Computing Exact Minimum Cuts Without Knowing the Graph

Aviad Rubinstein[*1], **Tselil Schramm**[†2], **and S. Matthew Weinberg**[‡3]

1 **Department of Computer Science, Harvard University, Cambridge, MA, USA**
aviad@seas.harvard.edu
2 **Department of Computer Science, UC Berkeley, Berkeley, CA, USA**
tschramm@cs.berkeley.edu
3 **Department of Computer Science, Princeton University, Princeton, NJ, USA**
smweinberg@princeton.edu

Abstract

We give query-efficient algorithms for the global min-cut and the s-t cut problem in unweighted, undirected graphs. Our oracle model is inspired by the submodular function minimization problem: on query $S \subset V$, the oracle returns the size of the cut between S and $V \setminus S$.

We provide algorithms computing an exact minimum s-t cut in G with $\widetilde{O}(n^{5/3})$ queries, and computing an exact global minimum cut of G with only $\widetilde{O}(n)$ queries (while learning the graph requires $\widetilde{\Theta}(n^2)$ queries).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Query complexity, minimum cut

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.39

1 Introduction

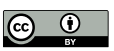
We give new algorithms for the minimum cut and s-t minimum cut problems in an unweighted, undirected graph $G = (V, E)$. Our algorithms do not assume access to the entire graph G; rather, they only interact with an oracle that, on query S, returns the value $c(S)$ of the cut between S and $V \setminus S$. Our goal is to minimize the number of queries to the oracle while computing (exact) optimum cuts.

Three easy algorithms

How many queries should we expect to be necessary? It is not hard to see that $\binom{n}{2}+n = O(n^2)$ queries suffice:[1] To find out whether there is an edge between u and v, we can query the oracle for $\{u\}$, $\{v\}$, and $\{u, v\}$. The edge is present iff $c(\{u\}) + c(\{v\}) - c(\{u, v\}) > 0$. After querying all n singletons and $\binom{n}{2}$ pairs, we have learned the entire graph. In fact, we can

* A. Rubinstein did most of the work while at UC Berkeley and a visitor at the Simons Institute for the Theory of Computing, supported by a Microsoft PhD Fellowship, as well as a Rabin Postdoctoral Fellowship, NSF grant CCF1408635, and Templeton Foundation grant 3966.
† T. Schramm is grateful for the support of an NSF Graduate Research Fellowship (1106400).
‡ S. M. Weinberg is grateful for support from NSF CCF-1717899. This work was completed in part while S. M. Weinberg was a research fellow at the Simons Institute for the Theory of Computing.

[1] We follow the standard convention and use $n = |V|$ to denote the number of vertices and $m = |E|$ to denote the number of edges. We also use $\widetilde{O}(x)$ to denote $O(x \cdot \text{polylog}(x))$, and similarly for $\tilde{\Omega}(\cdot)$ and $\tilde{\Theta}(\cdot)$.



improve slightly: the results of the cut queries are linear in $\binom{n}{2}$ unknown variables; thus $\binom{n}{2}$ linearly independent queries suffice.

For sparse graphs we can do even better: one can find a neighbor of a non-isolated vertex v in $O(\log n)$ queries using a "lion in the desert" algorithm (see Lemma 3). More generally, we can learn the entire graph using $\widetilde{O}(n+m)$ queries. But for dense graphs, $\widetilde{\Omega}(n^2)$ queries are necessary to learn the entire graph by a simple information theoretic argument (each query reveals at most $O(\log n)$ bits).

The search for a lower bound

Because $\widetilde{\Omega}(n^2)$ queries are needed to learn the graph, it is natural to conjecture that $\widetilde{\Theta}(n^2)$ is the optimal query complexity for computing the min cut. Such a lower bound would be of considerable interest: after breakthrough progress in recent years, $\widetilde{O}(n^2)$ is also the state of the art query complexity for the more general problem of submodular function minimization over subsets of n items [22].[2] Recent work of [6] indeed rules out certain kinds of algorithms with subquadratic query complexity, but determining whether submodular minimization requires $\widetilde{\Omega}(n^2)$ queries remains an exciting open problem (see Section 1.1 for more discussion), and graph cuts seemed like a promising candidate.

Our results

In defiance of our intuition, we provide algorithms for global minimum cut and minimum s-t cut that use a truly subquadratic number of queries. Our main results are:

▶ **Theorem 1** (Global Min Cut). *There exists a randomized algorithm that with high probability computes an exact global minimum cut in simple graphs using $\widetilde{O}(n)$ queries.*

▶ **Theorem 2** (Min s-t Cut). *There exists a randomized algorithm that with high probability computes an exact s-t minimum cut in simple graphs using $\widetilde{O}(n^{5/3})$ queries.*

It is worth mentioning that while our focus is query efficiency, all our algorithms run in polynomial time ($\widetilde{O}(n^2)$ or faster).

Techniques

All our algorithms are quite simple. Both results can be obtained using the following "meta-algorithm": (1) subsample a subquadratic number of edges; (2) compress the (original) graph by contracting all "safe" edges (i.e. those that do not cross the optimum cut, with high confidence, based on the subsample); and (3) learn all remaining edges.

Uniform sampling does not work for Step (1). We build on the *edge strength*-based sampling due to Benczúr and Karger [4], which in $\widetilde{O}(m)$ time yields graphs with few edges that approximate every cut well. Calculating the edge-strengths with $o(m)$ queries is non-trivial. Instead of computing the strength of every edge, we sub-sample the graph at different resolutions to classify the vertices of the graph into strongly connected components. See Section 3 for details.

For the global minimum cut problem we also provide an even simpler algorithm, which avoids edge-strength sampling: in a preprocessing step, we contract edges uniformly at

[2] Note that the more recent work of [6] requires $\tilde{O}(nM^3)$ queries, where the function is integral and M is the maximum value the function takes; for cuts in graphs, $M = n^2/4$!

random, as in Karger's algorithm [16]. After the right number of edge contractions, it suffices to sample edges uniformly at random in Step (1).

1.1 Related work

Graph cut minimization is a classical algorithms topic, with work dating back to Ford and Fulkerson [11], and too many consequent results to list. Of particular relevance to the present paper are the works of Karger and co-authors, including [16, 17, 19, 4, 18], which give randomized algorithms for computing minimum cuts and related quantities efficiently—in particular, this line of work establishes methodology for randomly compressing graphs while preserving cut information, which has been used in numerous follow-up works. Though our goal is *query* efficiency rather than runtime efficiency, we very much rely on their insights.

