Hypergraph Unreliability in Quasi-Polynomial Time

Ruoxu Cen
Duke University
Durham, USA
ruoxu.cen@duke.edu

Jason Li
Carnegie Mellon University
Pittsburgh, USA
jmli@cs.cmu.edu

Debmalya Panigrahi
Duke University
Durham, USA
debmalya@cs.duke.edu

ABSTRACT

The hypergraph unreliability problem asks for the probability that a hypergraph gets disconnected when every hyperedge fails independently with a given probability. For graphs, the unreliability problem has been studied over many decades, and multiple fully polynomial-time approximation schemes are known starting with the work of Karger (STOC 1995). In contrast, prior to this work, no non-trivial result was known for hypergraphs (of arbitrary rank).

In this paper, we give quasi-polynomial time approximation schemes for the hypergraph unreliability problem. For any fixed $\varepsilon \in (0,1)$, we first give a $(1+\varepsilon)$-approximation algorithm that runs in $m^{O(\log n)}$ time on an m-hyperedge, n-vertex hypergraph. Then, we improve the running time to $m \cdot n^{O(\log^2 n)}$ with an additional exponentially small additive term in the approximation.

CCS CONCEPTS

• **Theory of computation** → **Graph algorithms analysis**; • **Networks** → **Network reliability**.

KEYWORDS

Network reliability, Hypergraphs

ACM Reference Format:
Ruoxu Cen, Jason Li, and Debmalya Panigrahi. 2024. Hypergraph Unreliability in Quasi-Polynomial Time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649753

1 INTRODUCTION

In the hypergraph unreliability problem, we are given an unweighted hypergraph $G = (V, E)$ and a failure probability $0 < p < 1$. The goal is to compute the probability that the hypergraph disconnects[1] when every hyperedge is independently deleted with probability p. The probability of disconnection is called the unreliability of the hypergraph G and is denoted $u_G(p)$. The hypergraph unreliability problem is a natural generalization of network unreliability which is identically defined but on graphs (i.e., hypergraphs of rank[2] 2). The latter is a classical problem in the graph algorithms literature that was shown to be #P-hard by Valiant [21] and its algorithmic study dates back to at least the 1980s [1, 19]. By now, several fully polynomial-time approximation schemes achieving a $(1+\varepsilon)$-approximation are known for the network unreliability problem [3, 13, 15–18]. In contrast, to the best of our knowledge, no non-trivial approximation was known for the unreliability problem on hypergraphs of arbitrary rank prior to this work.

Reliability problems are at the heart of analyzing the robustness of networks to random failures. (This can be contrasted with minimum cut problems that analyze the robustness to *worst-case* failures.) Since real world networks often exhibit random failures, there is much practical interest in reliability algorithms with entire books devoted to the topic [4, 8]. However, many basic questions remain unanswered from a theoretical perspective. One bright spot from a theoretical standpoint is the network unreliability problem, for which the first FPTAS was given by Karger in STOC 1995 [15]. Since then, many other FPTAS have been discovered with ever-improving running times [3, 13, 16–18], the current record being a recent $\widetilde{O}(m + n^{1.5})$-time algorithm (for a fixed ε) due to Cen *et al.* [3]. (Throughout the paper, m and n respectively denote the number of (hyper)edges and vertices in the (hyper)graph.) At the heart of these algorithms is the well-known fact that a graph has a polynomial number of near-minimum cuts – cuts whose value exceeds that of the minimum cut by at most a constant factor [14]. This polynomial bound extends to hypergraphs of rank at most $O(\log n)$ [20] and as a result, the FPTAS for network unreliability also apply to such hypergraphs. However, this approach fails for hypergraphs of arbitrary rank. In general, a hypergraph of rank r can have as many as $\Omega(m \cdot 2^r)$ near-minimum cuts (see Kogan and Krauthgamer [20] for an example), which rules out an enumeration of the near-minimum cuts in polynomial time for hypergraphs of large rank. This presents the main technical challenge in obtaining an approximation algorithm for hypergraph unreliability, and the main barrier that we overcome in this paper.

In addition to being a natural and well-studied generalization of graphs in the combinatorics literature, hypergraphs have also gained prominence in recent years as a modeling tool for real world networks. While graphs are traditionally used to model networks with point-to-point connections, more complex "higher-order" interactions in modern networks are better captured by hypergraphs as observed by many authors in different domains (see, e.g., the many examples in the recent survey of higher order networks by Bick *et al.* [2]). Indeed, the use of random hypergraphs as a modeling tool for real world phenomena has also been observed previously [12]. Therefore, we believe that the study of reliability in hypergraphs is a natural tool for understanding the connectivity

[1] A hypergraph is said to disconnect due to the failure of a subset of hyperedges when there is a bi-partition of the vertices such that every surviving hyperedge is entirely contained on either side of the bi-partition. Equivalently, the failed hyperedges must contain all hyperedges in some cut of the hypergraph.

[2] The *rank* of a hypergraph is the maximum rank of any hyperedge in it, where the rank of a hyperedge is the number of vertices in it.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649753

properties of such real world networks subject to random failures. We initiate this line of research in this paper and hope that this will be further expanded in the future.

1.1 Our Results

We give two algorithms for hypergraph unreliability. The first algorithm is simpler and achieves the following result:

Theorem 1.1. *For any fixed $\varepsilon \in (0,1)$, there is a randomized Monte Carlo algorithm for the hypergraph unreliability problem that runs in $m^{O(\log n)}$ time on an m-hyperedge, n-vertex hypergraph and returns an estimator X that satisfies $X \in (1 \pm \varepsilon)u_G(p)$ whp.*[3]

The running time of the algorithm in the theorem above (and also that in the next theorem) is inversely polynomial in the accuracy parameter ε. For brevity, we assume that ε is fixed throughout the paper and do not explicitly state this dependence in our running time bounds.

Note that the number of hyperedges in a hypergraph can be exponential in n. This makes a quasi-polynomial-time hypergraph algorithm that has a running time of $\text{poly}(m) \cdot n^{\text{poly}\log n}$ qualitatively superior to one that has a running time of $m^{\text{poly}\log(n)}$. (Contrast this to graphs where the two bounds are qualitatively equivalent because $m = O(n^2)$.) To this end, we give a second (more involved) algorithm that achieves this sharper bound on the running time incurring a small additive error in the approximation guarantee.

Theorem 1.2. *For any fixed $\varepsilon \in (0,1)$ and any $\delta \in (0,1)$, there is a randomized Monte Carlo algorithm for the hypergraph unreliability problem that runs in $m \cdot n^{O(\log n \cdot \log\log(1/\delta))}$ time on an m-hyperedge, n-vertex hypergraph and returns an estimator X that satisfies $X \in (1 \pm \varepsilon)u_G(p) \pm \delta$ whp.*

To interpret this result, set $\delta = \exp(-n)$. Then, we get an algorithm that runs in $m \cdot n^{O(\log^2 n)}$ time and returns an estimator X that satisfies $X \in (1+\varepsilon)u_G(p) \pm \exp(-n)$ whp. In other words, we obtain the sharper running time bound that we were hoping for in exchange for an exponentially small additive error in the approximation. We may also note that in general, a simple Monte Carlo simulation of the hypergraph disconnection event also gives an estimator for $u_G(p)$ with an additive error. But, this additive error would be exponentially larger than the one in Theorem 1.2; in particular, in order to ensure that $X \in (1 \pm \varepsilon)u_G(p) + \exp(-n)$ whp, we would need to run the Monte Carlo simulation $\exp(n)$ times, thereby giving an exponential time algorithm as against the quasi-polynomial running time in Theorem 1.2.

1.2 Our Techniques

We now give a description of the main technical ideas that are used in our algorithms. Let us start with a rough (polynomial) approximation to $u_G(p)$. In graphs, this is easy. Let λ denote the value of a minimum cut. Since there is at least 1 and at most $O(n^2)$ minimum cuts [10], their collective contribution to $u_G(p)$ is between p^λ and $O(n^2) \cdot p^\lambda$. Now, since the number of cuts of value $\leq \alpha\lambda$ is at most $n^{O(\alpha)}$ [14], the collective contribution of *all* other cuts to $u_G(p)$ is also at most $O(n^2) \cdot p^\lambda$ (for sufficiently small p, else we can just use Monte Carlo sampling). The bound of $O(n^2)$ on the number of minimum cuts continues to hold in hypergraphs (see [6, 11]; this is implicitly shown in [9]). So, their collective contribution is still between p^λ and $O(n^2) \cdot p^\lambda$. But, the number of cuts of value $\leq \alpha\lambda$ can be exponential in the rank r, and therefore exponential in n for $r = \Omega(n)$ [20]. Therefore, a naïve union bound over these cuts only gives a trivial exponential approximation to $u_G(p)$.

Our first technical contribution is to show that somewhat surprisingly, the upper bound of $O(n^2) \cdot p^\lambda$ on the value of $u_G(p)$ continues to hold for hypergraphs of arbitrary rank. As described above, we can't simply use a union bound over cuts, but must go deeper into the interactions between different cuts. To this end, we consider an alternative view[4] of the random failure of hyperedges. For each hyperedge, we generate an independent exponential variable (at unit rate) and superpose the corresponding Poisson processes on a single timeline. We contract each hyperedge as it appears on this timeline; then, the disconnection event corresponds to having ≥ 2 vertices in the contracted hypergraph at time $\ln(1/p)$. As hyperedges contract, the vertices (which we call *supervertices*) of the contracted hypergraph represent a partition of the vertices of the original hypergraph; we assign leaders to the subsets in this partition in a way that we can argue that any two vertices survive as leaders till time $\ln(1/p)$ with probability at most p^λ. This allows us to recover the $O(n^2) \cdot p^\lambda$ bound on the value of $u_G(p)$ by a union bound on all vertex pairs.

