An FPRAS for Two Terminal Reliability in Directed Acyclic Graphs

Weiming Feng
Institute for Theoretical Studies, ETH Zürich, Switzerland

Heng Guo
School of Informatics, University of Edinburgh, UK

Abstract

We give a fully polynomial-time randomized approximation scheme (FPRAS) for two terminal reliability in directed acyclic graphs (DAGs). In contrast, we also show the complementing problem of approximating two terminal unreliability in DAGs is #**BIS**-hard.

2012 ACM Subject Classification Networks → Network reliability; Mathematics of computing → Approximation algorithms; Theory of computation → Generating random combinatorial structures

Keywords and phrases Approximate counting, Network reliability, Sampling algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.62

Category Track A: Algorithms, Complexity and Games

Related Version *Preprint*: `https://arxiv.org/abs/2310.00938`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 947778). Weiming Feng acknowledges support of Dr. Max Rössler, the Walter Haefner Foundation and the ETH Zürich Foundation.

Acknowledgements We thank Kuldeep S. Meel for bringing the problem to our attention, Antoine Amarilli for explaining their method to us, and Marcelo Arenas for insightful discussions. We also thank Zongchen Chen for suggesting a better presentation of Theorem 1, and Mark Jerrum for some useful insights. This work was done in part while Weiming Feng was visiting the Simons Institute for the Theory of Computing.

1 Introduction

Network reliability is one of the first problems studied in counting complexity. Indeed, $s-t$ reliability is listed as one of the first thirteen complete problems when Valiant [29] introduced the counting complexity class #**P**. The general setting is that given a (directed or undirected) graph G, each edge e of G fails independently with probability q_e. The problem of $s-t$ reliability is then asking the probability that in the remaining graph, the source vertex s can reach the sink t. There are also other variants, where one may ask the probability of various kinds of connectivity properties of the remaining graph. These problems have been extensively studied, and apparently most variants are #**P**-complete [6, 19, 8, 28, 7, 11].

While the exact complexity of reliability problems is quite well understood, their approximation complexity is not. Indeed, the approximation complexity of the first studied $s-t$ reliability is still open in either directed or undirected graphs. One main exception is the *all-terminal* version (where one is interested in the remaining graph being connected or disconnected). A famous result by Karger [21] gives the first fully polynomial-time randomized approximation scheme (FPRAS) for all-terminal *unreliability*, while about two decades later, Guo and Jerrum [16] give the first FPRAS for all-terminal *reliability*. The latter algorithm is under the partial rejection sampling framework [17], and the Markov chain Monte Carlo

(MCMC) method is also shown to be efficient shortly after [3, 12]. See [18, 22, 9], [15], and [4, 10] for more recent results and improved running times along the three lines above for the all-terminal version respectively.

The success of these methods implies that the solution space of all-terminal reliability is well-connected via local moves. However, this is not the case for the two-terminal version (namely the $s-t$ version). Instead, the natural local-move Markov chain for $s-t$ reliability is torpidly mixing. Here the solution space consists of all spanning subgraphs (namely a subset of edges) in which s can reach t. Consider a (directed or undirected) graph composed of two paths of equal length connecting s and t. Suppose we start from one path and leave the other path empty. Then before the other path is all included in the current state, we cannot remove any edge of the initial path. This creates an exponential bottleneck for local-move Markov chains, and it suggests that a different approach is required.

In this paper, we give an FPRAS for the $s-t$ reliability in directed acyclic graphs. Note that the exact version of this problem is $\#\mathbf{P}$-complete [28, Sec 3], even restricted to planar DAGs where the vertex degrees are at most 3 [27, Theorem 3]. Our result positively resolves an open problem by Zenklusen and Laumanns [30]. Without loss of generality, in the theorem below we assume that any vertex other than s has at least one incoming edge, and thus $|E| \geq |V| - 1$ for the input $G = (V, E)$.

▶ **Theorem 1.** *Let $G = (V, E)$ be a directed acyclic graph (DAG), failure probabilities $\mathbf{q} = (q_e)_{e \in E} \in [0, 1]^E$, two vertices $s, t \in V$, and $\varepsilon > 0$. There is a randomized algorithm that takes $(G, \mathbf{q}, s, t, \varepsilon)$ as inputs and outputs a $(1 \pm \varepsilon)$-approximation to the $s-t$ reliability with probability at least $3/4$ in time $\widetilde{O}(n^6 m^4 \max\{m^4, \varepsilon^{-4}\})$ where $n = |V|$, $m = |E|$, and $\widetilde{O}$ hides polylog(n/ϵ) factors.*

The running time of Theorem 1 is $\widetilde{O}(n^6 m^8)$ when $\varepsilon > 1/m$, and is $\widetilde{O}(n^6 m^4 / \varepsilon^4)$ when $\varepsilon < 1/m$. The reason behind this running time is that our algorithm always outputs at least a $(1 \pm 1/m)$-approximation. Thus, when $\varepsilon > 1/m$, it does not matter what ε actually is for the running time. This high level of precision is required for the correctness of the algorithm.

As hinted earlier, our method is a significant departure from the techniques for the all-terminal versions. Indeed, a classical result by Karp and Luby [23, 25] has shown how to efficiently estimate the size of a union of sets. A direct application of this method to $s-t$ reliability is efficient only for certain special cases [24, 30]. Our main observation is to use the Karp-Luby method as a subroutine in dynamic programming using the structure of DAGs. Let $s = v_1, \ldots, v_n = t$ be a topological ordering of the DAG G. (Note that we can ignore vertices before s and after t.) Let R_u be the $u-t$ reliability so that our goal is to estimate R_s. We inductively estimate R_{v_i} from $i = n$ to $i = 1$. For each vertex u, our algorithm maintains an estimator of R_u and a set S_u of samples of subgraphs in which u can reach t. In the induction step, we use the Karp-Luby method to generate the next estimator, and use a self-reduction similar to the Jerrum-Valiant-Vazirani sampling to counting reduction [20] to generate samples. Both tasks can be done efficiently using only what have been computed for previous vertices. Moreover, a naive implementation of this outline would require exponentially many samples. To avoid this, we reuse generated samples and carefully analyze the impact of doing so on the overall error bound. A more detailed overview is given in Section 1.1.

Our technique is inspired by an FPRAS for the number of accepting strings of non-deterministic finite automata (#NFA), found by Arenas, Croquevielle, Jayaram, and Riveros [5], which in turn used some techniques from a quasi-polynomial-time algorithm for sampling words from context-free languages by Gore, Jerrum, Kannan, Sweedyk, and

Mahaney [14]. Their #NFA algorithm runs in time $O\left(\left(\frac{n\ell}{\varepsilon}\right)^{17}\right)$,[1] where n is the number of states and ℓ is the string length. More recently, Meel, Chakraborty, and Mathur claim an improved running algorithm which runs in time $\widetilde{O}(\frac{n^4\ell^{11}}{\varepsilon^4})$ [26, Theorem 3]. These algorithms first normalize the NFA into a particular layered structure. Applying similar methods on the $s-t$ reliability problem can simplify the analysis, but would greatly slow down the algorithm. In contrast, our method works directly on the DAG. This makes our estimation and sampling subroutines interlock in an intricate way. To analyze the algorithm, we have to carefully separate out various sources of randomness. This leads to a considerably more sophisticated analysis, with a reward of a much better (albeit still high) running time.

Independently and simultaneously, Amarilli, van Bremen, and Meel [2] also found an FPRAS for $s-t$ reliability in DAGs. Their method is to reduce the problem to #NFA via a sequence of reductions, and then invoke the algorithm in [5]. Indeed, as Marcelo Arenas subsequently pointed out to us, counting the number of subgraphs of a DAG in which s can reach t belongs to a complexity class **SpanL** [1], where #NFA is **SpanL**-complete under polynomial-time parsimonious reductions. In particular, every problem in **SpanL** admits an FPRAS because #NFA admits one [5], which implies that $s-t$ reliability in DAGs admits an FPRAS if $q_e = 1/2$ for all edges. The method of [2] reduces a reliability instance of n vertices and m edges, where $q_e = 1/2$ for all edges, to estimating length m accepting strings of an NFA with $O(m^2)$ states.[2] As a consequence, their algorithm (even using the faster algorithm for #NFA [26]) has a running time of $O\left(m^{19}\varepsilon^{-4}\right)$. When $q_e \neq 1/2$, their reduction needs to expand the instance further to reduce to the $q_e = 1/2$ case, slowing down the algorithm even more. In contrast, our algorithm deals with all possible probabilities $0 \leq q_e < 1$ in a unified way. In any case, an algorithm via reductions is much slower than the direct algorithm in Theorem 1.

As both all-terminal reliability and unreliability in undirected graphs have FPRASes [21, 16], one may wonder if FPRAS exists for $s-t$ *unreliability* in DAGs. Here $s-t$ unreliability is the probability that s cannot reach t in the random subgraph. In contrast to Theorem 1, we show that this problem is #**BIS**-hard, where #**BIS** is the problem of counting independent sets in bipartite graphs, whose approximation complexity is still open. This is a central problem in the complexity of approximate counting [13], and is conjectured to have no FPRAS.