Another work of note is the recent result of Kawarabayashi and Thorup [20], who show that the global minimum cut can be computed deterministically in $\widetilde{O}(m)$ time. Though their setting differs from ours, our works are similar in that we too require structural theorems about the number of edges participating in minimum cuts in the graph (e.g. Lemma 8).

As mentioned above, our initial motivation for studying the min cut problem in this oracle model came from submodular function minimization (SFM). SFM was first studied by Grötschel, Lovász, and Schrijver in the 1980's [13], and has since been a popular topic of study (see e.g. [12] for a thorough treatment). For a submodular function over n items, the current best general algorithm requires $\widetilde{\Theta}(n^2)$ oracle queries [22]. Furthermore, [6] suggest that $\widetilde{\Theta}(n^2)$ is indeed the right bound: they prove an $\Omega(n^2)$ lower bound on the number of oracle calls made by a restricted class of algorithms (those that access the submodular function by naively evaluating the subgradient of its Lovász extension). For general algorithms, the current best lower bound is due to Harvey [14], who shows that $(\log_3 2 - o(1))n$ queries are needed. Graph cuts and s-t cuts are canonical examples of symmetric and asymmetric submodular functions,[3] and while it would be natural to conjecture that $\widetilde{\Omega}(n^2)$ queries are needed for graph cut problems, our work demonstrates that these problems do not provide a lower bound matching [22]'s algorithm (at least in unweighted graphs, and for randomized algorithms).

Note that there are works that bypass the $\widetilde{\Theta}(n^2)$ oracle queries barrier for special cases of SFM. For example, [6] provide an algorithm with $\widetilde{\Theta}(nM^3)$ oracle queries when the function value is integral and bounded within $[1, M]$ (for min cut M may be as large as $\Theta(n^2)$). Another special case of interest are decomposable submodular functions (e.g. [27, 25]).

We also mention a sequence of papers [8, 23, 5] which study the query efficiency of *learning* a graph under a similar query model (in which each cut query can be implemented in $O(1)$ queries). This series of papers establishes that $\Theta(m \log(n^2/m)/\log m)$ queries suffice to learn a graph on n vertices and m edges. This improves upon our naive algorithm for learning a graph (see Lemma 3) by polylogarithmic factors.

To our knowledge, no other works have previously considered the query complexity of graph cuts. However, the task of compressing graph cut information into efficient structures has been studied before from a variety of angles: sketching [2, 21], spectral sparsifiers [3], streaming spectral sparsifiers [15], skeletons [17, 4], backbones [7], and cactus representations [9, 24], to list a few. Note that an overwhelming majority of these works necessarily lose

[3] A submodular function is symmetric if $f(S) = f(\bar{S})$ for all S. $\emptyset$ and $[n]$ are always minimizers of a symmetric submodular function, so the "symmetric submodular function minimization problem" is to find a non-trivial minimizer (i.e. the minimizer $\notin \{\emptyset, [n]\}$), of which global min cut is a special case.

some (small) approximation factor through compression, and exact solutions are rare, but exist (e.g. [1]).

There is also an indirect connection between our work and lower bounds for distributed graph algorithms (e.g. [26, 10]), since our algorithms can be used to obtain upper bounds on the two-party communication complexity of min cuts in some models.[4]

1.2 Organization

In Section 2, we present our simple algorithm for global min cut, as well as important algorithmic primitives (such as subsampling edges). Then in Section 3, we introduce our query-efficient implementation of Benczúr and Karger's edge-strength based sampling, after which we demonstrate its application to global min cut in Section 4. Finally, Section 5 contains our result for min s-t cuts.

1.3 Discussion and Future Work

The main take-home message of our work is simple, randomized algorithms for *exact* global and s-t min cut with $\tilde{O}(n)$ and $\tilde{O}(n^{5/3})$ queries, respectively. In particular, our algorithm for global min cut learns (up to the polylog factors) just enough information to even specify one of the 2^n distinct cuts, and both are well below $\tilde{\Theta}(n^2)$. So in this natural oracle model, it is possible to find the exact global and s-t cut *without learning the underlying graph.*

Our work also motivates numerous directions for future work: Are weighted or directed graph cuts computable in $o(n^2)$ queries? Do deterministic min cut algorithms exist with truly subquadratic queries? Or can graph cuts still provide a $\Omega(n^2)$ submodular-function-minimization lower bound (perhaps for deterministic algorithms)? While graph cuts are indeed a very special case of submodular functions, can any of the ideas from our work be used in randomized algorithms for a broader class of submodular function minimization?

2 Global min-cut in $\widetilde{O}(n)$ queries

We begin by observing that if G has m edges, we can learn G entirely with $\widetilde{O}(m)$ queries. This is because locating a single edge takes only $O(\log n)$ time.

▶ **Lemma 3** (Learning an edge with $O(\log n)$ queries). *We can learn one neighbor of a vertex $v \in V$ in $O(\log n)$ queries.*

Proof. To find one neighbor of v, we perform the following recursive procedure: we partition $V \setminus v$ into two sets S_1 and S_2 of sizes $\lfloor \frac{n-1}{2} \rfloor, \lceil \frac{n-1}{2} \rceil$, respectively. We then query the cut values of $\{v\}$, S_i, and $S_i \cup \{v\}$, from which we can infer how many neighbors v has in S_i, for $i \in \{1, 2\}$ $\left(\left(c(\{v\}) + c(S_i) - c(S_i \cup \{v\})\right)/2\right)$. If v has no neighbors, return "no neighbors". Otherwise, if v has a neighbor in S_1, then proceed recursively in S_1; otherwise proceed recursively in S_2. ◀

The above observation suffices to learn the entire graph with $\widetilde{O}(m)$ queries, as it is easy to modify the algorithm to ignore known neighbors of v (if S_1 or S_2 contain only known neighbors of v, ignore them). If $m = \widetilde{O}(n)$, Theorem 1 follows easily.

[4] In a model where Alice and Bob can jointly compute a cut query in $O(\log n)$ communication, and have shared randomness, Theorem 1 (Theorem 2) provides a randomized protocol with $\tilde{O}(n)$ (resp. $\tilde{O}(n^{5/3})$) communication for computing the global (resp. s-t) min cut.

