TIGHT BOUNDS FOR POPPING ALGORITHMS

HENG GUO AND KUN HE

Abstract. We sharpen run-time analysis for algorithms under the partial rejection sampling framework. Our method yields improved bounds for

- the cluster-popping algorithm for approximating all-terminal network reliability;
- the cycle-popping algorithm for sampling rooted spanning trees;
- the sink-popping algorithm for sampling sink-free orientations.

In all three applications, our bounds are not only tight in order, but also optimal in constants.

1. Introduction

The counting complexity class #**P** was introduced by Valiant [Val79] to capture the apparent intractability of the permanent. Although exactly evaluating #**P**-complete problems is a task even harder than solving **NP**-complete problems [Tod91], efficient approximation algorithms may still exist. This possibility was first exhibited through the relationship between approximate counting and sampling [JVV86], and in particular, using the Markov chain Monte Carlo (MCMC) method [Jer03]. Early successes along this line include efficient algorithms to approximate the volume of a convex body [DFK91], to approximate the partition function of ferromagnetic Ising models [JS93], and to approximate the permanent of a non-negative matrix [JSV04]. More recently, a number of exciting new approaches were developed, utilising a variety of tools such as correlation decay [Wei06, BG08], zeros of graph polynomials [Bar16, PR17], and Lovász local lemma [Moi17].

Partial rejection sampling [GJL17] is yet another alternative approach to approximate counting and exact sampling. It takes an algorithmic Lovász local lemma [MT10] perspective for Wilson's cycle-popping algorithm [PW98] to sample rooted spanning trees. The algorithm is extremely simple. In order to sample from a product distribution conditional on avoiding a number of undesirable "bad" events, we randomly initialise all variables, and re-sample a subset of variables selected by a certain subroutine, until no bad event is present. In the so-called extremal instances, the resampled variables are just those involved in occurring bad events. Despite its simplicity, it can be applied to a number of situations that are seemingly unrelated to the local lemma. Unlike Markov chains, partial rejection sampling yields exact samplers. The most notable application is the first polynomial-time approximation algorithm for all-terminal network reliability [GJ18c].

In this paper we sharpen the run-time analysis for a number of algorithms under the partial rejection sampling framework for extremal instances. We apply our method to analyse cluster-, cycle-, and sink-popping algorithms. Denote by n the number of vertices in a graph, and m the number of edges (or arcs) in a undirected (or directed) graph. We summarise some background and our results below.

- Cluster-popping is first proposed by Gorodezky and Pak [GP14] to approximate a network reliability measure, called reachability, in directed graphs. They conjecture that the algorithm runs in polynomial-time in expectation on bi-directed graphs, which is confirmed by Guo and Jerrum [GJ18c]. It has also been shown in [GJ18c] that a polynomial-time approximation algorithm for all-terminal network reliability can be obtained using cluster-popping.

  We show a $\frac{p_{\max}}{1-p_{\max}}mn$ upper bound for the expected number of resampled variables on bi-directed graphs, where $p_{\max}$ is the maximum failure probability on edges. This

improves the previous $\frac{p_{\max}}{1-p_{\max}}m^2n$ bound [GJ18c]. We also provide an efficient implementation based on Tarjan's algorithm [Tar72] to obtain a faster algorithm to sample spanning connected subgraphs in $O(mn)$ time (assuming that $p_{\max}$ is a constant). Furthermore, we obtain a faster approximation algorithm for the all-terminal network reliability, which takes $O(mn^2 \log n)$ time, improving upon $O(m^2n^3)$ from [GJ18c]. For precise statements, see Theorem 6 and Theorem 13.

- Cycle-popping is introduced by Propp and Wilson [Wil96, PW98] to sample uniformly rooted spanning trees, a problem with a long line of history. We obtain a $2mn$ upper bound for the expected number of resampled variables, improving the constant from the previous $O(mn)$ upper bound [PW98]. See Theorem 15.
- Sampling sink-free orientations is introduced by Bubley and Dyer [BD97a] to show that the number of solutions to a special class of CNF formulas is #**P**-hard to count exactly, but can be counted in polynomial-time approximately. Bubley and Dyer showed that the natural Markov chain takes $O(m^3 \log 1/\varepsilon)$ time to generate a distribution ε-close (in total variation distance) to the uniform distribution over all sink-free orientations. Using the coupling from the past technique, Huber [Hub98] obtained a $O(m^4)$ exact sampler. Cohn, Pemantle, and Propp [CPP02] introduced an alternative exact sampler, called sink-popping. They show that sink-popping resamples $O(mn)$ random variables in expectation.

  We improve the expected number of resampled variables for sink-popping to at most $n(n-1)$. See Theorem 19.

In all three applications, we also construct examples to show that none of the constants (if explicitly stated) can be further improved. Our results yield best known running time for all problems mentioned above except sampling uniform spanning trees, for which the current best algorithm by Schild [Sch18] is in almost-linear time in m. We refer interested readers to the references therein for the vast literature on this problem. One should note that the corresponding counting problem for spanning trees is tractable via Kirchhoff's matrix-tree theorem, whereas for the other two exact counting is #**P**-hard [Jer81, PB83, BD97a]. It implies that their solution spaces are less structured, and renders sampling much more difficult than for spanning trees.

The starting point of our method is an exact formula [GJL17, Theorem 13] for the expected number of events resampled, for partial rejection sampling algorithm on extremal instances. Informally, it states that the expected number of resampled events equals to the ratio between the probability (under the product distribution) that exactly one bad event happens, and the probability that none of the bad events happens. This characterisation has played an important role in the confirmation of the conjecture by Gorodezky and Pak [GP14], which leads to the aforementioned all-terminal network reliability algorithm [GJ18c].

When bad events involve only constant number of variables, bounding this ratio is sufficient to obtain tight run-time bound. However, in all three popping algorithms mentioned above, bad events can involve as many variables as m or n, and simply applying a worst case bound (such as m or n) yields loose run-time upper bound. Our improvement comes from a refined expression of the number of variables, rather than events, resampled in expectation. (See (2) and Theorem 3.) We then apply a combinatorial encoding idea, and design injective mappings to bound the refined expression. Similar mappings have been obtained before in [GJL17, GJ18c], but our new mappings are more complicated and carefully designed in order to achieve tight bounds. We note that our analysis is completely different and significantly simpler than the original analysis for cycle-popping [PW98] and sink-popping [CPP02].

Since our bounds are tight, one has to go beyond partial rejection sampling to further accelerate cluster- and sink-popping. It remains an interesting open question whether $O(mn)$ is a barrier to uniformly sample spanning connected subgraphs.

2. Partial rejection sampling

Let $\{X_1, \ldots, X_n\}$ be a set of mutually independent random variables. Each X_i can have its own distribution and range. Let $\{A_1, \ldots, A_m\}$ be a set of "bad" events that depend on X_i's. For

example, for a constraint satisfaction problem (CSP) with variables X_i ($i \in [n]$) and constraints C_j ($j \in [m]$), each A_j is the set of unsatisfying assignments of C_j for $j \in [m]$. Let $\mathsf{var}(A_j)$ be the set of variables that A_j depends on.

Our goal is to sample from the product distribution of $(X_i)_{i\in[n]}$, conditional on none of the bad events $(A_j)_{j\in[m]}$ occurring. Denote by $\mu(\cdot)$ the product distribution and by $\pi(\cdot)$ the conditional one (which is our target distribution).

A breakthrough result in algorithmic Lovász local lemma is the Moser and Tardos algorithm [MT10], which simply iteratively eliminates occurring bad events. The form we will use is described in Algorithm 1.

Algorithm 1: Partial rejection sampling for extremal instances

Draw independent samples of all variables $X_1, \dots, X_n$ from their respective distributions;
while *at least one bad event occurs* **do**
  Find the set I of all occurring A_i;
  Independently resample all variables in $\bigcup_{i\in I} \mathsf{var}(A_i)$;
end
return *the final assignment*

The resampling table is a very useful concept of analysing Algorithm 1, introduced by [MT10], and similar ideas were also used in the analysis of cycle-popping [PW98] and sink-popping [CPP02]. Instead of drawing random samples, we associate an infinite stack to each random variable. Construct an infinite table such that each row represents random variables, and each entry is a sample of the random variable. The execution of Algorithm 1 can be thought of (but not really implemented this way) as first drawing the whole resampling table, and whenever we need to sample a variable, we simply use the next value of the table. Once the resampling table is fixed, Algorithm 1 becomes completely deterministic.

In general Algorithm 1 does not necessarily produce the desired distribution $\pi(\cdot)$. However, it turns out that it does for extremal instances (in the sense of Shearer [She85]).

Condition 1. *We say a collection of bad events $(A_i)_{i\in[m]}$ are* extremal*, if for any $i \neq j$, either $\mathsf{var}(A_i) \cap \mathsf{var}(A_j) = \emptyset$ or $\Pr_\mu(A_i \wedge A_j) = 0$.*

In other words, if the collection is extremal, then any two bad events are either disjoint or depend on distinct sets of variables (and therefore independent). It was shown [GJL17, Theorem 8] that if Condition 1 holds, then Algorithm 1 indeed draws from $\pi(\cdot)$ (conditioned on halting). In particular, Condition 1 guarantees that the set of occurring bad events have disjoint sets of variables.

