Uniform Sampling through the Lovász Local Lemma

Heng Guo
Berkeley, Jun 06 2017
Queen Mary, University of London
Draft: arxiv.org/abs/1611.01647

Joint with Mark Jerrum (QMUL) and Jingcheng Liu (Berkeley)
A tale of two algorithms

(Moser and Tardos meet Wilson)
Lovász Local Lemma

Φ: a $k$-CNF formula with degree $d$.

$$\Phi = C_1 \land C_2 \land \cdots \land C_m$$

Degree: any variable $x$ belongs to at most $d$ clauses.

Lovász Local Lemma [Erdős, Lovász 75]:
if $d \leq \frac{2^k}{ek}$, then there always exists a satisfying assignment to $\Phi$.

LLL only guarantees an exponentially small probability.
Lovász Local Lemma

\( \Phi \): a \( k \)-CNF formula with degree \( d \).

\[ \Phi = C_1 \land C_2 \land \cdots \land C_m \]

Degree: any variable \( x \) belongs to at most \( d \) clauses.

Lovász Local Lemma [Erdős, Lovász 75]:
if \( d \leq \frac{2^k}{e^k} \), then there always exists a satisfying assignment to \( \Phi \).

LLL only guarantees an exponentially small probability.
Lovász Local Lemma

\[ \Phi: \text{a } k\text{-CNF formula with degree } d. \]

\[ \Phi = C_1 \land C_2 \land \cdots \land C_m \]

**Degree:** any variable \( x \) belongs to at most \( d \) clauses.

**Lovász Local Lemma [Erdős, Lovász 75]:**

If \( d \leq \frac{2^k}{e^k} \), then there always exists a satisfying assignment to \( \Phi \).

LLL only guarantees an exponentially small probability.
A remarkable breakthrough is due to [Moser, Tardos 10], where they found an efficient version of LLL:

1. Initialize all variables randomly.
2. While there exists an unsatisfied clause: pick one (various rules) and resample all its variables.

[Moser, Tardos 10] showed that this algorithm is efficient under the same condition as LLL.
A remarkable breakthrough is due to [Moser, Tardos 10], where they found an efficient version of LLL:

1. Initialize all variables randomly.
2. While there exists an unsatisfied clause:
   pick one (various rules) and resample all its variables.

[Moser, Tardos 10] showed that this algorithm is efficient under the same condition as LLL.
A remarkable breakthrough is due to [Moser, Tardos 10], where they found an efficient version of LLL:

1. Initialize all variables randomly.
2. While there exists an unsatisfied clause: pick one (various rules) and resample all its variables.

[Moser, Tardos 10] showed that this algorithm is efficient under the same condition as LLL.
Moser-Tardos works for the general “variable” framework:

Variables $X_1, \ldots, X_n$  

“Bad” events $A_1, \ldots, A_m$

The goal is to find a “perfect” assignment of the variables avoiding all “bad” events.

Equivalently, this is a product distribution conditioned on none of $A_i$ occurring.

Symmetric LLL condition: $ep\Delta \leq 1$

$p$: probability of $A_i$  

$\Delta$: # of dependent events of $A_i$

For $k$-CNF, $p = 2^{-k}$ and $\Delta \leq (d - 1)k$. 
Moser-Tardos works for the general “variable” framework:

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

The goal is to find a “perfect” assignment of the variables avoiding all “bad” events.

Equivalently, this is a product distribution conditioned on none of $A_i$ occurring.

Symmetric LLL condition: $ep\Delta \leq 1$

$p$: probability of $A_i$  $\Delta$: # of dependent events of $A_i$

For $k$-CNF, $p = 2^{-k}$ and $\Delta \leq (d - 1)k$. 
Question
Instead of finding a solution, can we uniformly generate a solution?

Unfortunately, Moser-Tardos’s output is not necessarily uniform.
Consider independent sets on a path of length 2.

If a vertex starts unoccupied, it will stay unoccupied.

The empty set is favored.
Question
Instead of finding a solution, can we uniformly generate a solution?

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

Consider independent sets on a path of length 2.