We now use this rough $O(n^2)$ approximation to $u_G(p)$ in designing a recursive algorithm. We generate a random hypergraph H by contracting hyperedges in G with probability $1-q$ for some $q > p$. (See [3, 16–18] for the use of random contraction in network unreliability.) The intuition is that by coupling, these edges will survive if the failure probability is p; hence, contracting them does not affect the disconnection event. The algorithm now makes a recursive call on H with the conditional failure probability p/q and obtains an estimator for $u_H(p/q)$. But, how do we bound the variance due to the randomness of H? This is where the $O(n^2)$-approximation comes in handy – it bounds the range of $u_H(p/q)$ to $O(n^2) \cdot (p/q)^\lambda$, thereby giving a bound of $n^2 \cdot q^{-\lambda}$ on the (relative) variance of the overall estimator.[5] Thus, if we select q such that $q^{-\lambda} = \text{poly}(n)$, then we only need a polynomial number of random trials.

For this plan to work, we need to we make progress in the recursion, i.e., make recursive calls on subgraphs that are smaller by a constant factor. Unfortunately, we are unable to ensure this in hypergraphs of arbitrarily large rank. To see this, consider a hypergraph containing n hyperedges of rank $n-1$, i.e., $\lambda = n-1$. In this case, we have $n^2 \cdot q^{-n+1}$ trials and the probability of each trial returning the input hypergraph is q^n (if none of the n hyperedges is contracted). So, ≥ 1 recursive calls (in expectation) will run on the input hypergraph itself, which defeats the recursion. However, we show that this is really an extreme scenario and we can make sufficient progress in all hypergraphs *with rank at most $n/2$* – we call these *universally small* and the rest *existentially large* hypergraphs.

[3] whp = with high probability. Throughout the paper, we say that a property holds with high probability if it fails with probability bounded by an inverse polynomial in n.

[4] See [18] for a different use of this alternative view.

[5] The relative variance of a random variable X is defined as $\text{Var}[X]/\mathbb{E}^2[X]$.

We are now left to handle existentially large hypergraphs. This is where the two algorithms (Theorem 1.1 and Theorem 1.2) differ. The first algorithm (Theorem 1.1) simply enumerates over all outcomes (survival/failure) of the large hyperedges, i.e., those of rank $> n/2$. To do this efficiently, it orders the large hyperedges and creates a new recursive instance based on the first large hyperedge that is contracted in this order. This generates $\ell \leq m$ subproblems, where ℓ denotes the number of large hyperedges. In the last subproblem, all the ℓ large hyperedges fail (i.e., none of them is contracted) and we are left with a universally small hypergraph. In all the other subproblems, at least one large hyperedge is contracted and we are left with a hypergraph containing at most $n/2$ vertices. So, we make progress in either case.

The second algorithm (Theorem 1.2) cannot afford to enumerate over all large hyperedges. Instead, it partitions the set of hyperedges in G into the large and small hyperedges and creates two hypergraphs, G_{large} and G_{small}. Now, for G to be disconnected, both G_{small} and G_{large} must be disconnected (but not vice-versa!). Recall that earlier, we ran into a problem where our naïve sampling process could not make progress in terms of reducing the size of the hypergraph when sampling large hyperedges. This was epitomized by a hypergraph containing n hyperedges of rank $n-1$ each. But, if we think of this instance in isolation, then it is actually quite easy to estimate $u_G(p)$ in this hypergraph. This is because whenever the hypergraph disconnects, it does so at a degree cut[6] of a vertex. So, there are only n cuts that we need to enumerate over. In fact, this property is true for the hypergraph G_{large} obtained from any hypergraph G; since every pair of large hyperedges share at least one vertex, any disconnected sub-hypergraph must have an isolated vertex. We exploit this property by writing a DNF formula for all the degree cuts of G_{large} (where each variable denotes survival/failure of a large hyperedge) and use the classical importance sampling technique of Karp, Luby, and Madras [19] to generate a sample of G_{large} conditioned on it being disconnected.

How do we augment this sample in G_{small}? We have two cases. To understand the distinction, let us informally imagine that the minimum cuts of G_{large} and G_{small} coincide, and they form the minimum cut of G. (Of course, this is not true in general!) The two cases are defined based on the relative values of the minimum cuts in G_{large} and G_{small}. If G_{large} contributes most of the hyperedges to the mincut (we call this the *full revelation* case), then the probability that G_{small} gets disconnected is quite high (recall that $u_G(p) \geq p^{\lambda}$). In this case, it suffices to do Monte Carlo sampling in G_{small} to augment the sample obtained from G_{large}. The other case is when G_{small} contributes a sizeable number of hyperedges to the minimum cut (we call this the *partial revelation* case). Note that the extreme example of this second case is when G_{large} is empty, i.e., when the hypergraph is universally small. This suggests generalizing the use of random contraction from universally small hypergraphs to this case, i.e., failing hyperedges at a higher probability of $q > p$ in a recursive step. But, to synchronize the sample across G_{large} and G_{small}, we must use the same value q in G_{large} as well. Unfortunately, as we observed earlier, the algorithm might not make progress in terms of the size of the hypergraph in this case. To overcome this, we introduce a second recursive parameter, that of the value of the failure probability itself. This second recursive parameter now requires us to define a new base case when the probability of failure is very small (denote the threshold by a parameter δ) – this is where we incur the additive loss of δ in the approximation. The overall running time is now given by the fact that each subproblem branches into polynomially many subproblems, and the depth of the recursion is bounded by $\log n \log\log(1/\delta)$ where the first term comes from the recursion on size and the second term from that on the failure probability.

[6] A degree cut is a cut that separates one vertex from the rest of hypergraph.

Organization. We give some preliminary definitions and terminology in Section 2. We then establish Theorem 1.1 in Section 3. Finally, we establish Theorem 1.2 in Section 4. We give some concluding thoughts in Section 5. We note that all missing proofs are deferred to the full version of the paper.

2 PRELIMINARIES

Hypergraphs. We start with some basic notations for hypergraphs. A hypergraph $G = (V, E)$ comprises a set of vertices V and set of hyperedges E, where each hyperedge $e \in E$ is a non-empty subset of the vertices, i.e., $\emptyset \subset e \subseteq V$. The *rank* of a hyperedge e is $|e|$; the rank of a hypergraph G, denoted r_G, is the maximum rank of a hyperedge in G.

For any hypergraph $G = (V, E)$ and subset of hyperedges $F \subseteq E$, denote $G - F := (V, E \setminus F)$ to be the hypergraph after deleting the hyperedges in F from G. A *cut* in a hypergraph is defined as a set of edges C such that $G - C$ is disconnected. The value of a cut C is the number of hyperedges in C. A *minimum cut* of a hypergraph is a cut of minimum value. We denote the value of a minimum cut in a hypergraph G by λ_G. The following is a known result (follows from Theorem 4 in [5] using the maximum flow algorithm in [7]):

Theorem 2.1. *The minimum cut of a hypergraph can be computed in* $(\sum_e |e|)^{1+o(1)}$ *time.*

In this paper, we often make use of hyperedge *contractions*. Contracting a hyperedge e in a hypergraph G replaces the vertices in e by a single vertex to form a new hypergraph denoted $G/e := (V/e, E/e)$. Note that there is a natural surjective map $\phi : V \to V/e$ that maps vertices in e to the contracted supervertex in V/e, and maps vertices outside e to themselves. Each hyperedge $e \in E$ is replaced in E/e by an element-wise mapped set $\{u \in e : \phi(u)\}$. By extension, contracting a *set* of hyperedges $F = \{e_1, e_2, \ldots\}$ is equivalent to contracting all hyperedges in F in arbitrary order: we write $G/F := (((G/e_1)/e_2)\ldots)/e_k$. H is called a contracted hypergraph of $G = (V, E)$ if $H = G/F$ for some $F \subseteq E$. For distinction between the uncontracted vertices in G and the contracted vertices in H, we usually call the former *vertices* and the latter *supervertices*.

A key operation in our algorithm is uniform random hyperedge contraction. We use $H \sim G(q)$ for some $q \in (0, 1)$ to denote the distribution of a random contracted hypergraph H obtained from G by contracting each hyperedge independently with probability $1 - q$. The next lemma states that $u_H(p/q)$ is an unbiased estimator of $u_G(p)$:

Lemma 2.2. *Suppose* $H \sim G(q)$ *and* $q \geq p$. *Then,* $\mathbb{E}[u_H(p/q)] = u_G(p)$.

Random Variables. Next, we give some basic facts about random variables that we will use in this paper. All random variables considered in the paper are non-negative.

The *relative variance* of a random variable X is

$$\eta[X] = \frac{\mathrm{Var}[X]}{(\mathbb{E}[X])^2} = \frac{\mathbb{E}[X^2]}{(\mathbb{E}[X])^2} - 1.$$

Since we use a biased estimator in Theorem 1.2, we need a non-standard (capped) version of relative variance. We define it and state its properties below.

Definition 2.3 (Capped relative variance)**.** The *(δ-)capped relative variance* of random variable X is

$$\eta_\delta[X] = \frac{\mathrm{Var}[X]}{\max\{(\mathbb{E}[X])^2, \delta^2\}}.$$

We state some basic facts about capped relative variance. Note that relative variance is a special case of capped relative variance when $\delta = 0$. Therefore, these facts also hold for relative variance as a special case.