Condition 1 may seem rather restricted, but it turns out that many natural problems have an extremal formulation. Examples include the cycle-popping algorithm for uniform spanning trees [Wil96, PW98], the sink-popping algorithm for sink-free orientations [CPP02], and the cluster-popping algorithm for root-connected subgraphs [GP14, GJ18c]. In particular, the cluster-popping algorithm yields the first polynomial-time approximation algorithm for the all-terminal network reliability problem. More recently, another application is found to approximately count the number of bases in a bicircular matroid [GJ18a], which is known to be #**P**-hard to count exactly [GN06]. We refer interested readers to [GJL17, GJ18b] for further applications of the partial rejection sampling framework beyond extremal instances.

A particularly nice feature of partial rejection sampling algorithms for extremal instances is that we have an exact formula [GJL17, Theorem 13] for the expected number of events resampled. This formula makes the analysis of these algorithms much more tractable. However, it counts the number of resampled events. In the most interesting applications, bad events typically involve more than constantly many variables. Using the worst case bound of the number of variables involved in bad events yields loose bounds.

We give a sharper formula for the expected number of *variables* resampled next. Let T_i be the number of resamplings of event A_i. Let q_i be the probability such that exactly A_i occurs,

and $q_\emptyset$ be the probability such that none of $(A_i)_{i\in[m]}$ occurs. Suppose $q_\emptyset > 0$ as otherwise the support of $\pi(\cdot)$ is empty. For extremal instances, [GJL17, Lemma 12] and the first part of the proof of [GJL17, Theorem 13] yield

$$\mathbb{E}\, T_i = \frac{q_i}{q_\emptyset}. \tag{1}$$

The proof of (1) is via manipulating the moment-generating function of T_i. Let T be the number of resampled variables.[1] By linearity of expectation and (1),

$$\mathbb{E}\, T = \sum_{i=1}^{m} \frac{q_i \cdot |\mathsf{var}(A_i)|}{q_\emptyset}. \tag{2}$$

We note that an upper bound similar to the right hand side of (2) was first shown by Kolipaka and Szegedy [KS11], in a much more general setting but counting the number of resampled events. We will use (2) to derive both upper and lower bounds.

In order to upper bound the ratio in (2), we will use a combinatorial encoding idea, namely to design an injective mapping. For an assignment σ, let $\mathrm{wt}(\sigma)$ be its weight so that $\mathrm{wt}(\sigma) \propto \mathrm{Pr}_\mu(\sigma)$. Let Ω_{A_i} be the set of assignments so that exactly A_i occurs, and $\Omega_1 := \bigcup_{i=1}^m \Omega_{A_i}$. Note that $(\Omega_{A_i})_{i\in m}$ are mutually exclusive and are in fact a partition of Ω_1. Also, let

$$\Omega_1^{\mathsf{var}} := \{(\sigma, X) \mid \exists i,\ \sigma \in \Omega_{A_i} \text{ and } X \in \mathsf{var}(A_i)\}.$$

Moreover, let Ω_0 be the set of "perfect" assignments such that none of $(A_i)_{i\in[m]}$ occurs.

Definition 2. *For a constant $r > 0$ and an auxiliary set Aux, a mapping $\tau : \Omega_1^{\mathsf{var}} \to \Omega_0 \times \mathsf{Aux}$ is* r-preserving *if for any i, any $\sigma \in \Omega_{A_i}$ and $X \in \mathsf{var}(A_i)$,*

$$\mathrm{wt}(\sigma) \le r \cdot \mathrm{wt}(\tau'(\sigma)),$$

where τ' is the restriction of τ on the first coordinate.

A straightforward consequence of (2) is the following theorem, which will be our main technical tool.

Theorem 3. *If there exists a r-preserving injective mapping $\tau : \Omega_1^{\mathsf{var}} \to \Omega_0 \times \mathsf{Aux}$ for some auxiliary set Aux, then the expected number of resampled variables of Algorithm 1 is at most $r\,|\mathsf{Aux}|$.*

Proof. Note that $q_i \propto \sum_{\sigma\in\Omega_{A_i}} \mathrm{wt}(\sigma)$ and $q_\emptyset \propto \sum_{\sigma\in\Omega_0} \mathrm{wt}(\sigma)$. As τ is r-preserving and injective,

$$\sum_{i=1}^{m} \frac{q_i \cdot |\mathsf{var}(A_i)|}{q_\emptyset} \le r\,|\mathsf{Aux}|\,.$$

The theorem then follows from (2). □

3. Cluster-popping

Let $G = (V, A)$ be a directed graph with root r. The graph G is called *root-connected* if there is a directed path in G from every non-root vertex to r. Let $0 < p_a < 1$ be the failure probability for arc a, and define the weight of a subgraph $S \subseteq A$ to be $\mathrm{wt}(S) := \prod_{a\in S}(1-p_a)\prod_{a\notin S} p_a$. Thus the target distribution $\pi(\cdot)$ is over all root-connected subgraphs, and $\pi(S) \propto \mathrm{wt}(S)$.

The extremal formulation by Gorodezky and Pak [GP14] is the following. Each arc e is associated with a (distinct) random variable which is present with probability $1-p_a$. A cluster is a subset of vertices not containing r so that no arc is going out. We want all vertices to be able to reach r. Thus clusters are undesirable. However, Condition 1 is not satisfied if we simply let the bad events be clusters. Instead, we choose minimal clusters to be bad events.

More precisely, for each $C \subseteq V$, we associate it with a bad event B_C to indicate that C is a cluster and no proper subset $C' \subsetneq C$ is one. Each B_C depends on arcs going out of C for being a cluster, as well as arcs inside C for being minimal. In other words, $\mathsf{var}(B_C) = \{u \to v \mid u \in C\}$.

[1] This notation is different from that in [GJL17].

There are clearly exponentially many bad events. A description of the algorithm is given in Algorithm 2.

Algorithm 2: Cluster-popping

Let S be a subset of arcs by choosing each arc a with probability $1 - p_a$ independently;
while *there is a cluster in* (V, S) **do**
  Let $C_1, \ldots, C_k$ be all minimal clusters in (V, S);
  Re-randomize all arcs in $\bigcup_{i=1}^{k} \mathsf{var}(B_{C_i})$ to get a new S;
end
return S

Condition 1 is met, due to the following observation [GJ18c, Claim 3].

Claim 4. *Any minimal cluster is strongly connected.*

Since Condition 1 holds, Algorithm 2 draws from the desired distribution over root-connected subgraphs in a directed graph. It was further shown in [GJ18c] that the cluster-popping algorithm can be used to sample spanning connected subgraphs in an undirected graph, and to approximate all-terminal network reliability in expected polynomial time. We will first give an improved running time bound for the basic cluster-popping algorithm.

3.1. **Expected running time.** A graph $G = (V, A)$ is *bi-directed* if each arc $a \in A$ has an anti-parallel twin in A as well. For an arc $a = u \to v$, let $\overline{a} := v \to u$ be its reversal. Let $n = |V|$ and $m = |A|$. It was shown that cluster-popping can take exponential time in general [GP14] while in bi-directed graphs, the expected number of resampled variables is at most $\frac{p_{\max}}{1-p_{\max}} m^2 n$ [GJ18c], where $p_{\max} = \max_{a \in A} p_a$. We will now design a $\frac{p_{\max}}{1-p_{\max}}$-preserving injective mapping from Ω_1^{var} to $\Omega_0 \times V \times A$ for a connected and bi-directed graph G, where Ω_0 is the set of all root-connected subgraphs, and Ω_1 is the set of subgraphs with a unique minimal cluster. Applying Theorem 3 with $\mathsf{Aux} = V \times A$, the expected number of resampled variables is $\frac{p_{\max}}{1-p_{\max}} mn$. Thus a factor m is shaved.

Our $\frac{p_{\max}}{1-p_{\max}}$-preserving injective mapping $\varphi : \Omega_1^{\mathsf{var}} \to \Omega_0 \times V \times A$ is modified from the one in [GJ18c]. The main difference is that the domain is Ω_1^{var} instead of Ω_1, and thus we need to be able to recover the random variable in the encoding. The encoding is also more efficient. For example, we only record u instead of $u \to u'$ in [GJ18c] as explained below.

For completeness and clarity we include full details. We assume that the bi-directed graph G is connected so that $\Omega_0 \neq \emptyset$, and the root r is arbitrarily chosen. (Apparently in a bi-directed graph, weak connectivity is equivalent to strong connectivity.)

For each subgraph $S \in \Omega_1$, the rough idea is to "repair" S so that no minimal cluster is present. We fix in advance an arbitrary ordering of vertices and arcs. Let C be the unique minimal cluster in S and $v \to v'$ be an arc so that $v \in C$, namely $v \to v' \in \mathsf{var}(B_C)$. Let R denote the set of all vertices which can reach the root r in the subgraph S. Since $S \in \Omega_1$, $R \neq V$. Let $U = V \setminus R$. Since G is root-connected, there is at least one arc in A from U to R. Let $u \to u'$ be the first such arc, where $u \in U$ and $u' \in R$. Let

$$\varphi(S, v \to v') := (S_{\mathrm{fix}}, u, v \to v'), \tag{3}$$

where $S_{\mathrm{fix}} \in \Omega_0$ is defined in the same way as in [GJ18c] and will be presented shortly. In [GJ18c], the mapping is from S to $(S_{\mathrm{fix}}, v, u \to u')$.

