AN FPRAS FOR TWO TERMINAL RELIABILITY IN DIRECTED ACYCLIC GRAPHS

WEIMING FENG AND HENG GUO

Abstract. We give a fully polynomial-time randomized approximation scheme (FPRAS) for two terminal reliability in directed acyclic graphs.

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 [Val79] introduced the counting complexity class $\#\mathbf{P}$. 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 $\#\mathbf{P}$-complete [Bal80, Jer81, BP83, PB83, Bal86, Col87].

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 [Kar99] give the first fully polynomial-time randomized approximation scheme (FPRAS) for all-terminal *unreliability*, while about two decades later, Guo and Jerrum [GJ19] gives the first FPRAS for all-terminal *reliability*. The latter algorithm is under the partial rejection sampling framework [GJL19], and the Markov chain Monte Carlo (MCMC) method is also shown to be efficient shortly after [ALOV19, CGM21]. See [HS18, Kar20], [GH20], and [ALO$^+$21, CGZZ23] 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 #**P**-complete [PB83, Sec 3], even restricted to planar DAGs where the vertex degrees are at most 3 [Pro86, Theorem 3]. Our result positively resolves an open problem by Zenklusen and Laumanns [ZL11]. Without loss of generality, in the theorem below we assume that any vertex other than s has at least one incoming edge, and thus $|E| \geqslant |V| - 1$ for the input $G = (V, E)$.

(Weiming Feng) Simons Institute for the Theory of Computing, UC Berkeley, 121 Calvin Lab #2190, Berkeley, CA 94720-2190, United States.

(Heng Guo) School of Informatics, University of Edinburgh, Informatics Forum, Edinburgh, EH8 9AB, United Kingdom.

E-mail address: fwm1994@gmail.com, hguo@inf.ed.ac.uk.

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 $\widetilde{O}(n^8 m^8/\varepsilon^4)$ where $n = |V|$, $m = |E|$, and $\widetilde{O}$ hides polylog(n/ϵ) factors.*

The running time can be improved when the input error bound ε is small.

Theorem 2. *If $\varepsilon = O(\frac{1}{m})$, then the running time of the algorithm in Theorem 1 can be improved to $\widetilde{O}(n^8 m^4/\varepsilon^4)$.*

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 [KL83, KLM89] 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 [KL85, ZL11]. Our main observation is to use the Karp-Luby method as a subroutine in dynamic programming using the structure of DAGs.

To be more specific, 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 $i = 1$. At each step, we rewrite the set of subgraphs in which u can reach t as a union of sets, in each of which one out-neighbour u_i of u can reach t. However, to use the Karp-Luby method, we need to be able to sample from each such set. We thus, in addition, also maintain a set S_{v_i} from $i = n$ to $i = 1$, where for any $u \in V$, S_u is a multi-set of subgraph samples where u can reach t. Using these samples, the Karp-Luby method can estimate R_u, but we also need to efficiently generate these samples for u. This last sampling step is via a self-reduction similar to the Jerrum-Valiant-Vazirani sampling to counting reduction [JVV86]. This self-reduction makes use of R_u's, but as the reduction goes, we may also need to measure the reliability from any vertex in a subset to t. As there are exponential many subsets of vertices, we cannot store all these quantities. Instead we estimate these quantities adaptively, since in each run of the algorithm only a polynomial number of them can ever be encountered. The main difficulty here is to carefully control the errors accumulated throughout the process while preventing the running time from becoming too large.

Our technique is inspired by an FPRAS for the number of accepting strings of non-deterministic finite automatas (#NFA), found by Arenas, Croquevielle, Jayaram, and Riveros [ACJR21]. 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. At the beginning of their algorithm, there are a few steps to 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, Amarilli, van Bremen, and Meel [AvBM23] 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 [ACJR21]. 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** [ÀJ93], where #NFA is **SpanL**-complete under polynomial-time parsimonious reductions. In particular, every problem in **SpanL** admits an FPRAS because #NFA admits one [ACJR21], which implies that $s - t$ reliability in DAGs admits an FPRAS if $q_e = 1/2$ for all edges. The explicit reduction in [AvBM23] 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

[1]The running time of the algorithm in [ACJR21] is not explicitly given. This bound is obtained by going through their proof.

NFA with $O(m^2)$ states.[2] As a consequence, their algorithm has a running time of $O\left(m^{51}\varepsilon^{-17}\right)$. 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 $0 \leqslant q_e < 1$ in a unified way. In any case, an algorithm via reductions is much slower than the direct algorithm in Theorem 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.

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 $0 \leqslant q_e < 1$. (Any edge with $q_e = 1$ can be simply removed.) 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 $\mathbf{q} = (q_e)_{e \in E}$. The $s - t$ reliability problem is to compute

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

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 $\pi_{G,s,t,\mathbf{q}}$ satisfying

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

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 have at least one in-neighbour, and thus $m \geqslant n - 1$. 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 edges 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

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

[2] In fact, [AvBM23] first reduces the reliability instance to an nOBDD (non-deterministic ordered binary decision diagrams) 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, they do not give an explicit reduction for this particular problem. We provide a direct (and essentially the same) reduction in Appendix B.

Define the distribution π_u by

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

where the partition function

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

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 $R_{G,\mathbf{q}}(s,t)$ and the distribution $\pi_{G,s,t,\mathbf{q}}$ 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

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

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

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*

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

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. The main analysis is in Section 4.

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, $v_1, \dots, v_n$. It must hold that $v_1 = s$ and $v_n = t$. 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 $\Gamma_{out}(u) := \{w \mid (u,w) \in E\}$ denote the set of out-neighbours of u. Define a parameter

$$\ell := (60n + 150m)\left\lceil \frac{10^5 n^3 m^2}{\varepsilon^2} \right\rceil. \tag{6}$$

to 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. Our algorithm is outlined in Algorithm 1.

The base case (Line 1) of $v_n = t$ is trivial. The subroutine 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 ApproxCount$(V, E, u, (\widetilde{R}_w, S_w)_{\Gamma_{out}(u)})$ takes a graph $G = (V, E)$, a vertex u, and $(\widetilde{R}_w, S_w)$ for all $w \in \Gamma_{out}(u)$ as the input, and it outputs an approximation of the $v - t$ reliability in the graph G. We describe Sample in Section 3.2 and (a slightly more general version of) ApproxCount in Section 3.3.

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

Input: a DAG $G = (V, E)$, a vector $\mathbf{q} = (q_e)_{e \in E}$, 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 ℓ independent samples from π_t;
for k *from* $n-1$ *to* 1 **do**
  $\widetilde{R}_{v_k} \leftarrow \mathsf{ApproxCount}\Big(V_{v_k}, E_{v_k}, v_k, (\widetilde{R}_w, S_w)_{w \in \Gamma_{\mathrm{out}}(u)}\Big)$;
  $S_{v_k} \leftarrow \emptyset$;
  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 $\widetilde{R}_s$.

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 [JVV86]. 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;

Given any $\mathcal{E}$, we can uniquely define the following subset of vertices

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

In other words, let $G' = (V_u, \mathcal{E})$ and Λ is the set of vertices such 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 here we use a more general version of ApproxCount, described in Section 3.3. This is equivalent to contracting all vertices in Λ into a single vertex u, and then calculate the $u - t$ reliability in the resulting graph. Finally, let $E_2 := E_u \setminus E_1$ be the set of edges that have not been scanned yet by the algorithm. Sample is described in Algorithm 2.

Remark 4 (Crash of Sample). The subroutine Sample (Algorithm 2) may crash in following cases: (1) in Line 7, $\partial\Lambda = \emptyset$; (2) in Line 13, $q_e c_0 + (1 - q_e) c_1 = 0$; (3) in Line 20, $\frac{w_u(H)}{4 p_0 p} > 1$; (4) in Line 22, $F = 0$. If it crashes, we stop Algorithm 1 immediately and output $\widetilde{R}_s = 0$.

Algorithm 2 uses the subroutine $\mathsf{ApproxCount}(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w \in V_u \setminus \Lambda})$, which approximates the $\Lambda - t$ reliability in the input $G = (V, E)$. This reliability is the probability such that there exists one vertex in Λ being able to reach t if each edge $e \in E$ fails independently with probability q_e. Compared to ApproxCount in Algorithm 1, we generalise the reliability from one source s to a set Λ of sources. The general ApproxCount subroutine is given in Section 3.3.

Let $\mathcal{E}$ and E_1 be the set of edges maintained in the algorithm. If one assumes ApproxCount returns the exact reliability c_0, c_1, then we have

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

That is, the algorithm computes the marginal distribution of e induced from π_u conditional on $\mathcal{E} \subseteq E_1$ selected into the random subgraph H and $E_1 \setminus \mathcal{E}$ not selected. However, ApproxCount can only approximate the reliabilities c_0 and c_1. To handle the error from 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: Sample$\Big(u, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\dots,v_n\}}, \widetilde{R}_{v_k}\Big)$

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

1 $T \leftarrow \lceil 1000 \log \frac{n}{\varepsilon} \rceil$ and $F \leftarrow 0$;
2 $p_0 \leftarrow \widetilde{R}_u$;
3 **repeat**
4 let $p \leftarrow 1$;
5 let $E_1 \leftarrow \emptyset$, $E_2 \leftarrow E_u \setminus E_1$ and $\Lambda = \{u\}$;
6 **while** $t \notin \Lambda$ **do**
7 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 $w' \in \Lambda$;
8 let $\Lambda_1 \leftarrow \mathsf{rch}(u, V_u, \mathcal{E} \cup \{e\})$;
9 $E_1 \leftarrow E_1 \cup \{e\}$ and $E_2 \leftarrow E_2 \setminus \{e\}$;
10 $\partial\Lambda_1 \leftarrow \{w \notin \Lambda_1 \mid \exists w' \in \Lambda_1 \text{ s.t. } (w', w) \in E_2\}$ and $\partial\Lambda \leftarrow \{w \notin \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E_2\}$;
11 $c_0 \leftarrow \mathsf{ApproxCount}(V_u, E_2, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$;
12 $c_1 \leftarrow \mathsf{ApproxCount}(V_u, E_2, \Lambda_1, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda_1})$;
13 let $c \leftarrow 1$ with probability $\frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}$; otherwise $c \leftarrow 0$;
14 **if** $c = 1$, **then** let $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$, $\Lambda \leftarrow \Lambda_1$, $p \leftarrow p\frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}$;
15 **if** $c = 0$, **then** let $p \leftarrow p\Big(1 - \frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}\Big)$;
16 **for** *all edges* $e \in E_2$ **do**
17 let $c \leftarrow 1$ with probability $1 - q_e$; otherwise $c \leftarrow 0$;
18 **if** $c = 1$, **then** let $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$ and $p \leftarrow p(1 - q_e)$;
19 **if** $c = 0$, **then** let $p \leftarrow pq_e$;
20 let $F \leftarrow 1$ with probability $\frac{w_u(H)}{4pp_0}$, where $H = (V_u, \mathcal{E})$;
21 **until** $T < 0$ *or* $F = 1$;
22 **if** $F = 1$ then **return** $H = (V_u, \mathcal{E})$; **else return** $\perp$;

Line 20, 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. Ideally, H is a perfect sample from π_u conditional on H passes the filter. The detailed analysis of the error is given in Lemma 11 and in Section 4.3.

Before we go to the ApproxCount algorithm, we state one important property of Algorithm 2. The topological ordering in Line 7 is the ordering $s = v_1, v_2, \dots, v_n = t$ in G. For any two vertices v, v', we write $v \prec v'$ if $v = v_i$, $v' = v_j$ and $i < j$.

Fact 5. For any path $u_1, u_2, \dots, u_\ell$ in G, it holds that $u_1 \prec u_2 \prec \dots \prec u_\ell$.

Lemma 6. *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).

Proof. For any i, we use we use $X^{(i)}$ to denote some (vertex or edge) set X at the beginning of the i-th loop, where X can be $\Lambda, \partial\Lambda, \mathcal{E}, E_1, E_2$. We prove the lemma by contradiction. Suppose at the beginning of k-th loop, there exists $w \in \partial\Lambda^{(k)}$ such that $E_1^{(k)} \cap E_w \neq \emptyset$. We pick an arbitrary edge $(v, v') \in E_1^{(k)} \cap E_w$. Since $(v, v') \in E_1^{(k)}$, there must exist $j < k$ such that $(v, v') \notin E_1^{(j)}$ but $(v, v') \in E_1^{(j+1)}$, which means that the algorithm picks the edge (v, v') in the j-th loop. We will prove that such j cannot exist, which is a contradiction.

