability. This proves the existence of matrices A, B for any $T \in \mathcal{T}$.

Part 3. Observe that $C(i, j)$ reflects the maximum flow from x_i to z_j if $G[T]$ were a unit-capacity network with all edges directed from $X \to Y \to Z$. However, $G[T]$ is undirected, and by Menger's theorem $\kappa(x_i, z_j)$ is the maximum number of internally vertex disjoint paths from x_i, z_j. We consider paths of length 2 (corresponding to $C(i, j)$) and two types of paths of length 4.

For technical reasons we will modify the construction of A as follows. For each row i, set $A(i, k) = 1$ for $r = O(c \log n)$ values of k chosen uniformly at random, then set $A(i, k) = 1$ for $4n - r$ additional values of k for which (i, k) is eligible. This changes the expectation of $C(i, j)$ by $O(c \log n)/\sqrt{n} < 1$ but does not otherwise affect the analysis in Part 2.

[9] We define the half of minimum Hamming distance between two different matrices in $\mathcal{T}$ as the decoding radius of $\mathcal{T}$.

We claim that with high probability, the vertex connectivity of x_i, z_j is exactly

$$\kappa(x_i, z_j) = \min\{C(i,j) + 2(n-1), 4n\}.$$

In particular there are $\kappa(x_i, z_j)$ paths $P = P_0 \cup P_1 \cup P_2$, where $|P_0| = C(i,j)$ are length-2 paths, P_1 are $n-1$ paths internally disjoint from P_0 of the form $x_i \to Y \to X \to Y \to z_j$ and P_2 are up to $n-1$ paths disjoint from $P_0 \cup P_1$ of the form $x_i \to Y \to Z \to Y \to z_j$.

We construct P_1 as follows. Choose $Y_{1,x}$ to be any $n-1$ neighbors of x_i that are not part of P_0 paths, and $Y_{1,z}$ to be any $n-1$ neighbors of z_j that are not part of P_0 paths. Clearly $Y_{1,x} \cap Y_{1,z} = \emptyset$. Let $G_{1,x}, G_{1,z}$ be the subgraphs of $G[T]$ induced by $X - \{x_i\} \cup Y_{1,x}$ and $X - \{x_i\} \cup Y_{1,z}$, respectively. These graphs contain many edges of generated from A, but we only consider the r edges per vertex generated randomly.[10] Since $r = \Theta(c \log n)$ and $|Y| = cn$, vertices in both $G_{1,x}$ and $G_{1,z}$ have degree $\Theta(\log n)$ with high probability. It is well known that such graphs have perfect matchings w.h.p. (see [8] or [28]); call them $M_{1,x}, M_{1,z}$. Together with x_i, z_j, these form $n-1$ paths P_1 internally disjoint from P_0. The construction of paths P_2 is symmetric. Let $Y_{2,x}$ be $\min\{n-1, 4n - |P_0| - |P_1|\}$ neighbors of x_i not already included in P_0, P_1, and $Y_{2,z}$ be $|Y_{2,x}|$ neighbors of z_j not included in P_0, P_1. Note that both $Y_{2,x}$ and $Y_{2,z}$ have size at least $0.9n$ because $|P_0| \le 2.1n$ and $|P_1| \le n$. The graphs $G_{2,x}, G_{2,z}$ induced by $Z - \{z_j\} \cup Y_{2,x}$ and $Z - \{z_j\} \cup Y_{2,z}$ again contain perfect matchings $M_{2,x}, M_{2,z}$ with high probability.[11] Together with x_i and z_j these generate $|Y_{2,x}|$ paths internally disjoint from P_0, P_1.

We conclude that with high probability, $C(i,j) < 2n$ if and only if $\kappa(x_i, z_j) < 2n + 2(n-1)$ and, from Part 2, that there exist A, B such that $\tilde{T}, T$ have Hamming distance less than $n^2/6$. □

Corollary 3.2. *Suppose $\mathscr{D}(G,k)$ is a data structure for an undirected n-vertex graph G that can determine whether $\kappa(u,v) < k$ or $\kappa(u,v) \ge k$. Then $\mathscr{D}$ requires $\Omega(kn)$ bits of space in the worst case.*

Proof. Pick any $T \in \mathcal{T}$, and let $G[T]$ be the $(c+2)n$-vertex graph encoding of T from Theorem 3.1. Using $\mathscr{D}(G[T], k)$ with $k = 4n-2$, we can generate the matrix $\tilde{T}$, and then deduce T since $\tilde{T}$ is within its decoding radius $n^2/6$. Thus, $\mathscr{D}$ requires at least $\log|\mathcal{T}| = \Theta(n^2)$ bits of space. For general values of k, we can take $\Theta(n/k)$ disjoint copies of this construction, each having $\Theta(k)$ vertices and requiring $\Theta(k^2)$ space. □

Remark 3.3. The lower bound of Corollary 3.2 also applies to data structures $\mathscr{D}(G,k)$ that distinguish between $\kappa(u,v) = k$ and $\kappa(u,v) \ne k$. For this case we would use the construction of Theorem 3.1 with $k = 4n$ and let $\tilde{T}(i,j) = 1$ iff $\kappa(x_i, z_j) = 4n$.

Corollary 3.4. *Suppose $\mathscr{D}(G,k,\epsilon)$ is a data structure for an undirected n-vertex graph G that can distinguish between $\kappa(u,v) < (1-\epsilon)k$ or $\kappa(u,v) > (1+\epsilon)k$. Then $\mathscr{D}$ requires $\Omega(n\epsilon^{-2})$ bits of space for some ϵ.*

Proof. The construction of Theorem 3.1 and Corollary 3.2 still works when $k = \Theta(n)$ and $\epsilon = \Theta(1/\sqrt{n})$. □

[10] The edges included by the eligibility criterion have subtle dependencies, which makes reasoning about them somewhat tricky. Including truly random edges in A is a technical hack and surely not necessary.

[11] Note that $G_{2,x}, G_{2,z}$ correspond to submatrices of B, which was chosen randomly with density 1/2. Due to the Hamming weight restriction, the entries of B are slightly negatively correlated, which only *improves* the chance of finding perfect matchings.

4 VERTEX CONNECTIVITY ORACLES

Lemma 2.1 says that for any terminal set U, $\kappa'_{G,U}(u,v)$ *never* underestimates the true value $\kappa_G(u,v)$ and achieves the equality $\kappa'_{G,U}(u,v) = \kappa_G(u,v)$ if U captures u, v. This naturally induces a design pattern for vertex connectivity oracles: find some (usually random) terminal sets U_i, construct oracles for κ'_{G,U_i}, and output $\min_{i:\{u,v\}\subseteq U_i} \kappa'_{G,U_i}(u,v)$ to a query for $\kappa_G(u,v)$; if with high probability at least one U_i captures u, v, we succeed. Furthermore, $\kappa'_{G,U}$ can be succinctly represented as a Gomory-Hu tree (Definition 2.3) and queried via bottleneck edge queries [9, 16, 57] in $O(1)$ time. This is the basic idea of how Izsak and Nutov's [40] and our data structure work.

