SIAM J. COMPUT.
Vol. 0, No. 0, pp. 000–000

RANDOMIZED APPROXIMATION SCHEMES FOR CUTS AND FLOWS IN CAPACITATED GRAPHS*

ANDRÁS A. BENCZÚR† AND DAVID R. KARGER‡

David Karger wishes to dedicate this work to the memory of Rajeev Motwani. His compelling teaching and supportive advising inspired and enabled the line of research [18, 25, 19, 22] *that led to the results published here.*

Abstract. We describe random sampling techniques for approximately solving problems that involve cuts and flows in graphs. We give a near-linear-time randomized combinatorial construction that transforms any graph on n vertices into an $O(n \log n)$-edge graph on the same vertices whose cuts have approximately the same value as the original graph's. In this new graph, for example, we can run the $\tilde{O}(m^{3/2})$-time maximum flow algorithm of Goldberg and Rao to find an s-t minimum cut in $\tilde{O}(n^{3/2})$ time. This corresponds to a $(1+\epsilon)$-times minimum s-t cut in the original graph. A related approach leads to a randomized divide-and-conquer algorithm producing an approximately maximum flow in $\tilde{O}(m\sqrt{n})$ time. Our algorithm can also be used to improve the running time of sparsest cut approximation algorithms from $\tilde{O}(mn)$ to $\tilde{O}(n^2)$, and to accelerate several other recent cut and flow algorithms. Our algorithms are based on a general theorem analyzing the concentration of random graphs' cut values near their expectations. Our work draws only on elementary probability and graph theory.

Key words. AU MUST PROVIDE

AMS subject classification. AU MUST PROVIDE

DOI. 10.1137/070705970

1. Introduction.

This article gives results on random sampling methods for reducing the number of edges in any undirected graph while approximately preserving the values of its cuts and consequently flows. It then demonstrates how these techniques can be used in faster algorithms to approximate the values of minimum cuts and maximum flows in such graphs. We give an $\tilde{O}(m)$-time[1] *compression* algorithm to reduce the number of edges in any n-vertex graph to $O(n \log n)$ with only a small perturbation in cut values, and then use that compression method to find approximate minimum cuts in $\tilde{O}(n^2)$ time and approximate maximum flows in $\tilde{O}(m\sqrt{n})$ time.

1.1. Background.

Previous work [20, 19, 23] has shown that random sampling is an effective tool for problems involving cuts in graphs. A *cut* is a partition of a graph's vertices into two groups; its *value* is the number, or in weighted graphs the total weight, of edges with one endpoint on each side of the cut. Many problems depend only on cut values. The maximum flow that can be routed from s to t is the minimum value of any cut separating s and t [11]. A minimum bisection is the smallest cut that splits the graph into two equal-sized pieces. The *connectivity* or *minimum cut* of the graph, which we denote throughout by c, is equal to the minimum value of

*Received by the editors October 21, 2007; accepted for publication (in revised form) December 13, 2013; published electronically DATE.
http://www.siam.org/journals/sicomp/x-x/70597.html
†Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Hungary (benczur@sztaki.hu, http://www.sztaki.hu/∼benczur).
‡M.I.T. Computer Science and Artificial Intelligence Laboratory, Cambridge MA 02139 (karger@mit.edu, http://people.csail.mit.edu/∼karger). The work of this author was supported in part by a grant from the National Science Foundation.
[1]The notation $\tilde{O}(f)$ denotes $O(f \text{ polylog } n)$, where n is the input problem size.

any cut.

Random sampling "preserves" the values of cuts in a graph. If we pick each edge of a graph G with probability p, we get a new graph in which every cut has expected value exactly p times its value in G. A theorem by Karger [19] shows that if the graph has unit-weight edges and minimum cut c, then sampling with probability $\tilde{O}(1/\epsilon^2 c)$ gives cuts that are all, with high probability, within $1 \pm \epsilon$ of their expected values. In particular, the (global) minimum cut of the sampled graph corresponds to a $(1+\epsilon)$-times minimum cut of the original graph. Similarly, an s-t minimum cut of the sampled graph is a $(1+\epsilon)$-times minimum s-t cut of the original graph. Since the sampled graph has fewer edges (by a factor of $\tilde{O}(1/c)$ for any fixed ϵ), minimum cuts can be found in it faster than in the original graph. Working through the details shows that an approximately minimum cut can be found roughly c^2 times faster than an exact solution.

A variant of this approach finds approximate solutions to flow problems via *randomized divide-and-conquer*. If we randomly partition the edges of a graph into roughly $\epsilon^2 c$ subsets, each looks like the sample discussed in the previous paragraph and so has approximately accurate cuts. In other words, random division is a good approximation to evenly dividing up the capacities of all the cuts. By max-flow min-cut duality [11], this means that the s-t max-flow of G is also approximately evenly divided up. We can find a maximum flow in each of the subgraphs and add them together to get a flow in G that is at least $(1-\epsilon)$ times optimal. Again, detailed analysis [19] shows that finding this approximate flow can be done c times faster than finding the exact maximum flow.

Unfortunately, the requirement that $p = \tilde{\Omega}(1/c)$ (to keep the sample variance small) limits the effectiveness of this scheme. For cut approximation, it means that in a graph with m edges, we can only reduce the number of edges to m/c. Similarly for flow approximation, it means we can only divide the edges into c groups. Thus, when c is small, we gain little. Results can be even worse in weighted graphs, where the ratio of total edge weight to minimum cut value is unbounded.

1.2. Results. In this article, we show how *nonuniform sampling* can be used to remove graph sampling's dependence on the minimum cut c. Our main results are twofold: one for cut problems and one for flow problems. Both are based on a general theorem describing a *smoothness* condition under which a graph with random edge weights has all cut values concentrated near their expectations with high probability.

For cuts, we show that by sampling edges *nonuniformly*, paying greater attention to edges crossing small cuts, we can produce accurate samples with far less than m/c edges—rather, the resulting *compressed* graph has only $\tilde{O}(n/\epsilon^2)$ edges, regardless of the number of edges in the original graph. Our approach works for undirected graphs with arbitrary weights (capacities).

Even ignoring the algorithmic aspects, the fact that any graph can be approximated by a sparse graph via random sampling is of independent combinatorial interest.

In addition to proving that such sampling works, we give fast algorithms for determining the sampling importance of different edges and the correct sampling probabilities for them. This involves an extension of the *sparse certificate* technique of Nagamochi and Ibaraki [30].

Using these results, we demonstrate the following result.

THEOREM 1.1. *Given a graph G and an error parameter $\epsilon \leq 1$, there is a graph G' on the same vertices such that*

- *G' has $O(n \log n / \epsilon^2)$ edges and*

- *the value of every cut in G' is within $(1 \pm \epsilon)$ times the value of the corresponding cut in G.*

G' can be constructed in $O(m \log^2 n)$ time if G is unweighted or has polynomially bounded weights, and in $O(m \log^3 n)$ time for general weights.

It follows that, given any algorithm to (even approximately) solve a cut problem, if we are willing to accept an approximate answer, we can substitute $n \log n$ for any factor of m in the running time. At preliminary publication of this work [6], we gave the following corollaries.

COROLLARY 1.2. *In an undirected graph, given $\epsilon \leq 1$, a $(1+\epsilon)$-times minimum s-t cut can be found in $\tilde{O}(n^2/\epsilon^2)$ or $\tilde{O}(n^{3/2}/\epsilon^3 + m)$ time.*

COROLLARY 1.3. *In an undirected integer-weighted graph, given $\epsilon \leq 1$, a $(1+\epsilon)$-times minimum s-t cut of value v can be found in $\tilde{O}(nv/\epsilon^2 + m)$ time.*

COROLLARY 1.4. *An $O(\log n)$-approximation to the sparsest cut in an undirected graph can be found in $\tilde{O}(n^2)$ time.*

These corollaries followed by applying our sampling scheme to (respectively) the then-fastest maximum flow algorithms of Goldberg and Tarjan [14] and Goldberg and Rao [13], the classical augmenting-paths algorithm for maximum flow [11, 2], and the Klein–Stein–Tardos algorithm for approximating the sparsest cut [26].

Since that time, improvements to these corollaries have been achieved, each based on our sparsification approach. Arora, Hazan, and Kale [4] gave an algorithm for an $O(\sqrt{\log n})$-approximation to sparsest cut that runs in $\tilde{O}(mn)$ time and used our sparsification scheme to improve it to $\tilde{O}(n^2)$ time; Christiano et al. [8] gave an $\tilde{O}(m^{4/3})$-time algorithm for approximate max-flow and used sparsification to improve its runtime to $\tilde{O}(n^{4/3})$ for approximate min-cut and constant ϵ.

A related approach [19] helps solve flow problems: we divide edges crossing small cuts into several parallel pieces, so that no one edge forms a substantial fraction of any cut it crosses. We can then apply a randomized divide-and-conquer scheme. If we compute a maximum flow in each of the subgraphs of the random division using the Goldberg–Rao algorithm, and then add the flows into a flow in G, we deduce the following corollary.

COROLLARY 1.5. *A $(1-\epsilon)$-times maximum flow can be found in $\tilde{O}(m\sqrt{n/\epsilon})$ time.*

If we instead apply the recent approximate max-flow algorithm of Christiano et al. [8], we achieve a runtime of $\tilde{O}(mn^{1/3}/\epsilon^{4\frac{2}{3}})$.

The work presented here combines work presented earlier by Benczúr and Karger [6] and by Karger [21]. The presentation is significantly simplified, and details and slight improvements are given.

A companion paper [24] applies related methods to give an $\tilde{O}(nv)$-time *exact* max-flow algorithm (with no ϵ-dependence) based on augmenting randomly sampled paths. That algorithm is incomparable to the approximation algorithms, outperforming them for small flows but underperforming for large ones. It also offers slight improvements in the time for approximate max-flow, from $\tilde{O}(m\sqrt{n}/\epsilon)$ to $\tilde{O}(m\sqrt{n/\epsilon})$.

1.3. Method. We extend the approach of Karger [19]. That previous work proved it "safe" to sample edges with probability $p = \Omega((\log n)/c)$ from unweighted graphs with minimum cut c. Because the expected size of each sampled cut is large, a Chernoff bound can be used to show that all cuts are likely to have cut values close to their expectations, which are just p times the original cut values. So cut values can be approximated by computing them in the sparser sample.

This approach offers little benefit when the number of edges m is large but the

minimum cut c is small, preventing a substantial reduction in the large number of edges. However, we show that graphs with large numbers of edges necessarily contain *strong components*—induced subgraphs with large connectivities.

So suppose a graph, with possibly small c, contains a subgraph K of higher connectivity k. By the original reasoning, it is safe to sample edges *inside* K with probability $\tilde{\Omega}(1/k)$, even if the rest of the graph can only be sampled with probability $\tilde{\Omega}(1/c)$. Sampling at this lower rate would decrease the number of edges in the sample, allowing improved running times.

Of course, if we sample different edges with different probabilities, then the expected number of edges crossing a cut is the sum of those probabilities, which is no longer proportional to the number of crossing edges. This throws off our approximations. To correct them, the edges of K that we sample with probability $1/k$ are given *weight* k. The *expected weight* of each edge is now 1, meaning that the expected weight of edges crossing each cut is equal to the original number of edges crossing it. At the same time, since we are only *scaling* the the sample from K by a factor of k, the tight concentration of cut values around their (scaled) expectations is preserved.

It follows that sampling the edges *inside* K with probability $p = \tilde{\Omega}(1/k)$ (and then reweighting each sampled edge by $1/p$) does not significantly change cut values (i.e., the new weighted cut value is probably close to the original value). At the same time, sampling the edges *outside* K with probability $\tilde{\Omega}(1/c)$ (and reweighting) also does not significantly change cut values.

More generally, we define the *strength* k_e of edge e to be the maximum connectivity of any vertex-induced subgraph that contains e. We show that sampling each edge with probability $p_e = \tilde{\Omega}(1/k_e)$ and then giving it weight $1/p_e$ if sampled yields a graph whose expected cut weights are equal to the original and whose actual cut weights are tightly concentrated around those expectations.

Conveniently, we show that as the number of edges in a graph increases, so does the strength of edges, which means we can sample them with lower probability. These effects cancel out, enabling us always to create a good sample with $\tilde{O}(n/\epsilon^2)$ edges.