Fact 2.4. *The average of M independent samples of X has capped relative variance $\frac{\eta_\delta[X]}{M}$.*

Fact 2.5. *Suppose Y is an unbiased estimator of x, and conditioned on a fixed Y, Z is a biased estimator of Y with bias in $[-\delta, 0]$ and capped relative variance $\eta_\delta[Z|Y] \le h$. Then*

$$\eta_\delta[Z] \le 4 \cdot (\eta[Y] + 1) \cdot (h + 1).$$

In particular, when $\delta = 0$ (i.e. for relative variance of unbiased estimator Z), there is a stronger bound

$$\eta[Z] \le (\eta[Y] + 1) \cdot (h + 1) - 1.$$

Fact 2.6. *Suppose X and Z are independent random variables with expectation in $(0, 1)$, and $\delta \in [0, 1]$. Then*

$$\eta_\delta[XZ] \le \eta_\delta[X] \cdot \eta_\delta[Z] + \eta_\delta[X] + \eta_\delta[Z].$$

Lemma 2.7. *The median-of-average of $\frac{\eta_\delta[X]}{\varepsilon^2}$ independent samples of X is a $(1 \pm \varepsilon, \delta)$-approximation of $\mathbb{E}[X]$.*

The next two facts are for relative variance:

Fact 2.8. *If X is a convex combination of independent non-negative random variables $X_1, \ldots, X_k$, i.e., $X = \sum_{i \le k} \alpha_i X_i$ for $\alpha_i \ge 0$ and $\sum_{i \le k} \alpha_i = 1$, then $\eta[X] \le \max_{i \le k} \eta[X_i]$.*

Fact 2.9. *If a non-negative random variable X is upper bounded by M, then $\eta[X] \le \frac{M}{\mathbb{E}[X]} - 1$.*

Exponential distribution. Recall that the exponential distribution of rate r gives a continuous random variable $X \ge 0$ satisfying $\Pr[X \ge t] = e^{-rt}$ for all $t \ge 0$. We state some standard properties of the exponential variables:

Fact 2.10 (Moment generating function). *Let X follow exponential distribution of rate r. Then for any $t < r$, $\mathbb{E}[e^{tX}] = 1/(1 - \frac{t}{r})$.*

Fact 2.11 (Memoryless property). *Let X follow exponentail distribution. Then for any $s, t \ge 0$, $\Pr[X > s + t | X > s] = \Pr[X > t]$.*

Fact 2.12. *Let $X_1, X_2, \ldots, X_k$ be independent random variables with exponential distribution of rate r, and $X = \min_{i \le k}\{X_i\}$. Then, X follows exponential distribution of rate kr. Moreover, $X = X_i$ for every value of i with probability $1/k$.*

Monte Carlo sampling. Suppose we want to estimate the probability p_D that an event D happens. (For $u_G(p)$, D is the event that the hypergraph disconnects.) The Monte Carlo sampling algorithm first draws a sample from the underlying space. (For $u_G(p)$, it deletes each hyperedge independently with probability p.) The estimator returns 1 if D happens, and 0 otherwise. The following is a standard property of this estimator:

Lemma 2.13. *Monte Carlo sampling outputs an unbiased estimator of p_D with relative variance at most $\frac{1}{p_D}$ and δ-capped relative variance at most $\min\{\frac{1}{p_D}, \frac{1}{\delta}\}$.*

Given Lemma 2.13, we can use Lemma 2.7 to obtain the following:

Lemma 2.14. *We can obtain a $(1 + \varepsilon)$-approximation of p_D whp via $O\left(\frac{\log n}{\varepsilon^2 \cdot p_D}\right)$ Monte Carlo samples and a $(1 + \varepsilon, \delta)$-approximation whp via $O\left(\frac{\log n}{\varepsilon^2 \cdot \delta}\right)$ Monte Carlo samples.*

DNF probability. In the *DNF probability* problem, we are given a DNF formula F with N variables and M clauses and a value $p \in (0, 1)$. The goal is to estimate the probability $u_F(p)$ that F is satisfied when each variable is True with probability p independently. This problem is #P-hard even in the special case of $p = \frac{1}{2}$ [21]. In a seminal work, Karp, Luby and Madras [19] provided an FPRAS in $\widetilde{O}(NM)$ time.

Theorem 2.15 ([19]). *The DNF probability problem can be $(1 \pm \varepsilon)$-approximated with success probability $1 - \delta$ in $O(NM \ln(1/\delta)/\varepsilon^2)$ time.*

Our algorithm will need an unbiased estimator for DNF probability. The estimator in Theorem 2.15 could be biased, but we can get an unbiased estimator by using its primitive version, at the cost of a slower running time. We state this in the next two lemmas; these are essentially shown in [19].

Lemma 2.16. *An unbiased estimator of $u_F(p)$ with relative variance at most 1 can be computed in time $O(NM^2)$.*

Lemma 2.17 (DNF sampling). *There exists an algorithm that draws a sample of values in time $O(NM^2)$ according to the following distribution: Each variable independently takes value True with probability p and False with probability $1 - p$, conditioned on the fact that the values satisfy F.*

3 THE ENUMERATION-BASED UNRELIABILITY ALGORITHM

In this section, we design an $m^{O(\log n)}$-time algorithm that outputs an unbiased estimator of $u_G(p)$ with relative variance $O(1)$. It follows by Lemma 2.7 that a $(1 \pm \epsilon)$-approximation can be computed in $m^{O(\log n)}\varepsilon^{-2}$ time, thereby establishing Theorem 1.1.

3.1 Algorithm Description

Overview. The algorithm is recursive. Before describing the algorithm formally, we give some intuition for the recursive step. The recursive case is divided into two sub-cases depending on the maximum rank of the hyperedges. We call a hypergraph *universally small* if all edge ranks are at most $n/2$; otherwise, it is said to be

existentially large. If the hypergraph is universally small, the algorithm runs a single recursive step of random hyperedge contraction, and recursively estimates the unreliability of the contracted hypergraph. This is repeated poly(n) times to reduce the variance of the estimator, and the average of all estimates is taken as output. If the hypergraph is existentially large, the algorithm lists all large hyperedges of rank greater than $n/2$, enumerates the first large hyperedge in the list that does not fail, and recursively estimates the unreliability of the resulting subgraph. The algorithm also handles the case that all large hyperedges fail by recursing on the (universally small) sub-hypergraph formed by deleting all large edges.

Now, we describe the algorithm formally.

Base cases. There are three base cases:

(1) G is disconnected. In this case, we output 1.
(2) p is larger than $n^{-10/\lambda}$. In this case, we use Monte Carlo sampling (Lemma 2.13) and take average of n^{10} samples.
(3) The number of vertices n is a constant. In this case, we merge all parallel hyperedges to form weighted hyperedges. We need to estimate $u_G(p)$ when each weighted hyperedge e is removed with probability $p^{w(e)}$, where $w(e)$ is the weight of e. We enumerate over all possible subsets of weighted hyperedges that are deleted, and compute $u_G(p)$ exactly. The first step takes $O(m)$ time; the rest is $O(1)$ time. We have established the following lemma:

Lemma 3.1. *When $n = O(1)$, $u_G(p)$ can be exactly computed in $O(m)$ time.*

Recursive case. We start by classifying hypergraphs as follows:

Definition 3.2 (universally small, existentially large hypergraphs)**.** A hypergraph is *universally small* if all hyperedges are of rank at most $n/2$. A hypergraph is *existentially large* if there exists a hyperedge of rank greater than $n/2$.

Recursive algorithm for universally small hypergraphs. The algorithm repeats a random contraction step independently $2n^{12}$ times. In the i-th random contraction step, the algorithm samples $H_i \sim G(q)$ by contracting each edge with probability $1-q$ independently, where $q = n^{-10/\lambda}$. Note that $q \geq p$, otherwise we are in a base case. Then, the algorithm recursively estimates $u_{H_i}(p/q)$. We will show later that $u_{H_i}(p/q)$ is an unbiased estimator of $u_G(p)$ with bounded relative variance. After all $2n^{12}$ recursive calls, the algorithm takes the average of the estimators returned by these recursive calls to be the output.

Recursive algorithm for existentially large hypergraphs. Suppose there are ℓ large hyperedges, ordered arbitrarily as $e_1, e_2, \ldots, e_\ell$. Let E_i be the subset of first i hyperedges in the list; in particular, $E_0 = \emptyset$. We divide the event of hypergraph disconnection into $\ell+1$ disjoint events by enumerating the first hyperedge in the list that does not fail. Formally, for $i = 0, 1, \ldots, \ell-1$, let A_i be the event that first i hyperedges in the list all fail, but the $(i+1)$-th hyperedge survives; Let A_ℓ be the event that all ℓ hyperedges fail. Then $\Pr[A_i] = p^i(1-p)$ for $i \leq \ell-1$ and $\Pr[A_\ell] = p^\ell$. Conditioned on each event A_i, we can remove the failed hyperedges in E_i and contract the first surviving hyperedge e_{i+1} to form a subgraph H_i. Formally, let $H_i = (G-E_i)/e_{i+1}$ for $i = 0, 1, \ldots, \ell-1$, and $H_\ell = G-E_\ell$.

The event that G disconnects conditioned on A_i is equivalent to H_i disconnecting when each hyperedge is removed with probability p independently. We have

$$\begin{aligned} u_G(p) &= \sum_{i=0}^{\ell} \Pr[A_i] \cdot \Pr[G \text{ disconnects} | A_i] \\ &= p^\ell \cdot u_{H_\ell}(p) + \sum_{i=0}^{\ell-1} p^i(1-p) \cdot u_{H_i}(p) \end{aligned} \tag{1}$$

The algorithm runs $\ell + 1 = O(m)$ recursive calls on each H_i to get unbiased estimators X_i of $u_{H_i}(p)$. The overall estimator X of $u_G(p)$ is then given by $X = p^\ell \cdot X_\ell + \sum_{i=0}^{\ell-1} p^i(1-p) \cdot X_i$. Equation (1) shows that X is an unbiased estimator of $u_G(p)$.

The subproblems are easier because of the following reason: in H_i for $i \leq \ell-1$, we contracted a large hyperedge from G, so the number of vertices decreases by at least a half; In H_ℓ, we removed all large hyperedges from G, so H_ℓ is universally small.

In the rest of the paper, we call this the *enumeration-based* algorithm.

3.2 Correctness

In this section, we prove the following lemma that establishes correctness of the algorithm.

Lemma 3.3. *The enumeration-based algorithm outputs an unbiased estimator with relative variance at most* 1.

Note that the base cases of disconnected G and constant size output exact value of $u_G(p)$, and the base case of Monte Carlo sampling outputs an unbiased estimator of $u_G(p)$. Also, an enumeration step in the existentially large case does not introduce variance. So, we only need to bound the relative variance introduced in the universally small case. To do so, we first analyze the variance introduced in a random contraction step.

The key to bounding relative variance of a random contraction step is the following property of a random subgraph which we will prove later.

Lemma 3.4. $p^\lambda \leq u_G(p) \leq n^2 p^\lambda$.

