



Cut-Query Algorithms with Few Rounds

Yotam Kenneth-Mordoch 

Weizmann Institute of Science, Rehovot, Israel

Robert Krauthgamer 

The Harry Weinrebe Professorial Chair of Computer Science, Weizmann Institute of Science, Rehovot, Israel

Abstract

In the cut-query model, the algorithm can access the input graph $G = (V, E)$ only via cut queries that report, given a set $S \subseteq V$, the total weight of edges crossing the cut between S and $V \setminus S$. This model was introduced by Rubinfeld, Schramm and Weinberg [ITCS'18] and its investigation has so far focused on the number of queries needed to solve optimization problems, such as global minimum cut. We turn attention to the round complexity of cut-query algorithms, and show that several classical problems can be solved in this model with only a constant number of rounds.

Our main results are algorithms for finding a minimum cut in a graph, that offer different tradeoffs between round complexity and query complexity, where $n = |V|$ and $\delta(G)$ denotes the minimum degree of G : (i) $\tilde{O}(n^{4/3})$ cut queries in two rounds in unweighted graphs; (ii) $\tilde{O}(rn^{1+1/r}/\delta(G)^{1/r})$ queries in $2r+1$ rounds for any integer $r \geq 1$ again in unweighted graphs; and (iii) $\tilde{O}(rn^{1+(1+\log_n W)/r})$ queries in $4r+3$ rounds for any $r \geq 1$ in weighted graphs. We also provide algorithms that find a minimum (s, t) -cut and approximate the maximum cut in a few rounds.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Submodular optimization and polymatroids

Keywords and phrases Cut Queries, Round Complexity, Submodular Optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.100

Related Version A full version is available at: [arXiv:2506.20412](https://arxiv.org/abs/2506.20412)

Funding Robert Krauthgamer: Work partially supported by the Israel Science Foundation grant #1336/23, by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center, and by a research grant from the Estate of Harry Schutzman.

1 Introduction

Graph cuts are a fundamental object in computer science with numerous applications, in part because they are a basic example of submodular functions. Recall that a set function $f : 2^V \rightarrow \mathbb{R}$ is *submodular* if it satisfies the diminishing-returns property

$$\forall A \subset B \subset V, \forall v \notin B, \quad f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B).$$

From the viewpoint of submodular optimization, it is natural to study the complexity of graph algorithms in the *cut-query model*, which correspond to value queries to the submodular function. Here, the input graph $G = (V, E, w)$ can be accessed only via oracle queries to its cut function, namely to $\text{cut}_G : 2^V \rightarrow \mathbb{R}$, given by

$$\forall S \subseteq V, \quad \text{cut}_G(S) := \sum_{e \in E: |e \cap S|=1} w(e).$$

Here and throughout $w(e) > 0$ is the weight of the edge $e \in E$, precluding edges of zero weight as they cannot be detected by the algorithm. As usual, an unweighted graph models unit weights, i.e., $w(e) = 1$ for all e .



© Yotam Kenneth-Mordoch and Robert Krauthgamer;
licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 100; pp. 100:1–100:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *query complexity* of an algorithm is the number of queries it makes in the worst-case. While this is often the primary performance measure, it is highly desirable that algorithms can be parallelized, as such algorithms can utilize better the available computing resources. The literature on the parallelization of submodular optimization [9, 24, 25, 18], relies on a measure called *round complexity*, which count the number of sequential rounds of queries the algorithm makes.¹

► **Definition 1.1** (Round Complexity). *An algorithm has round complexity r if it performs r rounds of queries, meaning that queries in round $t \in [r]$ can depend only on the answers to queries in previous rounds $1, \dots, t-1$. An algorithm is called non-adaptive if it uses a single round of queries.*

We study the cut-query model, from the perspective of round complexity focusing on the problem of finding a global minimum cut. Previous work [33, 31, 3, 2] shows that a minimum cut of a graph can be found using $O(n)$ queries in unweighted graphs and $\tilde{O}(n)$ queries in weighted graphs. However, it paid no attention to the algorithms' round complexity, and we fill this gap by studying the tradeoff between round complexity and query complexity.

An analysis of the round complexity of existing algorithms, with some simple modifications, shows that it is possible to find a global minimum cut of a graph using $\tilde{O}(n)$ queries with round complexity $\tilde{O}(\log^2 n)$ [33, 31]. It should be noted that other algorithms do require round complexity $\Omega(n)$ [3, 2]. As another baseline, it is possible to recover the entire graph using $O(n^2)$ deterministic non-adaptive queries [14],² which clearly suffices to find a minimum cut. This motivates our main question

► **Question 1.2.** *What is the tradeoff between the round complexity and the query complexity of finding a global minimum cut in the cut-query model?*

1.1 Main Results

We show that the polylogarithmic round complexity of previous work is not necessary for finding a global minimum cut. We begin by presenting two results for unweighted graphs. The first one shows that using even two rounds, we can substantially improve on the $O(n^2)$ query complexity of the naive algorithm. All of our algorithms return a vertex set $S \subseteq V$ and the value of the minimum cut. In the case of unweighted graphs the algorithms can also report the edges of the cut using one additional round of queries and with the same asymptotic query complexity.

► **Theorem 1.3** (Unweighted Minimum Cut with 2 rounds). *Given an unweighted graph G on n vertices, it is possible to find a minimum cut of G using $\tilde{O}(n^{4/3})$ cut queries in 2 rounds. The algorithm is randomized and succeeds with probability $1 - n^{-2}$.*

We further provide a smooth tradeoff between the number of queries and the number of rounds, namely $\tilde{O}(n^{1+1/r}/\delta(G)^{1/r})$ cut queries with in $2r$ rounds, where $\delta(G) \geq 1$ is the minimum degree of G .³

¹ The aforementioned references use the term *adaptivity*, which is the number of rounds minus one.

