A SIMPLE FPRAS FOR BI-DIRECTED REACHABILITY

Heng Guo (University of Edinburgh)
Joint with Mark Jerrum (Queen Mary, University of London)

PKU TCS seminar, Dec 27 2017
Counting
The complexity of computing quantities

Complexity class $\mathbb{#P}$ by Valiant (1979):

- a counting analogue of $\text{NP}$.

- Evaluation of probabilities;
- Multivariate integration;
- Counting discrete structures ...
Network Reliability

**Reliability**: in a graph (or network) $G = (V, E)$, suppose each edge fails with probability $p$. What’s the probability that the remaining graph is connected?

In other words, we want to compute

$$Z_{rel}(G, p) := \sum_{R \subseteq E: (V, R) \text{ is connected}} p^{|E \setminus R|} (1 - p)^{|R|}.$$
The unweighted case (namely, $p = 0.5$) is among the original 17 \texttt{#P}-complete problem in [Valiant '79].

Exact evaluation is \texttt{#P}-complete [Jerrum '81] [Provan, Ball '83].

Karger (1999) gave an FPRAS for unreliability, but the complexity of approximating reliability is still open.
The unweighted case (namely, $p = 0.5$) is among the original 17 \#P-complete problem in [Valiant ’79].

Exact evaluation is \#P-complete [Jerrum ’81] [Provan, Ball ’83].

Karger (1999) gave an FPRAS for unreliability, but the complexity of approximating reliability is still open.
The unweighted case (namely, $p = 0.5$) is among the original 17 #P-complete problem in [Valiant ’79].

Exact evaluation is #P-complete [Jerrum ’81] [Provan, Ball ’83].

Karger (1999) gave an FPRAS for unreliability, but the complexity of approximating reliability is still open.
We say a directed graph $G$ with root $r$ is \textit{root-connected} if all vertices can reach $r$.

Ball and Provan (1983) defined \textbf{REACHABILITY}: in a directed graph with root $r$, suppose each arc fails with probability $p$, what’s the probability that the remaining graph is root-connected?

$$Z_{\text{reach}}(G, p) := \sum_{R \subseteq E : (V, R) \text{ is root-connected}} p^{|E \setminus R|} (1 - p)^{|R|}.$$
We say a directed graph $G$ with root $r$ is *root-connected* if all vertices can reach $r$.

Ball and Provan (1983) defined **reachability**: in a directed graph with root $r$, suppose each arc fails with probability $p$, what’s the probability that the remaining graph is root-connected?

$$Z_{\text{reach}}(G, p) := \sum_{R \subseteq E : (V, R) \text{ is root-connected}} p^{|E \setminus R|} (1 - p)^{|R|}.$$
Cluster popping

If \( G \) is bi-directed, approximating \( Z_{\text{reach}}(G, p) \) can be reduced to sampling root-connected subgraphs [Gorodezky, Pak 14].

Cluster: no edge going out.

Cluster popping [Gorodezky, Pak 14]: randomize edges and repeatedly pop minimal clusters.

[G., Jerrum 17]: the expected number of rounds in a bi-directed graph is \( O\left(\frac{mn}{1-p}\right) \).
If $G$ is bi-directed, approximating $Z_{\text{reach}}(G, p)$ can be reduced to sampling root-connected subgraphs [Gorodezky, Pak 14].

Cluster: no edge going out.

Cluster popping [Gorodezky, Pak 14]: randomize edges and repeatedly pop minimal clusters.

[G., Jerrum 17]: the expected number of rounds in a bi-directed graph is $O\left(\frac{mn}{1-p}\right)$. 
If $G$ is bi-directed, approximating $Z_{\text{reach}}(G, p)$ can be reduced to sampling root-connected subgraphs [Gorodezky, Pak 14].

Cluster: no edge going out.

Cluster popping [Gorodezky, Pak 14]: randomize edges and repeatedly pop minimal clusters.