Lemma 3.4 provides an upper bound on the relative variance of random contraction:

Lemma 3.5. *Suppose $H \sim G(q)$ and $q \geq p$. Then, the relative variance of $u_H(p/q)$ is at most $n^2 q^{-\lambda} - 1$.*

Proof. Because H is constructed by contraction from G, its min-cut value λ_H is at least the min-cut value λ in G. By Lemma 3.4,

$$u_H(p/q) \leq |V(H)|^2 (p/q)^{\lambda_H} \leq n^2 (p/q)^\lambda \tag{2}$$

because $|V(H)| \leq n$, $\lambda_H \geq \lambda$, and $q \geq p$.

$u_H(p/q)$ is an unbiased estimator of $u_G(p)$ by Lemma 2.2. Next we bound its relative variance $\eta[u_H(p/q)]$. By Fact 2.9, the relative variance is upper bounded by $\frac{\max_H u_H(p/q)}{u_G(p)} - 1$. We have $\max_H u_H(p/q) \leq n^2(p/q)^\lambda$ by Equation (2), and $u_G(p) \geq p^\lambda$ by Lemma 3.4. Therefore,

$$\eta[u_H(p/q)] \leq \frac{n^2(p/q)^\lambda}{p^\lambda} - 1 = n^2 q^{-\lambda} - 1. \qquad \square$$

We are now prepared to prove Lemma 3.3.

Proof of Lemma 3.3. We will prove the lemma by induction on number of vertices n, number of hyperedges m, as well as the value of $\left\lceil m \ln \frac{1}{p} \right\rceil$. The base case of $n = O(1)$ is given by Lemma 3.1. The base case of $m = 0$ is handled by the disconnected case in the algorithm. These two base cases output the exact value of $u_G(p)$. For the induction on $\left\lceil m \ln \frac{1}{p} \right\rceil$, notice that this value is a positive integer because $p \in (0,1)$ in all recursive calls. The base case $\left\lceil m \ln \frac{1}{p} \right\rceil \le \frac{10m \ln n}{\lambda}$ implies $p^\lambda \ge n^{-10}$. Hence, it is handled by Monte Carlo sampling in the algorithm, which outputs an unbiased estimator of $u_G(p)$ with relative variance at most $\frac{1}{u_G(p)} \le \frac{1}{p^\lambda} \le n^{10}$ by Lemma 2.13. After taking average of n^{10} samples, the relative variance is reduced to at most 1 by Fact 2.4.

For the inductive step, there are two cases. We first consider a random contraction step when the hypergraph is universally small. This step generates $2n^{12}$ random subgraphs $H_i \sim G(q)$, where $q^\lambda = n^{-10}$. Lemma 2.2 gives that $u_{H_i}(p/q)$ is an unbiased estimator of $u_G(p)$. Lemma 3.5 gives that $u_{H_i}(p/q)$ has relative variance at most $n^2 q^{-\lambda} - 1 = n^{12} - 1$. By the inductive hypothesis, each subproblem returns an unbiased estimator X_i of $u_{H_i}(p/q)$ with relative variance at most 1. By Fact 2.5, X_i is an unbiased estimator of $u_G(p)$ with relative variance at most $n^{12}(1+1) - 1 \le 2n^{12}$. Taking average over all $2n^{12}$ estimators, X_i gives an unbiased estimator of $u_G(p)$ with relative variance at most 1 by Fact 2.4.

Next, we consider a large hyperedge enumeration step when the input hypergraph is existentially large. The algorithm computes an estimator $X = p^\ell \cdot X_\ell + \sum_{i=1}^{\ell} p^i(1-p) \cdot X_i$, where X_i's are returned by recursive calls on H_i. By the inductive hypothesis, the recursive calls give unbiased estimators, i.e. $\mathbb{E}[X_i] = u_{H_i}(p)$. We have

$$\begin{aligned}\mathbb{E}[X] &= p^\ell \cdot \mathbb{E}[X_\ell] + \sum_{i=1}^{\ell} p^i(1-p) \cdot \mathbb{E}[X_i] \\ &= p^\ell \cdot u_{H_\ell}(p) + \sum_{i=0}^{\ell-1} p^i(1-p) \cdot u_{H_i}(p) \overset{(1)}{=} u_G(p).\end{aligned}$$

X is a convex combination of independent recursive estimators X_i, which have relative variance at most 1 by the inductive hypothesis. Hence, X also has relative variance at most 1 by Fact 2.8.

Finally, we argue that the induction is valid, i.e., that we always make progress on one of the inductive parameters in every recursive call. Whenever a hyperedge is contracted or deleted, we decrease n or m. It is possible that a random contraction step does not change the hypergraph. In that case, notice that p is changed to p/q in the subproblem. So, $m \ln \frac{1}{p}$ decreases by $m \ln \frac{1}{q} = \frac{10m \ln n}{\lambda} \ge 10$. Therefore, we also decrease $\left\lceil m \ln \frac{1}{p} \right\rceil$, and the induction is valid. □

In the rest of this subsection, we give a proof of our main technical lemma, Lemma 3.4.

Proof of Lemma 3.4. The lower bound of p^λ in Lemma 3.4 holds because a minimum cut fails with probability p^λ. The rest of the proof is devoted to the upper bound of $n^2 p^\lambda$.

We first assume that in the hypergraph G, the hyperedges are partitioned into pairs of parallel hyperedges. This is w.l.o.g. because we can replace each hyperedge by two copies and change the failure probability p to $\sqrt{p}$.

We introduce some definitions that are used only in the analysis (i.e., the algorithm does not need to compute them). For any contracted hypergraph of G, we choose an orientation of the hyperedges in the sense that in each hyperedge, one vertex is designated the head and all other vertices are tails. We require the orientation to satisfy the property that any pair of parallel hyperedges (in the partition of hyperedges into pairs) have different heads. This is always possible because the rank of each hyperedge is at least 2. Besides this property, the choice of heads are arbitrary. The orientation is chosen in a consistent way. That is, any fixed contracted hypergraph always chooses the same orientation throughout the analysis.

The orientation is used to define representatives of contracted supervertices. Each contracted supervertex during the contraction process will be assigned a representative vertex, which is an original vertex contracted into the supervertex. Initially, each vertex is its own representative. Whenever a hyperedge e is contracted, we assign the representative of the head of e to be the representative of the new contracted supervertex.

Claim 3.6. *For any pair of supervertices $u \ne v$ in a contracted hypergraph of G, there are at least λ hyperedges that contain at least one of u or v as a tail.*

We now proceed to prove the upper bound.

Exponential contraction process. For the sake of analysis, consider the following continuous time random process called the exponential contraction process. Let each hyperedge e independently arrive at a time Y_e following the exponential distribution of rate 1. Then, the probability that a hyperedge does not arrive before time $\ln \frac{1}{q}$ is $e^{-\ln \frac{1}{q}} = q$. Therefore, contracting hyperedges that arrive before time $\ln \frac{1}{q}$ produces the same distribution as $H \sim G(q)$.

In the contraction process, if the hypergraph is not contracted into a single supervertex at time $\ln \frac{1}{p}$ (which happens with probability $u_G(p)$), there are at least two supervertices. Consequently, at least two vertices survive as representatives. We will show that the probability that any pair of vertices s, t both survive is at most p^λ. By union bounding over all $\binom{n}{2} \le n^2$ pairs of vertices, we have $u_G(p) \le n^2 p^\lambda$ which completes the proof.

To bound the probability that any pair of vertices s, t both survive, we choose a set of *critical edges* during the contraction process as follows. When s and t are in different supervertices $\widetilde{s}$ and $\widetilde{t}$, we choose λ hyperedges that contain at least one of $\widetilde{s}$ or $\widetilde{t}$ as a tail guaranteed by Claim 3.6; otherwise, we choose λ arbitrary edges. (The critical edges may change after each contraction.) Note that whenever a critical edge arrives, one of s or t is no longer a representative. Hence, if s, t both survive as representatives after the contraction process, then no critical edge arrives during the contraction process. By Lemma 3.7, this happens with probability at most $e^{-\lambda \ln \frac{1}{p}} = p^\lambda$.

Lemma 3.7. *Suppose during the exponential contraction process from time 0 to T, we maintain a subset of uncontracted hyperedges called the* critical edges. *The critical edges could change immediately after each arrival of an uncontracted hyperedge, but do not change*

between two arrivals. Suppose that there are always at least λ critical edges. Then, the probability that no critical edge arrives up to time T is at most $e^{-\lambda T}$.

Proof. The proof is by induction on the number of uncontracted hyperedges m. The base case is $m = \lambda$, where the set of critical edges cannot change, and the probability that no critical edge arrives is $e^{-\lambda T}$.

For the inductive case, let $m_{\text{crit}} \geq \lambda$ be the current number of critical edges. We bound the probability that no critical edge arrives up to time T (denoted $p_G(T)$) by integrating over the earliest time t that a hyperedge e_i arrives. By Fact 2.12, t follows the exponential distribution of rate m, and e_i is not a critical edge with probability $1-\frac{m_{\text{crit}}}{m}$. After the earliest hyperedge e_i arrives, we apply the inductive hypothesis on hypergraph G/e_i and remaining time $T-t$.

$$\begin{aligned} p_G(T) &= \int_0^T me^{-mt}\,\mathrm{d}t \cdot \left(1-\frac{m_{\text{crit}}}{m}\right) p_{G/e_i}(T-t) + \int_T^\infty me^{-mt}\,\mathrm{d}t \\ &\leq e^{-mT} + \int_0^T me^{-mt}\,\mathrm{d}t \cdot \left(1-\frac{\lambda}{m}\right)\cdot e^{-\lambda(T-t)} \\ &= e^{-mT} + e^{-\lambda T}\int_0^T (m-\lambda)e^{-(m-\lambda)t}\,\mathrm{d}t \\ &= e^{-mT} + e^{-\lambda T}(1-e^{-(m-\lambda)T}) = e^{-\lambda T}, \end{aligned}$$

which completes the inductive case. □

3.3 Running Time

In this section, we prove the following lemma of running time of the algorithm.

