A Fast and Simple Unbiased Estimator for Network (Un)reliability

David R. Karger
MIT CSAIL
Cambridge, MA, USA
Email: karger@mit.edu

***Abstract*—The following procedure yields an *unbiased estimator* for the disconnection probability of an n-vertex graph with minimum cut c if every edge fails independently with probability p: (i) contract every edge independently with probability $1-n^{-2/c}$, then (ii) recursively compute the disconnection probability of the resulting tiny graph if each edge fails with probability $n^{2/c}p$. We give a short, simple, self-contained proof that this estimator can be computed in linear time and has relative variance $O(n^2)$. Combining these two facts with a standard sparsification argument yields an $O(n^3 \log n)$-time algorithm for estimating the (un)reliability of a network. We also show how the technique can be used to create unbiased samples of disconnected networks.**

***Keywords*-Network reliability; minimum cut; random graph**

I. Introduction

In the network reliability problem we are given a graph G and a failure probability p and seek to compute the probability $u_G(p)$ that G becomes disconnected when edges fail independently with probability p.

This problem is ♯P-complete [16], [15]. Karger [5] gave the first *fully polynomial randomized approximation scheme (FPRAS)* for $u_G(p)$, with a running time of roughly $\tilde{O}(n^5)$.[1] Harris and Srinivasan [3] improved the runtime to $n^{3+o(1)}$ at the cost of a more complex algorithm and analysis.

In this paper, we give a short, simple, self contained *unbiased estimator* for network reliability that can be computed in linear time and has relative variance $O(n^2)$. It uses only one nontrivial fact—that the cycle is the least reliable network with a given minimum cut. Combined with some standard methods of graph sparsification [2], this unbiased estimator yields an FPRAS for network reliability running in $O(n^3 \log n)$ time.

The estimator is based entirely on taking random samples and testing whether the resulting graph is connected—and therefore requires no complex data structures. As with previous algorithms, the challenge is dealing with too-reliable graphs where direct random simulation of edge failures is unlikely to generate any disconnected graphs that can be used to estimate the disconnection probability. Our key trick is to use a kind of *importance sampling:* we raise the edge failure rate so that the failures become common enough to measure in the sampling process, then adjust for our tinkering to determine the actual reliability.

As is already well known, reliability is tied to the minimum cut c of the graph G. Recall that c can be found in $\tilde{O}(m)$ time with high probability [6] or in $O(mn)$ time deterministically [13]; computing it is not a bottleneck in the algorithms we develop here.

A. Background: Monte Carlo Estimation

Our approach is *Monte Carlo estimation*. We devise a random variable X whose expectation is equal to the quantity we wish to measure. We then average samples from X to estimate this expectation. The number of samples needed is determined by the *relative variance* $r = \sigma^2/\mu^2$ where $\mu = E[X]$ and $\sigma = E[(X-\mu)^2]$. Since $E[(X-\mu)^2] = E[X^2] - \mu^2$ we can also write $r = E[X^2]/\mu - 1$.

Lemma I.1. *If X has relative variance $\epsilon^2/4$ then X is within a relative factor ϵ of its expectation with probability $1/4$.*

Proof: By the Markov (or Chebyshev) bound,

$$\begin{aligned}\Pr[|X-\mu| > \epsilon\mu] &= \Pr[(X-\mu)^2 > \epsilon^2\mu^2] \\ &\le E[(X-\mu)^2]/\epsilon^2\mu^2] \\ &\le 1/4\end{aligned}$$

given the relative variance of X. ■

Lemma I.2. *If X has relative variance r then the average of t independent samples of X has relative variance r/t.*

Proof: The sum of the samples has mean $t\mu$ and, since they are independent, variance $t\sigma^2$. ■

Setting $t = 4r\epsilon^{-2}$ and combining these two lemmas yields the general approach:

Corollary I.3. *If X has relative variance r then the average of $4r\epsilon^{-2}$ samples of X will yield a $(1 \pm \epsilon)$-approximation to $E[X]$ with probability $3/4$.*

Since Corollary I.3 yields a failure probability of $1/4$, repeating the entire experiment $O(\log 1/\delta)$ times and taking the median reduces the failure probability to δ. In other words, the number of samples needed for an (ϵ, δ) approximation (one that is accurate to within ϵ with probability $1/\delta$) is $O(r\epsilon^{-2} \log 1/\delta)$. As this $\epsilon^{-2} \log 1/\delta$ term attaches to any such sampling scheme, we'll ignore it in the rest of the paper.

This work was supported by a grant from the National Science Foundation.

[1]Technically, we are estimating network *unreliability*. For exact algorithms the two problems are identical, but an approximation to one does not translate to the other. Estimating reliability—i.e. the likelihood of staying connected—is most difficult on unreliable graphs which are unimportant in practice. There is no known FPRAS for this problem. But for reliable graphs an FPRAS for reliability is trivial while estimating unreliability is the harder problem.

A direct application of this idea to estimate $u_G(p)$ is to delete edges of G with probability p and test whether the result is disconnected. The indicator variable for this event has expectation $\mu = u_G(p)$ and variance $\sigma^2 = u_G(p)(1-u_G(p))$. Thus the relative variance is $1/u_G(p) - 1 \leq 1/u_G(p)$ and we get an (ϵ, δ) approximation using $\tilde{O}(1/u_G(p))$ samples. Each sample can be constructed in $O(m)$ time for an overall runtime of $O(m/u_G(p))$.

If $u_G(p) \geq (\log^2 n)/n^2$ this gives an $O(mn^2/\log^2 n)$ runtime. In particular, since the graph disconnects whenever a specific cut fails, we know $u_G(p) \geq p^c$, which means we can use this estimator so long as $p^c \geq (\log^2 n)/n^2$. But for smaller p the runtime becomes prohibitively large. Previous work switched to a technique based on enumeration of small cuts with a running time of $\tilde{O}(n^5)$. We take a different approach.

II. A Fast Estimator

When $p^c \leq (\log^2 n)/n^2$, the following procedure provides an unbiased estimator for $u_G(p)$—that is, a random variable whose expectation is $u_G(p)$.

1) Set $q \geq p$ such that $q^{c/2} = (\log n)/n$, and contract edges of G with probability $1-q$ to yield a graph H
2) Return $u_H(p/q)$, which can be computed easily since H will be tiny

To see that this is an unbiased estimator, note that in the first phase each edge is *not* contracted with probability q. In the second phase we then consider deleting these uncontracted edges with probability p/q, and assess the probability that a cut of H (which is also a cut of G) fails. Putting these two phases together, we are assessing the probability that the set of edges that go uncontracted in the first phase (probability q) and then get deleted in the second phase (probability p/q, for an overall probability of $q \cdot (p/q) = p$) includes a cut of G. In other words, $E_H[u_H(p/q)] = u_G(p)$.