Otherwise, if $m \gg n$, a natural idea is to randomly subsample the edges of G until we are left with a sparse graph, and use this sparse graph to learn useful data about G. Indeed, we show that after a preprocessing step of n queries, sampling each random edge only requires $O(\log n)$ queries:

▶ **Corollary 4** (Sampling a random edge with $O(\log n)$ queries). *Given oracle access to the cut values of a graph on n vertices, after performing n initial queries we can sample a random edge in $O(\log n)$ additional queries (i.e. k uniformly random edges can be drawn in $n + O(k \log n)$ queries).*

Proof. First, as a preprocessing step, we perform n queries to determine the degree of every vertex. Now, we choose a random edge by choosing a random vertex v with probability proportional to its degree, then performing the procedure detailed in the proof of Lemma 3, but choosing to recurse on either S_1 or S_2 randomly with probability proportional to the degree of v into each set. ◀

Because we can sample random edges, we might hope to subsample G and obtain a sparse graph G' which has the same approximate cut values as G. In particular, if the minimum cut of G has value c, and we sample each edge independently with probability $\log n/c$, then the subsampled graph G' preserves all cuts with high probability within a $(\log n/c)(1 \pm \epsilon)$ factor. However, sampling with probability much smaller than $\log n/c$ will yield poor cut concentration in G'.

So if $c \approx \frac{m}{n}$, the next step in our algorithm is to do this simple uniform subsampling and work with G', which will have $\approx n \log n$ edges in expectation. But if $c \ll \frac{m}{n}$, the resulting G' will still have too many edges to learn, so we need some additional work.

Fortunately, when the minimum cut size c is small compared to the average degree, we can preprocess G to an intermediate G^* whose average degree is $\approx c$ without destroying the minimum cut via random contractions. Our preprocessing step essentially runs Karger's Algorithm [16] (reproduced here for completeness) for a well-chosen number of steps (not all the way to termination).

▶ **Algorithm 5** (Karger's Algorithm [16]).
Input: *A graph G.*
1. *For $j = 1, \ldots, n-2$:*
   (a) *Sample a random edge of G, and contract its two endpoints into a single "super-vertex".*
   (b) *Retain multi-edges, but remove self-loops.*

Output: *The cut between the two remaining super-vertices, which form a partition of G's vertices into two sets.*

In Karger's seminal paper, he proves that this algorithm finds the minimum cut in a graph with probability at least $\frac{1}{n^2}$, yielding a randomized algorithm for minimum cut.

We will not run Karger's algorithm to its completion, but rather only until there are cn total edges remaining in the graph (we can guess c within a factor of 2 at the cost of $\log n$ additional iterations). The following simple lemma shows that with constant probability, the minimum cut will survive:

▶ **Lemma 6** (Karger's Algorithm on small cuts). *Let G be a graph with minimum cut value $c > 0$. If we run Karger's algorithm on G until there are at most cn edges in the graph, then the minimum cut survives with constant probability.*

We will prove Lemma 6 in Section 2.1. Of course, we must verify that we can run T steps of Karger's algorithm with $\widetilde{O}(T)$ oracle queries:

▶ **Proposition 7.** *Given oracle access to the cut values of G, we can run T steps of Karger's algorithm using $\widetilde{O}(T)$ queries.*

Proof. The key observation is that keeping track of super-vertices requires no additional queries—it is simply a matter of treating all vertices belonging to a super-vertex as a single entity.

Each step of Karger's algorithm requires sampling a random edge, which we have already seen requires $O(\log n)$ oracle queries assuming the degree of every vertex is known. In order to keep track of the degree of super-vertices, we require only a single oracle query after every edge contraction: we ask for the cut value between the super-vertex and the remainder of the graph. ◀

At this point, our algorithm is as follows: we first run Karger's algorithm until the min cut size is comparable to the average degree, then subsample the graph to obtain a sparse graph that approximates the cuts of the original graph well. Applying concentration arguments, it's easy to see that this algorithm immediately yields an approximate min cut.

However, the following observation allows us to improve upon this, and learn the minimum cut exactly! Since the cuts in G' approximate the cuts in G well, any two nodes that are together in *every* approximate minimum cut in G' are safe to contract into a super-node (because they certainly aren't separated by the min cut). After these contractions, if there are sufficiently few edges remaining between the super-vertices, we can learn the entire remaining graph between the super-vertices and find the true minimum cut.

The following structural result shows that this is indeed the case: the total number of edges that participate in non-singleton approximately-minimum cuts is at most $O(n)$.[5]

▶ **Lemma 8** (Covering approximate min cuts with $O(n)$ edges)**.** *Let $G = (V, E)$ be an unweighted graph with minimum degree d and minimum cut value c. Let $\mathcal{C}$ be the set of all non-singleton approximate-minimum cuts in the graph, with cut value at most $c + \epsilon d$, for $\epsilon < 1$. Then $|\cup_{C\in\mathcal{C}} C|$ (the total number of edges that participate in cuts in $\mathcal{C}$) is $O(n)$.*

We remark that a similar claim is proven in [20]. While the theorem of [20] would be sufficient for our purposes,[6] our proof is extremely simple, and so we include it in Section 2.3.

This concludes our global min-cut algorithm. Below, we summarize the algorithm, and formally prove that it is correct.

▶ **Algorithm 9** (Global Min Cut with $\widetilde{O}(n)$ oracle queries)**.**
Input: *Oracle access to the cut values of an unweighted simple graph G.*
1. *Compute all of the single-vertex cuts.*
2. *For $c = 2^j$ for $j = 0, 1, \ldots, \log n$,*
   (a) *Repeat $\log n$ times:*
      (i) *Run Karger's Algorithm until there are a total of cn edges between the components in the graph, call the resulting graph G_1.*
      (ii) *Subsample each edge with probability $p = \frac{80 \ln}{\epsilon^2 c}$ to obtain a graph G_2 (any $\epsilon \in (0, 1/3)$ suffices)*

[5] A cut is non-singleton if each side has at least two nodes.
[6] Their result is stronger in the sense that they also show how to locate the cover in deterministic time $O(m)$, while our result is slightly simpler, and we only require $O(n)$ edges rather than $\widetilde{O}(n)$.