At the first glance it seems a bit paradoxical because, Hassin and Levin [37] have shown pairwise vertex connectivity cannot be captured by a Gomory-Hu-type tree representation, while we are constructing Gomory-Hu trees of element connectivity to answer it, where element connectivity is a generalization of vertex connectivity. However, there is no contradiction, because we maintain a set of carefully designed terminal sets, each captures a subset of pairs, and they together, captures all pairwise vertex connectivity.

Izsak and Nutov's [40] ingenious algorithm proceeds by sampling several terminal sets with probability $1/k$. Each terminal set includes both u and v with probability $1/k^2$ and *avoids* $C_{u,v}$ with constant probability if $|C_{u,v}| \le k$, hence $O(k^2 \log n)$ terminal sets suffice to accurately capture all vertex connectivities up to k with high probability. The space for the centralized data structure is just that of storing $O(k^2 \log n)$ Gomory-Hu trees and data structures for answering bottleneck-edge queries. Each tree is on $O(n/k)$ terminals, for a total of $O(kn \log n)$ space. A query v-conn(u,v) needs to examine all terminal sets containing *both* u and v. This is a classic SetIntersection query. Each of u, v is in $\Theta(k \log n)$ sets, and are jointly in $\Theta(\log n)$ sets. And their intersection requires checking through $\Theta(k \log n)$ sets.

In this section we show how to build appropriate SetIntersection instances such that queries are answered in (optimal) $O(\log n)$ time. (If minimum cuts are associated with edges in the Gomory-Hu tree, a v-cut(u,v) query can be answered in $O(1)$ additional time be returning a pointer to the appropriate cut. This increases the space to $O(k^2 n \log n)$.)

First, we show how to find $O(k^2 \log n)$ terminal sets that capture all pairs in V^2 using a construction based on affine planes and 3-wise independent hash functions.

Lemma 4.1. *There is an algorithm using $O(\log^2 n)$ random bits that generates terminal sets $\mathcal{U}$ with the following properties.*

- *$|\mathcal{U}| = O(k^2 \log n)$ and each $U \in \mathcal{U}$ has $|U| = O(n/k)$.*
- *Given vertices u, v with $\kappa(u,v) \le k$ we can find $O(\log n)$ sets $U_1, \ldots, U_{O(\log n)}$ in $\mathcal{U}$ costing $O(\log n)$ time, such that each, independently, captures u, v with constant probability. As a consequence, w.h.p., $\mathcal{U}$ captures all of V^2.*

Proof. Let p_0 be the first prime larger than $n = |V|$ and p be the first prime larger than $2k$. Let $H = (P, L)$ be a subset of an affine plane, defined as follows. $P = \{u_{i,j} \mid i \in [\lceil p_0/p \rceil], j \in [p]\}$ is a set of points arranged in a rectangular grid and $L = \{\ell_{s,j} \mid s, j \in [p]\}$ is a set of lines, where $\ell_{s,j} = \{u_{t, j+st \bmod p} \mid t \in [\lceil p_0/p \rceil]\}$ is the line with slope s passing through $u_{0,j}$. We pick a hash function

$h : V \to P$ of the form

$$h(x) = u_{\lfloor \bar{h}(x)/p \rfloor, \bar{h}(x) \bmod p},$$

where $\bar{h}(x) = (ax^2 + bx + c) \bmod p_0$ with a, b, c chosen uniformly at random from $[p_0]$, then form the $p^2 = \Theta(k^2)$ terminal sets $\mathcal{U}[h] = \{U_{s,j}\}$ as follows.

$$U_{s,j} = \{v \mid h(v) \in \ell_{s,j}\}.$$

Now fix any two vertices u, v and a $\kappa(u,v)$ cut C with $\kappa(u,v) \le k$. Then $\mathcal{U}[h]$ will capture u, v if (i) $h(u), h(v)$ differ in their 1st coordinates of the rectangular grid P, and, assuming this happens, (ii) $C \cap U_{s,j} = \emptyset$ where $\ell_{s,j}$ is the unique line containing $h(u), h(v)$. The probability of (i) is $1 - p/p_0 = 1 - \Theta(k/n)$ and the probability of (ii) is

$$\begin{aligned}
&\Pr(C \cap U_{s,j} = \emptyset \mid h(u), h(v)) \\
\ge &1 - \sum_{x \in C \cap V} \Pr(h(x) \in \ell_{s,j} \mid h(u), h(v)) \\
\ge &1 - k \lceil p_0/p \rceil / p_0 \ge 1/2,
\end{aligned}$$

where the second last inequality is because $|\ell_{s,j}| \le \lceil p_0/p \rceil$, $|C| \le k$, and h is sampled from a 3-wise independent hash family.

Let $\mathcal{U}$ be the union of $\mathcal{U}[h_1], \ldots, \mathcal{U}[h_{O(\log n)}]$ using independent hash functions $h_1, \ldots, h_{O(\log n)}$. Then, it follows that $|\mathcal{U}| = O(p^2 \log n) = O(k^2 \log n)$.

Preprocessing a table of inverses modulo p, given u, v with $\kappa(u,v) \le k$, we can clearly identify whether points $h_i(u), h_i(v)$ differing in 1st coordinate, and when so, identify $\ell_{s,j}$ (and hence $U_{s,j} \in \mathcal{U}[h_i]$) containing them all in $O(1)$ time. Therefore, we can identify $U_1, \ldots, U_{O(\log n)} \in \mathcal{U}$ in $O(\log n)$ time such that at least one of these terminal sets captures u, v w.h.p.

We now turn to the load balancing condition $|U| = O(n/k)$. Note that if the coefficients of h satisfy $(a, b, c) \ne (0, 0, c)$ then h is at most a 2-to-1 function (as the polynomial defining h has degree 2), meaning that each $U \in \mathcal{U}$ has $|U| < 2\lceil p_0/p \rceil = O(n/k)$. The performance of the algorithm is clearly quite bad when $a = b = 0$ (each terminal set U is either $\emptyset$ or V) so we can remove these hash functions from the hash family and only improve the probability of success. □

Remark 4.2. One way that $\mathcal{U}[h]$ can fail to capture u, v is if $h(u)$ and $h(v)$ agree on their 1st coordinate. This could be rectified by including the lines $l_{\infty,j} = \{u_{i,q} : i \equiv j \pmod p, q \in [p]\}$ for $j \in [p]$, which occupies $O(n/k)$ points.

Now, by using the collection $\mathcal{U}$ of terminal sets with additional structure from Lemma 4.1 instead of the $O(k^2 \log n)$ random terminal sets generated independently as used in [40], we can speed up the query time from $O(k \log n)$ to $O(\log n)$[12]. Below, we state the guarantee of our oracle formally.

Theorem 4.3. *Given an undirected graph $G = (V, E)$ and $k \in [1, n]$, a data structure with size $O(kn \log n)$ can be constructed in $O(m) + T_{\text{uflow}}(nk)\text{poly}(k, \log n) = O(m) + n^{4/3+o(1)}\text{poly}(k)$ time such that $\text{v-conn}(u,v) = \min\{\kappa_G(u,v), k\}$ queries can be answered in $O(\log n)$ time. Using space $O(k^2 n \log n)$, a $\text{v-cut}(u,v)$ query can be answered in $O(1)$ additional time; it returns a pointer to a $\kappa_G(u,v)$-size u-v cut whenever $\kappa_G(u,v) < k$.*

[12] Finding all U_i that contains u, v takes $O(\log n)$ time, and each $\kappa'_{G,U_i}(u,v)$ takes $O(1)$ time, so in total $\min_{i:\{u,v\} \subseteq U_i}(u,v)$ takes $O(\log n)$ time. And each U_i captures u, v with constant probability, so at least one U_i captures u, v, i.e., the output is correct, with high probability.