1.4. Outline. We conclude the introduction with relevant definitions. We then use section 2 to introduce and motivate the main concepts of the paper through a special case that is particularly easy to follow. In section 3 we provide background from Karger's earlier work [19] on uniform random sampling of graph edges. To make this article self-contained, we give a (new, and cleaner) proof of that result. In section 4 we define a *strong connectivity* measure of the best connectivity of a subgraph containing each edge. We define *smooth random graphs* (section 5) to be those where no edge weight is large compared to its own strong component, and prove that any smooth graph has cuts near its expectations. In section 6, we apply smoothness to show that a *compression* scheme that samples nonuniformly based on the strong connectivity measure produces good cut approximations. Our application to *s-t* min-cuts is immediate. In section 7 we introduce *graph smoothing*, a variation on compression that can be used for flow approximation. Finally, in section 8, we show how the strong connectivities needed to set sampling rates can be estimated quickly.

1.5. Definitions. We consider undirected graphs with positive-valued weights on the edges. We use the term "unweighted graph" to refer to a graph in which all edges have weight 1. We will use G to mean a graph with n vertices and m edges; parallel edges are allowed.

A *cut* $\mathcal{C}$ is a partition of the vertices into two subsets. The *value* of the cut in

unweighted (resp., weighted) graph G is the total number (resp., weight) of edges with endpoints in different subsets.

We simplify our presentation with a vector notation. The term x_E denotes a vector assigning some value x_e to each $e \in E$. All operations on vectors in this paper are *coordinatewise*. The interpretation of $x_E + y_E$ is standard, as is the pointwise product γx_E for any constant γ. However, we let $x_E \circ y_E$ denote the (Hadamard) product z_E with $z_e = x_e y_e$. Similarly, let $1/x_E$ denote the vector z_E such that $z_e = 1/x_e$ (pointwise inverse), and let y_E/x_E be the vector z_E with $z_e = y_e/x_e$.

A weighted graph G can be thought of as the vector (indexed by edge set E) of its edge weights. (An unweighted graph has value 1 in all coordinates.) Applying our vector notation, when r_E is a vector over the edge set, we let $r_E \circ G$ denote a graph where edge e has weight multiplied by r_e. If r is a scalar, then rG simply multiplies each weight by r. Similarly, if G and H are graphs on the same vertices, then $G + H$ denotes the graph whose edge weight vector is the sum of those graphs'.

We also introduce a sampling notation. As is traditional, we let $G(p)$ denote a graph in which each (possibly weighted) edge of G is incorporated with probability p. Generalizing, we let $G(p_E)$ denote a random subgraph of G generated by including each edge e of G (with its original weight) independently with probability p_e. We define the *expected value graph* $E[G(p_E)] = p_E \circ G$, since the expected value of any edge in $G(p_E)$ is equal to the value of that edge in $p_E \circ G$. It follows that the expected value of each cut in $G(p_E)$ is the value of the corresponding cut in the expected value graph.

We say that an event occurs with *high probability* if its probability is $1 - O(n^{-d})$ for some constant d. The constant can generally be modified arbitrarily by changing certain other constants hidden in the asymptotic notation.

2. Core ideas. We begin our presentation with an overview that aims to introduce and motivate the main ideas of the paper free of the many details and equations that are required to formalize them. We do so by working through a particular easy case—a particular graph—where the "right" sampling approach and its analysis are easy to see. This section can be skipped by those who prefer to dive right into the details.

We begin with a previous result of Karger [19], given here as Basic Sampling Theorem 3.1. The theorem shows that in a graph with minimum cut c, we can set a parameter $\rho = O(\log n)$, sample each edge with probability $p = \rho/c = \tilde{\Omega}(1/c)$, and get a graph where all cuts are near their expected value with high probability. The proof of this theorem starts with the Chernoff bound, which is immediately applicable to show that any *particular* cut is near its expectation with high probability (since that expectation exceeds $O(\log n)$). A union bound over all exponentially many cuts then shows (somewhat surprisingly) that tight concentration near the expectation is likely for all cuts simultaneously.

We can use this sampled graph to approximately solve cut problems. As a particular example, consider a graph that is a union of c (arbitrary) spanning trees. This graph has $(n-1)c$ edges and minimum cut c, since at least one edge of each tree crosses each cut. Sampling with probability ρ/c preserves approximate cut values but reduces the number of edges to $O(n \log n)$—a very good outcome.

Unfortunately, some graphs with minimum cut c may have far more edges. If so, our rate-$\tilde{O}(1/c)$ sample based on Basic Sampling Theorem 3.1 will similarly have far more edges than we would like. The problem is that just one small cut in the graph suffices to impose a significant limitation on our ability to apply Theorem 3.1, even if

the rest of the graph has many edges.

But intuitively, a graph with many edges ought to have "dense" regions of high connectivity. We will formalize this later, but for now consider a particular case. Take our c-trees graph above. Choose a set K of r of the n vertices and remove all the edges between them; replace those edges with a set of $k \gg c$ spanning trees on the r vertices. The graph now has $(n-r)c+(r-1)k$ edges. But it still has minimum cut c. Now Basic Sampling Theorem 3.1 only lets us reduce the edge count to $\tilde{O}(n+rk/c)$, which could be arbitrarily large for large k.

To address this, note that the graph induced by K has connectivity $k \gg c$. Applying Theorem 3.1 to K tells us that the edges of K can be sampled at rate ρ/k while still preserving expectations. Doing so will reduce the $(r-1)k$ edges inside K to $\tilde{O}(r)$, a much better outcome.

If we sample only the edges inside K, we get tight concentration near expectations, but those expectations are no longer useful for approximating cuts in G. Because a cut of G, made up of some edges inside K and some outside, will have an expected value dependent on the portion of edges in K, expected cut values in G will no longer be proportional to their original values. We can correct for this, however, using *compression* (formalized in section 6): when sampling edges of K with probability ρ/k, we set the *weight* of each sampled edge to be k/ρ. Scaling all of graph K this way is irrelevant to Theorem 3.1. But with this change, the expected sampled value of every edge in K is 1, so the expected values of sampled cuts of G are equal to the original values of those cuts. Thus, we can conclude that in this sample all cuts will be near their original values in G.

This takes care of sparsifying the particularly dense component K. But we want the rest of the graph to be sparse as well. To arrange that, we can sample the remaining graph edges with probability ρ/c as we did originally (but set all edge weights to c/ρ to preserve expectations). The edges inside K, having already been sampled, can remain fixed instead of participating in the sampling process; this can only reduce the variation caused by sampling. Thus, Theorem 3.1 applies here as well, and we can again conclude that cuts remain near their expectations.

In summary, we sample edges inside K with probability ρ/k (while giving them weight k/ρ) and sample edges outside K with probability ρ/c (while giving them weight c/ρ). The expected number of edges that will remain is $O((n-r)c(\rho/c) + rk(\rho/k)) = O(n\rho) = O(n \log n)$, as desired.

A natural generalization is as follows: for *each* edge e in G find the best connectivity k_e of an induced subgraph K that contains e. We will formalize this with the definition of *strong connectivity* in section 4. Intuitively, Theorem 3.1 suggests that we can sample e with probability ρ/k_e and correct for this sampling by setting the weight be k_e/ρ. We will prove later that if there are many edges (which will require small sampling probabilities if we want to produce a sparse graph), then most of those edges must be inside well-connected subgraphs where such small sampling probabilities will be permitted.

The problem with applying Theorem 3.1 directly to the entire graph in this experiment is that the reweighting of edges can produce some edges whose weights are much larger than the expected minimum cut value in the sample. However, we have arranged that each edge weight be small compared to its own strong component, so that Theorem 3.1 can be applied to that component. We define graphs that meet this condition to be *smooth*, and show that smooth graphs' cuts stay near their expectations with high probability.

One additional refinement will be important. In the analysis above, we applied Theorem 3.1 twice in sequence. In each case, we can assert a deviation by an ϵ factor for each cut. But since some cuts involve both types of edges, we arrive at a total deviation factor of 2ϵ. If we start considering edges with *many* different sampling rates, we may find many factors of ϵ piling up in our error bound. To avoid this we arrange to sample "in parallel" instead of sequentially.

Continuing our example, we think of each weight-k/ρ edge of K as two edges: one of weight c/ρ and one of weight $(k-c)/\rho$. The edges of weight $(k-c)/\rho$ form a scaled version of the graph in K. Separately, we *combine* the edges of weight c/ρ in K with the weight c/ρ edges being sampled *outside* K to form a new graph. Note that the weight of sampled edges, and thus the value of cuts, is divided among the two graphs we have specified. In other words, the value of each cut in G is the sum of the corresponding cut values in the two graphs. When we conduct our sampling experiment on G, we can "project" it as a sampling experiment on each of the two graphs. We will argue that each of the two graphs has sampled cut values within ϵ of their expectations. Thus each cut of G, which is a sum of the corresponding cut values in the two graphs, will have values within ϵ of expectations.

The first graph, of edges of K given weight $(k-c)/\rho$, is a scaled version of K. So Theorem 3.1 immediately applies to show that samples from it are near their expectations. For the second graph, we claim that the *expected weight* of any cut in it is at least c, while each sampled edge weight is c/ρ. Thus, Theorem 3.1 (with all edge weights scaled by a factor of c/ρ) applies to this graph as well. To show the claim, first consider any cut that does not cross K. It therefore cuts the c-connected underlying part of G, where we have already argued that the expected cut values are equal to their original values which are at least c. If, on the other hand, the cut does cut K, then the claim follows from the k-connectedness of the underlying graph K: each cut in K has at least k edges, each being sampled with probability ρ/k and given weight c/ρ, so the expected cut weight is at least $k(\rho/k)(c/\rho) = c$.

Note that we do not actually run two separate sampling experiments to create these two graphs—instead, we use a thought experiment to consider what happens to different parts of the graph G when we sample from it, and use these thought experiments to understand the outcome in G. The sampling experiments in the two graphs are not independent of each other, but we can still apply a union bound to argue that both thought experiments play out as desired.

In summary, instead of applying Theorem 3.1 repetitively to the entire graph, accumulating multiple ϵ error factors, we decompose the graph into two parts, each carrying a portion of the value of each cut, and argue that each graph introduces an ϵ error on *its part* of each cut, so that the overall error introduced is still bounded by ϵ. This approach will allow us to bound the error even as we decompose the graph into many parts to cope with many distinct connectivity values.

Intriguingly, although the Chernoff bound still holds for each individual cut in our revised sampling approach, we show in section 5.2 that the union bound over cuts, which worked to prove Theorem 3.1, *cannot* be used to prove that all cuts simultaneously remain near the expectations with high probability. Instead, our decomposition approach is necessary.

3. Background: Uniform graph sampling. To make this article self-contained, we provide background on sampling a graph's edges *uniformly*. We re-prove the following theorem, which provides the basis for our new work.

THEOREM 3.1 (basic sampling [19]). *Let G be a graph in which the edges have mutually independent random weights, each distributed in the interval $[0,1]$. If the expected weight of every cut in G exceeds $\rho_\epsilon = 3(d+2)(\ln n)/\epsilon^2$ for some $\epsilon \leq 1$ and d, then with probability $1 - O(n^{-d})$ every cut in G' has value within $(1 \pm \epsilon)$ of its expectation.*

Karger has published several different proofs of this theorem [19, 22], but here we introduce a proof that takes a different approach, related to one developed by Lomonosov and Polesskii [27], using a *coupling argument*.

We begin with the well-known Chernoff–Hoeffding bound.

LEMMA 3.2 (see [7, 15]). *Given any set of independent random variables X_i with values distributed in the range $[0,1]$, let $\mu = E[\sum X_i]$ and let $\epsilon < 1$. Then*

$$\Pr\left(\sum X_i \notin [(1-\epsilon)\mu, (1+\epsilon)\mu]\right) \leq 2e^{-\epsilon^2\mu/3}.$$

The Chernoff–Hoeffding bound's requirement that each $X_i \leq 1$ is needed to prevent any one random variable from "dominating" the outcome of the sampling experiment. For example, if one variable takes on value S with probability $1/S$ and 0 otherwise, while all other variables are uniformly 0, then the (relatively rare, but nonnegligible) outcome of taking on value S will take the sum far away from its expectation of 1.

We apply this Chernoff bound to each cut in the random graph of Basic Sampling Theorem 3.1. We wish to bound the probability that some cut deviates by more than $\epsilon < 1$ from its expectation. Given expected edge weights μ_e and a particular cut C whose expected value is $\mu_C = \sum_{e \in C} \mu_e$, the Chernoff bound implies that the deviation probability is at most $p_C = 2e^{-\epsilon^2\mu_C/3}$. The hypothesis of Theorem 3.1 upper bounds $p_C < 2n^{-(d+2)}$. So each *particular* cut is close to its expectation with high probability.