Since $w \in \partial\Lambda^{(k)}$, there must exist $w' \in \Lambda^{(k)}$ such that $(w', w) \in E_2^{(k)} \subseteq E$, where E is the set of edges in the input graph G. By the definition of $\Lambda^{(k)}$, there exists a path $w_0, w_1, \ldots, w_{t-1}$ such that

- $w_0 = u$ and $w_{t-1} = w'$;
- for all $1 \leqslant i \leqslant t-1$, $(w_{i-1}, w_i) \in \mathcal{E}^{(k)}$.

Hence, $\{w_0, w_1, \ldots, w_{t-1}, w_t\}$, where $w_t = w$, is a path in G. Note that (v, v') is an edge in the graph G_w. By the definition of G_w, $w \prec v'$ (the vertex v may be w). By Fact 5, we have

$$w_0 \prec w_1 \prec \ldots \prec w_t \prec v'. \tag{7}$$

Note that $w_0 = u \in \Lambda^{(j)}$ for all $j \geqslant 1$. Also, since $w_t = w \notin \Lambda^{(k)}$, $w_t \notin \Lambda^{(j)}$ for all $j < k$. Hence, for any $1 \leqslant j < k$, there is an index $i^* \in \{1, 2 \ldots, t\}$ such that $w_{i^*-1} \in \Lambda^{(j)}$ and $w_{i^*} \notin \Lambda^{(j)}$. We claim that $(w_{i^*-1}, w_{i^*}) \in E_2^{(j)}$.

- If $i^* = t$, then $(w_{t-1}, w_t) \in E_2^{(k)}$. Since $E_2^{(k)} \subseteq E_2^{(j)}$, $(w_{t-1}, w_t) \in E_2^{(j)}$;
- If $i^* \leqslant t-1$, then $(w_{i^*-1}, w_{i^*}) \in \mathcal{E}^{(k)}$. Suppose $(w_{i^*-1}, w_{i^*}) \notin E_2^{(j)}$. Then (w_{i^*-1}, w_{i^*}) has been scanned before the j-th loop, namely $(w_{i^*-1}, w_{i^*}) \in E_1^{(j)}$. However, $w_{i^*} \notin \Lambda^{(j)}$, namely that w_{i^*} cannot be reached using $\mathcal{E}^{(j)}$. It implies that $(w_{i^*-1}, w_{i^*}) \notin \mathcal{E}^{(j)}$ as otherwise w_{i^*} can be reached because $w_{i^*-1} \in \Lambda^{(j)}$. This means that (w_{i^*-1}, w_{i^*}) has been scanned, but not added into $\mathcal{E}$, which implies that $(w_{i^*-1}, w_{i^*}) \notin \mathcal{E}^{(k)}$. A contradiction. Hence, $(w_{i^*-1}, w_{i^*}) \in E_2^{(j)}$.

Thus the claim holds. It implies that $w_{i^*} \in \partial\Lambda^{(j)}$. On the other hand, by (7), $w_{i^*} \prec v'$. The way the next edge is chosen in Line 7 implies that in the j-th loop, the algorithm may only choose an edge (s, s') such that $s' \in \partial\Lambda^{(j)}$ and $s' \preccurlyeq w_{i^*}$. In particular, the vertex s' cannot be v' and the edge (v, v') cannot be chosen. As this argument holds for all $j < k$, it proves the lemma. $\square$

Corollary 7. *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 6. $\square$

3.3. **Approximate counting.** Our ApproxCount subroutine is used in both Algorithm 1 and Algorithm 2. Consider the following more general version of ApproxCount$(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w \in \partial\Lambda})$:

- the input is a DAG $G = (V, E)$, where the graph contains 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;
- let $\partial\Lambda = \{w \in V \setminus \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E\}$;
- 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 the points above hold when Algorithm 3 is evoked by Algorithm 1 or Algorithm 2. The first three points are easy to verify and the last one is verified in Section 4.

For the subset Λ, define the reliability

$$R_\Lambda = \Pr_{\mathcal{G}}\left[\Lambda \rightsquigarrow_{\mathcal{G}} t\right],$$

where $\mathcal{G}$ is a random spanning subgraph of G such that each edge e is removed independently with probability q_e, and $\Lambda \rightsquigarrow_{\mathcal{G}} t$ denotes the event that $\exists w \in \Lambda$ s.t. $w \rightsquigarrow_{\mathcal{G}} t$.

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.[3]

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

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

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 by

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

For the set Λ, we use $\partial\Lambda = \{u_1, \ldots, u_d\}$ to denote the out-neighbours of Λ in graph G. Let $\partial\Lambda$ be listed as $\{u_1, \ldots, u_d\}$ for some $d \in [n]$. Note that $d \geqslant 1$ 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 $1 \leqslant i \leqslant d$, 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 8. *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[4]

$$R_\Lambda^{(i)} := \sum_{H \in \Omega_\Lambda^{(i)}} w_\Lambda(H), \tag{11}$$

$$\forall H \in \Omega_\Lambda^{(i)}, \quad \pi_\Lambda^{(i)}(H) := \frac{w_\Lambda(H)}{R_\Lambda^{(i)}}. \tag{12}$$

Suppose for now we can do the following three things:[5]

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 $H \in \Omega_\Lambda^{(i)}$.

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 $H \in \Omega_\Lambda^{(j)}$.

---

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

[4] One may find that if $\Omega_\Lambda^{(i)} = \emptyset$, then $\pi_\Lambda^{(i)}$ is not well-defined. Remark 17 explains that $\Omega_\Lambda^{(i)} = \emptyset$ never happens because of Lemma 16.

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

It is straightforward to see that

$$\mathbb{E}[Z_\Lambda] = \sum_{i=1}^{d} \frac{R_\Lambda^{(i)}}{\sum_{j=1}^{d} R_\Lambda^{(j)}} \sum_{H \in \Omega_\Lambda^{(i)}} \pi_\Lambda^{(i)}(H) \cdot \mathbf{1}\left[H \in \Omega_\Lambda^{(i)} \wedge \left(\forall j < i, H \notin \Omega_\Lambda^{(j)}\right)\right]$$

$$= \frac{\sum_{i=1}^{d} \sum_{H \in \Omega_\Lambda^{(i)}} w_\Lambda(H) \cdot \mathbf{1}\left[H \in \Omega_\Lambda^{(i)} \wedge \left(\forall j < i, H \notin \Omega_\Lambda^{(j)}\right)\right]}{\sum_{j=1}^{d} R_\Lambda^{(j)}}$$

$$\text{(by Lemma 8)} \quad = \frac{R_\Lambda}{\sum_{j=1}^{d} R_\Lambda^{(j)}} \geqslant \frac{1}{d} \geqslant \frac{1}{n}, \tag{13}$$

where the first inequality holds because each H belongs to at most d different sets. Since Z_Λ is a $0/1$ random variable, $\mathrm{Var}(Z_\Lambda) \leqslant 1$. 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 $u_i \in \partial\Lambda$. Next, we describe how our algorithm uses these estimates.

Let G be the input graph of the ApproxCount. For any $i \in [d]$ (namely $u_i \in \partial\Lambda$), we write G_{u_i} instead of $G_{\{u_i\}}$, and similarly V_{u_i} etc. Moreover, define

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

Lemma 9. *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*

$$\mathbf{1}[D \neq \emptyset] \cdot \prod_{e \in \delta_\Lambda(u_i) \cap D} (1 - q_e) \prod_{f \in \delta_\Lambda(u_i) \setminus D} q_f; \tag{14}$$

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

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 the definitions (11) and (12), we have that $\pi_\Lambda^{(i)}$ 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. □

Next, we show that the second step in Lemma 9 can be done efficiently.

Lemma 10. *There is an algorithm such that given a set* $S = \{1, 2, \ldots, n\}$ *and* n *numbers* $0 \leqslant q_1, q_2, \ldots, q_n < 1$, *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* $O(n)$.

Proof. Note that D can be obtained by sampling each i in S independently with probability $1 - q_i$ conditional on the outcome is non-empty. A natural idea is to use rejection sampling, but $1 - \prod_{i=1}^n q_i$ can be very small. Here we do this in a more efficient way.

We view any subset $D \subseteq S$ as an n-dimensional vector $\sigma \in \{0, 1\}^S$. We sample σ_i for i from 1 to n one by one. In every step, conditional on $\sigma_1 = c_1, \sigma_2 = c_2, \ldots, \sigma_{i-1} = c_{i-1} \in \{0, 1\}$, we compute the marginal of σ_i and sample from the marginal. The marginal can be computed as follows: for any $c_i \in \{0, 1\}$,

$$\Pr\left[\sigma_i = c_i \mid \forall j < i, \sigma_j = c_j\right] = \frac{\Pr[\forall j \leqslant i, \sigma_j = c_j]}{\Pr[\forall j \leqslant i - 1, \sigma_j = c_j]}.$$

It suffices to compute $\Pr[\forall j \leqslant i, \sigma_j = c_j]$ for any $1 \leqslant i \leqslant n$. Let Ω denote the set of all assignments for $\{i + 1, i + 2, \ldots, n\}$. For any $\tau \in \Omega$, τ is an $(n - i)$-dimensional vector, where $\tau_k \in \{0, 1\}$ is the value for $k \geqslant i + 1$. We use $(c_j)_{j \leqslant i} + \tau$ to denote an n-dimensional vector. For any j, let $f_j(0) = q_j$ and $f_j(1) = 1 - q_j$. Note that

$$\Pr\left[\forall j \leqslant i, \sigma_j = c_j\right] = \frac{\prod_{j=1}^i f_j(c_j) \sum_{\tau \in \Omega} \prod_{k=i+1}^n f_k(\tau_k) \mathbf{1}\left[(c_j)_{j \leqslant i} + \tau \text{ is not zero vector}\right]}{1 - \prod_{j=1}^n q_i}.$$

Hence, if $c_1 + c_2 + \ldots + c_i \geqslant 1$, then

$$\Pr\left[\forall j \leqslant i, \sigma_j = c_j\right] = \frac{\prod_{j=1}^i f_j(c_j)}{1 - \prod_{j=1}^n q_i}.$$

If $c_1 + c_2 + \ldots + c_i = 0$, then

$$\Pr\left[\forall j \leqslant i, \sigma_j = c_j\right] = \frac{(1 - \prod_{k=i+1}^n q_k) \prod_{j=1}^i f_j(c_j)}{1 - \prod_{j=1}^n q_i}.$$

Hence, every conditional marginal can be computed by the formula above in time $O(n)$. An naive sampling implementation takes $O(n^2)$ time to compute all the marginal probabilities, but it is not hard to see that a lot of prefix or suffix products can be reused and the total running time of sampling can be reduced to $O(n)$. □

Now, we are 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 9 and Lemma 10, we can efficiently approximate $R_\Lambda^{(i)}$ and generate approximate samples from $\pi_\Lambda^{(i)}$. Hence, we can simulate the process described earlier to estimate R_Λ. A detailed description of ApproxCount is given in Algorithm 3.