The claims of Theorem 4.3 concerning construction time are substantiated in Section 5. In the context of our vertex connectivity oracle, note that we can assume that our original graph has $O(nk)$ edges by applying Nagamochi-Ibaraki algorithm [55] with $O(m)$ running time to reduce the number of edges to $O(kn)$. So the construction time is $n^{4/3+o(1)}\text{poly}(k)$ using the max-flow algorithm for unit-capacity graphs by Kathuria, Liu, and Sidford [45].

5 GOMORY-HU TREES FOR ELEMENT CONNECTIVITY

The goal in this section is to prove the following:

Theorem 5.1. *A k-Gomory-Hu tree for element connectivity w.r.t. graph G and terminal set U can be constructed in $\tilde{O}(k \cdot T_{\text{uflow}}(m + k|U|))$ time.*

Note that, given the above theorem, we can indeed conclude Theorem 4.3 because there are $O(k^2 \log n)$ terminal sets in the oracle construction and we just need to build a Gomory-Hu tree for each terminal set by calling Theorem 5.1. As $m = O(nk)$ and $|U| = O(n/k)$ by Lemma 4.1, this takes $T_{\text{uflow}}(m + k|U|) = T_{\text{uflow}}(O(nk))$ for each terminal set.

Obstacles in Adapting Algorithms of [51] for Element Connectivity. The proof of Theorem 5.1 is obtained by adapting the Gomory-Hu tree construction for *edge connectivity* by Li and Panigrahi [51] to work for *element connectivity*. Although we use the same high-level approach, element connectivity introduces some extra complication that we need to deal with.

For example, given an input graph G, all Gomory-Hu tree algorithms for edge connectivity proceed by finding a minimum edge cut (A, B), contract one side, say B, of the cut into a single vertex b, and recurse on the contracted graph denoted by G'. By submodularity of edge cuts, we have that the edge connectivity between any two vertices $a_1, a_2 \in A$ are preserved in G'. This is crucial for the correctness of the whole algorithm.

Unfortunately, the direct analog of this statement fails completely for element connectivity. For example, suppose $p, q \notin B$ are disconnected by an element cut C of graph G. Then in graph G', $C' = \{b\} \cup (C \setminus B)$ becomes an element cut disconnecting p and q. As long as C contains more than one element in B (an edge, or a non-terminal vertex), $|C'| < |C|$, so $\kappa'(p,q)$ decreased.

To bypass this complication, we actually exploit the generality of element connectivity. When we recurse in a contracted graph, the trick is to add the contracted node into a terminal set for element connectivity. That is, the terminal set will change throughout the recursion so that we can preserve element connectivity between vertices inside the subject graph.

In the rest of this section, we formally prove Theorem 5.1.

5.1 The Approximate Element Connectivity Gomory-Hu Tree Algorithm

Instead of proving Theorem 5.1 directly, it is more convenient to prove the following $(1+\epsilon)$-approximation version, which is precisely the element connectivity analog of the result in [51].

Theorem 5.2. *A $(1+\epsilon)$-approximate Gomory-Hu tree for element connectivity w.r.t. graph G and terminal set U can be computed in $\tilde{O}(\epsilon^{-1}T_{\text{flow}}(m))$ time, where $T_{\text{flow}}(m)$ is the time to compute max flow in an m-edge graph.*

Theorem 5.1 almost follows from Theorem 5.2 just by setting $\epsilon = 1/k$. However, there is some minor things to take care of, and we give the formal proof in Section 5.2.

Before we give the proof of Theorem 5.2, observe that as a simplifying assumption, Lemma 2.4 (isolating cuts with forbidden terminals) required that $I \cup F$ be an independent set. We can force any instance to satisfy this property by subdividing all edges in $E \cap (I \cup F)^2$. As a consequence, from now we can assume that all element cuts in the modified graph consist solely of vertices.[13]

5.1.1 Algorithm. The precise algorithm for Theorem 5.2 is described in Algorithm 1. For the reader to better understand, we first briefly explain how the algorithm works, how the input and the output of the algorithm relate to Definition 2.3.

The basic framework is a recursion. For a random set of terminals R^i, we call Lemma 2.4 to compute its isolating cuts S^i_v. We select those v such that $S^i_v \le (1+\epsilon)\lambda$ and $\left|S^i_v \cap U\right| \le |U|/2$ into set R^i_{sm}. Now ∂S^i_v is a $(1+\epsilon)$-approximation to the minimum element cut between S^i_v and $V \setminus (S^i_v \cup \partial S^i_v)$, so we split the problem into subgraphs G_v generated by S^i_v for each $v \in R^i_{\text{sm}}$ and one large part, which we call G_{lg}. We specify the new parameters added in the algorithm.

As mentioned before, to avoid the sub-cases underestimate the original element connectivity, we add a new parameter: the forbidden set F. The vertices in F are terminals counted when computing the connectivity, but queries related to them are not supported. Accordingly, the output T is a $(1+\epsilon)$-approximate Gomory-Hu tree, that represents element connectivity in G with the terminal set $U \cup F$, while the tree nodes only represent vertices from U.[14] For $u, v \in U$, the bottleneck edge weight between $f(u)$ and $f(v)$ on T is $\kappa'_{G,U\cup F}(u,v)$. In the top-level call F is set to $\emptyset$, but may accumulate up to $O(|U|)$ vertices in the recursion.

As will be proved in Lemma 5.3, the element connectivity between vertices inside U_{lg} are preserved exactly in the contracted graph G_{lg}. For the small graphs G_v, by Lemma 5.4, we accept a $(1+\epsilon)$-factor approximation to element connectivity, which accrues to $(1+\epsilon)^{\log|U|}$ as a vertex can only appear in the *small* branch of the recursion $\log|U|$ times.

To link the sub-trees at correct nodes and to compute f, we added a tool function $g : V \to V_T \cup \{\perp\}$ that maps vertices of G into tree nodes of T, or a symbol $\perp$. g indicates at which tree nodes we should link two sub-trees. Moreover, the algorithm ensures that at the end, $g(v) \neq \perp$ for every $v \in U$. From the definition of g, $g(v)$ with $v \in U$ satisfies the properties of the embedding function f in Definition 2.3.

To summarize, to compute $(1+\epsilon)$-approximate Gomory-Hu tree, call Algorithm 1 with desired G, U, ϵ, and $F = \emptyset$. It outputs (T, g, C). We set the embedding $f(v) = g(v)$ for all $v \in U$, and then return (T, f, C), which satisfies Definition 2.3. At this point, every node $t \in T$ are in the range of f, because t is only created in the base case, where g maps a vertex $u \in U$ to t.