If a vertex starts unoccupied, it will stay unoccupied.

The empty set is favored.
Question
Instead of finding a solution, can we uniformly generate a solution?

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

Consider independent sets on a path of length 2.

If a vertex starts unoccupied, it will stay unoccupied.

The empty set is favored.
Searching vs. Sampling

**Question**
Instead of finding a solution, can we uniformly generate a solution?

Unfortunately, Moser-Tardos’s output is not necessarily uniform. Consider independent sets on a path of length 2.

If a vertex starts unoccupied, it will stay unoccupied.

The empty set is favored.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a 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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

→ 3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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 vertices along all cycles.

3. Output.

When this process stops, there is no cycle and it results in a spanning tree.
Wilson’s “cycle-popping” algorithm

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? Wilson’s proof is *ad hoc*. Is there a general criteria?
Wilson’s “cycle-popping” algorithm

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? Wilson’s proof is ad hoc. Is there a general criteria?
Wilson’s “cycle-popping” algorithm

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? Wilson’s proof is ad hoc. Is there a general criteria?
Why is Wilson’s algorithm uniform?
Dependency graph $G = (V, E)$:

- $V$ corresponds to events;
- $(i, j) \notin E \implies A_i$ and $A_j$ are independent.
  (In the variable framework, $\text{var}(A_i) \cap \text{var}(A_j) = \emptyset$.)

Then $\Delta$ is the maximum degree in $G$.

($\Delta$: max # of dependent events of $A_i$)

LLL condition: $e p \Delta \leq 1$. 


Dependency graph \( G = (V, E) \):

- \( V \) corresponds to events;
- \((i, j) \notin E \Rightarrow A_i \) and \( A_j \) are independent.
  
  (In the variable framework, \( \text{var}(A_i) \cap \text{var}(A_j) = \emptyset \).)

Then \( \Delta \) is the maximum degree in \( G \).

(\( \Delta \): max # of dependent events of \( A_i \))

LLL condition: \( ep\Delta \leq 1 \).
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 (given the same dependency graph). [Shearer 85]
- Moser-Tardos is the slowest on extremal instances.
- Slowest for searching, best for sampling.

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

*For extremal instances, 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 (given the same dependency graph). [Shearer 85]
- Moser-Tardos is the slowest on extremal instances.
- Slowest for searching, best for sampling.

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

*For extremal instances, 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 (given the same dependency graph). [Shearer 85]
- **Moser-Tardos** is the slowest on extremal instances.
- Slowest for searching, best for sampling.

Theorem (G., Jerrum, Liu 17)

*For extremal instances, 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 (given the same dependency graph). [Shearer 85]
- Moser-Tardos is the slowest on extremal instances.
- Slowest for searching, best for sampling.

Theorem (G., Jerrum, Liu 17)

*For extremal instances, 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 (given the same dependency graph). [Shearer 85]
- Moser-Tardos is the slowest on extremal instances.
- Slowest for searching, best for sampling.

Theorem (G., Jerrum, Liu 17)

For extremal instances, 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 the same by following the arrow!

Other extremal instances:

• 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]

• Extremal CNF formulas
  (dependent clauses contain opposite literals)
Wilson’s setup is extremal:

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

Other extremal instances:

• 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]

• Extremal CNF formulas
  (dependent clauses contain opposite literals)
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 the same by following the arrow!

Other extremal instances:

- 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]

- Extremal CNF formulas
  (dependent clauses contain opposite literals)
Wilson’s setup is extremal:

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

Other extremal instances:

- 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]

- Extremal CNF formulas
  (dependent clauses contain opposite literals)
Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$

<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></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>

When we need to resample, draw the next value in the stack.
Resampling table

Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$

<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></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>

When we need to resample, draw the next value in the stack.
Resampling table

Assisting an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_{3,0}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td>$X_{4,0}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When we need to resample, draw the next value in the stack.
Resampling table

Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$

<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>

When we need to resample, draw the next value in the stack.
Resampling table

Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$

<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>

When we need to resample, draw the next value in the stack.
Resampling table

Associate an infinite stack $X_{i,0}, X_{i,1}, \ldots$ to each random variable $X_i$

<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></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>

When we need to resample, draw the next value in the stack.
For extremal instances, replacing a perfect assignment with another one will not change the resampling history!