▶ **Theorem 2.** *There is no FPRAS to estimate* $s-t$ unreliability *in DAGs unless there is an FPRAS for* #**BIS**. *This is still true even if all edges fail with the same probability.*

Theorem 2 is proved in Section 5. The hardness of $s-t$ unreliability does not contradict Theorem 1. This is because a good relative approximation of x does not necessarily provide a good approximation of $1-x$, especially when x is close to 1.

The complexity of estimating $s-t$ reliability in general directed or undirected graphs remains open. We hope that our work sheds some new light on these decades old problems. Another open problem is to reduce the running time of Theorem 1, as currently the exponent of the polynomial is still high.

[1] The running time of the algorithm in [5] is not explicitly given. This bound is obtained by going through their proof.

[2] In fact, [2] first reduces the reliability instance to an nOBDD (non-deterministic ordered binary decision diagram) of size $O(m)$, which can be further reduced to an NFA of size $O(m^2)$. As they are working with a more general context, no explicit reduction is given for the $s-t$ reliability problem in DAGs. We provide a direct (and essentially the same) reduction in the full version of this paper.

1.1 Algorithm overview

Here we give an overview of our algorithm. For simplicity, we assume that $q_e = 1/2$ for all edges. The general case of $0 \le q_e < 1$ can be solved with very small tweaks.

Let $s = v_1 \prec \cdots \prec v_n = t$ be a topological ordering of the DAG G. (Note that we can ignore vertices before s and after t.) Let R_u be the $u - t$ reliability so that our goal is to estimate R_s. Note that R_{v_i} depends only on the subgraph induced by the vertex set $\{v_i, v_{i+1}, \ldots, v_n\}$. We denote this subgraph by $G_{v_i} = (V_{v_i}, E_{v_i})$. As we assumed $q_e = 1/2$, estimating R_{v_i} is equivalent to estimating the number of (spanning) subgraphs of G_{v_i} in which v_i can reach t. Denote the latter quantity by Z_{v_i} so that $R_{v_i} = Z_{v_i}/2^{|E_{v_i}|}$. For each vertex v_i, our algorithm maintains an estimator and a multi-set of random samples:

- $\widetilde{Z}_{v_i}$:[3] an estimator that approximates Z_{v_i} with high probability;
- S_{v_i}: a multi-set of random subgraphs, where each $H \in S_{v_i}$ is an approximate sample from π_{v_i} and π_{v_i} is the uniform distribution over all spanning subgraphs of G_{v_i} in which v_i can reach t.

Our algorithm computes $\widetilde{Z}_{v_i}$ and S_{v_i} for i from n to 1 by dynamic programming. The base case $v_n = t$ is trivial. In the induction step, suppose the vertex v_i has d out-neighbors $u_1, u_2, \ldots, u_d$. Note that each u_j for $j \in [d]$ comes after v_i in the topological ordering. Let us further assume $v_i \prec u_1 \prec, \ldots, \prec u_d$. For any subgraph H of G_{v_i}, if v_i can reach t in H, then there exists a neighbor u_j such that v_i can reach u_j and u_j can reach t in H. We can write Z_{v_i} as the size of the union $\Omega := \cup_{j=1}^{d} \Omega^{(j)}$, where $\Omega^{(j)}$ contains all subgraph of G_{v_i} where v_i can reach t through the neighbor u_j. Note that it is straightforward to estimate the size of $\Omega^{(j)}$ given $\widetilde{Z}_{u_j}$, and to generate uniform random subgraphs from $\Omega^{(j)}$ using samples in S_{u_j}. Given the size and samples from $\Omega^{(j)}$, a classical algorithm by Karp and Luby [23, 25] can be applied to efficiently estimate the size of the union of sets, namely to compute $\widetilde{Z}_{v_i}$.

The more complicated task is to generate the samples of S_{v_i}. We use a sampling to counting self-reduction á la Jerrum-Valiant-Vazirani [20]. To generate a sample H, we go through each edge e in G_{v_i}, deciding if e is added into H according to its conditional marginal probability. The first edge to consider is (v_i, u_1). Its marginal probability depends on how many subgraphs in Ω contain it. This quantity is the same as the number of subgraphs in which Λ can reach t, where Λ is a new vertex after contracting v_i and u_1. To estimate this number, denoted by Z_Λ, we use the Karp-Luby algorithm again. Note that the Karp-Luby algorithm requires estimates and samples from all out-neighbours of Λ, which have been computed already as these vertices are larger in the topological ordering than v_i. Having estimated Z_Λ, we estimate the marginal probability of (v_i, u_1) and decide if it is included in H. The process then continues to consider the next edge. In each step, we contract all vertices that can be reached from v_i into Λ, and keep estimating new Z_Λ using the Karp-Luby algorithm to compute the conditional marginal probabilities, until all edges are considered to generate H.

A naive implementation of the process above is to generate fresh samples every time S_u is used, which would lead to an exponential number of samples required. To maintain efficiency, the key property of our algorithm is to *reuse* random samples. For any vertex u, the algorithm generates the multi-set of samples S_u only once. Whenever the algorithm needs to use random samples for u, it always picks one sample from the same set S_u. Hence, one sample may be used multiple times during the whole algorithm. Reusing samples

[3] Our algorithm actually directly maintains an estimate $\widetilde{R}_{v_i}$ to the reliability R_{v_i}. In this overview, we use $\widetilde{Z}_{v_i}$ instead for simplicity.

introduces very complicated correlation among all $(\widetilde{Z}_u, S_u)$'s, which is a challenge to proving the correctness of the algorithm. Essentially, our analysis shows that as long as the estimates ($\widetilde{Z}_u$'s) and the samples are accurate enough, the overall error can be controlled. Accurate estimates of Z_u's allow us to bound the total variation (TV) distance between our samples and perfect samples. In turn, the small TV distance implies that there is a coupling between them, which helps us bound the errors of the estimates. This way, we circumvent the effect of correlation on the analysis and achieve the desired overall error bound.

For the overall running time, there are two tasks for each vertex, namely the estimation step (based on Karp-Luby) and the sample generation step. As we also need to perform estimation steps as subroutines when generating samples, the running time is dominated by the time for the sampling step. Let ℓ be the number of samples required per vertex, so that the total number of samples generated is $n\ell$. Roughly speaking, because we can only use the union bound due to various correlation, and because errors accumulate throughout dynamic programming, we set the error to be $\delta := n^{-1}\min\{m^{-1}, \varepsilon\}$ for the estimation step. Each estimation has two stages, first getting a constant approximation and then using the crude estimation to tune the parameters and obtain a $1 \pm \delta$ approximation. This succeeds with constant probability using $O(n\delta^{-2})$ samples. The estimator itself requires $O(m)$ time to compute, and thus the total running time for each estimation step with constant success probability is $\widetilde{O}(mn\delta^{-2})$. However, as samples are reused, we need to apply a union bound to control the error over all possible values of the samples, which are exponentially many. This requires us to amplify the success probability of the estimation step to $\exp(-\Omega(m))$, which means we need to repeat the algorithm $O(m)$ times and take the median. Each estimation step thus takes $\widetilde{O}(m^2n\delta^{-2})$ time and $O(mn\delta^{-2})$ samples. Instead of maintaining $O(mn\delta^{-2})$ samples for every vertex, the aforementioned two-stage estimation allows us to spread the cost and maintain $\ell := \frac{O(mn\delta^{-2})}{n} = O(n^2 m \max\{m^2, \varepsilon^{-2}\})$ samples per vertex, which we show suffice with high probability. As we may do up to $O(m)$ estimation steps during each sampling step, the overall running time is then bounded by $\widetilde{O}(n\ell \cdot m \cdot m^2 n\delta^{-2}) = \widetilde{O}\big(n^6 m^4 \max\{m^4, \varepsilon^{-4}\}\big)$.

2 Preliminaries

2.1 Problem definitions

Let $G = (V, E)$ be a directed acyclic graph (DAG). Each directed edge (or arc) $e = (u, v)$ is associated with a failure probability $0 \le q_e < 1$. (Any edge with $q_e = 1$ can be simply removed.) We also assume graph G is simple because parallel edges with failure probabilities $q_{e_1}, q_{e_2}, \ldots, q_{e_k}$ can be replace with one edge with failure probability $q_e = \prod_{i=1}^{m} q_{e_i}$. Given two vertices $s, t \in V$, the $s-t$ reliability problem asks the probability that s can reach t if each edge $e \in E$ fails (namely gets removed) independently with probability q_e. Formally, let $\boldsymbol{q} = (q_e)_{e\in E}$. The $s-t$ reliability problem is to compute

$$R_{G,\boldsymbol{q}}(s,t) := \Pr_{\mathcal{G}}[\,\text{there is a path from } s \text{ to } t \text{ in } \mathcal{G}\,], \tag{1}$$

where $\mathcal{G} = (V, \mathcal{E})$ is a random subgraph of $G = (V, E)$ such that each $e \in E$ is added independently to $\mathcal{E}$ with probability $1 - q_e$.