Lemma 3.8. *The expected running time of the enumeration-based algorithm is $m^{O(\log n)}$.*

Size decrease bound. In order to bound the size of the recursion tree, we ideally want each random contraction step to reduce the number of supervertices by a constant factor, so that the recursion tree has depth $O(\log n)$. This does not generally hold with high probability when some hyperedge's rank is large. As an extreme example, consider the hypergraph that consists of n distinct hyperedges of rank $n-1$. The probability to contract nothing is q^n, while the contraction step is repeated $n^2q^{-\lambda} = n^2q^{-(n-1)}$ times, so we expect to see at least one bad subproblem where the number of vertices does not decrease. If this happens, then the size of the recursion tree would be unbounded.

This is where the maximum rank assumption in the universally small case becomes useful: we can control the probability to get a bad subproblem to be small enough, which is crucial in bounding the size of the recursion tree.

Lemma 3.9. *Fix constants $A > B > 1$. Suppose that all hyperedges have rank at most $R = n/A$, and n is larger than some constant depending on A. Let $n^* = \lceil BR \rceil$, and let $H \sim G(q)$. Then, $\Pr[|V(H)| \geq n^*+1] < n^{A^2/B}q^{B\lambda}$.*

Proof. Define $t(G)$ to be the stopping time in the contraction process when the vertex size of the contracted hypergraph decreases to at most n^*. By definition

$$\Pr[|V(H)| > n^*] = \Pr\left[t(G) > \ln\frac{1}{q}\right].$$

By Markov's inequality,

$$\Pr\left[t(G) > \ln\frac{1}{q}\right] = \Pr[e^{B\lambda\cdot t(G)} > q^{-B\lambda}] \leq \frac{\mathbb{E}[e^{B\lambda\cdot t(G)}]}{q^{-B\lambda}}.$$

By choosing the same parameters A, B in Lemma 3.11 (which we will prove later),

$$\begin{aligned} \Pr\left[t(G) > \ln\frac{1}{q}\right] &\leq \mathbb{E}[e^{B\lambda\cdot t(G)}]q^{B\lambda} \leq \left(\frac{en}{n^*}\right)^{A\ln n} q^{B\lambda} \\ &\leq (Ae/B)^{A\ln n}q^{B\lambda} \leq n^{A(1+\ln\frac{A}{B})}q^{B\lambda} \leq n^{A^2/B}q^{B\lambda}, \end{aligned}$$

where the last inequality uses that $1+\ln x \leq x$ for all $x \geq 0$. □

In Lemma 3.9, we set $A = 2$ and $B = 1.5$ to get:

Corollary 3.10. *Assume that all hyperedges have rank at most $n/2$ and n is larger than a suitable constant. Let $H \sim G(q)$. Then, $\Pr[|V(H)| \geq 0.8n] \leq n^{2.7}q^{1.5\lambda}$.*

Lemma 3.11. *Fix constants $A > B > 1$. Suppose the maximum rank is upper bounded by R and R is larger than some constant depending on A. For any hypergraph $\widetilde{G}$ formed by contraction from G, define $t(\widetilde{G})$ to be the following stopping time: In a contraction process starting at $\widetilde{G}$, the vertex size of the contracted hypergraph decreases to at most $n^* = \lceil BR \rceil$. Suppose $|V(\widetilde{G})| \leq N$, where $N = AR$. Let $\widetilde{\lambda}$ be the min-cut value in $\widetilde{G}$. Then,*

$$\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G})}\right] \leq \max\left\{\left(\frac{|V(\widetilde{G})|}{n^*/e}\right)^{A\ln N}, 1\right\}.$$

Proof. Our proof is by induction on $\widetilde{n} = |V(\widetilde{G})|$. As the base case, when $\widetilde{n} \leq n^*$, by definition $t(\widetilde{G}) = 0$ and $e^{B\widetilde{\lambda}\cdot t(\widetilde{G})} = 1$, so the statement holds.

Next consider the inductive step where $\widetilde{n} > n^*$. Let $\bar{r}$ be the average rank in $\widetilde{G}$, i.e., $\bar{r} = \frac{1}{\widetilde{m}}\sum_{e\in E(\widetilde{G})} r(e)$, where $\widetilde{m} = |E(\widetilde{G})|$.

Let $\tau = \min_{e\in E(\widetilde{G})} X_e$ be the earliest arrival time of a hyperedge in $\widetilde{G}$. Because the X_e's are sampled from i.i.d. exponential distributions of rate 1, the random variable τ follows the exponential distribution of rate $\widetilde{m}$ by Fact 2.12. Note that any degree cut in $\widetilde{G}$ has value at least the min-cut value $\widetilde{\lambda}$ in $\widetilde{G}$. By summing over degrees of all vertices, we have

$$\widetilde{n}\widetilde{\lambda} \leq \sum_{e\in E(\widetilde{G})} r(e) = \widetilde{m}\bar{r} \implies \widetilde{m} \geq \frac{\widetilde{n}\widetilde{\lambda}}{\bar{r}}.$$

Denote rate$(\cdot)$ to be the rate of a exponential distributed random variable. Because $\widetilde{n} > n^*$ and $\bar{r} \leq R$, we have $\text{rate}(\tau) = \widetilde{m} \geq \frac{\widetilde{n}\widetilde{\lambda}}{\bar{r}} > \frac{n^*\widetilde{\lambda}}{R} \geq B\widetilde{\lambda}$.

By the moment generating function of the exponential distribution in Fact 2.10,

$$\mathbb{E}\left[e^{B\widetilde{\lambda}\tau}\right] = \frac{1}{1-\frac{B\widetilde{\lambda}}{\text{rate}(\tau)}} \leq \frac{1}{1-\frac{B\widetilde{\lambda}}{\widetilde{n}\widetilde{\lambda}/\bar{r}}} = \frac{1}{1-\frac{B\bar{r}}{\widetilde{n}}} \tag{3}$$

Now, consider the distribution of $t(\widetilde{G})$ after revealing τ. To decrease size from $\widetilde{n} > n^*$ to below the threshold n^*, at least one hyperedge needs to be contracted; so $t(\widetilde{G}) \geq \tau$. Moreover, by the memoryless property of the exponential distribution in Fact 2.11, τ

and $t(\widetilde{G}) - \tau$ are independent. If we further reveal the information that the hyperedge that arrives the earliest is e_i, then we know the random process after the earliest arrival is equivalent to running the contraction process starting from $\widetilde{G}/e_i$. Thus, $t(\widetilde{G}) - \tau$ follows the same distribution as $t(\widetilde{G}/e_i)$ conditioned on e_i arrives first. Because the hyperedges follow i.i.d. exponential distribution, they are equally likely to arrive the earliest. Therefore, unconditionally we have that $t(\widetilde{G}) - \tau$ follows distribution of $t(\widetilde{G}/e_i)$ for each e_i with probability $\frac{1}{\widetilde{m}}$. Formally,

$$\begin{aligned}\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G})}\right] &= \mathbb{E}\left[e^{B\widetilde{\lambda}\tau}\right]\cdot\mathbb{E}\left[e^{B\widetilde{\lambda}(t(\widetilde{G})-\tau)}\right]\\ &= \mathbb{E}\left[e^{B\widetilde{\lambda}\tau}\right]\sum_{e_i\in E(\widetilde{G})}\frac{1}{\widetilde{m}}\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G}/e_i)}\right] \quad (4)\end{aligned}$$

Note that the min-cut value λ_i in $\widetilde{G}/e_i$ is at least $\widetilde{\lambda}$ because a contraction cannot decrease the min-cut value. So,

$$\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G}/e_i)}\right] \le \mathbb{E}\left[e^{B\lambda_i\cdot t(\widetilde{G}/e_i)}\right].$$

We now apply the inductive hypothesis to bound this term.

$$\begin{aligned}\mathbb{E}\left[e^{B\lambda_i\cdot t(\widetilde{G}/e_i)}\right] &\le \max\left\{\left(\frac{\widetilde{n}-r(e_i)+1}{n^*/e}\right)^{A\ln N},1\right\}\\ &= \frac{(\max\{\widetilde{n}-r(e_i)+1,n^*/e\})^{A\ln N}}{(n^*/e)^{A\ln N}}\\ &\le \frac{(\max\{\widetilde{n}-r(e_i)+1,\widetilde{n}/e\})^{A\ln N}}{\widetilde{n}^{A\ln N}}\cdot\left(\frac{e\widetilde{n}}{n^*}\right)^{A\ln N}\\ &= \max\left\{\left(1-\frac{r(e_i)-1}{\widetilde{n}}\right)^{A\ln N},N^{-A}\right\}\cdot\left(\frac{e\widetilde{n}}{n^*}\right)^{A\ln N}\end{aligned}$$

Next, we prove that $\max\{(1-\frac{r-1}{\widetilde{n}})^{A\ln N}, N^{-A}\} \le 1-\frac{Br}{\widetilde{n}}$ for all $r\in[2,R]$. For the second term, we have $N^{-A} < \frac{1}{N}$, while

$$1-\frac{Br}{\widetilde{n}} \ge 1-\frac{BR}{\widetilde{n}} \ge 1-\frac{n^*}{n^*+1} = \frac{1}{n^*+1} \ge \frac{1}{N}.$$

For the first term, define $x = \frac{r(e_i)-1}{\widetilde{n}}$ in terms of $r(e_i)$, which satisfies $x\in[\frac{1}{\widetilde{n}},\frac{R-1}{\widetilde{n}}]$. Then, $r(e_i) = \widetilde{n}x+1$. We want to bound $(1-\frac{r-1}{\widetilde{n}})^{A\ln N} \le 1-\frac{Br}{\widetilde{n}}$ for all $r\in[2,R]$, which is equivalent to

$$(1-x)^{A\ln N} \le 1-Bx-\frac{B}{\widetilde{n}}. \quad (5)$$

Note that the LHS is convex on $(0,1)$ when $A\ln N \ge 2$, and the RHS is linear. So we only need to prove (5) for the two endpoints $x=\frac{1}{\widetilde{n}}$ and $x=\frac{R-1}{\widetilde{n}}$. For $x=\frac{1}{\widetilde{n}}$, we have

$$\left(1-\frac{1}{\widetilde{n}}\right)^{A\ln N} \le e^{-A\ln N/\widetilde{n}} \le 1-\frac{A\ln N}{2\widetilde{n}} \le 1-\frac{2B}{\widetilde{n}}$$

Here, the second inequality is by $e^{-x}\le 1-\frac{x}{2}$ for $x\in[0,1]$ or $A\ln N/\widetilde{n}\le 1$, which holds when N is larger than some constant (depending on A). The last inequality holds when $\ln N\ge 4$.