[G., Jerrum 17]: the expected number of rounds in a bi-directed graph is $O \left( \frac{mn}{1-p} \right)$. 
If $G$ is bi-directed, approximating $Z_{\text{reach}}(G, p)$ can be reduced to sampling root-connected subgraphs [Gorodezky, Pak 14].

Cluster: no edge going out.

Cluster popping [Gorodezky, Pak 14]: randomize edges and repeatedly pop minimal clusters.

[G., Jerrum 17]: the expected number of rounds in a bi-directed graph is $O\left(\frac{mn}{1-p}\right)$. 
If $G$ is bi-directed, approximating $Z_{\text{reach}}(G, p)$ can be reduced to sampling root-connected subgraphs [Gorodezky, Pak 14].

Cluster: no edge going out.

Cluster popping [Gorodezky, Pak 14]: randomize edges and repeatedly pop minimal clusters.

[G., Jerrum 17]: the expected number of rounds in a bi-directed graph is $O\left(\frac{mn}{1-p}\right)$. 
PARTIAL REJECTION SAMPLING

(WHY IS CLUSTER-POPPING CORRECT AND EFFICIENT?)
A random walk SAT-solver

The prototypical NP-complete problem: given a CNF formula, does it have a satisfying assignment?

\[(x_1 \lor \overline{x}_3 \lor x_5) \land (x_2 \lor x_3) \land (\overline{x}_3 \lor x_4) \land (x_1 \lor \overline{x}_5 \lor x_6 \lor x_7) \ldots\]

Rejection sampling: assign each variable uniformly at random and independently. If not satisfying, reject and repeat.

Walk-SAT: while there is a violated clause, re-randomize all its variables.

It is optimal in a very general setting! [Moser, Tardos 10]
The prototypical **NP-complete** problem:
given a CNF formula, does it have a satisfying assignment?

\[(x_1 \lor x_3 \lor x_5) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_1 \lor x_5 \lor x_6 \lor x_7) \ldots\]

**Rejection sampling**: assign each variable uniformly at random and independently. If not satisfying, reject and repeat.

**Walk-SAT**: while there is a violated clause, re-randomize all its variables.

It is optimal in a very general setting! [Moser, Tardos 10]
The prototypical **NP-complete** problem: given a CNF formula, does it have a satisfying assignment?

\[(x_1 \lor \overline{x_3} \lor x_5) \land (x_2 \lor x_3) \land (\overline{x_3} \lor \overline{x_4}) \land (x_1 \lor \overline{x_5} \lor x_6 \lor x_7) \ldots\]

*Rejection sampling*: assign each variable uniformly at random and independently. If not satisfying, reject and repeat.

Walk-SAT: while there is a violated clause, re-randomize all its variables.

It is optimal in a very general setting! [Moser, Tardos 10]
The prototypical NP-complete problem: given a CNF formula, does it have a satisfying assignment?

\[(x_1 \lor \overline{x}_3 \lor x_5) \land (x_2 \lor x_3) \land (\overline{x}_3 \lor \overline{x}_4) \land (x_1 \lor \overline{x}_5 \lor x_6 \lor x_7) \ldots\]

*Rejection sampling*: assign each variable uniformly at random and independently. If not satisfying, reject and repeat.

Walk-SAT: while there is a violated clause, re-randomize all its variables.

It is optimal in a very general setting! [Moser, Tardos 10]
Find a “perfect” assignment of the variables avoiding all “bad” events.

Variables $X_1, \ldots, X_n$ \hspace{1cm} “Bad” events $A_1, \ldots, A_m$

Dependency graph: $A_i$ and $A_j$ are adjacent if $\text{var}(A_i) \cap \text{var}(A_j) \neq \emptyset$.

Erdős and Lovász (1975): $4p\Delta \leq 1 \Rightarrow$ existence of a perfect assignment.

$p$: max probability of $A_i$ \hspace{1cm} $\Delta$: max degree of the dependency graph

Lovász (1977) improved the condition to $ep(\Delta + 1) \leq 1$.