We now show that this unbiased estimator is useful: that its variance is small, and that it can be computed quickly, because H is probably tiny.

A. Bounding the Variance

We begin by bounding the variance. Let $G(p)$ denote the (random) result of deleting edges from G with probability p.

Lemma II.1 (Cycle Lemma)**.** *If G has n vertices and min-cut c then the probability that $G(p)$ has more than k components is at most $\binom{n}{k}p^{kc/2} \leq (p^{c/2}en/k)^k$.*

Corollary II.2. $p^c \leq u_G(p) \leq n^2 p^c$.

Corollary II.3 (Shrinkage Corollary)**.** *If $p^{c/2}n \geq \log n$ then $G(p)$ has $O(p^{c/2}n)$ components with high probability.*

Proof: Let C be an n-vertex cycle where each pair of adjacent vertices is connected by a "bundle" of $c/2$ edges. As shown by Benczur and Karger [2] (we give the proof in an appendix for completeness), the number of components produced by edge failures in G is *stochastically dominated* by the number of components produced by edge failures in C. In other words, if h_G and h_C are random variables respectively denoting the number of components produced by the edge failures in G and in C, then $\Pr[h_G \geq k] \leq \Pr[h_C \leq k]$. In C, the probability we get k or more components is just the probability that at least k bundles fail. We can union bound it by considering all $\binom{n}{k}$ groups of k bundles and noting that the probability that all them fail is $(p^{c/2})^k$.

For the first corollary, note for the lower bound that $u_G(p)$ must exceed the probability that a single min-cut fails. For the upper bound, just set $k = 2$ in Cycle Lemma II.1.

For the second corollary, if $p^{c/2}n \geq \log n$ then we can set $k = b \cdot 2ep^{c/2}n$ in Cycle Lemma II.1 and get a bound of $(1/2)^{\Omega(b \log n)} = n^{-\Omega(b)}$ which can be made an arbitrarily small polynomial by choice of constant b.

This result can be proven 2 other ways [10], [6]. ■

We now bound the variance $E[u_H(p/q)^2]$. Let h be the (random) number of vertices of H. Write $u_H = u_H(p/q)$.

Cycle Lemma II.1 shows that the cycle is the most unreliable h-vertex graph, with unreliability $u_H \leq h^2(p/q)^c$. Thus, $E[u_H^2] \leq E[h^2(p/q)^c u_H] = E[h^2 u_H]$. Shrinkage Corollary II.3 shows that that there is a $k = O(\log n)$ such that $h \leq k$ with high probability. Intuitively, this means that conditioning on $h \leq k$ should not change outcomes. If so, $E[u_H^2] \leq E[k^2(p/q)^c u_H] = k^2(p/q)^c E[u_H] = k^2(p/q)^c u_G(p)$. This would make the relative variance at most $k^2(p/q)^c/u_G(p)$. Since $u_G(p) \geq p^c$ (Corollary II.2), the relative variance is at most $k^2/q^c = O((\log^2 n)/((\log n)/n)^2) = O(n^2)$.

This argument is incomplete since it ignores the unlikely but possible event of H being too large. To formalize it, we introduce a lemma.

Lemma II.4. *If X with mean μ has $X \leq r\mu$ with probability $1-\epsilon^2$ at least and $X \leq \mu/\epsilon$ with probability 1, then X has relative variance at most r.*

Proof: Let I be the indicator that $X \leq r\mu$. Then

$$\begin{aligned}
E[X^2] &\leq E[(I + 1 - I)X^2] \\
&= E[IX^2] + E[(1-I)X^2] \\
&\leq E[(IX)X] + E[(1-I)(\mu/\epsilon)^2] && \text{since } X \leq \mu/\epsilon \\
&\leq E[r\mu X] + (\mu/\epsilon)^2 E[(1-I)] && I = 0 \text{ if } X > r\mu \\
&= r\mu^2 + (\mu/\epsilon)^2 \Pr[\overline{I}] \\
&= r\mu^2 + \mu^2 && \text{since } \Pr[\overline{I}] \leq \epsilon^2
\end{aligned}$$

Thus the relative variance is $E[X^2]/\mu^2 - 1 = r$. ■

We use this lemma to formalize our argument. Take $\epsilon = O(\log^2 n)/n^4$. We know that $E[u_H(p/q)] = u_G(p) \geq p^c$. Furthermore, since any H we generate is a contraction of G and thus no less reliable, we know that $u_H(p/q) \leq u_G(p/q) \leq n^2(p/q)^c = O(n^4 p^c/\log^2 n) = p^c/\epsilon$ with certainty. At the same time, we've argued that there is a $k = O(\log n)$ such that the probability that H has fewer than k vertices, making $u_H(p/q) \leq k^2(p/q)^c$, is $1 - 1/n^8 \geq 1 - \epsilon^2$. Thus, by our lemma, $u_H(p/q)$ has relative variance at most $k^2/q^c = O(n^2)$.

B. Small Graphs

Generating the sample H is straightforward in $O(m)$ time. But our algorithm also needs a base-case algorithm for computing the reliability of these small sampled graphs. We could use any existing polynomial time approximation algorithm—since it runs on graphs of size $O(\log n)$, the exact runtime is unimportant. But this is unsatisfactory. First, it requires invoking a different, more complex algorithm. Second, since these algorithms are approximation algorithms, they return an inexact value, which means that our algorithm no longer offers an unbiased estimator for the reliability.

An alternative is to use a brute-force exact algorithm. We can enumerate over all possible edge configurations, summing the probabilities of those that produce disconnected graphs. But this takes exponential time which is too slow even on our tiny sample.

Thus, given our sampled subgraph H of size $O(\log n)$ and the target failure probability $p' = p/q$ we instead apply, recursively, a variant of our main algorithm to H. We run the main algorithm on H but with a different intermediate sampling probability q' such that $(q')^c = n^{-1/2}$.

This means that if $u_H(p') \geq n^{-1/2}$ we use naive Monte Carlo sampling. For smaller unreliabilities, we generate a contracted graph $H' = H(q')$ and recursively compute its reliability $u_{H'}(p/q')$. According to Shrinkage Corollary II.3, the probability that H' has size exceeding 16 vertices is at most

$$\left(\frac{e \log n}{16\sqrt{n}}\right)^{16} = O(n^{-7}).$$

Of course, we can exactly compute the reliability of a 16-vertex graph in constant time.

Our analysis shows that the relative variance of this sub-experiment as an estimator for $u_H(p/q)$ is $O(\sqrt{n})$ in either case (naive Monte Carlo or the contracted graph estimator) which is very large. However, we can reduce this variance by repeating the experiment: Lemma I.2 shows that doing so $\sqrt{n}$ times and taking the average yields an (unbiased) estimator for the reliability of H with relative variance $O(1)$.

Since H has size $O(\log n)$, the time to run these experiments is $O(\sqrt{n} \log^2 n)$. Which is negligible compared to the $O(m)$ time it takes to generate H in the first place.

