Breaking the $n^k$ Barrier for Minimum $k$-cut on Simple Graphs

Zhiyang He∗ Jason Li†

December 2, 2021

Abstract

In the minimum $k$-cut problem, we want to find the minimum number of edges whose deletion breaks the input graph into at least $k$ connected components. The classic algorithm of Karger and Stein [KS96] runs in $\tilde{O}(n^{2k-2})$ time,\(^1\) and recent, exciting developments have improved the running time to $O(n^k)$ [GHLL20]. For general, weighted graphs, this is tight assuming popular hardness conjectures.

In this work, we show that perhaps surprisingly, $O(n^k)$ is not the right answer for simple, unweighted graphs. We design an algorithm that runs in time $O(n^{(1-\epsilon)k})$ where $\epsilon > 0$ is an absolute constant, breaking the natural $n^k$ barrier. This establishes a separation of the two problems in the unweighted and weighted cases.

\*Department of Mathematics, Massachusetts Institute of Technology, Cambridge, United States.
†Simons Institute, UC Berkeley, Berkeley, United States.

\(^1\) $\tilde{O}(\cdot)$ denotes omission of polylogarithmic factors in $n$. 
1 Introduction

In this paper, we study the (unweighted) minimum $k$-cut problem: given an undirected graph $G = (V, E)$ and an integer $k$, we want to delete the minimum number of edges to split the graph into at least $k$ connected components. Throughout the paper, let $\lambda_k$ denote this minimum number of edges. Note that the $k$-cut problem generalizes the global minimum cut problem, which is the special case $k = 2$.

For fixed constant $k \geq 2$, the first polynomial-time algorithm for this problem is due to Goldschmidt and Hochbaum [GH94], who designed an algorithm running in $n^{O(k^2)}$ time. Subsequently, Karger and Stein showed that their (recursive) randomized contraction algorithm solves the problem in $\tilde{O}(n^{2k-2})$ time. This was later matched by a deterministic algorithm of Thorup [Tho08] based on tree packing, which runs in $\tilde{O}(n^{2k})$ time.

These algorithms remained the state of the art until a few years ago, when new progress was established on the problem [GLL18, GLL19, Li19], culminating in the $\tilde{O}(n^k)$ time algorithm of Gupta, Harris, Lee, and Li [GHL20] which is, surprisingly enough, just the original Karger-Stein recursive contraction algorithm with an improved analysis. The $\tilde{O}(n^k)$ time algorithm also works for weighted graphs, and they show by a reduction to max-weight $k$-clique that their algorithm is asymptotically optimal, assuming the popular conjecture that max-weight $k$-clique cannot be solved faster than $\Omega(n^{k-O(1)})$ time. However, whether the algorithm is optimal for unweighted graphs was left open; indeed, the (unweighted) $k$-clique problem can be solved in $n^{(\omega/3)k+O(1)}$ time through fast matrix multiplication.\footnote{As standard, we define $\omega$ as the smallest constant such that two $n \times n$ matrices can be multiplied in $O(n^{\omega+o(1)})$ time. The best bound known is $\omega < 2.373$ [AW21], although $\omega = 2$ is widely believed.}

In this paper, we make partial progress on this last question by showing that for simple graphs, the right answer is asymptotically bounded away from $n^k$:

**Theorem 1.1.** There is an absolute constant $\epsilon > 0$ such that the minimum $k$-cut problem can be solved in $n^{(1-\epsilon)k+O(1)}$ time.

In fact, we give evidence that $n^{(\omega/3)k+O(1)}$ may indeed be the right answer (assuming the popular conjecture that $k$-clique cannot be solved any faster). This is discussed more in the statement of Theorem 1.3.

1.1 Our Techniques

Our high-level strategy mimics that of Li [Li19], in that we make use of the Kawarabayashi-Thorup graph sparsification technique on simple graphs, but our approach differs by exploiting matrix multiplication-based methods as well. Below, we describe these two techniques and how we apply them.

**Kawarabayashi-Thorup Graph Sparsification** Our first algorithmic ingredient is the (vertex) graph sparsification technique of Kawarabayashi and Thorup [KT18], originally developed to solve the deterministic minimum cut problem on simple graphs. At a high level, the sparsification process
Matrix Multiplication  What if the minimum k-cut has components that are singleton vertices? If all but one component is a singleton, then we can use a matrix multiplication-based algorithm similar to the Nešetřil and Poljak’s algorithm for k-clique [NP85], which runs in \( n^{(\omega/3)k+O(1)} \) time. Thus, the main difficulty is to handle minimum k-cuts where some components are singletons, but not k − 1 many. The following definition will be at the core of all our discussions for the rest of this paper.