Shearer (1985) gave the optimal condition of LLL.

LLL only guarantees an exponentially small probability.
Find a “perfect” assignment of the variables avoiding all “bad” events.

Variables $X_1, \ldots, X_n$  “Bad” events $A_1, \ldots, A_m$

Dependency graph: $A_i$ and $A_j$ are adjacent if $\var(A_i) \cap \var(A_j) \neq \emptyset$.

Erdős and Lovász (1975): $4p\Delta \leq 1 \Rightarrow$ existence of a perfect assignment.

$p$: max probability of $A_i$  $\Delta$: max degree of the dependency graph

Lovász (1977) improved the condition to $ep(\Delta + 1) \leq 1$.

Shearer (1985) gave the optimal condition of LLL.

LLL only guarantees an exponentially small probability.
Find a “perfect” assignment of the variables avoiding all “bad” events.

Variables $X_1, \ldots, X_n$  
“Bad” events $A_1, \ldots, A_m$

Dependency graph: $A_i$ and $A_j$ are adjacent if $\text{var}(A_i) \cap \text{var}(A_j) \neq \emptyset$.

Erdős and Lovász (1975): $4p\Delta \leq 1 \Rightarrow$ existence of a perfect assignment.

$p$: max probability of $A_i$  
$\Delta$: max degree of the dependency graph

Lovász (1977) improved the condition to $ep(\Delta + 1) \leq 1$.

Shearer (1985) gave the optimal condition of LLL.

LLL only guarantees an exponentially small probability.
Variable framework of the Lovász Local Lemma

Find a “perfect” assignment of the variables avoiding all “bad” events.

Variables $X_1, \ldots, X_n$  
“Bad” events $A_1, \ldots, A_m$

Dependency graph: $A_i$ and $A_j$ are adjacent if $\text{var}(A_i) \cap \text{var}(A_j) \neq \emptyset$.

Erdős and Lovász (1975): $4p\Delta \leq 1 \Rightarrow$ existence of a perfect assignment.  
$p$: max probability of $A_i$  
$\Delta$: max degree of the dependency graph

Lovász (1977) improved the condition to $ep(\Delta + 1) \leq 1$.  
Shearer (1985) gave the optimal condition of LLL.

LLL only guarantees an exponentially small probability.
Find a “perfect” assignment of the variables avoiding all “bad” events.

Variables $X_1, \ldots, X_n$  "Bad" events $A_1, \ldots, A_m$

Dependency graph: $A_i$ and $A_j$ are adjacent if $\text{var}(A_i) \cap \text{var}(A_j) \neq \emptyset$.

Erdős and Lovász (1975): $4p\Delta \leq 1 \Rightarrow$ existence of a perfect assignment.

$p$: max probability of $A_i$  \hspace{1cm} \Delta$: max degree of the dependency graph

Lovász (1977) improved the condition to $ep(\Delta + 1) \leq 1$.

Shearer (1985) gave the optimal condition of LLL.

LLL only guarantees an exponentially small probability.
Beck (1991) showed that an algorithmic version is possible, starting a long line of research.

Moser and Tardos (2010) found a very elegant algorithm:

1. Initialize all variables randomly.

2. While there exists an occurring bad event:
   pick one (various rules) and resample all its variables.

Many developments since then:

[Haeupler, Saha, Srinivasan 11], [Kolipaka, Szegedy 11], [Harris, Srinivasan 13],
[Achlioptas, Iliopoulos 16], [Harvey, Vondrak 15], [He, Li, Liu, Wang, Xia 17].
Beck (1991) showed that an algorithmic version is possible, starting a long line of research.