Since we do not compute the reliability of H exactly, we need to track some additional error in estimating $u_G(p)$. Previously, we bounded the relative variance of $u_H(p/q)$ (computed exactly) as an estimator for $u_G(p)$, showing that $E[u_H(p/q)^2] = O(n^2 E_H[u_H(p/q)]^2)$. Now instead of returning $u_H(p/q)$ we are returning an estimator X for $u_H(p/q)$ that has relative variance $O(1)$. That is, $E_{H'}[X^2] = O(E_{H'}[X]^2)$. But $E_{H'}[X] = u_H(p/q)$, so $E_{H'}[X^2] = O(u_H(p/q)^2)$, which in turn implies that $E_H[E_{H'}[X^2]] = O(E_H[u_H(p/q)^2]) = O(n^2 u_G(p))$. In other words, X has relative variance $O(n^2)$ as an unbiased estimator for $u_G(p)$.

The above algorithm is Monte Carlo—it runs in polynomial time with high probability. In the rare event that a sample is too large the brute-force base algorithm will be too slow. For a Las Vegas algorithm—one with expected polynomial running time—we need to work a little bit harder, ensuring that we don't waste too much time in the slow/failure case. To do so we simply run our original unbiased estimator algorithm recursively. We address this detail below.

In summary, using the sampling algorithm recursively yields an unbiased estimator for $u_H(p)$ that can be computed in $O(m)$ time with high probability and has relative variance $O(n^2)$. Combining with the naive Monte Carlo algorithm for $p^c > n^{-2}$ and applying Corollary I.3 immediately yields an $O(m(n/\epsilon)^2 \log 1/\delta)$-time approximation algorithm for $u_G(p)$.

C. A More Precise Statement

Our algorithm and analysis relies on two key observations. First, we used the fact that $G(q)$ has $O(\log n)$ vertices with high probability. Second, we bounded the denominator $u_G(p) \geq p^c$. Thus, our analysis (and algorithm) provides the stronger result that for *any* q such that $G(q)$ has size $O(\log n)$ with high probability, the relative variance is bounded by

$$O\left(\frac{p^c \log^2 n}{q^c u_G(p)}\right)$$

In our work, we used $q^c = (\log^2 n)/n^2$ and $u_G(p) \geq p^c$ to get a bound of $O(1/n^2)$. But let's consider some specific graphs.

First, consider a pair of cliques connected by c edges. In this graph, $u_G(p) \approx p^c$ and we can only "discover" this in $G(q)$ if the unique min-cut survives the sample, which happens with probability q^c. It follows that $1/q^c$ is a lower bound on the relative variance as well as on the number of samples we must take to estimate the reliability with our approach.

On the other hand, consider the n-vertex cycle. In this graph, at sampling rate q^c we expect $nq^{c/2}$ "bundles" of $c/2$ edges to fail constructing G, meaning H will have $nq^{c/2}$ components. Thus, we can't make q^c much bigger than n^{-2} if we want H to be small—which we rely on both to prove a small relative variance and to compute $u_H(p/q)$ quickly.

In a sense, these two graphs are the extremes: the pair of cliques is the most reliable graph with min-cut c and the cycle is the least reliable. Combining the observations about these two graphs suggests that our algorithm is picking the best possible q: one as large as possible so that we won't miss the min-cut between cliques, but not too large to make the cycle tractable.

But there may be a way out, which is to choose q dependent on G. In the n-vertex cycle, the denominator $u_G(p) = \Theta(n^2 p^c)$ when $p^c \leq n^{-2}$, which yields a relative variance of $O(1)$ when $q^c = n^{-2}$. On the other hand, for pair of cliques the reliability is the worst case p^c. However, we can set $q^c = 1/2$ and conclude that $G(q)$ has 2 vertices with high probability; thus with this q we will have relative variance $O(q^{-c}) = O(1)$.

We conjecture that these two extremes also reflect the middle: that any graph where the reliability $u_G(p) \ll n^2 p^c$ is also a graph where $G(q)$ has few components even when $q \gg 1/n^2$. This would allow us to build our estimator using a large value of q, which would improve the relative variance which is proportional to $1/q^c$. If true, this conjecture would yield a better runtime for approximating network reliability.

D. A Las Vegas Algorithm

The algorithm we described above is Monte Carlo; it runs quickly with high probability but may be slow in the (unlikely) event that our samples H have size exceeding $O(\log n)$. We now address this imperfection in a natural way. Instead of assuming that H has size $O(\log n)$ so that we can run $\sqrt{n}$ sub-experiments to estimate its variance, we simply apply our original estimation algorithm, recursively, regardless of the size of H. The resulting algorithm for an n-vertex graph G is

1) if n is less than a suitable constant compute $u_G(p)$ by brute force;
2) otherwise if $p^c > n^2$ then average $O(n^2)$ trials of naive Monte Carlo estimation;
3) otherwise generate $O(n^2)$ samples $H \sim G(q)$,
4) for each, recursively compute an estimator of constant relative variance for $u_H(p/q)$,
5) and return the average of the results

We argue by induction that this algorithm produces an estimator with relative variance less than some constant r. This is immediate in the first two cases. If on the other hand we recurse, then by induction our recursive call for each $u_H(p/q)$ provides an estimate X_i of $u_H(p/q)$ with relative variance r. But $u_H(p/q)$ is itself an estimator of relative variance $O(n^2)$ for $u_G(p)$. Thus, as we argued before, each X_i has relative variance $O(rn^2)$ as an estimator for $u_G(p)$. It follows from Lemma I.2 that if we average $O(rn^2) = O(n^2)$ of these recursive estimates, we get an estimator for $u_G(p)$ with relative variance r as required for the induction.

As for the runtime, we know that *all* the subproblems have size $O(\log n)$ with high probability while in the worst case (which happens with probability say $O(1/n^3)$) all have size n. It follows that we can write a probabilistic recurrence [11] for the runtime, that $T(n) \leq n^2(m + T(H))$ where $T(H)$ denotes the (random) runtime on the largest subproblem. Taking expectations, we conclude that $ET(n) \leq n^2(m + ET(\log n) + n^{-1}T(n))$, which we can solve as $ET(n) \leq (1 + 1/n)n^2(m + ET(\log n)) = O(mn^2)$.

This analysis highlights the convenience of working with unbiased estimators instead of the more typical "probably approximately correct" estimators. Since unbiased estimators are characterized entirely by their relative variance, which is an expectation, we don't need to to keep track of or perform union bounds over low probability events over the entire recursion—we simply accumulate expectations.