² $\tilde{O}(|E|)$ queries suffice, however without some bound on the size of the graph the algorithm must make $O(n^2)$ queries in the worst case.

³ We can assume that the graph is connected since connectivity can be checked using $\tilde{O}(n)$ cut queries in 1 round (non-adaptive) [4].

► **Theorem 1.4** (Unweighted Minimum Cut with $O(r)$ rounds). *Given an unweighted graph G on n vertices and a parameter $r \in \{1, 2, \dots, \log n\}$, it is possible to find a minimum cut of G using $\tilde{O}(rn^{1+1/r}\delta(G)^{-1/r})$ cut queries in $2r + 1$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-1}$.*

We also obtain a similar tradeoff also for weighted graphs.

► **Theorem 1.5** (Weighted Minimum Cut Graphs with $O(r)$ rounds). *Given a weighted graph G on n vertices with integer edge weights bounded by W , and a parameter $r \in \{1, 2, \dots, \log n\}$, it is possible to find a minimum cut of G using $\tilde{O}(rn^{1+(1+\log_n W)/r})$ cut queries in $4r + 3$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-2}$.*

Finally, we note that our techniques can also be applied to approximating the maximum cut and finding a minimum (s, t) -cut with low adaptivity. Furthermore, our result for weighted minimum cuts can also be applied to finding a minimum cut in dynamic streams, providing a smooth tradeoff between the number of passes and the storage complexity of the algorithm. This holds since each cut query in a given round can be computed using $O(\log n)$ storage in a single pass by simply counting the edges in the cut. Therefore, the round complexity of our algorithm translates immediately to the number of passes required by a streaming algorithm. These results are detailed in the full version of the paper.

1.2 Related Work

Algorithms in the cut query model. In recent years there have been several works on cut problems in the cut-query model, most of which focused on query complexity and not round complexity. We will discuss the known results for several problems in the cut-query model.

Beginning with finding a minimum cut, $O(n)$ randomized queries suffice for simple graphs [33, 3]. Meanwhile, finding a minimum cut in a weighted graph requires $\tilde{O}(n)$ randomized queries [31]. Finally, in a recent work it was shown that non-trivial deterministic algorithms exist for finding the minimum cut of a simple graph using $\tilde{O}(n^{5/3})$ queries [2]. For the easier problem of determining whether a graph is connected, it is known that $O(n)$ randomized queries suffice [19, 30]. Additionally, there exists a one round randomized algorithm for connectivity that uses $\tilde{O}(n)$ queries [4].

Moving to the problem of finding a minimum (s, t) -cut. In [33] it was shown that it is possible to compute the minimum (s, t) -cut of a simple graph using $O(n^{5/3})$ queries, by showing that using a cut sparsifier it is possible to find a small set of edges that contains the minimum (s, t) -cut. This query complexity was matched in the deterministic setting by [2].

We also mention the problem of finding a $1 - \epsilon$ approximation of the maximum cut. One way to achieve this is by constructing a cut sparsifier, this was shown in [33] for simple graphs and in [32] for weighted graphs, both using $\tilde{O}(\epsilon^{-2}n)$ cut queries. Furthermore, an algorithm for $1/2$ -approximation using $O(\log n)$ cut queries in one round and a lower bound of $\tilde{\Omega}(n)$ on the query complexity needed to find a $1 - \epsilon$ approximation of the maximum cut are also known [32].

Finally, there have been several works on graph problems in different query models, such as additive queries [22, 14, 15], quantum cut queries [29, 6, 3], matrix multiplication queries [34, 6, 3], Bipartite Independent Set queries [11, 4, 1] and more.

Adaptivity in submodular optimization. In recent years, there has been a growing interest in the adaptivity of algorithms for submodular optimization. This line of work was initiated by the work of [9], which showed that constrained submodular maximization can be solved

within a constant approximation factor in $O(\log n)$ rounds with a polynomial number of queries, the work also provides a matching lower bound. A string of later works improved on the query complexity and the approximation guarantee, culminating in an algorithm with an optimal $1 - 1/e$ approximation using $\tilde{O}(n)$ queries [9, 7, 8, 24, 25, 21]. The round complexity of unconstrained submodular maximization was also studied in [20].

The adaptivity of algorithms for submodular minimization was also studied in several works. It was shown that solving the minimum (s, t) -cut problem in weighted graphs using $O(\log n)$ rounds of queries requires $\tilde{\Omega}(n^2)$ queries [5]. For more general submodular functions, a bound of $\Omega(n/\log n)$ rounds for algorithms with a polynomial number of queries is known [16, 10, 17]. Note that all these hard instances are non symmetric, and hence do not apply to the minimum cut problem. It remains open to study the round complexity of symmetric submodular minimization problems in future work.

2 Technical Overview

Each of our three algorithms for minimum cut is based on a different insight. Our two-round algorithm uses contractions in the style of [28] to create a smaller graph that preserves the minimum cut. Similar contractions have been used in several minimum cut algorithm [28, 26, 3], however the existing algorithms aim to reduce the number of vertices in the graph, while our variant, called τ -star contraction, reduces the number of edges.

Our $(2r)$ -round algorithm for unweighted graphs employs a known approach of applying a contraction procedure and then packing (in the contracted graph) edge-disjoint forests. The main challenge to ensure that forests are edge disjoint when packing them in parallel, and we overcome it by leveraging an elegant connection to cut sparsification, which shows that amplifying the sampling probabilities of the sparsifier construction by a factor of k ensures finding k edge-disjoint forests.