Definition 1.2 (Border and Islands). Given a k-cut C with exactly r singleton components, we denote the singleton components as \( S_1 = \{v_1\}, \ldots, S_r = \{v_r\} \) and denote the other components \( S_{r+1}, \ldots, S_k \). A border of C is a cut obtained by merging some singleton components into larger components. More precisely, a border is defined by a subset \( I \subseteq [r] \) and a function \( \sigma : I \rightarrow [k \setminus [r]] \). Given I and \( \sigma \), we let \( S'_1 = S_1 \cup \{v_j : j \in I, \sigma(j) = i\} \), then the border \( B_{I,\sigma} \) is the \((k - |I|)\)-cut defined by the components \( S'_{r+1}, \ldots, S'_k \), together with the unmerged singleton components \( S_j \) where \( j \in [r \setminus I] \). The set of vertices \( \{v_i : i \in [I]\} \), corresponding to the merged singleton components, is called the islands.

Given this definition, our main technical contribution is as follows: we show that if the cut C has exactly r singleton components, then we can first apply Kawarabayashi-Thorup sparsification to compute a graph \( G' \) of size \( \tilde{O}(n/\lambda_k) \) that preserves some borders of C. We then use the algorithm of [GHLL20] on \( G' \) to discover a border, which will succeed with probability roughly \( 1/\tilde{O}(n/\lambda_k)^{k-|I|} \). Finally, we run a matrix multiplication-based algorithm to locate the islands in an additional \( n^{(\omega/3)|I|+O(1)} \) time. Altogether, the runtime becomes \( \tilde{O}((n/\lambda_k)^{k-s}) \cdot n^{(\omega/3)s+O(1)} \), which is \( n^{(1-\epsilon')k+O(1)} \) as long as \( \lambda_k \geq n^\epsilon \), where \( \epsilon' \) depends on \( \epsilon \).

We summarize our discussions with the following theorem, which is the real result of this paper.

**Theorem 1.3.** Suppose there exists an algorithm that takes in a simple, unweighted graph G, and returns its minimum k-cut in time \( \lambda_k^{ck}n^{O(1)} \). Let \( c = \max\left(\frac{\epsilon}{1+\epsilon}, \frac{\omega}{3}\right) \). Then we can compute a minimum k-cut of a simple, unweighted graph in \( O(n^{ck+O(1)}) \).
Recently, Lokshtanov, Saurabh, and Surianarayanan [LSS20] showed an algorithm for exact minimum $k$-cut that runs in time $\lambda^{O(k)} n^{O(1)}$. Combining their result with Theorem 1.3, we obtain a minimum $k$-cut that runs in time $O(n^{ck+O(1)})$ for some constant $c < 1$. We further note that we use their algorithm in a black-box manner, which means if one could derive an exact algorithm with a better constant $t$, then our algorithm will have an improved runtime up to $O(n^{\omega k/3+O(1)})$.

2 Main Algorithm

In this section, we discuss our algorithm in detail. Given a simple, unweighted graph $G$, we first run an approximate $k$-cut algorithm to determine the magnitude of $\lambda_k$. If $\lambda_k \leq O(n^{1/t+1})$, then we can run the exact algorithm on $G$ and output its result. Otherwise, we apply Lemma 2.2, which is a modified version of Kawarabayashi-Thorup sparsification [KT18] for $k$-cuts. These modifications, discussed in Section 3, will give us a graph $G'$ on $O_k(n/\lambda_k)$ vertices that preserves at least one border for every minimum $k$-cut of $G$. Now we fix any minimum $k$-cut of $G$, and fix its border $B_{I,\sigma}$ specified by Lemma 2.2. For every possible value of $|I|$, we run Lemma 2.3 to discover $B_{I,\sigma}$ with high probability.

Once we found the border, locating the islands is simple. In Section 5, we present a slight variant of Nešetřil and Poljak’s $k$-clique algorithm [NP85] that solves the following problem in $O(n^{\frac{3r}{\omega}+O(1)})$ time.

Definition 2.1 ($r$-Island Problem). Given a graph $G$, find the optimal $r+1$-cut $C$ which has exactly $r$ singleton components.

This enables us to recover the minimum $k$-cut in $G$ by guessing the number of islands in each non-singleton component specified by the border, and finding them independently. The total runtime is $O(n^{\frac{3r}{\omega}|I|+O(1)})$ since the number of islands in any non-singleton component is at most the total number of islands $|I|$. This proves Theorem 1.3.

Our methods are summarized in the following algorithm. Note that for the initial $O(1)$-approximation step, various algorithms can be used.