Closely connected to estimating $s-t$ reliability is a sampling problem, which we call the $s-t$ *subgraph sampling* problem. Here the goal is to sample a random (spanning) subgraph G' conditional on that there is at least one path from s to t in G'. Formally, let $\Omega_{G,s,t}$ be

the set of all subgraphs $H = (V, E_H)$ of G such that $E_H \subseteq E$ and s can reach t in H. The algorithm needs to draw samples from the distribution $\pi_{G,s,t,\boldsymbol{q}}$ whose support is $\Omega_{G,s,t}$ and for any $H = (V, E_H) \in \Omega_{G,s,t}$,

$$\pi_{G,s,t,\boldsymbol{q}}(H) = \frac{1}{R_{G,\boldsymbol{q}}(s,t)} \cdot \prod_{e \in E_H} (1 - q_e) \prod_{f \in E \setminus E_H} q_f. \tag{2}$$

2.2 More notations

Fix a DAG $G = (V, E)$. For any two vertices u and v, we use $u \rightsquigarrow_G v$ to denote that u can reach v in the graph G and use $u \not\rightsquigarrow_G v$ to denote that u cannot reach v in the graph G. It always holds that $u \rightsquigarrow_G u$. Fix two vertices s and t, where s is the source and t is the sink. The failure probabilities $\mathbf{q}$, s, and t will be the same throughout the paper, and thus we omit them from the subscripts. For any vertex $u \in V$, we use $G_u = G[V_u]$ to denote the subgraph of G induced by the vertex set

$$V_u := \{w \in V \mid u \rightsquigarrow_G w \wedge w \rightsquigarrow_G t\}. \tag{3}$$

Without loss of generality, we assume $G_s = G$. This means that all vertices except s have at least one in-neighbour, and thus $m \geq n - 1$. If $G_s \neq G$, then all vertices and edges in $G - G_s$ have no effect on $s - t$ reliability and we can simply ignore them. For the sampling problem, we can first solve it on the graph G_s and then independently add each edge e in $G - G_s$ with probability $1 - q_e$.

Our algorithm actually solves the $u - t$ reliability and $u - t$ subgraph sampling problems in G_u for all $u \in V$. Let $G_u = (V_u, E_u)$. For any subgraph $H = (V_u, E_H)$ of G_u, define the weight function

$$w_u(H) := \begin{cases} \prod_{e \in E_H} (1 - q_e) \prod_{f \in E_u \setminus E_H} q_f & \text{if } u \rightsquigarrow_H t; \\ 0 & \text{if } u \not\rightsquigarrow_H t. \end{cases} \tag{4}$$

Define the distribution π_u by

$$\pi_u(H) := \frac{w_u(H)}{R_u},$$

where the partition function (the normalizing factor)

$$R_u := \sum_{H\text{: subgraph of } G_u} w_u(H)$$

is exactly the $u - t$ reliability in the graph G_u. Finally, let

$$\Omega_u := \{H = (V_u, E_H) \mid E_H \subseteq E_u \wedge u \rightsquigarrow_H t\} \tag{5}$$

be the support of π_u. Also note that R_s and π_s are the probability $R_{G,\boldsymbol{q}}(s,t)$ and the distribution $\pi_{G,s,t,\boldsymbol{q}}$ defined in (1) and (2), respectively. The set Ω_s is the set $\Omega_{G,s,t}$ in Section 2.1.

2.3 The total variation distance and coupling

Let μ and ν be two discrete distributions over Ω. The *total variation distance* between μ and ν is defined by

$$d_{\mathrm{TV}}(\mu, \nu) := \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|.$$

If $X \sim \mu$ and $Y \sim \nu$ are two random variables, we also abuse the notation and write $d_{\mathrm{TV}}(X, Y) := d_{\mathrm{TV}}(\mu, \nu)$.

A *coupling* between μ and ν is a joint distribution (X, Y) such that $X \sim \mu$ and $Y \sim \nu$. The following coupling inequality is well-known.

▶ **Lemma 3.** *For any coupling $\mathcal{C}$ between two random variables $X \sim \mu$ and $Y \sim \nu$, it holds that*

$$\Pr_{\mathcal{C}}[X \neq Y] \geq d_{TV}(\mu, \nu).$$

Moreover, there exists an optimal coupling that achieves equality.

3 The algorithm

In this section we give our algorithm. We also give intuitions behind various design choices, and give some basic properties of the algorithm along the way. We give a sketch of the analysis in Section 4 and the full analysis is in the full version of this paper.

3.1 The framework of the algorithm

As $G = G_s$ is a DAG, there is a topological ordering of all vertices. There may exist many topological orderings. We pick an arbitrary one, say, $v_1, \ldots, v_n$. It must hold that $v_1 = s$ and $v_n = t$. The topological ordering guarantees that if (u, v) is an edge, u must come before v in the ordering, denoted $u \prec v$.

On a high level, our algorithm is to inductively compute an estimator $\widetilde{R}_u$ of R_u, from $u = v_n$ to $u = v_1$. In addition to $\widetilde{R}_u$, we also maintain a multi-set S_u of samples from π_u over Ω_u. For any vertex $u \in V$, let $\Gamma_{\mathrm{out}}(u) := \{w \mid (u, w) \in E\}$ denote the set of out-neighbours of u. Let

$$\ell := (60n + 150m)(400n + 500\left\lceil 10^4 n^2 \max\{m^2, \varepsilon^{-2}\}\right\rceil) = O((n+m)n^2 \max\{m^2, \epsilon^{-2}\}) \tag{6}$$

be the size of S_v for all $v \in V_u$, where n is the number of vertices in G and m is the number of edges in G. The choice of this parameter is explained in the last paragraph of Section 1.1. Our algorithm is outlined in Algorithm 1. Note that G_t is an isolated vertex. For consistency, we let S_t contain ℓ copies of $\emptyset$.

The base case of $v_n = t$ is trivial. The subroutine $\mathsf{Sample}(\cdot)$ uses $(\widetilde{R}_{v_i}, S_{v_i})$ for all $i > k$ and $\widetilde{R}_{v_k}$ to generate samples in S_{v_k}. The subroutine $\mathsf{ApproxCount}(V, E, u, (\widetilde{R}_w, S_w)_{\Gamma_{\mathrm{out}}(u)})$ takes a graph $G = (V, E)$, a vertex u, and $(\widetilde{R}_w, S_w)$ for all $w \in \Gamma_{\mathrm{out}}(u)$ as the input, and it outputs an approximation of the $u - t$ reliability in the graph G. We describe Sample in Section 3.2 and (a slightly more general version of) $\mathsf{ApproxCount}$ in Section 3.3.

3.2 Generate samples

Let $u = v_k$ where $k < n$. Recall that $G_u = (V_u, E_u)$ is the graph defined in (3). The sampling algorithm aims to output a random spanning subgraph $H = (V_u, \mathcal{E})$ from the distribution π_u. The algorithm is based on the sampling-to-counting reduction in [20]. It scans each edge e in E_u and decides whether to put e into the set $\mathcal{E}$ or not. The algorithm maintains two edge sets:

- $E_1 \subseteq E_u$: the set of edges that have been scanned by the algorithm;
- $\mathcal{E} \subseteq E_1$: the current set of edges sampled by the algorithm.

Algorithm 1 An FPRAS for $s-t$ reliabilities in DAGs.

Input: a DAG $G=(V,E)$, a vector $\boldsymbol{q}=(q_e)_{e\in E}$, the source s, the sink t, and an error bound $0<\varepsilon<1$, where $G=G_s$ and $V=\{v_1,v_2,\ldots,v_n\}$ is topologically ordered with $v_1=s$ and $v_n=t$

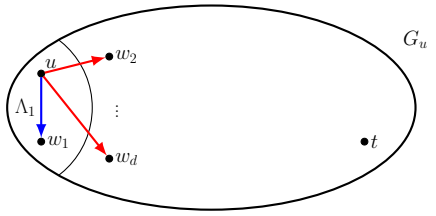
Output: an estimator $\widetilde{R}_s$ of R_s

let $\widetilde{R}_t=1$ and S_t be a multi set of ℓ $\emptyset$'s;
for k *from* $n-1$ *to* 1 **do**
  $\widetilde{R}_{v_k} \leftarrow \mathsf{ApproxCount}\Big(V_{v_k}, E_{v_k}, v_k, (\widetilde{R}_w, S_w)_{w\in\Gamma_{out}(v_k)}\Big)$;
  $S_{v_k} \leftarrow \emptyset$;
  for j *from* 1 *to* ℓ **do**
    $S_{v_k} \leftarrow S_{v_k} \cup \mathsf{Sample}\Big(v_k, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\ldots,v_n\}}, \widetilde{R}_{v_k}\Big)$;
return $\widetilde{R}_s$.