(iii) *Find all non-singleton cuts of size at most* $(1+3\epsilon)pc$ *in the graph, and contract any two nodes which are together in all such cuts, call the resulting graph* G_3

(iv) *Learn all of* G*'s edges between the super-vertices of* G_3 *to obtain* G_4 *(unless there are more than* $n \log n$ *edges, in which case abort and return to step 2a).*

(v) *Compute the minimum cut in* G_4*, and if it is the best seen so far, keep track of it.*

Output: *Return the best cut seen over the course of the algorithm.*

▶ **Theorem 10** (Mincut). *Algorithm 9 uses* $\widetilde{O}(n)$ *queries and finds the exact minimum cut in* G *with high probability.*

Proof. First, we will prove the correctness of the algorithm. Clearly, if one of the single-vertex cuts is the minimum cut, the algorithm finds this cut in step 1, so suppose that the best cut has value $\hat{c} < d_{\min}$, where $d_{\min}$ is the minimum degree in G.

In one of the iterations of step 2, c is within a factor of 2 of $\hat{c}$, and we focus on this iteration. In step 1, by Lemma 6, the minimum cut survives with at least constant probability. By the concentration arguments given in Lemma 11 and Corollary 13, in step 2 every cut in G_2 is close to the value of the cut in G_1 with high probability,[7] and so no edge in the minimum cut is contracted in step 3. Therefore, with constant probability, we find the minimum cut in step 5. Since we repeat this process $\log n$ times in step 2a, the total probability that we miss the global min cut in every iteration is polynomially small. This proves the correctness of the algorithm.

Now, we argue that at most $\widetilde{O}(n)$ queries are required. At every iteration of the inner loop, we run Karger's Algorithm for $O(n)$ steps ($\widetilde{O}(n)$ queries by Proposition 7). Then, we subsample each of cn edges each with probability $\widetilde{O}(1/c)$, or equivalently, we sample $\widetilde{O}(n)$ random edges ($\widetilde{O}(n)$ queries by Corollary 13). step 3 does not require any queries. By Lemma 8, step 4 requires learning only $O(n)$ edges ($\widetilde{O}(n)$ queries) if c is the true value of the minimum cut, and otherwise the step is aborted. Finally, step 5 requires no additional queries. Since the inner loop is repeated $\log^2 n$ times, this concludes the proof. ◀

In the following subsections, we provide proofs of key intermediate lemmas.

2.1 Compressing the graph with Karger's algorithm

▶ **Lemma 6** (restated). *[Karger's Algorithm on small cuts] Let* G *be a graph with minimum cut value* $c > 0$*. If we run Karger's algorithm on* G *until there are at most* cn *edges in the graph, then the minimum cut survives with constant probability.*

Proof. Fix a specific min cut C. We apply Karger's algorithm until the total number of edges drops to cn. At each step, the probability that we contract an edge from C is at most $1/n$, and we have at most n steps, so the probability that C survives is at least $(1-1/n)^n > 1/4$ for $n > 2$. ◀

2.2 Subsampling the graph

First, we show that if we sample with probability proportional to $\widetilde{O}(1/c)$, every cut in the subsampled graph has value close to its expectation. Because there are 2^n cuts, a simple

[7] where "close" means that the value of the cut in G_2 is within $(1 \pm \varepsilon)pk$, where k is the value of the cut in G_1.

Chernoff bound followed by a union bound is insufficient. Instead, we perform a more careful union bound by appealing to a polynomial bound on the number of approximately minimum cuts (as is standard in this setting, see e.g. [17]).

▶ **Lemma 11.** *Let $G = (V, E)$ be a multigraph with minimum cut value c, and let $G' = (V, E')$ be the result of sampling each edge of E with probability $p \geq \min\left(\frac{40 \ln n}{\epsilon^2 c}, 1\right)$. Then with high probability, every cut of value k in G has value $(1 \pm \epsilon)pk$ in G'.*

Proof. For each edge $e \in E$, consider the random binary variable $X_e \triangleq \begin{cases} 1 & e \in E' \\ 0 & \text{otherwise} \end{cases}$. Notice that $\mathbb{E}(X_e) = p$. Let C be a cut of size k. By a Chernoff bound, the probability that C has cut value deviating from its expectation by more than an ϵ-factor in G' is bounded by:

$$\mathbb{P}\left[\left|\sum_{e \in C^*} X_e - pk\right| > p\epsilon k\right] \leq 2\exp\left(-\frac{\epsilon^2 pk}{2}\right) \leq 2n^{-10k/c}, \tag{1}$$

where the last inequality follows by our choice of p.

Now, it follows from the analysis of Karger's algorithm (Lemma 12 below) that for every integer $\ell > 0$ there are at most $(2n)^{2\ell}$ cuts of value at most ℓc. Consider a cut C with value in $[\ell c, (\ell + 1)c]$ in G. Using (1), we have that the probability that its value in G' deviates from expectation by more than $\pm p(\ell + 1)\epsilon c$ is at most $n^{-10\ell}$. Taking a union bound over all such C and all values of ℓ the soundness holds with probability at least $1 - n^{-6}$. ◀

The following lemma, which we employed in order to bound the number of cuts of each size, is an oft-used consequence of Karger's algorithm (see e.g. [19]).

▶ **Lemma 12** (Bound on the number of small cuts)**.** *If a graph on n vertices has a minimum cut of size c, then there are at most $(2n)^{2\ell}$ cuts of size ℓc.*

Proof. Fix a specific cut C, such that $|C| = \ell c$. Consider Karger's algorithm, in which we contract a uniformly random edge in each step. After t steps, there are at least $(n - t)c/2$ edges in the graph (since no vertex can ever have degree less than c in Karger's algorithm). Then in the t-th step of Karger's algorithm there is probability at most $\frac{2\ell c}{(n-t)c}$ that an edge from C is contracted. Using a telescoping product argument, the probability that C survives for $n - 2\ell$ steps of the algorithm is at least $\prod_{t=0}^{n-2\ell}\left(1 - \frac{2\ell}{n-t}\right) = \frac{(2\ell)!}{n(n-1)\cdots(n-2\ell+1)} \geq n^{-2\ell}$. After $n - 2\ell$ steps, there are 2ℓ vertices remaining, so less than $2^{2\ell}$ cuts survived. Therefore in total there can only be $(2n)^{2\ell}$ such cuts in the original graph. ◀