Algorithm 3: ApproxCount$\left(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda}\right)$

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

1 **if** $t \in \Lambda$, **then return** 0; **if** Λ cannot reach t in G, **then return** 1;

2 for each vertex $u_i \in \partial\Lambda$, compute $\widetilde{R}_\Lambda^{(i)} := \left(1 - \prod_{u\in\delta_\Lambda(u_i)} q_{(u,u_i)}\right)\widetilde{R}_{u_i}$;

3 for each vertex $u_i \in \partial\Lambda$, construct $S_{u_i}^{(j)} \subseteq S_{u_i}$ for $1 \leqslant j \leqslant B$ by partitioning S_{u_i} into B multi-sets, where $B = 60n + 150m$ and each $S_{u_i}^{(j)}$ has $\ell_0 = \left\lceil \frac{10^5 n^3 m^2}{\varepsilon^2} \right\rceil$ samples;

4 **for** j *from* 1 *to* B **do**

5   **for** k *from* 1 *to* ℓ_0 **do**

6   draw an index $i \in [d]$ such that i is drawn with probability proportional to $\widetilde{R}_\Lambda^{(i)}$;

7   pick an arbitrary sample $H = (V_{u_i}, E_H)$ from the set $S_{u_i}^{(j)}$ and remove H from the set $S_{u_i}^{(j)}$;

8   do the following transformation on H to get $H' = (V, E_{H'})$;

9   • let $E_{H'} \leftarrow E_H$;

10   • draw $D \subseteq \delta_\Lambda(u_i)$ with probability proportional to (14) in Lemma 9, and let $E_{H'} \leftarrow E_{H'} \cup D$;

11   • for each $e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))$, add e into $E_{H'}$ independently with probability $1 - q_e$;

12   let $Z_\Lambda^{(j,k)} \in \{0, 1\}$ indicate whether i is the smallest index $t \in [d]$ satisfying $H \in \Omega_\Lambda^{(t)}$;

13     $Q_\Lambda^{(j)} \leftarrow \left(\frac{1}{\ell_0}\sum_{k=1}^{\ell_0} Z_\Lambda^{(j,k)}\right)\sum_{i=1}^{d} \widetilde{R}_\Lambda^{(i)}$;

14 **return** $\widetilde{R}_\Lambda :=$ median $\left\{Q_\Lambda^{(1)}, Q_\Lambda^{(2)}, \dots, Q_\Lambda^{(B)}\right\}$;

Recall that ℓ is defined in (6). For any $u_i \in \partial\Lambda$, we divide all ℓ samples in S_{u_i} into B blocks, where $B = 60n + 150m$ and each block has $\ell_0 = \lceil \frac{10^5 n^3 m^2}{\varepsilon^2} \rceil$ samples. Denote the B blocks by $S_{u_i}^{(1)}, S_{u_i}^{(j)}, \dots, S_{u_i}^{(B)}$, where each $S_{u_i}^{(j)}$ is a multi-set of samples. Our algorithm generates B estimators $Q_\Lambda^{(j)}$ for the value R_Λ and return their median. To compute each $Q_\Lambda^{(j)}$, we only use the samples in $S_{u_i}^{(j)}$ for $1 \leqslant i \leqslant d$. We compute $Z_\Lambda^{(j,k)}$ for $1 \leqslant k \leqslant \ell_0$, take their average, and let $Q_\Lambda^{(j)}$ be the product of the average value and $\sum_{i=1}^{d} \widetilde{R}_\Lambda^{(i)}$. Note that each $S_{u_i}^{(j)}$ has ℓ_0 samples, so the set $S_{u_i}^{(j)}$ cannot be empty in Line 7. Also, in Line 7, we only remove elements from the set $S_{u_i}^{(j)}$, but the set S_{u_i} remains unchanged. In fact, the set S_{u_i} would be reused by other instances of ApproxCount.

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

4. Analysis

In this section we analyze all the algorithms.

4.1. **Analysis of Sample.** Let $G = (V, E)$ be the input graph of Algorithm 1. Consider the subroutine $\mathsf{Sample}\left(v_k, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\dots,v_n\}}, \widetilde{R}_{v_k}\right)$ as being called by Algorithm 1. Let $u := v_k$, and the subroutine runs on the graph $G_u = (V_u, E_u)$. In this section we consider a modified version of Sample, and handle the real version in Section 4.3. Let m denote the number of edges in G. Suppose we can access an oracle $\mathcal{P}$ satisfying:

- given $u \in V_u$, $\mathcal{P}$ returns p_0 such that

$$1 - \frac{1}{10m} \leqslant \frac{p_0}{R(V_u, E_u, \{u\})} \leqslant 1 + \frac{1}{10m}; \tag{15}$$

- given any $E_2 \subseteq E_u$ and $\Lambda, \Lambda_1 \subseteq V$ in Line 11 and Line 12 of Algorithm 2, $\mathcal{P}$ returns $c_0(V_u, E_2, \Lambda)$ and $c_1(V_u, E_2, \Lambda_1)$ such that

$$1 - \frac{1}{10m} \leqslant \frac{c_0(V_u, E_2, \Lambda)}{R(V_u, E_2, \Lambda)} \leqslant 1 + \frac{1}{10m}, \text{ and} \tag{16}$$

$$1 - \frac{1}{10m} \leqslant \frac{c_1(V_u, E_2, \Lambda_1)}{R(V_u, E_2, \Lambda_1)} \leqslant 1 + \frac{1}{10m}. \tag{17}$$

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. Abstractly, one can view $\mathcal{P}$ as a random vector $\mathcal{X}_{\mathcal{P}}$, where

$$\mathcal{X}_{\mathcal{P}} = \{p_0\} \cup \{c_0(V_u, E_2, \Lambda), c_1(V_u, E_2, \Lambda_1) \mid \text{ for all possible } V_u, E_2, \Lambda, \Lambda_1\}.$$

The dimension of $\mathcal{X}_{\mathcal{P}}$ is huge because there may be exponentially many possible $V_u, E_2, \Lambda, \Lambda_1$ in Line 11 and Line 12. The oracle first draw a sample $x_{\mathcal{P}}$ of $\mathcal{X}_{\mathcal{P}}$, then answers queries by looking at $x_{\mathcal{P}}$ on the corresponding coordinate. The conditions above are assumed to be satisfied with probability 1. Note that this $\mathcal{X}_{\mathcal{P}}$ is only for analysis purposes and is not part of the real implementation.

The modified sampling algorithm replaces Line 2, Line 11 and Line 12 of Algorithm 2 by calling the oracle $\mathcal{P}$. In that case we do not need the estimates $(\widetilde{R}_w, S_w)$ for $w = v_{k+1}, \dots, v_n$ and $\widetilde{R}_{v_k}$, and thus may assume that the input is only $u = v_k$. Recall that n is the number of vertices in the input graph.

Lemma 11. *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, where the probability is over the independent randomness $\mathcal{D}_u$ inside the Sample subroutine. 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.*

Proof. Throughout this proof, we fix a sample $x_{\mathcal{P}}$ of $\mathcal{X}_{\mathcal{P}}$ in advance. The oracle $\mathcal{P}$ uses $x_{\mathcal{P}}$ to answer the queries. We will prove that the lemma holds for any $x_{\mathcal{P}}$ satisfying the three conditions above.

We first describe an ideal sampling algorithm. The algorithm maintains the set E_1, E_2 and $\mathcal{E}$ as in the Sample algorithm. At each step, we pick an edge e according to Line 7 of Algorithm 2. We compute the conditional marginal probability of $\alpha_e = \Pr_{\mathcal{G}=(V_u,E')\sim\pi_u}[e \in E' \mid E' \cap E_1 = \mathcal{E}]$, and add e into $\mathcal{E}$ with probability α_e. Then we update E_1, E_2 and Λ. Once $t \in \Lambda$, $\alpha_e = 1 - q_e$ for all $e \in E_2$ and we can add all subsequent edges independently. The ideal sampling algorithm returns an independent perfect sample.

The modified algorithm simulates the ideal process, but uses the oracle $\mathcal{P}$ to compute each conditional marginal distribution α_e. By the definition of conditional probability,

$$\alpha_e = \frac{\Pr_{\mathcal{G}=(V_u,E')\sim\pi_u}[e \in E' \wedge E' \cap E_1 = \mathcal{E}]}{\Pr_{\mathcal{G}=(V_u,E')\sim\pi_u}[e \in E' \wedge E' \cap E_1 = \mathcal{E}] + \Pr_{\mathcal{G}=(V_u,E')\sim\pi_u}[e \notin E' \wedge E' \cap E_1 = \mathcal{E}]}. \tag{18}$$

Recall $E_2 = E_u \setminus E_1$. Let $E_2' = E_2 \setminus e$. Recall Λ_1 is the set of vertices u can reach if $\mathcal{E} \cup \{e\}$ is selected. Conditional on that $\mathcal{E} \cup \{e\}$ is selected, the probability that u can reach t is exactly the same as the probability Λ_1 can reach t in the remaining graph (V_u, E_2'). Then the numerator of (18) can be written as

$$(1-q_e)\prod_{f\in E_1\cap\mathcal{E}}(1-q_f)\prod_{f'\in E_1\cap\mathcal{E}}q_{f'}\cdot R(V_u, E_2', \Lambda_1),$$

where $R(V_u, E_2', \Lambda_1)$ is the Λ_1-t reliability in the graph (V_u, E_2'). Similarly, the second term of the denominator of (18) can be written as

$$q_e\prod_{f\in E_1\cap\mathcal{E}}(1-q_f)\prod_{f'\in E_1\cap\mathcal{E}}q_{f'}\cdot R(V_u, E_2', \Lambda).$$

Putting them together implies

$$\alpha_e = \frac{(1-q_e)R(V_u, E_2', \Lambda_1)}{(1-q_e)R(V_u, E_2', \Lambda_1) + q_e R(V_u, E_2', \Lambda)}.$$

If c_0 and c_1 in Line 11 and Line 12 are exactly $R(V_u, E_2', \Lambda)$ and $R(V_u, E_2', \Lambda_1)$, then Line 5 to Line 19 in Algorithm 2 are the same as the ideal algorithm described above. Under this assumption, Algorithm 2 cannot crash in Line 7 or Line 13. Consider the modified algorithm in the lemma. Note that the state of the algorithm can be uniquely determined by the pair $(E_2, \mathcal{E})$. By the assumption of $\mathcal{P}$, we know that $R(V_u, E_2', \Lambda) = 0$ if and only if $c_0 = 0$ and $R(V_u, E_2', \Lambda_1) = 0$ if and only if $c_1 = 0$. Hence, any state $(E_2, \mathcal{E})$ appears in the modified algorithm with positive probability if and only if it appears in the ideal algorithm with positive probability. This implies the modified algorithm cannot crash in Line 7 or Line 13.

By the assumption of the oracle $\mathcal{P}$ again, we have

$$1-\frac{1}{4m} \leqslant \frac{10m-1}{10m+1} \leqslant \frac{\frac{(1-q_e)c_1}{(1-q_e)c_1+q_e c_0}}{\alpha_e} \leqslant \frac{10m+1}{10m-1} \leqslant 1+\frac{1}{4m},$$
$$1-\frac{1}{4m} \leqslant \frac{10m-1}{10m+1} \leqslant \frac{\frac{q_e c_0}{(1-q_e)c_1+q_e c_0}}{1-\alpha_e} \leqslant \frac{10m+1}{10m-1} \leqslant 1+\frac{1}{4m}.$$

When the algorithm exits the whole loop, u can reach t and we have the remaining marginals exactly.

Finally, the algorithm gets a random subgraph H and a value p, where $p = p(H)$ is the probability that the algorithm generates H. Note that there are at most m edges in E_u. Taking the product of all conditional marginals gives

$$\exp(-1/2) \leqslant \left(1-\frac{1}{4m}\right)^m \leqslant \frac{p(H)}{\pi_u(H)} \leqslant \left(1+\frac{1}{4m}\right)^m \leqslant \exp(1/4).$$

Recall that

$$\pi_u(H) = \frac{w_u(H)}{R(V_u, E_u, u)}.$$

By the assumption of the oracle $\mathcal{P}$, we have

$$\frac{9}{10} \leqslant \frac{p_0}{R(V_u, E_u, u)} \leqslant \frac{11}{10}.$$

The parameter $p_0 > 0$ because the input of Algorithm 2 must satisfy $R(V_u, E_u, u) > 0$. The filter probability $f = \Pr[F = 1 \mid H]$ in Line 20 of Algorithm 2 satisfies

$$\frac{1}{16} \leqslant f = \frac{w_u(H)}{4p(H)p_0} \leqslant 1.$$

Hence, f is a valid probability and $f \geqslant \frac{1}{16}$. The algorithm cannot crash in Line 20. The algorithm outputs H if $F = 1$. By the analysis above, we know that $p(H) > 0 \Leftrightarrow \pi_u(H) > 0$ and

$$\Pr[\text{Sample outputs } H] \propto p(H)\frac{w_u(H)}{p(H)p_0} = \frac{w_u(H)}{p_0} \propto w_u(H),$$

where the last "proportional to" holds because p_0 is a constant (independent from H). Conditional on $F = 1$, H is a perfect sample. We repeat the process for $T = 1000 \log \frac{n}{\varepsilon}$ times, and each time the algorithm succeeds with probability at least $\frac{1}{16}$. The overall probability of success is at least $1 - (\varepsilon/n)^{200}$.

