GENERALIZED LOOP-ERASED RANDOM WALKS AND APPROXIMATE REACHABILITY

IGOR GORODEZKY* AND IGOR PAK†

Abstract. In this paper we extend the *loop-erased random walk* (LERW) to the directed hypergraph setting. We then generalize Wilson's algorithm for uniform sampling of spanning trees to directed hypergraphs. In several special cases, this algorithm perfectly samples spanning hypertrees in expected polynomial time.

Our main application is to the *reachability problem*, also known as the directed all-terminal network reliability problem. This classical problem is known to be $\#P$-complete, hence is most likely intractable [BP2]. We show that in the case of *bi-directed graphs*, a conjectured polynomial bound for the expected running time of the generalized Wilson algorithm implies a FPRAS for approximating reachability.

1. Introduction

In the past several decades, MCMC methods have revolutionized the theory of computing, giving new tools for random sampling and approximate counting of combinatorial objects, even in cases of #P-completeness. Network reliability problems are some of the oldest and most classical NP- and #P-complete problems [Col] (see also [BP1, BP2, KL]). In fact, the two-terminal versions of directed and undirected network reliability were two of Valiant's original thirteen #P-complete problems [Val]. Due to the inherent difficulty of network reliability problems, especially in the directed graph case, little progress has been made towards their approximate solution (with the notable exception of Karger's fully polynomial randomized approximation scheme (FPRAS) for undirected all-terminal reliability [Kar], see Subsection 10.6). In this paper we propose a new approach to this problem based on an extension of the loop-erased random walk (LERW) technique.

Although classical, LERW came into prominence after Wilson's algorithm [Wil], which showed that a loop-erased random walk can be used for uniform sampling of spanning trees in both directed and undirected graphs. This led to partial unification and advancements in both topics [Law] (see also [LSW, Mar]). Despite a number of interesting potential applications, until now no generalizations of LERW and Wilson's algorithm have been found.

Our main result is Theorem 2.2 which shows that the generalized Wilson algorithm generates random spanning (directed) hypertrees in a given (directed) hypergraph, from a uniform distribution. Although the basic idea of the proof is superficially similar to Wilson's, the delicate definition of the hypertrees is what makes the result possible, with the difficulties in the proof hidden by these technicalities. In fact, since our main application is to directed reachability, which uses a more elegant language, to ease the language barrier we state the proof of the more general Theorem 2.2 in that language as well.

The sampling of random trees is a well-studied problem with connections to graph polynomials and their complexity (see [Wel]). The problem is also at the heart of several random walks studies, with generalizations to matroids and other combinatorial structures (see Subsection 10.4). Unfortunately,

*Center for Applied Mathematics, Cornell University, Ithaca, NY 14850, USA, and
Palantir Technologies, Palo Alto, CA 94301, USA. Email: i.gorodezky@gmail.com.

†Department of Mathematics, UCLA, Los Angeles, CA 90095, USA. Email: pak@math.ucla.edu.

little if any progress has been made in sampling of random hypertrees. This is in part due to their difficult structure and the fact that there are many fundamentally different definitions of hypertrees (see Subsection 10.2). The type of hypertrees studied in this paper are uniquely suited for random sampling. The root-connected subgraphs that we sample to approximate reachability are just one important special case of our hypertrees (see below).

Finally, let us mention that as a byproduct of our approach, the hypertrees (and root-connected subgraphs) that we sample come from an exactly uniform distribution. This gives a new example of *perfect sampling*, with applications of independent interest (see Subsection 10.5).

The rest of the paper is structured as follows. In the next section (Section 2) we give definitions and state the main result. In Section 3 we present a construction and a complete proof of correctness of Wilson's algorithm. Although these results are not new and are close to the original presentation [Wil], they prepare the reader for the (similar but more technical) proof of the main theorem which we give in Sections 4 and 5. We then consider the running time of the algorithm in Section 7. Examples and special cases are given in Section 6. The last two sections (Section 8 and 9) have more traditional CS features: we prove that the bi-directed version of the reachability problem is #P-complete, and construct a FPRAS for reachability in bi-directed graphs under the conjecture that our algorithm runs in expected polynomial time in this case. We conclude with final remarks and open problems in Section 10.

2. Definitions and main results

Here we only state the main results, which will be repeated and proved later in the paper.

2.1. **Hypergraphs and hypertrees.** Let V be a finite set of vertices, and let $R = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots\} \subseteq V$ be a set of *root vertices*, or simply *roots*. A *directed hyperedge* is a subset $B \subseteq V$, with a designated *tail vertex* $\tau = \tau(B) \in B \setminus R$. In other words, the tail can never be a root vertex. A *rooted directed hypergraph* is a pair $\mathcal{H} = (V, \mathcal{E})$, where $\mathcal{E} \subseteq 2^V$ is a subset of directed hyperedges. Throughout the paper we consider only directed hyperedges and rooted directed hypergraphs, so in the future, to simplify the notation, we refer to *hyperedges* and *hypergraphs*.

We say that one *goes from x to y along B*, writen $x \to_B y$, if $x = \tau(B)$ and $y \in B$, $y \neq x$. More generally, we say that one *goes from x to y along hyperedges in $\mathcal{H} = (V, \mathcal{E})$*, if

$$x = v_1 \to_{B_1} v_2 \to_{B_2} v_3 \to \ldots \to_{B_\ell} v_{\ell+1} = y\,, \qquad \text{where } \; v_i = \tau(B_i), B_i \in \mathcal{E}\,.$$

In words, when walking along hyperedges one walks from tail vertex to tail vertex.

A hypergraph $\mathcal{H}' = (V, \mathcal{E}')$ is called a *spanning hypertree* in $\mathcal{H} = (V, \mathcal{E})$, if $\mathcal{E}' \subset \mathcal{E}$, every non-root vertex is a tail of exactly one hyperedge, and from every vertex $v \in V$ one can go to some root vertex $\boldsymbol{r}_s$ along the hyperedges in $\mathcal{E}'$.

Number of Spanning Hypertrees (#SH) Problem
Input: A directed hypergraph $\mathcal{H} = (V, \mathcal{E})$ with root set R.
Output: The number of spanning hypertrees in $\mathcal{H}$.

In a special case of a directed graph $G = (V, E)$ and a single root vertex $\boldsymbol{r} \in V$, spanning hypertrees are the (usual) trees directed towards the root. This problem can be solved in polynomial time using the (directed) matrix-tree theorem. On the other hand, for general hypergraphs the problem is #P-complete (as we later prove).

The generalized Wilson algorithm, defined below, is an algorithm for uniformly generating spanning hypertrees of directed hypergraphs. It therefore can be used to count them approximately.

Remark 2.1. In contrast with other definitions of hypertrees (see Subsection 10.2), there is not necessarily a hyperedge B with tail $v = \tau(B)$, such that $(V - v, \mathcal{E} - B)$ is a smaller hypertree. The hypertree with two hyperedges $(1 \to 2, \boldsymbol{r})$ and $(2 \to 1, \boldsymbol{r})$ is the simplest example. Note also that not every hypergraph contains a spanning hypertree as the natural connectivity condition is a necessary condition.

2.2. **Sampling hypertrees.** In this section we will define a *generalized loop-erased random walk* (GLERW) and the generalized Wilson algorithm. First, a few definitions. In a hypergraph $\mathcal{H} = (V, \mathcal{E})$ the set $S \subseteq V \setminus R$ is called *strongly connected* if $x \to_{\mathcal{E}} y$ for all $x, y \in S$. The set $S \subseteq V \setminus R$ is called *closed* if $x \to_{\mathcal{E}} y$ and $x \in S$ implies that $y \in S$. Finally, a *cycle* in $\mathcal{H}$ is a set of vertices $S \subseteq V \setminus R$ that is closed and strongly connected, and minimal with respect to these properties. Note that by definition, an isolated vertex is also a cycle.

Now we can define the GLERW. Fix a hypergraph $\mathcal{H} = (V, \mathcal{E})$, and a family of probability distributions Φ_v on all hyperedges $B \in \mathcal{E}$ with tail at v (that is, $\tau(B) = v$), for all $v \in V$. Consider the following Markov process $\{X_t, t = 0, 1, \ldots\}$. Start with $X_0 = (v, \varnothing)$. For $t \geq 0$, suppose X_t is a hypergraph $(V_t, \mathcal{E}_t)$, where $V_t \subseteq V$ and $\mathcal{E}_t \subseteq \mathcal{E}$, and such that every vertex is a tail of at most one hyperedge. For every $v \in V$ which is *not* a tail of a hyperedge, sample a hyperedge from Φ_v. Add to X_t all new hyperedges and vertices in these hyperedges. Now find all the resulting cycles in the hypergraph created, and remove those whose tails are cycle vertices.[1] Let $X_{t+1} = (V_{t+1}, \mathcal{E}_{t+1})$ be the set of resulting vertices and hyperedges.