Consider the subgraph $H = (U, S[U])$, where

$$S[U] := \{x \to y \mid x \in U,\ y \in U,\ x \to y \in S\}.$$

We consider the directed acyclic graph (DAG) of strongly connected components of H, and call it $\widehat{H}$. (We use the decoration $\widehat{\ }$ to denote arcs, vertices, etc. in $\widehat{H}$.) To be more precise, we replace each strongly connected component by a single vertex. For a vertex $w \in U$, let $[w]$ denote the strongly connected component containing w. For example, $[v]$ is the same as

the minimal cluster C by Claim 4. We may also view $[w]$ as a vertex in $\widehat{H}$ and we do not distinguish the two views. The arcs in $\widehat{H}$ are naturally induced by $S[U]$. Namely, for $[x] \neq [y]$, an arc $[x] \to [y]$ is present in $\widehat{H}$ if there exists $x' \in [x]$, $y' \in [y]$ such that $x' \to y' \in S$.

We claim that $\widehat{H}$ is root-connected with root $[v]$. This is because $[v]$ must be the unique sink in $\widehat{H}$ and $\widehat{H}$ is acyclic. If there is another sink $[w]$ where $v \notin [w]$, then $[w]$ is a minimal cluster in H. This contradicts to $S \in \Omega_1$.

Since $\widehat{H}$ is root-connected, there is at least one path from $[u]$ to $[v]$. Let $\widehat{W}$ denote the set of vertices of $\widehat{H}$ that can be reached from $[u]$ in $\widehat{H}$ (including $[u]$), and $W := \{x \mid [x] \in \widehat{W}\}$. Then W is a cluster and $[u]$ is the unique source in $\widehat{H}[\widehat{W}]$. As $\widehat{H}$ is root-connected, $[v] \in \widehat{W}$. To define S_{fix}, we reverse all arcs in $S[W]$ and add the arc $u \to u'$ to eliminate the cut. Formally, let

$$S_{\text{fix}} := (S \setminus S[W]) \cup \{u \to u'\} \cup \{y \to x \mid x \to y \in S[W]\}.$$

Let $\widehat{H}_{\text{fix}}$ be the graph obtained from $\widehat{H}$ by reversing all arcs induced by $S[W]$. Observe that $[u]$ becomes the unique sink in $\widehat{H}_{\text{fix}}[\widehat{W}]$ (and $[v]$ becomes the unique source).

We verify that $S_{\text{fix}} \in \Omega_0$. For any $x \in R$, x can still reach r in (V, S_{fix}) since the path from x to r in (V, S) is not changed. Since $u \to u' \in S_{\text{fix}}$, u can reach $u' \in R$ and hence r. For any $y \in W$, y can reach u as $[u]$ is the unique sink in $\widehat{H}_{\text{fix}}[\widehat{W}]$. For any $z \in U \setminus W$, z can reach $v \in W$ since the path from z to v in (V, S) is not changed.

Lemma 5. *The mapping $\varphi : \Omega_1^{\mathsf{var}} \to \Omega_0 \times V \times A$ defined in (3) is $\frac{p_{\max}}{1-p_{\max}}$-preserving and injective.*

Proof. It is easy to verify $\frac{p_{\max}}{1-p_{\max}}$-preservingness, since flipping arcs leaves its weight unchanged. The only move changing the weight is to add the arc $u \to u'$ in S_{fix}, which results in at most $\frac{p_{\max}}{1-p_{\max}}$ change.

Next we verify that φ is injective. To do so, we show that we can recover S and $v \to v'$ given S_{fix}, u, and $v \to v'$. Clearly, it suffices to recover just S.

First observe that in S_{fix}, u' is the first vertex on any path from u to r. Thus we can recover $u \to u'$. Remove it from S_{fix}. The set of vertices which can reach r in $(V, S_{\text{fix}} \setminus \{u \to u'\})$ is exactly R in (V, S). Namely we can recover U and R. As a consequence, we can recover all arcs in S that are incident with R, as these arcs are unchanged.

What is left to do is to recover arcs in $S[U]$. To do so, we need to find out which arcs have been flipped. We claim that $\widehat{H}_{\text{fix}}$ is acyclic. Suppose there is a cycle in $\widehat{H}_{\text{fix}}$. Since $\widehat{H}$ is acyclic, the cycle must involve flipped arcs and thus vertices in $\widehat{W}$. Let $[x] \in \widehat{W}$ be the lowest one under the topological ordering of $\widehat{H}[\widehat{W}]$. Since $\widehat{W}$ is a cluster, the outgoing arc $[x] \to [y]$ along the cycle in $\widehat{H}_{\text{fix}}$ must have been flipped, implying that $[y] \in \widehat{W}$ and $[y] \to [x]$ is in $\widehat{H}[\widehat{W}]$. This contradicts to the minimality of $[x]$.

Since $\widehat{H}_{\text{fix}}$ is acyclic, the strongly connected components of $H_{\text{fix}} := (U, S_{\text{fix}}[U])$ are identical to those of $H = (U, S[U])$. (Note that flipping all arcs in $S[W]$ leaves strongly connected components inside W unchanged.) Hence contracting all strongly connected components of H_{fix} results in exactly $\widehat{H}_{\text{fix}}$. All we need to recover now is the set $\widehat{W}$. Let $\widehat{W}'$ be the set of vertices reachable from $[v]$ in $\widehat{H}_{\text{fix}}$. It is easy to see that $\widehat{W} \subseteq \widehat{W}'$ since we were flipping arcs. We claim that actually $\widehat{W} = \widehat{W}'$. For any $[x] \in \widehat{W}'$, there is a path from $[v]$ to $[x]$ in $\widehat{H}_{\text{fix}}$. Suppose $[x] \notin \widehat{W}$. Since $[v] \in \widehat{W}$, we may assume that $[y]$ is the first vertex along the path such that $[y] \to [z]$ where $[z] \notin \widehat{W}$. Thus $[y] \to [z]$ has not been flipped and is present in $\widehat{H}$. However, this contradicts the fact that $\widehat{W}$ is a cluster in $\widehat{H}$.

To summarize, given S_{fix}, u, and $v \to v'$, we may uniquely recover S and thus $v \to v'$. Hence the mapping φ is injective. □

Combining Lemma 5 and Theorem 3 (with $\mathsf{Aux} = V \times A$) implies an upper bound of the number of random variables drawn in expectation for Algorithm 2.

Theorem 6. *To sample edge-weighted root-connected subgraphs, the expected number of random variables drawn in Algorithm 2 on a connected bi-directed graph $G = (V, A)$ is at most $m + \frac{p_{\max} mn}{1-p_{\max}}$, where $n = |V|$ and $m = |A|$.*

Another consequence of Lemma 5 is that we can bound the expected number of resamplings of each individual variable. Denote by T_a the number of resamplings of a. Then, by (1),

$$\mathbb{E}\, T_a = \sum_{C\colon a \in \mathsf{var}(B_C)} \frac{q_C}{q_\emptyset},$$

where q_C is the probability that under the product distribution, there is a unique minimal cluster C, and $q_\emptyset$ is the probability that under the product distribution, there is no cluster. Through the $\frac{p_{\max}}{1-p_{\max}}$-preserving injective mapping φ, namely (3), if we fix a, it is easy to see that

$$\sum_{C \in \mathcal{C}_a} \frac{q_C}{q_\emptyset} \le \frac{p_{\max} n}{1 - p_{\max}}.$$

As the above holds for any a, we have that the expected number of resampling of any variable is at most $\frac{p_{\max} n}{1-p_{\max}}$. This is an upper bound for the expected depth of the resampling table.

3.2. **An efficient implementation.** Theorem 6 bounds the number of random variables drawn. However, regarding the running time of Algorithm 2, in addition to drawing random variables, a naive implementation of Algorithm 2 may need to find clusters in every iteration of the while loop, which may take as much as $O(m)$ time by, for example, Tarjan's algorithm [Tar72]. In Algorithm 3 we give an efficient implementation, which can be viewed as a dynamic version of Tarjan's algorithm.

Algorithm 3 is sequential, and its correctness relies on the fact that the order of resampling events for extremal instances does not affect the final output. See [GJ18a, Section 4] for a similar sequential (but efficient) implementation of "bicycle-popping".

Two key modifications in Algorithm 3 comparing to Tarjan's algorithm are:

(1) in the DFS, once the root r is reached, all vertices along the path are "set" and will not be resampled any more;
(2) the first output of Tarjan's algorithm is always a strongly connected component with no outgoing arcs. Such a component (if it does not contain the root r) is a minimal cluster and will immediately be resampled in Algorithm 3.

Let $G = (V, A)$ be the input graph. For $v \in V$, let $\mathsf{arc}(v) = \{v \to w \mid v \to w \in A\}$ be the set of arcs going out from v. Recall that for $C \subseteq V$, $\mathsf{var}(B_C) = \bigcup_{v \in C} \mathsf{arc}(v)$.

We first observe that in Algorithm 3, $index - 1$ is always the number of variables that have already been indexed. Furthermore, inside the function "Dynamic-DFS" of Algorithm 3, for any $v \in V$, if $v.root$ is defined, then v can reach the vertex indexed by $v.root$. This can be shown by a straightforward induction on the recursion depth, and observing that resampling in any Dynamic-DFS(v) will not affect arcs whose heads are indexed before v.