E. Capacitated Graphs and Varying Failure Probabilities

So far we've worked with each edge separately and assumed all have the same failure probability. More generally we might want to consider *capacitated* graphs with (integer) capacities representing the number of edges connecting a pair of vertices. Our algorithms apply unchanged in this setting. They are based only on the primitive operation of sampling each edge with probability p (for naive Monte Carlo) or q and then p/q (for small p). In either case, we can sample an entire edge of capacity w in constant time by sampling a binomial distribution with parameters w and p (or q).

We address varying failure probabilities using a thought experiment. Given a graph with failure probability p_e on edge e, choose a large k and imagine replacing e with $\lfloor -k \ln p_e \rfloor$ edges that all fail with probability $1 - 1/k$. Their (common) endpoints disconnect when all the parallel edges fail, which happens with probability $(1 - 1/k)^{-\lfloor k \ln p_e \rfloor} \to p_e$ in the limit of large k. This shows that we can simulate the varying edge failures with the limit of a uniform $1 - 1/k$ failure rate.

To implement our algorithm efficiently we don't want to actually generate these k-multiplicity edges, and we don't have to. We simply need to determine how many of these edges survive in our sample $G(p)$ or $G(q)$, which involves sampling from a binomial distribution with parameters q and k. Taking the limit as $k \to \infty$ (and q shrinks to compensate) this distribution converges to a Poisson distribution whose samples can be generated easily.

III. Sparse Sampling

Our estimator is simple but slowed by its dependence on m. We now show how to reduce m to $O(n \log n)$ using some ideas from graph sparsification [2]. This improves our estimation algorithm's overall runtime to $O(n^3 \log n)$.

A. Inclusion Sampling

Our first insight is that since most edges won't be sampled, we shouldn't bother looking at them to build the sample. Suppose first that $p^c \geq (\log^2 n)/n^2$ so that we use naive Monte Carlo sampling. If we write the probability $\delta = 1 - p$ that a particular edge *survives*, then the relation $(\log^2 n)/n^2 < p^c = (1-\delta)^c < e^{-\delta c}$ means that $\delta = O(\log n)/c$. (Intuitively, if we want a probability exceeding $1/n^2$ to sample no edges from a min-cut, then the expected number of edges we sample from the min-cut must be $O(\log n)$.) It follows that the expected number of edges we sample is $O(m(\log n)/c)$. And since this a (large) set of independent edge samples, the bound holds with high probability as well.

This gives us a fast alternative to testing disconnection by deleting edges. To generate a sample, first use a binomial distribution $B(m, \delta)$ to choose the number of edges that survive. Then, sample that many edges from the graph. If the edges are stored in an array, then these edges can be sampled at a cost of $O(1)$ per sample. This gives us a set of $O(m(\log n)/c)$ edges (with high probability); we can determine whether the sample is connected in one $O(m(\log n)/c)$ time connected components computation. We'll refer to this technique as *inclusion sampling.*

Using inclusion sampling, we can run our naive Monte Carlo sampling algorithm in $O(m(\log n)/c)$ time per sample. We use this algorithm when $p^c \geq (\log^2 n)/n^2$, which means that we need $O(n^2/\log^2 n)$ samples for an overall running time of $O(mn^2/c \log n)$.

In a similar vein, the small-p version of our unbiased estimator begins by sampling edges from G with probability q where $q^c = O((\log^2 n)/n^2)$. So just as already argued above, inclusion sampling will sample only $O(m(\log n)/c)$ edges from G in the same time bound.

B. Sparsification

We've reduced our sampling cost by a factor of c, but still have an m factor in the runtime. We now invoke *sparsification* [2] to reduce m to nc, which reduces the overall runtime to $O((m/c)n^2 \log n) = O(n^3 \log n)$. We will argue that almost all edges of G are contained in *strong components* whose likelihood of disconnection is tiny.

Definition III.1. In any graph, we call an edge *k-strong* if it is contained in some k-connected subgraph.

Lemma III.2 (Weak Edges)**.** *Fewer than kn edges of a graph G are* not *k-strong.*

Proof: If every connected component of G is k-connected then every edge is k-strong. Conversely, if some edge is not k-strong then it is in a component of connectivity less than k. Find some cut in this component with less than k edges, and remove them. Repeat this process until all remaining edges are in k-connected components. Since each repetition increases the number of components, it can happen at most $n-1$ times before we partition the graph into n components that must be isolated vertices, which would mean there are no edges left to remove. Since each iteration removes less than k edges, the total number of edges removed is less than $k(n-1)$. ■

C. Application to Naive Monte Carlo

We can apply the strong components idea to speed up our naive Monte Carlo estimator algorithm. First, if $p^c > 1/n$, then the relative variance of a naive Monte Carlo sample is less than n, which means we only need $O(n)$ samples to get an unbiased estimator with constant relative variance. Thus, even if each sample takes m time the total time for estimating $u_G(p)$ is $O(mn) = O(n^3)$.

So now suppose that $p^c < 1/n$. In this case, $p^{5c} < p^c/n^4$. So consider the $5c$-strong components of G. According to Cycle Lemma II.1, considering each strong component as its own graph in isolation, the probability that one of them becomes disconnected is at most $n^2p^{5c} \leq p^c/n^2$. There are most n of them, so the probability that *any* of them becomes disconnected is at most p^c/n. Thus, the disconnection events on these strong components contribute a negligible amount to $u_G(p) \geq p^c$. It follows that conditioning on their *not* becoming disconnected causes a negligible change in $u_G(p)$.

Conditioning on these components being connected means that we can contract these strong components before we run our estimator algorithm. Weak Edges Lemma III.2 above says that after we contract these components only $m' \leq 5cn$ edges will remain. Thus we can apply the sampling method of the previous section to construct each sample of $O(m'(\log n)/c) = O(n \log n)$ edges from this contracted graph in $O(n \log n)$ time. This improves the time for our $O(n^2)$ trials $O(n^3 \log n)$.

There remains the detail of *finding* the $5c$-strong components. This can be done as is implied in the lemma, by repeatedly finding and removing cuts of G so long as they are less than $5c$. We can use any min-cut algorithm for this, for example the $\tilde{O}(m)$ time minimum cut algorithm of Karger [6]. Each cut we remove increases the number of components by 1, so we'll need at most $\tilde{O}(mn) = \tilde{O}(n^3)$ time to find the strong components. Alternatively, we can use Matula's simpler deterministic linear-time approximation algorithm [12] that finds a cut at most 3 times the minimum; if we delete any cut it finds of value less than $15c$ then we can be assured that once it terminates all components will be $5c$-strong while only $15nc$ edges cross between components.

Benczur and Karger [2] give an even faster solution: an $\tilde{O}(m)$-time *deterministic* algorithm that doesn't find the $5c$-strong components exactly, but instead finds a *refinement* of these components (into smaller pieces) such that at $O(nc)$ edges cross between components of the refinement. If we contract the components of this refinement, we are only contracting edges *inside* the $5c$-strong components, so our overall analysis applies unchanged. Unlike n minimum cut algorithms, this fast sparsification algorithm is dominated by the time it takes to run the estimator after; thus the overall runtime for estimating $u_G(p)$ remains $O(n^3 \log n)$.