Algorithm 1: ApproxElemConnGHTree$(G(V,E), U, F, \epsilon)$

input : The graph $G = (V, E)$, the terminal set U, the forbidden set F, the approximation accuracy ϵ
output: A $(1+\epsilon)$-approximate element-connectivity Gomory-Hu tree (T, g, C)

1 **if** $|U| = 1$ **then** `// The Base Case`
2 Construct T with one node t, $g(u) \leftarrow t$ for all $u \in V(G)$ and C an empty function
3 **return** (T, g, C)
4 **end**
5 Let $\lambda \leftarrow$ the global minimum element connectivity `// See Remark 5.7`
6 Call `CutThresholdStep`$(G, U, F, (1+\epsilon)\lambda)$ and store its output $s, \{R^j_{\text{sm}}, R^j, S^j_v\}$
7 Fix $i \in \{0, 1, \cdots, \lfloor \log|U| \rfloor\}$ that maximizes $\left|\cup_{v\in R^i_{\text{sm}}}(S^i_v \cap U)\right|$.
8 **foreach** $v \in R^i_{\text{sm}}$ **do**
9 Let $G_v \leftarrow$ the graph with $V \setminus (S^i_v \cup \partial S^i_v)$ contracted into a vertex x_v.
10 Let $U_v \leftarrow S^i_v \cap U$.
11 Let $(T_v, g_v, C_v) \leftarrow$ `ApproxElemConnGHTree`$(G_v, U_v, F \cup \{x_v\}, \epsilon)$.
12 **end**
13 Let $G_{\text{lg}} \leftarrow$ the graph G with S^i_v contracted into a vertex y_v for each $v \in R^i_{\text{sm}}$.
14 Let $U_{\text{lg}} \leftarrow U \setminus \cup_{v\in R^i_{\text{sm}}}(S^i_v \cap U)$.
15 Let $(T_{\text{lg}}, g_{\text{lg}}, C_{\text{lg}}) \leftarrow$ `ApproxElemConnGHTree`$(G_{\text{lg}}, U_{\text{lg}}, F \cup \{y_v \mid v \in R^i_{\text{sm}}\}, \epsilon)$.
16 Initialize $T \leftarrow T_{\text{lg}} \cup (\cup_{v\in R^i_{\text{sm}}} T_v)$, and then add edges $(g_v(x_v), g_{\text{lg}}(y_v))$ with weight $\left|\partial S^i_v\right|$. Let g inherit values from g_{lg} or one of the $\{g_v\}$. If the value for v is defined in more than one such function, set $g(v) \leftarrow \perp$. Let C inherit the assignment of C_{lg} and $\{C_v\}$, and for the new edges set $C((g_v(x_v), g_{\text{lg}}(y_v))) \leftarrow \partial S^i_v$.
17 **return** (T, g, C).

5.1.2 Correctness. We prove the results needed to show the correctness of Algorithm 1. Denote $F_{\text{lg}} = F \cup \{y_v | v \in R^i_{\text{sm}}\}$ and $F_v = F \cup \{x_v\}$.

Lemma 5.3. *For any two vertices $p, q \in U_{\text{lg}}$, we have*

$$\kappa'_{G_{\text{lg}}, U_{\text{lg}}\cup F_{\text{lg}}}(p,q) = \kappa'_{G,U\cup F}(p,q).$$

[13] If (x, y) is subdivided into (x, v_{xy}), (v_{xy}, y), then any vertex-only element cut containing v_{xy} in the modified instance contains (x, y) in the original instance, and vice-versa.
[14] I.e., $f : U \to V(T)$ does not embed F in T.

Algorithm 2: CutThresholdStep($G = (V, E), U, F_{in}, W$)

Set $s \leftarrow$ a uniformly random vertex in U.
$R^0 \leftarrow U$.
for *j from* 0 *to* $\lfloor \log |U| \rfloor$ **do**
  Call Lemma 2.4 to compute sets $\{S_v^j : v \in R^j\}$, with $I = R^j$ and $F = (U \setminus R^j) \cup F_{in}$.
  Let $R^j_{\text{sm}} \leftarrow \{v \in R^j \setminus \{s\} : \left|S_v^j \cap U\right| \le |U|/2$ and $\left|\partial S_v^j\right| \le W\}$.
  $R^{j+1} \leftarrow$ sampling each vertex of R^j with probability $\frac{1}{2}$, but s with probability 1.
end
return s and, for each j, $\{R^j_{\text{sm}}, R^j, \{S_v^j\}_{v \in R^j_{\text{sm}}}\}$.

Proof. We first show that $\kappa'_{G_{\text{lg}}, U_{\text{lg}} \cup F_{\text{lg}}}(p, q) \ge \kappa'_{G,U\cup F}(p, q)$. Suppose C is a minimum element cut disconnecting p and q in G_{lg} with terminal set $U_{\text{lg}} \cup F_{\text{lg}}$. Then C does not contain any y_v, so C is still an element cut for p and q in G, and therefore $\kappa'_{G_{\text{lg}}, U_{\text{lg}} \cup F_{\text{lg}}}(p, q) = |C| \ge \kappa'_{G,U\cup F}(p, q)$.

It remains to show that $\kappa'_{G_{\text{lg}}, U_{\text{lg}} \cup F_{\text{lg}}}(p, q) \le \kappa'_{G,U\cup F}(p, q)$. Suppose C is a minimum element cut disconnecting p and q in G with terminal set $U \cup F$, and let A, B be the sides containing p, q, respectively. Every $v \in R^i_{\text{sm}}$ is by definition not in C; let the side of C containing v be D_v.

Without loss of generality we assume v and q are in different sides. For brevity, all the following unspecified element cuts are with respect to G and $U \cup F$. Let $H = D_v \cup A$, we have that ∂H is a minimum element cut between $\{v, p\}$ and $\{q\}$, and $\partial(H \cup S_v^i)$ is an element cut between $\{v, p\}$ and $\{q\}$, so $\left|\partial(H \cup S_v^i)\right| \ge |\partial H|$. Now that ∂S_v^i is also a minimum element cut between $\{v\}$ and $R^i \setminus \{v\}$, and $\partial(S_v^i \cap H)$ is also an element cut between $\{v\}$ and $R^i \setminus \{v\}$, we have $\left|\partial(H \cap S_v^i)\right| \ge \left|\partial S_v^i\right|$. By the submodularity of element-connectivity,

$$|\partial H| + \left|\partial S_v^i\right| \ge \left|\partial(H \cap S_v^i)\right| + \left|\partial(H \cup S_v^i)\right|.$$

Hence this inequality holds with equality, so $\left|\partial(H \cap S_v^i)\right| = \left|\partial S_v^i\right|$. But by Lemma 2.4, S_v^i is also minimum in size among minimum isolating cuts, so $H \cap S_v^i = S_v^i$, and therefore $S_v^i \subseteq H$.

If $D_v = A$, we already have $S_v^i \subseteq D_v$. If $D_v \ne A$, then noticing that in $G - C$, D_v is not connected to A, while S_v^i is connected to v, we know that $S_v^i \cap A = \emptyset$ and $S_v^i \subseteq D_v$.