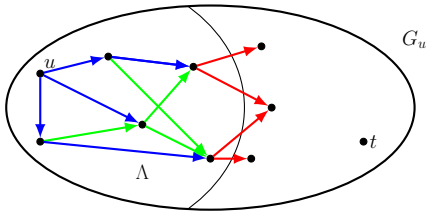
Also, let $E_2 := E_u \setminus E_1$ be the set of edges that have not been scanned yet by the algorithm. Given any $\mathcal{E}$, we can uniquely define the following subset of vertices

$$\Lambda = \Lambda_{\mathcal{E}} := \mathsf{rch}(u, V_u, \mathcal{E}) = \{w \in V_u \mid u \text{ can reach } w \text{ through edges in } \mathcal{E}\}. \tag{7}$$

In other words, let $G'=(V_u,\mathcal{E})$ and Λ is the set of vertices that u can reach in G'. Note that $u\in\Lambda$ for any $\mathcal{E}$. We will keep updating Λ as $\mathcal{E}$ expands. When calculating the marginal probability of the next edge, the path to t can start from any vertex in Λ. Thus we need to estimate the reliability from Λ to t. Instead of having a single source, as called in Algorithm 1, we use a slightly more general version of ApproxCount, described in Section 3.3, to allow a set Λ of sources. This subroutine ApproxCount takes input $(V,E,\Lambda,(\widetilde{R}_w,S_w)_{w\in\partial\Lambda})$ and approximates the $\Lambda-t$ reliability in (V,E), which is the probability that there exists at least one vertex in Λ being able to reach t if each edge $e\in E$ fails independently with probability q_e. An equivalent way of seeing it is to contract all vertices in Λ into a single vertex u first, and then calculate the $u-t$ reliability in the resulting graph. Sample is described in Algorithm 2, and some illustration is given in Figure 1.



(a) At the start, we consider the first edge (u,w_1). To compute its marginal, we estimate two reliability, where the source is either $\Lambda_1=\{u,w_1\}$ (shown in picture) or just u, respectively.



(b) As Algorithm 2 progresses, there are chosen edges $\mathcal{E}$ (blue), not chosen edges $E_1\setminus\mathcal{E}$ (green), and the boundary edges (red). The set Λ contains vertices reachable from u using only $\mathcal{E}$.

Figure 1 An illustration of sampling from π_u.

▶ Remark 4 (Crash of Sample). The subroutine Sample (Algorithm 2) may crash in following cases: (1) in Algorithm 2, $\partial\Lambda=\emptyset$; (2) in Algorithm 2, $q_ec_0+(1-q_e)c_1=0$; (3) in Algorithm 2, $\frac{w_u(H)}{4p_0p}>1$; (4) in Algorithm 2, $F=0$. If it crashes, we stop Algorithm 1 immediately and output $\widetilde{R}_s=0$.

Algorithm 2 $\mathsf{Sample}\Big(u, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\ldots,v_n\}}, \widetilde{R}_{v_k}\Big)$.

Input: a vertex $u = v_k$, all $(\widetilde{R}_w, S_w)$ for $w \in \{v_{k+1}, v_{k+2}, \ldots, v_n\}$, and $\widetilde{R}_{v_k} = \widetilde{R}_u$
Output: a random subgraph $H = (V_u, \mathcal{E})$

$T \leftarrow \lceil 1000 \log \frac{n}{\varepsilon} \rceil$ and $F \leftarrow 0$;
$p_0 \leftarrow \widetilde{R}_u$;
repeat
  let $p \leftarrow 1$;
  let $E_1 \leftarrow \emptyset$, $E_2 \leftarrow E_u \setminus E_1$ and $\Lambda = \{u\}$;
  while $t \notin \Lambda$ **do**
    let $\partial\Lambda \leftarrow \{w \notin \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E_2\}$; let $w^* \in \partial\Lambda$ be the smallest vertex in the topological ordering; pick an arbitrary edge $e = (w', w^*) \in E_2$ such that $w' \in \Lambda$;
    let $\Lambda_1 \leftarrow \mathsf{rch}(u, V_u, \mathcal{E} \cup \{e\})$;
    $E_1 \leftarrow E_1 \cup \{e\}$ and $E_2 \leftarrow E_2 \setminus \{e\}$;
    $\partial\Lambda_1 \leftarrow \{w \notin \Lambda_1 \mid \exists w' \in \Lambda_1 \text{ s.t. } (w', w) \in E_2\}$ and $\partial\Lambda \leftarrow \{w \notin \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E_2\}$;
    $c_0 \leftarrow \mathsf{ApproxCount}(V_u, E_2, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$;
    $c_1 \leftarrow \mathsf{ApproxCount}(V_u, E_2, \Lambda_1, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda_1})$;
    let $c \leftarrow 1$ with probability $\frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}$; otherwise $c \leftarrow 0$;
    if $c = 1$, **then** let $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$, $\Lambda \leftarrow \Lambda_1$, $p \leftarrow p\frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}$;
    if $c = 0$, **then** let $p \leftarrow p\Big(1 - \frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}\Big)$;
  for *all edges* $e \in E_2$ **do**
    let $c \leftarrow 1$ with probability $1 - q_e$; otherwise $c \leftarrow 0$;
    if $c = 1$, **then** let $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$ and $p \leftarrow p(1 - q_e)$;
    if $c = 0$, **then** let $p \leftarrow pq_e$;
  let $F \leftarrow 1$ with probability $\frac{w_u(H)}{4pp_0}$, where $H = (V_u, \mathcal{E})$;
  $T \leftarrow T - 1$;
until $T < 0$ *or* $F = 1$;
if $F = 1$ then **return** $H = (V_u, \mathcal{E})$; **else return** $\perp$;

The algorithm needs c_0 and c_1 in order to compute the marginal probability of e in Algorithm 2. The quantity c_0 is an estimate to the reliability conditional on e not selected and all choices made so far (namely $E_H \cap E_1 = \mathcal{E}$). Similarly, c_1 is an estimate to the reliability conditional on e selected and $E_H \cap E_1 = \mathcal{E}$. Thus, if $\mathsf{ApproxCount}$ returns exact values of c_0 and c_1, then

$$\Pr_{H=(V_u,E_H)\sim\pi_u}\left[e \in \mathcal{E}_H \mid E_H \cap E_1 = \mathcal{E}\right] = \frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}.$$

However, $\mathsf{ApproxCount}$ can only approximate the reliabilities c_0 and c_1. To handle the error from $\mathsf{ApproxCount}$, our algorithm maintains a number p, which is the probability of selecting the edges in $\mathcal{E}$ and not selecting those in $E_1 \setminus \mathcal{E}$. By the time we reach Algorithm 2, p becomes the probability that H is generated by the algorithm. Then the algorithm uses a filter (with filter probability $\frac{w_u(H)}{4p_0 p}$) to correct the distribution of H. Conditional on passing the filter, H is a perfect sample from π_u. The detailed analysis of the error is given in the full version of this paper.

Before we go to the ApproxCount algorithm, we state one important property of Algorithm 2.

▶ **Lemma 5.** *In Algorithm 2, the following property holds: at the beginning of every while-loop, for any $w \in \partial\Lambda$, $E_1 \cap E_w = \emptyset$, where E_w is the edge set of the graph $G_w = (V_w, E_w)$ defined in* (3)*.*

The above lemma holds because we always pick the smallest vertex in the topological ordering at the beginning of the while-loop. The formal proof is in the full version of this paper.

▶ **Corollary 6.** *In Algorithm 2, in every while-loop, $\Lambda_1 = \Lambda \cup \{w^*\}$.*

Proof. Clearly $\Lambda \cup \{w^*\} \subseteq \Lambda_1$. Suppose there exists $w_0 \neq w^*$ such that $w_0 \in \Lambda_1 \setminus \Lambda$. Then $w_0 \in V_u$ can be reached from w^* through edges in $\mathcal{E}$. The vertex w_0 must be in G_{w^*}, which implies $\mathcal{E} \cap E_{w^*} \neq \emptyset$, and thus $E_1 \cap E_{w^*} \neq \emptyset$. A contradiction to Lemma 5. ◀

3.3 Approximate counting

Our ApproxCount subroutine is used in both Algorithm 1 and Algorithm 2. To suit Algorithm 2, ApproxCount takes input $(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$ and approximates the $\Lambda - t$ reliability $R_\Lambda := \Pr_{\mathcal{G}}[\Lambda \rightsquigarrow_{\mathcal{G}} t]$, where $\mathcal{G}$ is a random spanning subgraph of G obtained by removing each edge e independently with probability q_e, and $\Lambda \rightsquigarrow_{\mathcal{G}} t$ denotes the event that $\exists w \in \Lambda$ s.t. $w \rightsquigarrow_{\mathcal{G}} t$. When called by Algorithm 1, Λ is a single vertex, and when called by Algorithm 2, Λ is the set defined in (7). Moreover, the input satisfies:

- $G = (V, E)$ is a DAG containing the sink t and each edge has a failure probability q_e (for simplicity, we do not write t and all q_e explicitly in the input as they do not change throughout Algorithm 1 and Algorithm 2);
- $\Lambda \subseteq V$ is a subset of vertices that act as sources and $\partial\Lambda := \{w \in V \setminus \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E\}$;
- for any $w \in \partial\Lambda$, $\widetilde{R}_w$ is an approximation of the $w - t$ reliability in G_w and S_w is a set of ℓ approximate random samples from the distribution π_w, where G_w is defined in (3).