Our algorithm for weighted minimum cut follows the approach of [31], which constructs a $(1 \pm \epsilon)$ -cut-sparsifier and then solves a monotone-matrix problem that is easily parallelizable. Prior work in the cut-query model provided cut sparsifiers [33, 32], but their core step of weight-proportional edge sampling requires $O(\log n)$ rounds. We eliminate this bottleneck with our two-round weight-proportional edge sampling primitive (Lemma 2.2), and thereby build a $(1 \pm \epsilon)$ -cut-sparsifier in $3r$ rounds, and complete the entire algorithm in $O(r)$ rounds.

A common thread to all our algorithms is the use of edge sampling, both directly as part of the algorithms and to construct complex primitives such as forest packings and cut sparsifiers. In edge sampling, the input is a source vertex s and a target vertex set $T \subseteq V \setminus \{s\}$, and the goal is to sample an edge from $E(s, T) := E \cap (\{s\} \times T)$. We need two types of edge sampling, both returning a sampled edge $e \in E(s, T)$ and its weight $w(e)$. The first one is *uniform edge sampling*, which picks each $e \in E(s, T)$ with equal probability $1/|E(s, T)|$ regardless of its weight, similarly to l_0 sampling; the second one is *weight-proportional edge sampling*, which picks each $e \in E(s, T)$ with probability $w(e)/w(E(s, T))$, similarly to l_1 sampling. In unweighted graphs these two definitions clearly coincide.

Weight-proportional edge sampling was used extensively in previous algorithms for the cut-query model [33, 31, 3, 32], although it had a naive implementation that requires $O(\log n)$ rounds. This naive implementation, which was proposed in [33], follows a binary search approach. Each step partitions the set T at random into two sets T_1, T_2 of equal cardinality, and finds the total weight of the edges in $E(s, T_1)$ and $E(s, T_2)$. The algorithm then decides whether to continue with T_1 or T_2 randomly in proportion to their weights. This approach clearly requires $\Omega(\log n)$ rounds, which exceeds the $O(1)$ round complexity we aim for, and furthermore yields only weight-proportional sampling, while some of our algorithms require uniform sampling in weighted graphs.

2.1 Edge Sampling Primitives

Our first technical contribution is a procedure for non-adaptive uniform edge sampling using $O(\log^3 n / \log \log n)$ queries that returns the weight of the sampled edge. An algorithm for uniform edge sampling using $O(\log^3 n)$ cut queries in one round (i.e. non-adaptively) was proposed in [4].⁴ However, their algorithm is based on group testing principles using BIS queries⁵ and returns only the endpoints of the edge, but not its weight, though it is probably not difficult to adapt their algorithm to return the weight as well. Our algorithm slightly improves on the query complexity of their algorithm, while also returning the edge's weight in the same round.

► **Lemma 2.1** (Uniform Edge Sampling). *Given a (possibly weighted) graph G on n vertices, a source vertex $s \in V$, and a target vertex set $T \subseteq V \setminus \{s\}$, one can return an edge e uniformly sampled from $E(s, T)$, along with its weight $w(e)$, using $O(\log^3 n / \log \log n)$ cut queries in one round, i.e. non-adaptively. The algorithm succeeds with probability $1 - n^{-4}$.*

The algorithm works by subsampling the vertices in T into sets $T = T_0 \supseteq T_1 \supseteq \dots \supseteq T_{\log n}$, and finds a level i such that $0 < |E(s, T_i)| \leq O(\log n)$. It then leverages sparse recovery techniques to recover all the edges in $E(s, T_i)$ and return one of them at random. To perform this algorithm in one round the algorithm recovers $w(E(s, T_i))$ and performs the queries needed for sparse recovery for all levels in parallel. It then applies the sparse recovery algorithm to the last level with non-zero number of edges, i.e. the last level with $w(E(s, T_i)) > 0$. We note that it is easy to decrease the query complexity of Lemma 2.1 by a factor of $O(\log n)$ by adding a round of queries and performing the sparse recovery only on the relevant level.

Our second procedure performs weight-proportional edge sampling in two rounds.

► **Lemma 2.2** (Weight-Proportional Edge Sampling). *Given a graph G on n vertices with integer edge weights bounded by W , a source vertex $s \in V$, and a target vertex set $T \subseteq V \setminus \{s\}$, one can return an edge e from $E(s, T)$, along with its weight $w(e)$, sampled with probability $w(e)/w(E(s, T))$ using $O(\log^2 n \log^2(nW))$ queries in two rounds. The algorithm succeeds with probability $1 - n^{-4}$.*

The algorithm follows the same basic approach as the uniform edge sampling primitive, i.e. creating nested subsets $T = T_0 \supseteq T_1 \supseteq \dots \supseteq T_{\log(nW)}$ by subsampling. However, instead of recovering only the edges that were sampled into the last level with non-zero weight, the algorithm recovers in each level i all edges $e \in E(s, T_i)$ such that $w(e) \approx w(E(s, T))/2^i$. The recovery procedure is based on a Count-Min data structure [23], which we show can be implemented using few non-adaptive cut queries. The algorithm then uses another round of queries to learn the exact weights of all edges that were recovered in all the levels. It then concludes by sampling an edge from the distribution $\{p_e\}$, where for every edge e recovered in the i -th level, it sets $p_e \propto 2^i w(e)/w(E(s, T))$; with probability $1 - \sum_e p_e$, the algorithm fails. Since the probability of being sampled into the i -th level is 2^{-i} , these probabilities yield the desired distribution over the recovered edges.

To prove that the algorithm returns an edge with high probability, we show that there is a constant probability that the outlined procedure returns some edge $e \in E(s, T)$, and hence repeating the algorithm $O(\log n)$ times yields with high probability an edge sampled from

⁴ The algorithm requires $O(\log^2 n \log(1/\delta))$ cut queries and succeeds with probability $1 - \delta$, setting $\delta = n^{-2}$ yields the bound.