Note that in a directed graph G as above, GLERW coincides with the usual LERW (described in Section 3). In particular, the set of vertices and hyperedges can both increase and decrease. Note also that $X_t = X_{t+1}$ only for a hypertree, but not necessarily a spanning hypertree.

The *generalized Wilson algorithm* is now defined as follows. Choose a vertex v and run a GLERW. Once a hypertree is obtained, if it is not a spanning hypertree, choose another vertex and run a GLERW until a larger hypertree is obtained. Repeat until a spanning hypertree T is obtained. Output T.

Theorem 2.2 (Main theorem)**.** *The generalized Wilson algorithm outputs spanning hypertree $T = (V, \mathcal{E}')$ with probability proportional to*

$$\prod_{B \subseteq \mathcal{E}'} \Phi_{\tau(B)}(B).$$

2.3. **Approximate reachability.** We begin by stating the following classical problem:

The Directed Reachability (DR) Problem (directed all-terminal network reliability)
Input: A directed graph $G = (V, A)$ with root $\boldsymbol{r}$, and a rational $0 < p < 1$.[2]
Output: The probability that there remains a directed path from every non-root vertex to $\boldsymbol{r}$ if all directed edges are allowed to fail independently with probability p.

Given a directed graph $G = (V, A)$ with root $\boldsymbol{r}$, a *root-connected* spanning subgraph is a subgraph with vertex set V that contains a directed path from every non-root vertex to $\boldsymbol{r}$. Therefore the reachability problem asks to compute the probability that independent edge failures result in a root-connected spanning subgraph. Since we will only be interested in spanning subgraphs throughout the paper, we will generally omit "spanning" when discussing *root-connected* spanning subgraphs.

Let $(G, \boldsymbol{r}, p)$ be an instance of Directed Reachability. The probability that some $A' \subseteq A$ remains after all edges are allowed to fail independently is

$$p[A'] = p^{|A|-|A'|}(1-p)^{|A'|}. \tag{2.1}$$

[1] This step requires delicacy as a priori it is not well defined. We further discuss this in Subsection 3.3.

[2] We require that p be rational for complexity-theoretic reasons.

Our approach to approximating reachability will be to show that the generalized Wilson algorithm can be used to perfectly sample root-connected subgraphs in directed graphs, and then use standard techniques to leverage this sampling algorithm into an approximation algorithm for the Directed Reachability problem. By *perfect-sampling algorithm* we mean an algorithm such that if (V, A') is a root-connected spanning subgraph then our algorithm produces it with probability proportional to $p[A']$ from equation 2.1.

How can the generalized Wilson algorithm relate help with sampling root-connected subgraphs? Fix a directed graph $G = (V, A)$ with root $\boldsymbol{r}$. Given a vertex $v \in V$, let $A^+(v)$ be the set of its out-edges in G, that is,

$$A^+(v) = \{e \in A \mid e = (v \to u) \text{ for some } u \in V\}.$$

We will refer to a subset of $A^+(v)$ as a *rowel*. Define a directed hypergraph $\mathcal{H} = (V, \mathcal{E})$ with root set $R = \{\boldsymbol{r}\}$ by setting its set of hyperedges to be the set of all rowels; the rowel $B \subseteq A^+(v)$ has v as its tail. We call this the *derived hypergraph*.

Now observe that the spanning hypertrees in $\mathcal{H}$ correspond exactly to the root-connected subgraphs of G. Moreover, given some rational failure probability p, we will show that if we set $\Phi_v(B)$ appropriately, then running the generalized Wilson algorithm on $\mathcal{H}$ will sample root-connected subgraphs perfectly.

The application to Directed Reachability so simplifies the generalized Wilson algorithm that for clarity of presentation we will from now on for the most part discuss the latter as it applies to the former problem. In particular, if G is a rooted directed graph then "running Wilson's algorithm on G" should be understood as running the algorithm on the derived hypergraph $\mathcal{H}$ as defined above.

2.4. **Bi-directed reachability.** Our approximation algorithm provides a provably good approximation, but in general it will not run in polynomial time. We conjecture that it is polynomial time in the case of *bi-directed graphs*, which are directed graphs in which the existence of an edge $(u \to v)$ implies the existence of the edge $(v \to u)$ (the two edges are allowed to fail independently). We emphasize this class of graphs due to the following sequence of results.

Theorem 8.1. *Bi-directed reachability is #P-complete.*

Conjecture 7.1. (Main Conjecture) *If $\{G = (V, A), \mathbf{r} \in V, 0 < p < 1\}$ is an instance of reachability and G is bi-directed, then the expected running time of the generalized Wilson algorithm is polynomial in the size of G and $(1 - p)^{-1}$.*

Theorem 7.2. *The Main Conjecture implies a FPRAS for the reachability problem on bi-directed graphs.*

3. Wilson's original algorithm

3.1. **Statement of the algorithm.** Our work on approximating reachabililty will rely on a generalization of Wilson's algorithm for uniformly generating rooted spanning trees. Therefore we begin by describing Wilson's algorithm and its analysis; see the original paper [Wil] for details we omit. Given a connected directed graph $G = (V, E)$ and root vertex $\boldsymbol{r}$, Wilson's algorithm uniformly generates a directed spanning tree of G *rooted at* $\mathbf{r}$, meaning a directed spanning tree whose edges are oriented towards $\boldsymbol{r}$.

Wilson's algorithm can be stated as a *loop-erased random walk* on G as follows. The algorithm maintains a subtree T, initialized to consist of $\boldsymbol{r}$ alone. While there remain vertices not in T, the algorithm performs a random walk in G from one such vertex v, erasing loops (i.e. cycles) as they are created, until the walk encounters a vertex in T. Then the cycle-erased simple path from v to T is added to T. This algorithm clearly outputs a random directed spanning tree oriented towards $\boldsymbol{r}$;

it is a minor miracle that this output is in fact distributed uniformly over all such trees. We prove this fact in this section.

Let us begin by stating and analyzing a seemingly different algorithm, the *cycle-popping* algorithm. We will prove that this algorithm has the desired properties, and then argue that it is equivalent to the loop-erased random walk (LERW).

The cycle-popping algorithm works as follows. Given G and $\boldsymbol{r}$, associate with each non-root vertex v an infinite *stack* of directed edges whose tail is v. More formally, to each $v \neq \boldsymbol{r}$ we associate

$$\mathcal{S}_v = [e_0, e_1, e_2, \dots],$$

where each e_i is uniformly (and independently) sampled from the set of edges $\{e \in E \mid e = (v \to u)\}$. Each stack is sampled independently of all other.

We refer to the left-most element above as the *top* of $\mathcal{S}_v$, and by *popping* the stack $\mathcal{S}_v$ we mean removing this top edge from $\mathcal{S}_v$. Define the *stack graph* $G_{\mathcal{S}}$ to be the directed graph on V induced by the set of edges at the tops of the stacks.

If there is a directed cycle C in $G_{\mathcal{S}}$ we may pop it, by which we mean pop $\mathcal{S}_v$ for every $v \in C$. This creates a new stack graph $G_{\mathcal{S}}$.

Wilson's cycle-popping algorithm:

(1) Create a stack $\mathcal{S}_v$ for every $v \neq \boldsymbol{r}$.
(2) While $G_{\mathcal{S}}$ contains directed cycles, pop a cycle from the stacks.
(3) If this process ever terminates, output $G_{\mathcal{S}}$.

3.2. **Analysis.** It is immediate that a stack graph $G_{\mathcal{S}}$ either is a directed spanning tree rooted at $\boldsymbol{r}$, or must contain a directed cycle. Therefore if the cycle-popping algorithm ever terminates then its output is a spanning tree rooted at $\boldsymbol{r}$. We will show that as long as such a directed spanning tree exists in G then the algorithm terminates with probability 1, and moreover generates spanning trees rooted at $\boldsymbol{r}$ uniformly.

To this end, some more definitions: let us say that given a stack $\mathcal{S}_v$, the edge e_i is at *level* i. The level of an edge in a stack is static, and is defined when the stack is created. That is, the level of e_i does not change even if e_i advances to the top of the stack as a result of the stack getting popped. We regard the sequence of stack graphs produced by the algorithm as *leveled* stack graphs by labeling each directed edge with its level in the stack associated to its tail. In the same way, we regard cycles encountered by the algorithm as leveled cycles, and we can regard the tree produced by the algorithm (if indeed one is produced) as a leveled tree.

The analysis of the algorithm relies on the following commutativity (or "diamond") lemma (Theorem 4 in [Wil]), which tells us that the order in which the algorithm pops cycles is irrelevant.