D. Batch Sampling the Small Estimators

We've shown how to run naive Monte Carlo quickly; now we consider our unbiased estimator for $p^c < n^{-2}$. The first step of sampling $H \sim G(q)$ can proceed exactly as with the naive Monte Carlo algorithm: after contracting the strong components, we can sample the set of $O(n \log n)$ edges in $G(q)$ and construct their connected components. But our estimator has a second step: computing $u_H(p/q)$. This requires constructing the graph H, which naively would require looking through all m edges of G to find which ones cross between different components of H. This would increase our time per sample to $O(m)$ which is too large. In this section, we show how to get around that problem by *batching*, over our many samples, the work needed to find which edges cross between components of H.

In particular, we'll show that among the many samples H_i, the same vertices of G are contracted together and thus the same edges are eliminated by those contractions. Therefore, we can construct a batch of the tiny graphs H_i by first performing the contractions they share in common, then building each H_i quickly by starting from the resulting small "in common" graph.

So consider a set of k distinct samples H_i. We'll shift frequently between considering each H_i as a vertex partition that defines a cut of the vertices of G, and considering it as a contraction (of each component of its partition) that defines a tiny graph. Considering the H_i as partitions, define their *coarsest common refinement (CCR)* R to be the partition that equates two vertices of G if and only if they are in the same component of *every* H_i in the group. It follows that each H_i partition can be created by merging some of the components of R. Equivalently, taking the graph perspective, if we contract each component of R, we get a contracted version of G from which we can construct each H_i by performing additional contractions.

The CCR can be constructed quickly. Start by making R a partition with a single component of all the vertices of G. Then, for each partition H_i, go through each component of R and divide it into smaller components based on the partition H_i. This takes $O(n)$ time per partition. If two vertices are in different components of any H_i, they will be separated into distinct components of R when we process H_i; conversely, vertices that are always in the same component will never be separated. Once we have constructed the vertex partition, we can go through the edges of G in $O(m)$ time and convert them to edges of R by looking up their endpoints' components in R. Thus, the overall time to construct R is $O(m+kn)$. We can also merge parallel edges, using the capacitated representation discussed in Section II-E, to ensure that the number of edges is at worst quadratic in R.

As a first step, we show that for each graph H we generate, there are very few edges of G connecting the components.

Lemma III.3. *Suppose H is generated as $G(q)$ (we write $H \sim G(q)$). Then in expectation and with high probability there are $O(cq^{c/2}n)$ edges of G connecting different components of H.*

Proof: Consider constructing $G(q)$ in two steps just as we did for $G(p)$ in defining our unbiased estimator. First, contract edges of G with probability $1-s$ where $s = 2^{1/c}q$, producing a graph F. Then delete edges of F with probability $2^{-1/c}$. Just as we argued for $G(p)$ above, this procedure overall produces a graph sampled from $G(q)$.

Because $s^c = 2q^c$, according to Cycle Lemma II.1 the graph F will have $f = O(s^{c/2}n) = O(q^{c/2}n)$ components. We now construct H by deleting edges of F with probability $2^{-1/c}$ and contracting what remains. Equivalently, we contract each edge of F with probability $1 - 2^{-1/c} = \Theta(1/c)$.

To analyze the number of edges this leaves crossing H, we apply an argument of Karger, Klein, and Tarjan [8]. Go through the edges of F in arbitrary order to determine which get contracted to form H. For each, if its endpoints have not already been merged by previous contracts, flip the coin with probability $\Theta(1/c)$ to see if it gets contracted. If it is not, we assume that it is one of the edges crossing between components of H. But now observe that each time we find an edge that gets contracted, we decrease the number of components of H. Since initially F has f components, this can only happen f times before we discover that H is fully connected and there are no more crossing edges to find.

It follows that the number of edges we discover that cross between components of H is at most the number of ($1/c$-biased) coin flips that we perform before performing f contraction. This is a negative binomial distribution with expectation (and high probability bound) $O(cf) = O(cq^{c/2}n)$.

Note that as an alternative to applying the argument of [8], we can prove directly [9], [6] that no cut of value exceeding $O(c \cdot q^{c/2}n \cdot)$ fails in $G(q)$ with high probability. But given that we've already bounded the vertex count using Cycle Lemma II.1, this method is quicker. ■

Corollary III.4. *If k graphs H_i are sampled from $G(q)$, then their coarsest common refinement has $O(k \cdot q^{c/2}n)$ components.*

Proof: We've just proven that each H_i has $O(cq^{c/2}n)$ edges crossing the CCR. Thus, the k graphs H_i together have $O(kcq^{c/2}n)$ such edges. Now suppose the CCR has r components. Then since the CCR is a partition of G, it has at least c edges leaving each component, for a total of at least $rc/2$ edges. It follows that $rc/2 \leq O(kcq^{c/2}n)$. Now divide by c. ■

Observe that we are leveraging a very special feature of the underlying graphs H_i whose vertex partitions define our common refinement. In general, one can define a set of just $\log_2 n$ 2-way vertex partitions whose common refinement has all n vertices in distinct components—simply define the i^{th} partition based on the i^{th} bit of each vertex number. But we are fortunate to be working only with partitions that have few crossing edges and correspondingly few components in their coarsest common refinement.

E. A Recursive Batching Algorithm

We now apply our batching idea recursively. We've shown that the CCR of k samples H_i has at most $\beta k \log n$ components for some constant β. Suppose we are able to begin with a set of k vertex partitions of graphs H_i on a set of at most $2\beta k \log n$ vertices and wish to construct the graphs H_i. First, we compute the coarsest common refinement R of the k partitions, merge vertices in each component, and combine edges to give us a graph with at most $\beta k \log n$ vertices. We then divide the H_i arbitrarily into two groups and recursively solve the problem for each group starting with R. Note that each recursion involves $k/2$ partitions and a graph R with $\beta k \log n$ vertices, so it fits the assumption of our recursion. At the base of the recursion, we construct the coarsest common refinement of a single partition H_i, and the edges crossing that coarsest common refinement, which is precisely the graph H_i.

We can write a simple recurrence for the runtime of our algorithm in terms of k. As we argued above, constructing the coarsest common refinement R on a graph with N vertices and M edges takes time $O(kN)$ to construct and contract the coarsest common refinement's edge partition followed by $O(M)$ time to merge parallel edges in R. In our recursion $N = O(k \log n)$ while $M = O(k^2 \log^2 n)$; thus the second step dominates for an overall runtime of $O(k^2 \log^2 n)$. We then solve two subproblems on $k/2$ partitions recursively. Thus, our recurrence is $T(k) = O(k^2 \log^2 n) + 2T(k/2) = O(k^2 \log^2 n)$.