⁵ A BIS query is given two disjoint sets $A, B \subseteq V$ and returns true if there exists some edge connecting A, B and false otherwise.

$E(s, T)$ with the desired distribution. The query complexity of the algorithm is dominated by the Count-Min data structure, which requires $O(\log n \log(nW))$ cut queries in each level. Applying the recovery procedure on $O(\log(nW))$ levels, and repeating the procedure $O(\log n)$ times yields the stated query complexity.

2.2 Unweighted Minimum Cut in Two Rounds

The key idea of our algorithm is to apply an edge contraction technique that produces a graph with substantially fewer edges while preserving each minimum cut with constant probability. We then recover the entire contracted graph using a limited number of cut queries by applying existing graph recovery methods using $\tilde{O}(m)$ additive queries [22, 14]. We show that it is possible to simulate k additive queries using $O(n + k)$ cut queries, which allows us to recover a graph with $m \geq \tilde{O}(n)$ edges using $O(m)$ cut queries.

► **Corollary 2.3.** *Given a weighted graph G on n vertices and m edges, one can recover the graph G using $O(m + n)$ non-adaptive cut queries.*

While there has been frequent use of contraction procedures for finding minimum cuts, existing algorithms focused on reducing the number of vertices in the graph [28, 26, 3], typically to $\tilde{O}(n/\delta(G))$. This does not suffice to guarantee a graph with few edges, for example, applying such a contraction on a graph G with minimum degree $\delta(G) = \log^2 n$ and $\Omega(n)$ vertices of degree n , might yield a graph with $\tilde{\Omega}(n^2)$ edges. In contrast, our goal is to reduce the number of edges in the graph, allowing efficient recovery of the contracted graph using Corollary 2.3. The main technical contribution of this section is a new contraction algorithm, which we call τ -star contraction, that reduces the number of edges to $\tilde{O}((n/\tau)^2 + n\tau)$.

Our contraction algorithm is a thresholded version of the star-contraction algorithm introduced in [3]. The original version of the star-contraction algorithm of [3] can be roughly described as follows. Sample a subset of *center vertices* $R \subseteq V$ uniformly at random with probability $p = O(\log n/\delta(G))$. For every $v \in V \setminus R$, choose uniformly a vertex in $r \in N_G(v) \cap R$ and contract the edge (v, r) , keeping parallel edges, which yields a contracted multigraph G' . The main guarantee of the algorithm is that if the minimum cut of G is non-trivial, i.e. not composed of a single vertex, then $\lambda(G) = \lambda(G')$ with constant probability, where $\lambda(G)$ denotes the value of a minimum cut in a graph. The following theorem states this formally.

► **Theorem 2.4** (Theorem 2.2 in [3]). *Let $G = (V, E)$ be an unweighted graph on n vertices with a non-trivial minimum-cut value $\lambda(G)$. Then, the star-contraction algorithm yields a contracted graph G' that,*

1. *has at most $O(n \log n/\delta(G))$ vertices with probability at least $1 - 1/n^4$, and*
2. *has $\lambda(G') \geq \lambda(G)$ always, equality is achieved with probability at least $2 \cdot 3^{-13}$.*

Note that in the cut query model, the algorithm cannot contract an edge because the underlying cut function remains constant. However, it can simulate the contraction of $e = (u, v)$ by merging the vertices u and v into a supervertex in all subsequent queries, which yields a contracted cut function. We now define our τ -star contraction algorithm.

► **Definition 2.5** (τ -star contraction). *Given an unweighted graph $G = (V, E)$ on n vertices and a threshold $\tau \in [n]$, τ -star contraction is the following operation.*

1. *Let $H = \{v \in V \mid d(v) \geq \tau\}$ be the set of high-degree vertices.*
2. *Form the set $R \subseteq V$ by sampling each $v \in V$ independently with probability $p = \Theta(\log n/\tau)$.*
3. *For each $u \in H \setminus R$, pick an edge from $E(u, R)$, if non-empty, uniformly at random and contract it (keeping parallel edges).*

Notice that τ -star contraction depends on uniform edge sampling, since each vertex $u \in H \setminus R$ samples a neighbor in R . In our case, we can use the uniform edge sampling procedure described above to sample edges uniformly from $E(u, R)$.

We now sketch the proof that τ -star contraction yields a graph with few edges. To bound the number of edges, partition the vertices remaining after the contraction into two sets, R , i.e. the center vertices sampled in the sampling process, and $L = V \setminus H$, i.e. the low degree vertices. This analysis omits vertices in $H \setminus R$ that were not sampled into R or contracted, because every $v \in H \setminus R$ has in expectation $d_G(v) \cdot \log n / \tau \in \Omega(\log n)$ neighbors in R ; hence with high probability it will have some neighbor in R and be contracted.

To bound the number of edges observe that $|R| \leq \tilde{O}(n/\tau)$ with high probability, and hence the number of edges between vertices in R is $O(|R|^2) \leq \tilde{O}(n^2/\tau^2)$. In addition, since the degree of the vertices of L is bounded by τ , the total number of edges incident on L , even before the contraction, is $O(n\tau)$. Hence, the total number of edges in G' is with high probability $\tilde{O}(n^2/\tau^2 + n\tau)$. The following lemma, proved in the full version of the paper, formalizes the guarantees of τ -star contraction.

► **Lemma 2.6.** *Let $G = (V, E)$ be an unweighted graph on n vertices with a non-trivial minimum-cut $\lambda(G)$ and let $\tau \in [n]$. A τ -star contraction of G gives G' that,*

1. *has at most $\tilde{O}(n^2/\tau^2 + n\tau)$ edges with probability at least $1 - 1/n^4$, and*
2. *has $\lambda(G') \geq \lambda(G)$ always, equality is achieved with probability at least $2 \cdot 3^{-13}$.*

We present in Algorithm 1 a procedure for finding the global minimum cut in two rounds using τ -star contraction. We will use it to prove Theorem 1.3, but first we need the following claim.