<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>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
<td>⋯</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>⋯</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>⋯</td>
</tr>
</tbody>
</table>
Change the future, not the past

For \textit{extremal} instances, replacing a \textit{perfect} assignment with another one will not change the resampling history!

\[
\begin{array}{cccccc}
X_1 & X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} & \cdots \\
X_2 & & X_{2,2} & X_{2,3} & X_{2,4} & \cdots \\
X_3 & A_1 & & X_{3,3} & X_{3,4} & \cdots \\
X_4 & & A_2 & & X_{4,3} & X_{4,4} & \cdots \\
\end{array}
\]
Change the future, not the past

For extremal instances, replacing a perfect assignment with another one will not change the resampling history!

\[
\begin{array}{c|cccc}
\hline
X_1 \quad X_1' \quad X_{1,0} & X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} & \cdots \\
\hline
X_2 & A_1 & X_2' & X_{2,1} & X_{2,2} & X_{2,3} & X_{2,4} & \cdots \\
\hline
X_3 & A_2 & X_3' & X_{3,2} & X_{3,3} & X_{3,4} & \cdots \\
\hline
X_4 & X_4' & X_{4,1} & X_{4,2} & X_{4,3} & X_{4,4} & \cdots \\
\hline
\end{array}
\]
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>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<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>$X_2$</td>
<td>$A_1$</td>
<td>$X'_{2,1}$</td>
<td>$X_{2,2}$</td>
<td>$X_{2,3}$</td>
<td>$X_{2,4}$</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$A_2$</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>$X'_{4,1}$</td>
<td>$X_{4,2}$</td>
<td>$X_{4,3}$</td>
<td>$X_{4,4}$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>

For any output $\sigma$ and $\tau$, there is a **bijection** between trajectories leading to $\sigma$ and $\tau$. 
Running time of Moser-Tardos

**Theorem (Kolipaka, Szegedy 11)**

*Under Shearer’s condition, \( \mathbb{E} T \leq \sum_{i=1}^{m} \frac{q_i}{q_\emptyset} \)*

(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 \( -p_i \).)

For extremal instances:
- \( q_\emptyset \) is the prob. of perfect assignments (no \( A_i \) holds);
- \( q_i \) is the prob. of assignments such that only \( A_i \) holds.

Thus,
\[
\sum_{i=1}^{m} \frac{q_i}{q_\emptyset} = \frac{\# \text{ near-perfect assignments}}{\# \text{ perfect assignments}}
\]
Theorem (Kolipaka, Szegedy 11)

Under Shearer's condition, \( \mathbb{E} T \leq \sum_{i=1}^{m} \frac{q_i}{q_\emptyset}. \)

(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 \( -p_i \).)

For extremal instances:
- \( q_\emptyset \) is the prob. of perfect assignments (no \( A_i \) holds);
- \( q_i \) is the prob. of assignments such that only \( A_i \) holds.

Thus,
\[
\sum_{i=1}^{m} \frac{q_i}{q_\emptyset} = \frac{\# \text{ near-perfect assignments}}{\# \text{ perfect assignments}}
\]
Running time on extremal instances

Theorem (G., Jerrum, Liu 17)

Under Shearer’s condition, for extremal instances,

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

In other words, Moser-Tardos on extremal instances is slowest.

New consequences:

1. The expected number of “popped cycles” in Wilson’s algorithm is at most \(mn\).
2. The expected number of “popped sinks” for sink-free orientations is linear in \(n\) if the graph is \(d\)-regular where \(d \geq 3\).
For positive weighted independent sets, Weitz (2006) works up to the uniqueness threshold, with running time $n^{O(\log \Delta)}$. The MCMC approach runs in time $\tilde{O}(n^2)$ for a smaller region. [Efthymiou, Hayes, Štefankovič, Vigoda, Yin 16]

When $p$ satisfies Shearer’s condition with constant slack in $G$, we can approximate $q_\emptyset(G, -p)$ in time $n^{O(\log \Delta)}$.
[Harvey, Srivastava, Vondrak 16] [Patel, Regts, 16]

Is there an algorithm that doesn’t have $\Delta$ in the exponent?
Approximating the independence polynomial?