However, we wish to bound the probability that *any* cut deviates. The most obvious tool is the union bound. It seems fated to fail, as each cut has only an inverse-polynomial likelihood of deviation, while there are exponentially many cuts that may deviate. Surprisingly, it works anyway.

The union bound implies that the probability that *any* cut deviates by ϵ is at most $\sum_C p_C$. We bound this quantity by considering a different experiment. Write $p_e = e^{-\epsilon^2\mu_e/3}$ and conclude that $p_C = 2\prod_{e \in C} p_e$. Now consider our graph G but suppose that edge e is *deleted* from G with probability p_e. Then p_C is precisely twice the probability that every edge in C is deleted, i.e., that the graph is disconnected at cut C. It follows that our union bound $\sum p_C$ is twice *the expected number of cuts that are left empty by the edge deletions* in this alternate experiment. It is this quantity that we proceed to bound.

Consider the connected components induced by the edges that are not deleted. The empty cuts are precisely those that partition the connected components into two groups without cutting any component; thus, if there are R components, the number of empty cuts is $2^{R-1} - 1$ (we must place each component on one of the two sides for 2^R possibilities, divide by 2 because reversing the sides selects the same cut, and subtract 1 to rule out placing all components on the same side). The expected number of empty cuts is thus $E[2^{R-1} - 1] = \frac{1}{2}E[2^R] - 1$, where R is the random variable denoting the number of components into which G is partitioned by deleting each edge e with probability p_e. We bound this quantity.

We start with a special case, where each $p_e = p$ and the graph has minimum cut c, an even integer.

LEMMA 3.3. *Over all n-vertex graphs with min-cut c an even integer, and for*

any deletion probability p, the quantity $E[2^R]$ is maximized by a cycle on n vertices, where each adjacent pair is connected by $c/2$ edges.

This lemma was first proven by Lomonosov and Polesskii [27]. It makes intuitive sense, as the cycle has the fewest possible edges of any graph with minimum cut c and is thus plausibly the "least reliable.'.

Proof. We use a *coupling* argument. Coupling [3, 16] is a powerful way to show that $E[A] \geq E[B]$ for random variables A and B. We define a procedure for generating a *pair* of samples a and b such that (i) a is (marginally) distributed according to A, (ii) b is (marginally) distributed according to B, and (iii) $a \geq b$ in every sample pair. Criterion (iii) means that our variables a and b are very much *not* independent, but this does not affect their expectations. By criterion (i), $E[A]$ is the expected value of the first element of a sample pair, while by criterion (ii), $E[B]$ is the expectation of the second element. Since, by criterion (iii), the first element is never less than the second, it follows that $E[A] \geq E[B]$.

Let Y be the cycle described in this lemma. Any n-vertex graph G with min-cut c has minimum degree c. Thus its edge count $m \geq nc/2$ is no less than the cycle's, which is exactly $nc/2$. Augment the cycle with $m - nc/2$ arbitrary self-loop edges (which have no impact on the outcome number of components) so that the two graphs have the same number of edges. We use a coupling argument to compare R_Y, the number of components produced by deletions from the cycle, to R_G, the number produced by deletions from the graph G.

We determine the number of components R by *contracting* all edges that are not deleted—that is, we unify their endpoints into a single vertex. Then R will be the number of vertices in the contracted graph. One way to produce this contracted graph is to generate a random variable representing the number k of edges that get contracted and distributed as a binomial distribution with parameters $1 - p$ and m, and then to choose k edges uniformly at random in sequence and contract each. Contracting a self-loop leaves R unchanged, while contracting any other edge decrements R.

We carry out this procedure on G and Y simultaneously in a coupled fashion. Our coupling generates random contractions of G and Y simultaneously, each with the correct distribution. But it also ensures (inductively) that Y never has fewer contracted vertices than G. It follows that under every possible sampling outcome $R_Y \geq R_G$, which in turn proves that $E[2^{R_Y}] \geq E[2^{R_G}]$, as claimed.

The coupling is done as follows. First, we select the same number of edges k to contract in both graphs, according to the binomial distribution $B(m, 1 - p)$. This is correct as both graphs have m edges. Then, for each contraction step, we create a particular bijective pairing of the not-yet-contracted edges of G and Y. We choose a uniformly random edge of G to contract, which fulfills the goal of contracting edges of G in random order. At the same time, we contract its mate in Y. Since the pairing of edges is bijective, it follows that the edges of Y are also being contracted in uniform random order, as required. The order of contraction of Y is not independent of the order of contraction of G, but this does not affect $E[2^{R_Y}]$.

We define a new edge pairing at each step. We assume by induction that $R_Y \geq R_G$. If $R_Y > R_G$, the pairing can be arbitrary—since one contraction decreases R_Y (and R_G) by at most one (or zero if the edge is a self-loop), we will still have $R_Y \geq R_G$ after the contraction, as required. Suppose, on the other hand, that $R_Y = R_G$. Since G's min-cut is never decreased by contractions, G has min-cut and thus min-degree at least c. Thus, any contraction of G will have at least $cR_G/2$ edges that have not yet

been contracted to self-loops, while the cycle Y (which remains a cycle throughout the contractions) will have exactly $cR_Y/2$ such edges. Since $R_Y = R_G$ we can pair every nonloop edge of Y with a nonloop edge of G, and pair the remaining edges arbitrarily. It follows that if R_Y decreases because a nonloop edge was contracted, then R_G decreases as well. Thus, R_Y cannot become less than R_G, and the invariant $R_Y \geq R_G$ is preserved.

In fact, we've shown the stronger result that Y *stochastically dominates* R_G, so the expectation inequality would hold for any mononotonic function of R. □

COROLLARY 3.4. *If $p^c = n^{-(d+2)}$ for $d \geq 0$, then $E[2^{R-1} - 1] = O(n^{-d})$.*

Proof. We have just shown that the n-vertex cycle maximizes $E[2^R]$. On the cycle, the number of components R is equal to the number of $c/2$-edge "bundles" (of edges connecting the same endpoints) that are completely deleted, except that it is 1 if no bundle is deleted. A bundle is deleted with probability $p^{c/2}$, so the number of deleted bundles follows a binomial distribution with this parameter. It follows that

$$\begin{aligned}
E[2^R] &= (1-p^{c/2})^n \cdot 2^1 + \sum_{r=1}^{n} \binom{n}{r} (p^{c/2})^r (1-p^{c/2})^{n-r} \cdot 2^r \\
&= (1-p^{c/2})^n + (1-p^{c/2})^n + \sum_{r=1}^{n} \binom{n}{r} (2p^{c/2})^r (1-p^{c/2})^{n-r} \\
&= (1-p^{c/2})^n + \sum_{r=0}^{n} \binom{n}{r} (2p^{c/2})^r (1-p^{c/2})^{n-r} \\
&= (1-p^{c/2})^n + (2p^{c/2} + 1 - p^{c/2})^n \\
&= (1-p^{c/2})^n + (1+p^{c/2})^n \\
&= 2 + O(n^2 p^c) \quad \text{when } n^2 p^c \leq 1.
\end{aligned}$$

Thus

$$\begin{aligned}
E[2^{R-1} - 1] &= \frac{1}{2} E[2^R] - 1 \\
&= O(n^2 p^c). \qquad \square
\end{aligned}$$

It remains to generalize from our special case $p_e = p$ to arbitrary probabilities. We do so in the following "reliability lemma."

LEMMA 3.5 (reliability). *In an n-vertex graph with edge parameters $p_e \leq 1$, Writ $p_C = \prod_{e \in C} p_e$ for each cut C. If $\max_C p_C = \delta \leq 1$, then $\sum p_C = O(n^2 \delta)$.*

This corollary bounds the expected number of failed cuts and thus (by the union bound) the probability that *any* cut fails.

Proof. Observe that an edge e with failure probability p_e can be "simulated" by a bundle of $k = 2\lceil \ln p_e / 2 \ln p \rceil$ parallel edges of failure probability p. That is, the probability of the entire bundle failing is approximately p_e (approaching p_e from below in the limit as $p \to 1$ with consequent $k \to \infty$). Thus each p_C in the simulated graph converges from below to its value in the graph being simulated. It follows that $\sum p_C$ approaches the desired limit and that $\max_C p_C \leq 1$. We have also ensured that each k is even. Thus, we have replaced all edges the any graph with edges of uniform failure probability, to which the previous results apply. □

Basic Sampling Theorem 3.1 is an immediate corollary of Reliability Lemma 3.5, as we recall that $p_e = e^{-\epsilon^2 \mu_e / 3}$ so that p_C becomes the Chernoff bound on the probability of cut C deviating by ϵ from its expectation.

4. Strong connectivity. In this section, we formalize the notion of subgraphs with large connectivities. As was discussed in section 2, if we identify a subgraph with connectivity $k \gg c$, then we might hope, given Basic Sampling Theorem 3.1, to sample edges in this subgraph with probability ρ/k, producing a graph much sparser than if we sample with probability ρ/c.

4.1. Definitions.

DEFINITION 4.1. *A graph G is* k-connected *if the value of each cut in G is at least k.*

DEFINITION 4.2. *A* k-strong component *of G is a maximal k-connected vertex-induced subgraph of G.*

Each individual vertex is trivially an ∞-strong component, but we will not count these as k-strong components since no edges have their strength defined by them.

DEFINITION 4.3. *The* strong connectivity *or* strength *of an edge e, denoted k_e, is the maximum value of k such that a k-strong component contains (both endpoints of) e. We say that e is* k-strong *if its strong connectivity is k or more, and* k-weak *otherwise.*

Note that the above definition of strong connectivity of an edge differs from the standard definition of connectivity.

DEFINITION 4.4. *The* (standard) connectivity *of an edge e is the minimum value of a cut separating its endpoints.*

An edge's strong connectivity is no greater than its connectivity, since any cut that separates a k-strong edge's endpoints must cut its k-strong component. However, it may be much less. Consider the graph $K(1, 1, n)$ with n vertices v_i and two distinct vertices s and t, unit-weight edges (s, v_i) and (v_i, t) for $i = 1, \ldots, n$, and edge (s, t). Edge (s, t) has (standard) connectivity $n + 1$ but only has strong connectivity 2.

4.2. Structure. As was discussed in section 2, analysis of our sampling scheme is based on a careful decomposition of the graph. In this section we begin to characterize that decomposition.

Strong-connectivity exhibits two useful "consistency" properties.

LEMMA 4.5. *Deleting k-weak edges does not change the strength of any k-strong edge.*

Proof. Clearly deleting edges cannot increase strength, so we need to consider only decreases.

Any k-strong edge e is inside a k-connected induced subgraph K; if edge f is k-weak, then, by definition, it is not in K. So removing f does not reduce the connectivity of K, which means that e remains k-strong. □

LEMMA 4.6. *Contracting k-strong edges does not change the strength of any k-weak edge.*

Proof. Clearly, contraction cannot decrease strengths, so we need to consider only increases.

We prove the stronger claim that contracting an entire k-strong component K does not increase the strength of any k-weak edge e. Suppose that it did. This would mean that e was inside an induced subgraph, K', which becomes k-connected when K is contracted. We will show that the graph $K \cup K'$ is k-connected in G, implying $e \in K'$ is k-strong before the contraction, a contradiction.

K' cannot be k-connected before the contraction—otherwise e would be k-strong before the contraction. So K must intersect K'—otherwise contracting K would not change K' and so could not make K' k-connected. Consider any cut of the graph induced by $K \cup K'$ in G. Because K and K' intersect, this cut must induce a cut in at

least one of K and K'. If it induces a cut in K, then its value is at least k since K is k-connected. If not, meaning that all of K is on one side of the cut, then it corresponds to a cut of K' with K contracted, which we posited has value at least k. In other words, all cuts of $K \cup K'$ have value at least k, which means that e is k-strong in G. □

Lemma 4.5 gives us a way understand the structure of strong components.

DEFINITION 4.7. *A family of sets is* laminar *if, for any two sets that intersect, one is contained in the other.*

LEMMA 4.8. *The strong components of G form a laminar family.*

Proof. Consider the following procedure that "unpacks" the strong components of G. Take some connected component K of G that has min-cut k. Then all edges of K have strength at least k, and the min-cut edges of K have strength exactly k. It follows that K is a (clearly maximal) k-connected subgraph of G, i.e., a k-strong component.

Removing all strength-k edges from K will split K into multiple components since all edges of a min-cut are removed. By Lemma 4.5, all other (larger) edge strengths and strong components are unchanged in K, so the connected components left over inside K are additional strong components of G. We will refer to the new strong components as *children* of *parent* K.