■ **Algorithm 1** Unweighted Minimum Cut in 2 Rounds.

```

1: Input: Graph  $G = (V, E)$ 
2: Output: A minimum cut  $\mathcal{C} \subseteq V$ 
3: sample a set  $R \subseteq V$  uniformly with probability  $p = 400 \cdot \log n / \tau$ 
   // Query round 1
4: query  $\text{cut}_G(v)$  for all  $v \in V$                                 ▷ find  $d_G(v)$  for all  $v \in V$ 
5: for each  $v \in V \setminus R$ , let  $n(v) \leftarrow$  a random neighbor in  $N_G(v) \cap R$     ▷ use Lemma 2.1
6:  $\tau \leftarrow \max\{n^{1/3}, \delta(G)\}$ 
7:  $V' \leftarrow V$ 
8: for  $v \in \{V \setminus R \mid \text{cut}_G(v) \geq \tau\}$  do
9:   update  $V'$  by contracting the edge  $(v, n(v))$ 
   // Query round 2
10: recover  $G' = (V', E')$  with  $\tilde{O}(n^{2-2/3} + n^{1+1/3})$  queries    ▷ use Corollary 2.3, assuming
     $|E'| = \tilde{O}(n^{2-2/3} + n^{1+1/3})$ 
11: return  $\min(\lambda(G'), \delta(G))$                                 ▷ can return also a set  $S \subseteq V$  achieving the cut

```

▷ **Claim 2.7.** With probability $1 - n^{-3}$, line 9 of Algorithm 1 implements τ -star contraction with $\tau = \max\{n^{1/3}, \delta(G)\}$.

Proof. Notice that R is sampled uniformly from V and that line 9 is executed only for vertices in $H \setminus R$. It remains to show that the edge $(v, n(v))$ is sampled uniformly from $N_G(v) \cap R$, which indeed follows from Lemma 2.1. To analyze the success probability, note that the algorithm samples at most n edges in line 5 and by the guarantee of Lemma 2.1 this fails with probability at most $n \cdot n^{-4} = n^{-3}$. ◁

Proof of Theorem 1.3. The main argument is that Algorithm 1 finds a minimum cut of G with constant probability. Then, running this algorithm $O(\log n)$ times in parallel and returning the minimum output obtained gives the desired result. Notice that by Lemma 2.6, the connectivity of G' can only increase, hence, if the algorithm finds a minimum cut of G in some execution it will be returned. We assume throughout the proof that in all executions of the algorithm the size guarantee of Lemma 2.6 holds, i.e. $|E'| = \tilde{O}(n^{2-2/3} + n^{1+1/3})$, note that this holds with probability $1 - \tilde{O}(n^{-4})$. Using a union bound with Claim 2.7 above, we find that the algorithm succeeds with probability at least $1 - n^{-2}$.

By Claim 2.7 the graph G' is the result of the τ -star contraction with $\tau = n^{1/3}$, and thus $\lambda(G') = \lambda(G)$ with constant probability by Lemma 2.6. Therefore, recovering the graph G' and returning the minimum of $\lambda(G')$ and $\delta(G)$ gives a minimum cut of G with constant probability.

To bound the query complexity, observe that the query complexity of the algorithm is dominated by the number of queries in the graph recovery step, which is given by $\tilde{O}(n^{2-2/3} + n^{1+1/3}) = \tilde{O}(n^{4/3})$. In the first round, we use $\tilde{O}(n)$ queries as finding the degree of each vertex requires n queries, and sampling a neighbor in $E(v, R)$ can be done using $O(\log^3 n / \log \log n)$ queries by Lemma 2.1. This concludes the analysis of the query complexity of the algorithm, which is $\tilde{O}(n^{4/3})$.

Finally, notice that the algorithm proceeds in two rounds; in the first round the algorithm only queries the degrees of the vertices and samples an edge in $E(v, R)$ for every v , which does not require any prior information on the graph. In the second round, it recovers G' round using Corollary 2.3. This concludes the proof of Theorem 1.3. \blacktriangleleft

2.3 Unweighted Minimum Cut in $2r$ rounds

This section provides an algorithm for finding a minimum cut in $2r$ rounds, thus proving Theorem 1.4. The algorithm uses a known approach of first applying a contraction algorithm, yielding a graph with $\tilde{O}(n/\delta(G))$ vertices, and then packing $\delta(G)$ edge-disjoint forests in the contracted graph. The main challenge in adapting this algorithm to low round complexity, is ensuring that the forests are edge disjoint when packing them in parallel. This is achieved by leveraging a connection to cut sparsification, which shows that sampling edges with probabilities amplified by a factor of k relative to those needed for sparsifier construction ensures finding k edge disjoint forests within the sampled subset. We note that we also offer another algorithm for finding a minimum cut, that has slightly worse query complexity of $O(rn^{1+1/r})$, and is based on constructing a cut sparsifier and the results of [33], see the full version of the paper.

The contraction we use is the *2-out contraction* procedure of [26], which is obtained by sampling uniformly two edges incident to each vertex and contracting all the sampled edges. The following theorem states the guarantees of the 2-out contraction.

► **Theorem 2.8** (Theorem 2.4 in [26]). *Let $G = (V, E)$ be an unweighted graph on n vertices with a non-trivial minimum-cut value $\lambda(G)$. Then, applying a 2-out contraction algorithm gives G' where,*

1. G' has at most $\tilde{O}(n/\delta(G))$ vertices with high probability, and
2. has $\lambda(G') \geq \lambda(G)$ always, equality is achieved with constant probability.

The other building block of the algorithm is maximal k -packing of forests.