Moser and Tardos (2010) found a very elegant algorithm:

1. Initialize all variables randomly.

2. While there exists an occurring bad event:
   pick one (various rules) and resample all its variables.

Many developments since then:

[Haeupler, Saha, Srinivasan 11], [Kolipaka, Szegedy 11], [Harris, Srinivasan 13], [Achlioptas, Iliopoulos 16], [Harvey, Vondrak 15], [He, Li, Liu, Wang, Xia 17].
Beck (1991) showed that an algorithmic version is possible, starting a long line of research.

Moser and Tardos (2010) found a very elegant algorithm:

1. Initialize all variables randomly.

2. While there exists an occurring bad event: pick one (various rules) and resample all its variables.

Many developments since then:

[Haeupler, Saha, Srinivasan 11], [Kolipaka, Szegedy 11], [Harris, Srinivasan 13], [Achlioptas, Iliopoulos 16], [Harvey, Vondrak 15], [He, Li, Liu, Wang, Xia 17].
**Question**

Instead of finding a solution, can we generate a random solution?

Unfortunately, Moser-Tardos’s output is not necessarily uniform.
Question

Instead of finding a solution, can we generate a random solution?

Unfortunately, Moser-Tardos’s output is not necessarily uniform.
Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

1. Randomize each vertex.
2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

- Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
- The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
Sampling independent sets?

Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

→ 1. Randomize each vertex.
→ 2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

• Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
• The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

1. Randomize each vertex.

→ 2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

• Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
• The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
**Sampling Independent Sets?**

Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

1. Randomize each vertex.

→ 2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

- Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
- The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

1. Randomize each vertex.

→ 2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

- Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
- The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

1. Randomize each vertex.
2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

- Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
- The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
Target distribution: uniform on independent sets.

Adapting Moser-Tardos:

1. Randomize each vertex.
2. Resample all connected component of size at least 2, until there is none.

This does not draw from the target distribution:

• Once a vertex is unoccupied, it will stay unoccupied till the end. Hence the empty set is overly favored.
• The process converges too fast. However uniformly sampling independent set is NP-hard (even approximately).
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + $n - 1$ edges $\Rightarrow$ Spanning Tree
Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + $n - 1$ edges $\Rightarrow$ Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root \( r \).

1. For each \( v \neq r \), assign a random arrow from \( v \) to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + \( n - 1 \) edges \( \Rightarrow \) Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root \( r \).

1. For each \( v \neq r \), assign a random arrow from \( v \) to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + \( n - 1 \) edges \( \Rightarrow \) Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

→ 2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + $n - 1$ edges $\Rightarrow$ Spanning Tree
Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + $n - 1$ edges $\Rightarrow$ Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle $+ n - 1$ edges $\Rightarrow$ Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

→ 2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle $+ n - 1$ edges $\Rightarrow$ Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root $r$.

1. For each $v \neq r$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + $n - 1$ edges $\Rightarrow$ Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root \( r \).

1. For each \( v \neq r \), assign a random arrow from \( v \) to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

→ 3. Output.

No cycle + \( n - 1 \) edges \( \Rightarrow \) Spanning Tree
Wilson’s “cycle-popping” algorithm (1996)

Goal: sample a uniform spanning tree with root \( r \).

1. For each \( v \neq r \), assign a random arrow from \( v \) to one of its neighbours.

2. While there is a (directed) cycle in the current graph, resample all arrows along all cycles.

3. Output.

No cycle + \( n - 1 \) edges \( \Rightarrow \) Spanning Tree
Cycle-popping is a special case of Moser-Tardos:

Arrows are variables. Cycles are “bad” events.

Wilson (1996) showed that the output is uniform.

But why? What is the general criterion?
Cycle-popping is a special case of Moser-Tardos:

Arrows are variables. Cycles are “bad” events.

Wilson (1996) showed that the output is uniform.

But why? What is the general criterion?
Cycle-popping is a special case of Moser-Tardos:

- Arrows are variables.
- Cycles are “bad” events.

Wilson (1996) showed that the output is uniform.

But why? What is the general criterion?
Extremal instances

We call an instance extremal:

if any two “bad” events $A_i$ and $A_j$ are either independent or disjoint.