This procedure defines a tree structure (or forest if G is initially disconnected) on the strong components of G such that each strong component is *strictly* contained in its parent (because each parent has at least two children on the two sides of its min-cut). By induction, each strong component is (strictly) contained in its ancestors, (strictly) contains its descendants, and is disjoint from all other strong components. Thus, the strong components form a laminar family. □

COROLLARY 4.9. *A graph G on n vertices has at most $n-1$ distinct nontrivial strong components (ignoring individual vertices), and thus $n-1$ distinct edge strengths.*

Proof. We use induction on the *rank* r of G, defined as the number of edges in a spanning forest of G and thus at most $n-1$, to show that the number of strong components is at most r. Consider the procedure of the previous lemma for identifying the strong components. Note that one step of the procedure will remove edges of one strength and destroy at least one strong component while decreasing the rank by at least 1, and proceed by induction. □

REMARK 1. *While* Lemma 4.6 *and* Corollary 4.9 *are true of standard connectivities as well as strong connectivities,* Lemma 4.5 *distinguishes strong connectivity from standard connectivity and is critical in our sampling proofs, which is why we must rely on strong rather than standard connectivity in proceeding. Consider the graph $K(1,1,n)$ discussed above; edge (s,t) is the (unique) edge with standard connectivity exceeding* 2*, but deleting all the lower connectivity edges yields the subgraph consisting only of that one edge, which now only has standard connectivity* 1*.*

4.3. Weighting. Recalling, from our motivating example in section 2, our intention to sample edges with probability inversely proportional to their strong connectivities, the following lemmas help us analyze the outcome.

LEMMA 4.10. *If connected graph G has edge strengths k_e, then the graph $1/k_E \circ G$ has minimum cut exactly* 1*.*

Proof. Consider any minimum cut in G, of value c. Each edge in the cut has strength c, giving it weight $1/c$ in $1/k_E \circ G$. Thus, the cut has value 1 in $1/k_E \circ G$. It follows that the minimum cut in $1/k_E \circ G$ is at most 1.

Now consider any cut, of value k in G. Each edge crossing the cut has strength at most k, meaning it gets weight at least $1/k$ in $1/k_E \circ G$. Since k edges cross this cut, it follows that the cut has weight at least $k(1/k) \geq 1$. This shows that the minimum cut in $1/k_E \circ G$ is at least 1. □

LEMMA 4.11. *In a weighted graph with edge weights u_e and strengths k_e,*

$$\sum u_e/k_e \leq n-1.$$

Proof. Define the *cost* of edge e to be u_e/k_e. We show, by induction on the rank of G, that the total edge cost is bounded by the rank, which is at most $n-1$ (achieving this maximum for a connected graph).

If the rank is 0, then G has no edges, so the base case is trivial. Otherwise, let G have rank $r > 0$. Let K be any connected component of G; by the previous lemma it has a cut of cost exactly 1. By removing the cut edges we can break K in two, producing a new graph G', which has one more connected component than G, so its rank is $r-1$. By Lemma 4.5, strengths in G' are no greater than those in G, which means that costs in G' are no less. By induction, the cost of edges in G' is at most $r-1$; it follows that the same holds for the cost of those edges in G. Adding back the unit-cost cut completes the induction. □

5. Smooth graphs. Edge strength gives us the measure we need to formalize section 2. Instead of comparing the capacity of each edge being sampled to the minimum cut of the entire graph, we compare it to the minimum cut of the edge's strong component, which is larger and thus gives us more flexibility in sampling the edge. We aim to sample different edges with different probabilities depending on their strong components. In this section, we consider which such sampling probabilities will preserve expected cut values.

Instead of limiting our analysis to the simple coin-flip sampling experiment of section 2, we consider a general distribution of graph edge weights, as in Basic Sampling Theorem 3.1, but compare each edge's distribution to its own strength.

DEFINITION 5.1. *Let G be a random graph in which the weight U_e of edge e is a random variable in the range $[0, m_e]$ with expectation u_e. Let k_e be the strength of edge e in the* expected graph *$E[G]$, where each edge e gets weight $u_e = E[U_e]$. We say that G is* c-smooth *if, for every edge e, $cm_e \leq k_e$.*

Note that we use $E[G]$ to denote the expectation of G and *not* the edge set of G.

The random graph of Basic Sampling Theorem 3.1 with edge weights in $[0, 1]$ and minimum expected cut c has smoothness at least c. But the smoothness condition asserts that each edge satisfies the relative-size conditions of Theorem 3.1 *relative only to its own strong component*, which is a less stringent condition since some strengths can greatly exceed c.

5.1. Concentration of smooth graph cuts.

THEOREM 5.2 (concentration). *If random graph G is c-smooth for $c = 3(d+3)(\ln n)/\epsilon^2$, then with probability $1 - O(n^{-d})$ every cut in G has value within $(1 \pm \epsilon)$ times its expectation.*

Note that c is almost identical to the ρ of Basic Sampling Theorem 3.1, but 2 has been replaced by 3.

We devote the this section to proving Concentration Theorem 5.2. As was discussed in section 2, the basic approach is to apply Basic Sampling Theorem 3.1 separately to components of different connectivities in G—which we have now identified as the k-strong components. However, in order to prevent the error terms in different

components from accumulating excessively, we need to take care in decomposing the graph for analysis.

LEMMA 5.3 (decomposition). *Any c-smooth graph on n vertices can be decomposed as a positive-weighted sum of at most $n-1$ (dependent) random graphs, each with maximum edge weight at most 1 and minimum expected cut at least c.*

Proof. The building blocks of our decomposition are the $r < n$ strong components $K_1, \ldots, K_r$ of $E[G]$, where component i has strength k_i. Recall that these sets form a tree-structured laminar family (Lemma 4.8). To simplify notation, we will define a "metaroot" K_0 with "strength" $k_0 = 0$, and make it a parent of every root strong component in the laminar family. Thus, the laminar family is guaranteed to be a tree, and every actual strong component has a parent.

For each $i > 1$, let p_i denote the index of the parent strong component of K_i in the laminar family, so that K_i has parent K_{p_i}. Then we define the ith graph F_i in the decomposition to be the one on the same edges as K_i, but with edge e given weight cU_e/k_e. The coefficient for this component in the weighted sum is $(k_i - k_{p_i})/c$. In other words,

$$G = \sum_{i \geq 1} (k_i - k_{p_i})/c \circ F_i = \sum (k_i - k_{p_i})/k_e \circ K_i.$$

(Recall that $1/k_e \circ K_i$ scales the weight of each edge e in K_i by $1/k_e$.) To see that this sum is correct, note that edge e appears in its own strong component K and in those on the path through ancestors of that strong component up to the metaroot K_0; let us write this chain of strong components as $K = K_{i_1}, K_{i_2}, \ldots, K_{i_\ell} = K_0$. Thus the total weight of coefficients assigned to graphs containing edge e is $\sum (k_{i_j} - k_{i_{j+1}})/k_e = (1/k_e) \sum (k_{i_j} - k_{i_{j+1}})$. This sum telescopes: after canceling inner terms, the first term contributes $k_{i_1}/k_e = k_e/k_e = 1$, while the last term subtracts $k_0/k_e = 0$. Thus the overall sum of coefficients multiplying U_e is 1, as required.

It remains to show that each F_i meets the criteria of the decomposition. Because K_i is a strong component of $E[G]$, Lemma 4.5 tells us that the strengths of edges in the subgraph induced by K_i in $E[G]$ are the same k_e as those edge's strengths in $E[G]$. It follows from Lemma 4.10 that the graph on K_i with edge e given weight u_e/k_e has minimum cut exactly 1. But this graph is precisely $E[F_i]$ since each edge in F_i gets weight U_e/k_e. In other words, the minimum expected cut value in F_i is 1. At the same time, since edge e gets weight cU_e/k_e, the maximum value it takes on in F_i is $cm_e/k_e \leq 1$ by the smoothness criterion. In other words, our graph satisfies the decomposition criteria. ∎

Given Decomposition Lemma 5.3, proving Concentration Theorem 5.2 is straightforward. We have given a weighted-sum decomposition of G as a sum of less than n (random) graphs $\sum \alpha_i F_i$. It follows that $E[G] = \sum \alpha_i E[F_i]$. Each F_i meets the conditions of Basic Sampling Theorem 3.1. Thus, F_i has cuts within $(1 \pm \epsilon)E[F_i]$ with probability $1 - O(n^{-(d+1)})$. (The exponent $d+1$ arises from our using $d+3$ in the smoothness parameter instead of $d+2$ as in Theorem 3.1.) The graphs are not independent (each involving scaled versions of the random variables U_e), but we can still apply a union bound: the probability that any one of the n graphs diverges from its expectation is $n \cdot O(n^{-(d+1)}) = O(n^{-d})$. If every graph is within $(1 \pm \epsilon)$ of its expectation, then the (positive) weighted sum of these graphs is within $(1 \pm \epsilon)$ of the weighted sum of their expectations, which is $E[G]$.

REMARK 2. *Our decomposition proof demands our use of strong, rather than standard, connectivities. The natural analogue using standard connectivities would define*

components F_i as all edges of standard connectivity λ_i or greater, and could derive an analogous telescoping set of coefficients. But consider the graph $G = K(1, 1, n)$ described earlier, where all edges get (deterministic) weight 1 except that edge (s, t) gets weight ϵn with probability $1/\epsilon n$ and 0 otherwise. In $E[G]$ all edges have expected weight 1, so edge (s, t) has standard connectivity $n + 1$, while all other edges have standard connectivity 2. This would make G "$1/\epsilon$-smooth" (for $\epsilon < 1/2$) according to the definition above. The decomposition approach above would then place edge (s, t) into its own connectivity-$(n + 1)$ "component." But in this component, the maximum achieved edge weight (ϵn) is far larger than the minimum expected cut (1).

Note, however, that despite our inability to decompose this particular random graph the way we would like, it still exhibits tight concentration around its cut values. Thus, tight concentrate might hold even with respect to standard connectivity; we just cannot prove it using our approach.

5.2. Tightness. Basic Sampling Theorem 3.1 proved an inverse polynomial bound on the *expected number of divergent cuts*, which yields the unlikelihood of any cut diverging as a (weaker) corollary via the Markov inequality or union bound. Concentration Theorem 5.2 proves only the weaker claim of a low deviation probability. This is unavoidable; the stronger claim is not true for smooth graphs, as the following example demonstrates.

Consider a graph G on n vertices where the only edges are $96 \ln n$ edges connecting two particular vertices s and t; all other vertices are isolated. Each edge is present with probability $1/2$ and has weight 2 if present. The expected weight of every edge is 1; thus the strength of every edge in $E[G]$ is $96 \ln n$. It follows that graph G is c-smooth for $c = 48 \ln n$, which means that we can apply Concentration Theorem 5.2 with $d = 1$ to conclude that the probability of any cut deviating by $\epsilon = 1/2$ from its expectation is $O(1/n)$.

At the same time, with probability $2^{-96 \ln n} = n^{-96 \ln 2}$ none of the (s, t) edges is present. In this case every (s, t)-cut becomes empty, i.e., a factor 1 less than its expectation, violating the $\epsilon < 1$ deviation requirement. There are 2^{n-2} such (s, t) cuts, defined by placing every other vertex on one of the two sides. It follows that the expected number of deviating cuts is $n^{-96 \ln 2} \cdot 2^{n-2}$, which is exponential.

In summary, we have given an example where Concentration Theorem 5.2 shows that the probability of cut deviation is polynomially small but where the expected number of deviating cuts is exponential. This seems unavoidable since as soon as one component deviates it produces a huge number of deviating cuts based on the varying placement of other components. So we cannot hope to prove a polynomial bound on the number of deviating cuts as we did in Basic Sampling Theorem 3.1. By the same argument, Theorem 5.2 *cannot* be proven by a union bound over the deviation probabilities of the individual cuts, as was Theorem 3.1, since this union bound can diverge.

For simplicity we used a disconnected graph, but adding a complete graph of infinitesimal-weight edges does not change that overall outcome, so the same argument applies to connected graphs. The underlying problem is that deviation of a single cut in a single component can be "amplified" into many deviating cuts of which it is a part.

This example fits the special case of *graph compression*, discussed in the next section, and thus serves to show the same limitation for graph compression (Compression Theorem 6.2) as for smooth graphs: while the probability of cut deviation is small, the expected number of deviating cuts may be huge.