We have shown that $S_v^i \subseteq D_v$. When contracting S_v^i into a vertex x_v, the cut $C = \partial D_v$ is not affected, so C is still an element cut between p and q in G_{lg} with terminal set $U_{\text{lg}} \cup F_{\text{lg}}$, and $\kappa'_{G_{\text{lg}}, U_{\text{lg}} \cup F_{\text{lg}}}(p, q) \le |C| = \kappa'_{G,U\cup F}(p, q)$. □

Lemma 5.4. *For any two vertices $p, q \in U_v$, we have*

$$\kappa'_{G,U\cup F}(p, q) \le \kappa'_{G_v, U_v \cup F_v}(p, q) \le (1+\epsilon)\kappa'_{G,U\cup F}(p, q).$$

Proof. We first show that $\kappa'_{G_v, U_v \cup F_v}(p, q) \ge \kappa'_{G,U\cup F}(p, q)$. Suppose C is a minimum element cut disconnecting p and q in G_v with terminal set $U_v \cup F_v$. Then C does not contain x_v, so C is still an element cut disconnecting p and q in G, so $\kappa'_{G,U\cup F}(p, q) \le |C| = \kappa'_{G_v, U_v \cup F_v}(p, q)$.

Next, we show that $\kappa'_{G_v, U_v \cup F_v}(p, q) \le (1+\epsilon)\kappa'_{G,U\cup F}(p, q)$. Suppose C is a minimum element cut disconnecting p and q in G with terminal set $U \cup F$ and let A, B be the sides of p, q. Without loss of generality, suppose A does not contain s. The following unspecified element cuts are with respect to G and $U \cup F$.

We have that $\partial(S_v^i \cup A)$ is an element cut disconnecting p and s. Therefore, $\left|\partial(S_v^i \cup A)\right| \ge \lambda$. Furthermore, by the definition of S_v^i, $\left|\partial S_v^i\right| \le (1+\epsilon)\lambda$. Therefore, by the submodularity of element cuts,

$$\begin{aligned}(1+\epsilon)\lambda + |\partial A| &\ge \left|\partial S_v^i\right| + |\partial A| \\ &\ge \left|\partial(S_v^i \cup A)\right| + \left|\partial(S_v^i \cap A)\right| \ge \lambda + \left|\partial S_v^i \cap A\right|.\end{aligned}$$

Now that $S_v^i \cap A$ contains p but not q, $\partial(S_v^i \cap A)$ is an element cut disconnecting p and q, and since it is contained in G_v, it is still an element cut in G_v with terminal set $U_v \cup F_v$, so $\left|\partial(S_v^i \cap A)\right| \ge \kappa'_{G_v, U_v \cup F_v}(p, q)$. And by definition $|\partial A| = \kappa'_{G,U\cup F}(p, q) \ge \lambda$.

Therefore,

$$\kappa'_{G_v, U_v \cup F_v}(p, q) \le \epsilon\lambda + |\partial A| \le (1+\epsilon)\kappa'_{G,U\cup F}(p, q).$$

□

Lemma 5.5. *The assignment $g(v) = \perp$ occurs if and only if v appears in $C(e)$ for some edge. Therefore, g value of vertices in $U \cup F$ never equals $\perp$.*

Proof. From the construction of G_{lg} and G_v, it can be seen that only the vertices in $\partial S_v^i = C((g_v(x_v), g_{\text{lg}}(y_v)))$ are defined twice (or more), so their g-value are set to $\perp$. These are all the vertices such that $g(v) = \perp$. □

Remark 5.6. From Lemma 5.3, Lemma 5.4 and Lemma 5.5, in line 17 of Algorithm 1, the g-value of x_v, y_v are not $\perp$, so linking the sub-trees would be successful. And $g(v)$ for $v \in U$ equals to $\perp$.

Remark 5.7. As stated the algorithm computes the global element connectivity λ. In reality λ is a lower bound on this quantity, which is increased once we are sure it has increased by a $1+\epsilon$ factor. As we show in the proof of Theorem 5.2, with high probability, the global element connectivity increases by a factor of $1 + \epsilon$ every $O(\log^3 n)$ steps taken in the "G_{lg}" branch of the recursion tree. Therefore, what the algorithm actually does is initialize $\lambda = 1$, record the recursion depth on the G_{lg} branch, and update $\lambda \leftarrow (1+\epsilon)\lambda$ whenever this depth is a multiple of $\Theta(\log^3 n)$. In this way, λ never exceeds the global element connectivity w.h.p., which suffices to establish correctness.

5.1.3 Running Time Analysis. The running time analysis in this section closely follows Li and Panigrahi [51].

Lemma 5.8. *Keeping definitions of $R^i, R^i_{\text{sm}}, S^i_v$ as in line 4, 5, 6 of Algorithm 2. Keeping definitions of $G_{\text{lg}}, U_{\text{lg}}, F_{\text{lg}}$ as in line 13, 14, 15 of Algorithm 1. Define $P \subset U^2$ to be*

$$P = \{(u, v) : \kappa'_{G,U\cup F}(u, v) \le W, \text{and } |A \cap U| \le |U|/2 \text{ where } A \text{ is the side of the minimum } u\text{-}v \text{ element cut containing } u\}.$$

Similarly define P_{lg} with $G_{\text{lg}}, U_{\text{lg}}, F_{\text{lg}}$ and W. Then

$$\mathbb{E}(|P_{\text{lg}}|) \le \left(1 - \Omega\left(\frac{1}{\log^2 |U|}\right)\right)|P|.$$

Proof. We need to lower bound the size of $Q = P \backslash P_{\text{lg}}$. Define $U^i_{\text{sm}} = \cup_{v \in R^i_{\text{sm}}} (S^i_v \cap U)$, $U_{\text{sm}} = \cup_{i=1}^{\lfloor \log |U| \rfloor} U^i_{\text{sm}}$, and $U^* = \{v \mid (v,s) \in P\}$. We will show that $\mathbb{E}(|Q|) \geq \Omega(\frac{1}{\log^2 n}) |P|$ follows from the following three claims.

(1) For each $u \in U^*$, there are at least $|U|/2$ vertices v such that $(u,v) \in P$.
(2) $\mathbb{E}(|U_{\text{sm}}|) \geq \Omega(|U^*|/\log |U|)$.
(3) For each $(u,v) \in P$, $u \in U^*$ with probability at least $1/2$;

For Claim (1), consider the minimum element cut disconnecting s and $u \in U^*$. There are at least $|U|/2$ terminals v not in the same side as u, and each $(u,v) \in P$. Claim (2) is proved in Lemma 5.9. Claim (3) holds because s is randomly chosen from U and there are at least $|U| - |U|/2 = |U|/2$ terminals not in the side of u. When s is such a terminal, $u \in U^*$.

The algorithm fixes $i \in \{1, 2, \cdots, \lfloor \log |U| \rfloor\}$ that maximizes $\left|U^i_{\text{sm}}\right|$. Then,

$$\begin{aligned}\mathbb{E}(|Q|) &\geq \frac{|U|}{2}\mathbb{E}(\left|U^i_{\text{sm}}\right|) \geq \frac{|U|}{2\log|U|}\mathbb{E}(|U_{\text{sm}}|) \\ &\geq \frac{|U|}{2\log|U|}\Omega\left(\frac{|U^*|}{\log|U|}\right) \geq \Omega\left(\frac{1}{\log^2|U|}\right)|P|.\end{aligned}$$

For the first inequality, Claim (1) implies that each $v \in U^*_{\text{sm}}$ is involved in $|U|/2$ pairs in P, all of which do not appear in P_{lg} whenever $v \in U^i_{\text{sm}}$. The second inequality follows from the choice of i. The third inequality follows from Claim (2). The fourth inequality follows from Claim (3) and the following bound on the size of P.

$$|P|/2 \leq \mathbb{E}(\left|\{(u,v) \in P : u \in U^*\}\right|) \leq |U| \cdot \left|U^*\right|.$$

Note each u appears in at most $|U|$ pairs of P. □

Lemma 5.9 (Claim (2) restated)**.** $\mathbb{E}(|U_{\text{sm}}|) = \Omega(|U^*|/\log|U|)$.