- Extremal instances minimize the probability of solutions (in some precise sense) [Shearer 85].
- Moser-Tardos runs slowest in extremal instances.
- Slowest for searching, best for sampling.

**Theorem (G., Jerrum, Liu 17)**

When the instance is extremal, the output of Moser-Tardos is uniform.
We call an instance *extremal*:

if any two “bad” events $A_i$ and $A_j$ are either independent or disjoint.

- Extremal instances **minimize** the probability of solutions (in some precise sense) [Shearer 85].
- Moser-Tardos runs slowest in extremal instances.
- Slowest for searching, best for sampling.

**Theorem** (G., Jerrum, Liu 17)

*When the instance is extremal, the output of Moser-Tardos is uniform.*
**Extremal instances**

We call an instance **extremal**:

if any two “bad” events $A_i$ and $A_j$ are either **independent** or **disjoint**.

- Extremal instances **minimize** the probability of solutions (in some precise sense) [Shearer 85].
- **Moser-Tardos** runs slowest in extremal instances.
- Slowest for searching, best for sampling.

**Theorem (G., Jerrum, Liu 17)**

*When the instance is extremal, the output of Moser-Tardos is uniform.*
We call an instance **extremal**:

if any two “bad” events $A_i$ and $A_j$ are either **independent** or **disjoint**.

- Extremal instances **minimize** the probability of solutions (in some precise sense) [Shearer 85].
- **Moser-Tardos** runs slowest in extremal instances.
- Slowest for searching, best for sampling.

**Theorem (G., Jerrum, Liu 17)**

When the instance is extremal, the output of Moser-Tardos is uniform.
**Extremal instances**

We call an instance **extremal**: 

if any two “bad” events $A_i$ and $A_j$ are either **independent** or **disjoint**.

- Extremal instances **minimize** the probability of solutions (in some precise sense) [Shearer 85].
- **Moser-Tardos** runs slowest in extremal instances.
- Slowest for searching, best for sampling.

**Theorem (G., Jerrum, Liu 17)**

*When the instance is extremal, the output of Moser-Tardos is uniform.*
Wilson’s setup is extremal:

If two cycles share a vertex (dependent) and they both occur (overlapping), then these two cycles must be identical by following the arrow!

Other extremal instances:

- “Cluster-popping” [Gorodezky, Pak 14]
- Sink-free orientations [Bubley, Dyer 97] [Cohn, Pemantle, Propp 02] Reintroduced to show distributed LLL lower bound [Brandt, Fischer, Hirvonen, Keller, Lempiäinen, Rybicki, Suomela, Uitto 16]

We may give weights to the variables. Thus the target distribution is a product distribution conditioned on none of “bad” events occurring.
**Extremal instances**

Wilson’s setup is extremal:

If two cycles share a vertex (dependent) and they both occur (overlapping), then these two cycles must be identical by following the arrow!

Other extremal instances:

- “Cluster-popping” [Gorodezky, Pak 14]
- Sink-free orientations [Bubley, Dyer 97] [Cohn, Pemantle, Propp 02]
  Reintroduced to show distributed LLL lower bound
  [Brandt, Fischer, Hirvonen, Keller, Lempiäinen, Rybicki, Suomela, Uitto 16]

We may give weights to the variables. Thus the target distribution is a product distribution conditioned on none of “bad” events occurring.
Wilson’s setup is extremal:

If two cycles share a vertex (dependent) and they both occur (overlapping), then these two cycles must be identical by following the arrow!

Other extremal instances:

- “Cluster-popping” [Gorodezky, Pak 14]
- Sink-free orientations [Bubley, Dyer 97] [Cohn, Pemantle, Propp 02]

Reintroduced to show distributed LLL lower bound
[Brandt, Fischer, Hirvonen, Keller, Lempiäinen, Rybicki, Suomela, Uitto 16]

We may give weights to the variables. Thus the target distribution is a product distribution conditioned on none of “bad” events occurring.
Wilson’s setup is extremal:

If two cycles share a vertex (dependent) and they both occur (overlapping), then these two cycles must be identical by following the arrow!