6. Approximating cuts by graph compression. We now specialize the results above to algorithms for approximating cuts. Concentration Theorem 5.2 considers a broad class of edge weight distributions. But, as was discussed in section 2, we will use a particularly simple weight distribution in our cut approximation algorithms: flipping an appropriately weighted coin to decide whether to keep each edge, while scaling the weights of kept edges to compensate for the coin flips.

We will use a fixed *compression factor* ρ_ϵ chosen to satisfy a target error bound ϵ:

$$\rho_\epsilon = 3(d+3)(\ln n)/\epsilon^2 \ .$$

When ϵ is clear from context we will simply write ρ.

DEFINITION 6.1 (compression). *Given a graph G with edge weights u_e and* compression probabilities *p_e, the compressed graph $G[p_e]$ includes edge e with probability p_e and gives it weight u_e/p_e if included.*

Note that each edge e has expected value u_e, so $E[G[p_e]] = G$. The expected number of edges chosen in compression is $\sum p_e$, so we would like to minimize these p_e to get the sparsest possible graph while preserving cut values near their expectations.

THEOREM 6.2 (compression). *Let G be a graph with edge weights u_e and strengths k_e. Given ϵ and a corresponding ρ_ϵ, for each edge e, let $p_e = \min\{1, \rho_\epsilon u_e/k_e\}$ and consider $G[p_e]$ from Definition* 6.1*. Then with probability $1 - O(n^{-d})$,*

1. *the graph $G[p_e]$ has $O(n\rho_\epsilon)$ edges, and*
2. *every cut in $G[p_e]$ has value between $(1-\epsilon)$ and $(1+\epsilon)$ times its value in G.*

This theorem is the detailed version of Theorem 1.1 from the introduction. In particular, for any target constant ϵ, the necessary ρ_ϵ will yield $O(n \log n)$ edges in the compressed graph.

Proof. The bound on edge count is immediate from Lemma 4.11. The expected number of edges is $\rho_\epsilon \sum u_e/k_e = O(\rho_\epsilon n)$, and a Chernoff bound shows that the outcome is close to this expectation with high probability.

For the bound on cut deviation, note that since $E[G[p_e]] = G$, the edge strengths in $E[G[p_e]]$ are the same k_e as in G. The (maximum) weight for edge e when sampled is $m_e = u_e/p_e$. If $p_e = \rho u_e/k_e$, then $m_e = k_e/\rho$, so the edge satisfies the ρ-smoothness definition. If, on the other hand, p_e is capped at 1, we might violate that condition. But in this case the edge e is being taken deterministically (with weight u_e), which can only help us. Formally, we can imagine subdividing the edge into numerous parallel edges of weight less than k_e/ρ, each taken with probability 1, which now each satisfy the smoothness condition. This does not change the outcome edge weight.

Since all edges meet the smoothness condition, the concentration of cut values follows immediately from Concentration Theorem 5.2. □

REMARK 3. Compression Theorem 6.2 *is the special case of* Concentration Theorem 5.2 *that gives each weight the highest possible variance subject to the condition that $cm_e \le k_e$—namely, weight k_e/c with probability p_e and 0 otherwise. It is thus analogous to the basic Chernoff bound that considers only $\{0,1\}$ random variables. The more general* Theorem 5.2 *is analogous to the Chernoff–Hoeffding bound that allows arbitrary $[0,1]$ distributions. As with the proof of the Chernoff–Hoeffding bound, it might be possible to prove the special case* (Theorem 6.2) *and then use a convexity argument to conclude that it upper bounds the general case.*

6.1. Using approximate strengths. Our analysis above assumed that edge strengths were known. While edge strengths can be computed exactly using max-flow, we do not know how to do so fast enough for use in our cut and flow approximation

algorithms. Examining the proofs above, however, shows that we do not need to work with exact edge strengths.

DEFINITION 6.3. *Given a graph G with n vertices, edge weights u_e, and edge strengths k_e, a set of edge values $\tilde{k}_e$ are* tight strength bounds *if*

1. $\tilde{k}_e \leq k_e$ *and*
2. $\sum u_e/\tilde{k}_e = O(n)$.

THEOREM 6.4. *The compression theorem remains asymptotically correct even if tight strength bounds are used in place of exact strength values.*

Proof. The bound on the number of edges in the compressed graph follows directly from the fact that $\sum u_e/k_e \leq n$; for tight strength bounds this summation remains asymptotically correct.

For cut accuracy, note that using lower bounds on strengths can only increase the p_e used in compression, which can only decrease the weights of sampled edges without changing expectations, which can only produce a larger (better) smoothness parameter for the sample graph. ∎

Tight strength bounds are much easier to compute than exact strengths.

THEOREM 6.5. *Given any m-edge, n-vertex graph, tight strength bounds can be computed in $O(m \log^2 n)$ time for unweighted graphs or graphs with polynomially bounded integer weights and $O(m \log^3 n)$ time for weighted graphs.*

Proof. We prove this theorem in section 8. ∎

6.2. Applications. We have shown that graphs can be compressed based on edge strengths while preserving cut values. We will show in section 8 that we can compute tight strength bounds in $\tilde{O}(m)$ time. We can then generate the compressed graph $G[p_E]$ as described in the compression theorem. The graph will have $O(\rho n) = O(n \log n/\epsilon^2)$ edges. Cut problems can therefore be approximately solved by working with the compressed graph as a surrogate for the original graph. We use this fact to prove the application corollaries from the introduction.

6.2.1. Minimum *s-t* cuts. Fix a pair of vertices s and t. Without loss of generality assume $\epsilon < 1/3$. Let $\hat{v}$ be the value of some minimum cut separating s from t in the compressed graph $G[p_E]$. We show that the minimum s-t cut value v in G is within $(1 \pm 3\epsilon)\hat{v}$. By Compression Theorem 6.2, with high probability any particular s-t cut of minimum value v in G has value at most $(1+\epsilon)v$ in $G[p_E]$. Thus $\hat{v} \leq (1+\epsilon)v$. Furthermore, with high probability every cut of G with value exceeding $(1+3\epsilon)v$ in G will have value at least $(1-\epsilon)(1+3\epsilon)v = (1+2\epsilon-3\epsilon^2)v \geq (1+\epsilon)v$ (assuming $\epsilon < 1/3$) in $G[p_E]$ and therefore will not be the minimum cut of $G[p_E]$. It follows that the minimum cut in $G[p_E]$ corresponds to a cut of value at most $(1+3\epsilon)v$ in G.

We can find this cut by computing a maximum flow in the $O(n \log n/\epsilon^2)$-edge graph $G[p_E]$. The maximum flow algorithm of Goldberg and Tarjan [14] has a running time of $O(nm \log(n^2/m))$, which leads to a running time of $O(n^2 \log^2 n/\epsilon^2)$ after compression. Similarly, the Goldberg–Rao algorithm [13], which runs in $\tilde{O}(m^{3/2})$ time, leads to a running time of $\tilde{O}(n^{3/2}/\epsilon^3)$ after compression. The recent max-flow approximation algorithm of Christiano et al. [8] runs in $O(m^{4/3}/\epsilon^3)$ time; applying compression modifies this time to $\tilde{O}(n^{4/3}/\epsilon^{17/3})$. It also increases the overall error to 2ϵ, but this can be dealt with by halving the initial target ϵ without affecting the asymptotic runtime. This analysis proves Corollary 1.2 (which includes an $\tilde{O}(m)$ term, which will dominate in dense graphs, to reflect the time to construct the compressed graph).

In an integer-weighted graph with small flow value, we may wish to apply the

classical augmenting path algorithm [11, 2] that finds a flow of value v in v augmentations. As described, the graph-compression process can produce noninteger edge weights ρ/k_e, precluding the use of augmenting paths in the compressed graph. However, if we decrease each compression weight to the next lower integer (and increase the sampling probability by a negligible amount to compensate) then compression will produce an integer-weighted graph in which the augmenting paths algorithm can be applied to find an s-t cut of value at most $(1+\epsilon)v$ in time $O(nv \log n/\epsilon^2)$. This proves Corollary 1.3.

6.2.2. Sparsest cuts. A *sparsest cut* of a graph G minimizes the ratio between the cut value and the product of the numbers of vertices on the two sides. It is $\mathcal{NP}$-hard to find the value of a sparsest cut. To find an α-approximate value of a sparsest cut, we use the approach of the previous subsection: we compute a β-approximate sparsest cut in the compressed graph $G[p_E]$ for $\beta = \alpha/(1+\epsilon)$. This cut is then an $\alpha = (1+\epsilon)\beta$-approximate sparsest cut of G.

An algorithm of Klein, Stein, and Tardos [26] finds an $O(\log n)$-approximation to a sparsest cut in $O(m^2 \log m)$ time. By running their algorithm on $G[p_E]$, we will find an $O(\log n)$-approximate sparsest cut in $O(n^2 \log^3 n/\epsilon^4)$ time. Our small cut-sampling error is lost asymptotically in the larger error of the approximation algorithm. This proves Corollary Corollary 1.4.

More recently, Arora, Hazan, and Kale [4] gave an $\tilde{O}(mn)$-time algorithm achieving an $O(\sqrt{\log n})$ approximation for sparsest cut, and applied our compression method to improve the runtime to $\tilde{O}(n^2)$. Our approach has been applied in a similar way to improve the running time of a spectral partitioning algorithm [17].

7. Approximating flows by graph smoothing. Until now we have focused on cut problems. Our compression scheme produces a graph with nearly the same cut values as the original, so cut problems can be approximated in the compressed graph. But consider a maximum flow problem. One could try to approximate this maximum flow by finding a maximum flow in the compressed graph. By saturating an approximately minimum s-t cut, this approach does indeed give an approximation to the *value* of the maximum flow. But since edges in the compressed graph have *larger* capacity than the original graph edges, a feasible flow in the compressed graph may not be feasible for the original graph.

Previous work [19] tackled the flow approximation problem with a divide-and-conquer approach. The edges of G are randomly divided into groups, producing several random subgraphs of G. Basic Sampling Theorem 3.1 shows that each subgraph has cut, and thus flow, values near its expectation. By computing a flow in each subgraph and adding the flows, we find a flow of value $(1-\epsilon)$ times the maximum flow in G. Because we are dividing up capacity without increasing it, the flow is feasible in G.

This approach suffered the same limitation as the uniform sampling approach for cuts: the probability of each edge occurring in each subgraph must be $\tilde{\Omega}(1/c)$ to preserve cut values. This translates into a restriction that we divide into $\tilde{O}(c)$ groups, which limits the power of the scheme on a graph with small minimum cuts. Graph compression's nonuniform sampling approach does not seem to provide an immediate answer: clearly we cannot simultaneously divide each edge with strength k_e among k_e distinct subgraphs. Instead we seek a consistent rule that divides all edges among a fixed number of subgraphs. In this case, each subgraph must necessarily look like a *uniform* sample from the original graph.

Unfortunately, uniform sampling can violate the smoothness condition that un-

derlies Concentration Theorem 5.2 (and its special case, Compression Theorem 6.2). Under graph compression, weak edges were given small weight (and sampled with higher probability) so that their presence or absence would not significantly impact the sampled cut weight. The need for uniform sampling rules this out. So instead, we *subdivide* weak edges into numerous edges of smaller weight, achieving the same goal of limiting the impact of individual edge outcomes on the cuts they are in.

Dividing *all* the graph edges is pointless: splitting all edges in half doubles the minimum cut (allowing us to sample at half the original rate while preserving approximate cut values), but since we double the number of edges, we end up with the same number of sampled edges as before. But since only a small fraction of the graph's edges are weak, dividing *only* those weak edges does not add very many. Thus, algorithms based on the uniform samples remain efficient.

7.1. Smooth graphs revisited. We extend the standard notation $G(p)$ to weighted graphs, to denote taking each capacitated edge with (uniform) probability p, then prove that sampling works. As discussed above, the problem is that a single capacitated edge might account for much of the capacity crossing a cut. The presence or absence of this edge has a major impact on the value of this cut in the sampled graph. However, edge strength gives us a useful bound on how much impact a given edge can have. We have already defined c-smoothness for random graphs; there is an obvious special case for nonrandom graphs.

DEFINITION 7.1. *A (nonrandom) graph G with edge capacities u_e and edge strengths k_e is* c-smooth *if, for every edge, $k_e \geq cu_e$.*

This fits the definition of random-graph smoothness, since for deterministic graphs u_e is both the expected value and maximum value for edge e. Note that a graph with integer edge weights and minimum cut c has smoothness at most c but possibly much less. We now consider smoothness as a criterion for applying uniform sampling to weighted graphs.