Proof. Root the element connectivity Gomory-Hu tree T at s. For each vertex $v \in U$, let U_v be the set of terminals in the subtree rooted at v. For a terminal $v \in U$, we find the edge $e(v)$ along the path from s to v with minimum weight, and when not unique, the one with maximum depth. Let $r(v)$ be the deeper endpoint of $e(v)$. By the definition of U^*, a terminal $v \in U^*$ if and only if $w(e(v)) \leq W$ and $\left|U_{r(v)}\right| \leq |U|/2$.

We say that a vertex $v \in U^*$ is *active* if $v \in R^{i(v)}$ where $i(v) = \lfloor \log \left|U_{r(v)}\right| \rfloor$. In addition, if $U_{r(v)} \cap R^{i(v)} = \{v\}$, then we say that v *hits* all of the vertices in $U_{r(v)}$, including itself. For completeness, we define vertices in $U \backslash U^*$ to be inactive; they do not hit other vertices. Now we show that

(a) each vertex that is hit is in U_{sm};
(b) the total number of pairs (u,v) for which $v \in U^*$ hits u is $\Omega(|U^*|)$ in expectation;
(c) each vertex u is hit by at most $O(\log|U|)$ vertices in $v \in U^*$.

For (a), suppose u is hit by v. Then by definition, $U_{r(v)} \cap R^{i(v)} = \{v\}$. The isolating cut for v returned by Lemma 2.4 corresponds to the edge joining $r(v)$ to its parent, so all vertices in $U_{r(v)}$ are on v's side of the cut, and appear in U_{sm}, because $v \in R^i_{\text{sm}}$, since $\left|S^{i(v)}_v \cap U\right| = \left|U_{r(v)}\right| \leq |U|/2$ and $w(e(v)) \leq W$.

For (b), the probability that $v \in R^{i(v)}$ and v is the only such vertex is $(1-2^{-i(v)})^{|U_{r(v)}|-1}2^{-i(v)} = \Theta(1/2^i)$, and when it happens, it hits $\left|U_{r(v)}\right| = \Omega(2^{i(v)})$ vertices, so the contribution in the expectation is $\Omega(1)$. Since each $v \in U^*$ contributes $\Omega(1)$ in expectation, their sum is $\Omega(|U^*|)$.

For (c), we first show that for any different vertices $v, w \in U^*$ that both hit u, $i(v) \neq i(w)$. Since $u \in U_{r(v)}$ and $u \in U_{r(w)}$, without loss of generality we assume $r(v) \in U_{r(w)}$, so $U_{r(v)} \subseteq U_{r(w)}$. From the definition of R^i, $R^0 \subseteq R^1 \subseteq \cdots R^{\lfloor \log |U| \rfloor}$, so $R^{i(v)} \cap R^{i(w)} = R^{\max(i(v),i(w))}$. Then,

$$\begin{aligned}\emptyset = \{v\} \cap \{w\} &= (R^{i(v)} \cap U_{r(v)}) \cap (R^{i(w)} \cap U_{r(w)}) \\ &= R^{\max(i(v),i(w))} \cap U_{r(v)},\end{aligned}$$

and because $R^{i(v)} \cap U_{r(v)} = \{v\} \neq \emptyset$, we infer that $\max(i(v), i(w)) > i(v)$, so $i(v) < i(w)$. Then, since $i(v) \in [1, \log|U|]$ has at most $O(\log|U|)$ kinds of choices, u is hit by at most $O(\log|U|)$ vertices.

Finally, the proof follows from

$$\begin{aligned}\mathbb{E}[U_{\text{sm}}] &\geq \mathbb{E}[|\{u : u \text{ is hit}\}|] \\ &\geq \frac{\mathbb{E}[|\{(u,v) : v \in U^*, u \text{ is hit by } v\}|]}{O(\log|U|)} \geq \Omega(\frac{\mathbb{E}[U^*]}{\log|U|}).\end{aligned}$$

The first inequality is because claim (a); the second inequality is because claim (c); and the third inequality is because claim (b). □

5.2 Proof of Theorem 5.2 and Theorem 5.1

Now we are ready to give the proof for Theorem 5.2 and Theorem 5.1.

Proof of Theorem 5.2. The recursion makes progress in one of two ways. In the "non-G_{lg}" branches $\{G_v\}$, each G_v contains at most half the number of terminals. Suppose we follow the "G_{lg}" branch $\Theta(\log^3 n)$ times, yielding G', U', F' and P'. By Lemma 5.8, with $W = (1+\epsilon)\lambda$, $\mathbb{E}(|P'|) \leq (1-\Omega(1/\log^2 n))^{\Theta(\log^3 n)}|P| = n^{-\Omega(1)}$, meaning $P' = \emptyset$ is empty w.h.p. and the global minimum element cut of $G', U' \cup F'$ has increased to at least $(1+\epsilon)\lambda$ and we can update λ accordingly.

This implies the total depth of recursion is $O(\epsilon^{-1}\log^4 n)$ w.h.p. The total size of all graphs on each layer of recursion is $O(m)$, hence by Lemma 2.4, the total time is $O(\epsilon^{-1}\log^4 n \cdot T_{\text{flow}}(m)\log n) = \tilde{O}(\epsilon^{-1}T_{\text{flow}}(m))$.

As for the correctness, by Lemma 5.3 the G_{lg}-branch preserves the exact value of $\kappa'_{G,U\cup F}$, and all the non-G_{lg} branches $\{G_v\}$ introduce a $(1+\epsilon)$-factor approximation to the element connectivity. Since the depth of recursion in the non-G_{lg} branches is at most $\log|U|$, the tree returned is a $(1+\epsilon)^{\log|U|} = 1+\epsilon_0$ approximate Gomory-Hu tree, for $\epsilon = \epsilon_0/\log|U|$. Expressed in terms of ϵ_0, the running time is still $\tilde{O}(\epsilon_0^{-1}T_{\text{flow}}(m))$. □

Proof of Theorem 5.1. The proof follows from the proof of Theorem 5.2 by setting $\epsilon = \Theta(\frac{1}{k})$, and we only address the difference of k-Gomory-Hu tree.

To prove correctness, a new base case is added[15]: if $\lambda > k$, stop recursion, construct T with one node t, set $g(u) \leftarrow t$ for all $u \in G$ and C an empty function and return (T, g, C). At the final output,

[15] It can be inserted between line 5 and 6 in Algorithm 1

construct f as $f(v) \leftarrow g(v)$ for all $v \in U$. Then (T, f, C) works exactly as in Definition 2.3.