Lemma 3.1. *For a given set of stacks, either the cycle-popping algorithm never terminates, or there exists a unique leveled spanning tree T rooted at $\mathbf{r}$ such that the algorithm outputs T irrespective of the order in which cycles are popped.*

Proof. Fix a set of stacks $\{\mathcal{S}_v\}_{v \neq r}$. Consider a leveled cycle C that is *pop-able*, i.e. there exists a sequence of leveled cycles ending in C

$$C_1, C_2, \dots, C_k = C$$

that can be popped one after the other. We claim that if the algorithm pops any leveled cycle not equal to C, then there still must exist a series of leveled cycles that ends in C and that can be popped in sequence. Less formally, we claim that if C is pop-able then it remains pop-able until it is popped.

Let C' be a leveled cycle popped by the algorithm. If C' shares no vertices with $C_1, \ldots, C_k$, then the claim is clearly true. Assume then that C' shares a vertex with a leveled cycle in $C_1, \ldots, C_k$ and let C_i be the first in this sequence for which this is true. We will show that $C' = C_i$.

Assume for the sake of contradiction that $C_i \neq C'$. Then C_i and C' must share a vertex w that has different successors in C' and C_i. Thus there must exist two distinct edges $e \in C_i$ and $e' \in C'$ that have the same tail w. On the other hand, none of the $C_1, \ldots, C_{i-1}$ contain w by definition of C_i. This implies that e and e' have the same level in the stack $\mathcal{S}_w$ and must therefore be equal, a contradiction. This proves $C_i = C'$. It follows that

$$C' = C_i, C_1, C_2, \ldots, C_{i-1}, C_{i+1}, \ldots, C_k = C$$

is a series of leveled cycles that can be popped in sequence. This proves our claim that C remains pop-able no matter which non-C cycle is popped.

We conclude that given a set of stacks either there is an infinite number of pop-able cycles, in which case the algorithm will never terminate, or there is a finite number of such cycles. In the latter case, every one of these cycles is eventually popped, and the algorithm produces a spanning tree T rooted at $\boldsymbol{r}$. Given a vertex $v \neq \boldsymbol{r}$, the level of the (unique) edge in T whose tail is v is given by the number of popped cycles that contained v. □

Theorem 3.2. *If there exists a directed spanning tree in G oriented towards $\mathbf{r}$ then the cycle-popping algorithm terminates with probability 1, and the tree that it outputs is a uniformly sampled spanning tree rooted at $\mathbf{r}$.*

Proof. Let T be a directed spanning tree in G rooted at $\boldsymbol{r}$; the stacks generated in the first step of the algorithm will contain T, and hence the algorithm will terminate, with probability 1. This proves the first part of the claim.

For the second part, fix a spanning tree T rooted at $\boldsymbol{r}$ and let $\mathbf{T}$ be the event that T is produced by the algorithm. Given a collection of leveled cycles $\mathcal{C}$, write $\boldsymbol{\mathcal{C}}$ for the event that $\mathcal{C}$ is the set of leveled cycles popped by the algorithm before it terminates. Finally, let $\boldsymbol{\mathcal{C}} \wedge \mathbf{T}$ be the event that the algorithm popped the leveled cycles in $\mathcal{C}$ and then terminated, with the resulting *leveled* tree equal to T. We will prove the second part of the claim by showing that $\mathbf{P}[\mathbf{T}]$ is independent of T.

Let q_T be the probability that after sampling an edge from each stack, the resulting stack graph is equal to T. By the independence of the stacks and of the entries within any given stack, we have

$$\mathbf{P}[\boldsymbol{\mathcal{C}} \wedge \mathbf{T}] = \mathbf{P}[\boldsymbol{\mathcal{C}}] \cdot q_T.$$

It follows that

$$\mathbf{P}[\mathbf{T}] = \sum_{\mathcal{C}} \mathbf{P}[\boldsymbol{\mathcal{C}} \wedge \mathbf{T}] = q_T \sum_{\mathcal{C}} \mathbf{P}[\boldsymbol{\mathcal{C}}].$$

Since q_T is clearly independent of T, the proof is complete. □

3.3. **Equivalence of cycle-popping and LERW.** We have shown that the cycle-popping algorithm uniformly generates spanning trees rooted at $\boldsymbol{r}$ (assuming at least one exists). It remains to observe that the LERW algorithm is equivalent to the cycle-popping algorithm. Instead of initially generating the (infinitely long) stacks and then looking for cycles to pop, the LERW generates stack elements as necessary.[3] If the LERW encounters a loop, then it has found a cycle in the stack graph induced by the stacks that the LERW generates. Erasing the loop is equivalent to popping this cycle. Theorem 3.2 now implies that the LERW algorithm generates spanning trees rooted at $\boldsymbol{r}$ uniformly.

[3]Computer scientists will recognize this as the Principle of Deferred Decisions.

3.4. **Running time.** Wilson proves the following in [Wil] (we omit the proof for the sake of brevity).

Theorem 3.3. *Let G be a connected graph with cover time C. The expected number of stack pops that take place in the execution of the cycle-popping algorithm on G is $O(C)$.* □

Since the number of stack pops dominates the running time, this shows that the expected running time is $O(C)$. It is well known that the cover time of bi-directed graphs is polynomially bounded (a proof can be found, for instance, in [MR]). Thus for bi-directed graphs, Wilson's algorithm runs in expected polynomial time.

The bi-directedness condition is necessary, since there are examples of directed graphs on which Wilson's algorithm takes exponential time in expectation. See Figure 1 for a simple example: a LERW would take expected exponential time to reach the root.

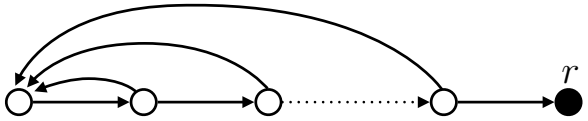

FIGURE 1. An example of a directed graph for which a LERW takes exponential time in expectation.

4. Generalizing Wilson's algorithm: cluster popping

In this section we present the cluster-popping algorithm, which is equivalent to the generalized Wilson's algorithm as originally defined in Section 2.2. In analogy to Wilson's original argument, we will prove facts about the generalized Wilson algorithm by analyzing the cluster-popping algorithm.

In this section, G will be a directed, rooted graph (V, A) with root $\boldsymbol{r}$.

4.1. **Setup: clusters.** Let $G' = (V, A')$ be a subgraph of G (note that G' contains all vertices of G) and $U \subseteq V \setminus \{\boldsymbol{r}\}$ a subset of vertices. We say that U is a *cluster in* G' if all edges in G' whose tail is in U also point into U. That is, U is a cluster if all edges in G' whose tail is in U are in the subgraph induced by U. If G' is clear from context we will simply say that U is a cluster.

Example. We illustrate this definition in Figure 2. The set of selected vertices in (a) is not a cluster in the pictured graph because vertex 3 is the tail of an edge that points out of the set. The set of vertices in (b) is a cluster; all edges whose tail is in the set point at another vertex in the set. The set of vertices in (c) is a *minimal cluster*, which the set in (b) is not.

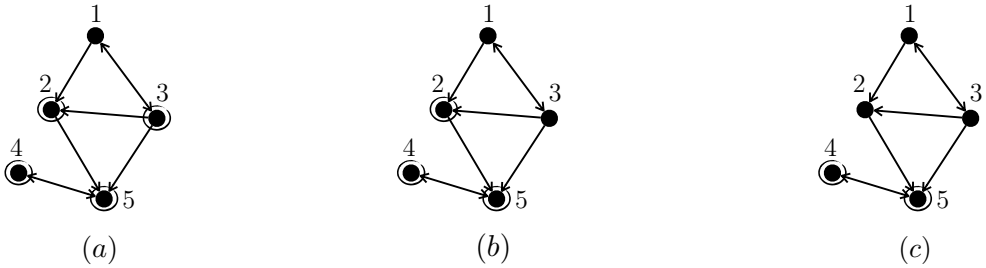

FIGURE 2. An example.

Observe that the minimal clusters in G correspond to the cycles in the derived hypergraph $\mathcal{H}$ (as defined in Section 2.3; see also Section 2.2 for the definition of a cycle in a hypergraph).

4.2. **Statement of the algorithm.** Our algorithm will parallel the cycle-popping formulation of Wilson's algorithm. We will be referencing the definitions from Section 2.3.

Define for each vertex $v \neq \boldsymbol{r}$ a *stack*

$$\mathcal{S}_v = [R_0, R_1, R_2, \dots],$$

where each R_i is randomly chosen rowel, sampled by letting all edges in $A^+(v)$ fail independently and setting R_i to be the set of surviving edges. More formally, given a rowel R, we have

$$\mathbf{P}[R_i = R] = p^{|A^+(v)|-|R|}(1-p)^{|R|}. \tag{4.1}$$

All R_i are sampled independently, and all stacks are independent as well.

We refer to the left-most rowel in a stack as the *top* of $\mathcal{S}_v$, and by *popping* the stack $\mathcal{S}_v$ we mean removing this top rowel from $\mathcal{S}_v$.

Define the *stack graph* $G_{\mathcal{S}}$ to be the spanning subgraph of G induced by the collection of stacks $\mathcal{S}$, i.e. the subgraph whose edge set is the union of the rowels at the tops of the stacks. If $G_{\mathcal{S}}$ contains a cluster U we may pop it, by which we mean pop $\mathcal{S}_v$ for every $v \in U$. This creates a new $G_{\mathcal{S}}$.