The running time is dominated by the oracle calls. We can easily use data structures to maintain $\partial\Lambda$, Λ, E_2 and $\mathcal{E}$. The total running time is $\tilde{O}((|V_u| + |E_u|)N)$. $\square$

The above only deals with the modified algorithm. Analysing the real algorithm relies on the analysis of ApproxCount, and we defer that to Section 4.3.

The Algorithm 2 can only be evoked by Algorithm 1. Fix $u = v_k$. Suppose we use Algorithm 2 to draw samples from π_u. For later analysis, we need to make clear how each random variable depends on various sources of randomness. We abstract the modified algorithm as follows. The oracle $\mathcal{P}$ is determined by a random vector $\mathcal{X}_\mathcal{P}$. The algorithm generates the inside independent randomness $\mathcal{D}_u$. The algorithm constructs a random subgraph $H = H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)$ and a random indicator variable $F = F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)$, where H and F denote the random variables of the same name in the last line of Algorithm 2. Lemma 11 shows that conditional on $F = 1$, H is an independent sample (independent from $\mathcal{X}_\mathcal{P}$) that follows π_u. We denote it by

$$H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)|_{F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)=1} \sim \pi_u.$$

Here, for any random variable X and event E, we use $X|_E$ to denote the random variable X conditional on E. In fact, a following stronger result can be obtained from the above proof

$$\forall x_\mathcal{P} \in \Omega_\mathcal{P}, \quad H|_{F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)=1 \wedge \mathcal{X}_\mathcal{P}=x_\mathcal{P}} \sim \pi_u.$$

where $\Omega_\mathcal{P}$ denotes the support of $\mathcal{X}_\mathcal{P}$. And it holds that

$$\forall x_\mathcal{P} \in \Omega_\mathcal{P}, \quad \Pr[F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u) = 1 \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}] \geqslant 1 - \frac{1}{(n/\varepsilon)^{200}}.$$

Note that the event $F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u) = 1$ depends on the input random variable $\mathcal{X}_\mathcal{P}$. In the analysis in Section 4.3, we need to define a event $\mathcal{C}$ such that $\Pr[\mathcal{C}] \geqslant 1 - (\varepsilon/n)^{200}$, $\mathcal{C}$ is independent from $\mathcal{X}_\mathcal{P}$ and $H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)|_\mathcal{C} \sim \pi_u$. We actually define this event $\mathcal{C}$ in a more refined probability space. The proof below defines this event explicitly. We also include an alternative, more conceptual, and perhaps simpler proof in Appendix A, where the event $\mathcal{C}$ is defined implicitly.

Consider the following algorithm NewSample. Recall that $T = \lceil 1000 \log \frac{n}{\varepsilon} \rceil$ is the parameter in Algorithm 2.

Definition 12 (NewSample)**.** The algorithm NewSample is the same as the Sample in Algorithm 2. The only difference is that before Line 22, NewSample computes the value

$$p_K := \frac{1 - \frac{\epsilon^{200}}{n^{200}}}{1 - (1 - \frac{R_u}{4p_0})^T}. \tag{19}$$

If $0 \leqslant p_K \leqslant 1$, then independently sample $K \in \{0, 1\}$ such that $\Pr[K = 1] = p_K$; otherwise, let $K = 0$.

We remark that in the above definition, K is sampled using independent randomness. Formally, let $\mathcal{D}_u$ be the inside randomness of NewSample. We partition $\mathcal{D}_u$ into two disjoint random strings $\mathcal{D}_u^{(1)}$ and $\mathcal{D}_u^{(2)}$. We use $\mathcal{D}_u^{(1)}$ to simulate all steps in Algorithm 2 and use $\mathcal{D}_u^{(2)}$ to sample K.

The value R_u is the exact u-t reliability in graph G_u. Indeed, we cannot compute the exact value of R_u in polynomial time. We only use the algorithm NewSample in analysis, and do not need to implement this algorithm. NewSample draws a random variable K but never uses it at all. Its sole purpose is to further refine the probability space. Thus, the following observation is straightforward to verify.

Observation 13. *Given the same input, the outputs of two algorithms NewSample and Sample follow the same distribution.*

A natural question here is that why do we even define NewSample? By Observation 13, we can focus only on NewSample in later analysis (in particular, the analysis of the correctness of our algorithm). NewSample has one additional random variable $K \in \{0, 1\}$, which helps defining the event $\mathcal{C}$ below.

Similarly, we can defined a modified version of NewSample such that we use the oracle $\mathcal{P}$ to compute p_0, c_0 and c_1. NewSample also generates the same random subgraph $H = H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})$ and the same random indicator $F = F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})$ as Sample. In addition, it generates a new random variable $K = K(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(2)})$. We define the following event $\mathcal{C}$ for NewSample

$$\mathcal{C}: \quad F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) = 1 \wedge K(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(2)}) = 1. \tag{20}$$

Lemma 14. *Suppose $\mathcal{P}$ satisfies the conditions in (15), (16) and (17). Then with probability 1, $0 \leqslant p_K \leqslant 1$. Furthermore, it holds that*

- *$\mathcal{C}$ is independent from $\mathcal{X}_\mathcal{P}$, which implies that $\mathcal{C}$ depends only on $\mathcal{D}_u$;*
- *$\Pr_{\mathcal{D}_u}[\mathcal{C}] = 1 - \varepsilon^{200}/n^{200}$;*
- *conditional on $\mathcal{C}$, NewSample does not crash and outputs an independent sample $H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) \sim \pi_u$.*

Proof. Suppose $\mathcal{P}$ satisfies the conditions in (15), (16) and (17). We first fix $\mathcal{X}_\mathcal{P} = x_\mathcal{P}$ for an arbitrary $x_\mathcal{P} \in \Omega_\mathcal{P}$. By the same analysis as in Lemma 11, in each of the repeat-until loop, the probability f of $F = 1$ is

$$\frac{1}{16} \leqslant f = \sum_{H \in \Omega_u} p(H) \cdot \frac{w_u(H)}{4p(H)p_0} = \frac{R_u}{4p_0} \leqslant 1.$$

As the repeat-until loop is repeated independently for at most T times until $F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u) = 1$, we have

$$\Pr[F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u) = 1 \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}] = 1 - \left(1 - \frac{R_u}{4p_0}\right)^T \geqslant 1 - \frac{\varepsilon^{200}}{n^{200}}.$$

Hence, $0 \leqslant p_K \leqslant 1$. Next, note that given $\mathcal{X}_\mathcal{P} = x_\mathcal{P}$, the value of p_K is fixed and K is sampled independently. We have that $K(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(2)})$ and $F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})$ are independent conditional on $\mathcal{X}_\mathcal{P} = x_\mathcal{P}$. This implies

$$\begin{aligned}\Pr[\mathcal{C} \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}] &= \Pr[F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) = 1 \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}] \Pr[K(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(2)}) = 1 \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}] \\ &= \left(1 - \left(1 - \frac{R_u}{4p_0}\right)^T\right) p_K = 1 - \frac{\varepsilon^{200}}{n^{200}}.\end{aligned}$$

The probability $1 - \varepsilon^{200}/n^{200}$ is independent from $x_\mathcal{P}$. Hence, the event $\mathcal{C}$ is independent from $\mathcal{X}_\mathcal{P}$.

Finally, we analyze the distribution of H conditional on $\mathcal{C}$. We first condition on $\mathcal{X}_\mathcal{P} = x_\mathcal{P}$. If we further conditional on $F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) = 1$, the same analysis as in Lemma 11 shows that the algorithm does not crash and $H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) \sim \pi_u$. Note that $K(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(2)})$ is sampled independently with a fixed probability p_K (since $\mathcal{X}_\mathcal{P} = x_\mathcal{P}$ has been fixed). Hence, $K(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(2)})$ is independent from both $F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})$ and $H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})$ conditional on $\mathcal{X}_\mathcal{P} = x_\mathcal{P}$. We have

$$H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})|_{\mathcal{X}_\mathcal{P} = x_\mathcal{P} \wedge \mathcal{C}} \equiv H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)})|_{\mathcal{X}_\mathcal{P} = x_\mathcal{P} \wedge F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) = 1} \sim \pi_u,$$

where we use $X \equiv Y$ to denote that two random variables X and Y have the same distribution. Note that the distribution π_u on the RHS is independent from $x_\mathcal{P}$. Summing over $x_\mathcal{P} \in \Omega_\mathcal{P}$ gives that conditioned on $\mathcal{C}$, the output $H = H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u^{(1)}) \sim \pi_u$. ∎

4.2. **Analysis of ApproxCount.** Now we turn our attention to $\text{ApproxCount}(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$, where $G = (V, E)$ is a DAG and $t \in \Lambda$. 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. Let S_w^{ideal} be a multi-set of ℓ *independent* and *prefect* samples from π_w. Recall that ℓ_0 and B are parameters in ApproxCount, Algorithm 3, and $d = |\partial\Lambda|$. In the next lemma, we assume $(\widetilde{R}_w)_{w\in\partial\Lambda}$ is fixed and $(S_w)_{w\in\partial\Lambda}$ is random. When ApproxCount is called, we use $\mathcal{D}(V, E, \Lambda)$ to denote the internal randomness in the execution of ApproxCount.

Lemma 15. *Suppose the following conditions are satisfied*

- *for all* $w \in \partial\Lambda$, $w \rightsquigarrow_G t$*;*
- *for any* $w \in \partial\Lambda$, $1 - \varepsilon_0 \leqslant \frac{\widetilde{R}_w}{R_w} \leqslant 1 + \varepsilon_0$ *for some* $\varepsilon_0 < 1/2$*;*
- $d_{TV}\left((S_w)_{w\in\partial\Lambda}, (S_w^{\text{ideal}})_{w\in\partial\Lambda}\right) \leqslant \delta_0$.

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

$$1 - \varepsilon_0 - \frac{6\sqrt{d}}{\sqrt{\ell_0}} \leqslant \frac{\widetilde{R}_\Lambda}{R_\Lambda} \leqslant 1 + \varepsilon_0 + \frac{6\sqrt{d}}{\sqrt{\ell_0}},$$

where the probability is taken over the input randomness of $(S_w)_{w\in\partial\Lambda}$ *and the independent randomness* $\mathcal{D}(V, E, \Lambda)$ *inside the* ApproxCount *algorithm. The running time of* ApproxCount *is* $O((|E| + |V|)d\ell)$.

Proof. If $t \in \Lambda$, then Algorithm 3 returns $\widetilde{R}_w = R_w = 1$ in Line 1. If $t \notin \Lambda$ and $\partial\Lambda = \emptyset$, then Algorithm 3 returns $\widetilde{R}_w = R_w = 0$ in Line 1. In the following, we assume $t \notin \Lambda$ and $\partial\Lambda \neq \emptyset$. By the first condition, we have $d \geqslant 1$ in Algorithm 3 and all distributions $\pi_\Lambda^{(i)}$ for $i \in [d]$ are well-defined.