We need to construct n^2 graphs H_i. We could simply set $k = n$ in our recursive algorithm but this drastically overcounts the cost. Instead, divide the n^2 graphs into $n \log n$ groups of $k = n/\log n$ graphs H_i. Since we start with graph G having n vertices, each group fulfills the precondition on the recursive algorithm relating k to n. Thus, we can construct the H_i in each group in $O(k^2 \log^2 n) = O(n^2)$ time. Since there are $n \log n$ groups we can construct all n^2 of the required H_i in $O(n^3 \log n)$ time. Once we have done so, we proceed to estimate and average in each $u_H(p/q)$ in $o(n)$ time as previously discussed. This produces our final unbiased estimator in $O(n^3 \log n)$ time overall.

F. Sparse and Unbiased

The above scheme works, but by contracting strong components to reduce the effective m to nc it introduces a small change in the reliability of the graph, which means we have lost our unbiased estimator. We now show how to overcome this problem.

Our contraction of the strong components was driven by the observation that they will be connected with high probability in any sample we take, so that contracting them doesn't change things very much. To produce an unbiased estimator, instead of *assuming* that every strong component is connected in each sample H, we *check* whether every strong component is connected. If it isn't, we use the basic $O(m)$-time algorithm to construct that sample. Since this happens rarely, it won't affect the expected or high probability runtime of our estimator. All we need to do is perform our check quickly enough that the runtime is not affected.

So how can we quickly test whether the strong components stay connected? These strong components may have any number of edges, which means that our inclusion sampling trick may fail to keep the sample size small.

To address this problem, we invoke the idea of *sparse certificates* developed by Nagamochi and Ibaraki [14]. A *sparse t-certificate* of a graph can be constructed by repeating t times the process of finding and then deleting a spanning forest of the graph. Because each spanning forest will include an edge from every nonempty cut, the t-certificate will include all or at least t edges of each cut of the graph, which means it is t-connected if the original graph is. At the same time, the certificate will contain at most tn edges by construction. Nagamochi and Ibaraki give an MST-like algorithm that can construct a t-certificate in $O(m)$ time regardless of t.

We apply the sparse t-certificate idea to each of the $5c$-strong components of G, taking $t = 5c$. Consider some such component S with s vertices, and suppose we construct a sparse t-certificate T of the component. We have just asserted that T is t-connected with $t = 5c$. Thus, our previous analysis applies to T, showing that T will be connected with high probability in the sample. This gives us a fast way to test whether S is connected in our sample. First, use our inclusion-sampling trick trick on T; since it has $ts = O(cs)$ edges the sample will have $O(s \log n)$ edges as discussed previously. But also as discussed this (small) sample will be connected with high probability, in which case we can proceed without bothering to look at the other edges of S. If the high-speed sample does *not* connect T, which happens rarely, then we sample all the other edges of S the slow way in order to determine the sampling outcome for the component. This provides an algorithm that samples a strong component in time linear in its number of vertices with high probability and in expectation.

We apply this idea to sample the graph as a whole. After finding the k-strong components, we construct a graph K by contracting those components; K has $kn = O(cn)$ edges. With high probability, our samples are "consistent" with K; only cuts of K become disconnected by the sample. Thus, to build a sample, first we consider each k-strong component of G and use sampling from its sparse certificate to confirm that it is connected; since a component with r vertices has $O(rc)$ certificate edges from which $O(r \log n)$ are sampled with high probability, the total number of edges we sample in this step is $O(n \log n)$. All these components are connected with high probability. Assuming this happens, we sample from K to determine the connected components of the sample. If some component is *not* connected, then we sample from all the non-certificate edges of the component to complete our sample, but this happens too rarely to affect the expected or high probability runtime.

In summary, we have given a procedure that (after some preprocessing) constructs *unbiased* sampled graphs from $G(q)$ in amortized $O(n \log n)$ time per sample with high probability and in expectation. Using this procedure as our starting point provides to an unbiased estimator for $u_G(p)$ with runtime $O(n^3 \log n)$.

IV. Generating Disconnected Graphs

Our approach to producing an unbiased estimator of the disconnection probability can also be used to sample disconnected graphs—that is, to sample a $G(p)$ conditioned on the result being disconnected.

Since we have an unbiased estimator for the disconnection probability we could use the usual self-reducibility approach [4]. Essentially, we consider edges in sequence and include or remove each edge with an appropriate probability that can be determined using our estimator. However, this standard approach introduces a small error in the probability estimation: it is based on computing a ratio of two probabilities, for each of which we have an unbiased estimator, but where the ratio itself may not be unbiased. This means that we only get a (arbitrarily) close *approximation* to sampling from the disconnected distribution.

We use a different approach to sample *exactly*. We use a type of *rejection sampling*. To sample from a (hard) distribution $h(x)$ we instead sample from some (easy) distribution $e(x)$ but then *keep* our sample with a probability chosen to adjust for our starting from the wrong distribution. If we keep our sample that is our output; if we choose not to keep it then we repeat the algorithm until we do choose to keep something.

A. A Rejection-Sampling Algorithm

When $u_G(p)$ is large, we can sample a disconnected graph using the straightforward rejection-sampling analogue of Monte Carlo estimation. We simply generate graphs from $G(p)$ until we get one that is disconnected. Since each attempt succeeds with probability $u_G(p)$, the number of attempts we need to make is $1/u_G(p)$ in expectation.

For smaller $u_G(p)$, as with our estimator, we consider the problem of generating our graph from $G(p)$ in two steps. First, we generate a graph H from $G(q)$ for a large $q > p$ (contracting the connected components of H to yield a smaller graph). Then we generate a graph K from $H(p/q)$. We've already argued that this means K is sampled from $G(p)$. We now condition on the event U that K is disconnected; in other words, we wish to generate K with probability $\Pr[K \mid U]$. We instead consider generating both H *and* K, with probability $\Pr[H, K \mid U]$. We can write $\Pr[H, K \mid U] = \Pr[H \mid U] \cdot \Pr[K \mid H, U]$. In other words, we first generate the graph H from $G(q)$ but conditioned on K being disconnected. Then we generate K by sampling from $H(p/q)$ conditioned on U. The second problem is the same as our original one except that, as we argued for our estimator, the graph H is tiny.

The challenging task, then, is to sample H conditioned on U. For this, we adapt the rejection sampling idea we've already used. First, we sample H from $G(q)$. Then, we *keep* H with probability $M \cdot \Pr[U \mid H]$ for a constant M we'll choose later. Otherwise, we discard H and start over, repeating until we choose to keep a sample. This is a generalization of our previous rejection sampling approach; in particular if H is connected then we keep it with probability 0 (reject it) as before. But now even disconnected graphs are sometimes rejected to normalize their probabilities.