Next, we consider the second endpoint $x=\frac{R-1}{\widetilde{n}}$. The assumption $N=AR\ge\widetilde{n}$ implies $\frac{R}{\widetilde{n}}\ge\frac{1}{A}$. So, we have

$$\left(1-\frac{R-1}{\widetilde{n}}\right)^{A\ln N} \le \left(1-\frac{1}{A}+\frac{1}{\widetilde{n}}\right)^{A\ln N} \le \frac{1}{N}$$

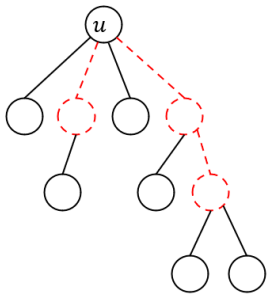

Figure 1: A depiction of a portion of the computation tree. The failed recursive calls are shown in dashed red, while the successful ones are shown in solid black. Lemma 3.12 analyzes the expected size of the recursion tree.

Here, the last inequality is equivalent to $1-\frac{1}{A}+\frac{1}{\widetilde{n}}\le e^{-1/A}$, which holds when $\widetilde{n}>3A^2$ and $A>1$.

Now, we have

$$\begin{aligned}\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G})}\right] &\overset{(4)}{=} \mathbb{E}\left[e^{B\widetilde{\lambda}\tau}\right]\cdot\frac{1}{\widetilde{m}}\sum_{e_i\in E(\widetilde{G})}\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G}/e)}\right]\\ &\overset{(3)}{\le} \frac{1}{1-\frac{B\widetilde{r}}{\widetilde{n}}}\cdot\frac{1}{\widetilde{m}}\sum_{e_i\in E(\widetilde{G})}\mathbb{E}\left[e^{B\widetilde{\lambda}\cdot t(\widetilde{G}/e)}\right]\\ &\le \frac{1}{1-\frac{B\widetilde{r}}{\widetilde{n}}}\cdot\frac{1}{\widetilde{m}}\sum_{e_i}\left(1-\frac{Br(e_i)}{\widetilde{n}}\right)\cdot\left(\frac{e\widetilde{n}}{n^*}\right)^{A\ln N} \le \left(\frac{e\widetilde{n}}{n^*}\right)^{A\ln N}\end{aligned}$$

which establishes the inductive case. □

Proof of Lemma 3.8. We color each recursive call as black or red. Intuitively, they represent a "success" or "failure" in recursive calls respectively. A call is black if it decreases size by a constant factor, more precisely when $|V(H)|\le 0.8n$. Otherwise, the call is red.

The recursion tree has the following properties:

(1) Each subproblem makes at most $M=m^{O(1)}$ recursive calls. This is guaranteed by the algorithm.
(2) The algorithm reaches base case after $O(\log n)$ black recursive calls. This is because each black recursive call decreases n by a constant factor, and $O(\log n)$ black calls reduces n to a constant, which is a base case.
(3) For each subproblem, the expected number of red recursive calls is at most $1/2$. This is because in random contraction, we have $O(n^2q^{-\lambda})$ recursive calls, and each call fails (to get size decrease) with probability at most $n^{2.7}q^{1.5\lambda}$ by Corollary 3.10. The expected number of red calls is their product, which is upper bounded by $n^{4.7}q^{0.5\lambda}=o(1)$ when $q^\lambda=n^{-10}$.

Lemma 3.12 below shows that these properties give a upper bound of $m^{O(\log n)}$ on the number of recursive calls. If we charge the time of sampling a random contracted hypergraph to the subproblem on the contracted hypergraph, then each subproblem spends $O(nm)$ time outside the recursive calls. Therefore, the overall expected running time is $m^{O(\log n)}$. This concludes the proof of Lemma 3.8.

Lemma 3.12. *Suppose in a randomly growing tree, each node u is either a leaf, or has $M(u) \le M$ children, where $M = \Omega(n)$ is a parameter. Each edge from u to its children is colored red with probability $f(u)$ such that $M(u)f(u) \le \theta = \frac{1}{2}$, and black otherwise. The different children at a parent node are independent (including independence betwen the parent-child edges); Also, a subtree is independent of everything outside the subtree. Moreover, on any path from root to leaf, there can be at most L black edges. Then, the expected number of nodes in the tree is at most $M^{O(L)}$.*

Proof. We say a node w is a red descendant of node u if u is an ancestor of w, and the path from u to w is formed by red edges only. See Figure 1 for an illustration.

Let K be the number of red descendants of some node u. Let K_i be the number of red descendants of u that are i steps deeper than u, so that $K = \sum_{i\ge 1} K_i$. We have $\mathbb{E}[K_1] = M(u)f(u) \le \theta$. Inductively, suppose at level i, there are K_i red descendants $\{u_1, \dots, u_{k_i}\}$. By definition, the red descendants in level $i+1$ must be children of red descendants in level i. Each u_j will generate at most θ red edges in expectation. So, $\mathbb{E}[K_{i+1}] \le \theta \cdot \mathbb{E}[K_i]$. By induction, $\mathbb{E}[K_i] \le \theta^i$. Note that the sum $\sum_{i\ge 1} \theta^i = \frac{\theta}{1-\theta} \le 1$ converges and K_i's are nonnegative, so we can apply Fubini's theorem to get

$$\mathbb{E}[K] = \mathbb{E}\left[\sum_{i\ge 1} K_i\right] = \sum_{i\ge 1} \mathbb{E}[K_i] \le \sum_{i\ge 1} \theta^i \le 1.$$

Let V_i be the set of nodes that have k black edges from the root. We prove by induction that $\mathbb{E}[|V_k|] \le 2(2M)^k$. It follows that the expected number of nodes is $\mathbb{E}[\sum_{k=0}^{L} |V_k|] = O((2M)^L) = M^{O(L)}$. The base case for $k = 0$ is the expected number of red descendants of the root, as well as the root itself, which is at most $1 + 1 = 2 = 2(2M)^0$. Next consider the inductive step. For any node $w \in V_{k+1}$, let (u, v) be the black edge closest to w on the path from root to w. Then, $u \in V_k$, and w is either a red descendent of v or v itself. Each node u in V_k generates $M(u)$ children v, and each v generates at most 1 red descendants in expectation. Therefore, $\mathbb{E}[|V_{k+1}|] \le \mathbb{E}[|V_k|] \cdot M \cdot (1 + 1) \le 2(2M^{k+1})$. □

4 THE SAMPLING-BASED UNRELIABILITY ALGORITHM

In this section, we strengthen the previous algorithm to obtain a running time of $m \cdot n^{O(\log n \cdot \log\log \frac{1}{\delta})}$, at the cost of an additive error of δ, i.e. the output estimator is within $(1 \pm \varepsilon)u_G(p) \pm \delta$ whp (Theorem 1.2). When $\delta = 2^{-\mathrm{poly}(n)}$, the running time of the algorithm is $m \cdot n^{O(\log^2 n)}$. We show that the algorithm outputs an estimator of $u_G(p)$ with bias at most δ and δ-capped relative variance $O(1)$. Theorem 1.2 then follows by Lemma 2.7.

We denote $N = \log_2 \frac{1}{\delta}$. We can assume wlog $N \ge \log_2 n$, i.e. $\delta \le \frac{1}{n}$. This is because we can run a simple Monte Carlo simulation when $\delta \ge \frac{1}{n}$ in $O\left(\frac{\log n}{\delta\varepsilon^2} \cdot nm\right) = \widetilde{O}(n^2 m\varepsilon^{-2})$ time by Lemma 2.14.

4.1 Algorithm Description

The algorithm is recursive. We start by defining the simple base cases. Then, we introduce the definition of large hyperedges, which characterize the last base case. Finally, we define the recursive cases.

4.1.1 Base Cases. There are four base cases, three of which are the following:

(1) When the number of vertices n is a constant, we enumerate all possible outcomes by brute force, which is identical to Lemma 3.1.
(2) When the hypergraph is already disconnected, output 1.
(3) When $p^\lambda < 2^{-3N}$, output 0.

These three base cases are deterministic. The algorithms for the first two base cases return the exact value of $u_G(p)$. The third case has an additive bias of $u_G(p)$, which is at most $n^2 p^\lambda$ by Lemma 3.4. We assumed $N \ge \log_2 n$ and $p^\lambda < 2^{-3N}$, so $n^2 p^\lambda \le (2^N)^2 \cdot 2^{-3N} \le 2^{-N} \le \delta$.

The fourth base case is called full revelation; it will be described later in the section.

4.1.2 Large and Small Hyperedges. Before proceeding with the formal description of the remaining algorithm, let us provide some intuition. Recall from Section 3 that the large hyperedges are the bottleneck in the random contraction algorithm: we branch m times to halve the number of vertices, which leads to the $m^{O(\log n)}$ running time. However, if we ignore the small hyperedges and consider the hypergraph with large hyperedges only, it turns out that the structure of cuts becomes much simpler. This motivates us to partition the set of hyperedges E into two sets, E_{large} and E_{small}. Intuitively, these are sets of hyperedges of large and small rank respectively, but for technical reasons, the precise definition needs to be more nuanced.

We now formally define the set E_{large}. It depends on *phase nodes* in the recursive computation tree, which we define first. Initially, the root of the computation tree is a phase node. For any non-root node w of the computation tree, let u be the closest ancestor node of w that is a phase node (which exists because the root is a phase node). w is a phase node if and only if the number of vertices n_u in u and n_w in w satisfy $n_w \le 0.8n_u$. If w is not a phase node, such a u is called the *phase ancestor* of w. Define a *phase* with phase node u to be all computation nodes whose phase ancestor if u, as well as u itself. See Figure 2 for an illustration.