CLUSTER-POPPING ALGORITHM:

(1) Create a stack $\mathcal{S}_v$ for every $v \in V \setminus \{\boldsymbol{r}\}$.
(2) While $G_{\mathcal{S}}$ contains any clusters, pop a minimal cluster from the stacks.
(3) If this process ever terminates, output $G_{\mathcal{S}}$.

Note that we only allow the popping of minimal clusters. This is a crucial distinction, as will become clear in the next section.

5. Analysis of the cluster-popping algorithm

It follows immediately from the definitions in Sections 4.1 and 4.2 that given a stack graph $G_{\mathcal{S}}$, either there is a directed path in $G_{\mathcal{S}}$ from every $v \neq \boldsymbol{r}$ to $\boldsymbol{r}$, or $G_{\mathcal{S}}$ contains $U \subseteq V \setminus \{\boldsymbol{r}\}$ that is a cluster in $G_{\mathcal{S}}$. Therefore, if the cluster-popping algorithm ever terminates then its output is a root-connected subgraph.

5.1. **Perfect sampling.** We will prove that the cluster-popping algorithm perfectly samples root-connected subgraphs. The first step is an analogue of Lemma 3.1, the commutativity lemma from the cycle-popping algorithm. To prove it we will make use of the following simple observation.

Lemma 5.1. *Let G', G'' be two subgraphs of $G = (V, A)$ that both contain every vertex in V, and let $U', U'' \subseteq V \setminus \{\mathbf{r}\}$ be two vertex subsets such that U' is a cluster in G' and U'' is a cluster in G''. If $U' \cap U'' \neq \varnothing$ then either $U = U'$ or there exists $v \in U' \cap U''$ whose out-neighborhoods in G' and G'' are not equal.*

Proof. The proof is essentially an unpacking of the definitions. Choose some $w \in U' \cap U''$ and note that by minimality, U' is the set of vertices in G' reachable from w and U'' is the set of vertices in G'' reachable from w. We perform a depth-first search starting from w in both G' and G''; we either eventually find a vertex in $U' \cap U''$ with different out-neighborhoods in G' and G'', or otherwise conclude that the same set of vertices is reachable from w in G' and in G'', meaning $U' = U''$. □

Now we can prove an analogue of Lemma 3.1.

Lemma 5.2. *Given a directed graph G with root $\mathbf{r}$ and a collection of stacks, either the cluster-popping algorithm never terminates or there exists a unique root-connected subgraph G' such that the algorithm outputs G' irrespective of the order in which minimal clusters are popped.*

Proof. Fix a set of stacks $\{\mathcal{S}_v\}_{v\neq r}$. Consider a minimal cluster U that is pop-able, i.e. there exist minimal clusters

$$U_1, U_2, \dots, U_k = U$$

that can be popped in sequence. As before, we claim that if the algorithm pops any minimal cluster not equal to U, then there still must exist a series of minimal clusters that ends in U and that can be popped in sequence. Less formally, we claim that if U is pop-able then it remains pop-able until it is popped.

Let U' be a minimal cluster popped by the algorithm. Note that if $U' \cap U_i = \varnothing$ for $i = 1, \dots, k-1$ then $U_1, \dots, U_k = U$ is a pop-able sequence even after popping U', and the claim holds. Let us assume then that $U' \cap U_i \neq \varnothing$ for some i. Indeed, let i be the smallest index for which this is true. We will show that $U' = U_i$.

Let G' be the stack graph induced by the stacks before U' or any U_j is popped, and let G'' be the stack graph induced by the stacks after $U_1, \dots, U_{i-1}$ are popped in sequence (but U' is never popped). Observe that U' is a cluster in G' while U_i is a cluster in G''.

Assume for the sake of contradiction that $U' \neq U_i$. Since $U' \cap U_i \neq \varnothing$, Lemma 5.1 tells us that there exists $v \in U' \cap U_i$ with different out-neighborhoods in G' and G''. In other words, the rowels at the top of the stack $\mathcal{S}_v$ are different in the two stack graphs G' and G''.

On the other hand, none of the $U_1, \dots, U_{i-1}$ can include v by definition of U_i, since otherwise $U' \cap U_j \neq \varnothing$ for some $j < i$. That is, $\mathcal{S}_v$ is never popped when popping U_j for $j < i$. This implies that the out-neighborhoods of v are the same in G' and G'', a contradiction. This proves $U' = U_i$.

It follows that

$$U' = U_i, U_1, U_2, \dots, U_{i-1}, U_{i+1}, \dots, U_k = U$$

is a series of minimal clusters that can be popped in sequence. This proves our claim that U remains pop-able no matter which non-U cluster is popped. The rest of the proof follows Lemma 3.1. □

The following is an analogue of Theorem 3.2.

Theorem 5.3. *Given a directed G with root $\mathbf{r}$, if G contains a spanning tree rooted at $\mathbf{r}$, then the cluster-popping algorithm terminates with probability 1. Moreover, if the algorithm terminates, then its output is a perfectly sampled root-connected subgraph.*

Proof. If G contains a spanning tree rooted at $\boldsymbol{r}$ then it also contains at least one root-connected spanning subgraph, and this subgraph will appear in a collection of stacks $\{\mathcal{S}_v\}$ with probability 1. This proves the first part of the claim.

To prove perfect sampling, we need to show that the probability that the algorithm outputs a given root-connected subgraph (V, A') is proportional to $p[A']$ as defined in equation (2.1). Let $(\mathbf{V}, \mathbf{A}')$ be the event that (V, A') is produced by the algorithm, so that our goal is to show $\mathbf{P}[(\mathbf{V}, \mathbf{A}')] \propto p[A']$.

Given an ordered set of clusters $\mathcal{U}$, write $\boldsymbol{\mathcal{U}}$ for the event that $\mathcal{U}$ is the set of clusters popped by the algorithm before it terminates. Finally, let $\boldsymbol{\mathcal{U}} \wedge (\mathbf{V}, \mathbf{A}')$ be the event that the algorithm popped the clusters in $\mathcal{U}$ and terminated, with the output equal to (V, A').

Let $p[(V, A')]$ be the probability that after sampling a rowel from each stack $\mathcal{S}_v$, the resulting stack graph is equal to (V, A'). By the independence of stacks and rowels within a stack, we have

$$\mathbf{P}\big[\boldsymbol{\mathcal{U}} \wedge (\mathbf{V}, \mathbf{A}')\big] = \mathbf{P}\big[\boldsymbol{\mathcal{U}}\big] \cdot p[(V, A')].$$

But $p[(V, A')] = p[A']$ since rowels are sampled according to the distribution of equation (4.1). Therefore

$$\mathbf{P}[(\mathbf{V}, \mathbf{A}')] = \sum_{\mathcal{U}} \mathbf{P}\big[\boldsymbol{\mathcal{U}} \wedge (\mathbf{V}, \mathbf{A}')\big] = p[A'] \sum_{\mathcal{U}} \mathbf{P}\big[\boldsymbol{\mathcal{U}}\big],$$

which is proportional to $p[A']$ as desired. □

This completes the analysis of the cluster-popping algorithm. Observe that it is simple to recover the cycle-popping algorithm by changing the distribution according to which stack elements are sampled. Specifically, if the rowels in each stack are sampled not with the distribution in equation (4.1) but rather uniformly from the set of singleton rowels, we are back to Wilson's original formulation.

5.2. **Proof of the Main Theorem.** The proof of the Main Theorem (Theorem 2.2) follows from Theorem 5.3; we need only observe, based on the previously-noted equivalence between minimal clusters in a directed graph G and cycles in its derived hypergraph (cf. Section 2.3), that the cluster-popping algorithm on G is equivalent to the generalized Wilson algorithm on the derived hypergraph with

$$\Phi_v(B) = p^{|A^+(v)|-|B|}(1-p)^{|B|}$$

for every $v \in V, B \subseteq A^+(v)$.

This proves the Main Theorem, and implies the fact that the generalized Wilson algorithm on a directed graph samples root-connected subgraphs perfectly. There remains the question of running time, which is the topic of the Main Conjecture, discussed in Section 7.

6. Olio of examples and special cases

6.1. **Posets.** Let $G = (V, E)$ be an acyclic digraph strongly connected to the root $\boldsymbol{r}$. It defines a poset $(V, \prec)$ with the order given by the directions of the edges, and a unique maximal element $\widehat{1} = \boldsymbol{r}$. Consider the Directed Reachability Problem on G with probability p. In this case, the only clusters are singletons, and popping them all takes $O(1/p)$ steps.

A nice example of this is the case when $G = [k \times \ell]$ is the grid graph and one is trying to sample directed p-percolations from the origin which form a single cluster. Our cluster-popping algorithm finishes in $O(k\ell/p)$ steps.