► **Definition 2.9** (Maximal k -Packing of Forests). *Given a (possibly weighted) graph G , a maximal k -packing of forests is a set of edge-disjoint forests T_1, \dots, T_k that are maximal, i.e. for every $i \in [k]$ and $e \in E \setminus \cup_j T_j$, the edge set $T_i \cup \{e\}$ contains a cycle.*

► **Lemma 2.10.** *Given a (possibly weighted) graph G on n vertices and parameters $r \in \{1, \dots, \log n\}$ and $k \in [n]$, one can find a maximal k -packing of forests of G using $\tilde{O}(krn^{1+1/r})$ cut queries in $2r$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-4}$.*

We now outline the algorithm of Lemma 2.10, for further details see the full version of the paper. The algorithm begins by creating a set of empty trees T_1, \dots, T_k . Then, in each of $r - 1$ rounds it uniformly samples $\tilde{O}(kn^{1+1/r})$ edges from the graph, augments the trees $\{T_i\}$ by adding all sampled edges that do not form a cycle, and contracts all vertices that are connected in all trees. We show that the number of edges decreases by an $n^{1/r}$ factor in each round; hence, after $r - 1$ rounds the number of remaining edges is $O(n^{1+1/r})$. Finally, the algorithm recovers all remaining edges by sampling $\tilde{O}(n^{1+1/r})$ edges uniformly, which recovers all edges by a standard coupon collector argument, and augments the trees appropriately.

To guarantee the decrease in the number of edges we leverage an elegant connection to cut sparsifier contraction. The seminal work [13] showed that if we sample (and reweight) every edge with probability proportional to its strong connectivity, then the resulting graph approximates all the cuts of the original graph with high probability.

► **Definition 2.11** (Strength of an Edge). *Let $G = (V, E, w)$ be a weighted graph. We say that $S \subseteq V$ is κ -strong if the minimum cut of the induced graph $G[S]$ is at least κ . An edge $e \in E$ is κ -strong if it is contained within some κ -strong component S .*

We also need the following lemma concerning the existence of κ -strong components in a graph.

► **Lemma 2.12** (Lemma 3.1 in [12]). *Every weighted graph H with n vertices and total edge weight at least $\kappa(n - 1)$ must have a κ -strong component.*

We can now detail the process of edge reduction of Lemma 2.10. Let G be a graph with m edges and n vertices. Observe that after one round of sampling, the algorithm contracts every $(m/n^{1+1/r})$ -strong component of G , and thus by Lemma 2.12 the number of edges remaining in the graph is at most $m/n^{1/r}$. Let $C \subseteq V$ be a maximal $\Omega(m/n^{1+1/r})$ -strong component of G , and assume the algorithm samples every edge of the graph with probability $p_e \approx k/\kappa$. In expectation the algorithm samples at least $p_e \cdot m/n^{1+1/r} \approx k$ edges crossing each cut $S \subseteq C$, which also holds with high probability by standard concentration and cut counting results. Therefore, for every $u, v \in C$, the minimum cut in the sampled subgraph has at least k edges, hence they are connected in k edge disjoint forests and will be contracted. In conclusion, in every iteration the algorithm contracts all $(m/n^{1+1/r})$ -strong components of the graph, and the number of edges in the graph decreases by a factor of $n^{1/r}$.

Proof of Theorem 1.4. The algorithm begins by performing a 2-out contraction on G to obtain a graph G' with $\tilde{O}(n/\delta(G))$ vertices which preserves each non-trivial minimum cut of G with constant probability by Theorem 2.8. In parallel, the algorithm learns $\delta(G)$ by querying the degree of each vertex. Then, it packs $k = \delta(G)$ trees $\{T_i\}_{i=1}^k$ in G' using Lemma 2.10. Notice that since the minimum cut of G is at most $\delta(G)$ and each edge has weight at least 1 then $\delta(G)$ trees in G' will include all edges in the minimum cut. Hence, returning the minimum between $\delta(G)$ and the minimum cut of $H = (V, E_H = \cup_i T_i)$ will yield the minimum cut of G with constant probability. To amplify the success probability we repeat the algorithm $O(\log n)$ times in parallel and return the minimum cut found.

To analyze the query complexity notice that by Lemma 2.1, sampling the edges for the 2-out contraction takes $\tilde{O}(n)$ queries in one round. Hence, the algorithm constructs G' in $\tilde{O}(n)$ queries in one round. Then, it packs $k = \delta(G)$ trees in G' , which has only $\tilde{O}(n/k)$ vertices, using $\tilde{O}(\delta(G)r(n/\delta(G))^{1+1/r})$ queries in $2r$ rounds by Lemma 2.10. Hence, overall the algorithm require $\tilde{O}(\delta(G)^{-1/r}rn^{1+1/r})$ queries in $2r + 1$ rounds.

We conclude the proof by analyzing the success probability of the algorithm. Our implementation of the 2-out contraction succeeds with probability $1 - n \cdot n^{-4}$ since it only requires sampling $O(n)$ edges using Lemma 2.1. Then, each tree packing succeeds with probability $1 - 2n^{-2}$ by Lemma 2.10. By a union bound, the entire algorithm succeeds with probability $1 - n^{-1}$, which concludes the proof of Theorem 1.4. \blacktriangleleft

2.4 Weighted Minimum Cut in $O(r)$ rounds

In this section we give an overview of our algorithm for minimum cut in a weighted graph with low adaptivity. It is an adaptation of the minimum cut algorithm of [31], which reduces the problem to constructing a cut sparsifier and solving a monotone matrix problem which is easily parallelizable. The main difficulty lies within the construction of a cut sparsifier; while previous work in the cut-query model already provided cut sparsifiers [33, 32], its core step, weight-proportional edge sampling, uses $O(\log n)$ rounds. Our main technical contribution here is our two-round weight-proportional edge sampling primitive (Lemma 2.2), which enables the construction of a $(1 \pm \epsilon)$ -cut-sparsifier in $3r$ rounds and the completion of the entire reduction within $O(r)$ rounds.