THEOREM 7.2 (smooth sampling). *Let G be a c-smooth deterministic graph. Let $p = \rho_\epsilon/c$, where $\rho_\epsilon = O((\log n)/\epsilon^2)$ as in Compression Theorem 6.2. Then with high probability, every cut in $G(p)$ has value in the range $(1\pm\epsilon)$ times its expectation (which is p times its original value).*

Proof. The graph $G(p)$ is a random graph in which edge e has maximum possible weight u_e. Since $E[G(p)] = p \cdot G$, the strength of e in $E[G(p)]$ is pk_e. It follows that $G(p)$ meets the definition of a pc-smooth random graph, so the proof is immediate from Concentration Theorem 5.2. ∎

7.2. Making graphs smooth. We have argued that good uniform sampling will lead to good flow algorithms, and then shown that smooth deterministic graphs can be sampled uniformly. We now give an algorithm for transforming any graph into a smooth one.

LEMMA 7.3. *Given an m-edge graph, a* smoothness parameter *c, and the strengths k_e of all edges, we can transform the graph into an $(m + cn)$-edge c-smooth graph in $O(m + cn)$ time.*

Proof. Divide edge e into $\lceil cu_e/k_e \rceil$ parallel edges, each of capacity $u_e/\lceil cu_e/k_e \rceil \leq k_e/c$, so the total capacity is u_e. These edges remain k_e-strong but now satisfy the smoothness criterion.

It remains to prove that this division creates at most cn new edges. The number

of edges in our smoothed graph is

$$\begin{aligned}\sum_e \lceil cu_e/k_e \rceil &\le \sum_e (1 + cu_e/k_e) \\ &\le m + \sum cu_e/k_e \\ &= m + c\sum u_e/k_e \\ &\le m + cn,\end{aligned}$$

where the last line follows from Lemma 4.11. □

COROLLARY 7.4. *Given edge strengths, in $O(m)$ time we can transform any m-edge capacitated graph into an $O(m)$-edge capacitated (m/n)-smooth graph.*

The corollary follows by setting the smoothness parameter $c = m/n$. This is in some sense optimal. Any smaller smoothness parameter leads to worse sampling performance without decreasing the asymptotic number of edges (which is always at least m). A larger smoothness parameter provides better sampling behavior but linearly increases the number of edges such that the gains from sparser sampling are canceled out.

7.3. Approximate max-flows. To approximate flows, we use graph smoothing. As was argued in Theorem 6.4, graph smoothing works in an unchanged way even if we use tight strength bounds, rather than exact strengths, in the computation.

After computing tight strength bounds in $\tilde{O}(m)$ time (as will be described in section 8), we can apply Smooth Sampling Theorem 7.2, which states that in any c-smooth graph, sampling with probability p produces a graph in which with high probability all cuts are within $(1 \pm \epsilon)$ of their expected values. This fact is the only one used in the uncapacitated graph flow algorithms of [19]. Therefore, those results immediately generalize to the smooth graphs defined here—we simply replace "minimum cut" with "smoothness" in all of those results. The generalization is as follows.

LEMMA 7.5. *Let $T(m, n, v, c)$ be the time to find a maximum flow in a graph with m edges, n vertices, flow v, and smoothness c. Then for any ϵ, a flow of value $(1-\epsilon)v$ on an m-edge, n-vertex, smoothness-c graph can be found in*

$$\tilde{O}\left(\frac{1}{p}T(pm, n, pv, pc)\right)$$

time, where $p = \Theta((\log n)/\epsilon^2 c)$. In particular, setting $c = m/n$, we can achieve a time of

$$\tilde{O}\left(\frac{\epsilon^2 m}{n}T\left(\frac{n}{\epsilon^2}, n, \frac{nv}{\epsilon^2 m}, \frac{\log n}{\epsilon^2}\right)\right).$$

Proof. Divide the graph edges into $1/p$ random groups. Each group defines an edge-disjoint subgraph with pm edges. Since the minimum s-t cut of G is v, the minimum expected s-t cut in each subgraph is pv. By Smooth Sampling Theorem 7.2, each subgraph has minimum s-t cut, and thus maximum s-t flow, at least $(1-\epsilon)pv$. Since cut values are preserved, strengths are also near their expectations in the sampled graph and thus scale by p. Find a flow in each subgraph and combine the results. This total flow will be $(1/p)(1-\epsilon)pv = (1-\epsilon)v$. □

COROLLARY 7.6. *In any undirected graph, given edge strengths, a $(1-\epsilon)$-times maximum flow can found in $\tilde{O}(m\sqrt{n}/\epsilon)$ time.*

The companion article [24] slightly improves this bound to $\tilde{O}(m\sqrt{n/\epsilon})$ time.

Proof. Begin by converting the graph to an $O(m)$-edge (m/n)-smooth graph, as discussed in Lemma 7.3. The Goldberg–Rao flow algorithm [13] gives $T(m,n) = \tilde{O}(m^{3/2})$ for the previous lemma. (Since we are already giving up a factor of ϵ, we can scale and round all edge capacities to be polynomial, thus eliminating the capacity scaling term in their algorithm.) Plugging this in gives a time bound of $\tilde{O}(m\sqrt{n}/\epsilon)$. □

If we instead use the new algorithm of Christiano et al. [8], the same analysis yields a runtime of $\tilde{O}(mn^{1/3})$ for constant ϵ.

Unlike our algorithm for minimum cuts, it is not possible to use the standard augmenting paths algorithm to find a max-flow of value v in $\tilde{O}(nv/\epsilon^2)$ time. The graph smoothing process may subdivide unit-capacity edges, producing fractional-capacity edges to which unit-capacity augmenting flows cannot be applied.

In previous work [21], Karger used the above smoothing technique to compute exact flows more quickly than before; however, this work has been superseded by better algorithms (also based on edge strength but not using smoothing) presented in the companion article [24].

Graph smoothing actually offers an alternative approach to approximating min-cuts as well: once a graph has been made (m/n)-smooth as above, choosing edges with probability $\rho_\epsilon n/m$ will meet the conditions of Smooth Sampling Theorem 7.2 and yield a graph with $(1\pm\epsilon)$-accurate cuts; the sampled graph will have $O(\rho_\epsilon n)$ edges with high probability, just as in Compression Theorem 6.2, so it can be used the same way to estimate min-cuts. Compression thus seems to be a generally weaker approach than smoothing. However, for cuts it seems more elegant and skips the smoothing step. This also means it preserves integrality, which is essential for the $\tilde{O}(nv/\epsilon^2)$ time bound.

8. Finding strong connectivities. To efficiently compress and smooth graphs we would like to efficiently find the strong connectivities of edges. Unfortunately, it is not clear how quickly this can be done (iteratively breaking up strong components using n minimum-cut computations is one slow solution). But as discussed in Theorem 6.4, we do not require the exact values k_e. We now show that it is possible to find tight strength bounds $\tilde{k}_e$ that satisfy the two key requirements of Definition 6.3: that $\tilde{k}_e \le k_e$ and $\sum 1/\tilde{k}_e = O(n)$. These suffice for the cut and flow algorithms described above.

Our basic approach is to repeatedly find, label, and remove the weakest edges in the graph; Lemma 4.5 tells us that this does not change the strengths of the other edges, so we can iterate to find all strengths. We use a *sparse certificate* algorithm of Nagamochi and Ibaraki [30] to find all edges crossing a small cut (and this weak). While this does not find *all* weak edges, we show that it carves off many of the strong components and thus makes significant progress, so that a few iterations do find all weak edges. We also slightly tweak the algorithm to reduce the number of false positives—edges marked by the certificate algorithm that are actually strong—to avoid a wasted $\log n$ factor in the overall strength bounds. Because there are potentially n distinct strengths, we batch together all edges with strength in the same power of 2—which reduces the number of batches we need to find to $O(\log n)$. When graphs are superpolynomially weighted, the number of batches may be large, so we need to devise a few more tricks to keep the overall computation efficient.

We will focus most of our discussion on unweighted graphs, although the algorithms work unchanged for graphs with polynomially bounded integer weights. We

then show in section 8.6 how to generalize our algorithms to arbitrarily weighted graphs.

8.1. Grouping weak edges. Our approach begins for unweighted graphs with the following lemma.

LEMMA 8.1. *The total weight of a graph's k-weak edges is at most $k(n-1)$. In particular, any unweighted graph with total edge weight exceeding $k(n-1)$ has a nontrivial k-strong component (which may be the entire graph).*

Proof. Let S be the set of k-weak edges, and suppose that the total weight of edges in S exceeds $k(n-1)$. Then

$$\begin{aligned}\sum u_e/k_e &\geq \sum_{e\in S} u_e/k_e \\ &> \sum_{e\in S} u_e/k \\ &> k(n-1)/k \\ &= n-1,\end{aligned}$$

which contradicts Lemma 4.11. ∎

We apply this lemma first to unweighted graphs, for which Lemma 8.1 implies that there are at most $k(n-1)$ edges that are k-weak. For each value $k = 1, 2, 4, 8, \ldots, m$, we will find a set of $k(n-1)$ edges containing all the k-weak edges (note that every edge is $(m+1)$-weak). We set $k'_e = k/2$ for all edges that are in the k-weak set but not the $k/2$-weak set, thus establishing lower bounds $k'_e \leq k_e$ for which the compression theorem works. The strength-bound summation under this basic scheme would be

$$\sum \rho/k'_e \leq \sum_{i=0}^{\log m} 2^i(n-1)(\rho/2^{i-1}) = O(\rho n \log m).$$

We will eventually describe a more sophisticated scheme that eliminates the factor of $\log m$. It will also handle weighted graphs efficiently.

8.2. Sparse certificates. A basic tool that we use is *sparse certificates,* defined by Nagamochi and Ibaraki [30].

DEFINITION 8.2. *A* sparse k-connectivity certificate, *or simply a* k-certificate, *for an n-vertex unweighted graph G is a subgraph H of G such that*

1. *H has $k(n-1)$ edges, and*
2. *H contains all edges crossing cuts of value k or less.*

Note that any maximal set of k forests of G is a certificate, since it must contain all edges of any cut of value k or less—otherwise, some forest has no edge crossing the cut and can incorporate any unused one.

Nagamochi and Ibaraki gave an algorithm [30], which we call `Certificate`, that constructs a sparse k-connectivity certificate in $O(m)$ time on unweighted graphs, independent of k.

8.3. Finding k-weak edges. Although a sparse k-certificate contains all edges with *standard* connectivity less than k, it need not contain all edges with *strong* connectivity less than k, since some such edges might not cross any cut of value less than k. (For example, in a square with an added diagonal, the diagonal edge has strength only 2 but crosses no cut of value 2.) We must therefore perform some extra

PROCEDURE `WeakEdges`(G, k).

do $\log_2 n$ **times**
  $E' \leftarrow$ `Certificate`$(G, 4k)$
  output E'
  $G \leftarrow G - E'$
end do

FIG. 1. *Procedure* `WeakEdges` *for identifying* $k_e < k$.

work. In Figure 1 we give an algorithm `WeakEdges` for identifying edges with $k_e < k$. It uses the Nagamochi–Ibaraki `Certificate` algorithm as a subroutine.

THEOREM 8.3. `WeakEdges` *outputs a set of* $O(kn \log n)$ *edges containing all the* k*-weak edges of* G.

Proof. First suppose that G has no k-strong components, i.e., that $k_e < k$ for all edges. Then by Lemma 8.1 there are at most $k(n-1)$ edges in G; hence at least half of the vertices have at most $4k$ incident edges that define a cut of value at most $4k$ with a single vertex on one side. In an iteration of the loop in `WeakEdges`, these vertices become isolated after removing the sparse $4k$-certificate edges. Thus in a single loop iteration half of the nonisolated vertices of G become isolated. The remaining graph still has no k-strong edges, so we can repeat the argument. Hence in $\log_2 n$ rounds we isolate *all* vertices of G, which can only be done by removing all the edges. Thus all the edges of G are output by `WeakEdges`.

In the general case, let us obtain a new graph H by contracting each k-strong component of G to a vertex. Note that this leaves all k-weak edges uncontracted and k weak by Lemma 4.6. Any sparse $4k$-certificate of G contains the edges of a sparse $4k$-certificate of H as well. Thus by the previous paragraph, all edges of H are output by `WeakEdges`. But these are all the k-weak edges of G. □

8.4. Sparse partitions. Algorithm `WeakEdges` can be implemented via $O(\log n)$ calls to `Certificate`. It follows that it runs in $O(m \log n)$ time on unweighted graphs and outputs a set of at most $4k(n-1)\log n$ edges.[2] In this section, we eliminate a $\log n$ factor in this approach by finding edge sets that are "sparser" than the Nagamochi–Ibaraki certificate.