All points above hold when Algorithm 3 is evoked by Algorithm 1 or Algorithm 2. The first two points are easy to verify and the last one is verified in the full version of this paper.

The algorithm first rules out the following two trivial cases:

- if $t \in \Lambda$, the algorithm returns 1;
- if Λ cannot reach t in graph G, the algorithm returns 0.[4]

As we are dealing with the more general set-to-vertex reliability, we need some more definitions. Define

$$\Omega_\Lambda := \{H = (V, E_H) \mid E_H \subseteq E \wedge \Lambda \rightsquigarrow_H t\}. \tag{8}$$

For any subgraph $H = (V, E_H)$ of $G = (V, E)$, define the weight function

$$w_\Lambda(H) := \begin{cases} \prod_{e\in E_H}(1 - q_e)\prod_{f\in E\setminus E_H} q_f & \text{if } \Lambda \rightsquigarrow_H t; \\ 0 & \text{if } \Lambda \not\rightsquigarrow_H t. \end{cases} \tag{9}$$

Define the distribution π_Λ, whose support is Ω_Λ, by

$$\pi_\Lambda(H) := \frac{w_\Lambda(H)}{R_\Lambda}, \quad \text{where } R_\Lambda := \sum_{H\in\Omega_\Lambda} w_\Lambda(H). \tag{10}$$

[4] Since all $q_e < 1$, $R_\Lambda > 0$ if and only if $\Lambda \rightsquigarrow_G t$.

Let $\partial\Lambda$ be listed as $\{u_1, \ldots, u_d\}$ for some $d \in [n]$. Note that $d \geq 1$ because $R_\Lambda > 0$ and $t \notin \Lambda$. To estimate R_Λ, we first write Ω_Λ in (8) as a union of d sets. For each $1 \leq i \leq d$, define

$$\Omega_\Lambda^{(i)} := \{H = (V, E_H) \mid (E_H \subseteq E) \wedge (\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in E) \wedge (u_i \rightsquigarrow_H t)\}.$$

▶ **Lemma 7.** *If $t \notin \Lambda$, $\Omega_\Lambda = \cup_{i=1}^d \Omega_\Lambda^{(i)}$.*

Proof. We first show $\cup_{i=1}^d \Omega_\Lambda^{(i)} \subseteq \Omega_\Lambda$. Fix any $H \in \cup_{i=1}^d \Omega_\Lambda^{(i)}$, say $H \in \Omega_\Lambda^{(i^*)}$ ($i^* \in [d]$ may not be unique, in which case we pick an arbitrary one). Then Λ can reach t in H, because we can first move from Λ to u_{i^*} and then move from u_{i^*} to t. This implies $H \in \Omega_\Lambda$. We next show $\Omega_\Lambda \subseteq \cup_{i=1}^d \Omega_\Lambda^{(i)}$. Fix any $H \in \Omega_\Lambda$. There is a path from Λ to t in H. Say the path is $w_1, w_2, \ldots, w_p = t$. Then $w_1 \in \Lambda$ (hence $w_1 \neq t$) and $w_2 = u_{i^*}$ for some $i^* \in [d]$. Hence, H contains the edge (w_1, u_{i^*}) and $u_{i^*} \rightsquigarrow_H t$. This implies $H \in \cup_{i=1}^d \Omega_\Lambda^{(i)}$. ◀

Similar to (10), we define $\pi^{(i)}$,[5] whose support is $\Omega_\Lambda^{(i)}$, by

$$\pi_\Lambda^{(i)}(H) := \frac{w_\Lambda(H)}{R_\Lambda^{(i)}}, \quad \text{where } R_\Lambda^{(i)} := \sum_{H \in \Omega_\Lambda^{(i)}} w_\Lambda(H). \tag{11}$$

In order to perform the Karp-Luby style estimation, we need to be able to do the following three things:

1. compute the value $R_\Lambda^{(i)}$ for each $i \in [d]$;
2. draw samples from $\pi_\Lambda^{(i)}$ for each $i \in [d]$;
3. given any $i \in [d]$ and $H \in \Omega_\Lambda$, determine whether $H \in \Omega_\Lambda^{(i)}$.

Suppose we can do them for now.[6] Consider the following estimator Z_Λ:

1. draw an index $i \in [d]$ such that i is drawn with probability proportional to $R_\Lambda^{(i)}$;
2. draw an sample H from $\pi_\Lambda^{(i)}$;
3. let $Z_\Lambda \in \{0, 1\}$ indicate whether i is the smallest index $j \in [d]$ satisfying $H \in \Omega_\Lambda^{(j)}$.

It is straightforward to see that

$$\begin{aligned}
\mathbb{E}[Z_\Lambda] &= \sum_{i=1}^d \frac{R_\Lambda^{(i)}}{\sum_{j=1}^d R_\Lambda^{(j)}} \sum_{H \in \Omega_\Lambda^{(i)}} \pi_\Lambda^{(i)}(H) \cdot \mathbf{1}\left[H \in \Omega_\Lambda^{(i)} \wedge \left(\forall j < i, H \notin \Omega_\Lambda^{(j)}\right)\right] \\
&= \frac{\sum_{i=1}^d \sum_{H \in \Omega_\Lambda^{(i)}} w_\Lambda(H) \cdot \mathbf{1}\left[H \in \Omega_\Lambda^{(i)} \wedge \left(\forall j < i, H \notin \Omega_\Lambda^{(j)}\right)\right]}{\sum_{j=1}^d R_\Lambda^{(j)}} \\
\text{(by Lemma 7)} \quad &= \frac{R_\Lambda}{\sum_{j=1}^d R_\Lambda^{(j)}} \geq \frac{1}{d} \geq \frac{1}{n},
\end{aligned} \tag{12}$$

where the first inequality holds because each H belongs to at most d different sets. Since Z_Λ is a 0/1 random variable, $\mathrm{Var}(Z_\Lambda) \leq 1$. Hence, we can first estimate the expectation of Z_Λ by repeating the process above and taking the average, and then use $\mathbb{E}[Z_\Lambda] \sum_{j=1}^d R_\Lambda^{(j)}$ as the estimator $\widetilde{R}_\Lambda$ for R_Λ. However, in the input of ApproxCount, we only have estimates $\widetilde{R}_{u_i}$ and a limited set of samples S_{u_i} for each $u_i \in \partial\Lambda$. Our algorithm will need to handle these imperfections.

[5] One may notice that if $\Omega_\Lambda^{(i)} = \emptyset$, then $\pi_\Lambda^{(i)}$ is not well-defined. In the full version of this paper, we prove that $\Omega_\Lambda^{(i)} = \emptyset$ never happens.

[6] We will do (1) and (2) approximately rather than exactly. This will incur some error that will be controlled later.

The third step of implementing the estimator Z_Λ is straightforward by a BFS. For the first step, $R_\Lambda^{(i)}$ can be easily computed given R_{u_i}, and we will use the estimates $\widetilde{R}_{u_i}$ instead. For the second step, define

$$\delta_\Lambda(u_i) := \{(w, u_i) \in E \mid w \in \Lambda\}.$$

To sample from $\pi_\Lambda^{(i)}$, we shall sample at least one edge in $\delta_\Lambda(u_i)$, sample a subgraph H from π_{u_i}, and add all other edges independently. These are summarized by the next lemma.

▶ **Lemma 8.** *For any $i \in [d]$, it holds that $R_\Lambda^{(i)} = \left(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}\right) R_{u_i}$ and a random sample $H' = (V, E_{H'}) \sim \pi_\Lambda^{(i)}$ can be generated by the following procedure:*

- *sample $H = (V_{u_i}, E_H) \sim \pi_{u_i}$;*
- *let $E_{H'} = E_H \cup D$, where $D \subseteq \delta_\Lambda(u_i)$ is a random subset with probability proportional to*

$$\mathbf{1}[D \neq \emptyset] \cdot \prod_{e \in \delta_\Lambda(u_i) \cap D} (1 - q_e) \prod_{f \in \delta_\Lambda(u_i) \setminus D} q_f; \tag{13}$$

- *add each $e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))$ into $E_{H'}$ independently with probability $1 - q_e$.*

All three steps above handle mutually exclusive edge sets and thus are mutually independent.