For positive weighted independent sets, Weitz (2006) works up to the uniqueness threshold, with running time $n^{O(\log \Delta)}$. The MCMC approach runs in time $\tilde{O}(n^2)$ for a smaller region. [Efthymiou, Hayes, Štefankovič, Vigoda, Yin 16]

When $p$ satisfies Shearer’s condition with constant slack in $G$, we can approximate $q_{\emptyset}(G, -p)$ in time $n^{O(\log \Delta)}$. [Harvey, Srivastava, Vondrak 16] [Patel, Regts, 16]

Is there an algorithm that doesn’t have $\Delta$ in the exponent?
For positive weighted independent sets, Weitz (2006) works up to the uniqueness threshold, with running time $n^{O(\log \Delta)}$. The MCMC approach runs in time $\tilde{O}(n^2)$ for a smaller region. [Efthymiou, Hayes, Štefankovič, Vigoda, Yin 16]

When $p$ satisfies Shearer’s condition with constant slack in $G$, we can approximate $q_\emptyset(G, -p)$ in time $n^{O(\log \Delta)}$.
[Harvey, Srivastava, Vondrak 16] [Patel, Regts, 16]

Is there an algorithm that doesn’t have $\Delta$ in the exponent?
Approximating the independence polynomial?

**Extremal:** \( \Pr(\text{perfect assignment}) = q_\emptyset(G, -p). \)

Given \( G \) and \( p \), if there are \( x_j \)'s and events \( A_i \)'s so that:

- \( \Pr(A_i) = p_i \);
- \( G \) is the dependency graph;
- \( A_i \)'s are extremal,

then we could use the uniform sampler (Moser-Tardos) to estimate \( q_\emptyset \). With constant slack, Moser-Tardos runs in expected \( O(n) \) time.

A simple construction exists if \( p_i \leq 2^{-d_i} \) (in contrast to Shearer’s threshold \( \approx \frac{1}{e\Delta} \)).

Unfortunately, gaps exist between “abstract” and “variable” versions of the local lemma. [Kolipaka, Szegedy 11] [He, Li, Liu, Wang, Xia 17]

This approach does not work near Shearer’s threshold. The situation is similar to the positive weight case, but for a different reason.
Approximating the independence polynomial?

**Extremal:** \( \Pr(\text{perfect assignment}) = q_\emptyset(G, -p) \).

Given \( G \) and \( p \), if there are \( x_j \)'s and events \( A_i \)'s so that:

- \( \Pr(A_i) = p_i \);
- \( G \) is the dependency graph;
- \( A_i \)'s are extremal,

then we could use the uniform sampler (**Moser-Tardos**) to estimate \( q_\emptyset \). With constant slack, **Moser-Tardos** runs in expected \( O(n) \) time.

A simple construction exists if \( p_i \leq 2^{-d_i} \) (in contrast to Shearer’s threshold \( \approx \frac{1}{e\Delta} \)).

Unfortunately, gaps exist between “abstract” and “variable” versions of the local lemma. [Kolipaka, Szegedy 11] [He, Li, Liu, Wang, Xia 17]

This approach does not work near Shearer’s threshold. The situation is similar to the positive weight case, but for a different reason.
Approximating the independence polynomial?

**Extremal:** $\Pr(\text{perfect assignment}) = q_\emptyset(G, -p)$.

Given $G$ and $p$, if there are $x_j$’s and events $A_i$’s so that:
- $\Pr(A_i) = p_i$;
- $G$ is the dependency graph;
- $A_i$’s are extremal,
then we could use the uniform sampler (Moser-Tardos) to estimate $q_\emptyset$. With constant slack, Moser-Tardos runs in expected $O(n)$ time.