Algorithm 1 Main Algorithm

1: Run an 2-approximation algorithm of $k$-CUT in polynomial time [SV95], and let its output be $\lambda_{k}$.
2: if $\lambda_{k} \leq 10^{1/t}$ then
3: Run the given exact algorithm for $k$-CUT
4: else
5: Apply Lemma 2.2 to obtain a graph $G'$ on $O_k(n/\lambda_k)$ vertices.
6: for each $i = 0, 1, 2, \ldots, r$ do $\triangleright$ Iterate over possible values of $i = |I|$ of the border $B_{I,\sigma}$
7: Run Lemma 2.3 with parameter $\beta = 1 - (1 - 2/\log n)i/k$.
8: for each cut $C$ output by Lemma 2.3 do $\triangleright$ at most $(n/\lambda_k)^{k-(1-2/\log n)i+O(1)}$ many
9: Guess the number of islands in each non-singleton component of $C$.
10: Run the Island Discovery Algorithm in each non-singleton component.
11: end for
12: end for
13: end if
2.1 Analysis

Our analysis is divided into three parts, each corresponding to one section of the algorithm. The first part concerns the Kawarabayashi-Thorup sparsification, and the following theorem is proved in Section 3.

Lemma 2.2. For any simple graph, we can compute in $k^{O(k)}n^{O(1)}$ time a partition $V_1, \ldots, V_q$ of $V$ such that $q = (k \log n)^{O(1)}n/\lambda_k$ and the following holds:

(*) For any minimum $k$-cut $C$ with exactly $r$ singleton components, there exists $I \subset [r]$ and a function $\sigma : I \rightarrow [k] \setminus [r]$ such that the border of $C$ defined by $I$ and $\sigma$, namely $B_{I, \sigma}$, agrees with the partition $V_1, \ldots, V_q$. In other words, all edges of $B_{I, \sigma}$ are between some pair of parts $V_i, V_j$. Moreover, we have $|B_{I, \sigma}| \leq \lambda_k - (1 - 2/\log n)|I|\lambda_k/k$.

Contracting each $V_i$ into a single vertex, we obtain a graph $G'$ on $\tilde{O}(n/\lambda_k)$ vertices that preserves $B_{I, \sigma}$.

Next, we describe and analyze the algorithm that computes the border. The following lemma is proved in Section 4.

Lemma 2.3. Fix an integer $2 \leq s \leq k$ and a parameter $\beta \leq 1$, and consider an $s$-cut $C$ of size at most $\beta \lambda_k$. There is an $n^{3\beta k + O(1)}$ time algorithm that computes a list of $n^{\beta k + O(1)}$ $s$-cuts such that with high probability, $C$ is listed as one of the cuts.

Finally, we present and analyze the algorithm that extends the border by computing the missing islands in each non-singleton component. The following lemma is proved in Section 5.

Lemma 2.4. There is a $O_r(n^{2k/3} + O(1))$ deterministic algorithm that solves the $r$-Island problem.

With these three lemmas in hand, we now analyze Algorithm 1.

Fix a minimum $k$-cut. The initial Kawarabayashi-Thorup sparsification takes $k^{O(k)}n^{O(1)}$ time by Lemma 2.2, and the border $B_{I, \sigma}$ is preserved by the partition and has size at most $\lambda_k - (1 - 2/\log n)|I|\lambda_k/k$. For the correct guess of $|I|$, Lemma 2.3 detects $B_{I, \sigma}$ with high probability among a collection of $n/\lambda_k^{k-1-2/\log n} + O(1)$ many $(k-i)$-cuts. Finally, for the $(k-i)$-cut $C = B_{I, \sigma}$, the Island Discovery Algorithm extends it to a minimum $k$-cut in time $n^{(\omega/3) + O(1)}$. The total running time is therefore

$$k^{O(k)}n^{O(1)} + \left(n/\lambda_k\right)^{k-1-2/\log n} + O(1) \cdot n^{(\omega/3) + O(1)} \leq k^{O(k)}n^{O(1)} + \left(n/\lambda_k\right)^{k-1-2/\log n + O(1)} + \left(n/\lambda_k\right)^{k-1-2t/\log n + O(1)} \cdot n^{(\omega/3) + O(1)}.$$

The $n^{2t/\log n}$ term is at most $O(1)^{2t}$, which is negligible. The running time is dominated by either $i = 0$ or $i = k$, depending on which of $\frac{i}{t-1}$ and $\omega/3$ is greater. This concludes the analysis of Algorithm 1 and the proof of Theorem 1.3.

3 Kawarabayashi-Thorup Sparsification

In this section, we prove the following Kawarabayashi-Thorup sparsification theorem of any simple graph. Rather than view it as a vertex sparsification process where groups of vertices are contracted,
we work with the grouping of vertices itself, which is a partition of the vertex set. We use parts to denote the vertex sets of the partition to distinguish them from the components of a k-cut.

Most of the arguments in this section originate from Kawarabayashi and Thorup’s original paper [KT18], though we find it more convenient to follow the presentations of [GLL21] and [Li19].