6.2. **Small p.** Suppose in the Directed Reachability Problem, we have $p = o(1/n^3)$. Modify the cluster-popping algorithm as follows: run it by choosing exactly one outgoing edge from every vertex. Observe that this becomes the usual cycle-popping algorithm, which converges in $O(n^3)$ steps. Observe also that for p that small, w.h.p. every vertex will have exactly one outgoing edge, so this "modified algorithm" is nearly-perfect in total variation distance.

6.3. **Path graph.** Let $G = P_n$ be a bi-directed path on n vertices $1, \ldots, n$, arranged from left to right, with $1 = \boldsymbol{r}$. Consider a Directed Reachability Problem with probability $p > 0$. In this case sampling is very easy: take a path $n \to n-1 \to \ldots \to 1$ and add randomly the remaining $(i, i+1)$ edges, each with probability p.

Now, in the cluster-popping algorithm, the clusters are either single vertices or adjacent sequences of bi-directed pairs of edges. Take the leftmost singleton. By assumption, the vertex $i-1$ is root-connected. After i is popped, with probability p it is also root-connected. Similarly, take the leftmost pair $(i, i+1)$. Either $i-1$ is root-connected, or has an $(i-1, i)$ edge, or both. If $i-1$ is root-connected, as in the previous case, with probability p vertex i is now also root-connected. In the second case, if $(i-1, i)$ is present, then with probability p the leftmost pair of edges moves to the left and with probability $< p$ to the right. This means that the resulting random walk will hit the root-connected vertex after at most $O(n^2/p)$ steps, and thus the total cost of the cluster-popping algorithm is at most $O(n^3/p)$.

6.4. **Expander graph.** Let G be a $2d$-regular bi-directed expander on n vertices with root $\boldsymbol{r}$. A variation on the argument in [ABS] (see also [MP]) implies that for a constant p large enough, directed percolation clusters have constant size $\theta(n)$ with constant probability, including the component directed to $\boldsymbol{r}$. The clusters themselves are expanders, and after being popped they will have at least constant proportion sub-clusters reattached to other clusters. Using this, an easy argument similar to the complete graph gives $O(n \log n)$ running time for GLERW. We conjecture that in this case $O(n)$ steps suffice.

6.5. k**-trees.** Let $G = K_n$ be a complete bi-directed graph, and let $R = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_s\}$, be extra root vertices. A k-tree is a spanning hypertree where each edge has exactly $(k+1)$ vertices. For example, 1-tree is the usual directed spanning tree.

Consider now the generalized Wilson algorithm, in the cluster-popping form. Recall that root-connected hyperedges never get popped. At every step, every vertex in a cluster has $1-\binom{n-1}{k}/\binom{n-1+s+i}{k}$ probability of being connected to a directed hyperedge containing a root vertex $\boldsymbol{r}_j$, given that this component already has i vertices in addition to s roots. Summing over $i = 0 \ldots n-1$, we conclude that the total number of steps taken by the algorithm is at most $O(n \log n)$.

6.6. 1**-intersecting families hypergraph.** Let $L = (V, E)$ be a bipartite graph on vertices $V = X \cup Y$ with all vertices in Y of degree 2. Define a hypergraph $\mathcal{H}_L = (Y, A)$ where A is the set of hyperedges $(y \to y_1, \ldots, y_r)$ if $\{y, y_1, \ldots, y_r\}$ is the set of all neighbors of some $x \in X$. Note that if viewed as the usual (undirected) hypergraph, all hyperedges intersect at one vertex or not at all. We consider perfect distributions Φ_y on two hyperedges with tails at vertex y.

To analyze the generalized Wilson lgorithm on $\mathcal{H}_L$, consider a different graph $G_L = (X, W)$ where W is the set of edges (x, x') such that $(x, y), (x', y) \in E$ for some $y \in Y$. In this case, the hypertrees become acyclic orientations of edges of G_L, with sink vertices corresponding to the root-directed hyperedges. Already in this case, the cluster-popping algorithm is an interesting challenge, which becomes an absorbing Markov chain on general orientations (tournaments).

Here is an attractive special case of the above setting. Color unit squares in the grid $\mathbb{Z}^2$ in checkerboard fashion. Let $X \subset [n] \times [n]$ be any set of black squares on the square grid. Denote by $V \subset \mathbb{Z}^2$ the set of their vertices (with vertices connected to only one black square removed), and by $\mathcal{E}$ the set of hyperedges $(v \to x, y, z)$, where both x, y, z and v lie in the same black square in X. Fix one vertex to be the root $\boldsymbol{r}$. Consider the corresponding hypergraph $\mathcal{H} = (V, \mathcal{E})$ with these 4-edges. Each vertex is then the tail of one or two hyperedges, and we take Φ_v to be uniform for all $v \in V$. The spanning hyperedges in this case correspond to "orientations" of all black squares.

7. The Main Conjecture

7.1. **The conjecture.** There remains the question of the cluster-popping/generalized Wilson algorithm's running time. Recall that there is a polynomial bound on the expected running time of the cycle-popping algorithm on bi-directed graphs (Theorem 3.3), but running time can be exponential on general directed graphs. The cluster-popping algorithm can also take exponential time in expectation on general directed graphs; indeed, an example of such a graph is the graph in Section 3.3, which was the exponential-time example for cycle-popping.

The Main Conjecture states that just like cycle-popping, cluster-popping may be exponential in expectation on general directed graphs, but it is polynomial in expectation on bi-directed graphs.

Conjecture 7.1 (Main Conjecture)**.** *If $\{G = (V, A), \mathbf{r} \in V, 0 < p < 1\}$ is an instance of reachability and G is bi-directed, then the expected running time of the cluster-popping algorithm is polynomial in the size of G and $(1-p)^{-1}$.*

7.2. **Application to reachability.** A proof of the Main Conjecture leads to a randomized approximation algorithm for the reachability problem on bi-directed graphs, specifically a *fully polynomial randomized approximation scheme (FPRAS)*, which is the standard formalization of approximate counting.

Let us review the definition. Consider a counting problem, i.e. the problem of computing a function $f : \Sigma^* \to \mathbb{N}$ over some set of instances Σ^*. A FPRAS for this problem is a function $g_\epsilon : \Sigma^* \to \mathbb{N}$ such that given an input instance x and an error parameter $\epsilon > 0$, we have

$$\mathbf{P}\big[|f(x) - g_\epsilon(x)| \le \epsilon f(x)\big] \ge \frac{3}{4},$$

and moreover g_ϵ can be computed in time polynomial in ϵ^{-1} and the size of the instance $|x|$. Note that in the case of reachability the size of an instance is a function of both the size of the input graph G and the failure probability p.

Theorem 7.2. *The Main Conjecture implies a FPRAS for the reachability problem on bi-directed graphs.*

The proof of this theorem is standard and can be found in Section 9. It uses a canonical technique for leveraging a black-box sampling algorithm into one for approximate counting. This method is usually used in the more general setting of approximate rather than perfect sampling (see the book chapter [JS] by Jerrum and Sinclair).

8. Bi-directed reachability is #P-complete

In this section we will prove the following theorem.

Theorem 8.1. *Bi-directed reachability is #P-complete.*

The proof is by a two-step reduction from the following problem, which is shown in [BP2] to be #P-complete (see part 2 of that paper's main theorem).

Problem: Bipartite Independent Sets
Input: An undirected bipartite graph G.
Output: The number of independent sets in G.

We will reduce from this problem to bi-directed reachability using the following intermediate problem. Recall that in a directed graph $G = (V, A)$ with two terminals $s, t \in V$, an st-*cut* is a subset of A whose complement contains no directed paths from s to t.

Problem: Minimum Cardinality st-cuts in Bi-directed Graphs
Input: A bi-directed graph $G = (V, A)$ and $s, t \in V$.
Output: The number of minimum cardinality st-cuts in G.

Lemma 8.2. *Counting minimum cardinality* st-*cuts in bi-directed graphs is #P-complete.*

Before we proceed to the proof of Theorem 8.1 and Lemma 8.2, we note that in [BP2], Ball and Provan prove that reachablity is #P-complete in general directed graphs as well as undirected graphs. In the latter case, Ball and Provan first reduce from undirected reachability to counting minimum st-cuts in undirected graphs, and then from this latter problem to counting independent sets in bipartite graphs. Our proof of Theorem 8.1 is a straightforward modification of their technique: we reduce from bi-directed reachability to counting minimum st-cuts in bi-directed graphs, and then to counting independent sets in bipartite graphs. Adapting the reductions in [BP2] to the bi-directed setting, as we do below, requires only minor modifications.

Proof of Theorem 8.1. By Lemma 8.2, it suffices to reduce to the problem of counting minimum st-cuts in a bi-directed graph. We closely follow the proof of the analogous result in [BP2] (part 8B of the main theorem).