Lemma 7. *At the beginning of each iteration of the while loop in Algorithm 3, all vertices whose indices are defined can reach the root r, and they will not be resampled any more.*

Proof. We do an induction on the number of while loops executed. The base case is trivial as only r is indexed before the first iteration. Suppose we are to execute Dynamic-DFS(v) for some v. By the induction hypothesis, for any w such that $w.index < v.index$, w can reach the root r. As $u.root$ is non-increasing for any $u \in V$, the only possibility to exit any (recursive) call of Dynamic-DFS(u) is that $u.root < u.index$. Suppose after finishing Dynamic-DFS(v), u is the lowest vertex that is newly indexed but cannot reach r, and $u.root = w.index$ for some w. Then u can reach w and $w.index < u.index$. However, the latter implies that w can reach r by the choice of u, which is a contradiction. □

In particular, Lemma 7 implies that once the algorithm halts, all vertices can reach the root r, and the output is a root-connected subgraph.

Algorithm 3: Cluster-popping with Tarjan's algorithm

Input : A directed graph $G = (V, A)$ with a special root vertex r;
Output: $R \subseteq A$ drawn according to $\pi(\cdot)$;
Let $R \subseteq A$ be obtained by drawing each arc $a \in A$ with probability $1 - p_a$ independently;
$r.index \leftarrow 1$, $r.root \leftarrow 1$, $index \leftarrow 2$;
Let S be a stack consisting of only r;
while $\exists v \in V$, $v.index =$ UNDEFINED **do**
  Dynamic-DFS(v);
end
return R;
Function Dynamic-DFS(v)**:**
  $v.index \leftarrow index$;
  $v.root \leftarrow v.index$;
  $index \leftarrow index + 1$;
  $S.push(v)$;
  for each $v \to w \in R$ **do**
    if $w.index =$ UNDEFINED **then**
      Dynamic-DFS(w);
      $v.root \leftarrow \min\{v.root, w.root\}$;
    else
      $v.root \leftarrow \min\{v.root, w.index\}$;
    end
  end
  if $v.root = v.index$ **then** `// A minimal cluster is found`
    repeat
      $w \leftarrow S.pop()$;
      $w.index \leftarrow$ UNDEFINED, $w.root \leftarrow$ UNDEFINED, $index \leftarrow index - 1$;
      $R \leftarrow R \setminus \mathsf{arc}(w)$, draw each arc $a \in \mathsf{arc}(w)$ with probability $1 - p_a$
       independently, and add them to R; `// Resampling step`
    until $w = v$;
    Dynamic-DFS(v);
  end
end

Lemma 8. *Inside the function "Dynamic-DFS(v)" of Algorithm 3, if $v.root = v.index$ happens, then a minimal cluster is resampled.*

Proof. Suppose $v.index = v.root$. We want to show that all vertices indexed after v form a minimal cluster. Let $U = \{u \mid u.index \geq v.index\}$.

Clearly v can reach all vertices in U. For any $u \in U$, if $u.root < v.index$, then $v.root \leq u.root < v.index$, contradicting to our assumption. Thus $u.root \geq v.index$ for all $u \in U$. Let $u_0 \in U$ be the lowest indexed vertex that cannot reach v. Since the recursive call of u_0 has exited, $v.index \leq u_0.root < u_0.index$. Thus $u_0.root$ is a vertex in U and can reach v due to the choice of u_0. It implies that u_0 can reach $u_0.root$ and thus v, a contradiction. Hence, all of U can reach v, and thus U is strongly connected.

If there is any arc in the current R going out from U, say $u \to w$ for some $u \in U$, then it must be that $w.index < v.index$. However, this implies that $u.root \leq w.index < v.index$, contradicting to $u.root \geq v.index$ for all $u \in U$. This implies that there is no arc going out from U. Thus, U is a cluster and is strongly connected, and it must be a minimal cluster. □

Now we are ready to show the correctness and the efficiency of Algorithm 3.

Theorem 9. *The output distribution of Algorithm 3 is the same as that of Algorithm 2, and the expected running time is* $O\left(m + \frac{p_{\max} mn}{1-p_{\max}}\right)$.

Proof. By Lemma 7 and Lemma 8, in Algorithm 3, only minimal clusters are resampled and the halting rule is when no minimal cluster is present. In other words, the resampling rules are the same for Algorithm 3 and Algorithm 2, except the difference in orderings. Given a resampling table, our claim is that the resampled variables are exactly the same for Algorithm 3 and Algorithm 2, which leads to an identical final state. The claim can be verified straightforwardly, or we can use a result of Eriksson [Eri96]. If any two minimal clusters are present, then they must be disjoint (Condition 1), and thus different orders of resampling them lead to the same state for a fixed resampling table. This is the *polygon property* in [Eri96], which by [Eri96, Theorem 2.1] implies the *strong convergence property*. The latter is exactly our claim.

Regarding the running time of Algorithm 3, we observe that its running time is linear in the final output and the number of resampled variables. The expected number of resampled variables of Algorithm 3 is the same as that of Algorithm 2 due to the claim above. Thus, Theorem 6 implies the claimed run-time bound. □

We can further combine Algorithm 3 with the coupling procedure [GJ18c, Section 5] to yield a sampler for edge-weighted spanning connected subgraphs in an *undirected* graph, which is key to the FPRAS of all-terminal network reliability. The coupling performs one scan over all edges. Thus we have the following corollary of Theorem 9.

Corollary 10. *There is an algorithm to sample edge-weighted spanning connected subgraphs in an undirected graph* $G = (V, E)$ *with expected running time* $O\left(m + \frac{p_{\max} mn}{1-p_{\max}}\right)$, *where* $n = |V|$ *and* $m = |E|$.

3.3. **A tight example.** In this section, we present an example that the expected number of random variables drawn in Algorithm 2 (and thus Algorithm 3) is $2m + (1 - o(1))\frac{pmn}{1-p}$, where $p_a = p$ for all $a \in A$. Thus, the bound in Theorem 6 is tight.

Our example is the bi-directed version of the "lollipop" graph, where a simple path P of length n_1 is attached to a clique K of size n_2. A picture is drawn in Figure 1.

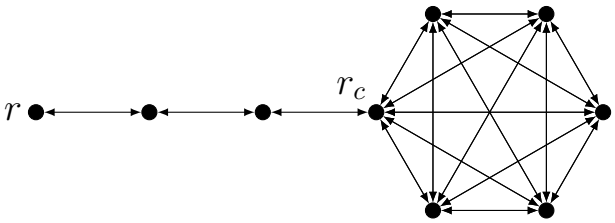

FIGURE 1. A bi-directed lollipop graph with $n_1 = 3$ and $n_2 = 6$.

The main tool is still the formula (2). We have constructed an injective mapping $\varphi : \Omega_1^{\mathsf{var}} \to \Omega_0 \times V \times A$. Thus, to derive a lower bound, we just need to lower bound the weighted ratio between $\varphi(\Omega_1^{\mathsf{var}})$ and $\Omega_0 \times V \times A$. The main observation is that for most $S' \in \Omega_0$, the tuple $(S', u, v \to v') \in \varphi(\Omega_1^{\mathsf{var}})$ as long as $u \in P$ is not the right endpoint of P, and $v \to v'$ is an arc in K. We may choose n_1 and n_2 so that $|P| = n_1 = (1 - o(1))n$ and the number of arcs in K is $n_2(n_2 - 1) = (1 - o(1))m$. The bound in Algorithm 2 is tight with this choice.

For concreteness, let $n_1 = \lceil n_2^{1+\varepsilon} \rceil$ for some $0 < \varepsilon < 1$ and consider $n_2 \to \infty$. Then the number of vertices is $n = n_1 + n_2 = (1 + n_2^{-\varepsilon})n_1$ and the number of arcs is $m = 2n_1 + n_2(n_2 - 1) = (1 + o(1))n_2^2$. Let the root r be the leftmost vertex of P, and r_c be the vertex where P and K intersect. A subgraph $S' \in \Omega_0$ must contain a directed path from r_c to r, as well as a root-connected subgraph in K with root r_c. For any constant p, since K is a clique, with high probability, a random subgraph (by drawing each edge independently with probability $1 - p$) in

K is strongly connected.[2] Let Ω_0' be the set of subgraphs which contain a directed path from r to r_c and strongly connected inside K. Clearly the total weight of Ω_0' is $1-o(1)$ of that of Ω_0.

For each $S' \in \Omega_0'$, $u \in P \setminus \{r_c\}$, and $v \to v'$ an arc in K, it is straightforward to verify that the "repairing" procedure in Lemma 5 goes through. We first remove the arc $u' \to u$, where u' is the next vertex on P, and call the vertex set that cannot reach the root U. Since $S' \in \Omega_0'$, if we contract strongly connected components in $S'[U]$, it must be a directed path. Flip all arcs in $S'[U]$ to get S. The subgraph $S[U]$ must have the same collection of strongly connected components as $S'[U]$, and the contracted graph is a directed path in the reverse direction. Clearly there is a unique sink, namely a minimal cluster in $S[U]$, which is the clique K with possibly a few vertices along the path P. Hence $v \to v'$ is in the unique minimal cluster of S.

To summarise, $S \in \Omega_1$, $\mathrm{wt}(S) = \frac{p}{1-p} \cdot \mathrm{wt}(S')$, and $\varphi(S, v \to v') = (S', u, v \to v')$. Thus, by (2), the expected number of random variables drawn of Algorithm 2 is at least

$$2m + (1-o(1)) \cdot \frac{p}{1-p} \cdot |P \setminus \{r_c\}| \cdot n_2(n_2-1) = 2m + (1-o(1))\frac{pmn}{1-p},$$

where the term $2m$ accounts for the initialisation.