**Lemma 2.2.** For any simple graph, we can compute in \( k^{O(k)}n^{O(1)} \) time a partition \( V_1, \ldots, V_q \) of \( V \) such that \( q = (k \log n)^{O(1)} n / \lambda_k \) and the following holds:

\[ (*) \text{ For any minimum } k\text{-cut } C \text{ with exactly } r \text{ singleton components, there exists } I \subset [r] \text{ and a function } \sigma : I \to [k] \setminus [r] \text{ such that the border of } C \text{ defined by } I \text{ and } \sigma, \text{ namely } B_{I,\sigma}, \text{ agrees with the partition } V_1, \ldots, V_q. \text{ In other words, all edges of } B_{I,\sigma} \text{ are between some pair of parts } V_i, V_j. \]

Moreover, we have \( |B_{I,\sigma}| \leq \lambda_k - (1 - 2/\log n)|I|\lambda_k / k. \)

Contracting each \( V_i \) into a single vertex, we obtain a graph \( G' \) on \( \tilde{O}(n/\lambda_k) \) vertices that preserves \( B_{I,\sigma} \).

### 3.1 Regularization Step

We first “regularize” the graph to obey a few natural conditions, which is done at no asymptotic cost to the number of clusters. In particular, we ensure that \( m \leq O(\lambda_k n) \), i.e., there are not too many edges, and \( \delta \geq \lambda_k / k \), i.e., the minimum degree is comparable to the size of the k-cut.

**Nagamochi-Ibaraki sparsification.** First, we show that we can freely assume \( m = O(\lambda_k n) \) through an initial graph sparsification step due to Nagamochi and Ibaraki; the specific theorem statement here is from [Li19].

**Theorem 3.1** (Nagamochi and Ibaraki [NI92], Theorem 3.3 in [Li19]). Given a simple graph \( G \) and parameter \( s \), there is a polynomial-time algorithm that computes a subgraph \( H \) with at most \( sn \) edges such that all \( k \)-cuts of size at most \( s \) are preserved. More formally, for all \( k \)-cuts \( S_1, \ldots, S_k \) satisfying \( |E_G[S_1, \ldots, S_k]| \leq s \), we have \( E_G[S_1, \ldots, S_k] = E_H[S_1, \ldots, S_k] \).

Compute a \( (1 + 1/k) \)-approximation \( \tilde{\lambda}_k \in [\lambda_k, (1 + 1/k)\lambda_k] \) in time \( k^{O(k)}n^{O(1)} \) [LSS20], apply Theorem 3.1 with parameter \( s = \tilde{\lambda}_k \), and replace \( G \) with the returned graph \( H \). This allows us to assume \( m \leq (1 + 1/k)\lambda_k n \) henceforth.

**Lower bound the minimum degree.** Next, we would like to ensure that the graph \( G \) has minimum degree comparable to \( \lambda_k \). While there exists a vertex of degree less than \( \frac{\tilde{\lambda}_k}{(1 + 1/k)(k-1)} \), declare that vertex as a trivial part in the final partition, and remove it from \( G \). We claim that we can remove at most \( k - 1 \) such vertices; otherwise, the vertices together form a \( k \)-cut of size less than \((k - 1) \cdot \frac{\tilde{\lambda}_k}{(1 + 1/k)(k-1)} = \frac{\tilde{\lambda}_k}{(1 + 1/k)} \leq \lambda_k \), contradicting the value \( \lambda_k \) of the minimum \( k \)-cut. We have thus removed at most \( k - 1 \) vertices. The remaining task is to compute a partition of the remaining graph which has minimum degree at least \( \frac{\tilde{\lambda}_k}{(1 + 1/k)(k-1)} \geq \lambda_k / k \). We then add a singleton set for each of the singleton vertices removed, which is at most \( k - 1 \) extra parts, which is negligible since we aim for \((k \log n)^{O(1)} n / \lambda_k \) many parts in total.
3.2 Kawarabayashi-Thorup Sparsification

It remains to prove the following lemma, which is Lemma 2.2 with the additional assumptions $m \leq 2\lambda_k n$ and $\delta \geq \lambda_k / k$.

**Lemma 3.2.** Suppose we are given a simple graph with $m \leq 2\lambda_k n$ and $\delta \geq \lambda_k / k$. Then, we can compute a partition $V_1, \ldots, V_q$ of $V$ such that $q = (k \log n)^{O(1)} n / \lambda_k$ and the following holds:

(*) For any minimum $k$-cut $C$ with exactly $r$ singleton components, there exists $I \subset [r]$ and a function $\sigma : I \rightarrow [k] \setminus [r]$ such that the border of $C$ defined by $I$ and $\sigma$, namely $B_{I,\sigma}$, agrees with the partition $V_1, \ldots, V_q$. In other words, all edges of $B_{I,\sigma}$ are between some pair of parts $V_i, V_j$.