Given the definition of phase nodes, we define E_{large} as follows: In a phase node, E_{large} is the set of all hyperedges of rank $> n/2$. In a non-phase node w, E_{large} is inherited from its phase ancestor u, i.e. the set of all hyperedges of rank $> n_u/2$ in u. Let G_{large} denote the hypergraph $(V(G), E_{\text{large}})$. We let E_{small} be the complement set of hyperedges $E \setminus E_{\text{large}}$, and $G_{\text{small}} = (V(G), E_{\text{small}})$.

4.1.3 The Last Base Case: Full Revelation. We are now ready to describe the last base case that we call full revelation. Let $\beta = \lambda - \lambda_L$, where λ_L is the min-cut value in G_{large}. The last base case is invoked when $\beta < \lambda/N$.

The algorithm samples a random subgraph $H \sim G(p)$ conditioned on the event that the contracted hyperedges in E_{large} do not contract the whole hypergraph into a singleton. This is done in two steps. First, we write a DNF formula for the disconnection event in G_{large}, and apply Lemma 2.17 to contract each hyperedge in E_{large} with probability $1 - p$ conditioned on the event that G_{large} is not contracted into a single vertex. Second, we directly sample

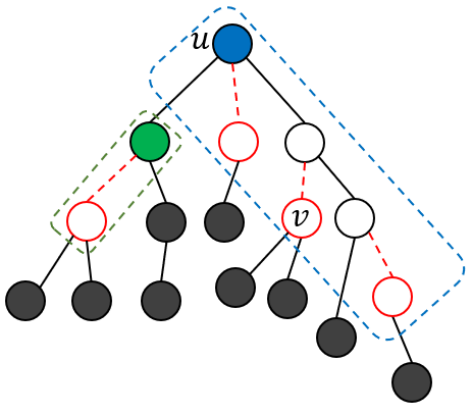

Figure 2: A depiction of phases in the computation tree. The filled in nodes are phase nodes. The blue and green nodes respectively root the blue and green phases. Each phase can contain successful recursive steps, shown by solid black edges and black nodes, and failed recursive steps, shown by dashed red edges and red nodes. In a phase, every node has a phase ancestor which is the root node of the phase; for instance, u is the phase ancestor of v (and of every other node in the blue phase).

the remaining uncontracted hyperedges in E_{small}, that is we independently contract each of those hyperedges with probability $1-p$. The resulting hypergraph H follows the desired distribution.

The algorithm repeats $8n^2$ independent samples of the above process to obtain samples H_i, and estimates $X_i = 0$ if H_i is contracted into a singleton, and $X_i = 1$ otherwise. Let X be the average of all these estimators X_i. Next, we use the DNF counting algorithm in Lemma 2.16 to get an unbiased estimator Z of $u_{G_{\text{large}}}(p)$. The product XZ is the estimator of $u_G(p)$ output by the algorithm.

We are now left to describe the recursive step of the algorithm. For this purpose, we need to first establish some properties of large edges:

4.1.4 Properties of Large Edges. The first property is that the association of E_{large} with large ranks and E_{small} with small ranks is approximately correct:

Fact 4.1. *Any hyperedge in E_{large} has rank at least $0.3n$, and any hyperedge in E_{small} has rank at most $0.7n$.*

The algorithm only contracts non-trivial hyperedges during recursion, i.e., hyperedges that are contracted into a singleton supervertex are removed. We assert that all large hyperedges are candidates for contraction:

Fact 4.2. *Suppose w is a non-phase node and $E_{\text{large}}(w)$ is inherited from the phase ancestor u of w. Then, every hyperedge in $E_{\text{large}}(u)$ still appears in $E_{\text{large}}(w)$, but may be partially contracted.*

Finally, we come to the most important property, that of the simple structure of cuts in G_{large}. To explain this, let us introduce the following property:

Definition 4.3 (pairwise intersecting property)**.** A set of hyperedges is *pairwise intersecting* if any two hyperedges in the set share at least one vertex.

Note that in particular, any set of hyperedges of rank $> n/2$ satisfy the pairwise intersecting property because the sum of ranks of any two hyperedges is more than n. We assert that this property continues to hold with our modified definition of large hyperedges:

Fact 4.4. *The set of hyperedges E_{large} satisfies the pairwise intersection property.*

This allows us to conclude that G_{large} gets disconnected if and only if any degree cut fails.

Lemma 4.5. *If a hypergraph G satisfies the pairwise intersecting property, then G disconnects if and only if some degree cut fails. In particular, by Fact 4.4, this is true for G_{large}.*

Now, the event of G_{large} getting disconnected can be written into a DNF formula F, where each variable represents the failure of a hyperedge and each clause represents the failure of a degree cut in G_{large} as the logical AND of the failure of all hyperedges in the cut. F has n clauses and m variables. Therefore, by Lemmas 2.16 and 2.17, we have the following:

Lemma 4.6. *We can do the following in $O(n^2 m)$ time:*

(1) Compute an unbiased estimator of $u_{G_{\text{large}}}(p)$ with relative variance at most 1.

(2) Sample a contracted hypergraph H of G_{large} where every hyperedge in G_{large} is contracted with probability $1-p$, conditioned on the event that H is not a singleton supervertex.

4.1.5 Recursive Cases. In universally small hypergraphs, the algorithm is identical to the enumeration-based algorithm in Section 3. We run random contraction with $q^{\lambda} = n^{-10}$, and repeat the step $16n^{12}$ times.

The algorithm for the existentially large case is now different because we cannot afford to enumerate m events. Recall that when there exist large hyperedges, the algorithm divides into two cases depending on the value of $\beta = \lambda - \lambda_L$, where λ_L is the min-cut value in G_{large}. When $\beta < \lambda/N$, we get full revelation that we have already described as the last base case.

We call the remaining case when $\beta \geq \lambda/N$ *partial revelation*. In that partial revelation case, the algorithm still runs a form of random contraction, but only in a subspace of the entire probability space. We use the parameter β to control the speed of random contraction. By Lemma 4.5, $\lambda_L = \min_u d_{\text{large}}(u)$, i.e., the minimum degree of a vertex in G_{large}. Intuitively, β is used to control the number of small hyperedges in each degree cut, which measures the speed of random contraction when no large hyperedges get contracted. Note that $0 \leq \beta \leq \lambda$. Ideally, we want to decrease β to as small as λ/N, which reduces to the full revelation case. However, β can be non-monotone as both λ and λ_L can increase because of contraction during recursion. So, we define another parameter γ that can be related to β to bound the depth of recursion in a phase. Let $\gamma = \ell - \lambda_L$, where $\ell = |E_{\text{large}}|$ is the number of large hyperedges. We show that unlike β, γ is monotone in a phase:

Lemma 4.7. *Suppose v, w are nodes in the same phase. If w is a descendant of v, then $\gamma_v \geq \gamma_w$.*

Algorithm for partial revelation. The algorithm runs random contraction at a more aggressive rate $q^\beta = n^{-700}$.[7] This is done in two steps. First, we write a DNF formula for the disconnection in G_{large}, and apply Lemma 4.6 to contract each hyperedge in E_{large} with probability $1-q$ conditioned on G_{large} not being contracted into a singleton. Second, we independently contract each uncontracted hyperedge in E_{small} with probability $1-q$. The resulting hypergraph H follows the distribution of $H \sim G(q)$ conditioned on the event that the contracted hyperedges in E_{large} do not contract the whole hypergraph into a singleton.

The algorithm repeats $32n^{704}$ independent samples H_i, and recursively computes a (biased) estimator X_i of $u_{H_i}(p/q)$. Let X be the average of all these estimators X_i. Next, we use the DNF counting algorithm in Lemma 4.6 to get an unbiased estimator Z of $u_{G_{\text{large}}}(q)$. The product XZ is the estimator of $u_G(p)$ output by the algorithm.

In the rest of the paper, we call this the *sampling-based* algorithm.

4.2 Bias of the Estimator

We first show that all base cases have bias at most δ, and the recursive steps are unbiased. Then, we prove by induction that the recursion keeps the same bound δ on bias.

We introduce some notations when E_{large} and E_{small} are uniquely defined in context. Let $G(p_1, p_2)$ be the random subgraph formed by independently contracting each hyperedge in E_{large} with probability $1-p_1$, and each hyperedge in E_{small} with probability $1-p_2$. Let D_L be the event that in some random contraction, the contracted hyperedges in E_{large} do not contract the whole hypergraph into a singleton.

Base cases. The first base case of $n = O(1)$ outputs the exact value of $u_G(p)$ by Lemma 3.1. The second base case of disconnected G is trivial. In the third base case, the bias is $0 - u_G(p) \in [-\delta, 0]$. Next, we prove that the algorithm in full revelation case is unbiased.

Lemma 4.8. *The algorithm in the full revelation case outputs an unbiased estimator of $u_G(p)$.*

Recursive cases. A random contraction step in the universally small case is unbiased by Lemma 2.2. So, we only need to show this for the partial revelation case. We do this in two steps. First, we assume that the inductive subproblems in this case return exact estimators, and show that the resulting estimator after this step is unbiased. Then, we use this fact to show that if the inductive subproblems return biased estimators, then the bias does not increase after the partial revelation step.

Define a *partial revelation step* to be that of the algorithm in the partial revelation case, except that we now directly use $u_{G_{\text{large}}}(q)$ times average of $u_{H_i}(p/q)$ as the estimator instead of recursively estimating them.

Lemma 4.9. *A partial revelation step is an unbiased estimator of $u_G(p)$.*

We now prove the inductive claim on the bias of the estimator.

[7] The reason for this large polynomial in n is as follows. In the proof of Lemma 4.12, the algorithm needs to repeat the random contraction $O(n^4 q^{-\beta})$ times, and we will show that each trial has failure probability $n^2 q^{1.01\beta}$. We need their product $O(n^6 q^{0.01\beta})$ to be $o(1)$, hence the choice $q^\beta = n^{-700}$.

Lemma 4.10. *The sampling-based unreliability algorithm outputs an estimator with negatively one-sided bias of at most δ.*

Proof. We prove by induction. In the base case of $p^\lambda < 2^{-3N}$, the output is 0; so, the bias is negatively one-sided and upper bounded by $u_G(p) \le n^2 p^\lambda \le (2^N)^2 \cdot 2^{-3N} = 2^{-N} = \delta$. The other base cases are unbiased.