A simple construction exists if $p_i \leq 2^{-d_i}$ (in contrast to Shearer’s threshold $\approx \frac{1}{e\Delta}$).

Unfortunately, gaps exist between “abstract” and “variable” versions of the local lemma. [Kolipaka, Szegedy 11] [He, Li, Liu, Wang, Xia 17]

This approach does not work near Shearer’s threshold. The situation is similar to the positive weight case, but for a different reason.
What else can we sample?

1. For each $v$, assign a random arrow from $v$ to one of its neighbours.
2. While there is a "small" cycle, resample all vertices along all cycles.
3. Output. When this process stops, there is no small cycle and what is left is a Hamiltonian cycle.
→ 1. For each $v$, assign a random arrow from $v$ to one of its neighbours.
What else can we sample?

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

→ 2. While there is a “small” cycle, resample all vertices along all cycles.
What else can we sample?

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

→ 2. While there is a “small” cycle, resample all vertices along all cycles.
1. For each $v$, assign a random arrow from $v$ to one of its neighbours.

→ 2. While there is a “small” cycle, resample all vertices along all cycles.
What else can we sample?

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

2. While there is a “small” cycle, resample all vertices along all cycles.
1. For each $v$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a “small” cycle, resample all vertices along all cycles.
What else can we sample?

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

2. While there is a “small” cycle, resample all vertices along all cycles.
1. For each $v$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a “small” cycle, resample all vertices along all cycles.

→ 3. Output.
1. For each $v$, assign a random arrow from $v$ to one of its neighbours.

2. While there is a “small” cycle, resample all vertices along all cycles.

3. Output.

When this process stops, there is no small cycle and what is left is a Hamiltonian cycle.
Can we sample Hamiltonian cycles efficiently?

Recall that $\mathbb{E} T = \frac{\# \text{near-perfect assignments}}{\# \text{perfect assignments}}$.

In our setting, a near-perfect assignment is a uni-cyclic arrow set.

Unfortunately, this ratio is exponentially large in a complete graph.

[Dyer, Frieze, Jerrum 98]:

In dense graphs ($\delta = (1/2 + \varepsilon)n$), Hamiltonian cycles are sufficiently dense among all 2-factors, which can be approximately sampled.

Open: Is there an efficient and exact sampler for Hamiltonian cycles in some interesting graph families?
Can we sample Hamiltonian cycles efficiently?

Recall that $\mathbb{E} T = \frac{\# \text{near-perfect assignments}}{\# \text{perfect assignments}}$.

In our setting, a near-perfect assignment is a uni-cyclic arrow set.

Unfortunately, this ratio is exponentially large in a complete graph.

[Dyer, Frieze, Jerrum 98]:

In dense graphs ($\delta = (1/2 + \varepsilon)n$), Hamiltonian cycles are sufficiently dense among all 2-factors, which can be approximately sampled.

Open: Is there an efficient and exact sampler for Hamiltonian cycles in some interesting graph families?
Can we sample Hamiltonian cycles efficiently?

Recall that \( \mathbb{E} T = \frac{\# \text{near-perfect assignments}}{\# \text{perfect assignments}} \).

In our setting, a near-perfect assignment is a uni-cyclic arrow set.

Unfortunately, this ratio is exponentially large in a complete graph.

[Dyer, Frieze, Jerrum 98]:

In dense graphs \( (\delta = (1/2 + \epsilon)n) \), Hamiltonian cycles are sufficiently dense among all 2-factors, which can be approximately sampled.

Open: Is there an efficient and exact sampler for Hamiltonian cycles in some interesting graph families?
Beyond Extremal Instances
Inspired by [Moser, Tardos 10], we found a new uniform sampler.

Partial Rejection Sampling [G., Jerrum, Liu 17]:

1. Initialize $\sigma$ — randomize all variables independently.
2. While $\sigma$ is not perfect:
   - choose an appropriate subset of events, $\text{Resample}(\sigma)$;
   - re-randomize all variables in $\text{Resample}(\sigma)$.

For extremal instances, $\text{Resample}(\sigma)$ is simply $\text{Bad}(\sigma)$.