To bound running time, since we are not concerned with cut-values exceeding k, when calling Lemma 2.4 we can use any unit capacity flow algorithm, which runs in $O(T_{\text{uflow}}(m + k|U|))$ time, so the total time bound is $\tilde{O}(kT_{\text{uflow}}(m + k|U|))$. □

6 CONCLUSION

In this paper we proved that $\Omega(kn/\log n)$ space is necessary for encoding vertex connectivity information up to k. This establishes the optimality of several previous results. For example, Nagamochi-Ibaraki [55] sparsifiers encode all vertex connectivities up to k, but their space cannot be improved much, *even if the format of the representation is not constrained to be a graph.* It also implies that even the *average* length of the Izsak-Nutov [40] labeling scheme cannot be improved much. We improved [40] to have near-optimal query time $O(\log n)$, independent of k, and improved its the construction time to nearly max-flow time.

Here we highlight a few open problems.

- There is a trivial $\Omega(kn)$ space lower bound for data structures answering v-cut queries. Our data structure (and Izsak-Nutov [40]) can be augmented to support fast v-cut queries with $O(k^2 n \log n)$ space. Is this necessary? Note that if it is, the lower bound cannot be purely information-theoretic; it must hinge on the requirement that queries be answered *efficiently.*[16]
- A special case of the vertex connectivity oracle problem is answering v-conn(u, v) queries when $k = \kappa(G) + 1$. In other words, decide whether u, v are separated by a globally minimum cut. Globally minimum vertex cuts have plenty of structure [15, 58], but it is still not clear whether $\tilde{\Omega}(kn)$ bits are necessary to answer such queries.
- Is there a $(1+\epsilon)$-approximate vertex connectivity oracle with space $\tilde{O}(n/\epsilon^2)$? Our lower bounds still permit this possibility.

REFERENCES

[1] Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. 2019. Faster Algorithms for All-Pairs Bounded Min-Cuts. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 132)*, Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 7:1–7:15. https://doi.org/10.4230/LIPIcs.ICALP.2019.7

[2] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. 2020. Cut-Equivalent Trees are Optimal for Min-Cut Queries. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. 105–118. https://doi.org/10.1109/FOCS46700.2020.00019

[3] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. 2021. APMF < APSP? Gomory-Hu Tree for Unweighted Graphs in Almost-Quadratic Time. *Accepted to FOCS'21* (2021). https://doi.org/10.48550/ARXIV.2106.02981 arXiv:2106.02981.

[4] G. Di Battista and R. Tamassia. 1996. On-line maintenance of triconnected components with SPQR-Trees. *Algorithmica* 15 (1996), 302–318.

[5] A. A. Benczúr. 1995. Counterexamples for Directed and Node Capacitated Cut-Trees. *SIAM J. Comput.* 24, 3 (1995), 505–510.

[6] A. A. Benczúr. 1995. A Representation of Cuts within 6/5 Times the Edge Connectivity with Applications. In *Proceedings 36th IEEE Symposium on Foundations of Computer Science (FOCS)*. 92–102.

[7] A. A. Benczúr and M. X. Goemans. 2008. Deformable polygon representation and near-mincuts. In *Building Bridges: Between Mathematics and Computer Science*, M. Grötschel and G. O. H. Katona (Eds.). Bolyai Society Mathematical Studies, Vol. 19. Springer, 103–135.

[8] Béla Bollobás. 2011. *Random Graphs, Second Edition.* Cambridge Studies in Advanced Mathematics, Vol. 73. Cambridge University Press. https://doi.org/10.1017/CBO9780511814068

[9] B. Chazelle. 1987. Computing on a free tree via complexity-preserving mappings. *Algorithmica* 2, 3 (1987), 337–361.

[10] Chandra Chekuri. 2015. Some open problems in element connectivity. (2015). Unpublished Survey. Available at http://chekuri. cs. illinois. edu/papers/elem-connectivity-open-probs. pdf.

[11] Chandra Chekuri, Thapanapong Rukkanchanunt, and Chao Xu. 2015. On element-connectivity preserving graph simplification. In *Proceedings 29th Annual European Symposium on Algorithms (ESA)*. 313–324.

[12] Chung-Kuan Cheng and T. C. Hu. 1991. Ancestor tree for arbitrary multi-terminal cut functions. *Ann. Oper. Res.* 33, 3 (1991), 199–213. https://doi.org/10.1007/BF02115755

[13] Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. 2013. Graph Connectivities, Network Coding, and Expander Graphs. *SIAM J. Comput.* 42, 3 (2013), 733–751. https://doi.org/10.1137/110844970

[14] Julia Chuzhoy and Sanjeev Khanna. 2012. An $O(k^3 \log n)$-Approximation Algorithm for Vertex-Connectivity Survivable Network Design. *Theory Comput.* 8, 1 (2012), 401–413. https://doi.org/10.4086/toc.2012.v008a018

[15] Robert F Cohen, Giuseppe Di Battista, Arkady Kanevsky, and Roberto Tamassia. 1993. Reinventing the wheel: an optimal data structure for connectivity queries (extended abstract). In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*. 194–200.

[16] Erik D. Demaine, Gad M. Landau, and Oren Weimann. 2009. On Cartesian Trees and Range Minimum Queries. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*. 341–353. https://doi.org/10.1007/978-3-642-02927-1_29

[17] E. A. Dinic, A. V. Karzanov, and M. V. Lomonosov. 1976. On the structure of the system of minimum edge cuts in a graph. *Studies in Discrete Optimization* (1976), 290–306. (in Russian).

[18] Y. Dinitz and Z. Nutov. 1995. A 2-level cactus model for the system of minimum and minimum+1 edge-cuts in a graph and its incremental maintenance. In *Proceedings 27th ACM Symposium on Theory of Computing (STOC)*. 509–518.

[19] Y. Dinitz and Z. Nutov. 1999. A 2-level cactus tree model for the system of minimum and minimum+1 edge cuts of a graph and its incremental maintenance. Part I: the odd case. (1999). Unpublished manuscript.

[20] Y. Dinitz and Z. Nutov. 1999. A 2-level cactus tree model for the system of minimum and minimum+1 edge cuts of a graph and its incremental maintenance. Part II: the even case. (1999). Unpublished manuscript.

[21] Yefim Dinitz and Alek Vainshtein. 1994. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*. 716–725. https://doi.org/10.1145/195058.195442

[22] Yefim Dinitz and Alek Vainshtein. 1995. Locally Orientable Graphs, Cell Structures, and a New Algorithm for the Incremental Maintenance of Connectivity Carcasses. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 302–311. http://dl.acm.org/citation.cfm?id=313651.313711

[23] Yefim Dinitz and Alek Vainshtein. 2000. The General Structure of Edge-Connectivity of a Vertex Subset in a Graph and its Incremental Maintenance. Odd Case. *SIAM J. Comput.* 30, 3 (2000), 753–808. https://doi.org/10.1137/S0097539797330045

[24] Shimon Even, Gene Itkis, and Sergio Rajsbaum. 1998. On Mixed Connectivity Certificates. *Theor. Comput. Sci.* 203, 2 (1998), 253–269. https://doi.org/10.1016/S0304-3975(98)00023-1

[25] Donatella Firmani, Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. 2016. Strong Articulation Points and Strong Bridges in Large Scale Graphs. *Algorithmica* 74, 3 (2016), 1123–1147. https://doi.org/10.1007/s00453-015-9991-z

[26] Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. 2020. Computing and Testing Small Connectivity in Near-Linear Time and Queries via Fast Local Cut Algorithms. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2046–2065. https://doi.org/10.1137/1.9781611975994.126

[27] András Frank, Toshihide Ibaraki, and Hiroshi Nagamochi. 1993. On Sparse Subgraphs Preserving Connectivity Properties. *J. Graph Theory* 17, 3 (jul 1993), 275–281. https://doi.org/10.1002/jgt.3190170302

[28] Alan Frieze and Michał Karoński. 2016. *Introduction to Random Graphs.* Cambridge University Press.