Other extremal instances:

- “Cluster-popping” [Gorodezky, Pak 14]
- Sink-free orientations [Bubley, Dyer 97] [Cohn, Pemantle, Propp 02]
  Reintroduced to show distributed LLL lower bound
  [Brandt, Fischer, Hirvonen, Keller, Lempiäinen, Rybicki, Suomela, Uitto 16]

We may give weights to the variables. Thus the target distribution is a product distribution conditioned on none of “bad” events occurring.
Wilson’s setup is extremal:

If two cycles share a vertex (dependent) and they both occur (overlapping), then these two cycles must be identical by following the arrow!

Other extremal instances:

- “Cluster-popping” [Gorodezky, Pak 14]
- Sink-free orientations [Bubley, Dyer 97] [Cohn, Pemantle, Propp 02]
  Reintroduced to show distributed LLL lower bound
  [Brandt, Fischer, Hirvonen, Keller, Lempiäinen, Rybicki, Suomela, Uitto 16]

We may give weights to the variables. Thus the target distribution is a product distribution conditioned on none of “bad” events occurring.
Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$. When we need to resample, draw the next value in the stack.

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{1,0}$</td>
<td>$X_{1,1}$</td>
<td>$X_{1,2}$</td>
<td>$X_{1,3}$</td>
<td>$X_{1,4}$</td>
</tr>
<tr>
<td>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
</tr>
<tr>
<td>$X_{3,0}$</td>
<td>$X_{3,1}$</td>
<td>$X_{3,2}$</td>
<td>$X_{3,3}$</td>
<td>$X_{3,4}$</td>
</tr>
<tr>
<td>$X_{4,0}$</td>
<td>$X_{4,1}$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
</tr>
</tbody>
</table>
Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$. When we need to resample, draw the next value in the stack.

<table>
<thead>
<tr>
<th></th>
<th>$X_{1,0}$</th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_{3,0}$</td>
<td>$X_{3,1}$</td>
<td>$X_{3,2}$</td>
<td>$X_{3,3}$</td>
<td>$X_{3,4}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$X_4$</td>
<td>$X_{4,0}$</td>
<td>$X_{4,1}$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$. When we need to resample, draw the next value in the stack.

<table>
<thead>
<tr>
<th></th>
<th>$X_{1,0}$</th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_{3,0}$</td>
<td>$X_{3,1}$</td>
<td>$X_{3,2}$</td>
<td>$X_{3,3}$</td>
<td>$X_{3,4}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$X_4$</td>
<td>$X_{4,0}$</td>
<td>$X_{4,1}$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>
Resampling Table

Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$. When we need to resample, draw the next value in the stack.

<table>
<thead>
<tr>
<th></th>
<th>$X_{1,0}$</th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$. When we need to resample, draw the next value in the stack.

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_{1,0}$</th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_2$</td>
<td>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_{3,0}$</td>
<td>$X_{3,1}$</td>
<td>$X_{3,2}$</td>
<td>$X_{3,3}$</td>
<td>$X_{3,4}$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$X_4$</td>
<td>$X_{4,0}$</td>
<td>$X_{4,1}$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$. When we need to resample, draw the next value in the stack.

<table>
<thead>
<tr>
<th></th>
<th>$X_{1,0}$</th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_{3,0}$</td>
<td>$X_{3,1}$</td>
<td>$X_{3,2}$</td>
<td>$X_{3,3}$</td>
<td>$X_{3,4}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$X_2$</td>
<td>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$X_4$</td>
<td>$X_{4,0}$</td>
<td>$X_{4,1}$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>
For extremal instances, replacing a perfect assignment with another one will not change the resampling history!

For any output and , there is a bijection between trajectories leading to and .
For extremal instances, replacing a perfect assignment with another one will not change the resampling history!