Remark. *If we set $\varepsilon = 0$, then the running time becomes $\Omega(n^3)$. However, the optimal constant (measured in n^3) is not clear.*

An interesting observation is that the running time of Algorithm 1 depends on the choice of r in the example above, although in the reliability approximation algorithm, r can be chosen arbitrarily. (See [GJ18c, Section 5].) However, choosing the best r does not help reducing the order of the running time. In a "barbell" graph, where two cliques are joined by a path, no matter where we choose r, there is a rooted induced subgraph of the same structure as the example above, leading to the same $\Omega(n^3)$ running time when $\varepsilon = 0$.

3.4. **Faster reliability approximation.** The main application of Algorithm 2 is to approximate the network reliability of a undirected graph $G = (V, E)$, which is the probability that, assuming each edge e fails with probability p_e independently, the remaining graph is still connected. Let $\mathbf{p} = (p_e)_{e \in E}$ be the vector of the failure probabilities. Then the reliability is the following quantity:

$$Z_{\mathrm{rel}}(\mathbf{p}) = \sum_{\substack{S \subseteq E \\ (V,S) \text{ is connected}}} \prod_{e \in S} (1-p_e) \prod_{e \notin S} p_e. \tag{4}$$

The approximate counting algorithm in [GP14, GJ18c] takes $O(n^2/\varepsilon^2)$ samples of spanning connected subgraphs to produce a $1 \pm \varepsilon$ approximation of Z_{rel}. However, we can rewrite (4) as a partition function of the Gibbs distribution. Thus we can take advantage of faster approximation algorithms, such as the one by Kolmogorov [Kol18].

Let Ω be a finite set, and the Gibbs distribution $\pi(\cdot)$ over Ω is one taking the following form:

$$\pi_\beta(X) = \frac{1}{Z(\beta)} \exp(-\beta H(X)),$$

where β is the *temperature*, $H(X) \geq 0$ is the *Hamiltonian*, and $Z(\beta) = \sum_{X \in \Omega} \exp(-\beta H(X))$ is the normalising factor (namely the partition function) of the Gibbs distribution. We would like to turn the sampling algorithm into an approximation algorithm to $Z(\beta)$. Typically, this involves calling the sampling oracle in a range of temperatures, which we denote $[\beta_{\min}, \beta_{\max}]$. (This process is usually called simulated annealing.) Let $Q := \frac{Z(\beta_{\min})}{Z(\beta_{\max})}$, $q = \log Q$, and $N = \max_{X \in \Omega} H(X)$. The following result is due to Kolmogorov [Kol18, Theorem 8 and Theorem 9].

Proposition 11. *Suppose we have a sampling oracle from the distribution π_β for any $\beta \in [\beta_{\min}, \beta_{\max}]$. There is an algorithm to approximate Q within $1 \pm \varepsilon$ multiplicative errors using $O(q \log N/\varepsilon^2)$ oracle calls in average.*

[2] This fact is easy to prove. Recall that the analogue connectivity threshold for Erdős-Rényi random graph is $p = \frac{\log n}{n}$.

Moreover, the sampling oracle $\widetilde{\mu}_\beta$ can be approximate as long as $\|\mu_\beta - \widetilde{\mu}_\beta\|_{TV} = O(1/(q \log N))$ where $\|\cdot\|_{TV}$ is the variation distance.

A straightforward application of Proposition 11 to our problem requires $O(m \log n)$ samples. This is because annealing will be done on all edges. Instead, we will choose a spanning tree, and perform annealing only on its edges, whose cardinality is $n-1$. This approach uses only $O(n \log n)$ samples, but it requires the following slight generalisation of Proposition 11.

Let $\rho_\beta(\cdot)$ be the following distribution over a finite set Ω:

$$\rho_\beta(X) = \frac{1}{Z(\beta)} \exp(-\beta H(X)) \cdot F(X), \tag{5}$$

where $F : \Omega \to \mathbb{R}^+$ is a non-negative function and, with a little abuse of notation, $Z(\beta) = \sum_{X \in \Omega} \exp(-\beta H(X)) \cdot F(X)$ is the normalising factor. Still, let $Q := \frac{Z(\beta_{\min})}{Z(\beta_{\max})}$, $q = \log Q$, and $N = \max_{X \in \Omega} H(X)$.

Lemma 12. *Suppose we have a sampling oracle from the distribution ρ_β defined in (5) for any $\beta \in [\beta_{\min}, \beta_{\max}]$. There is an algorithm to approximate Q within $1 \pm \varepsilon$ multiplicative errors using $O(q \log N / \varepsilon^{-2})$ oracle calls.*

Proof. We claim that we can straightforwardly apply the algorithm in Proposition 11 to get an approximation to Q for ρ_β.

To see this, let $\ell \geq 0$ be an integer. Let Ω' be the multi-set of Ω where each X is duplicated $\lfloor \ell F(X) \rfloor$ times. To avoid multi-set, we can simply give each duplicated X an index to make Ω' an ordinary set. Consider the following Gibbs distribution over Ω':

$$\pi_\beta(X) = \frac{1}{Z'(\beta)} \exp(-\beta H(X)),$$

where $Z'(\beta) = \sum_{X \in \Omega'} \exp(-\beta H(X))$. Let

$$\delta_\ell := \min_{X \in \Omega} \left(1 - \frac{\lfloor \ell F(X) \rfloor}{\ell F(X)}\right).$$

We have that

$$1 - \delta_\ell \leq \frac{Z'(\beta)}{\ell Z(\beta)} \leq 1.$$

Clearly, as $\ell \to \infty$, $\delta_\ell \to 0$. We choose ℓ large enough so that $\delta_\ell = O(1/q \log N)$ is within the threshold in Proposition 11 for approximate sampling oracle. Thus, the output of directly applying the algorithm in Proposition 11 with sampling oracle ρ_β also yields an ε-approximation to $Q' := \frac{Z'(\beta_{\min})}{Z'(\beta_{\max})}$. (Note that we do not really run the algorithm on Ω'.) Since $\frac{Q'}{Q} \to 1$ as $\ell \to \infty$, the output of directly applying Proposition 11 is in fact an ε-approximation to Q. □

Suppose we want to evaluate $Z_{\text{rel}}(\mathbf{p})$ for a connected undirected graph $G = (V, E)$. Since G is connected, $m \geq n - 1$, where $m = |E|$ and $n = |V|$. Fix an arbitrary spanning tree T of G in advance. Let

$$F(S) := \prod_{e \in T \setminus S} \frac{p_e}{1 - p_e} \prod_{e \in S \setminus T} (1 - p_e) \prod_{e \in E \setminus (T \cup S)} p_e.$$

Let Ω_0 be the set of all spanning connected subgraphs of G, and $\rho_\beta(S)$ be the following distribution over Ω_0:

$$\rho_\beta(S) = \frac{1}{Z(\beta)} \exp(-\beta H(S)) \cdot F(S),$$

where $\beta \geq 0$ is the temperature, $H(S) := |T \setminus S| = n - 1 - |T \cap S|$ is the Hamiltonian, and the normalising factor $Z(\beta) = \sum_{S \in \Omega_0} \exp(-\beta H(S)) \cdot F(S)$. For any $\beta \geq 0$, let $0 < p'_e \leq p_e$ be the

probability such that $\frac{p_e \exp(-\beta)}{1-p_e} = \frac{p'_e}{1-p'_e}$. We have that for any $S \in \Omega_0$,

$$\begin{aligned}\rho_\beta(S) &= \frac{1}{Z(\beta)} \prod_{e\in T\setminus S} \frac{p_e \exp(-\beta)}{1-p_e} \prod_{e\in S\setminus T} (1-p_e) \prod_{e\in E\setminus(T\cup S)} p_e \\ &= \frac{1}{Z(\beta)} \prod_{e\in T\setminus S} \frac{p'_e}{1-p'_e} \prod_{e\in S\setminus T} (1-p_e) \prod_{e\in E\setminus(T\cup S)} p_e.\end{aligned}$$

To draw a sample from ρ_β, we use Corollary 10, for a vector $\mathbf{p}$ such that every $e \in T$ fails with probability p'_e, and every other $e \notin T$ fails with probability p_e. To see that this recovers the distribution ρ_β, notice that the weight $\mathrm{wt}(S)$ assigned to each subgraph $S \in \Omega_0$ is

$$\begin{aligned}\mathrm{wt}(S) &= \prod_{e\in T\cap S} (1-p'_e) \prod_{e\in T\setminus S} p'_e \prod_{e\in S\setminus T} (1-p_e) \prod_{e\in E\setminus(T\cup S)} p_e \\ &= \rho_\beta(S) \cdot Z(\beta) \prod_{e\in T} (1-p'_e) \propto \rho_\beta(S).\end{aligned}$$

Let $\beta_{\min} = 0$ and $\beta_{\max} = \infty$. Indeed, $\beta = \infty$ corresponds to $p'_e = 0$. Hence, $\rho_\infty(S) \neq 0$ if and only if $T \subseteq S$. This condition implies that $S \in \Omega_0$, and

$$\exp(-\infty \cdot H(S)) \cdot F(S) = \begin{cases} \prod_{e\in S\setminus T}(1-p_e) \prod_{e\in E\setminus(T\cup S)} p_e & \text{if } T \subseteq S; \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $Z(\infty) = 1$.