By Lemma 9 and the assumption in this lemma, all $\widetilde{R}_\Lambda^{(i)}$ computed in Line 2 satisfy

$$1 - \varepsilon_0 \leqslant \frac{\widetilde{R}_\Lambda^{(i)}}{R_\Lambda^{(i)}} \leqslant 1 + \varepsilon_0. \tag{21}$$

If ApproxCount uses perfect samples from $(S_w^{\text{ideal}})_{w\in\partial\Lambda}$, then by Lemma 9, H' obtained in Line 11 is a perfect sample from $\pi_\Lambda^{(i)}$. For any $j \in [B]$ and $k \in [\ell_0]$, we have

$$\mathbb{E}[Z_\Lambda^{(j,k)}] = \sum_{i=1}^{d} \frac{\widetilde{R}_\Lambda^{(i)}}{\sum_{t\in[d]} \widetilde{R}_\Lambda^{(t)}} \cdot \sum_{H'\in\Omega_\Lambda^{(i)}} \underbrace{\frac{w_\Lambda(H')}{R_\Lambda^{(i)}}}_{=\pi_\Lambda^{(i)}(H')} \cdot \mathbf{1}\left[H' \in \Omega_\Lambda^{(i)} \wedge \forall t < i, H' \notin \Omega_\Lambda^{(t)}\right]$$

By (21) and a calculation similar to that in (13), we have

$$(1 - \varepsilon_0)\frac{R_\Lambda}{\sum_{i=1}^{d} \widetilde{R}_\Lambda^{(i)}} \leqslant \mathbb{E}[Z_\Lambda^{(j,k)}] \leqslant (1 + \varepsilon_0)\frac{R_\Lambda}{\sum_{i=1}^{d} \widetilde{R}_\Lambda^{(i)}} \tag{22}$$

Using (21), we have

$$\mathbb{E}[Z_\Lambda^{(j,k)}] \geqslant \frac{1 - \varepsilon_0}{1 + \varepsilon_0} \cdot \frac{R_\Lambda}{\sum_{i=1}^{d} \widetilde{R}_\Lambda^{(i)}} \geqslant \frac{1}{4d}. \tag{23}$$

Also recall that

$$Q_\Lambda^{(j)} = \left(\frac{1}{\ell_0}\sum_{k=1}^{\ell_0} Z_\Lambda^{(j,k)}\right)\sum_{i=1}^{d} \widetilde{R}_\Lambda^{(i)}. \tag{24}$$

Let $\overline{Z} = \frac{1}{\ell_0}\sum_{k=1}^{\ell_0} Z_\Lambda^{(j,k)}$. Then $\mathrm{Var}\left(\overline{Z}\right) = \frac{\mathrm{Var}\left(Z_\Lambda^{(j,k)}\right)}{\ell_0}$, and by (23), $\mathbb{E}[\overline{Z}] \geqslant \frac{1}{4d}$. By Chebyshev's inequality and for any $j \in [B]$ and $k \in [\ell_0]$,

$$\begin{aligned}
&\Pr\left[\left|Q_\Lambda^{(j)} - \mathbb{E}[Q_\Lambda^{(j)}]\right| \geqslant \frac{4\sqrt{d}}{\sqrt{\ell_0}}\mathbb{E}[Q_\Lambda^{(j)}]\right] \\
&= \Pr\left[\left|\overline{Z} - \mathbb{E}[\overline{Z}]\right| \geqslant \frac{4\sqrt{d}}{\sqrt{\ell_0}}\mathbb{E}[\overline{Z}]\right] \leqslant \frac{\ell_0}{16d}\cdot\frac{\mathrm{Var}\left(\overline{Z}\right)}{(\mathbb{E}[\overline{Z}])^2} = \frac{\ell_0}{16d}\cdot\frac{\mathrm{Var}\left(Z_\Lambda^{(j,k)}\right)/\ell_0}{\left(\mathbb{E}[Z_\Lambda^{(j,k)}]\right)^2} \\
&= \frac{1}{16d}\left(\frac{\mathbb{E}\left[\left(Z_\Lambda^{(j,k)}\right)^2\right]}{\left(\mathbb{E}[Z_\Lambda^{(j,k)}]\right)^2} - 1\right) = \frac{1}{16d}\left(\frac{\mathbb{E}\left[Z_\Lambda^{(j,k)}\right]}{\left(\mathbb{E}[Z_\Lambda^{(j,k)}]\right)^2} - 1\right) \leqslant \frac{1}{4},
\end{aligned}$$

where the last inequality is due to (23). Combining the above inequality with (22) and (24), we know that with probability at least $3/4$,

$$1 - \varepsilon_0 - \frac{6\sqrt{d}}{\sqrt{\ell_0}} \leqslant \left(1 - \frac{4\sqrt{d}}{\sqrt{\ell_0}}\right)(1-\varepsilon_0) \leqslant \frac{Q_\Lambda^{(j)}}{R_\Lambda} \leqslant \left(1 + \frac{4\sqrt{d}}{\sqrt{\ell_0}}\right)(1+\varepsilon_0) \leqslant 1 + \varepsilon_0 + \frac{6\sqrt{d}}{\sqrt{\ell_0}}.$$

Since $\widetilde{R}_\Lambda$ is the median of B values $Q_\Lambda^{(j)}$, the success probability is boosted from $3/4$ to $1 - 2^{-B/30}$ by the Chernoff bound.

Finally, the algorithm actually uses the samples from $(S_w)_{w\in\partial\Lambda}$. Consider an optimal coupling between the real algorithm with the algorithm using ideal samples from $(S_w^{\mathrm{ideal}})_{w\in\partial\Lambda}$. Due to the assumption that $d_{TV}\left((S_w)_{w\in\partial\Lambda}, (S_w^{\mathrm{ideal}})_{w\in\partial\Lambda}\right) \leqslant \delta_0$ and Lemma 3, the two algorithms output the same answer with probability at least $1-\delta_0$. Hence, $1 - \varepsilon_0 - \frac{6\sqrt{d}}{\sqrt{\ell_0}} \leqslant \frac{\widetilde{R}_\Lambda}{R_\Lambda} \leqslant 1 + \varepsilon_0 + \frac{6\sqrt{d}}{\sqrt{\ell_0}}$ with probability at least $1 - \delta_0 - 2^{-B/30}$.

The running time of the inner loop is dominated by the time spent on Line 12. Each execution of Line 12 costs time $O((|E|+|V|)d)$. The total running time is $O(\ell_0 B(|E|+|V|)d) = O((|E|+|V|)d\ell)$. □

Lemma 15 treats ApproxCount as a standalone algorithm. However, in our main algorithm, we use ApproxCount as a subroutine. We need to make sure that the inpputs are consistent every time ApproxCount is called. Recall that $G = (V, E)$ denotes the input graph of Algorithm 1. Every time when ApproxCount is evoked, its input includes a subset of vertices $V_0 \subseteq V$, a subset of edges $E_0 \subseteq E$, a subset of vertices Λ_0 and $(\widetilde{R}_w, S_w)_{w\in\partial\Lambda_0}$. Recall that $\partial\Lambda_0 = \{w \in V_0 \setminus \Lambda_0 \mid \exists w \in \Lambda_0 \text{ s.t. } (w', w) \in E_0\}$. The properties we need are the following.

Lemma 16. *If ApproxCount is evoked with input $V_0 \subseteq V$, $E_0 \subseteq E$, $\Lambda_0 \subseteq V_0$ and $(\widetilde{R}_w, S_w)_{w\in\partial\Lambda_0}$, then*

- *for all $w \in \partial\Lambda_0$, $E_w \subseteq E_0$, where E_w is the edge set of G_w;*
- *for all $w \in \partial\Lambda_0$, $(\widetilde{R}_w, S_w)$ has already been computed.*

Remark 17. Lemma 16 guarantees that for any input (V_0, E_0, Λ_0) of Algorithm 3, for any $w \in \partial\Lambda_0$, $G_w^0 = G_w$, where $G^0 = (V_0, E_0)$. This implies that $w \rightsquigarrow_{G^0} t$ and all distributions in (12) are well-defined.

Proof of Lemma 16. Note that ApproxCount can be evoked either in Line 3 of Algorithm 1 or in Line 11 and Line 12 in Algorithm 2.

If ApproxCount is evoked by Algorithm 1, then $V_0 = V_{v_k}$, $E_0 = E_{v_k}$ and $\Lambda_0 = \{v_k\}$. For any $w \in V_{v_k} \setminus \{v_k\}$, G_w is a subgraph of $G_{v_k} = (V_0, E_0)$ and the first property holds. The second property holds because $v_k \prec w$ for all $w \in V_{v_k} \setminus \{v_k\}$.

Suppose next that ApproxCount is evoked by Algorithm 2. For the first property, by Lemma 6, at the beginning of every while-loop, for any $w \in \partial\Lambda$, $E_1 \cap E_w = \emptyset$, which implies $E_w \subseteq E_2$. In other words, the property holds at the beginning of the loop with $V_0 = V_u$, $E_0 = E_2$, and $\Lambda_0 = \Lambda$. Now consider Line 11 and Line 12 separately.

- Suppose ApproxCount is called in Line 11. In this case, comparing to the beginning of the loop, $\partial\Lambda$ can only be smaller, and the edge (w', w^*) is removed from E_2, where $w' \in \Lambda$. If the property does not hold, then there is some $w'' \in \partial\Lambda$ such that $E_{w''}$ is not contained in the current E_2. This means that $(w', w^*) \in E_{w''}$, which implies that $w'' \prec w^*$. This contradicts how w^* is chosen.
- Next consider Line 12. By Corollary 7, $\Lambda_1 = \Lambda \cup \{w^*\}$. Note that $\partial\Lambda_1 = \partial\Lambda \cup \Gamma_{\mathrm{out}}(w^*) \setminus \{w^*\}$, where $\Gamma_{\mathrm{out}}(w^*) = \{v \in V_u \setminus \Lambda \mid (w^*, v) \in E_2\}$. The property holds for all vertices in $\partial\Lambda \setminus \{w^*\}$ by the previous case. For $w'' \in \Gamma_{\mathrm{out}}(w^*) \setminus \partial\Lambda$, the removal of the edge (w', w^*) does not affect $G_{w''}$. Thus the property also holds.

The second property holds because $u \prec w$ for all $w \in V_u \setminus \{u\}$. □

In Algorithm 1, each $(\widetilde{R}_w, S_w)$ is computed with respect to G_w. Lemma 16 together with Lemma 15 shows that for every instance of ApproxCount evoked by Algorithm 1, we can reuse all $(\widetilde{R}_w, S_w)$ computed before.

We still need to take care of the case when Algorithm 3 is called by Algorithm 2. This requires a generalised version of Lemma 15. Again, let $G = (V, E)$ be the input of Algorithm 1 and $v_1, v_2, \ldots, v_n \in V$, where $v_1 = s$ and $v_n = t$, be the topological ordering in Algorithm 1. For i from n to 1, Algorithm 1 compute $\widetilde{R}_{v_i}$ and a multi-set S_{v_i} of ℓ random samples step by step. For any fixed i, we view each $(\widetilde{R}_{v_j}, S_{v_j})_{j>i}$ as a random variable following a joint distribution.

Every time when ApproxCount (described in Algorithm 3) is evoked by Algorithm 1 or Algorithm 2, its input includes a subset of vertices $V_0 \subseteq V$ with $t \in V_0$, a subset of edges $E_0 \subseteq E$, a subset of vertices $\Lambda_0 \subseteq V_0$ and $(\widetilde{R}_w, S_w)_{w \in \partial\Lambda_0}$, where $\partial\Lambda_0 = \{w \in V_0 \setminus \Lambda_0 \mid \exists w' \in \Lambda_0 \text{ s.t. } (w', w) \in E_0\}$. For any i, define Φ_i as a set of tuples (V_0, E_0, Λ_0) such that

- $(V_0, E_0, \Lambda_0) \in 2^V \times 2^E \times 2^{V_0}$;
- for all $w \in \partial\Lambda_0$, $E_w \subseteq E_0$, where E_w is the edge set of G_w;
- $\partial\Lambda_0 \subseteq \{v_{i+1}, \ldots, v_n\}$.

By Lemma 16 and the way Algorithm 1 works, Φ_i contains all possible inputs of ApproxCount when we compute $\widetilde{R}_{v_i}$ and S_{v_i} (including the recursive calls). For any $(V_0, E_0, \Lambda_0) \in \Phi_i$, let $R(V_0, E_0, \Lambda_0)$ denote the $\Lambda_0 - t$ reliability in the graph $G_0 = (V_0, E_0)$, where every edge $e \in E_0$ fails independently with probability q_e. Suppose the random tuples $(\widetilde{R}_{v_j}, S_{v_j})_{j>i}$ have been generated by Algorithm 1. If we run ApproxCount on (V_0, E_0, Λ_0) and $(\widetilde{R}_w, S_w)_{w \in \partial\Lambda_0}$ (note that $\partial\Lambda_0 \subseteq \{v_{i+1}, \ldots, v_n\}$), it will return a random number $\widetilde{R}(V_0, E_0, \Lambda_0)$, where the randomness comes from the input randomness of $(\widetilde{R}_{v_j}, S_{v_j})_{j>i}$ and the independent randomness $\mathcal{D}(V_0, E_0, \Lambda_0)$ inside ApproxCount. Our implementation makes sure that any ApproxCount is evoked for every (V_0, E_0, Λ_0) at most once. Hence, the random variable $\widetilde{R}(V_0, E_0, \Lambda_0)$ is uniquely defined.