Moreover, we have $|B_{I,\sigma}| \leq \lambda_k - (1 - 2 / \log n) |I| \lambda_k / k$.

Our treatment follows closely from Appendix B of [GLL21].

**Expander decomposition preliminaries.** We first introduce the concept of the conductance of a graph, as well as an expander, defined below.

**Definition 3.3 (Conductance).** Given a graph $G = (V, E)$, a set $S : \emptyset \subseteq S \subseteq V$ has conductance

$$\frac{|\partial_G S|}{\min\{\text{vol}(S), \text{vol}(V \setminus S)\}}$$

in the graph $G$, where $\text{vol}(S) := \sum_{v \in S} \text{deg}(v)$. The conductance of the graph $G$ is the minimum conductance of a set $S \subseteq V$ in $G$.

**Definition 3.4.** For any parameter $0 < \gamma \leq 1$, a graph is a $\gamma$-expander if its conductance is at least $\gamma$.

The following is a well-known result about decomposing a graph into expanders, for which we provide an easy proof below for convenience.

**Theorem 3.5 (Expander Decomposition).** For any graph $G = (V, E)$ with $m$ edges and a parameter $\gamma < 1$, there exists a partition $U_1, \ldots, U_p$ of $V$ such that:

1. For all $i \in [p]$, $G[U_i]$ is a $\gamma$-expander.
2. $|E[U_1, \ldots, U_p]| \leq O(\gamma m \log m)$.

**The partitioning algorithm.** To compute the partition $V_1, \ldots, V_q$, we execute the same algorithm from Section B of [GLL21], except we add an additional step 4. Throughout the algorithm, we fix parameter $\epsilon := 1 / (k \log n)$.

1. Compute an expander decomposition with parameter $\gamma := 1 / \delta$, and let $U_1, \ldots, U_p$ be the resulting partition of $V$. 

6
2. Initialize the set $S \leftarrow \emptyset$, and initialize $C_i \leftarrow U_i$ for each $i \in [p]$. While there exists some $i \in [p]$ and a vertex $v \in C_i$ satisfying $\deg_{G[C_i]}(v) \leq \frac{2}{5} \deg_G(v)$, i.e., vertex $v$ loses at least $\frac{3}{5}$ fraction of its degree when restricted to the current $C_i$, remove $v$ from $C_i$ and add it to $S$. The set $S$ is called the set of singleton vertices. Note that some $C_i$ can become empty after this procedure. At this point, we call each $C_i$ a cluster of the graph. This procedure is called the trimming step in [KT18].

3. Initialize the set $L := \bigcup_{i \in [p]} \{ v \in C_i \mid \deg_{G[C_i]}(v) \leq (1 - \epsilon) \deg_G(v) \}$, i.e., for each $i \in [p]$ and vertex $v \in C_i$ that loses at least $\epsilon$ fraction of its degree when restricted to $C_i$, add $v$ to $L$ (but do not remove it from $C_i$ yet). Then, add $L$ to the singletons $S$ (i.e., update $S \leftarrow S \cup L$) and define the core of a cluster $C_i$ as $A_i \leftarrow C_i \setminus L$. For a given core $A_i$, let $C(A_i)$ denote the cluster whose core is $A_i$. This procedure is called the shaving step in [KT18].

4. For each core $A_i$ with at most $k$ vertices, we shatter the core by adding $A_i$ to the singletons $S$ (i.e., update $S \leftarrow S \cup A_i$) and updating $A_i \leftarrow \emptyset$. This is the only additional step relative to [GLL21].

5. Suppose there are $p' \leq p$ nonempty cores $A_i$. Let us re-order the cores $A_1, \ldots, A_{p'}$ so that $A_1, \ldots, A_{p'}$ are precisely the nonempty cores. The final partition $\mathcal{P} = \{ V_1, V_2, \ldots \}$ of $V$ is $\bigcup_{i \in [p']} \{ A_i \} \cup \bigcup_{v \in S} \{ \{ v \} \}$. In other words, we take each nonempty core $A_i$ as its own set in the partition, and add each vertex $v \in S$ as a singleton set. We call each nonempty core $A_i$ a core in the partition, and each vertex $v \in S$ as a singleton in the partition.

The lemmas below are stated identically to those in [GLL21], so we omit the proofs and direct interested readers to [GLL21].

**Lemma 3.6** (Lemma B.11 of [GLL21]). Fix a parameter $\alpha \geq 1$ that satisfies $\alpha < o(\delta / \log n)$. For each nonempty cluster $C$ and a subset $S \subseteq V$ satisfying $|\partial_G S| \leq \alpha \delta$, we have either $|C \cap S| \leq 3\alpha$ or $|C \setminus S| \leq 3\alpha$.

The lemma below from [GLL21] is true for the algorithm without step 4.