As a corollary of Lemma 11, we have that the approximately minimum cuts of the subsampled graph correspond to approximately minimum cuts in the original graph:

▶ **Corollary 13.** *Let $G = (V, E)$ be a graph with minimum minimum cut value c. Let $G' = (V, E')$ be the result of sampling each edge in E with probability $p = \min\left(\frac{40 \ln n}{\epsilon^2 c}, 1\right)$, with $\epsilon \leq 1/3$. Then the following events occur with high probability:*

Completeness *the minimum cut of G has value at most $p(1 + \epsilon)c$ in G'.*

Soundness *every cut of value at most $p(1 + \epsilon)c$ in G' has value at most $(1 + 3\epsilon)c$ in G. Furthermore, no cut has value less than $p(1 - \epsilon)c$ in G'.*

Proof. This follows immediately from Lemma 11, because with high probability, every cut concentrates to within a $(1 \pm \epsilon)$ factor of its expectation. ◀

2.3 Covering approximate min cuts with $O(n)$ edges

▶ **Lemma 8** (restated). *[Covering approximate min cuts with $O(n)$ edges] Let $G = (V, E)$ be an unweighted graph with minimum degree d and minimum cut value c. Let $\mathcal{C}$ be the set of all non-singleton approximate-minimum cuts in the graph, with cut value at most $c + \epsilon d$, for $\epsilon < 1$. Then $|\cup_{C\in\mathcal{C}} C|$ (the total number of edges that participate in cuts in $\mathcal{C}$) is $O(n)$.*

Proof. Notice that any subset of $\mathcal{C}$ induces a partition over V, where two vertices are in the same component if they are on the same side of every cut. Let $K = C_1, \ldots, C_k$ be a minimal subset of $\mathcal{C}$, such that no additional cut $C \in \mathcal{C}$ splits any existing components into two components both of size $\geq \beta d$ when added to K. By definition, $k \leq n/\beta d$ and $|\cup_{i=1}^{k} C_i| \leq (c + \epsilon d) \cdot k$. The number of vertices with at least αd incident edges in K is therefore at most $2ck/\alpha d$. Call this set of vertices S.

Now, by definition of K, adding any other cut $C \in \mathcal{C}$ (not already covered by the edges in K) can only split an existing component if the size of at least one of them, B is small, $|B| < \beta d$. We argue that $B \subseteq S$. Suppose by contradiction that there exists a vertex $v \in B \setminus S$. Then v cannot have more than βd edges to other nodes inside B, because B is small. Since $v \notin S$, v can have at most αd edges that are already in the cuts $C_1, \ldots, C_k$. Thus, v has at least $\deg(v) - d(\alpha + \beta)$ edges crossing the new cut.

Since the new cut is not a singleton cut, we can move v to the other side of the cut, decreasing the size of the cut by $\deg(v) - 2d(\alpha + \beta))$.

We have that if $2(\alpha + \beta) < 1 - \epsilon$,

$$\deg(v) - 2d(\alpha + \beta) \geq d(1 - 2(\alpha + \beta)) > \epsilon d,$$

which is a contradiction, since this would give a cut of value less than c. So choosing $\alpha = \beta < (1 - \epsilon)/4$, we can conclude that S is the only set of vertices which will be separated into additional components if we refine our partition by adding minimum cuts from $\mathcal{C}$.

Therefore, the set $C_1, \ldots, C_k$ and all edges incident on S cover all edges participating in any non-singleton minimum cut. The cover consists of at most

$$(c + \epsilon d)k + |S| \cdot d \leq \frac{n(c + \epsilon d)}{\beta d} + \frac{2cn}{\alpha\beta d}$$

edges, so the conclusion follows. ◀

3 Connectivity-preserving sampling in the oracle model

Now we show how to subsample a graph with arbitrary connectivity to obtain a sparse graph in which all cut values are well-approximated (also known as a *sparsifier*). The algorithm and analysis are inspired by [4], but we must make modifications to both in order to optimize query efficiency. We begin with some definitions.

▶ **Definition 14.** A graph G is *k-strongly-connected* if there is no cut of size less than k in G. The *strong connectivity* of G, denoted $K(G)$, is the size of G's minimum cut.

▶ **Definition 15.** Given a graph $G = (V, E)$ and an edge $e = (u, v) \in E$, define e's *strength* k_e to be the maximum of the strong connectivities over all vertex-induced subgraphs of G containing e:

$$k_e = \max_{S\subseteq V \,:\, u,v\in S} K(G[S]),$$

where $G[S]$ denotes the vertex-induced subgraph of G on S.

The following theorem, due to Benczúr and Karger, shows that if we sample each edge with probability inversely proportional to its strength, every cut will be well-preserved.

▶ **Theorem 16** (Benczúr and Karger [4]). *Let $G = (V, E)$ be an unweighted graph. For each edge $e \in E$, let k_e denote the edge strength of e. Suppose we are given $\{k'_e\}_{e \in E}$ such that $\frac{1}{4}k_e \leq k'_e \leq k_e$. Let H be the graph formed by sampling each edge e with probability*

$$p_e = \min\left(\frac{100 \ln n}{k'_e \epsilon^2}, 1\right),$$

and then including it with weight $1/p_e$. Then with high probability, H has $O(n \ln n/\epsilon^2)$ edges, and every cut in H has value $(1 \pm \epsilon)$ of the original value in G.

While Benczúr and Karger give efficient algorithms for computing approximate edge strengths when the graph is known, in our setting we cannot afford to look at every edge. The following algorithm shows how to compute approximate edge strengths, and how to compute the sparsifier H, with $\widetilde{O}(n/\epsilon^2)$ oracle queries.