Proof. If each edge e in G is removed independently with probability q_e, we have a random spanning subgraph $\mathcal{G} = (V, \mathcal{E})$. By the definition of $R_\Lambda^{(i)}$,

$$\begin{aligned} R_\Lambda^{(i)} &= \Pr\left[\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E} \wedge u_i \rightsquigarrow_{\mathcal{G}} t\right] \\ &= \Pr\left[\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}\right] \cdot \Pr\left[u_i \rightsquigarrow_{\mathcal{G}} t \mid \exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}\right]. \end{aligned}$$

It is easy to see $\Pr[\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}] = 1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}$. For the second conditional probability, note that the event $u_i \rightsquigarrow_{\mathcal{G}} t$ depends only on the randomness of edges in graph G_{u_i}. In other words, for any edges $e \in E \setminus E_{u_i}$, whether or not e is removed has no effect on the $u_i - t$ reachability. Due to acyclicity, all edges in $\delta_\Lambda(u_i)$ are not in the graph G_{u_i}. We have

$$R_\Lambda^{(i)} = \left(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}\right) R_{u_i}.$$

By definition (11), $\pi_\Lambda^{(i)}$ is the distribution of $\mathcal{G} = (V, \mathcal{E})$ conditional on $\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}$ and $u_i \rightsquigarrow_{\mathcal{G}} t$. For any graph $H' = (V, E_{H'})$ satisfying $\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in E_{H'}$ and $u_i \rightsquigarrow_{H'} t$, we have

$$\begin{aligned} \pi_\Lambda^{(i)}(H') &= \Pr\left[\mathcal{G} = H' \mid \exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E} \wedge u_i \rightsquigarrow_{\mathcal{G}} t\right] \\ &= \frac{\Pr[\mathcal{G} = H']}{R_\Lambda^{(i)}} = \frac{\Pr[\mathcal{G} = H']}{(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}) R_{u_i}} \\ &= \prod_{e \in E_{H'} : e \in E \setminus E_{u_i}} (1 - q_e) \prod_{f \notin E_{H'} : f \in E \setminus E_{u_i}} q_f \cdot \frac{w_{u_i}(H'[V_{u_i}])}{(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}) R_{u_i}} \\ &= \pi_{u_i}(H'[V_{u_i}]) \cdot \frac{\prod_{e \in \delta_\Lambda(u_i) \cap E_{H'}} (1 - q_e) \prod_{f \in \delta_\Lambda(u_i) \setminus E_{H'}} q_f}{1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}} \\ &\quad \cdot \prod_{\substack{e \in E_{H'} : \\ e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))}} (1 - q_e) \prod_{\substack{f \notin E_{H'} : \\ f \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))}} q_f. \end{aligned}$$

The probability above exactly matches the procedure in the lemma. ◀

The first sampling step in Lemma 8 can be done by directly using the samples from S_{u_i}. We still need to show that the second step in Lemma 8 can be done efficiently. The proof of Lemma 9 is in the full version of this paper.

▶ **Lemma 9.** *There is an algorithm such that given a set $S = \{1, 2, \dots, n\}$ and n numbers $0 \le q_1, q_2, \dots, q_n < 1$, it return a random non-empty subset $D \subseteq S$ with probability proportional to $\mathbf{1}[D \neq \emptyset] \prod_{i \in D}(1 - q_i) \prod_{j \in S \setminus D} q_j$ in time $O(n)$.*

Now, we are almost ready to describe ApproxCount (Algorithm 3). For any u_i, we have an approximate value $\widetilde{R}_{u_i}$ of R_{u_i} and we also have a set S_{u_i} of ℓ approximate samples from the distribution π_{u_i}. By Lemma 8 and Lemma 9, we can efficiently approximate $R^{(i)}_\Lambda$ and generate approximate samples from $\pi^{(i)}_\Lambda$. Hence, we can simulate the process described below Lemma 7 to estimate R_Λ.

However, to save the number of samples, there is a further complication. Our algorithm estimates the expectation of $\mathbb{E}[Z_\Lambda]$ in two rounds and then takes the median of estimators. Recall (6). We further divide ℓ by introducing the following parameters:

$$\begin{aligned} &\ell = B\ell_0, \quad B := 60n + 150m, \quad \ell_0 := \ell_1 + 500\ell_2, \\ &\ell_1 := 400n, \quad \ell_2 := \lceil 10^4 n^2 \max\{m^2, \epsilon^{-2}\} \rceil. \end{aligned} \tag{14}$$

Then we do the following.

- For any $u_i \in \partial\Lambda$, we divide all ℓ samples in S_{u_i} into B blocks, each containing ℓ_0 samples. Denote the B blocks by $S^{(1)}_{u_i}, S^{(2)}_{u_i}, \dots, S^{(B)}_{u_i}$. Each block here is used for one estimatior.
- For each $i \in [d]$ and $j \in [B]$, we further partition $S^{(j)}_{u_i}$ into two multi-sets $S^{(j,1)}_{u_i}$ and $S^{(j,2)}_{u_i}$, where $S^{(j,1)}_{u_i}$ has ℓ_1 samples and $S^{(j,2)}_{u_i}$ has $500\ell_2$ samples, used for the two rounds, respectively.
- For each $j \in [B]$, we do the following two round estimation:
  1. use samples in $(S^{(j,1)}_{u_i})_{i \in [d]}$ to obtain a *constant-error* estimation $\widehat{Z}^{(j)}_\Lambda$ of $\mathbb{E}[Z_\Lambda]$;
  2. use $\widehat{Z}^{(j)}_\Lambda$ and samples in $(S^{(j,2)}_{u_i})_{i \in [d]}$ to obtain a more accurate *estimation* $\widetilde{Z}^{(j)}_\Lambda$ of $\mathbb{E}[Z_\Lambda]$;
  3. let $Q^{(j)}_\Lambda \leftarrow \widetilde{Z}^{(j)}_\Lambda \sum_{i=1}^{d} \widetilde{R}^{(i)}_\Lambda$, where $\widetilde{R}^{(i)}_\Lambda := \big(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}\big)\widetilde{R}_{u_i}$ for each $i \in [d]$.
- Return the median number $\widetilde{R}_\Lambda := \text{median}\left\{Q^{(1)}_\Lambda, Q^{(2)}_\Lambda, \dots, Q^{(B)}_\Lambda\right\}$.

A detailed description of ApproxCount is given in Algorithm 3. It uses a subroutine Estimate, which generates the Karp-Luby style estimator and is described in Algorithm 4.

Each time Algorithm 3 finishes, its input (V, E, Λ) and output $\widetilde{R}_\Lambda$ are stored in the memory. If Algorithm 3 is ever evoked again with the same (V, E, Λ), we simply return $\widetilde{R}_\Lambda$ from the memory.

Algorithm 3 first obtains the constant-error estimation $\widehat{Z}^{(j)}_\Lambda$ in Algorithm 3. Next, it puts $\widehat{Z}^{(j)}_\Lambda$ into the parameters and run the subroutine Estimate again to get a more accurate estimation $\widetilde{Z}^{(j)}_\Lambda$. The benefit of this two-round estimation is that we can save the number of samples maintained for each vertex. It costs only a small number of samples to get the crude estiamtion, but the crude estimation carries information of the ratio $\frac{\sum_{t \in [d]} R^{(t)}_\Lambda}{R_\Lambda}$, which allows us to fine tune the number of samples required per vertex for the good estimation. To be more specific, in the second call of the subroutine Estimate, the number of overall samples, namely the parameter T which depends on $\widehat{Z}^{(j)}_\Lambda$, can still be as large as $\Omega(\ell_2 n)$ in the worst case, and yet each $S^{(j,2)}_{u_i}$ has only $O(\ell_2)$ samples in the block. In the analysis, we will show that this many samples per vertex suffice with high probability and Algorithm 4 of Algorithm 4 (the failure case) is executed with low probability. The reason is that, roughly

Algorithm 3 $\mathsf{ApproxCount}\Big(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda}\Big)$.