[29] Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2020. Minimum Cut in $O(m \log^2 n)$ Time. In *Proceedings 47th International Colloquium on Automata, Languages, and Programming (ICALP) (LIPIcs, Vol. 168)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 57:1–57:15. https://doi.org/10.4230/LIPIcs.ICALP.2020.57

[16] There are two natural ways to answer v-cut queries "optimally," either enumerate their elements in $O(k)$ time, or return a pointer in $O(1)$ time to a pre-stored list of elements. The latter model was recently advocated by Nutov [56] and seems to be easier to characterize from a lower bound perspective.

[30] Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2021. A Note on a Recent Algorithm for Minimum Cut. In *Proceedings 4th SIAM Symposium on Simplicity in Algorithms (SOSA)*. 74–79. https://doi.org/10.1137/1.9781611976496.8

[31] Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2016. 2-Edge Connectivity in Directed Graphs. *ACM Trans. Algorithms* 13, 1 (2016), 9:1–9:24. https://doi.org/10.1145/2968448

[32] Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2018. 2-vertex connectivity in directed graphs. *Inf. Comput.* 261 (2018), 248–264. https://doi.org/10.1016/j.ic.2018.02.007

[33] R. E. Gomory and T. C. Hu. 1961. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.* 9 (1961).

[34] D. Gusfield and D. Naor. 1990. Efficient Algorithms for Generalized Cut Trees. In *Proceedings First ACM-SIAM Symposium on Discrete Algorithms*. 422–433.

[35] Dan Gusfield and Dalit Naor. 1993. Extracting maximal information about sets of minimum cuts. *Algorithmica* 10, 1 (1993), 64–89.

[36] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. 2007. Efficient algorithms for computing all low s-t edge connectivities and related problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 127–136. http://dl.acm.org/citation.cfm?id=1283383.1283398

[37] Refael Hassin and Asaf Levin. 2007. Flow trees for vertex-capacitated networks. *Discrete applied mathematics* 155, 4 (2007), 572–578.

[38] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. 2015. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *Proceedings 47th Annual ACM Symposium on Theory of Computing (STOC)*. 21–30.

[39] Tai-Hsin Hsu and Hsueh-I Lu. 2009. An optimal labeling for node connectivity. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*. Springer, 303–310.

[40] Rani Izsak and Zeev Nutov. 2012. A Note on Labeling Schemes for Graph Connectivity. *Inf. Process. Lett.* 112, 1–2 (2012), 39–43. https://doi.org/10.1016/j.ipl.2011.10.001

[41] Kamal Jain, Ion I. Mandoiu, Vijay V. Vazirani, and David P. Williamson. 2002. A primal-dual schema based approximation algorithm for the element connectivity problem. *J. Algorithms* 45, 1 (2002), 1–15. https://doi.org/10.1016/S0196-6774(02)00222-5

[42] A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen. 1991. On-Line Maintenance of the Four-Connected Components of a Graph. In *Proceedings 32nd IEEE Symposium on Foundations of Computer Science (FOCS)*. 793–801.

[43] D. R. Karger. 2000. Minimum cuts in near-linear time. *J. ACM* 47, 1 (2000), 46–76.

[44] David R. Karger and Debmalya Panigrahi. 2009. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 246–255. http://dl.acm.org/citation.cfm?id=1496770.1496798

[45] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. 2020. Unit Capacity Maxflow in Almost $O(m^{4/3})$ Time. In *Proceedings 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 119–130. https://doi.org/10.1109/FOCS46700.2020.00020

[46] Michal Katz, Nir A Katz, Amos Korman, and David Peleg. 2004. Labeling schemes for flow and connectivity. *SIAM J. Comput.* 34, 1 (2004), 23–40.

[47] T. Kopelowitz, S. Pettie, and E. Porat. 2016. Higher Lower Bounds from the 3SUM Conjecture. In *Proceedings 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1272–1287. https://doi.org/10.1137/1.9781611974331.ch89

[48] A. Korman. 2010. Labeling schemes for vertex connectivity. *ACM Trans. on Algorithms* 6, 2 (2010).

[49] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. 2021. Vertex Connectivity in Polylogarithmic Max-flows. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*.

[50] Jason Li and Debmalya Panigrahi. 2020. Deterministic Min-cut in Polylogarithmic Max-flows. In *Proceedings 61st IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*. 85–92. https://doi.org/10.1109/FOCS46700.2020.00017

[51] Jason Li and Debmalya Panigrahi. 2021. Approximate Gomory–Hu Tree is Faster than $n^{\tilde{}}1$ Max-Flows. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*. 1738—-1748. https://doi.org/10.1145/3406325.3451112

[52] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. 2021. A Nearly Optimal All-Pairs Min-Cuts Algorithm in Simple Graphs. *Accepted to FOCS'21* (2021). arXiv:2106.02233.

[53] Karl Menger. 1927. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae* 10 (1927), 96–115.

[54] Sagnik Mukhopadhyay and Danupon Nanongkai. 2020. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proccedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*. 496–509. https://doi.org/10.1145/3357713.3384334

[55] H. Nagamochi and T. Ibaraki. 1992. A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph. *Algorithmica* 7, 5&6 (1992), 583–596.

[56] Zeev Nutov. 2021. Data structure for node connectivity queries. *arXiv preprint arXiv:2110.09102* (2021).

[57] Seth Pettie. 2006. An Inverse-Ackermann Type Lower Bound For Online Minimum Spanning Tree Verification. *Combinatorica* 26, 2 (2006), 207–230. https://doi.org/10.1007/s00493-006-0014-1

[58] Seth Pettie and Longhui Yin. 2021. The Structure of Minimum Vertex Cuts. In *Proceedings 48th International Colloquium on Automata, Languages, and Programming (ICALP) (LIPIcs, Vol. 198)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 105:1–105:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.105

[59] J.-C. Picard and M. Queyranne. 1980. On the structure of all minimum cuts in a network and applications. In *Combinatorial Optimization II*. Mathematical Programming Studies, Vol. 13. Springer, 8–16.

[60] Thatchaphol Saranurak. 2021. A Simple Deterministic Algorithm for Edge Connectivity. In *Proceedings 4th SIAM Symposium on Simplicity in Algorithms, (SOSA)*. 80–85. https://doi.org/10.1137/1.9781611976496.9

[61] C.-P. Schnorr. 1979. Bottlenecks and Edge Connectivity in Unsymmetrical Networks. *SIAM J. Comput.* 8, 2 (1979), 265–274.

[62] Alexander Schrijver. 2003. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer.

[63] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2021. Minimum Cost Flows, MDPs, and ℓ_1-Regression in Nearly Linear Time for Dense Instances. arXiv:2101.05719 [cs.DS]