▶ **Algorithm 17** (Approximating Edge Strengths (and sampling a sparsifier H)).
Input: *An accuracy parameter ϵ, and a cut-query oracle for graph G.*
1. *(Initialize an empty graph H on n vertices).*
2. *For $j = 0, \ldots, \log n$, set $\kappa_j = n2^{-j}$ and:*
   - (a) *Subsample G' from G by taking each edge of G with probability $q_j = \min(100 \cdot 40 \cdot \frac{\ln n}{\kappa_j}, 1)$*
   - (b) *In each connected component of G':*
     - (i) *While there exists a cut of size $\leq q_j \cdot \frac{4}{5}\kappa_j$, remove the edges from that cut, and then recurse on the two sides. Let the connected components induced by removing the cut edges be $C_1, \ldots, C_r$.*
     - (ii) *For every $i \in [r]$ and every edge (known or unknown) with both endpoints in C_i, set the approximate edge strength $k'_e := \frac{1}{2}\kappa_j$ (alternatively, subsample every edge in $C_i \times C_i$ with probability $2q_j/\epsilon^2$ and add it to H with weight $\epsilon^2/2q_j$).*
     - (iii) *Update G by contracting C_i for each $i \in [r]$.*

Output: *The edge strength approximators $\{k'_e\}_{e \in E}$ (or the sparsifier H).*

▶ **Theorem 18.** *For each edge $e \in G$, the approximate edge strength given in Algorithm 17 is close to the true edge strength, $\frac{1}{4}k_e \leq k'_e \leq k_e$. Furthermore, the algorithm requires $\widetilde{O}(n/\epsilon^2)$ oracle queries to produce the sparsifier H, which satisfies:*
- *H has $O(n \ln n/\epsilon^2)$ edges*
- *The maximum weight of any edge e in H will be $O(\epsilon^2 k_e / \ln n)$*
- *Every cut in H is within a $(1 \pm \epsilon)$-factor of its value in G.*

Proof. The proof follows from two claims, which we state here and prove later:

▶ **Claim 19.** *At iteration $j = \lceil \log(n/k_e) \rceil$, the edge e is either assigned $k'_e = \frac{1}{2}\kappa_j = n/2^{j+1} \geq k_e/4$ or has already been assigned a larger value of k'_e.*

▶ **Claim 20.** *At iteration j, no edges e with $k_e < \frac{1}{2}\kappa_j$ are assigned a strength approximation.*

Given these two claims, we have that the approximate edge strength of every edge is within a factor of two of the true strength. Furthermore, to construct H, each iteration only requires $\widetilde{O}(n/\epsilon^2)$ cut queries. In step 2a, all components with strong connectivity larger than the current connectivity (κ_j) have been contracted, so there are no $2\kappa_j$-connected components. By Corollary 22 (stated shortly), the current G therefore has at most $O(n\kappa_j)$ edges. Therefore, in step 2a we have $q_j = \widetilde{O}(n/|E|)$, and the expected number of sampled

edges is therefore just $\widetilde{O}(n)$, and this step requires only $\widetilde{O}(n)$ cut queries. The operations in step 1 require no additional queries. Finally, again by Corollary 22, step 2 requires at most $\widetilde{O}(n/\epsilon^2)$ queries, and the consequent step requires no samples. The whole process is iterated $O(\log n)$ times, for a total of $\widetilde{O}(n/\epsilon^2)$ queries. The listed properties of H follow from Theorem 16.

Now, we prove our initial claims.

To prove Claim 19, consider the strongly connected component of strength k_e that e belongs to, C_e. Since we subsample edges with probability $q_j = 100 \cdot 40 \cdot \ln n/\kappa_j \geq 100 \cdot 40 \ln n/k_e$, with high probability every cut of C_e has size at least $\frac{9}{10} q_j k_e \geq \frac{9}{10} q_j \kappa_j$in G' (by concentration bounds identical to those in Lemma 11). Therefore, no minimum cut removed in step 1 will disconnect C_e. The claim follows.

To prove Claim 20, we note that by definition if $k_e < n/2^{j+1}$, then e cannot participate in any vertex-induced component with strong connectivity $\kappa_j/2$. We will prove that every component $C_1, \ldots, C_r$ created in step 1 is at least $(\kappa_j/2)$-connected. For this, it is necessary to prove that any cut of size less than $\kappa_j/2$ is removed. Let $C = \cup C_i$ be the components of G' after step 2a. First, we notice that at most n cuts in C are necessary to remove all non-strongly-connected edges. Let $S_1, \ldots, S_\ell$ be a sequence of at most $\ell \leq n$ cuts with sizes $a_1, \ldots, a_\ell$ respectively, so that $a_i \leq \kappa_j/2$ in G when restricted to the vertex-induced subgraph given by the vertices of C. Let $a'_1, \ldots, a'_\ell$ be the sizes of the cuts $S_1, \ldots, S_\ell$ in C (in the subsampled graph G').

By a Chernoff bound,

$$\mathbb{P}[a'_i - q_j a_i \geq s \cdot q_j a_i] \leq \begin{cases} \exp\left(-s q_j a_i/3\right) & s \geq 1 \\ \exp\left(-s^2 q_j a_i/3\right) & s \leq 1 \end{cases}$$

We choose $s = \frac{4}{5}\frac{\kappa_j}{a_i} - 1$ so that $(1+s) q_j a_i = q_j \cdot \frac{4}{5}\kappa_j$. Then because $a_i \leq \kappa_j/2$,

$$s q_j a_i = q_j \cdot \frac{4}{5} \cdot \kappa_j - q_j a_i \geq q_j \cdot \frac{3}{10} \cdot \kappa_j \geq 30 \ln n,$$

and and because $a_i \leq \kappa_j/2$, $s \geq \frac{3}{5}$, so

$$s^2 q_j a_i \geq 18 \ln n.$$

Thus, the probability that any of the cuts S_i has size $a'_i \geq \frac{4}{5} q_j \kappa_j$ in the subsampled graph G' is at most n^{-6}. Taking a union bound over all of the S_i, we have that with high probability, all of the S_i will be small enough in the subsampled graph to be removed. ◀

To argue that we did not sample too many edges (or require too many oracle queries) in step 2a, we must bound the number of edges with strength at least k and at most $2k$. The following lemma is the crux of the argument (this lemma is not novel and has appeared elsewhere, e.g. [4]).