Observe that a given k-certificate E' may contain edges that are inside a connected component of $G - E'$. The edges in $G - E'$ do not cross any cut of value at most k (by definition of a sparse certificate), so the same holds for any edge of E' whose endpoints are connected by a path in $G - E'$. We can therefore remove any such edge from E' and put it back into G without affecting the correctness of the proof of Theorem 8.3.

We can find the remaining edge set by contracting all edges not in E', yielding a new graph G'. This contracts all (and only) edges connected by a path in $G - E'$. But observe that any edge crossing a cut of value at most k in G also crosses such a cut in G' since we contract no edge that crosses such a small cut. Thus we can find

[2]It also follows that a $(4k \log n)$-sparse certificate will contain all k-weak edges, so they can be found with a single `Certificate` invocation. This gives a better running time than our algorithm. Indeed, since the Nagamochi–Ibaraki algorithm "labels" each edge with the value k for which it vanishes, we can use those labels (divided by $\log n$) as strength lower-bounds, producing a complete result in $O(m + n \log n)$ time. However, this approach produces an extra $\log n$ factor in the edge bound (or worse in weighted graphs) that we have been unable to remove.

all edges crossing a small cut via a certificate in G'. Since G' has fewer vertices, the certificate has fewer edges. We can iterate this procedure until all edges of G' are in the certificate or until G' becomes a single vertex. In the latter case, the original graph is k-connected, while in the former, if the current contracted graph has n' vertices, it has at most $k(n'-1)$ edges. This motivates the following definition.

DEFINITION 8.4. *A* sparse k-partition*, or* k-partition*, of G is a set E' of edges of G such that*

1. *E' contains all edges crossing cuts of value k or less in G, and*
2. *If $G - E'$ has r connected components, then E' contains at most $2k(r-1)$ edges.*

In fact, the construction just described yields a graph with at most kr edges, but we have relaxed the definition to $2kr$ edges to allow for a more efficient construction.

Procedure `Partition` in Figure 2 outputs a sparse partition. It uses the Nagamochi–Ibaraki `Certificate` algorithm and obtains a new graph G' by contracting those edges not in the certificate. It repeats this process until the graph is sufficiently sparse. It bears some similarity to a minimum cut approximation algorithm invented by Matula [28].

```
PROCEDURE Partition(G, k).

input: An n-vertex m-edge graph G

    if m ≤ 2k(n − 1) then
        output the edges of G
    else
        E' ← Certificate(G, k)
        G' ← contract all edges of G − E'
        Partition(G', k)
```

FIG. 2. `Partition` *finds low-connectivity edges.*

LEMMA 8.5. `Partition` *outputs a sparse k-partition partition in $O(m+n)$ time on unweighted graphs.*

Proof. Correctness is clear from Lemma 4.6 since no edge crossing a cut of value less than k is ever contracted and at termination $m \leq 2k(n-1)$; we need only bound the running time.

The algorithm iterates only when $m > 2k(n-1)$. It finds a sparse connectivity certificate with $m' \leq k(n-1) \leq m/2$ edges and then contracts all edges not in the certificate, so at most half the edges remain. Also, if $n' - 1 > (n-1)/2$, then in the following iteration we will have $m' \leq k(n-1) < 2k(n'-1)$ and the algorithm will terminate. It follows that the number of vertices (minus one) also halves in every recursive call except the last.

A single iteration invokes the $O(m+n)$-time sparse-certificate algorithm [30]. Thus, our algorithm satisfies the recurrence $T(m,n) = m+n+T(m/2,n/2) = O(m+n)$. □

LEMMA 8.6. *If procedure* `Partition` *is used instead of* `Certificate` *in a call to* `WeakEdges`(G,k) *(meaning we invoke* `Partition`$(G,4k)$ *instead of* `Certificate`$(G,4k)$*), then algorithm* `WeakEdges` *runs in $O(m \log n)$ time on unweighted graphs and returns a partition of G into r components for some r. There are at most $8k(r-1)$ cross-partition edges, and they include all the k-weak edges of G.*

Note that the partition output by `WeakEdges` is itself almost a sparse k-partition; it simply has twice as many edges as the definition allows. On the other hand, it contains all k-weak edges, not just the ones crossing small cuts.

Proof. The running time follows from Lemma 8.5. To prove the edge bound, consider a particular connected component H remaining in a particular iteration of `WeakEdges`. A call to `Partition`$(H, 2k)$ returns a set of $8k(s-1)$ edges whose removal breaks that component into s subcomponents (the multiplier 8 arises from the fact that we look for a $4k$-partition). That is, it removes at most $8k(s-1)$ edges to increase the number of connected components by $s-1$. We can therefore charge $8k$ edges to each of the new components that gets created. Accumulating these charges over all the calls to `Partition` shows that if `WeakEdges` outputs $8k(r-1)$ edges, then those edges must split the graph into at least r components. □

8.5. Assigning estimates. We now give algorithm `Estimation` in Figure 3 for estimating strong connectivities. We use subroutine `WeakEdges` to find a small edge set containing all edges e with $k_e < k$, but replace the Nagamochi–Ibaraki `Certificate` implementation with our algorithm `Partition` to reduce the number of output edges.

We assign values $\tilde{k}_e$ as follows. In the first step, we run `WeakEdges` on G with $k = 2$; we set $\tilde{k}_e = 1$ for the edges in the output edge set E_0. Then we delete E_0 from G; this breaks G into connected components $G_1, \ldots, G_\ell$. Note that each edge in G_i has $k_e \geq 2$ in G, though possibly not in G_i. Then we recursively repeat this procedure in each G_i, by setting $k = 4$ in `WeakEdges` and labeling all output edges with $\tilde{k}_e = 2$, then with $k = 8, 16, \ldots, m$. At the ith step, all as-yet unlabeled edges have $k_e \geq 2^i$; we separate all those with $k_e < 2^{i+1}$ and give them (valid lower bound) label $\tilde{k}_e = 2^i$. Thus we find all $\tilde{k}_e$-values in at most $\log m$ iterations since m is the maximum strength of an edge in an unweighted graph.

PROCEDURE `Estimation`(H, k).

input: subgraph H of G

```
E' ← WeakEdges(H,2k)
for each e ∈ E'
    k_e ← k
for each nontrivial connected component H' ⊂ H − E'
    Estimation(H',2k)
```

FIG. 3. *Procedure* `Estimation` *for assigning* $\tilde{k}_e$*-values.*

LEMMA 8.7. *If H is any subgraph of G, then* `Estimation`(H, k) *assigns bounds $\tilde{k}_e$ such that any edge $e \in H$ with $k_e \geq k$ in G receives $\tilde{k}_e \leq k_e$.*

COROLLARY 8.8. *After a call to* `Estimation`$(G, 1)$, *all the labels $\tilde{k}_e$ satisfy $\tilde{k}_e \leq k_e$.*

Proof. We prove the lemma by downward induction on k. The claim is vacuously true for $k = m + 1$ since no $k_e > m$. For the inductive step, consider a call to `Estimation`(H, k). We consider two possibilities for edge e in H. If e is in the set E' returned by `WeakEdges`$(H, 2k)$, then it receives label k, which is a valid lower bound for any edge with $k_e \geq k$. So the inductive step is proven for $e \in E'$. On the other hand, if $e \notin E'$, then e is in some H' upon which the algorithm is invoked recursively. By the correctness of `WeakEdges` we know $k_e \geq 2k$ (in H, and thus in G) in this case. Thus, the inductive hypothesis applies to show that e receives a valid lower bound

upon invocation of `WeakEdges`$(H', 2k)$. □

LEMMA 8.9. *Assume that in procedure* `WeakEdges`*, procedure* `Certificate` *is replaced by* `Partition`*. Then the values* $\tilde{k}_e$ *output by* `Estimation`$(G, 1)$ *are such that* $\sum 1/\tilde{k}_e = O(n)$.

Proof. The proof is similar to the proof in Lemma 4.11 that $\sum u_e/k_e \leq n$. Define the *cost* of edge e to be $1/\tilde{k}_e$. We prove that the total cost assigned to edges is $O(n)$. Consider a call to `Estimation`(H, k) on some connected subgraph H of G. It invokes `WeakEdges`(H, k), which returns a set of $4k(r-1)$ edges whose removal partitions H into r connected components. (Note that possibly $r = 1$ if H is k-connected.) The algorithm assigns values $\tilde{k}_e = k$ to the removed edges. It follows that the total *cost* assigned to these edges is $4(r-1)$. In other words, at a cost of $4(r-1)$, the algorithm has increased the number of connected components by $r-1$. Ultimately, when all vertices have been isolated by edge removals, there are n components; thus, the total cost of the component creations is at most $4(n-1)$. □

In summary, our estimates $\tilde{k}_e$ satisfy the necessary "tight strength bounds" conditions for Definition 6.3 used in our compression and smoothing applications: $\tilde{k}_e \leq k_e$ and $\sum 1/\tilde{k}_e = O(n)$.

LEMMA 8.10. `Estimation` *runs in* $O(m \log^2 n)$ *time on an unweighted graph.*

Proof. Each level of recursion of `Estimation` calls subroutine `WeakEdges` on graphs of total size m. An unweighted graph has maximum strong connectivity m and therefore has $O(\log m)$ levels of recursion. □

8.6. Weighted graphs. Until now, we have focused on the estimation of edge strengths for unweighted graphs. Weighted graphs present complications. Real weights can be rounded to nearby rationals while introducing negligible changes in connectivity and strength. Rationals can be scaled up so that all capacities are integers, integer edge weights thought of as parallel unweighted edges, and unweighted-graph algorithms applied to the result. But the number of unweighted edges produced this way can be unbounded, impacting the running times of our unweighted-graph algorithms.

Nagamochi and Ibaraki give an $O(m + n \log n)$-time weighted-graph implementation of their `Certificate` algorithm [29]. (In weighted graphs, the k-sparse sparse certificate has an upper bound of $k(n-1)$ on the *total weight* of edges incorporated.) We can use the Nagamochi–Ibaraki weighted-graph algorithm to implement `Partition`(G, k) in $O(m \log n)$ time for any value of k. Unlike the unweighted case, the repeated calls to `Certificate` need not decrease the *number* of edges substantially (though their total weight will decrease). However, the claimed halving in vertices still happens. Thus algorithm `Partition` satisfies a recurrence $T(m, n) = O(m + n \log n) + T(m, n/2) = O(m \log n)$. Since `Partition` runs in $O(m \log n)$ time, we deduce that `WeakEdges` runs in $O(m \log^2 n)$ time.

A bigger problem arises in the iterations of `Estimation`. In an integer-weighted graph with maximum edge weight W, the k_e values may be as large as mW, meaning that $\Omega(\log mW)$ levels of recursion will apparently be required in `Estimation`. This is a problem if W is superpolynomial.[3] To deal with this problem, we show how to localize our computation of strong connectivities to a small "window" of relevant connectivity values.

We begin by computing a rough underestimate for the edge strengths. Construct

[3]Recall that, for a strongly polynomial bound, we are concerned with the *number of arithmetic operations*. Thus, the problem is not that operations on large numbers are slow, but that large numbers may force us to do more operations (proportional to $\log W$ in our case).

a maximum spanning tree (MST) for G using the edge weights u_e. Let d_e be the minimum weight of an edge on the MST path between the endpoints of e. The quantities d_e can be determined in $O(m)$ time for all e using an MST *sensitivity analysis* algorithm [10] (practical algorithms run in $O(m \log n)$ time and will not dominate our running time). Since the MST path between the endpoints of e forms a (nonmaximal) d_e-connected subgraph spanning e, we know that $k_e \geq d_e$. However, if we remove all edges of weight d_e or greater, then we disconnect the endpoints of e (this follows from MST properties [32]). There are at most $\binom{n}{2}$ such edges, so the weight removed is less than $n^2 d_e$. Therefore, $k_e \leq n^2 d_e$. This gives us an initial factor-of-n^2 estimate $d_e \leq k_e < n^2 d_e$.

Given the d_e, we compute the $\tilde{k}_e$ in a series of phases, each focusing on a set of edges in a narrow range of d_e values. In particular, we will contract all edges with d_e above some upper bound and delete all edges with d_e below some lower bound. Then we will use `Estimation` to assign $\tilde{k}_e$ labels to the edges that remain.