We have the following generalised version of Lemma 15. Recall that S_w^{ideal} is a set of ℓ independent perfect samples from the distribution π_w.

Lemma 18. *Given the random tuples $(\widetilde{R}_{v_j}, S_{v_j})_{j>i}$ such that the following two conditions are satisfied*

- *for all* $j > i$, $1 - \varepsilon_0 \leqslant \frac{\widetilde{R}_{v_j}}{R_{v_j}} \leqslant 1 + \varepsilon_0$ *for some* $\varepsilon_0 < 1/2$;
- $d_{TV}\left((S_{v_j})_{j>i}, (S_{v_j}^{\mathrm{ideal}})_{j>i}\right) \leqslant \delta_0$.

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

$$\forall(V_0,E_0,\Lambda_0)\in\Phi_i,\quad 1-\varepsilon_0-\frac{6\sqrt{d}}{\sqrt{\ell_0}}\leqslant\frac{\widetilde{R}(V_0,E_0,\Lambda_0)}{R(V_0,E_0,\Lambda_0)}\leqslant 1-\varepsilon_0-\frac{6\sqrt{d}}{\sqrt{\ell_0}}, \tag{25}$$

where the probability is taken over the randomness of $(\widetilde{R}_{v_j},S_{v_j})_{j>i}$ *and the independent randomness of* $\mathcal{D}(V_0,E_0,\Lambda_0)$ *for* $(V_0,E_0,\Lambda_0)\in\Phi_i$.

Proof of Lemma 18. Strictly speaking, all $(\widetilde{R}_{v_j})_{j>i}$ are random variables, and the first condition means that the event for $(\widetilde{R}_{v_j})_{j>i}$ holds with probability 1. We will actually prove a stronger result. Namely, the lemma holds with probability at least $1-\delta_0-|\Phi_i|2^{-B/30}$, where the probability is taken over the randomness of $(S_{v_j})_{j>i}$ and the independent randomness of $\mathcal{D}(V_0,E_0,\Lambda_0)$ for $(V_0,E_0,\Lambda_0)\in\Phi_i$, for any fixed values of $\widetilde{R}_{v_j}$ as long as the first condition is met.

Note that for any $(V_0,E_0,\Lambda_0)\in\Phi_i$, for all $w\in\partial\Lambda_0$, $E_w\subseteq E_0$, where E_w is the edge set of G_w, and $\partial\Lambda_0=\{w\in V_0\setminus\Lambda_0\mid\exists w\in\Lambda_0\text{ s.t. }(w',w)\in E_0\}$, which also implies that w can reach t in the graph (V_0,E_0). The assumption in this lemma implies the assumption in Lemma 15, and the proof of this lemma is similar to the proof of Lemma 15.

Again, the cases where $t\in\Lambda_0$ and where $\partial\Lambda_0=\emptyset$ are trivial. The main case is when $t\notin\Lambda_0$ and $\partial\Lambda_0\neq\emptyset$. We first use $(S^{\text{ideal}}_{v_j})_{j>i}$ to run the algorithm. Let use denote the output of the algorithm by $\widetilde{R}^{\text{ideal}}(V_0,E_0,\Lambda_0)$. By the same argument as the one for Lemma 15, for any $(V_0,E_0,\Lambda_0)\in\Phi_i$, with probability at least $1-2^{-B/30}$,

$$1-\varepsilon_0-\frac{6\sqrt{d}}{\sqrt{\ell_0}}\leqslant\frac{\widetilde{R}^{\text{ideal}}(V_0,E_0,\Lambda_0)}{R(V_0,E_0,\Lambda_0)}\leqslant 1+\varepsilon_0+\frac{6\sqrt{d}}{\sqrt{\ell_0}}.$$

By a union bound over all $(V_0,E_0,\Lambda_0)\in\Phi_i$, we have that with probability at least $1-|\Phi_i|2^{-B/30}$,

$$\forall(V_0,E_0,\Lambda_0)\in\Phi_i,\quad 1-\varepsilon_0-\frac{6\sqrt{d}}{\sqrt{\ell_0}}\leqslant\frac{\widetilde{R}^{\text{ideal}}(V_0,E_0,\Lambda_0)}{R(V_0,E_0,\Lambda_0)}\leqslant 1+\varepsilon_0+\frac{6\sqrt{d}}{\sqrt{\ell_0}}. \tag{26}$$

Then we show the lemma using an optimal coupling between $(S_{v_j})_{j>i}$ and $(S^{\text{ideal}}_{v_j})_{j>i}$. To be more precise, we first sample $(S_{v_j})_{j>i}$ and $(S^{\text{ideal}}_{v_j})_{j>i}$ from their optimal coupling, then by Lemma 3 we have

$$\Pr\left[\forall j>i, S_{v_j}=S^{\text{ideal}}_{v_j}\right]\geqslant 1-\delta_0. \tag{27}$$

Next, we sample all $\mathcal{D}=(\mathcal{D}(V_0,E_0,\Lambda_0))_{(V_0,E_0,\Lambda)\in\Phi_i}$. When we use $(S_{v_j})_{j>i}$ and $\mathcal{D}$ to run ApproxCount on all of $(V_0,E_0,\Lambda)\in\Phi_i$, we obtain an output vector $\widetilde{R}=(\widetilde{R}(V_0,E_0,\Lambda_0))_{(V_0,E_0,\Lambda)\in\Phi_i}$. Similarly, denote by $\widetilde{R}^{\text{ideal}}=(\widetilde{R}^{\text{ideal}}(V_0,E_0,\Lambda_0))_{(V_0,E_0,\Lambda)\in\Phi_i}$ the output vector when we use $(S^{\text{ideal}}_{v_j})_{j>i}$ and $\mathcal{D}$ to run ApproxCount. Define two good events

- A_1: $\widetilde{R}^{\text{ideal}}=\widetilde{R}$. By (27), $\Pr[A_1]\geqslant 1-\delta_0$;
- A_2: (26) holds for $\widetilde{R}^{\text{ideal}}$. We know $\Pr[A_2]\geqslant 1-|\Phi_i|2^{-B/30}$.

If both A_1 and A_2 occur, then (25) holds. The probability is

$$\Pr[A_1\wedge A_2]=1-\Pr[\overline{A_1}\vee\overline{A_2}]\geqslant 1-\Pr[\overline{A_1}]-\Pr[\overline{A_2}]\geqslant 1-\delta_0-|\Phi_i|2^{-B/30}. \qquad\square$$

Note that Lemma 18 cannot be obtained by simply applying Lemma 15 with a union bound, as that will result in a failure probability of $|\Omega_i|\left(\delta_0+2^{-B/30}\right)$ instead of $\delta_0+|\Omega_i|\,2^{-B/30}$. This is crucial to the efficiency of our algorithm.

4.3. **Analyze the main algorithm.** Now, we are ready to put everything together and analyze the whole algorithm. Recall that we use n to denote the number of vertices in the input graph and $m \geqslant n-1$ the number of edges.

We will need a simple lemma.

Lemma 19. *Let X be a random variable over some finite state space Ω. Let $E \subseteq \Omega$ be an event that occurs with positive probability. Let Y be the random variable X conditional on E. Then,*

$$d_{TV}(X, Y) \leqslant \Pr[\overline{E}].$$

Proof. We couple X and Y as follows: (1) first sample an indicator variable whether the event E occurs; (2) if E occurs, couple X and Y perfectly; and (3) if E does not occur, independently sample X conditional of $\overline{E}$ and sample Y. By Lemma 3,

$$d_{\mathrm{TV}}(X, Y) \leqslant \Pr[X \neq Y] \leqslant \Pr[\overline{E}]. \qquad \square$$

The main goal of this section is to prove the following lemma. In the next lemma, we consider a variant of Algorithm 1, where we replace the subroutine Sample with the subroutine NewSample in Definition 12. Observation 13 shows that Sample and NewSample have the same output distribution. Hence, the replacement does not change the distributions of $\widetilde{R}_{v_i}$, $\widetilde{R}(V_0, E_0, \Lambda_0)$ and S_{v_i} for all $1 \leqslant i \leqslant n$ and all $(V_0, E_0, \Lambda_0) \in \Phi_i$. The only difference is that this variant of Algorithm 1 cannot be implemented in polynomial time. However, we only use the variant algorithm to analyze the approximation error of Algorithm 1. The running time of Algorithm 1 is analyzed separately in the proof of Theorem 1.

Lemma 20. *For any $1 \leqslant i \leqslant n$, there exists a good event $\mathcal{A}(i)$ such that $\mathcal{A}(i)$ occurs with probability at least $1 - \frac{n-i}{10n}$ and conditional on $A(i)$, it holds that*

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

$$d_{TV}\left((S_{v_j})_{j \geqslant i}, (S_{v_j}^{\mathrm{ideal}})_{j \geqslant i}\right) \leqslant 2^{-4m}(2^{n-i} - 1); \tag{28}$$

- *the following event occurs:*

$$\forall (V_0, E_0, \Lambda_0) \in \Phi_i, \qquad 1 - \frac{(n-i)\varepsilon}{50mn} \leqslant \frac{\widetilde{R}(V_0, E_0, \Lambda_0)}{R(V_0, E_0, \Lambda_0)} \leqslant 1 + \frac{(n-i)\varepsilon}{50mn}, \tag{29}$$

  and denote this event by $\mathcal{B}(i)$.

In particular, the event $\mathcal{B}(i)$ implies that,

$$1 - \frac{(n-i)\varepsilon}{50nm} \leqslant \frac{\widetilde{R}_{v_i}}{R_{v_i}} \leqslant 1 + \frac{(n-i)\varepsilon}{50nm}. \tag{30}$$

Proof. We first show that $\mathcal{B}(i)$ implies (30). If $i = n$, then $\widetilde{R}_{v_i} = R_{v_i}$ and (30) holds. For $i < n$, $\widetilde{R}_{v_i}$ is computed in Line 3 of Algorithm 1, where the input $(V_0, E_0, \Lambda_0) \in \Phi_i$. Hence, $\mathcal{B}(i)$ implies (30).

Next, we construct the event $\mathcal{A}(i)$ inductively from $i = n$ to $i = 1$ and prove (28) and (29). The base case is $i = n$, where $v_n = t$. In this case, the only possible sample in S_{v_n} is the empty graph with one vertex t, and thus (28) holds. Also note that if $(V_0, E_0, \Lambda_0) \in \Phi_n$, then $\partial\Lambda_0 = \emptyset$. By Line 1 in Algorithm 3, if $t \in \Lambda$, the output is exactly 1, and if $t \notin \Lambda$, the output is exactly 0. In either case, (29) holds. Hence, we simply let $\mathcal{A}(n)$ be the empty event, occurring with probability 1.