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_1,1$</th>
<th>$X_1,2$</th>
<th>$X_1,3$</th>
<th>$X_{1,4}$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td>$A_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td>$A_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For any output and input, there is a bijection between trajectories leading to $X_{i,0}$ and $X_{i,1}$.
For extremal instances, replacing a perfect assignment with another one will not change the resampling history!

<table>
<thead>
<tr>
<th></th>
<th>$X_1'$</th>
<th>$X_1,0$</th>
<th>$X_1,1$</th>
<th>$X_1,2$</th>
<th>$X_1,3$</th>
<th>$X_1,4$</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For any output and $A$, there is a bijection between trajectories leading to $A$ and $A'$. 
For extremal instances, replacing a perfect assignment with another one will not change the resampling history!

For any output $\sigma$ and $\tau$, there is a bijection between trajectories leading to $\sigma$ and $\tau$. 

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_1',0$</th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_2$</td>
<td>$X_{2,1}'$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_{3,1}'$</td>
<td>$X_{3,2}$</td>
<td>$X_{3,3}$</td>
<td>$X_{3,4}$</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td>$X_{4,1}'$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
**Theorem (G., Jerrum, Liu 17)**

*Under Shearer’s condition, for extremal instances,*

\[
\mathbb{E} T = \sum_{i=1}^{m} \frac{q_i}{q_\emptyset} = \frac{\# \text{near-perfect assignments}}{\# \text{perfect assignments}}.
\]

(Shearer’s condition: \(q_S \geq 0\) for all \(S \subseteq V\), where \(q_S\) is the independence polynomial on \(G \setminus \Gamma^+(S)\) with weight \(-p_j\) on vertex \(j\).)

In general (non-extremal), \(\mathbb{E} T \leq \sum_{i=1}^{m} \frac{q_i}{q_\emptyset}\) [Kolipaka, Szegedy 11].

Hence, Moser-Tardos on extremal instances is the slowest.
**Theorem (G., Jerrum, Liu 17)**

*Under Shearer’s condition, for extremal instances,*

\[
\mathbb{E} T = \sum_{i=1}^{m} \frac{q_i}{q_\emptyset} = \frac{\# \text{ near-perfect assignments}}{\# \text{ perfect assignments}}.
\]

(Shearer’s condition: \( q_S \geq 0 \) for all \( S \subseteq V \), where \( q_S \) is the independence polynomial on \( G \setminus \Gamma^+(S) \) with weight \(-p_j\) on vertex \( j \).)

In general (non-extremal), \( \mathbb{E} T \leq \sum_{i=1}^{m} \frac{q_i}{q_\emptyset} \) [Kolipaka, Szegedy 11].

Hence, Moser-Tardos on extremal instances is the slowest.
Cluster-popping: repeatedly resample minimal clusters.

Let $\Omega_k$ be the set of subgraphs with $k$ minimal clusters.

$$Z_k := \sum_{S \in \Omega_k} p^{d_S} (1 - p)^{|S|}$$

$E T = \frac{Z_1}{Z_0}$

[G., Jerrum 17]: for bi-directed graphs, $Z_1 \leq \frac{mn}{1-p} Z_0$.

We show this by designing an injective mapping $\Omega_1 \rightarrow \Omega_0 \times V \times E$.

**Theorem**

There is an FPRAS for REACHABILITY in bi-directed graphs. The running time is $O \left( \varepsilon^{-2} p (1 - p)^{-3} m^2 n^3 \right)$ for an $(1 \pm \varepsilon)$-approximation.
Cluster-popping: repeatedly resample minimal clusters.

Let $\Omega_k$ be the set of subgraphs with $k$ minimal clusters.

$$Z_k := \sum_{S \in \Omega_k} p^{|E \setminus S|} (1 - p)^{|S|}$$

$$\mathbb{E} T = \frac{Z_1}{Z_0}$$

[G., Jerrum 17]: for bi-directed graphs, $Z_1 \leq \frac{mn}{1-p} Z_0$.