**Lemma 3.7** (Corollary B.9 of [GLL21]). Suppose we skip step 4 of the algorithm. Then, there are $O(\frac{m \log m}{\delta^2})$ many sets in the partition $\mathcal{P}$.

Clearly, adding step 4 increases the number of parts by a factor of at most $k$, so the we obtain the following corollary.

**Corollary 3.8.** There are $O(\frac{k m \log m}{\delta^2})$ many sets in the partition $\mathcal{P}$.

Since $m \leq \lambda_k n$ and $\delta \geq \lambda_k / k$ by the assumption of Lemma 3.2, this fulfills the bound $q = (k \log n)^{O(1)} n / \lambda_k$ of Lemma 3.2. For the rest of this section, we prove property ($*$).

The following lemma is a combination of Lemma B.12 of [GLL21] and Lemma 16 of [Li19], and we provide a proof for completeness.
Lemma 3.9. Fix a parameter $\alpha \geq 1$ that satisfies $\alpha < o(\frac{\delta}{k \log n})$. For any nonempty core $A$ and any minimum $k$-cut of size at most $\alpha \delta$, there is exactly one component $S^*$ satisfying $|S^* \cap C(A)| > 3\alpha$, and any other component $S$ that is non-singleton must be disjoint from $A$. Moreover, each vertex $v \in A$ has at least $(1 - 2\epsilon) \deg(v)$ neighbors in $S^*$.

Proof. We first show that $|C(A)| > 3\alpha k$. Since $C(A)$ is nonempty, each vertex $v \in C(A)$ has at least $\frac{\delta}{n} \deg(v) \geq \frac{\delta}{2} \delta$ neighbors in $C(A)$, so $|C(A)| > \frac{\delta}{2} \delta - 1 > 3\alpha k$ by the assumption $\alpha < o(\frac{\delta}{k \log n})$.

By Lemma 3.6, each component $S$ must satisfy $|C(A) \cap S| \leq 3\alpha k$ or $|C(A) \setminus S| \leq 3\alpha k$, and the latter implies that $|C(A) \cap S| > |C(A)|/2$, which only one side $S$ can satisfy. Moreover, one such component $S^*$ must exist since otherwise, $|C(A)| = \sum_S |C(A) \cap S| \leq 3\alpha k$, a contradiction. Therefore, all but one component $S^*$ satisfy $|C(A) \cap S| \leq 3\alpha$.

Next, each vertex $v \in A$ has at least $(1 - \epsilon) \deg(v)$ neighbors in $C(A)$, and at most $3\alpha k$ of them can go to $C(A) \cap S$ for any component $S \neq S^*$. This leaves at least $(1 - \epsilon) \deg(v) - 3\alpha k$ neighbors in $S^*$, which is at least $(1 - 2\epsilon) \deg(v)$ since $\epsilon = 1/ \log n$ and $\alpha < o(\frac{\delta}{k \log n})$.

We now show that if $S$ is non-singleton and $|C(A) \cap S| \leq 3\alpha k$, then $S$ is disjoint from $A$. Suppose otherwise; then, any vertex $v \in A \cap S$ has at least $(1 - 2\epsilon) \deg(v)$ neighbors in $S^*$ as before. If we move $v$ from $S$ to $S^*$, then the result is still a $k$-cut since $S$ is non-singleton. Moreover, the edges from $v$ to $S$ are newly cut, and the edges from $v$ to $S^*$ are saved. The former is at most $\epsilon \deg(v) + 3\alpha k$, and the latter at least $(1 - 2\epsilon) \deg(v)$. Since $\epsilon = 1/ \log n$ and $\alpha < o(\frac{\delta}{k \log n})$, the new $k$-cut is smaller than the old one, a contradiction.

Finally, we prove property $(\ast)$ of Lemma 3.2.

Lemma 3.10. For any minimum $k$-cut $C$ with exactly $r$ singleton components, there exists $I \subset [r]$ and a function $\sigma : I \to [k] \setminus [r]$ such that the border of $C$ defined by $I$ and $\sigma$, namely $B_{I,\sigma}$, agrees with the partition $V_1, \ldots, V_q$. In other words, all edges of $B_{I,\sigma}$ are between some pair of parts $V_i, V_j$. Moreover, we have $|B_{I,\sigma}| \leq \lambda_k - (1 - 2/ \log n)|I| \lambda_k / k$.

Proof. Enumerate the singleton components as $S_1 = \{v_1\}, \ldots, S_r = \{v_r\}$. Let $T$ be the set of singleton components $S_i$ such that $S_i$ is contained in a part $V_j$ that has more vertices than just $v_i$ (i.e., $V_j \supseteq \{v_i\}$).