For $i < n$, we inductively define

$$\mathcal{A}(i) := \mathcal{A}(i+1) \wedge \mathcal{B}(i) \wedge \mathcal{C}(i),$$

where the event $\mathcal{C}(i)$ is defined next. Recall that Algorithm 1 calls Sample ℓ times to generate a multi-set of ℓ samples in S_{v_i}. By Observation 13, Sample and NewSample (Definition 12) have the same output distribution. For analysis purposes, suppose we call NewSample instead ℓ times to generate a multi-set of ℓ samples. In (20), we defined an event $\mathcal{C}$ for one instance of NewSample. Since we have ℓ of them, define

$$\mathcal{C}(i) : \text{for all } \ell \text{ calls of NewSample, the event } \mathcal{C} \text{ occurs.}$$

Clearly $\mathcal{A}(i)$ implies $\mathcal{B}(i)$ and (30). Before we show that $\mathcal{A}(i)$ implies (28), we first lower bound the probability of $\mathcal{A}(i)$ by $1-\frac{n-i}{10n}$. By the induction hypothesis, since $\mathcal{A}(i)$ implies $\mathcal{A}(i+1)$, conditional on $\mathcal{A}(i)$, we have

$$d_{TV}\left((S_{v_j})_{j>i}, (S^{\text{ideal}}_{v_j})_{j>i}\right) \leqslant 2^{-4m}(2^{n-i-1}-1). \tag{31}$$

In fact $\mathcal{A}(i+1)$ implies $\mathcal{A}(j)$ for all $j \geqslant i+1$. Thus, by (30),

$$\forall j \geqslant i+1, \quad 1-\frac{(n-i-1)\varepsilon}{50nm} \leqslant \frac{\widetilde{R}_{v_j}}{R_{v_j}} \leqslant 1+\frac{(n-i-1)\varepsilon}{50nm}, \tag{32}$$

as the worst case of the bound is when $j = i+1$. Combining (31), (32) and Lemma 18, with probability at least $1-2^{-4m}(2^{n-i-1}-1)-|\Phi_i|2^{-B/30}$, it holds that

$$\forall (V_0,E_0,\Lambda_0) \in \Phi_i, \quad 1-\frac{(n-i-1)\varepsilon}{50nm}-\frac{6\sqrt{d}}{\sqrt{\ell_0}} \leqslant \frac{\widetilde{R}(V_0,E_0,\Lambda_0)}{R(V_0,E_0,\Lambda_0)} \leqslant 1+\frac{(n-i-1)\varepsilon}{50nm}+\frac{6\sqrt{d}}{\sqrt{\ell_0}}. \tag{33}$$

Note that $d \leqslant n$. Also recall that $\ell_0 = \left\lceil \frac{10^5 n^3 m^2}{\varepsilon^2} \right\rceil$ and $B = 60n+150m$. We have

$$\frac{(n-i-1)\varepsilon}{50nm}+\frac{6\sqrt{d}}{\sqrt{\ell_0}} \leqslant \frac{(n-i)\varepsilon}{50nm}, \tag{34}$$

which means that (33) implies $\mathcal{B}(i)$. By Lemma 18, we have

$$\Pr[\mathcal{B}(i) \mid \mathcal{A}(i+1)] \geqslant 1-|\Phi_i|2^{-B/30}-2^{-4m}(2^{n-i-1}-1) \tag{35}$$

$$\geqslant 1-2^{-4m}-2^{-4m}(2^{n-i-1}-1) \geqslant 1-2^{-n}, \tag{36}$$

where we used the fact that $|\Phi_i| \leqslant 2^{m+2n}$.

Given $\mathcal{A}(i+1) \wedge \mathcal{B}(i)$, (30) also holds. The ℓ samples in S_{v_i} are generated by NewSample on the graph G_{v_i}. In Line 11 and Line 12 of Algorithm 2 (which are also in NewSample), Algorithm 3 is evoked to compute the value of c_0 and c_1. By Lemma 16, the input $(V_0,E_0,\Lambda_0) \in \Phi_i$. Also, by (30), $\widetilde{R}_{v_i}$ approximates R_{v_i}. Hence the subroutine Algorithm 3 behaves like the oracle $\mathcal{P}$ assumed in Lemma 14, satisfying conditions (15), (16), and (17). By the definition of $\mathcal{C}(i)$ and Lemma 14, it is independent from $\mathcal{A}(i+1) \wedge \mathcal{B}(i)$ because $\mathcal{C}(i)$ depends only on the independent randomness inside NewSample. By a union bound over ℓ calls of NewSample, we have

$$\Pr[\mathcal{C}(i) \mid \mathcal{A}(i+1) \wedge \mathcal{B}(i)] \geqslant 1-\frac{\varepsilon^{200}\ell}{n^{200}} \geqslant 1-\frac{1}{10n^2}, \quad \text{where } \ell = (60n+150m)\left\lceil \frac{10^5 n^3 m^2}{\varepsilon^2} \right\rceil. \tag{37}$$

By the induction hypothesis, (36), and (37),

$$\Pr[\mathcal{A}(i)] = \Pr[\mathcal{A}(i+1) \wedge \mathcal{B}(i) \wedge \mathcal{C}(i)] \geqslant \left(1-\frac{n-i-1}{10n}\right)\left(1-\frac{1}{10n^2}\right)\left(1-2^{-n}\right) \geqslant 1-\frac{n-i}{10n}.$$

We still need to show that $\mathcal{A}(i)$ implies (28). We use $(S_{v_j})_{j>i}|_{\mathcal{A}(i+1)}$ to the denote the random samples of $(S_{v_j})_{j>i}$ conditional on $\mathcal{A}(i+1)$. Similarly, we can define $((\widetilde{R}_{v_j})_{j>i}, (S_{v_j})_{j>i}, \mathcal{D})|_{\mathcal{A}(i+1)}$, where $\mathcal{D} = (\mathcal{D}(V_0,E_0,\Lambda_0))_{(V_0,E_0,\Lambda_0)\in\Phi_i}$, and $((\widetilde{R}_{v_j})_{j\geqslant i}, (S_{v_j})_{j\geqslant i}, \mathcal{D})|_{\mathcal{A}(i+1)\wedge\mathcal{B}(i)}$, where we further condition on $\mathcal{B}(i)$. Note that the event $\mathcal{B}(i)$ is determined by the random variables $((\widetilde{R}_{v_j})_{j>i}, (S_{v_j})_{j>i}, \mathcal{D})$. By letting E be $\mathcal{B}(i)$ conditional on $\mathcal{A}(i+1)$ in Lemma 19, we have that

$$\begin{aligned}
&d_{TV}\left(((\widetilde{R}_{v_j})_{j>i}, (S_{v_j})_{j>i}, \mathcal{D})|_{\mathcal{A}(i+1)\wedge\mathcal{B}(i)}, ((\widetilde{R}_{v_j})_{j>i}, (S_{v_j})_{j>i}, \mathcal{D})|_{\mathcal{A}(i+1)}\right)\\
&\leqslant 1 - \Pr[\mathcal{B}(i) \mid \mathcal{A}(i+1)]\\
\text{(by (35))}\quad &\leqslant |\Phi_i| 2^{-B/30} + 2^{-4m}(2^{n-i-1}-1) \leqslant 2^{-4m} + 2^{-4m}(2^{n-i-1}-1).
\end{aligned}$$

Projecting to $(S_{v_j})_{j>i}$ we have

$$d_{TV}\left((S_{v_j})_{j>i}|_{\mathcal{A}(i+1)\wedge\mathcal{B}(i)}, (S_{v_j})_{j>i}|_{\mathcal{A}(i+1)}\right) \leqslant 2^{-4m} + 2^{-4m}(2^{n-i-1}-1).$$

By the induction hypothesis, it holds that

$$d_{TV}\left((S_{v_j})_{j>i}|_{\mathcal{A}(i+1)}, (S^{\text{ideal}}_{v_j})_{j>i}\right) \leqslant 2^{-4m}(2^{n-i-1}-1).$$

Using the triangle inequality for total variation distances, we have

$$d_{TV}\left((S_{v_j})_{j>i}|_{\mathcal{A}(i+1)\wedge\mathcal{B}(i)}, (S^{\text{ideal}}_{v_j})_{j>i}\right) \leqslant 2^{-4m} + 2^{-4m}(2^{n-i-1}-1) \times 2 = 2^{-4m}(2^{n-i}-1).$$

Given $\mathcal{A}(i+1)\wedge\mathcal{B}(i)$, (30) also holds. The ℓ samples in S_{v_i} are generated by NewSample on the graph G_{v_i}. By Lemma 14, in that case, if $\mathcal{C}(i)$ occurs, S_{v_i} contains ℓ prefect independent samples. Furthermore, the event $\mathcal{C}(i)$ is independent from $(S_{v_j})_{j>i}$ (as by Lemma 14, $\mathcal{C}(i)$ depends only on the internal independent randomness of NewSample[6]). Hence,

$$\begin{aligned}
d_{TV}\left((S_{v_j})_{j\geqslant i}|_{\mathcal{A}(i)}, (S^{\text{ideal}}_{v_j})_{j\geqslant i}\right) &= d_{TV}\left((S_{v_j})_{j>i}|_{\mathcal{A}(i)}, (S^{\text{ideal}}_{v_j})_{j>i}\right)\\
&= d_{TV}\left((S_{v_j})_{j>i}|_{\mathcal{A}(i+1)\wedge\mathcal{B}(i)}, (S^{\text{ideal}}_{v_j})_{j>i}\right) \leqslant 2^{-4m}(2^{n-i}-1). \quad \square
\end{aligned}$$

We remark that the set S_{v_i} may be used multiple times throughout Algorithm 1. In particular, this means that there may be subtle correlation among $\widetilde{R}_i$'s. These correlations do not affect our approximation guarantee. This is because the conditions of Lemma 15 and Lemma 18 only involve marginals. Namely, as long as the marginals are in the suitable range, the correlation amongst them does not matter.

By (30) of Lemma 20 with $i = 1$, note that $v_1 = s$, we have

$$\Pr\left[1 - \frac{\varepsilon}{50m} \leqslant \frac{\widetilde{R}_s}{R_s} \leqslant 1 + \frac{\varepsilon}{50m}\right] \geqslant \Pr[\mathcal{A}(1)] \geqslant \frac{3}{4}.$$

Note that the events $\mathcal{A}(i)$ for $1 \leqslant i \leqslant n$ are defined for the variant of Algorithm 1, where we replace Sample with NewSample. By Observation 13, the variant and Algorithm 1 have the same output distribution. Hence, for the original Algorithm 1, we also have

$$\Pr\left[1 - \frac{\varepsilon}{50m} \leqslant \frac{\widetilde{R}_s}{R_s} \leqslant 1 + \frac{\varepsilon}{50m}\right] \geqslant \frac{3}{4}. \tag{38}$$

Proof of Theorem 1. The approximation guarantee follows directly from (38). Note that the guarantee is stronger than $1 \pm \varepsilon$ approximation, and this is because we need the estimates to be sufficiently accurate to control the error for the sampling subroutine.

We analyze the running time next. Recall that n is the number of vertices of the input graph and $m \geqslant n-1$ is number of edges in G. Recall

$$\ell = (60n + 150m)\left\lceil \frac{10^5 n^3 m^2}{\varepsilon^2} \right\rceil = O\left(\frac{n^3 m^3}{\varepsilon^2}\right).$$

[6] In fact, $(S_{v_j})_{j>i}$ is correlated with $\mathcal{X}_{\mathcal{P}}$ in Lemma 14 but independent from $\mathcal{D}_u$.

By Lemma 15, the running time of ApproxCount (Algorithm 3) is at most

$$T_{\mathsf{count}} = O(mn\ell) = O\left(\frac{n^4 m^4}{\varepsilon^2}\right).$$

By Lemma 11, the running time of Sample (Algorithm 2) is at most

$$T_{\mathsf{sample}} = \tilde{O}((n+m)T_{\mathsf{count}}) = \tilde{O}(mT_{\mathsf{count}}) = \tilde{O}\left(\frac{n^4 m^5}{\varepsilon^2}\right).$$

Hence, the running time of Algorithm 1 is

$$T = O\big(n\big(T_{\mathsf{count}} + \ell T_{\mathsf{sample}}\big)\big) = O\big(n\ell T_{\mathsf{sample}}\big) = \tilde{O}\left(\frac{n^8 m^8}{\varepsilon^4}\right). \qquad \square$$

Equation (38) actually guarantees a stronger condition

$$1 - \frac{\varepsilon}{50m} \leqslant \frac{\widetilde{R}_{\nu_1}}{R_{\nu_1}} \leqslant 1 + \frac{\varepsilon}{50m},$$

while Theorem 1 only requires $1-\varepsilon \leqslant \frac{\widetilde{R}_{\nu_1}}{R_{\nu_1}} \leqslant 1+\varepsilon$. But the error bound in (29) is necessary for our analysis, because we require Algorithm 3 to implement the oracle $\mathcal{P}$ in Lemma 11. Hence, the bottleneck of the running time is to generate random samples.

Next, we prove that if the input $\varepsilon = O(1/m)$, then we can improve the running time in Theorem 1.