Under this scheme, the probability that we output H on the first try is $\Pr[H] \cdot M \cdot \Pr[U \mid H]$. Summing over H, the probability that we output *some* graph is then $M \cdot \Pr[U]$. Which means that *conditioned* on outputting something, the probability that we output H is

$$\frac{M \cdot \Pr[U \mid H] \Pr[H]}{M \cdot \Pr[U]} = \Pr[H \mid U]$$

(by Bayes' Law) as desired. We may not output anything, but the same argument applied inductively tells us that if we keep trying, then whenever we do output a graph it will have the desired distribution.

How long will this take? As argued above, the probability that we output anything on one attempt is $M \cdot \Pr[U]$, which means that the expected number of attempts we will need to make is $1/M \Pr[U]$. This suggests making M as large as possible to minimize the time.

What limits us is the role of M in keeping samples. We do so with probability $M \cdot \Pr[U \mid H]$, which must be a probability (at most 1) for the analysis to be correct: there is no way to keep a sample with probability 2. Thus, we are restricted to choosing M such that $M \cdot \Pr[U \mid H] \leq 1$. The graph which most strongly bounds M is thus the graph maximizing $\Pr[U \mid H]$. Since every H is a contraction of G, it follows that this least reliable contraction is G itself—in other words, we can set $M = 1/\Pr[U \mid H = G] = 1/u_G(p/q)$ (or anything smaller). Since $\Pr[U] = u_G(p)$, it follows that the number of trials needed to generate our sample will be $u_G(p/q)/u_G(p)$.

To bound this quantity, recall from Cycle Lemma II.1 that $u_G(p/q) \leq n^2(p/q)^c$. We can therefore pessimistically set M to one over this quantity. On the other hand $u_G(p) \geq p^c$. It follows that the expected number of samples is at most n^2/q^c.

There remains one detail: our algorithm is supposed to keep H with probability $M \cdot \Pr[U \mid H] = M \cdot u_H(p/q)$. How do we compute this probability? By using our unbiased estimator algorithm for $u_H(p/q)$. This estimator returns a *random* quantity Z whose *expectation* is $u_H(p/q)$, but that is sufficient. Let I be the indicator that we keep H; we keep H with probability $\Pr[I] = E[I] = E_Z[E[I \mid Z]] = E_Z[Z] = u_H(p/q)$ as required. It is this step that demands a true unbiased estimator for $u_H(p/q)$ instead of an approximation for it.

B. Runtime Analysis

In summary, we've given an algorithm that generates n^2/q^c samples of H in expectation to find one from the desired distribution. Each sample can be generated in $O(m)$ time. If we take $q^c = O((\log^2 n)/n)$ as before, then every H has $O(\log n)$ vertices high probability. This means that the time to compute the necessary unbiased estimator for $u_H(p/q)$, to decide whether to keep H, is $O(\log^3 n)$ which is negligible. Thus, the total work to generate and test samples is $O(mn^4/\log^2 n)$.

Once we've found H, we still need to generate K. This simply requires us to sample K from $H(p/q)$ conditioned on K being disconnected. This is equivalent to the problem we started with, but on H instead of G. However, as was argued in the estimator analysis, H will have $O(\log n)$ vertices with high probability. Conditioning on U will of course bias the size of H; however, our high probability bound asserts that *all* of the graphs we generate while seeking H will have $O(\log n)$ vertices. In the worst case, the graph H that we generate will be G and will have n vertices.

If H has 2 vertices then we can immediately return $K = H$ as this is the only disconnected graph that H can produce. If H is larger, we can run our generation procedure recursively, sampling an H' from $H(p/q')$ conditioned on the final output being disconnected, and so on. This gives us a probabilistic recurrence relation:

$$T(n) = O(mn^4/\log^2 n) + T(H)$$

where H has size $O(\log n)$ with high probability and $O(n)$ in the worst case. Taking expectations, we conclude

$$E[T(n)] = O(mn^4/\log^2 n) + E[T(O(\log n))] + \frac{1}{n}E[T(n)]$$

where the final term captures the unlikely possibility that we may need to re-solve a large graph. This solves to $E[T(n)] = O(mn^4/\log^2 n)$.

Note that our analysis made the pessimistic and somewhat contradictory assumptions that $u_G(p/q) \approx n^2(p/q)^c$ while $u_G(p) \approx p^c$; in many cases one may be able to show better, more consistent bounds. For example, as discussed earlier, when G is a cycle both quantities have an n^2 factor and when G is two cliques with c crossing edges neither does, so in both cases the number of samples needed is a factor of n^2 better than the pessimistic analysis.

C. A Faster Algorithm

Because all the intermediate graphs H are are small, computing $u_H(p/q)$ for rejection sampling takes little time. The bottleneck in our algorithm is generating the samples H. But for this we can reuse the idea of batch sampling that sped up our estimator. We expect to need to generate $n^4/\log^2 n$ sampled graphs H_i before rejection sampling decides to keep one. So, generate that many in a single batch; our batch sampling algorithm does this in amortized $O(n \log n)$ time per sample instead of $O(m)$. We expect to find and return a result within one batch; if we don't we simply generate and test other batches until we succeed. The expected number of batches we need is then constant so our expected runtime is the same as the one just claimed for a single batch. This improves our runtime to $O(n^5 \log n)$.

We can do even better by recognizing that generating our samples dominates the step of testing them for rejection. If instead of setting $q^c = (\log^2 n)/n^2$ we set $q^c = n^{-4/3}$, then Cycle Lemma II.1 tells us our sample will have size $O(n^{1/3})$ with high probability, which means that testing the sample for rejection will take $\tilde{O}(n)$ time using our (sparsified) unbiased estimator. But the rejection ratio M now improves to $n^2/q^c = n^{10/3}$. Since generating and testing each sample takes $\tilde{O}(n)$ amortized time, our overall time bound improves to $O(n^{13/3})$, approaching $\tilde{O}(n^4)$.

V. Two Conjectures and Two Algorithms

The relative variance is a useful parameter for characterizing estimation algorithms. We have given an estimator (which is unbiased for *any* $q > p$) and shown that its relative variance is $O((\log^2 n)/n^2)$ when we set $q^c = (\log^2 n)/n^2$. More generally we've shown a bound of q^{-c} when $q \leq (\log^2 n)/n^2$. But we believe that a stronger claim holds:

Conjecture V.1. When H is sampled as $G(q)$ the relative variance of $u_H(p/q)$ is bounded by $1/q^c$.

This bound is tight for the two vertex graph that has only a single cut of value c. Here we have shown this claim to be true so long as $q^c \leq (\log^2 n)/n^2$.

Assuming this conjecture, there is a quadratic-time estimator with relative variance $O(\log n)$ which is strongly reminiscent of the Recursive Contraction Algorithm [10]. As here our goal is to generate many samples $H \sim G(q)$ and average the resulting $u_H(p/q)$ to estimate G. But using an approach almost identical to the Recursive Contraction Algorithm, we can share a lot of the work involved in generating the samples, reducing the overall runtime to $\tilde{O}(n^2)$.