To begin we make the following observation. Let $\{G = (V, A), \boldsymbol{r} \in V, 0 < p < 1\}$ be an instance of the bi-directed reachability problem, where A consists of m bi-directed edges. The probability that the reachability problem asks us to compute can be written as the polynomial

$$f(G, r, p) = \sum_{i=0}^{2m} \big|\mathcal{A}_i(G, r)\big| p^i (1-p)^{2m-i} = (1-p)^{2m} \sum_{i=0}^{2m} \big|\mathcal{A}_i(G, r)\big| \left(\frac{p}{1-p}\right)^i, \tag{8.1}$$

where $\mathcal{A}_i(G, r)$ is the set of $B \subseteq A$ with $|B| = i$ whose complement defines a root-connected subgraph. We call (8.1) the *reachability polynomial*.

Fix a bi-directed graph $G = (V, A)$ with terminals $s, t \in V$ in which we want to count minimum st-cuts. Let m be the number of bi-directed edges in G, so that $|A| = 2m$. For every fixed p, $0 < p < 1$, we can regard G with root t as an instance of bi-directed reachability with reachability polynomial

$$f(G, t, p) = (1-p)^{2m} \sum_{i=0}^{2m} \big|\mathcal{A}_i(G, t)\big| \left(\frac{p}{1-p}\right)^i. \tag{8.2}$$

Now construct G' by identifying s and t; call this merged vertex v_{st}. With v_{st} as root and the same p as above, we have another instance of bi-directed reachability with reachability polynomial

$$f(G', v_{st}, p) = (1-p)^{2m} \sum_{i=0}^{2m} \big|\mathcal{A}_i(G', v_{st})\big| \left(\frac{p}{1-p}\right)^i. \tag{8.3}$$

Now, $\mathcal{A}_i(G, t)$ consists of edge sets in G of size i whose complement admits a path from every vertex to t, and $\mathcal{A}_i(G', v_{st})$ consists of edge sets of size i whose complement admits a path from every vertex to *either* s or t. Therefore, $\mathcal{A}_i(G', v_{st}) \setminus \mathcal{A}_i(G, t)$ is the set of edge sets of size i whose complement admits a path from every vertex to s or t, but not every vertex has both a path to s *and* a path to t. In particular, any such edge set must contain an st-cut.

Let k be the size of a minimum st-cut in G (recall that k can be efficiently computed with a network-flow algorithm). Observe that $\mathcal{A}_k(G', v_{st}) \setminus \mathcal{A}_k(G, t)$ is the set of minimum st-cuts in G, and therefore the number of such cuts is $|\mathcal{A}_k(G', v_{st})| - |\mathcal{A}_k(G, t)|$.

It follows from the above that computing the coefficients of $f(G, t, p)$ and $f(G', v_{st}, p)$ will yield the number of minimum st-cuts in G. The coefficients of $f(G, t, p)$ can be computed by sampling the value of this polynomial, i.e. solving the reachability problem, for $2m + 1$ distinct values of p, and the same holds for $f(G', v_{st}, p)$. This completes the reduction. □

Proof of Lemma 8.2. The reduction is from counting bipartite independent sets, and closely follows part 6 of the the main theorem in [BP2]. Fix an undirected bipartite graph $G = (V, E)$ with $V = V_0 \cup V_1$ that is an instance of this problem, and let $\deg_G(v)$ be the degree of $v \in V$ in G. Create a bi-directed graph G' by

(1) making all undirected edges of G bi-directed,
(2) adding two terminals s and t,
(3) for each $v \in V_0$, adding a set of $\deg_G(v)$ bi-directed edges between s and v,
(4) for each $w \in V_1$, adding a set of $\deg_G(w)$ bi-directed edges between w and t.

This is illustrated in Figure 3.

For $v \in V_0$ we write A_v for the set of directed edges from s to v; there are $\deg_G(v)$ of them. Similarly for $w \in V_1$ we write A_w for the set of directed edges from w to t; there are $\deg_G(w)$.

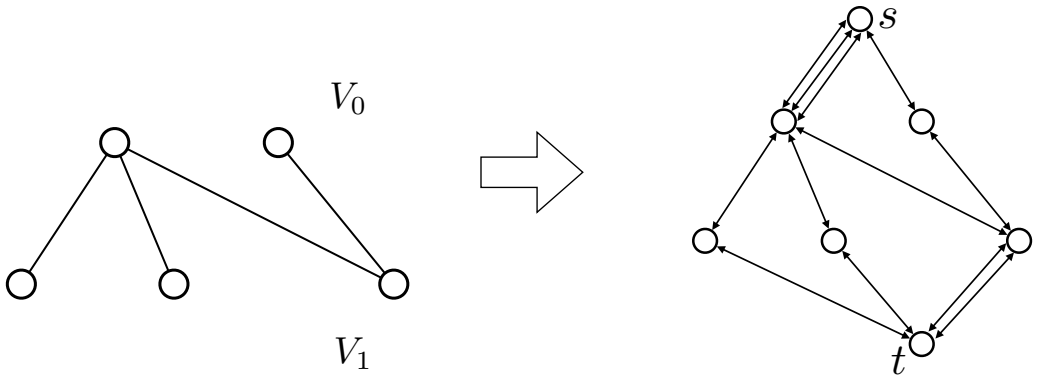

FIGURE 3. Turning bipartite G on the left (with vertex sets V_0, V_1) to bi-directed G', with added terminals s and t, on the right.

Observe that if a minimum st-cut in G' contains an edge in some A_v for $v \in V_0$ (A_w for $w \in V_1$), then it must contain all of A_v (A_w).

Next, observe that $|E|$ is the size of a minimum st-cut in G', because the cut consisting of all (directed) edges from V_0 to V_1 has this size. Moreover an st-flow of this size can be obtained by passing a flow of 1 along every edge from s to V_0 to V_1 to t.

Now, if C' is a minimum st-cut and $A_v, A_w \subseteq C'$ for $v \in V_0, w \in V_1$, then there could not have been an edge between v and w in G. Indeed, otherwise there is a bi-directed edge between them in G' and we can obtain a smaller cut by removing A_v and A_w, and adding all directed edges leaving v and all directed edges entering w (see Figure 4).

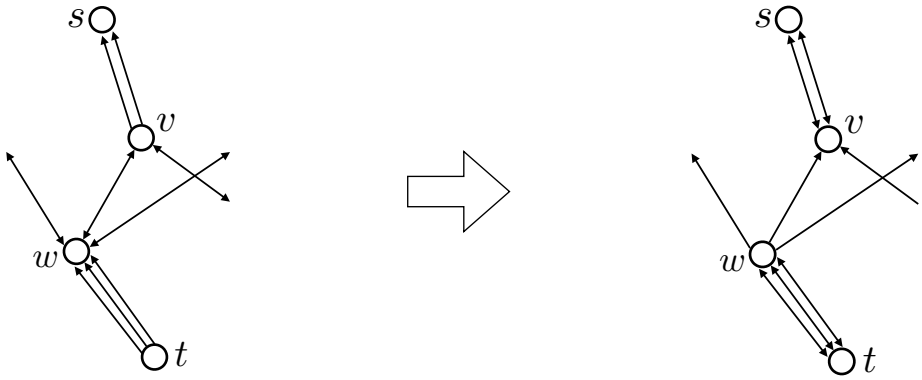

FIGURE 4. At left is the complement of an st-cut. If $v \leftrightarrow w$ in G', we can transform this cut into a smaller cut whose complement is shown at right.

Consider now

$$U_0 = \{v \in V_0 \mid A_v \subseteq C'\}, \; U_1 = \{w \in V_1 \mid A_w \subseteq C'\}.$$

It follows from the above that $U_0 \cup U_1$ is an independent set in G. Conversely, if $U_0 \cup U_1$ is an independent set in G with $U_0 \subseteq V_0$ and $U_1 \subseteq V_1$, then in G', the edge set

$$\left(\bigcup_{v \in U_0} A_v\right) \bigcup \left(\bigcup_{w \in U_1} A_w\right) \bigcup \Big\{v \to w \mid v \in V_0 \setminus U_0, w \in V_1 \setminus U_1\Big\}$$

is an st-cut of size $|E|$, hence a minimum st-cut. The establishes a bijection between independent sets in G and minimum st-cuts in G', proving the lemma. □

9. Proof of Theorem 7.2

Proof. Let $\{G = (V, A), \boldsymbol{r} \in V, 0 < p < 1\}$ be an instance of reachability with G bi-directed. Define a sequence of bi-directed graphs $G_0, G_1, \ldots, G_{n-1}$ as follows. Set $G_0 = G$. Given G_{i-1} with $i \geq 1$, define G_i by choosing a pair of vertices u_i and v_i in G_{i-1} with at least one bi-directed edge between them, removing all such edges, and identifying u_i and v_i. If one of u_i and v_i is $\boldsymbol{r}$, then label the new merged vertex $\boldsymbol{r}$. Note that $G_0 = G$ and $G_{n-1} = (\{\boldsymbol{r}\}, \varnothing)$. We will write V_i for the vertex set of G_i.