In a random contraction step of universally small case, we take the average $X = \frac{1}{M}\sum_{i\le M} X_i$. By the inductive hypothesis, each X_i satisfies $\mathbb{E}[X_i|H_i] - u_{H_i}(p/q) \in [-\delta, 0]$. Then,

$$\begin{aligned}\mathbb{E}[X] - u_G(p) &= \frac{1}{M}\sum_{i\le M} \mathbb{E}[X_i] - u_G(p)\\ &= \frac{1}{M}\sum_{i\le M} \mathbb{E}_{H_i}[\mathbb{E}[X_i|H_i]] - \mathbb{E}_{H_i}[u_{H_i}(p/q)]\\ &= \frac{1}{M}\sum_{i\le M} \mathbb{E}_{H_i}[\mathbb{E}[X_i|H_i] - u_{H_i}(p/q)] \in [-\delta, 0].\end{aligned}$$

In the partial revelation case, Z is the DNF sampling estimator of $u_L = u_{G_{\text{large}}}(q)$, which is unbiased and independent of X. Next, we bound the bias of X compared to $u_G(p)/u_L$. We take average $X = \frac{1}{M}\sum_{i\le M} X_i$, and each X_i is an estimator for $u_{H_i}(p/q)$ with $\mathbb{E}[X_i|H_i] - u_{H_i}(p/q) \in [-\delta, 0]$ by the inductive hypothesis. Note that here H_i is sampled from a different distribution where $\mathbb{E}[u_H(p/q)] = u_G(p)/u_L$ by Lemma 4.9. Then,

$$\begin{aligned}\mathbb{E}[X] - \frac{u_G(p)}{u_L} &= \frac{1}{M}\sum_{i\le M} \mathbb{E}[X_i] - \frac{u_G(p)}{u_L}\\ &= \frac{1}{M}\sum_{i\le M} \mathbb{E}_{H_i}[\mathbb{E}[X_i|H_i]] - \mathbb{E}_{H_i}[u_{H_i}(p/q)]\\ &= \frac{1}{M}\sum_{i\le M} \mathbb{E}_{H_i}[\mathbb{E}[X_i|H_i] - u_{H_i}(p/q)] \in [-\delta, 0].\end{aligned}$$

After scaling by $\mathbb{E}[Z] = u_L \le 1$, the overall bias of partial revelation case is

$$\begin{aligned}\mathbb{E}[XZ] - u_G(p) &= \mathbb{E}[X]\mathbb{E}[Z] - u_G(p)\\ &= \left(\mathbb{E}[X] - \frac{u_G(p)}{u_L}\right)\cdot u_L \in [-\delta, 0].\end{aligned}$$

□

4.3 Capped Relative Variance of the Estimator

We sketch the steps of the analysis that bound the relative variance of the estimator; the details are deferred to the full version. First, consider the base cases. There are two non-trivial cases: full revelation and when $p^\lambda < 2^{-3N}$. (Other base cases are deterministic and unbiased.) For the full revelation base case, we show that the estimator (which is unbiased) has bounded relative variance:

Lemma 4.11. *The algorithm for full revelation case outputs an unbiased estimator of relative variance at most* 3.

Finally, consider the base case when $p^\lambda < 2^{-3N}$. Here, the complication is that the estimator is biased. So, we bound its δ-capped relative variance (defined in Section 2) instead of the standard relative variance (recall that the bias of the estimator is bounded by δ by Lemma 4.10).

Next, we consider the recursive calls. We show that the recursive calls do not increase relative variance. First, consider the case of universally small hypergraphs. We have shown that a random contraction step has relative variance at most $n^2q^{-\lambda} = n^{12}$ in Lemma 3.5. Therefore, by repeating a sufficiently large number of times, this recursive step does not increase the (capped) relative variance.

Next, consider the other recursive case of a partial revelation step. By using the argument in Lemma 3.5 restricted to the case of disconnection in large edges, we show that each estimator $u_{H_i}(p/q)$ has relative variance at most $n^4q^{-\beta} = n^{704}$. So, by repeating a sufficiently large number of times, this recursive step also does not increase the (capped) relative variance.

Above, we argued the relative variance from a single recursive step. We then use an induction similar to Lemma 3.3 to bound the overall capped relative variance. The main difference here is that we use Facts 2.5 and 2.6 to compose the capped relative variance instead of the uncapped version. Finally, we get:

Lemma 4.12. *The sampling-based unreliability algorithm outputs an estimator X of $u_G(p)$ with δ-capped relative variance $\eta_\delta[X] \le 3$.*

4.4 Running Time

The argument is similar to Section 3.3; we give a sketch here and defer the details to the full version. We color each recursive call as black or red – intuitively, they represent "success" or "failure". For both the universally small and partial revelation cases, if the child subproblem is a phase node, then the recursive call is marked a success (i.e., a black node). This is the only type of success for the universally small case, which we call type 1 success. For the partial revelation case, we have an additional situation where we declare type 2 success: when the parameter γ decreases to 0.9γ and $|\ell - \lambda| \le 0.1\beta$. All other recursive calls are failures.

Now, the recursion tree satisfies the following properties:

(1) Each subproblem makes $n^{O(1)}$ recursive calls. This is clear from the algorithm description.

(2) The algorithm reaches the base case after $O(\log n \cdot \log N)$ black recursive calls (interleaved with red recursive calls):

Lemma 4.13. *There can be at most $O(\log n \cdot \log N)$ black recursive calls from root to a base case.*

(3) At each subproblem, the expected number of red recursive calls is $o(1)$. We prove this later in the section.

Lemma 3.12 shows that these properties give a upper bound of $n^{O(\log n \cdot \log N)}$ on the number of recursive calls. If we charge the time of DNF sampling and random contraction to the subproblem on the contracted hypergraph, then each subproblem spends $O(n^2m)$ time outside the recursive calls, where the bottlenecks are DNF sampling and DNF probability estimation given by Lemma 4.6. Therefore, the overall expected running time is $m \cdot n^{O(\log n \cdot \log N)}$.

5 CONCLUSION

In this paper, we initiated the study of unreliability in hypergraphs and provided quasi-polynomial time approximation schemes for the problem. The immediate open question is whether there is a PTAS (or even FPTAS) for this problem. More generally, we hope that our work will inspire further exploration of the rich space of reliability problems in hypergraphs.

ACKNOWLEDGMENTS

RC and DP were supported in part by NSF grants CCF-1750140 (CAREER), CCF-1955703, and CCF-2329230. This research was done at the Simons Institute, UC Berkeley under the aegis of the Fall 2023 semester program on *Data Structures and Optimization for Fast Algorithms*. DP also wishes to acknowledge the support of Google Research, where he held a part-time visiting appointment at the time of this research. RC and DP would like to thank William He and Davidson Zhu for useful discussions at the early stages of this research.

REFERENCES

[1] Noga Alon, Alan M. Frieze, and Dominic Welsh. 1995. Polynomial Time Randomized Approximation Schemes for Tutte-Gröthendieck Invariants: The Dense Case. *Random Struct. Algorithms* 6, 4 (1995), 459–478.

[2] Christian Bick, Elizabeth Gross, Heather A. Harrington, and Michael T. Schaub. 2023. What Are Higher-Order Networks? *SIAM Rev.* 65, 3 (2023), 686–731.

[3] Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. 2024. Beyond the Quadratic Time Barrier for Network Unreliability. In *2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1542–1567.

[4] Sanjay Kumar Chaturvedi. 2016. *Network reliability: measures and evaluation*. John Wiley & Sons.

[5] Chandra Chekuri and Kent Quanrud. 2021. Isolating Cuts, (Bi-)Submodularity, and Faster Algorithms for Connectivity. In *48th International Colloquium on Automata, Languages, and Programming (ICALP)*. 50:1–50:20.

[6] Chandra Chekuri and Chao Xu. 2018. Minimum cuts and sparsification in hypergraphs. *SIAM J. Comput.* 47, 6 (2018), 2118–2156.

[7] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 612–623.

[8] Charles J Colbourn. 1987. *The combinatorics of network reliability*. Oxford University Press, Inc.

[9] William H Cunningham. 1983. Decomposition of submodular functions. *Combinatorica* 3, 1 (1983), 53–68.

[10] E.A. Dinitz, A.V. Karzanov, and M.V. Lomonosov. 1976. On the structure of a family of minimal weighted cuts in a graph. *Studies in Discrete Optimization (in Russian), (ed. A.A. Fridman), Nauka, Moscow* (1976), 290–306.

[11] Mohsen Ghaffari, David R Karger, and Debmalya Panigrahi. 2017. Random contractions and sampling for hypergraph and hedge connectivity. In *2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1101–1114.

[12] Gourab Ghoshal, Vinko Zlatic, Guido Caldarelli, and M. E. J. Newman. 2009. Random hypergraphs and their applications. *Phys. Rev. E* 79 (2009), 066118. Issue 6.

[13] David G. Harris and Aravind Srinivasan. 2018. Improved bounds and algorithms for graph cuts and network reliability. *Random Structures & Algorithms* 52, 1 (2018), 74–135.

[14] David R. Karger. 1993. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 21–30.

[15] David R. Karger. 1999. A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem. *SIAM J. Comput.* 29, 2 (1999), 492–514.

[16] David R. Karger. 2016. A Fast and Simple Unbiased Estimator for Network (Un)reliability. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 635–644.

[17] David R. Karger. 2017. Faster (and Still Pretty Simple) Unbiased Estimators for Network (Un)reliability. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 755–766.

[18] David R. Karger. 2020. A Phase Transition and a Quadratic Time Unbiased Estimator for Network Reliability. In *52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 485–495.

[19] Richard M Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo approximation algorithms for enumeration problems. *Journal of algorithms* 10, 3 (1989), 429–448.

[20] Dmitry Kogan and Robert Krauthgamer. 2015. Sketching Cuts in Graphs and Hypergraphs. In *2015 Conference on Innovations in Theoretical Computer Science (ITCS)*. 367–376.

[21] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

Received 13-NOV-2023; accepted 2024-02-11