We show this by designing an injective mapping $\Omega_1 \rightarrow \Omega_0 \times V \times E$.

**Theorem**

There is an FPRAS for REACHABILITY in bi-directed graphs. The running time is $O(\varepsilon^{-2} p (1 - p)^{-3} m^2 n^3)$ for an $(1 \pm \varepsilon)$-approximation.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial\text{Bad}$.

4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
A correct algorithm for sampling independent sets

1. Randomize each vertex.
2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.
3. $\text{Res} = \text{Bad} \cup \partial\text{Bad}$.
4. Resample $\text{Res}$.
Check independence.

When the algorithm stops, it draws from the desired distribution.
A CORRECT ALGORITHM FOR SAMPLING INDEPENDENT SETS

1. Randomize each vertex.

→ 2. Let \textbf{Bad} be the set of vertices whose connected component has size \( \geq 2 \).

3. \textbf{Res} = \textbf{Bad} \cup \partial \textbf{Bad}.

4. Resample \textbf{Res}.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

$\Rightarrow 3. \quad \text{Res} = \text{Bad} \cup \partial \text{Bad}$.

4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
A CORRECT ALGORITHM FOR SAMPLING INDEPENDENT SETS

1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial \text{Bad}$.

→ 4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial \text{Bad}$.

→ 4. Resample $\text{Res}$.
    Check independence.

When the algorithm stops, it draws from the desired distribution.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial\text{Bad}$.

4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

$\rightarrow$ 3. $\text{Res} = \text{Bad} \cup \partial\text{Bad}$.

4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
A correct algorithm for sampling independent sets

1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial\text{Bad}$.

→ 4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial \text{Bad}$.

→ 4. Resample $\text{Res}$.
Check independence.

When the algorithm stops, it draws from the desired distribution.
1. Randomize each vertex.

2. Let $\text{Bad}$ be the set of vertices whose connected component has size $\geq 2$.

3. $\text{Res} = \text{Bad} \cup \partial \text{Bad}$.

4. Resample $\text{Res}$.
   Check independence.

When the algorithm stops, it draws from the desired distribution.
We also gave a general algorithm, and is optimal in certain restricted cases, up to constants [G., Jerrum, Liu 17].

But it falls short in general. My conjecture is that there is an efficient algorithm whenever $p\Delta^2 \leq C$ for some constant $C$.

On the other hand, there is a constant $C'$ such that if $p\Delta^2 \geq C'$, then sampling is NP-hard [Bezáková, Galanis, Goldberg, G., Štefankovič 16].
We also gave a general algorithm, and is optimal in certain restricted cases, up to constants [G., Jerrum, Liu 17].

But it falls short in general. My conjecture is that there is an efficient algorithm whenever $p\Delta^2 \leq C$ for some constant $C$.

On the other hand, there is a constant $C'$ such that if $p\Delta^2 \geq C'$, then sampling is $\text{NP}$-hard [Bezáková, Galanis, Goldberg, G., Štefankovič 16].
We also gave a general algorithm, and is optimal in certain restricted cases, up to constants [G., Jerrum, Liu 17].

But it falls short in general. My conjecture is that there is an efficient algorithm whenever $p\Delta^2 \leq C$ for some constant $C$.

On the other hand, there is a constant $C'$ such that if $p\Delta^2 \geq C'$, then sampling is \textbf{NP}-hard [Bezáková, Galanis, Goldberg, G., Štefankovič 16].
CONCLUDING REMARKS
Open Problems

• How to sample connected subgraphs (or approximate reliability)?

• What is the optimal sampling algorithm in the local lemma setting in general?

• Can we do this for perfect matchings - resampling permutations???
A professor is one who can speak on any subject for precisely fifty minutes.

— Norbert Wiener

THANK YOU!

arXiv:1611.01647
arXiv:1709.08561