The algorithm of [31] for finding a minimum cut in a weighted graph is actually based on the 2-respecting minimum-cut framework of [27]; where a spanning tree $T \subseteq E$ is called *2-respecting* for a cut $C \subseteq E$ if $|C \cap T| \leq 2$. The main insight here is that a given a spanning tree T that is 2-respecting for a minimum cut C , there are only $O(n^2)$ cuts that one needs to check to find a minimum cut, and they correspond to all pairs of edges in T . Originally [27], proposed a clever dynamic algorithm to calculate all these cuts quickly. In contrast, the algorithm of [31] leverages the structure of the 2-respecting spanning tree to show that the number of potential cuts is actually much smaller; restricting attention to sub-trees, $\{T_i\}$, that are all paths and whose total size is bounded. Furthermore, when T is a path the task of finding a minimum cut reduces to the following problem.

► Definition 2.13 (Monotone Matrix Problem). *Let $A \in \mathbb{R}_+^{a \times a}$ be a matrix. For every column j denote the first row in which j attains its minimum value by j_s and the last row in which j attains its minimum value by j_t . The matrix is called *monotone* if for every $i > j$ we have $j_t \leq i_s$; i.e. the row in which the minimum value is found increases monotonically as we move to the right.*

The monotone matrix problem is to find the minimum value of a monotone matrix (of dimension $a \times a$) while querying the value of as few entries as possible.

In the context of the minimum cut problem, each entry in the matrix corresponds to a cut in G and can be recovered using a single cut query. We summarize some results from [31] in the following lemma, whose proof is provided in the full version of the paper.

► Lemma 2.14. *Assume there is an algorithm to construct a $(1 \pm \epsilon)$ -cut-sparsifier for a graph G on n vertices using $q_S(n)$ queries in $r_S(n)$ rounds, and an algorithm that solves the monotone matrix problem on instances of size n using $q_M(n)$ queries in $r_M(n)$ rounds. If in addition the function q_M is convex, then it is possible to find a minimum cut of G using $\tilde{O}(q_S(n) + q_M(n) + n)$ cut queries in $r_S(n) + r_M(n) + 2$ rounds.*

Therefore, in order to find a minimum cut it suffices to construct a cut sparsifier and then solve the monotone matrix problem. Our main technical contribution in this section is a construction of a cut sparsifier with low adaptivity. We begin by formally defining the notion of a cut sparsifier.

► **Definition 2.15.** Let $G = (V, E, w)$ be a weighted graph and let $H = (V, E_H, w_H)$ be a weighted subgraph of G . We say that H is a $(1 \pm \epsilon)$ -cut-sparsifier of G if,

$$\forall S \subseteq V, \quad \text{cut}_H(S) \in (1 \pm \epsilon) \text{cut}_G(S).$$

► **Lemma 2.16.** Given a weighted graph $G = (V, E, w)$ on n vertices with integer edge weights bounded by W , and a parameter $r \in \{1, 2, \dots, \log n\}$, one can construct a $(1 + \epsilon)$ -cut-sparsifier of G using $\tilde{O}(\epsilon^{-2} r n^{1+(1+\log_n W)/r})$ cut queries and $3r + 3$ rounds. The algorithm is randomized and succeeds with probability $1 - n^{-2}$.

We now sketch the proof of Lemma 2.16, for more details see the full version of the paper. The sparsifier construction is similar to the one in [33, 32] but modified to work in r rounds. It follows the sampling framework of [13], that shows that if one samples every edge in a graph with probability proportional to its strong connectivity, then the sampled graph is a $(1 \pm \epsilon)$ -cut-sparsifier with high probability. The algorithm is based on iteratively finding the strongest components of the graph using weight-proportional edge sampling, assigning a strength estimate to all their edges, and contracting them. It repeats this process until all vertices are contracted. Finally, it samples all the edges in the graph according to their strength estimates, and returns the sampled edges as the sparsifier.

Our algorithm improves on the round complexity of the existing constructions in two ways: the first one is the adoption of a two-round weight-proportional sampling algorithm that eliminates an $O(\log n)$ factor in the round complexity. The second improvement is in the number of contraction steps performed. The algorithm of [33, 32] uses $O(\log(nW))$ steps, where in each step strengths are approximated within factor 2. Our algorithm uses instead r steps, where each step contracts all components with strength within an $n^{(1+\log_n W)/r}$ factor. Since the maximum component strength is $O(n^{1+\log_n W})$, the algorithm can contract all the components within r steps.

Our second technical contribution is an algorithm that solves the monotone matrix problem in few rounds, the proof is provided in the full version of the paper.

► **Lemma 2.17.** There exists an algorithm that solves the monotone matrix problem on instances of size n using $\tilde{O}(n^{1+1/r})$ queries in r rounds.

The algorithm for the monotone matrix problem in [31] uses a divide-and-conquer approach. At each step the algorithm reads the entire middle column j of the matrix A and finds the minimum value in the column. Then, it splits the matrix into two submatrices $A_L = A[1, \dots, j_s; 1, \dots, j]$ and $A_R = A[j_t, \dots, n; j+1, \dots, n]$ based on the location of the minimum value in the column. The algorithm is guaranteed to find the minimum value in each column by the monotonicity property of the matrix. Finally, it is easy to verify that as the size of the problem decreases by factor 2 in each iteration, the overall complexity of the algorithm is $\tilde{O}(n)$.

Unfortunately, the algorithm of [31] requires $O(\log n)$ rounds to complete all the recursion calls. Our algorithm instead partitions the matrix into $n^{1/r}$ blocks and then recurses on each block, which requires a total of $O(n^{1+1/r})$ queries in r rounds.