Consider every G_i to be an instance of the reachability problem with root $\boldsymbol{r}$ and failure probability p. Let us write $p[G_i]$ for the probability that reachability is preserved in G_i after edges are allowed to fail independently. The quantity that we wish to estimate is $p[G] = p[G_0]$, and we can write it as

$$p[G] = \frac{p[G_0]}{p[G_1]} \cdot \frac{p[G_1]}{p[G_2]} \cdots \frac{p[G_{n-2}]}{p[G_{n-1}]} \cdot p[G_{n-1}].$$

Observe that $p[G_{i-1}] \leq p[G_i]$ by construction and $p[G_{n-1}] = 1$. Moreover we claim that

$$(1-p)^2 p[G_i] \leq p[G_{i-1}].$$

To see why, let u_i, v_i be the vertices identified in going from G_{i-1} to G_i and let $u_i \leftrightarrow v_i$ be a bi-directed edge between them. We have

$$\begin{aligned}
p[G_{i-1}] &= \mathbf{P}\big[\text{reachability preserved in } G_{i-1}\big] \\
&\geq \mathbf{P}\big[\text{reachability preserved in } G_{i-1} \text{ and } u_i \leftrightarrow v_i \text{ survives}\big] \\
&\geq (1-p)^2\, \mathbf{P}\big[\text{reachability preserved in } G_{i-1} \mid u_i \leftrightarrow v_i \text{ survives}\big] \\
&\geq (1-p)^2\, \mathbf{P}\big[\text{reachability preserved in } G_i\big] \\
&\geq (1-p)^2 p[G_i].
\end{aligned}$$

By the above, if we define $\rho_i = p[G_{i-1}]/p[G_i]$ then we have

$$(1-p)^2 \leq \rho_i \leq 1 \tag{9.1}$$

and of course

$$p[G] = \rho_1 \rho_2 \ldots \rho_{n-1}. \tag{9.2}$$

We are now ready to give the details of our FPRAS, so let us fix some $\epsilon > 0$ (we will assume $\epsilon \leq 1$ and $n > 1$ to avoid trivialities). We want a random variable $\overline{\zeta}$ such that $\big|p[G] - \overline{\zeta}\big| \leq \epsilon p[G]$ with probability $3/4$. To this end, define Z_i to be the indicator random variable obtained by perfectly sampling a root-connected subgraph (V_i, A_i') in G_i, and setting

$$Z_i = \begin{cases} 1 \text{ if } (V_{i-1}, A_i') \text{ is root-connected in } G_{i-1}, \\ 0 \text{ otherwise.} \end{cases}$$

Because (V_i, A_i') is perfectly sampled, the probability of $Z_i = 1$ is $p[G_{i-1}]/p[G_i]$, i.e. $\mathbf{E}[Z_i] = \rho_i$.

Since Z_i is an indicator variable we have $\operatorname{Var} Z_i = \rho_i(1-\rho_i)$, which combined with inequality (9.1) implies

$$\frac{\operatorname{Var} Z_i}{\rho_i^2} \leq \frac{1}{\rho_i} - 1 \leq (1-p)^{-2}. \tag{9.3}$$

Now let $Z_i^{(1)}, \ldots, Z_i^{(s)}$ be $s = \lceil 5(1-p)^{-2}(n-1)/\epsilon^2 \rceil$ independent random samples of Z_i and set $\zeta_i = s^{-1} \sum_{j=1}^{s} Z_i^{(j)}$. We clearly have $\mathbf{E}(\zeta_i) = \rho_i$. Moreover, inequality (9.3) implies

$$\frac{\operatorname{Var} \zeta_i}{\rho_i^2} \leq (1-p)^{-2} s^{-1}. \tag{9.4}$$

We define our estimator for $p[G]$ to be

$$\overline{\zeta} = \zeta_1 \zeta_2 \dots \zeta_{n-1}.$$

If we set $\mu = \mathbf{E}(\overline{\zeta})$ then

$$\mu = \rho_1 \rho_2 \dots \rho_{n-1} = p[G]$$

by equation (9.2), so to prove that sampling $\overline{\zeta}$ is actually an FPRAS for $p[G]$ it only remains to bound the deviation of $\overline{\zeta}$ from its mean $\mu = p[G]$.

To this end, we observe that

$$\begin{aligned}
\frac{\operatorname{Var}\overline{\zeta}}{\mu^2} &= \frac{\operatorname{Var}\big[\zeta_1 \zeta_2 \dots \zeta_{n-1}\big]}{(\rho_1 \rho_2 \dots \rho_{n-1})^2} \\
&= \prod_{i=1}^{n-1} \left(1 + \frac{\operatorname{Var}\zeta_i}{\rho_i^2}\right) - 1 \\
&\le \left(1 + \frac{(1-p)^{-2}}{s}\right)^{n-1} - 1 \\
&\le e^{\epsilon^2/5} - 1 \\
&\le \frac{1}{4}\epsilon^2,
\end{aligned}$$

where the first inequality follows from inequality (9.4) and last inequality follows since $e^{x/5} \le 1 + x/4$ when $0 \le x \le 1$. It now follows from Chebyshev's inequality that

$$\mathbf{P}\Big[\big|p[G] - \overline{\zeta}\big| \ge \epsilon p[G]\Big] \le \frac{1}{4}.$$

This proves that our estimator $\overline{\zeta}$ is a sufficiently good approximation.

With regard to the running time, it is easy to see that it is dominated by the time required for sampling of root-connected subgraphs. The number of samples to be performed is polynomial in the size of G, $(1-p)^{-1}$, and ϵ^{-1}. If the Main Conjecture is true, then the cluster-popping algorithm can be used to perfectly sampled root-connected subgraphs in expected polynomial time. Running this algorithm and halting after a polynomial number of iterations yields a polynomial time Monte Carlo sampling algorithm with arbitrarily small error probability. Thus, the Main Conjecture implies a FPRAS for bi-directed reachability. □

10. Final remarks

10.1. Wilson in [Wil] summarizes the cycle-popping algorithm thusly: "The stacks uniquely define a tree together with a partially ordered set of cycles layered on top of it. The algorithm peels off these cycles to find the tree." Similarly, the cluster-popping algorithm peels off minimal clusters to find a root-connected subgraph.

10.2. Let us quickly survey different notions of hypertrees. First, there are inductive combinatorial definitions [BeiP, Dew] (see also [RR]). Second, there are generic combinatorial definitions coming from various notions of a path and cycle (weak, strong, in between), which are studied in hypergraph theory [Ber]. Then there are several topological definitions [Kal, LMR, MV] (see also [DKM]), based on different notions of *acyclic simplicial complexes*. Bases of matroids give yet another definition, coming fundamentally from linear algebra [Oxl]. One final notion is given in [Far], coming from commutative algebra associated with simplicial polytopes.

Now, when it comes to hypertrees in *directed* hypergraphs, only some of these notions can be extended. A general setup is always the same and coincides with our notion of a directed hypergraph [Aus], but beyond this point different authors considered different notions, studied in only a handful of papers. As far as we can tell, our definition seems to be new.

10.3. We should mention that *directed branching greedoids* give a natural extension of graphical matroids [KLS]. In case of a directed graph G, the bases of the corresponding branching greedoid are directed rooted trees. These are interval greedoids with a number of useful properties. Our hypertrees can also be stated in this setting, as the maximal independent sets of the directed hypergraph branching greedoid. Curiously, the root-connected subgraphs in the reachability problem can also be interpreted in this setting as spanning subsets of a directed branching greedoid, i.e. subsets which contain a basis (directed tree). We leave the details to the reader.

10.4. It is important to separate the mathematical and computational advancements in the random tree generation. From the computational points of view, there is a straightforward polynomial time algorithm which generates random (directed, rooted) spanning trees in a (directed) graph using the matrix-tree theorem [Gué]. This is in sharp contrast with probabilistic algorithms, including Wilson's algorithm, which require roughly the cover time [Ald, Bro, Wil]. As the example in Figure 1 in Subsection 3.4 shows, both the cover time and the running time of these algorithms is exponential.

Moving away from perfect sampling (see below), one can ask whether the usual MCMC approach allows polynomial time random sampling of spanning trees. In the undirected case this follows from the *Rayleigh property* of graphs [FM]. However, for general matroids, the corresponding *base exchange random walk* is not known to be rapidly mixing. This is conjectured to be true in even much greater generality of graphs of 0-1 polytopes with polynomial degree [Mih], but only very few special cases have been established (see [Jer]).

10.5. Although perfect sampling has no advantage over nearly-perfect sampling for approximate counting applications, having perfect sampling is rare and interesting in its own right. In addition, when the problem has a nice symmetric structure, the perfect sampling algorithm can have useful combinatorial applications; see e.g. [Ald, AP] which analyze two algorithm producing random spanning trees in a complete graph.