Input: a graph $G = (V, E)$, a subset $\Lambda \subseteq V$, all $(\widetilde{R}_w, S_w)$ for $w \in \partial\Lambda$, where $\partial\Lambda = \{w \in V \setminus \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E\}$;
Output: an estimator $\widetilde{R}_\Lambda$ of R_Λ
if $t \in \Lambda$, **then return** 0; **if** Λ cannot reach t in G, **then return** 1;
for $u_i \in \partial\Lambda$ **do**
  $\widetilde{R}^{(i)}_\Lambda \leftarrow \big(1 - \prod_{u\in\delta_\Lambda(u_i)} q_{(u,u_i)}\big)\widetilde{R}_{u_i}$;
  partition S_{u_i} (arbitrarily) into B multi-sets, denoted by $S^{(j)}_{u_i}$ for $j \in [B]$, where $B = 60n + 150m$ and each $S^{(j)}_{u_i}$ has $\ell_0 = \ell_1 + 500\ell_2$ samples;
  for each $j \in [B]$, partition $S^{(j)}_{u_i}$ further into two multi-sets $S^{(j,1)}_{u_i}$ and $S^{(j,2)}_{u_i}$, where $|S^{(j,1)}_{u_i}| = \ell_1$ and $|S^{(j,2)}_{u_i}| = 500\ell_2$;
for j *from* 1 *to* B **do**
  $\widehat{Z}^{(j)}_\Lambda \leftarrow \mathsf{Estimate}\Big((S^{(j,1)}_{u_i})_{i\in[d]}, \ell_1, \ell_1\Big)$;
  $\widetilde{Z}^{(j)}_\Lambda \leftarrow \mathsf{Estimate}\Big((S^{(j,2)}_{u_i})_{i\in[d]}, 500\ell_2, 25\ell_2 \cdot \min\{2/\widehat{Z}^{(j)}_\Lambda, 4n\}\Big)$;
  $Q^{(j)}_\Lambda \leftarrow \widetilde{Z}^{(j)}_\Lambda \sum_{i=1}^{d} \widetilde{R}^{(i)}_\Lambda$;
return $\widetilde{R}_\Lambda := \text{median}\Big\{Q^{(1)}_\Lambda, Q^{(2)}_\Lambda, \ldots, Q^{(B)}_\Lambda\Big\}$;

speaking, large T means large overlap among $\Omega^{(t)}_\Lambda$'s, and the chance of hitting each vertex is roughly the same, resulting in the number of samples required per vertex close to the average. Conversely, small T means little overlap, and some vertex or vertices may be sampled much more often than other vertices, but in this case the overall number of samples required, namely T, is small anyways. To summarize, with this two-round procedure, $O(\ell)$ overall samples per vertex are enough to obtain an estimation with the desired accuracy and high probability.

4 Sketch of analysis

In this section, we give a sketch of the analysis of our algorithm. The complete analysis is in the full version of this paper. Roughly speaking, what we need to show is that in the induction step, accurate samples (in terms of the TV distance) imply accurate estimates, and accurate estimates imply accurate samples.

First consider $\mathsf{Sample}\Big(v_k, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\ldots,v_n\}}, \widetilde{R}_{v_k}\Big)$ as being called by Algorithm 1. Let $u := v_k$, and the subroutine runs on the graph $G_u = (V_u, E_u)$. We show that in $\mathsf{Sample}(\cdot)$, if every value returned by the $\mathsf{ApproxCount}(\cdot)$ subroutine is accurate, then $\mathsf{Sample}(\cdot)$ also returns accurate samples. Formally, we assume access to an oracle $\mathcal{P}$ satisfying:

- given $u \in V_u$, $\mathcal{P}$ returns p_0 such that $1 - \frac{1}{10m} \le \frac{p_0}{R(V_u,E_u,\{u\})} \le 1 + \frac{1}{10m}$;
- given any $E_2 \subseteq E_u$ and $\Lambda, \Lambda_1 \subseteq V$ in Algorithm 2 and Algorithm 2 of Algorithm 2, $\mathcal{P}$ returns $c_0(V_u, E_2, \Lambda)$ and $c_1(V_u, E_2, \Lambda_1)$ such that $1 - \frac{1}{10m} \le \frac{c_0(V_u,E_2,\Lambda)}{R(V_u,E_2,\Lambda)} \le 1 + \frac{1}{10m}$, and $1 - \frac{1}{10m} \le \frac{c_1(V_u,E_2,\Lambda_1)}{R(V_u,E_2,\Lambda_1)} \le 1 + \frac{1}{10m}$.

Here, we use the convention $\frac{0}{0} = 1$ and $\frac{x}{0} = \infty$ for $x > 0$. For any V, E and $U \subseteq V_u$, $R(V, E, U)$ is the U-t reliability in the graph (V, E). The numbers p_0, c_0 and c_1 returned by $\mathcal{P}$ can be random variables, but we assume that the inequalities above are always satisfied.

Algorithm 4 Estimate$\left((S^{es}_{u_i})_{i\in[d]}, \ell_{es}, T\right)$.

Input: a set of samples $S^{es}_{u_i}$ for each $i \in [d]$, where $|S^{es}_{u_i}| = \ell_{es}$, a threshold T
Output: an estimator Z_{es} of $\mathbb{E}[Z_\Lambda]$

for each $i \in [d]$, let $c_i = 0$;
for k *from* 1 *to* T **do**
  draw an index $i \in [d]$ such that i is drawn with probability proportional to $\widetilde{R}^{(i)}_\Lambda$;
  $c_i \leftarrow c_i + 1$;
  if $c_i > \ell_{es}$ **then return** 0;
  let $H = (V_{u_i}, E_H)$ be the c_i-th sample from $S^{(j)}_{u_i}$;
  do the following transformation on H to get $H' = (V, E_{H'})$;
  - let $E_{H'} \leftarrow E_H$;
  - draw $D \subseteq \delta_\Lambda(u_i)$ with probability proportional to (13), and let $E_{H'} \leftarrow E_{H'} \cup D$;
  - for each $e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))$, add e into $E_{H'}$ independently with probability $1 - q_e$;

  let $Z^{(k)}_{es} \in \{0, 1\}$ indicate whether i is the smallest index $t \in [d]$ satisfying $H' \in \Omega^{(t)}_\Lambda$;
return $Z_{es} := \frac{1}{T}\sum_{k=1}^{T} Z^{(k)}_{es}$;

Consider the following modified sampling algorithm, in which we replace Algorithm 2, Algorithm 2 and Algorithm 2 of Algorithm 2 by calling the oracle $\mathcal{P}$. This modified algorithm has some nice properties, summarized in the next lemma.

▶ **Lemma 10.** *Given any $u = v_k \in V$, the with probability at least $1 - (\varepsilon/n)^{200}$, the modified sampling algorithm does not crash and returns a perfect independent sample from the distribution π_u. The running time is $\widetilde{O}(N(|E_u| + |V_u|))$, where N is the time cost for one oracle call and $\widetilde{O}$ hides polylog(n/ε) factors.*

In the full analysis, we need to handle the real sampling algorithm, the analysis of which also relies on the analysis of ApproxCount. ApproxCount behaves like $\mathcal{P}$ but only with high probability. To analyze the error bounds, we also need to identify several random sources and handle errors from different sources separately.

Next consider ApproxCount$(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$, where $G = (V, E)$ is a DAG and $t \notin \Lambda$. Recall that for any $w \in V$, the graph $G_w = G[V_w]$, where V_w contains all vertices v satisfying $w \rightsquigarrow_G v$ and $v \rightsquigarrow_G t$. Let R_w be the $w - t$ reliability in G_w. Recall that ℓ_2 and B are parameters in ApproxCount, Algorithm 3. The next lemma shows that if in the input, all $\widetilde{R}_w$ are accurate estimators of R_w and all S_w are accurate samples, then ApproxCount$(\cdot)$ outputs an accurate estimator to R_Λ. Let S^{ideal}_w be a multi-set of ℓ *independent* and *prefect* samples from π_w.

▶ **Lemma 11.** *Suppose the following conditions are satisfied*
- *for any $w \in \partial\Lambda$, $1 - \varepsilon_0 \le \frac{\widetilde{R}_w}{R_w} \le 1 + \varepsilon_0$ for some $\varepsilon_0 < 1/2$;*
- *$d_{TV}\left((S_w)_{w\in\partial\Lambda}, (S^{\text{ideal}}_w)_{w\in\partial\Lambda}\right) \le \delta_0$.*

Then with probability at least $1 - \delta_0 - 2^{-B/30}$, it holds that

$$1 - \varepsilon_0 - \frac{2}{\sqrt{\ell_2}} \le \frac{\widetilde{R}_\Lambda}{R_\Lambda} \le 1 + \varepsilon_0 + \frac{2}{\sqrt{\ell_2}}. \tag{15}$$

The running time of ApproxCount *is $O(n\ell(|V| + |E|))$.*

As mentioned before, due to reusing samples, there is correlation among $\widetilde{R}_w$ and S_w for different w's. However, the conditions of Lemma 11 do not rely on correlation, which is key to the correctness of the algorithm.

We note that the failure probability in Lemma 11 is exponentially small (we plug the bound in (16) as δ_0). This is necessary because we eventually need to apply a union bound over the exponential possibilities of spanning subgraphs.

Theorem 1 is proved by combining Lemma 10 and Lemma 11. We use an induction from $v_n = t$ to $v_1 = s$ to show that the following events occur with high probability:

- for any $j \geq i$, let $S_{v_j}^{\text{ideal}}$ be a multi-set of ℓ independent perfect samples from π_{v_j}, it holds that

$$d_{\text{TV}}\left((S_{v_j})_{j\geq i}, (S_{v_j}^{\text{ideal}})_{j\geq i}\right) \leq 2^{-4m}(2^{n-i}-1); \tag{16}$$

- the estimator $\widetilde{R}_{v_i}$ approximates the $v_i - t$ reliability R_{v_i}:

$$1 - \frac{n-i}{50n\max\{m,\varepsilon^{-1}\}} \leq \frac{\widetilde{R}_{v_i}}{R_{v_i}} \leq 1 + \frac{n-i}{50n\max\{m,\varepsilon^{-1}\}}. \tag{17}$$

The base case of $i = n$ is trivial. In the induction step, by using the I.H., we can show that the condition in Lemma 11 is satisfied and thus $\mathsf{ApproxCount}(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$ returns an accurate estimator of R_Λ. Hence, the subroutine $\mathsf{ApproxCount}(\cdot)$ behaves similarly like the oracle $\mathcal{P}$, which, by Lemma 10, implies that the $\mathsf{Sample}(\cdot)$ algorithm also generates accurate samples.