On the other hand, $Z(0) = \frac{Z_{\mathrm{rel}}(\mathbf{p})}{\prod_{e\in T}(1-p'_e)}$. Then

$$Q = \frac{Z(0)}{Z(\infty)} = \frac{Z_{\mathrm{rel}}(\mathbf{p})}{\prod_{e\in T}(1-p'_e)} \leq \prod_{e\in T}(1-p_e)^{-1},$$

and $q = \log Q \leq (n-1)\log\frac{1}{1-p_{\max}}$. Clearly, multiplicatively approximating Q is the same as approximating $Z_{\mathrm{rel}}(\mathbf{p})$. Moreover, $\max_{S\in\Omega_0} H(S) \leq n-1$. Thus, applying Lemma 12, we have an approximation algorithm for $Z(\beta)$ with $O(q \log N/\varepsilon^2) = O(n \log n \log\frac{1}{1-p_{\max}}/\varepsilon^2)$ oracle calls in expectation. Combining with Corollary 10, we have the following theorem.

Theorem 13. *There is fully polynomial-time randomised approximation scheme for $Z_{\mathrm{rel}}(\mathbf{p})$, which runs in time $O\left(\frac{mn^2 \log n}{\varepsilon^2(1-p_{\max})} \cdot \log\frac{1}{1-p_{\max}}\right)$ for an $(1\pm\varepsilon)$-approximation.*

4. Cycle-popping

Cycle-popping [Wil96, PW98] is a very simple algorithm to sample spanning trees in a connected undirected graph $G = (V, E)$. This algorithm actually generates rooted trees, so we will pick a special root vertex r. Of course, there is no real difference between rooted and unrooted spanning trees as the root can be chosen arbitrarily in any given spanning tree.

We consider a slightly more general setting, by giving each edge e a weight $w_e > 0$. For each vertex $v \in V$ other than r, we associate a random arc with v, which points to a neighbour of v so that u is chosen with probability proportional to $w_{(u,v)}$. We denote such an assignment by σ, and $\sigma(v)$ is the neighbour of v that is pointed at. Any such σ induces a directed graph with $n-1$ arcs, where $n = |V|$. The weight of σ is $\mathrm{wt}(\sigma) := \prod_{v\in V,\, v\neq r} w_{(v,\sigma(v))}$, and the target distribution $\pi(\cdot)$ is $\pi(\sigma) \propto \mathrm{wt}(\sigma)$ with support on the set of σ that induces a spanning tree.

Since we want trees in the end, cycles will be our bad events. For each cycle C, we associate with it a bad event B_C which indicates the presence of C. The set $\mathsf{var}(B_C)$ consists of random arcs associated with vertices along C. It is clear that if there is no cycle, the graph must be a spanning tree. The number of bad events can be exponentially large, since there can be exponentially many cycles in G. A description is given in Algorithm 4.

Condition 1 is easy to verify. Two cycles are dependent if they share at least one vertex. Suppose a cycle C is present, and $C' \neq C$ is another cycle that shares at least one vertex with C. If C' is also present, then we may start from any vertex $v \in C \cap C'$, and then follow the

Algorithm 4: Cycle-popping

For each vertex $v \neq r$, let $\sigma(v)$ be a neighbour of v with probability proportional to w_e independently;
while *there is at least one cycle under* σ **do**
  Find all cycles $C_1, \dots, C_k$ under σ;
  Re-randomize $\bigcup_{i=1}^{k} \mathsf{var}(B_{C_i})$;
end
return *the subgraph induced by the final* σ

arrows $v \to v'$. Since both C and C' are present, it must be that $v' \in C \cap C'$ as well. Continuing this argument we will go back to v eventually. Thus $C = C'$. Contradiction!

4.1. **Expected running time.** Next we turn to the running time of the cycle-popping algorithm, and define our injective mapping $\varphi : \Omega_1^{\mathsf{var}} \to \Omega_0 \times V \times A$, where A is the set of all ordered pairs from E, namely, $A = \{u \to v, v \to u \mid (u, v) \in E\}$. Hence $|A| = 2\,|E|$.

We fix in advance an arbitrary ordering of vertices and edges. Let $\sigma \in \Omega_C \subseteq \Omega_1$ be an assignment of random arcs so that there is a unique cycle C. Let u be a vertex on the cycle C and suppose $\sigma(u) = u'$. It is easy to see that there are two components in the subgraph induced by σ: a directed tree with root r, and the directed cycle with a number of directed subtrees rooted on the cycle. Since G is connected, there must be an edge joining the two components. Let this edge be (v_0, v_1), where v_0 is in the tree component and v_1 is in the unicyclic component. Starting from v_1 and following arcs under σ, we will eventually reach the cycle and arrive at u. Let vertices along this path be $v_2, v_3, \dots, v_\ell = u$. (It is possible that $\ell = 1$ or u' is along the path.)

To "fix" σ, we reassign the arrow out of v_i from v_{i+1} to v_{i-1} for all $1 \leq i \leq \ell$ (in case of $v_\ell = u$, it is rerouted from u' to $v_{\ell-1}$), and call the resulting assignment σ_{fix}. Namely, $\sigma_{\mathrm{fix}}(v_i) = v_{i-1}$ for all $i \in [\ell]$, and $\sigma_{\mathrm{fix}}(v) = \sigma(v)$ otherwise. It is easy to verify that $\sigma_{\mathrm{fix}} \in \Omega_0$.

Now we are ready to define the injective mapping φ. For $\sigma \in \Omega_C \subseteq \Omega_1$ and $u \in C$, let

$$\varphi(\sigma, u) := (\sigma_{\mathrm{fix}}, v_0, u \to u'), \tag{6}$$

where σ_{fix} and v_0 are defined above, and $u' = \sigma(u)$.

Let $r_{\max} := \max_{e,e' \in E} \frac{w_e}{w_{e'}}$.

Lemma 14. *The mapping $\varphi : \Omega_1^{\mathsf{var}} \to \Omega_0 \times V \times A$ defined in (6) is $r_{\max}$-preserving and injective.*

Proof. To verify that φ is injective, we just need to recover σ given σ_{fix}, v_0, and $u \to u'$. Since σ_{fix} is a spanning tree, there is a unique path between v_0 and u under σ_{fix}. This recovers $v_1, v_2, \dots, v_\ell$, and $\sigma(v) = \sigma_{\mathrm{fix}}(v)$ for all other vertices since their assignments are unchanged. Moreover, to recover $\sigma(v_i)$ for $i \in [\ell]$ is also easy — $\sigma(v_i) = v_{i+1}$ for $i \in [\ell - 1]$ and $\sigma(u) = u'$.

To verify that φ is $r_{\max}$-preserving, just notice that directions do not affect weights, and then the only difference between σ and σ_{fix} is the removal of (u, u') and the inclusion of (v_1, v_0). Thus the change in weights is at most $r_{\max}$. □

Combining Lemma 14 and Theorem 3 (with $\mathsf{Aux} = V \times A$) implies a run-time upper bound of the cycle-popping algorithm.

Theorem 15. *To sample edge-weighted spanning trees, the expected number of random variables drawn in Algorithm 4 on a connected undirected graph $G = (V, E)$ is at most $(n-1) + 2r_{\max}mn$, where $n = |V|$, $m = |E|$, and $r_{\max} := \max_{e,e' \in E} \frac{w_e}{w_{e'}}$.*

Remark. *It is very tempting to use $\mathsf{Aux} = V \times E$ instead of $V \times A$ in the proof above, which would yield an improvement by a factor 2. Unfortunately that choice does not work. The reason is that, if we use an unordered pair (u, u'), then given σ_{fix} and v_0, it is not always possible to distinguish u and u'. To see this, consider an assignment $\sigma \in \Omega_C$, and another assignment $\sigma' \in \Omega_C$ which is the same as σ except that the orientation on C is reversed. It is easy to check*

that the "fixed" versions of (σ, u) *and* (σ', u') *are the same if we do not record the direction of* (u, u')*. We will see next that the factor* 2 *in fact is unavoidable.*

4.2. **A tight example.** We also give a matching lower bound to complement Theorem 15. Recall Section 3.3. The general strategy is to construct an example where $\varphi(\Omega_1^{\mathsf{var}})$ constitutes most of $\Omega_0 \times V \times A$, and then invoke (2).

We use the undirected version of the same "lollipop" graph as in Section 3.3. (Recall Figure 1.) Namely, a clique K of size n_2 joined with a path P of length n_1, where $n_1 = \lceil n_2^{1+\varepsilon} \rceil$ for some $0 < \varepsilon < 1$. Consider $n_2 \to \infty$. The number of vertices is $n = n_1 + n_2 = (1 + n_2^{-\varepsilon})n_1$ and the number of edges is $m = n_1 + \frac{n_2(n_2-1)}{2} = (1/2 + o(1))n_2^2$. Let the root r be the leftmost vertex of P, and r_c be the vertex where P and K intersect. Moreover, we put weight w_c on all edges in K, and $w_p \le w_c$ on all edges in P.

For a tuple $(\sigma', v_0, u \to u')$ to belong to $\varphi(\Omega_1^{\mathsf{var}})$, the main constraint is that v_0 should be an ancestor of u in the spanning tree induced by σ'. In this example, any vertex $v \in P$ is an ancestor of any other vertex $u \in K$ in an arbitrary spanning tree. Thus, for any $\sigma' \in \Omega_0$, $v_0 \in P \setminus \{r_c\}$, and $u \to u'$ where $u, u' \in K$, we can apply the "repairing" procedure as given in Lemma 14 to get $\sigma \in \Omega_1$ so that $\varphi(\sigma, u) = (\sigma', v_0, u \to u')$. This is easy to verify, by finding the unique path between v_0 and u, and then reassign σ along the path. Since we remove one edge on the path and include one edge in the clique, $\mathrm{wt}(\sigma) = \frac{w_c}{w_p} \cdot \mathrm{wt}(\sigma') = r_{\max}\mathrm{wt}(\sigma')$. Thus, by (2), the expected number of random variables drawn of Algorithm 4 is at least

$$n - 1 + r_{\max} \cdot |P \setminus \{r_c\}| \cdot n_2(n_2 - 1) = n - 1 + (2 - o(1))r_{\max}mn,$$

where the term $n - 1$ accounts for the initialisation.