For every such component $S_i = \{v_i\}$, since $V_j \supseteq \{v_i\}$, we must have $|V_j| > k$, since otherwise it would have been shattered into singletons on step 4 of the algorithm. So there must be a non-singleton component $S^*_j$ of the minimum $k$-cut intersecting $V_j$ (which is unique by Lemma 3.9). This component must be the $S^*$ from Lemma 3.9. We define $\sigma(i) = i^*$.

As we’ve argued in the previous paragraph, the border $S'_r, \ldots, S'_k$ defined as $S'_i = S_i \cup \{v_j : j \in I, \sigma(j) = i\}$ agrees with the partition $V_1, \ldots, V_q$. It remains to show that $|E(S'_r, \ldots, S'_k)| \leq \lambda_k - (1 - 1/k)|I| \lambda_k / k$. For each component $S_i = \{v_i\}$ with $i \in I$, by Lemma 3.9, the vertex $v_i$ has at least $(1 - 2\epsilon) \deg(v)$ neighbors in $S^*_{\sigma(i)}$, so merging $v_i$ with $S^*_{\sigma(i)}$ decreases the cut value by at least $(1 - 2\epsilon) \deg(v)$. It follows that the border has size at most $\lambda_k - (1 - 2\epsilon)|I| \deg(v)$, which meets the bound since $\epsilon = 1/ \log n$ and $\deg(v) \geq \delta \geq \lambda_k / k$ by assumption.

With Lemma 3.10, this concludes the proof of Lemma 3.2.
4 Finding the Border

In this section, we develop an algorithm to compute the border. The main lemma is the following, where $C$ represents the border we wish to find.

**Lemma 2.3.** Fix an integer $2 \leq s \leq k$ and a parameter $\beta \leq 1$, and consider an $s$-cut $C$ of size at most $\beta \lambda_k$. There is an $n^{\beta k + O(1)}$ time algorithm that computes a list of $n^{\beta k + O(1)}$ $s$-cuts such that with high probability, $C$ is listed as one of the cuts.

Our algorithm follows Karger’s contraction algorithm, stated below, and its analysis from [GHLL20].

**Algorithm 2 Contraction Algorithm [GHLL20]**

1: while $|V| > \tau$ do
2: Choose an edge $e \in E$ at random from $G$, with probability proportional to its weight.
3: Contract the two vertices in $e$ and remove self-loops.
4: end while
5: Return a $k$-cut of $G$ chosen uniformly at random.

The key lemma we use is the following from [GHLL20].

**Lemma 4.1** (Lemma 17 of [GHLL20]). Suppose that $J$ is an edge set with $\alpha = |J|/\lambda_k$ and $n \geq \tau \geq 8\alpha k^2 + 2k$. Then $J$ survives lines 1 to 4 of the Contraction Algorithm with probability at least $(n/\tau)^{-\alpha k} k^{-O(\alpha k^2)}$.

The algorithm sets $\tau = 8\beta k^2 + 2k$, and by Lemma 4.1, any $s$-cut $C$ of size $\alpha \lambda_k$ for some $\alpha \leq \beta$ survives lines 1 to 4 of the Contraction Algorithm with probability $k^{-O(k^2)} n^{-\alpha k} \geq k^{-O(k^2)} n^{-\beta k}$. The algorithm sets $s$ for the parameter $k$, and $C$ is output with probability $1/r^\tau \geq k^{-O(k^2)}$. Overall, the probability of outputting $C$ is $k^{-O(k^2)} n^{-\beta k}$. Repeating the algorithm $k^{O(k^2)} n^{\beta k} \log n$ times, we can output a list of cuts that contains $C$ with high probability.

5 Finding the Islands

In this section, we prove the following lemma.

**Lemma 2.4.** There is a $O_r(n^{\frac{kr}{r} + O(1)})$ deterministic algorithm that solves the $r$-Island problem.

We present an algorithm for $r$-island which is a variant of Nešetřil and Poljak’s $k$-clique algorithm [NP85]. Given an input graph $G$, we want to find the optimal $r$ vertices to cut off from $G$. Note that this is similar to finding the minimum $r$-clique in $G$, except that we need to take into account the edges from the $r$ islands to the remaining giant component in $G$. We first consider the case where $r$ is divisible by 3.
Algorithm 3 Island Discovery Algorithm

1. We construct a weighted graph $G'$ as follows — for every subset of vertices $S$ such that $|S| = \frac{r}{3}$, create a vertex $v_S$. Denote the total number of edges among vertices in $S$ as $w_S$, and denote the total number of edges between $S$ and $V \setminus S$ as $w_{S'}$. For each pair of vertices $v_S, v_T$, let $w_{S,T}$ be the total number of edges between $S$ and $T$ if they are disjoint. Add an edge between them of weight $w_{S,T}$.