▶ **Lemma 21.** *Let $G = (V, E)$ be a weighted graph without self-loops, and let $|V| = n$. Denote by $w(E)$ the total weight of the edges in E. If $w(E) \geq d(n-1)$, then G contains a strongly d-connected component.*

Proof. The proof is by induction—if $n = 2$, the conclusion is obvious. Now, by contradiction, let n be the smallest integer for which this is not the case. Since G is not d-connected, by removing a set of edges of total weight $< d$, we can split G into two components C_1, C_2 of

size n_1 and n_2 with edge sets E_1 and E_2, so that the total weight of edges among the two parts is at least $w(E_1) + w(E_2) \geq d(n-2)+1$. Since G and all of its induced subgraphs have no d-strongly-connected subgraphs, by the induction hypothesis both C_1 and C_2 must have $w(E_1) \leq d(n_1-1)$ and $w(E_2) \leq d(n_2-1)$. But then $w(E_1)+w(E_2) \leq d(n_1+n_1-2) = d(n-2)$, which is a contradiction. This completes the proof. ◀

▶ **Corollary 22.** *In an graph on n vertices which has strong connectivity k and no components with strong connectivity $\geq 2k$, there are $\Theta(nk)$ edges.*

Proof. In a strongly k-connected component, every vertex must have degree at least k, which gives the lower bound. To see the upper bound, we invoke Lemma 21 (which gives the desired conclusion by taking $d = 2k$). ◀

4 Global min-cut revisited

Now that we are in possession of a more sensitive sampling algorithm, we give a simplified global min cut algorithm ("simplified" by pushing all the complexity to the sampling procedure).

▶ **Algorithm 23** (Simpler global Min Cut with $\widetilde{O}(n)$ oracle queries)**.**
Input: *Oracle access to the cut values of an unweighted simple graph G.*
1. *Compute all of the single-vertex cuts.*
2. *Compute a sparsifier H of G using Algorithm 17 with G and with small constant ϵ.*
3. *Find all non-singleton cuts of size at most $(1+3\epsilon)$ times the size of the minimum cut in H, and contract any edge which is not in such a cut, call the resulting graph G'.*
4. *If the number of edges between the super-vertices of G' is $O(n)$, learn all of the edges between the super-vertices of G', and compute the minimum cut.*

Output: *Return the best cut seen over the course of the algorithm.*

▶ **Theorem 24.** *Algorithm 23 uses $\widetilde{O}(n)$ queries and finds the exact minimum cut in G with high probability.*

Proof. Let C^* be a minimum cut in G, and suppose the size of C^* is c. By Theorem 18, the sampling performed in step 2 will ensure that with high probability the minimum cut of H has value at least $(1-\epsilon)c$, and that the size of C^* in H is at most $(1+\epsilon)c$. For $\epsilon < 1/3$,

$$\frac{(1+\epsilon)c}{(1-\epsilon)c} = 1 + \frac{2\epsilon}{1-\epsilon} < 1 + 3\epsilon\,.$$

Therefore, in step 3 no edge in C^* will be contracted. Finally, by Lemma 8 at most $O(n)$ edges are left between the super-vertices of G' in step 4 (whp, assuming that all cuts are indeed preserved within $(1 \pm \epsilon)$). Therefore, if C^* is a non-singleton cut, it (or a cut of the same size) will be found. No step requires more than $\widetilde{O}(n)$ queries. ◀

5 s-t min-cut in $\widetilde{O}(n^{5/3})$ queries

Now, we use the low-query sampling algorithm developed in Section 3 to obtain sub-quadratic query complexity for computing min s-t cuts in undirected and unweighted graphs. Our algorithm follows the same general strategy as the minimum cut algorithm from the previous section: sample a connectivity-preserving weighted graph from G, then compress the graph by contracting edges that do not participate in the minimum cut.

▶ **Algorithm 25** (s-t min cut with $\widetilde{O}(n^{5/3})$ queries).
Input: *Oracle access to the cut values of an unweighted simple graph G.*
1. *Compute a sparsifier H of G using Algorithm 17 with G and with $\epsilon = n^{-1/3}$.*
2. *Compute a maximum s-t flow in H, and remove the participating edges from H; denote the result H'*
3. *Obtain G' from G by contracting all components that are $3\epsilon \cdot c$-connected in H'.*
4. *Learn all edges of G' and compute the minimum s-t cut in the resulting graph.*

Output: *The minimum s-t cut computed in step 4.*

▶ **Theorem 26.** *Algorithm 25 finds an exact s-t minimum cut in $\tilde{O}(n^{5/3})$ oracle calls.*

Proof. Our proof is based on the following claim:

▶ **Claim 27.** *The number of edges between the super-vertices in G' is at most $O(n^{5/3})$.*

The sparsifier H output in step 1 by Algorithm 17 has $O(n \ln n/\epsilon^2)$ edges and preserves all cuts to within a multiplicative $(1 \pm \epsilon)$. In particular the value of the s-t maximum flow is at most n, and so it is preserved to within an additive $\pm\epsilon n$.

Then, in step 2 we compute an (exact) s-t maximum flow F in H, and subtract F from H to obtain the graph H'. Note that without loss of generality, F is integral and non-circular. Let $f_H, f_G \leq n$ denote the size of the minimum s-t-cut in G, H (respectively). Since each edge has strength at most n in G, each edge has weight at most $\epsilon^2 n$ in H Therefore, by Lemma 28 (stated shortly), the total weight in flow F is at most $O(n\sqrt{f_H \cdot n\epsilon^2})$; since $f_H \approx f_G \leq n$ (up to a $(1 \pm \epsilon)$ factor), this simplifies to $O(\epsilon n^2)$ total weight.

If we could subtract exactly the maximum flow in G, we could safely contract all remaining connected components (since the max flow certainly saturates a min s-t cut). Since the (exact) s-t maximum flow in H approximates the flow in G to within an additive $\pm\epsilon n$ error, we claim that we can safely contract any $3\epsilon n$-connected component in H':

Let C be a $3\epsilon n$-connected component in H'. Assume by contradiction that there is a minimum s-t cut that separates C. Because we preserved all cuts to within a multiplicative $(1 \pm \epsilon)$, the same cut has value at most $(1+\epsilon)f_G \leq f_H + 2\epsilon n$ in H. But because there is an s-t flow of value f_H in $H \setminus H'$, all cuts have value at least f_H in $H \setminus H'$. Therefore, this approximate min s-t cut *must* cut at most $2\epsilon n$ edges in H'. So immediately by definition of k-connectivity, we obtain a contradiction to this cut possibly separating a $3\epsilon n$-connected component in H'. This establishes the correctness of the algorithm, since the exact min s-t cut is not altered in step 2.

Once we contract the $3\epsilon n$-connected components in H', we are left (by Lemma 21) with a total weight of at most $3\epsilon n^2$ in H'.