Proof of Theorem 2. Suppose $\varepsilon = \frac{\alpha}{m}$ for some $\alpha = O(1)$. Then we can change parameters ℓ, ℓ_0, B to

$$\ell_0 = \left\lceil \frac{10^5 \max\{\alpha^2, 1\} n^3}{\varepsilon^2} \right\rceil, \quad B = 60n + 150m \text{ and } \ell = \ell_0 B. \tag{39}$$

We can change the event $\mathcal{B}(i)$ defined by (29) in Lemma 20 to

$$\forall (V_0, E_0, \Lambda_0) \in \Phi_i, \qquad 1 - \frac{(n-i)\varepsilon}{50\max\{\alpha,1\}n} \leqslant \frac{\widetilde{R}(V_0,E_0,\Lambda_0)}{R(V_0,E_0,\Lambda_0)} \leqslant 1 + \frac{(n-i)\varepsilon}{50\max\{\alpha,1\}n}. \tag{40}$$

The inequality above implies

$$1 - \frac{(n-i)\varepsilon}{50\max\{\alpha,1\}n} \leqslant \frac{\widetilde{R}_{\nu_i}}{R_{\nu_i}} \leqslant 1 + \frac{(n-i)\varepsilon}{50\max\{\alpha,1\}n}.$$

In particular, $1-\varepsilon \leqslant \frac{\widetilde{R}_{\nu_1}}{R_{\nu_1}} \leqslant 1+\varepsilon$, which is our approximation guarantee.

We need to verify that Lemma 20 with this new definition of $\mathcal{C}(i)$ holds under the parameter setting (39). Note that the following two points holds.

- In the inductive proof, since $\varepsilon = \frac{\alpha}{m}$, we always have for all $(V_0, E_0, \Lambda_0) \in \Phi_i$, $1 - \frac{1}{50m} \leqslant \frac{\widetilde{R}(V_0,E_0,\Lambda_0)}{R(V_0,E_0,\Lambda_0)} \leqslant 1 + \frac{1}{50m}$, which means that the subroutine Algorithm 3 behaves like the oracle $\mathcal{P}$ assumed in Lemma 11 and we can still use Lemma 11 in the proof.
- The inequality (34) can be replaced with $\frac{(n-i-1)\varepsilon}{50\max\{\alpha,1\}n} + \frac{6\sqrt{d}}{\sqrt{\ell_0}} \leqslant \frac{(n-i)\varepsilon}{50\max\{\alpha,1\}n}$.

We can go through the original proof to verify Lemma 20.

We obtain a better running time because ℓ in (39) is smaller. Specifically,

$$\begin{aligned}
\ell &= O\left(\frac{n^3 m}{\varepsilon^2}\right), \\
T_{\mathsf{count}} &= O(mn\ell) = O\left(\frac{n^4 m^2}{\varepsilon^2}\right), \\
T_{\mathsf{sample}} &= \tilde{O}((n+m)T_{\mathsf{count}}) = \tilde{O}\left(\frac{n^4 m^3}{\varepsilon^2}\right), \text{ and} \\
T &= O\left(n\left(T_{\mathsf{count}} + \ell T_{\mathsf{sample}}\right)\right) = \tilde{O}\left(\frac{n^8 m^4}{\varepsilon^4}\right).
\end{aligned}$$

This proves the theorem. □

Acknowledgement

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. 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).

References

[ACJR21] 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. 2

[ÀJ93] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993. 2

[ALO+21] 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. 1

[ALOV19] 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. 1

[AvBM23] Antoine Amarilli, Timothy van Bremen, and Kuldeep S. Meel. Conjunctive queries on probabilistic graphs: the limits of approximability. *arXiv*, abs/2309.13287, 2023. 2, 3, 26

[Bal80] Michael O. Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980. 1

[Bal86] Michael O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Trans. Rel.*, 35(3):230–239, 1986. 1

[BP83] Michael O. Ball and J. Scott Provan. Calculating bounds on reachability and connectedness in stochastic networks. *Networks*, 13(2):253–278, 1983. 1

[CGM21] Mary Cryan, Heng Guo, and Giorgos Mousa. Modified log-Sobolev inequalities for strongly log-concave distributions. *Ann. Probab.*, 49(1):506–525, 2021. 1

[CGZZ23] Xiaoyu Chen, Heng Guo, Xinyuan Zhang, and Zongrui Zou. Near-linear time samplers for matroid independent sets with applications. *arXiv*, abs/2308.09683, 2023. 1

[Col87] Charles J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987. 1

[GH20] Heng Guo and Kun He. Tight bounds for popping algorithms. *Random Struct. Algorithms*, 57(2):371–392, 2020. 1

[GJ19] Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM J. Comput.*, 48(3):964–978, 2019. 1

[GJL19] Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *J. ACM*, 66(3):18:1–18:31, 2019. 1

[HS18] David G. Harris and Aravind Srinivasan. Improved bounds and algorithms for graph cuts and network reliability. *Random Struct. Algorithms*, 52(1):74–135, 2018. 1

[Jer81] Mark Jerrum. *On the complexity of evaluating multivariate polynomials*. PhD thesis, The University of Edinburgh, 1981. 1

[JVV86] 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. 2, 5

[Kar99] 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. 1

[Kar20] David R. Karger. A phase transition and a quadratic time unbiased estimator for network reliability. In *STOC*, pages 485–495. ACM, 2020. 1

[KL83] Richard M. Karp and Michael Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64. IEEE Computer Society, 1983. 2

[KL85] Richard M. Karp and Michael Luby. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *J. Complex.*, 1(1):45–64, 1985. 2

[KLM89] Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989. 2

[PB83] 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. 1

[Pro86] J. Scott Provan. The complexity of reliability computations in planar and acyclic graphs. *SIAM J. Comput.*, 15(3):694–702, 1986. 1

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. 1

[ZL11] Rico Zenklusen and Marco Laumanns. High-confidence estimation of small *s*-*t* reliabilities in directed acyclic networks. *Networks*, 57(4):376–388, 2011. 1, 2

Appendix A. An alternative way to define the event $\mathcal{C}$

We start from the following abstract setting. Let $A \sim \mu_A$ and $B \sim \mu_B$ be two random variables over some state space Ω. Suppose for any $x \in \Omega$, it holds that

$$\mu_A(x) \geqslant (1-\varepsilon)\mu_B(x),$$

for some $0 \leqslant \varepsilon < 1$. Then, the distribution μ_A can be rewritten as

$$\mu_A = (1-\varepsilon)\mu_B + \varepsilon\nu,$$

where the distribution ν is defined by

$$\forall x \in \Omega, \quad \nu(x) = \frac{\mu_A(x) - (1-\varepsilon)\mu_B(x)}{\varepsilon}.$$

Then, we can draw a sample $A \sim \mu_A$ using the following procedure.

- Flip a coin with probability of HEADS being $1-\varepsilon$;
- If the outcome is HEADS, draw $A \sim \mu_B$;
- If the outcome is TAILS, draw $A \sim \nu$.

In this procedure, we can define an event $\mathcal{C}$ as "the outcome of the coin flip is HEADS". We know that conditional on $\mathcal{C}$, the distribution of A is μ_B. Such an event $\mathcal{C}$ is defined in an expanded space $\Omega \times \{\text{HEADS}, \text{TAILS}\}$.

Consider the modified version of Sample, where Sample is defined in Algorithm 2. Suppose we use it on π_u, where $u = v_k$. Let $\mathcal{X}_\mathcal{P}$ be the random variable associated with the oracle $\mathcal{P}$. Denote the distribution of $\mathcal{X}_\mathcal{P}$ by $\mu_\mathcal{P}$. Sample uses independent inside randomness $\mathcal{D}_u$ to generate $H = H(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)$ and $F = F(\mathcal{X}_\mathcal{P}, \mathcal{D}_u)$. Define

$$\begin{aligned} \mu_A: &\quad \text{the joint distribution of } \mathcal{X}_\mathcal{P} \text{ and } H, \\ \mu_B: &\quad \text{the product distribution of } \mu_\mathcal{P} \text{ and } \pi_u. \end{aligned}$$

For any $x_\mathcal{P}$ in the support of $\mathcal{X}_\mathcal{P}$ and any h in the support of π_u, we have

$$\frac{\mu_A(x_\mathcal{P}, h)}{\mu_B(x_\mathcal{P}, h)} = \frac{\Pr[\mathcal{X}_\mathcal{P} = x_\mathcal{P}]\Pr[H = h \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}]}{\Pr[\mathcal{X}_\mathcal{P} = x_\mathcal{P}]\pi_u(h)} = \frac{\Pr[H = h \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}]}{\pi_u(h)}.$$

Note that

$$\Pr[H = h \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}] \geqslant \Pr[F = 1 \mid \mathcal{X}_\mathcal{P} = x_\mathcal{P}]\Pr[H = h \mid F = 1 \land \mathcal{X}_\mathcal{P} = x_\mathcal{P}] \geqslant (1-\varepsilon)\pi_u(h),$$

where $\varepsilon = 1 - 1/n^{200}$. This implies

$$\frac{\mu_A(x_\mathcal{P}, h)}{\mu_B(x_\mathcal{P}, h)} \geqslant 1 - \varepsilon.$$

Using the above abstract result, we can define an equivalent process of Sample and find an event $\mathcal{C}$ such that $\Pr[\mathcal{C}] \geqslant 1 - \varepsilon$ and conditional on $\mathcal{C}$, $(\mathcal{X}_{\mathcal{P}}, H) \sim \mu_B$. By the definition of μ_B, we know that conditional on $\mathcal{C}$, $\mathcal{X}_{\mathcal{P}} \sim \mu_{\mathcal{P}}$ still follows the distribution specified by the oracle $\mathcal{P}$, which means that $\mathcal{C}$ is independent from $\mathcal{X}_{\mathcal{P}}$. And also, H is a perfect independent sample from π_u. Finally, we remark that in our analysis (see Observation 13 and Lemma 20), we only need to show that such an equivalent process and the event $\mathcal{C}$ exist. We do not need to implement the process nor certify the event in the algorithm.

Appendix B. Reducing counting $s-t$ connected subgraphs in DAGs to #NFA

Given a graph $G = (V, E)$ with a source s and a sink t, the set of $s-t$ connected subgraphs are $\{H = (V, E_H) \mid E_H \subseteq E \text{ s.t. } s \rightsquigarrow_H t\}$. Let $m := |E|$. Counting $s-t$ connected subgraphs is equivalent to computing the $s-t$ reliability of the same graph with $q_e = 1/2$ for all $e \in E$. In this section, we reduce counting $s-t$ connected subgraphs in DAGs to #NFA. This reduction is essentially the same as the one in [AvBM23], where it is not explicitly given.

Given a DAG G, we construct an NFA A_G such that the number of its accepting strings is the same as the number of $s-t$ connected subgraphs in G. The states of A_G consists of the starting state s, all edges, the accepting state t, a failure state, and some auxiliary states. We order all edges in G according to the head of the edge's topological order, say $e_1, \dots, e_m$. In particular, this means that if $f_1, \dots, f_k$ form a path, then $f_1 \prec f_2 \cdots \prec f_k$. Moreover, we want to connect s and t to their respective adjacent edges, and two edges if they share an endpoint. However, we want each bit of the input string to correspond to whether to have an edge or not, which implies that we need to absorb all intermediate inputs. Thus, instead, to connect e_i to e_j with $i < j$, we add auxiliary states $f_k^{(i,j)}$ from $k = i+1$ to $k = j-1$. We connect e_i to $f_{i+1}^{(i,j)}$, $f_{i+1}^{(i,j)}$ to $f_{i+2}^{(i,j)}$, etc., labelled with both 0 and 1. Lastly, we connect $f_{j-1}^{(i,j)}$ to e_j, labelled with only 1, and we connect $f_{j-1}^{(i,j)}$ to the failure state, labelled with 0. Once we are in the failure state, it can only move to itself, namely it has only a self-loop labelled with both 0 and 1. We also do the same as above by treating s as e_0 (whose tail is s and head does not matter) and t as e_{m+1} (whose head is t and tail does not matter). Note that there are $O(m^2)$ states and we are counting accepting strings of length $m+1$. The last bit of any accepting string has to be 1, and therefore each accepting string is an indicator vector for a subset of edges. It is easy to verify that the string is accepted if and only if s can reach t in the corresponding subgraph. This finishes the reduction.