We label our edges in a series of phases. In a phase, let D be the maximum d_e on any unlabeled edge. Each phase will decrease D. Since $k_e < n^2 d_e$, the maximum strength of any unlabeled edge is less than $n^2 D$. Our goal in this phase is to (validly) label all edges with $d_e \geq D/n$. We begin by contracting all edges of weight at least $n^2 D$ (which will already have labels from previous phases since $d_e \geq u_e$). These edges all trivially have strength at least $n^2 D$ which, by Lemma 4.6, means that the contractions do not affect strengths of edges with $k_e < n^2 D$ (which includes all unlabeled edges). In the resulting graph, let us delete all edges with $d_e < D/n$ (none of which we need to label in this phase). The deletions may decrease certain strengths but not increase them (Lemma 4.5). It follows that every unlabeled edge (all of which have $k_e < n^2 D$) has strength in the modified graph *no greater* than in G.

On each connected component H induced by the remaining edges, execute `Estimation`$(H, D/n)$. By Lemma 8.7, this assigns valid lower-bound labels to all edges e with strength at least D/n in the modified graph. In particular, the labels are valid for all e with $d_e \geq D/n$ (since any edge with $d_e \geq D/n$ is connected by a path of edges of value at least D/n, none of which get deleted in the phase). These labels are valid lower bounds for strengths in the modified graph and, as discussed in the previous paragraph, all unlabeled edges have strengths in the subgraph no greater than their strength in G. Thus, the computed labels can be used as valid labels for all the unlabeled edges with $d_e \geq D/n$.

The phase just described has computed labels for each unlabeled edge with $d_e \geq D/n$. We have therefore reduced the maximum d_e on any unlabeled edge by a factor of n. We iterate this process, repeatedly decreasing the maximum unlabeled d_e, until all edges are labeled.

Summarizing our discussion above gives the algorithm `WindowEstimation` listed in Figure 4. To make this procedure efficient, instead of repeatedly starting with G and deleting edges with small d_e in each phase, we start with a large D and all edges deleted, then repeatedly decrease D and "undelete" the edges that are needed in the given phase.

LEMMA 8.11. *Procedure* `WindowEstimation` *can be implemented to run in only* $O(m \log^3 n)$ *time.*

Proof. The contractions in `WindowEstimation` can be implemented using a standard union-find data structure [9]. Each time an edge is contracted, a union is called on its endpoints. Each time an edge is added from L, find operations can identify its endpoints. Therefore, the additions and contractions of edges do not affect the running

PROCEDURE `WindowEstimation`(G).

Sort the edges in decreasing order of d_e into a list L
Initialize G' as an empty graph on the vertices of G
repeat
  let $D \leftarrow$ maximum d_e among unlabeled edges in L
  contract every $e \in G'$ with $d_e \geq n^2 D$
  move every edge $e \in L$ with $d_e \geq D/n$ to G'
  call `Estimation`($G', D/n$) to get labels $\tilde{k}_e$
    for the *new* edges added from L in this phase
until no edges remain

FIG. 4. `WindowEstimation` *for weighted graphs.*

time. Instead, the running time is determined by the repeated calls to `Estimation`.

Consider a particular iteration of the loop with some D value. We initially contract all edges with $d_e \geq n^2 D$, so the total remaining weight and thus maximum strength in the resulting graph is less than $n^4 D$. We invoke `Estimation` with a starting strength argument of D/n, which means that it terminates in $O(\log n)$ iterations (the number of `Estimation` argument doublings from D/n to $n^4 D$). As to the size of the problem, recall that we contracted all edges with with $d_e \geq n^2 D$ and deleted all edges with $d_e < D/n$. It follows that our running time is proportional to $m' \log^3 n$, where m' is the number of edges with $D/n \leq d_e < D$. (Note that we can gather all relevant vertices by considering the extant edges, which lets us ignore isolated vertices and ensures that the number of vertices we work with is $O(m')$.)

Now we can bound the running time over all phases. An edge e is present (neither contracted nor deleted) if and only if $D/n \leq d_e < n^2 D$. Since the threshold D decreases by a factor of n each time, this means that edge e contributes to the size of the evaluated subgraph in at most three iterations. In other words, the sum of m' values over all iterations of our algorithm is $3m$. It follows that the overall running time of these iterations is $O(\sum m' \log^3 n) = O(m \log^3 n)$. □

LEMMA 8.12. *Procedure* `WindowEstimation` *assigns labels such that* $\sum u_e/\tilde{k}_e = O(n)$

Proof. Recall the definition in Lemma 8.9 of the cost of edge e as $u_e/\tilde{k}_e$. Our algorithm incorporates some of the labels computed by `Estimation` in each phase, contributing their cost (in that phase) to the final total cost. We show that the total cost of *all* labels computed over all the phases is $O(n)$.

Recall that the *rank* of a graph is equal to the number of edges in a spanning forest of the graph. Inspection of `Partition` shows that the total weight of edges returned by `Partition`(G, k) is at most 4 times the rank of G. Similarly, inspection of `Estimation` shows that on a rank-r graph its results satisfy $\sum u_e/\tilde{k}_e = O(r)$.

The phase at parameter D run with all edges of weight exceeding Dn^2 contracted and all edges of weight less than D/n not yet inserted. By the properties of MSTs, the resulting graph is precisely spanned by the set of MST edges with weights in this range. That is, the rank of this graph is equal to the number r_D of such MST edges. It follows that the total cost $\sum u_e/\tilde{k}_e$ of `Estimation` labels in this phase is $O(r_D)$. Now note that each MST edge contributes to r_D only when its weight is between D

and Dn^2, which happens in at most three phases since D decreases by n each phase. Thus, each edge contributes to three values r_D, so $\sum r_D \leq 3(n-1)$. This bounds the total cost by $O(\sum r_D) = O(n)$, as desired. □

9. Conclusion. We have given stronger applications of random sampling to problems involving cuts in graphs. The natural open question is whether these approximation algorithms can be made exact. An initial step toward the answer [19] only gave a useful speed-up for graphs with large minimum cuts. Later, sampling led to an exact linear-time algorithm for minimum cuts [23]; however, the techniques used there appear to be specialized to that particular problem. In the article immediately following this one, Karger and Levine [24] give very fast algorithms for small flows in unweighted graphs; the important remaining question is to develop fast exact algorithms for weighted graphs.

Subsequent to the initial publication of this work [6], more powerful results on graph compression have been attained. Spielman and Srivastava [31] gave a near-linear-time randomized algorithm that compressed graphs to satisfy all *quadratic forms* over the graph—a strict generalization of preserving cut values. Their algorithm uses a similar approach, compressing each edge with some probability. Instead of inverse to strong connectivities, their sampling probabilities are proportional to the *effective resistances* of graph edges. Intriguingly, (inverse) strong connectivity and effective resistance are *incomparable*, with each possibly smaller than the other by a factor of n [12]. This suggests that neither measure is the true "smallest achievable" sampling probability for cut sampling. It would be interesting to show that the minimum of the two sampling probabilities was permissible. Along these lines, Fung et al. [12] proved that it is valid to sample with probability inversely proportional to the *standard* connectivity of an edge's endpoints, which lower bounds both the above-mentioned measures. However, the constant of proportionality must be increased by an $O(\log n)$ factor, yielding a sampled graph with $O(n \log^2 n)$ edges.

Our general approach of choosing edges independently at random seems unlikely to yield compression to $o(n \log n)$ size. Even with just a single cut, the Chernoff bound breaks down when the expected number of edges in the sample is $o(\log n)$—indeed, with expectation $o(\log n)$ there is a nonnegligible chance of sampling no edges at all from a given cut! Nonetheless, Batson, Spielman, and Srivastava [5] showed that graphs could be compressed deterministically to $O(n)$ edges, improving on our $O(n \log n)$ result, although the algorithm given for doing so takes time $O(n^3 m)$, which is significantly slower. The proof techniques are heavily algebraic. An interesting open question is whether we can attain similar bounds using combinatorial arguments.

REFERENCES

[1] ACM-SIAM, *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.

[3] D. Aldous, *Random walks on finite groups and rapidly mixing Markov chains*, in Siminaire de Probabilites XVII 1981/1982, Lecture Notes in Math., Springer-Verlag, Berlin, 1983, pp. 243–297.

[4] S. Arora, E. Hazan, and S. Kale, $O(\sqrt{\log n})$ *approximation to* Sparsest cut *in* $\tilde{O}(n^2)$ *time*, SIAM J. Comput., 39 (2010), pp. 1748–1771.

[5] J. Batson, D. A. Spielman, and N. Srivastava, *Twice-Ramanujan sparsifiers*, SIAM J. Comput., 41 (2012), pp. 1704–1721.

[6] A. A. Benczúr and D. R. Karger, *Approximate s–t min-cuts in* $\tilde{O}(n^2)$ *time*, in Proceedings of the 28th ACM Symposium on Theory of Computing, Gary Miller, ed., ACM Press, New

York, 1996, pp. 47–55.
[7] H. Chernoff, *A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statist., 23 (1952), pp. 493–509.
[8] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng, *Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs*, in Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC '11), ACM, New York, 2011, pp. 273–282.
[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
[10] B. Dixon, M. Rauch, and R. E. Tarjan, *Verification and sensitivity analysis of minimum spanning trees in linear time*, SIAM J. Comput., 21 (1992), pp. 1184–1192.
[11] L. R. Ford, Jr., and D. R. Fulkerson, *Maximal flow through a network*, Canadian J. Math., 8 (1956), pp. 399–404.
[12] W. S. Fung, R. Hariharan, N. Harvey, A. J., and D. Panigrahi, *A general framework for graph sparsification*, in Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC '11), ACM, New York, 2011.
[13] A. Goldberg and S. Rao, *Beyond the flow decomposition barrier*, J. ACM, 45 (1998), pp. 783–797.
[14] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum flow problem*, J. ACM, 35 (1988), pp. 921–940.
[15] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1964), pp. 13–30.
[16] M. R. Jerrum, *A very simple algorithm for estimating the number of k-colourings of a low-degree graph*, Random Structures Algorithms, 7 (1995), pp. 157–165.
[17] R. Kannan, S. Vempala, and A. Vetta, *On clusterings: Good, bad and spectral*, J. ACM, 51 (2004), pp. 497–515.
[18] D. R. Karger, *Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple min-cut algorithm*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1993, pp. 21–30; later in J. ACM, 43.
[19] D. R. Karger, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res., 24 (1999), pp. 383–413.
[20] D. R. Karger, *Random Sampling in Graph Optimization Problems*, Ph.D. thesis, Stanford University, Stanford, CA, 1994.
[21] D. R. Karger, *Better random sampling algorithms for flows in undirected graphs*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1998, pp. 490–499.
[22] D. R. Karger, *A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem*, SIAM J. Comput., 29 (1999), pp. 492–514; corrected version in SIAM Rev., 43 (2001), pp. 499–522.
[23] D. R. Karger, *Minimum cuts in near-linear time*, J. ACM, 47 (2000), pp. 46–76.
[24] D. R. Karger and M. S. Levine, *Fast augmenting paths by random sampling from residual graphs*, SIAM J. Comput., (2014), to appear.
[25] D. R. Karger and C. Stein, *A new approach to the minimum cut problem*, J. ACM, 43 (1996), pp. 601–640.
[26] P. N. Klein, C. Stein, and É. Tardos, *Leighton-Rao might be practical: Faster approximation algorithms for concurrent flow with uniform capacities*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, ACM, 1990, pp. 310–321.
[27] M. V. Lomonosov and V. P. Polesskii, *Lower bound of network reliability*, Probl. Inf. Transm., 7 (1971), pp. 118–123.
[28] D. W. Matula, *A linear time $2+\epsilon$ approximation algorithm for edge connectivity*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 500–504.
[29] H. Nagamochi and T. Ibaraki, *Computing edge-connectivity in multigraphs and capacitated graphs*, SIAM J. Discrete Math., 5 (1992), pp. 54–66.
[30] H. Nagamochi and T. Ibaraki, *Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs*, Algorithmica, 7 (1992), pp. 583–596.
[31] D. A. Spielman and N. Srivastava, *Graph sparsification by effective resistances*, SIAM J. Comput., 40 (2011), pp. 1913–1926.
[32] R. E. Tarjan, *Data Structures and Network Algorithms*, CBMS-NSF Reg. Conf. Ser. Appl. Math. 44, SIAM, Philadelphia, 1983.
[33] S. Vadhan, ed., *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC'11)*, ACM, New York, 2011.