After applying the same contractions to H, we have that the total remaining weight is at most $3\epsilon n^2 + O(\epsilon n^2) = O(\epsilon n^2)$ (the sum of the flow and H'); and therefore, since the cut around each of the contracted vertices is the same in G and H up to a factor of $(1 \pm \epsilon)$, we have that the number of edges remaining in the contracted graph G' is also $|E'| = O(\epsilon n^2)$.

The total number of queries necessary is $\widetilde{O}(n/\epsilon^2)$ in step 1, and then another $|E'|$ in step 4. Choosing $\epsilon = n^{-1/3}$ balances the terms, so that we have $|E'|, n/\epsilon^2 \leq n^{5/3}$. This concludes the proof. ◀

5.1 Covering s-t min cuts with $O(n^{3/2})$ edges

▶ **Lemma 28** (Flow cover). *In an undirected graph $G = (V, E)$ with integral weights from $[0, W]$, every non-circular s-t flow (for any $s, t \in V$) of value f uses edges of at most $O(n\sqrt{fW})$ total weight.*

Proof. Consider the induced *flow graph*, i.e. the DAG that has an edge from u to v with weight equal to the flow from u to v. Fix a topological sorting of the flow graph. We define the *length* of an edge to be the difference between its endpoints in the sorting.

Bucket all the edges into $O(\log W)$ buckets according to their weights, with bucket B_w containing all the edges of weight in $[w, 2w-1]$. Let d_w denote the (unweighted) average incoming degree when only considering edges from B_w, and let ℓ_w denote the (unweighted) average length of edges in B_w.

For each $i \in [n-1]$, at most f/w edges from B_w cross the cut C_i between the first i vertices and the last $n-i$ vertices in the topological ordering (because each such edge has weight $\geq w$ and the total flow crossing any of these cuts is exactly f). Similarly, the number of cuts each that edge in B_w crosses is exactly equal to its length. We can count the total number of pairs (e, C_i) such that edge $e \in B_w$ crosses cut C_i in two different ways: summing across $(n-1)$ cuts, or summing across $|B_w|$ edges. We therefore have that

$$(n-1) \cdot f/w \geq |B_w| \cdot \ell_w. \tag{2}$$

For each vertex v, each incoming edge has a different length; therefore, the average length among its incoming edges is at least half of its degree. By the Cauchy-Schwartz inequality it follows that this is also true on average across all edges and vertices:[8]

$$\ell_w \geq d_w/2. \tag{3}$$

Observe also that $|B_w| = n \cdot d_w$. Combining this observation with Inequalities (2) and (3), we have that

$$f/w \geq \ell_w \cdot \left(\frac{B_w}{n}\right) \geq d_w^2/2.$$

In particular, for each bucket, the number of edges is bounded by $|B_w| = O(n\sqrt{f/w})$

Therefore, the total weight of all edges is bounded by

$$\sum_w |B_w| \cdot 2w = \sum_w O(n\sqrt{fw}) = O(n\sqrt{fW}).$$

◀

Acknowledgements. The authors thank Robert Krauthgamer, Satish Rao, Aaron Schild, and anonymous reviewers for helpful conversations and suggestions.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012. URL: `http://portal.acm.org/citation.cfm?id=2095156&CFID=63838676&CFTOKEN=79617016`.

[8] To see this, we can compute the average length ($\ell(e)$ denotes the length of edge e) as $(\sum_v \sum_{e \text{ incoming to } v} \ell(e))/\sum_v d_v \geq (\sum_v d_v^2/2)/\sum_v d_v \geq ((\sum_v d_v)^2/2n)/\sum_v d_v \geq \sum_v d_v/2n = d_w/2$.

2 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 311–319. ACM, 2016. doi:10.1145/2840728.2840753.

3 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Review*, 56(2):315–334, 2014. doi:10.1137/130949117.

4 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.

5 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.

6 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic submodular function minimization. *CoRR*, abs/1610.09800, 2016. arXiv:1610.09800.

7 Shiri Chechik, Yuval Emek, Boaz Patt-Shamir, and David Peleg. Sparse reliable graph backbones. *Inf. Comput.*, 210:31–39, 2012. doi:10.1016/j.ic.2011.10.007.

8 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 749–758. ACM, 2008. doi:10.1145/1374376.1374484.

9 Efim A. Dinitz, Alexander V. Karzanov, and Micael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. *Studies in Discrete Optimization*, pages 290–306, 1976.

10 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 367–376. ACM, 2014. doi:10.1145/2611462.2611493.

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

12 Satoru Fujishige. *Submodular Functions and Optimization*. Elsevier Science, 2005.

13 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. doi:10.1007/BF02579273.

14 Nicholas J. A. Harvey. Matroid intersection, pointer chasing, and young's seminormal representation of S_n. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 542–549. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347142.

15 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.66.

16 David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 21–30. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313605.

17 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999. doi:10.1287/moor.24.2.383.

18 David R. Karger and Matthew S. Levine. Fast augmenting paths by random sampling from residual graphs. *SIAM J. Comput.*, 44(2):320–339, 2015. doi:10.1137/070705994.

19 David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. doi:10.1145/234533.234534.

20 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 665–674. ACM, 2015. doi:10.1145/2746539.2746588.

21 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 367–376. ACM, 2015. doi:10.1145/2688073.2688093.

22 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.68.

23 Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 608–615. SIAM, 2010. doi:10.1137/1.9781611973075.51.

24 Dalit Naor and Vijay V. Vazirani. Representing and enumerating edge connectivity cuts in RNC. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 2nd Workshop WADS '91, Ottawa, Canada, August 14-16, 1991, Proceedings*, volume 519 of *Lecture Notes in Computer Science*, pages 273–285. Springer, 1991. doi:10.1007/BFb0028269.

25 Robert Nishihara, Stefanie Jegelka, and Michael I. Jordan. On the convergence rate of decomposable submodular function minimization. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 640–648, 2014. URL: http://papers.nips.cc/paper/5255-on-the-convergence-rate-of-decomposable-submodular-function-minimization.

26 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

27 Peter Stobbe and Andreas Krause. Efficient minimization of decomposable submodular functions. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pages 2208–2216. Curran Associates, Inc., 2010. URL: http://papers.nips.cc/paper/4028-efficient-minimization-of-decomposable-submodular-functions.