Similarly to Section 3.3, the choice of r affects the running time of Algorithm 4. Indeed, in Wilson's algorithm [Wil96, PW98], the root is chosen randomly according to the stationary distribution of the random walk, and with that choice the running time is $O(n^2)$ in a lollipop graph (with $\varepsilon = 0$, namely $n_1 \asymp n_2$). However, also similarly to Section 3.3, the running time in a "barbell" graph is still $\Omega(n^3)$. Once again, the optimal constant measured in n^3 is still not clear.

5. Sink-popping

In this section, we describe and analyse the sink-popping algorithm by Cohn, Pemantle, and Propp [CPP02]. The goal is to sample a sink-free orientation in an undirected graph. This problem was introduced by Bubley and Dyer [BD97a] as an early showcase of the power of path coupling for Markov chains [BD97b]. This problem was also reintroduced more recently to show lower bounds for distributed Lovász local lemma algorithms [BFH+16], where the goal is to find, instead of to sample, a sink-free orientation.

The formulation is as follows. In an undirected graph $G = (V, E)$, we associate a random variable a_e to indicate the orientation for each edge $e \in E$, and associate a bad event B_v for each $v \in V$ to indicate that v is a sink. Then $\mathsf{var}(B_v) = \{a_e \mid v \text{ is an endpoint of } e\}$, and $|\mathsf{var}(B_v)| = d_v$ where d_v is the degree of v. Condition 1 is easy to verify — if a vertex v is a sink, then none of its neighbours can be a sink. A description of the algorithm is given in Algorithm 5.

Algorithm 5: Sink-popping

```
Orient each edge independently and uniformly at random;
while there is at least one sink do
    Re-orient all edges that are adjacent to sinks uniformly at random;
end
return the final orientation
```

As usual, let Ω_v be the set of orientations where $v \in V$ is the unique sink, $\Omega_1 = \bigcup_{v \in V} \Omega_v$, and Ω_0 be the set of all sink-free orientations. The set Ω_1^{var} is also defined as usual. The general

strategy is once again to "repair" orientations in Ω_v. The first step is to associate a path to each $v' \in V$ such that it can be flipped without creating new sinks, and v' is guaranteed not a sink. For each $(v, v') \in E$, we then flip $v \leftarrow v'$, and flip the path if v' is a sink now. However, there are a few cases where we cannot recover the original orientations if we simply record v' and the other endpoint of the path. For example, if v is along the path, then $v \leftarrow v'$ is flipped twice, and there is no hope to find out v. There are ways to fix these "special" cases, and it is relatively straightforward to design the mapping if we are happy to hardcode each special case. However, to achieve a tight bound, we will do a more complicated mapping so that the special cases can be detected given the image.

A simple observation is that a graph has a sink-free orientation if and only if no connected component of it is a tree. Moreover, the expected running time of Algorithm 5 on a graph with more than one component is simply the sum of the expected running time of each component. We will assume that G is connected and not a tree, and denote $n = |V|$ and $m = |E|$. Since G is not a tree, there must be a cycle C in G. Contract the cycle C, and pick an arbitrary spanning tree in the resulting graph. Denote by $R \subseteq E$ this spanning tree combined with the cycle C. Thus $|R| = |V| = n$ and (V, R) is unicyclic, namely (V, R) is composed of C attached with a number of trees. Define $\mathrm{depth}(v)$ by the distance from v to C in (V, R), and $\mathrm{depth}(v) = 0$ if $v \in C$.

Fix an arbitrary ordering of all vertices and edges of G. Let $\Gamma(v) = \{v' \mid (v, v') \in E\}$ be the neighbourhood of v in G. An equivalent way of writing Ω_1^{var} is $\{(\sigma, v') \mid \sigma \in \Omega_v,\ v' \in \Gamma(v)\}$. We say a vertex is *good* (under an orientation σ) if it has at least two outgoing arcs in (V, R).

Lemma 16. *Let $\sigma \in \Omega_v$ be an orientation with the unique sink v. Restricted to the unicyclic subgraph (V, R), there must be a good vertex u such that $\mathrm{depth}(u) < \max\{1, \mathrm{depth}(v)\}$. In particular, if $\mathrm{depth}(u) = \mathrm{depth}(v) = 0$, u can be chosen so that it has two outgoing arcs in C.*

Proof. First we prune all vertices below v and other trees not containing v attached to C. Call this remaining graph H_v, which is still unicyclic and has as many edges as its vertices. (If $v \in C$ then $H_v = C$.) Clearly v in H_v is still a sink under σ. Namely the out-degree of v is 0 in H_v. The total out-degree is same as the number of vertices in H_v. Thus, there must be a vertex u with out-degree at least 2. By construction u satisfies the other requirements as well. □

We note that $\mathrm{depth}(u) < \max\{1, \mathrm{depth}(v)\}$ implies that $\mathrm{depth}(u) < \mathrm{depth}(v)$ if $v \notin C$, and $\mathrm{depth}(u) = \mathrm{depth}(v) = 0$ otherwise. For $\sigma \in \Omega_v$ and $v' \in \Gamma(v)$, choose a vertex u as follows.

- If $(v, v') \in R$, then let u be the good vertex in Lemma 16 that is closest to v'.
- If $(v, v') \notin R$, then consider whether v' is a sink in the pruned subgraph $H_{v'}$ of (V, R) defined as above.
  - If v' is a sink in $H_{v'}$, then we apply Lemma 16 to $H_{v'}$ and sink v'. Choose the closest good vertex to v'. Note that in this case, if u is on C, then it must have at least one outgoing arc in C and not along the path between u and v'.
  - If v' is not, then we choose $u = v'$.

All ties are broken according to the ordering chosen a priori. Observe that if $u \neq v'$, then $\mathrm{depth}(u) < \max\{1, \mathrm{depth}(v')\}$.

We "repair" σ by flipping a path between u and v' in (V, R). If u and v' are both on C, then we choose the one that does not contain v. (If neither contains v, then we pick the shortest one. Further ties are broken according to the ordering chosen a priori.) Otherwise we simply choose the shortest path between u and v' in (V, R). (Again, ties are broken according to the ordering chosen a priori.) Denote this path by $v_\ell = u, v_{\ell-1}, \ldots, v_1, v_0 = v'$ where $\ell \geq 0$. After flipping the path, we further flip the edge (v, v') and denote the resulting orientation by σ_{fix}.

We claim that $\sigma_{\mathrm{fix}} \in \Omega_0$. If $\ell = 0$, then v' has at least two outgoing arcs and only $v \leftarrow v'$ is flipped. The claim holds. Otherwise $\ell \geq 1$ and none of $u = v_\ell, \ldots, v_2$ can be a sink under σ_{fix}. For u, this is because it is good under σ, and only one of its adjacent edges is flipped. For $v_{\ell-1}, \ldots, v_2$, they cannot have two outgoing arcs under σ since u is the closest good vertex to v'. Thus after flipping, at least one of their adjacent edges along the path is still outgoing. For v_1, v_0 and v, there are two cases:

(1) if $v_1 \neq v$, then v_1 cannot be the sink either due to the same reasoning above. Moreover, since $u \neq v'$, it must be $v_1 \to v' \to v$ under σ, and $v_1 \leftarrow v' \leftarrow v$ under σ_{fix}. Hence v' and v are not sink either;
(2) otherwise $v_1 = v$. In this case $v \leftarrow v'$ is flipped twice. It must be that $\ell \geq 2$, and thus v is not a sink as $v_2 \leftarrow v = v_1$ under σ_{fix}. No orientation of an adjacent edge of v' is changed, so it is still not a sink.

All other vertices are unchanged between σ and σ_{fix}. Thus, $\sigma_{\text{fix}} \in \Omega_0$.

Observe that $v \leftarrow v'$ is flipped twice only if $\text{depth}(v') = \text{depth}(v) + 1$ and $(v, v') \in R$. We say (σ, v') is *special* if $(v, v') \in R$ and,

(1) $\text{depth}(v) = \text{depth}(v') = \text{depth}(u) = 0$, or,
(2) $\text{depth}(v') = \text{depth}(v) + 1$.

Define the mapping $\varphi : \Omega_1^{\mathsf{var}} \to \Omega_0 \times \mathsf{Aux}$:

$$\varphi(\sigma, v') = \begin{cases} (\sigma_{\text{fix}}, u, v') & \text{if } (\sigma, v') \text{ is special;} \\ (\sigma_{\text{fix}}, v, u) & \text{otherwise,} \end{cases} \tag{7}$$

where $\sigma \in \Omega_v$, $v' \in \Gamma(v)$, and Aux is the set of all possible pairs of vertices appearing in the definition above.

We make a simple observation first.