2. We want to find the minimum weight triangle in the graph $G'$. To do so, we guess the weight of a minimum weight triangle as follows: Denote the three vertices as $v_{S_1}, v_{S_2}, v_{S_3}$. Guess $w_{S_1}, w_{S_2}, w_{S_3}, w_{S_1}', w_{S_2}', w_{S_3}'$, and $w_{S_1,s_2}, w_{S_2,s_3}, w_{S_3,s_1}$.

3. Denote $A$ as the binary adjacency matrix for $G'$. Let $F_1$ denotes the set of vertices $v_S$ such that $w_S = w_{S_1}, w_{S_2}' = w_{S_3}'$. Define $A_{1,2}$ to be the matrix $A$ with the rows restricted to vertices in $F_1$, and columns restricted to vertices in $F_2$. Additionally, for $v_S \in F_1, v_T \in F_2$, if $w_{S,T} \neq w_{S_1,s_2}$, set $A_{1,2}[S,T] = 0$. Define $A_{2,3}, A_{3,1}$ similarly.

4. Compute the matrix product $B = A_{1,2} \times A_{2,3}$. If there exists $v_S \in F_1$, $v_T \in F_3$ such that $B[S,T] \neq 0, A_{3,1} = 1$, then find $v_R$ such that $A_{1,2}[S,R] = A_{2,3}[R,T] = 1$ and return $S, R, T$. Otherwise, return Null.

Claim 5.1. Algorithm 3 returns an optimal $(r+1)$-cut with $r$ islands with probability at least $\frac{1}{O(r^4n^3)}$.

Proof. We first note that given the nine parameters $w_{S_1}, w_{S_2}, w_{S_3}, w_{S_1}', w_{S_2}', w_{S_3}'$, and $w_{S_1,s_2}, w_{S_2,s_3}, w_{S_3,s_1}$, the weight of the returned cut would be $w_{S_1} + w_{S_2} + w_{S_3} + w_{S_1,s_2} + w_{S_2,s_3} + w_{S_3,s_1} + (w_{S_1}' - w_{S_1}s_2 - w_{S_3,s_1}) + (w_{S_2}' - w_{S_1,s_2} - w_{S_2,s_3}) + (w_{S_3}' - w_{S_2,s_3} - w_{S_3,s_1})$. In other words, the nine parameters precisely specify the weight of the returned cut. Therefore, if we guess the parameters correctly, our algorithm will return $r$-vertices that gives the minimum $r + 1$ cut with $r$ islands. Note that $w_{S_1}, w_{S_2}, w_{S_3}$ and $w_{S_1,s_2}, w_{S_2,s_3}, w_{S_3,s_1}$ each have $O(r^2)$ possible values, while $w_{S_1}', w_{S_2}', w_{S_3}'$ each have $O(rn)$ possible values. Therefore there are at most $O(r^{15}n^3)$ possible combination of values for the nine parameters, which means we guess correctly with probability as least $\frac{1}{O(r^4n^3)}$. The rest of the algorithm is a standard triangle detection algorithm using matrix multiplication, which has runtime $O(n^{\frac{15}{2}})$.

If $r$ is not divisible by 3, we can add up to two isolated vertices into the graph and reduce to the case where $r$ is divisible by 3. This increase the runtime by a factor of $n^{O(1)}$. Now note that our above algorithm can be easily made deterministic by going over all $O(r^{15}n^3)$ possible combinations of the nine parameters instead of guessing them. This proves Lemma 2.4.

Acknowledgements

The authors would like to thank Anupam Gupta for many constructive discussions and comments.
References

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[GH94] Olivier Goldschmidt and Dorit S. Hochbaum. A polynomial algorithm for the $k$-cut problem for fixed $k$. *Math. Oper. Res.*, 19(1):24–37, 1994.

[GHLL20] Anupam Gupta, David G Harris, Euiwoong Lee, and Jason Li. Optimal bounds for the $k$-cut problem. *arXiv preprint arXiv:2005.08301*, 2020.

[GLL18] Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for $k$-cut. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 113–123. IEEE, 2018.

[GLL19] Anupam Gupta, Euiwoong Lee, and Jason Li. The number of minimum $k$-cuts: Improving the karger-stein bound. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 229–240, 2019.

[GLL21] Anupam Gupta, Euiwoong Lee, and Jason Li. The connectivity threshold for dense graphs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 89–105. SIAM, 2021.

[KS96] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

[KT18] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM (JACM)*, 66(1):1–50, 2018.

[Li19] Jason Li. Faster minimum $k$-cut of a simple graph. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1056–1077. IEEE, 2019.

[LSS20] Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min $k$-cut. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 798–809. IEEE, 2020.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.*, 5(1):54–66, 1992.

[NP85] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[SV95] Huzur Saran and Vijay V. Vazirani. Finding $k$-cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995.
[Tho08] Mikkel Thorup. Minimum $k$-way cuts via deterministic greedy tree packing. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 159–166. ACM, 2008.