To conclude this section, notice that Theorem 1.5 follows by combining Lemma 2.14, Lemma 2.16, and Lemma 2.17.

2.5 Open Questions

Optimal Tradeoff Between Rounds vs Queries Complexity. An interesting open question is whether our constructive results have matching lower bounds or, alternatively, if better algorithms exist. Currently, there are no known query lower bounds even for algorithms

that use a single round of queries. We note that connectivity can be solved in one round with $\tilde{O}(n)$ queries [4], however it remains unclear whether this can be extended to finding a minimum cut, or whether there exists a lower bound of $\Omega(n^{1+\epsilon})$ cut queries for one round algorithms.

Adaptivity of Deterministic Algorithms. A deterministic algorithm for finding a minimum cut in simple graphs using $\tilde{O}(n^{5/3})$ cut queries was recently shown in [2], however its round complexity is polynomial in n , and it seems challenging to achieve polylogarithmic round complexity. In comparison, graph connectivity can be solved deterministically using $\tilde{O}(n^{1+1/r})$ cut queries in $O(r)$ rounds [4],⁶ though these techniques do not seem to extend to finding a minimum cut.

References

- 1 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPICs*, pages 2:1–2:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ESA.2022.2.
- 2 Aditya Anand, Thatchaphol Saranurak, and Yunfan Wang. Deterministic edge connectivity and max flow using subquadratic cut queries. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 124–142. SIAM, 2025. doi:10.1137/1.9781611978322.4.
- 3 Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 507–518. IEEE, 2022. doi:10.1109/FOCS54457.2022.00055.
- 4 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPICs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.7.
- 5 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 265–276. ACM, 2019. doi:10.1145/3313276.3316361.
- 6 Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *CoRR*, abs/2109.02115, 2021. arXiv:2109.02115.
- 7 Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone submodular maximization in exponentially fewer iterations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, pages 2359–2370, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/a42a596fc71e17828440030074d15e74-Abstract.html>.
- 8 Eric Balkanski, Aviad Rubinfeld, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 283–302. SIAM, 2019. doi:10.1137/1.9781611975482.19.
- 9 Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1138–1151. ACM, 2018. doi:10.1145/3188745.3188752.

⁶ The result is for the weaker Bipartite Independent Set (BIS) query model, which can be simulated using cut queries.

- 10 Eric Balkanski and Yaron Singer. A lower bound for parallel submodular minimization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 130–139. ACM, 2020. doi:10.1145/3357713.3384287.
- 11 Paul Beame, Sarel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi:10.1145/3404867.
- 12 András A. Benczúr and David R. Karger. Approximating $s - t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.
- 13 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. doi:10.1137/070705970.
- 14 Nader H. Bshouty and Hanna Mazzawi. Reconstructing weighted graphs with minimal query complexity. *Theor. Comput. Sci.*, 412(19):1782–1790, 2011. doi:10.1016/J.TCS.2010.12.055.
- 15 Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theor. Comput. Sci.*, 417:23–35, 2012. doi:10.1016/J.TCS.2011.09.005.
- 16 Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. A polynomial lower bound on the number of rounds for parallel submodular function minimization. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 37–48. IEEE, 2021. doi:10.1109/FOCS52979.2021.00013.
- 17 Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Improved lower bounds for submodular function minimization. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 245–254. IEEE, 2022. doi:10.1109/FOCS54457.2022.00030.
- 18 Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Parallel submodular function minimization. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/d8a7f2f7e346410e8ac7b39d9ff28c4a-Abstract-Conference.html.
- 19 Deeparnab Chakrabarty and Hang Liao. A query algorithm for learning a spanning forest in weighted undirected graphs. In *International Conference on Algorithmic Learning Theory 2023*, volume 201 of *Proceedings of Machine Learning Research*, pages 259–274. PMLR, 2023. URL: <https://proceedings.mlr.press/v201/chakrabarty23a.html>.
- 20 Lin Chen, Moran Feldman, and Amin Karbasi. Unconstrained submodular maximization with constant adaptive complexity. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 102–113. ACM, 2019. doi:10.1145/3313276.3316327.
- 21 Yixin Chen, Tommoy Dey, and Alan Kuhnle. Best of both worlds: Practical and theoretically optimal submodular maximization in parallel. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021*, pages 25528–25539, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/d63fbf8c3173730f82b150c5ef38b8ff-Abstract.html>.
- 22 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 749–758. ACM, 2008. doi:10.1145/1374376.1374484.
- 23 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/J.JALGOR.2003.12.001.
- 24 Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 274–282. SIAM, 2019. doi:10.1137/1.9781611975482.18.

- 25 Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 255–273. SIAM, 2019. doi:10.1137/1.9781611975482.17.
- 26 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1260–1279. SIAM, 2020. doi:10.1137/1.9781611975994.77.
- 27 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.
- 28 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.
- 29 Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 939–958. SIAM, 2021. doi:10.1137/1.9781611976465.59.
- 30 Hang Liao and Deeparnab Chakrabarty. Learning spanning forests optimally in weighted undirected graphs with CUT queries. In *International Conference on Algorithmic Learning Theory*, volume 237 of *Proceedings of Machine Learning Research*, pages 785–807. PMLR, 2024. URL: <https://proceedings.mlr.press/v237/liao24b.html>.
- 31 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.
- 32 Orestis Plevrakis, Seyoon Ragavan, and S. Matthew Weinberg. On the cut-query complexity of approximating max-cut. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*, volume 297 of *LIPICs*, pages 115:1–115:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ICALP.2024.115.
- 33 Aviad Rubinfeld, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018*, 2018. doi:10.4230/LIPICs.ITCS.2018.39.
- 34 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPICs*, pages 94:1–94:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.94.