Here, we briefly explain why we set the parameter $\ell_2 = O(n^2\max\{m^2, \varepsilon^{-2}\})$, which controls the number of samples maintained (as $\ell = \Theta(B\ell_2)$, recall (14)) and thus the overall running-time of the algorithm. We want to guarantee that $\widetilde{R}_\Lambda$ returned by $\mathsf{ApproxCount}(\cdot)$ achieves the same accuracy as that returned by $\mathcal{P}$. Note that the relative error in (17) serves as ε_0 in (15). Therefore combining (15) and (17), we need

$$\frac{n-i}{50n\max\{m,\varepsilon^{-1}\}} + \frac{2}{\sqrt{\ell_2}} \leq \frac{1}{50\max\{m,\varepsilon^{-1}\}} + \frac{2}{\sqrt{\ell_2}} \leq \frac{1}{10m}. \tag{18}$$

To inductively prove (17), we also need

$$\frac{n-(i+1)}{50n\max\{m,\varepsilon^{-1}\}} + \frac{2}{\sqrt{\ell_2}} \leq \frac{n-i}{50n\max\{m,\varepsilon^{-1}\}}. \tag{19}$$

The condition in (18) forces us to assume a strong induction hypothesis such that for each v_i, $\widetilde{R}_{v_i}$ has $1 \pm O(\frac{1}{m})$ approximation error. And the condition in (19) is the bottleneck for setting parameter ℓ_2.

In the full analysis, we use a more complicated induction hypothesis in the induction proof. Moreover, we need to define certain bad and good events to carefully analyze where errors come from and how they accumulate in every step.

The approximation guarantee in Theorem 1 follows directly from (17). Note that this guarantee is always at least a $(1 \pm 1/m)$-approximation and is stronger than a $(1 \pm \varepsilon)$-approximation when $\varepsilon > 1/m$. For the running time, recall that the number of samples per vertex is $\ell = O\big(n^2 m\max\{m^2, \varepsilon^{-2}\}\big)$. The running time of $\mathsf{ApproxCount}$ (Algorithm 3) is at most

$$T_{\mathsf{count}} = O(mn\ell) = O\big(n^3m^2\max\{m^2, \varepsilon^{-2}\}\big).$$

The running time of Sample (Algorithm 2) is at most

$$T_{\mathsf{sample}} = \widetilde{O}((n+m)T_{\mathsf{count}}) = \widetilde{O}(mT_{\mathsf{count}}) = \widetilde{O}\big(n^3m^3\max\{m^2,\varepsilon^{-2}\}\big).$$

Hence, the running time of Algorithm 1 is

$$T = O(n(T_{\mathsf{count}} + \ell T_{\mathsf{sample}})) = O(n\ell T_{\mathsf{sample}}) = \widetilde{O}\big(n^6m^4\max\{m^4,\varepsilon^{-4}\}\big).$$

5 #BIS-hardness for $s-t$ unreliability

In this section we show Theorem 2. We first reduce #**BIS** to $s-t$ unreliability where each vertex (other than s and t) fails with $1/2$ probability independently. Note that in this version of the problem no edge would fail. Given a DAG $D=(V\cup\{s,t\},A)$, this is equivalent to counting the number of subsets $S\subseteq V$ such that in the induced subgraph $D[S\cup\{s,t\}]$, s cannot reach t. We call S a $s\not\rightsquigarrow t$ set.

Given a bipartite graph $G=(V,E)$, let its two partitions be L and R. We add two special vertices s and t, and connect, with directed edges, s to all vertices in L and all vertices in R to t. Lastly, for any edge $\{u,v\}\in E$, where $u\in L$ and $v\in R$, we replace it with a directed edge (u,v). Call the new directed graph D_G. Clearly it is a DAG.

For any independent set I in G, we claim that in $D_G[I\cup\{s,t\}]$, s cannot reach t. This is because for any $e\in E$, there is at least one vertex unoccupied. Thus s cannot reach t using the directed version of e, and this holds for any $e\in E$.

In the other direction, let S be a $s\not\rightsquigarrow t$ subset of V. This means that for any edge $\{u,v\}\in E$, either $u\notin S$ or $v\notin S$, as otherwise $s\to u\to v\to t$. This means that S is an independent set of G.

Thus, there is a one-to-one correspondence between independent sets of G and $s\not\rightsquigarrow t$ subsets of V. Namely, $s-t$ unreliability where vertices (other than s and t) fail with $1/2$ probability is #**BIS**-hard.

Next, we reduce $s-t$ unreliability from the vertex version. For this, we can replace each vertex v (other than s and t) by two vertices v,v' and a directed edge $v\to v'$. All incoming edges of v still goes into v, and all outgoing edges of v goes out from v'. Assign to the new edges the failure probabilities of their corresponding vertices, and assign failure probability 0 to all original edges. Clearly the unreliability is the same with these changes. To make failure probabilities uniform, we can replace edges with failure probability 0 by k parallel edges. Effectively, the connection fails only if all the parallel edges fail at the same time. If the failure probability of each edge is q, the probability of all parallel edges failing is q^k. As this probability approaches 0 exponentially fast, it is easy to set a polynomially bounded k so that the new unreliability is a sufficiently good approximation of the original.

As a side note, the last reduction also works for reliability. Thus Theorem 1 also works for $s-t$ reliability in DAGs where vertices rather than edges fail independently.

References

1 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993.

2 Antoine Amarilli, Timothy van Bremen, and Kuldeep S. Meel. Conjunctive queries on probabilistic graphs: the limits of approximability. *arXiv*, abs/2309.13287, 2023. `arXiv:2309.13287`.

3 Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: high-dimensional walks and an FPRAS for counting bases of a matroid. In *STOC*, pages 1–12. ACM, 2019.

4 Nima Anari, Kuikui Liu, Shayan Oveis Gharan, Cynthia Vinzant, and Thuy-Duong Vuong. Log-concave polynomials IV: approximate exchange, tight mixing times, and near-optimal sampling of forests. In *STOC*, pages 408–420. ACM, 2021.

5 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, 68(6):48:1–48:40, 2021.

6 Michael O. Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980.

7 Michael O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Trans. Rel.*, 35(3):230–239, 1986.

8 Michael O. Ball and J. Scott Provan. Calculating bounds on reachability and connectedness in stochastic networks. *Networks*, 13(2):253–278, 1983.

9 Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Beyond the quadratic time barrier for network unreliability. *arXiv*, abs/2304.06552, 2023. `arXiv:2304.06552`.

10 Xiaoyu Chen, Heng Guo, Xinyuan Zhang, and Zongrui Zou. Near-linear time samplers for matroid independent sets with applications. *arXiv*, abs/2308.09683, 2023. `arXiv:2308.09683`.

11 Charles J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987.

12 Mary Cryan, Heng Guo, and Giorgos Mousa. Modified log-Sobolev inequalities for strongly log-concave distributions. *Ann. Probab.*, 49(1):506–525, 2021.

13 Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

14 Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Inf. Comput.*, 134(1):59–74, 1997.

15 Heng Guo and Kun He. Tight bounds for popping algorithms. *Random Struct. Algorithms*, 57(2):371–392, 2020.

16 Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM J. Comput.*, 48(3):964–978, 2019.

17 Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *J. ACM*, 66(3):18:1–18:31, 2019.

18 David G. Harris and Aravind Srinivasan. Improved bounds and algorithms for graph cuts and network reliability. *Random Struct. Algorithms*, 52(1):74–135, 2018.

19 Mark Jerrum. *On the complexity of evaluating multivariate polynomials*. PhD thesis, The University of Edinburgh, 1981.

20 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

21 David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, 29(2):492–514, 1999.

22 David R. Karger. A phase transition and a quadratic time unbiased estimator for network reliability. In *STOC*, pages 485–495. ACM, 2020.

23 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64. IEEE Computer Society, 1983.

24 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *J. Complex.*, 1(1):45–64, 1985.

25 Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

26 Kuldeep S. Meel, Sourav Chakraborty, and Umang Mathur. A faster FPRAS for #NFA. *arXiv*, abs/2312.13320, 2023. `arXiv:2312.13320`.

27 J. Scott Provan. The complexity of reliability computations in planar and acyclic graphs. *SIAM J. Comput.*, 15(3):694–702, 1986.

28 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

29 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

30 Rico Zenklusen and Marco Laumanns. High-confidence estimation of small s-t reliabilities in directed acyclic networks. *Networks*, 57(4):376–388, 2011.