The general *coupling from the past* technique allows one to give perfect sampling from *any* distribution if running time is not an issue [PW]. This approach has other technical issues, including the "lifetime commitment", which is now largely resolved. It is also known to work efficiently in a few nice combinatorial special cases [Pro].

There are other, more direct perfect sampling algorithms, which are especially rare and impressive in the case of #P-complete problems. These include sampling solutions to DNF formulae [JVV] (see also [KL]), certain colorings of bounded-degree graphs [Hub], sink-free orientations of undirected graphs [Hub, CPP], and a few others.

Recall that counting reachability-preserving subgraphs of bi-directed graphs is #P-complete (Theorem 8.1). Now the Main Conjecture implies that perfect sampling can be done in expected polynomial time, which would add the reachability to the list above. This also suggests that the conjecture is of theoretical interest beyond its application to approximating reachability.

10.6. A quick comparison of our Directed Reachability Problem and Karger's classical result in [Kar]. First, Karger analyzes an *undirected* rather than *directed* graph reliability; the undirected version cannot be even stated in our general directed hypergraph setting. Second, Karger analyzes all-terminal reliability, which in the directed setting roughly corresponds to strong connectivity, not root-connectivity. Finally, while Karger's algorithm approximates the probability of disconnectedness (or "failure"), our algorithm approximates the probability of root-connectivity (or "success"). In the

regime where these probabilities are polynomially bounded regimes, these notions are interrelated, of course. However, on the margins $p \twoheadrightarrow 0$ and $p \twoheadrightarrow 1$, the probabilities become superpolynomially small/large, in which case these problems are incomparable.

Now, let us mention that despite superficial similarities, the *bi-directed* reachability problem is different from the undirected all-terminal reliability studied by Karger. In fact, the former depends on the root assignment, and differs in the most simple examples. For a complete graph K_3 with one vertex designated as a root $\boldsymbol{r}$, the two probabilities are

$$P_{\text{bi-directed}} \,=\, 3p^2 + p^3\,, \qquad P_{\text{all-terminal}} \,=\, 3p^2 - 2p^3\,.$$

10.7. The generalized LERW we defined in Section 4 is no longer Markovian, but a mixture of interactive particle process and a branching process. See [Kol] for a related example of this connection and [Lig] for the background on interactive particle systems.

Acknowledgements. We are grateful to Carly Klivans and Jeremy Martin for helpful conversations on topological hypertrees, and to Marek Biskup, Mark Huber, Laurent Saloff-Coste, Alistair Sinclair, and Prasad Tetali for probabilistic remarks and encouragement at various stages of this project.

This work was initiated and completed during a special program at the Institute for Pure and Applied Mathematics (IPAM) at UCLA; we are thankful to IPAM and the program organizers for their hospitality. The first author was supported by IPAM and an NSF Graduate Research Fellowship. The second author is partially supported by the NSF and BSF grants.

References

[AP] J. Alappattu and J. Pitman, Coloured loop-erased random walk on the complete graph, *Combin. Probab. Comput.* **17** (2008), 727–740.

[Ald] D. J. Aldous, The random walk construction of uniform spanning trees and uniform labelled trees, *SIAM J. Discrete Math.* **3** (1990), 450–465.

[ABS] N. Alon, I. Benjamini and A. Stacey, Percolation on finite graphs and isoperimetric inequalities, *Ann. Probab.* **32** (2004), 1727–1745.

[Aus] G. Ausiello, Directed hypergraphs: data structures and applications, *Lecture Notes in Comput. Sci.* **299**, Springer, Berlin, 1988.

[BP1] M. O. Ball and J. S. Provan, Calculating bounds on reachability and connectedness in stochastic networks, *Networks* **13** (1983), no. 2, 253–278.

[BP2] M. O. Ball and J. S. Provan, The complexity of counting cuts and of computing the probability that a graph is connected, *SIAM J. Comput.* **12** (1983), 777–788.

[BeiP] L. W. Beineke and R. E. Pippert, Properties and characterizations of k-trees, *Mathematika* **18** (1971), 141–151.

[Ber] C. Berge, *Hypergraphs. Combinatorics of finite sets*, North-Holland, Amsterdam, 1989.

[Bro] A. Z. Broder, Generating Random Spanning Trees, in *Proc. 30th FOCS*, IEEE, 1989, 442–447.

[CPP] H. Cohn, R. Pemantle and J. G. Propp, Generating a random sink-free orientation in quadratic time, *Electron. J. Combin.* **9** (2002), no. 1, RP 10, 13 pp.

[Col] C. J. Colbourn, *The combinatorics of network reliability*, Oxford University Press, New York, 1987.

[Dew] A. K. Dewdney, Higher-dimensional tree structures, *J. Comb. Theory, Ser. B* **17** (1974), 160-169.

[DKM] A. M. Duval, C. J. Klivans and J. L. Martin, Simplicial matrix-tree theorems, *Trans. AMS* **361** (2009), 6073–6114.

[Far] S. Faridi, The facet ideal of a simplicial complex, *Manuscripta Math.* **109** (2002), 159–174.

[FM] T. Feder and M. Mihail, Balanced matroids, in *Proc. 24th STOC*, ACM, 1992, 26–38.

[Gué] A. Guénoche, Random spanning tree (in French), *J. Algorithms* **4** (1983), 214–220.

[Hub] M. L. Huber, Exact sampling and approximate counting techniques, in *Proc. 30th STOC*, ACM, 1999, 31–40.

[Jer] M. R. Jerrum, Two remarks concerning balanced matroids, *Combinatorica* **26** (2006), 733–742.

[JS] M. R. Jerrum and A. Sinclair, The Markov Chain Monte Carlo method: an approach to approximate counting and integration, in *Approximation Algorithms for NP-hard Problems*, PWS Publishing, Boston, 1996.

[JVV] M. R. Jerrum, L. G. Valiant and V. V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.* **43** (1986), 169–188.

[Kal] G. Kalai, Enumeration of $\mathbb{Q}$-acyclic simplicial complexes, *Israel J. Math.* **45** (1983), 337–351.

[Kar] D. R. Karger, A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem, *SIAM J. Comput.* **29** (1999), 492–514.

[KL] R. M. Karp and M. G. Luby, Monte Carlo algorithms for enumeration and reliability problems, *J. Complexity* **1** (1985), 45–64.

[KLM] R. M. Karp, M. G. Luby and N. Madras, Monte Carlo approximation algorithms for enumeration problems, *J. Algorithms* **10** (1989), 429–448.

[Kol] V. F. Kolchin, Branching processes and random hypertrees, *Discrete Math. Appl.* **9** (1999), 7–23.

[KLS] B. Korte, L. Lovász and R. Schrader, *Greedoids*, Springer, Berlin, 1991.

[Law] G. F. Lawler, Loop-erased random walk, in *Perplexing problems in probability*, Birkhäuser, Boston, MA, 1999, 197–217.

[LSW] G. F. Lawler, O. Schramm and W. Werner, Conformal invariance of planar loop-erased random walks and uniform spanning trees, *Ann. Probab.* **32** (2004), 939–995.

[Lig] T. M. Liggett, *Interacting particle systems* (Reprint of the 1985 original), Springer, Berlin, 2005.

[LMR] N. Linial, R. Meshulam and M. Rosenthal, Sum complexes—a new family of hypertrees, *Discrete Comput. Geom.* **44** (2010), 622–636.

[MP] C. Malon and I. Pak, Percolation of finite Cayley graphs, *Combin. Probab. Comput.* **15** (2006), 571–588.

[Mar] P. Marchal, Loop-erased random walks, spanning trees and Hamiltonian cycles, *Electron. Comm. Probab.* **5** (2000), 39–50.

[MV] G. Masbaum and A. Vaintrob, A new matrix-tree theorem, *IMRN* **2002**, no. 27, 1397–1426.

[Mih] M. Mihail, On the expansion of combinatorial polytopes, in *Lecture Notes in Comput. Sci.* **629**, Springer, Berlin, 1992, 37–49.

[MR] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[Oxl] J. G. Oxley, *Matroid theory*, Oxford University Press, New York, 1992.

[Pro] J. Propp, Generating random elements of finite distributive lattices, *Electron. J. Combin.* **4** (1997), no. 2, RP 15, 12 pp.

[PW] J. G. Propp and D. B. Wilson, Exact sampling with coupled Markov chains and applications to statistical mechanics, *Random Structures and Algorithms* **9** (1996), 223–252.

[RR] C. Rényi and A. Rényi, The Prüfer code for k-trees, in *Combinatorial theory and its applications, III*, North-Holland, Amsterdam, 1970, 945–971.

[Val] L. G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8** (1979), 410–421.

[Wel] D. J. A. Welsh, *Complexity: knots, colourings and counting*, Cambridge University Press, Cambridge, 1993.

[Wil] D. B. Wilson, Generating random spanning trees more quickly than the cover time, in *Proc. 28th STOC*, ACM, 1996, 296–303.