Lemma 17. *Let $v, u, \sigma_{\text{fix}}$ be given as above. If $\text{depth}(v) = \text{depth}(u) = 0$, then u must have at least one outgoing arc in C under σ_{fix}.*

Proof. By our choice of u, it has either two outgoing arcs in C under σ (when u is chosen directly using Lemma 16), or one outgoing arc in C and one pointing towards v' (when u is chosen by applying Lemma 16 to $H_{v'}$). After repairing, σ_{fix} only flips one of the adjacent edge of u, leaving the other outgoing arc unchanged. In particular, in the latter case, the one flipped is not in C. □

The main technical lemma is the following.

Lemma 18. *The mapping φ is 1-preserving and injective.*

Proof. Since there is no weight involved, φ is 1-preserving. To verify the injectivity, we just need to recover (σ, v') from $(\sigma_{\text{fix}}, v_1, v_2)$. We need to figure out whether (σ, v') is special first. There are a few cases:

- First we check if v_1 is a sink under σ_{fix} in (V, R). This happens if and only if $(v, v') \notin R$, $v_1 = v$, and (σ, v') is not special.
- We may now assume $(v, v') \in R$ and v_1 is not a sink. If $v_1 = v_2$, then it must be a special case. Otherwise no matter whether $\text{depth}(v') = \text{depth}(v) \pm 1$ or $\text{depth}(v') = \text{depth}(v) = 0$, $\text{depth}(u) \leq \text{depth}(v)$ and $\text{depth}(u) \leq \text{depth}(v')$. Thus, if $\text{depth}(v_1) < \text{depth}(v_2)$ or $\text{depth}(v_1) > \text{depth}(v_2)$, the one with smaller depth is u, and we can tell whether it is special or not.

  The remaining case is that $\text{depth}(v_1) = \text{depth}(v_2) = 0$. This case must be special, since if not, $v_1 = v$ and $\text{depth}(v) = 0$. Since $(v, v') \in R$, either $\text{depth}(v') = 1$ or $\text{depth}(v') = 0$. Both cases are special. Contradiction.

In summary, we can distinguish the special case and the other one.

If (σ, v') is not special, then v has a unique outgoing arc under σ_{fix}, which is $v \to v'$. We recover v' and thus the path between u and v', and it is easy to figure out the rest.

If (σ, v') is special. We handle the two special cases differently:

(1) if $\text{depth}(v') = \text{depth}(u) = 0$, then the path between v' and u must goes from v' to u under σ_{fix} along C. Moreover, v must be the unique vertex going toward v' under σ_{fix}, since we choose the path between v' and u not passing v.
(2) if $\text{depth}(v') = \text{depth}(v) + 1$, then the path between v' and u is the shortest path, and v must be the ancestor of v' in R (namely the first vertex on the path from v' to u).

In both cases, we can recover the path between v' and u as well as the original sink v. Given these information, recovering σ is straightforward. □

Lemma 18 directly gives an n^2 upper bound for $\frac{|\Omega_1^{\mathsf{var}}|}{|\Omega_0|}$. We can improve it to $n(n-1)$. For any fixed σ_{fix}, we want to bound the number of pairs of vertices that can be the possible output of φ along with σ_{fix}. If v is fixed in the non-special case of (7), then $u \neq v$. In the special case, if u is fixed, then in the first special case, v' cannot be the vertex u points to along C under σ_{fix} (recall Lemma 17); whereas in the second special case, v' cannot be u. Thus Lemma 18 implies that $\frac{|\Omega_1^{\mathsf{var}}|}{|\Omega_0|} \leq n(n-1)$. Then (2) yields the following theorem.

Theorem 19. *Let $G = (V, E)$ be a connected graph that is not a tree. The expected number of random variables drawn in Algorithm 5 on G to sample a sink-free orientation is at most $m + n(n-1)$, where $n = |V|$ and $m = |E|$.*

It is easy to see that Theorem 19 is tight, since on a cycle of length n the upper bound is achieved.

6. Concluding remarks

Perhaps the most interesting open problem is whether there are faster algorithms to sample root-connected subgraphs or sink-free orientations. For spanning trees, Kelner and Mądry [KM09] has shown that the random walk based approach can be accelerated to $O(m\sqrt{n})$, which has kindled a sequence of improvements [MST15, DKP+17, DPPR17]. This line of research culminates in the almost-linear time algorithm by Schild [Sch18]. It would be interesting to see whether these (such as graph sparsification and fast linear system solver) or other ideas can be used to accelerate cluster-popping and sink-popping as well.

Acknowledgements

Part of the work was done while HG was visiting the Institute of Theoretical Computer Science, Shanghai University of Finance and Economics, and he would like to thank their hospitality. HG would also like to thank Mark Jerrum for helpful discussion. KH would like to thank Xiaoming Sun for helpful discussion.

References

[Bar16] Alexander Barvinok. *Combinatorics and complexity of partition functions*, volume 30 of *Algorithms and Combinatorics*. Springer, 2016. 1

[BD97a] Russ Bubley and Martin E. Dyer. Graph orientations with no sink and an approximation for a hard case of #SAT. In *SODA*, pages 248–257, 1997. 2, 14

[BD97b] Russ Bubley and Martin E. Dyer. Path coupling: A technique for proving rapid mixing in Markov chains. In *FOCS*, pages 223–231, 1997. 14

[BFH+16] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In *STOC*, pages 479–488, 2016. 14

[BG08] Antar Bandyopadhyay and David Gamarnik. Counting without sampling: Asymptotics of the log-partition function for certain statistical physics models. *Random Struct. Algorithms*, 33(4):452–479, 2008. 1

[CPP02] Henry Cohn, Robin Pemantle, and James G. Propp. Generating a random sink-free orientation in quadratic time. *Electron. J. Combin.*, 9(1):10:1–10:13, 2002. 2, 3, 14

[DFK91] Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991. 1

[DKP+17] David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *STOC*, pages 730–742, 2017. 17

[DPPR17] David Durfee, John Peebles, Richard Peng, and Anup B. Rao. Determinant-preserving sparsification of SDDM matrices with applications to counting and sampling spanning trees. In *FOCS*, pages 926–937, 2017. 17

[Eri96] Kimmo Eriksson. Strong convergence and a game of numbers. *European J. Combin.*, 17(4):379–390, 1996. 9

[GJ18a] Heng Guo and Mark Jerrum. Approximately counting bases of bicircular matroids. *CoRR*, abs/1808.09548, 2018. 3, 7

[GJ18b] Heng Guo and Mark Jerrum. Perfect simulation of the hard disks model by partial rejection sampling. In *ICALP*, volume 107 of *LIPIcs*, pages 69:1–69:10, 2018. 3

[GJ18c] Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. In *ICALP*, volume 107 of *LIPIcs*, pages 68:1–68:12, 2018. 1, 2, 3, 5, 9, 10

[GJL17] Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovasz local lemma. In *STOC*, pages 342–355, 2017. 1, 2, 3, 4

[GN06] Omer Giménez and Marc Noy. On the complexity of computing the Tutte polynomial of bicircular matroids. *Combin. Probab. Comput.*, 15(3):385395, 2006. 3

[GP14] Igor Gorodezky and Igor Pak. Generalized loop-erased random walks and approximate reachability. *Random Struct. Algorithms*, 44(2):201–223, 2014. 1, 2, 3, 4, 5, 10

[Hub98] Mark Huber. Exact sampling and approximate counting techniques. In *STOC*, pages 31–40, 1998. 2

[Jer81] Mark Jerrum. On the complexity of evaluating multivariate polynomials. *Ph.D. dissertation*. Technical Report CST-11-81, Dept. Comput. Sci., Univ. Edinburgh, 1981. 2

[Jer03] Mark Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003. 1

[JS93] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993. 1

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004. 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. 1

[KM09] Jonathan A. Kelner and Aleksander Mądry. Faster generation of random spanning trees. In *FOCS*, pages 13–21, 2009. 17

[Kol18] Vladimir Kolmogorov. A faster approximation algorithm for the Gibbs partition function. In *COLT*, volume 75 of *PMLR*, pages 228–249, 2018. 10

[KS11] Kashyap Babu Rao Kolipaka and Mario Szegedy. Moser and Tardos meet Lovász. In *STOC*, pages 235–244, 2011. 4

[Moi17] Ankur Moitra. Approximate counting, the Lovasz local lemma, and inference in graphical models. In *STOC*, pages 356–369. ACM, 2017. 1

[MST15] Aleksander Mądry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *SODA*, pages 2019–2036, 2015. 17

[MT10] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász Local Lemma. *J. ACM*, 57(2), 2010. 1, 3

[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. 2

[PR17] Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM J. Comput.*, 46(6):1893–1919, 2017. 1

[PW98] James G. Propp and David B. Wilson. How to get a perfectly random sample from a generic markov chain and generate a random spanning tree of a directed graph. *J. Algorithms*, 27(2):170–217, 1998. 1, 2, 3, 12, 14

[Sch18] Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *STOC*, pages 214–227, 2018. 2, 17

[She85] James B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985. 3

[Tar72] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. 2, 7

[Tod91] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. 1

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. 1

[Wei06] Dror Weitz. Counting independent sets up to the tree threshold. In *STOC*, pages 140–149. ACM, 2006. 1

[Wil96] David B. Wilson. Generating random spanning trees more quickly than the cover time. In *STOC*, pages 296–303, 1996. 2, 3, 12, 14

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

Email address: hguo@inf.ed.ac.uk

(Kun He) Institute of Computing Technology, Chinese Academy of Sciences, No.6 Kexueyuan South Road, 100190, Beijing, China

Email address: hekun@ict.ac.cn