An $\tilde{O}(n^2)$-time algorithm is optimal for dense graphs, but we've seen previously that we can sparsify graphs to nearly $O(n)$ edges for reliability estimation so there is still room to improve. One option is to prove the following conjecture, which would immediately yield a near-linear-time algorithm:

Conjecture V.2. For any graph G, there is a q such that $G(q)$ has $O(\log n)$ vertices with high probability and such that taking $H \sim G(q)$ and returning $u_H(p/q)$ yields an estimator whose relative variance is constant.

We have seen that this conjecture holds for both the cycle ($q^c = 1/n^2$) and two cliques connected by a min-cut ($q^c = 1/2$) which are the two extremes when it comes to reliability and minimum cuts. The intuition behind our conjecture is that since small cuts are not able to overlap too much, they are "mostly" independent so that adding more of them only *reduces* the relative variance. In other words, the two-cliques graph should be the worst case one.

VI. Conclusion

We have given a particularly simple algorithm for estimating the reliability of a network by constructing an *unbiased estimator* with low variance. While all other algorithms for the problem relied on extensive analysis and complex algorithms and data structures, ours uses the most obvious possible approach of simply simulating edge failures and seeing how often network failures happen. The key nontrivial insight is that in order to measure the failure probability when it is extremely small (and thus cannot be detected by direct Monte Carlo simulation) we need to amplify the edge failure probability to cause failures, then adjust for that amplification in order to recover the network failure probability.

Our estimator approach quickly yields an algorithm with an $O(mn^2)$ runtime and a one-page analysis. If we sacrifice some simplicity and incorporate some standard ideas from graph sparsification, this runtime can be improved to $O(n^3 \log n)$, the best currently known.

Unlike previous algorithms, ours does not detour through any enumeration of minimum cuts. Instead it aims directly at sampling the quantity of interest. However, if you look inside the algorithm, it has a strong similarity to the Contraction Algorithm [10] and our original reliability algorithm [7]. Our estimator contracts edges at random to produce a graph of $O(\log n)$ vertices; the Contraction Algorithm contracts random edges to produce a graph of 2 vertices. Both are designed to preserve any min-cut with probability $\Omega(1/n^2)$, and both therefore perform the same number $O(n^2)$ of trials to ensure that every min-cut is found, which is in some sense critical for estimating reliability since min-cuts play the key role there.

At this point, however, the two algorithms diverge. The original reliability algorithm emphasized *approximately* minimum cuts in reliability. It worked harder (and slower) to enumerate all of them and determine their overall contribution. Our new estimator is more careless; it is reasonably likely to find near-minimum cuts but may miss some; since we only aim to be accurate *in expectation* this doesn't matter.

A good part of this paper was devoted to designing and analyzing an *unbiased estimator* for unreliability. If we'd only wanted a fully polynomial approximation scheme, we could have taken some short cuts in sparsification. But the unbiased estimator approach is appealing and elegant. In particular, the $\epsilon^{-2} \log 1/\delta$ dependence on ϵ and δ for an (ϵ, δ) approximation is clear and modular; there is no need to worry about either parameter during the relative variance analysis. Furthermore, the relative variance argument comes with no "high probability" caveat typical of an FPRAS. In particular, when we make use of such unbiased estimators, it is simple to analyze even algorithms (such as ours) that may recursively generate tiny subproblems for which "low probability" in the problem is is no longer particularly unlikely.

While our bounds are already better than any previously known, I believe they can be improved. In particular, that the relative variance of our estimator is actually far smaller than we showed here. For a particular n and c, the least reliable network is the cycle while the most reliable network is two cliques connected by a minimum cut. For both these networks, our estimator's relative variance can be improved to a constant by choosing a more suitable q, yielding an $\tilde{O}(m)$-time algorithm for estimating network unreliability. I've included 2 conjectures that point towards potential improvements.

Appendix

In this appendix, we restate the result [2] the the cycle is the least reliable network of minimum cut c in a particularly strong sense. The application bounding the probability of disconnection into k components that we need in this work can also be derived in two other ways [9], [6].

Lemma A.1. *Let G be any graph with minimum cut c and let Y by the cycle on n vertices where each pair of adjacent vertices is connected by $c/2$ edges. If edges are deleted independently with probability p, then the number of components in C stochastically dominates the number of components in G.*

Proof: We use a *coupling* argument [1], [4] Coupling is a powerful way to show that $\Pr[A \geq k] \geq \Pr[B \geq k]$ 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 facts (i) and (ii). It follows that $A \geq k$ whenever B is, which implies that $\Pr[A \geq k] \geq \Pr[B \geq k]$.

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 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, 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 $R_Y \geq R_G$ 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 the analysis.

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 non-loop edge of Y with a non-loop edge of G, and pair the remaining edges arbitrarily. It follows that if R_Y decreases because a non-loop 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. ■

References

[1] D Aldous. Random walks on finite groups and rapidly mixing markov chains. In *Siminaire de Probabilites XVII 1981/1982. Lecture Notes in Mathematics*, pages 243–297. Springer-Verlag, 1983.

[2] András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

[3] David G. Harris and Arvind Srinivasan. Improved bounds and algorithms for graph cuts and network reliability. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, ACM Press, January 2014.

[4] Mark R. Jerrum. A very simple algorithm for estimating the number of k-colourings of a low-degree graph. *Random Structures and Algorithms*, 7:157–165, 1995.

[5] David R. Karger. A randomized fully polynomial approximation scheme for the all terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492–514, 1999. A preliminary version appeared in Proceedings of the 27th ACM Symposium on Theory of Computing. A corrected version was published in SIAM Review 43(3).

[6] David R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, January 2000. A preliminary version appeared in Proceedings of the 28th ACM Symposium on Theory of Computing.

[7] David R. Karger. A randomized fully polynomial approximation scheme for the all terminal network reliability problem. *SIAM Review*, 43(3):499–522, 2001. A preliminary version appeared in Proceedings of the 27th ACM Symposium on Theory of Computing. This corrects a version published in SICOMP.

[8] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, March 1995.

[9] David R. Karger and Clifford Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In Alok Aggarwal, editor, *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 757–765. ACM, ACM Press, May 1993. Journal version appears in Journal of the ACM 43(4).

[10] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, July 1996. Preliminary portions appeared in SODA 1992 and STOC 1993.

[11] Richard M. Karp. Probabilistic recurrence relations. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 190–197. ACM, ACM Press, May 1991.

[12] D. W. Matula. A linear time $2+\epsilon$ approximation algorithm for edge connectivity. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 500–504. ACM-SIAM, January 1993.

[13] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, February 1992.

[14] Hiroshi Nagamochi and Toshihide Ibaraki. Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs. *Algorithmica*, 7:583–596, 1992.

[15] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a network remains connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[16] Leslie Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.