How to choose $\text{Resample}(\sigma)$ to guarantee uniformity?
Inspired by [Moser, Tardos 10], we found a new uniform sampler.

**Partial Rejection Sampling** [G., Jerrum, Liu 17]:

1. Initialize $\sigma$ — randomize all variables independently.
2. While $\sigma$ is not perfect:
   - choose an appropriate subset of events, $\text{Resample}(\sigma)$;
   - re-randomize all variables in $\text{Resample}(\sigma)$.

For extremal instances, $\text{Resample}(\sigma)$ is simply $\text{Bad}(\sigma)$.

How to choose $\text{Resample}(\sigma)$ to guarantee uniformity?
Partial Rejection Sampling

Inspired by [Moser, Tardos 10], we found a new uniform sampler.

Partial Rejection Sampling [G., Jerrum, Liu 17]:

1. Initialize $\sigma$ — randomize all variables independently.
2. While $\sigma$ is not perfect:
   - choose an appropriate subset of events, $\text{Resample}(\sigma)$;
   - re-randomize all variables in $\text{Resample}(\sigma)$.

For extremal instances, $\text{Resample}(\sigma)$ is simply $\text{Bad}(\sigma)$.

How to choose $\text{Resample}(\sigma)$ to guarantee uniformity?
Partial Rejection Sampling

Inspired by [Moser, Tardos 10], we found a new uniform sampler.

Partial Rejection Sampling [G., Jerrum, Liu 17]:

1. Initialize $\sigma$ — randomize all variables independently.
2. While $\sigma$ is not perfect:
   - choose an appropriate subset of events, $\text{Resample}(\sigma)$;
   - re-randomize all variables in $\text{Resample}(\sigma)$.

For extremal instances, $\text{Resample}(\sigma)$ is simply $\text{Bad}(\sigma)$.

How to choose $\text{Resample}(\sigma)$ to guarantee uniformity?
What set to resample?

Let $T$ be the stopping time and $\mathcal{R} = R_1, \ldots, R_T$ be the set sequence of resampled variables.

**Goal:** conditioned on $\mathcal{R}$, all perfect assignments are reachable.

**Unblocking:** under an assignment $\sigma$, a subset $S$ of variables is unblocking, if all events intersecting $S$ are determined by $\sigma|_S$.

(only need to worry about events intersecting both $S$ and $\overline{S}$.)

**Examples:**

The set of all variables is unblocking.

For independent sets, $S$ is unblocking if $\partial S$ are all unoccupied.
What set to resample?

Let $T$ be the stopping time and $\mathcal{R} = R_1, \ldots, R_T$ be the set sequence of resampled variables.

**Goal**: conditioned on $\mathcal{R}$, all perfect assignments are reachable.

**Unblocking**: under an assignment $\sigma$, a subset $S$ of variables is unblocking, if all events intersecting $S$ are determined by $\sigma|_S$.

(only need to worry about events intersecting both $S$ and $\overline{S}$.)

Examples:

The set of all variables is unblocking.

For independent sets, $S$ is unblocking if $\partial S$ are all unoccupied.
Let $T$ be the stopping time and $\mathcal{R} = R_1, \ldots, R_T$ be the set sequence of resampled variables.

**Goal:** conditioned on $\mathcal{R}$, all perfect assignments are *reachable*.

**Unblocking:** under an assignment $\sigma$, a subset $S$ of variables is *unblocking*, if all events intersecting $S$ are determined by $\sigma|_S$.

(only need to worry about events intersecting both $S$ and $\bar{S}$.)

**Examples:**

The set of *all* variables is *unblocking*.

For independent sets, $S$ is *unblocking* if $\partial S$ are all unoccupied.
Let $T$ be the stopping time and $\mathcal{R} = R_1, \ldots, R_T$ be the set sequence of resampled variables.

**Goal:** conditioned on $\mathcal{R}$, all perfect assignments are **reachable**.

**Unblocking:** under an assignment $\sigma$, a subset $S$ of variables is **unblocking**, if all events intersecting $S$ are determined by $\sigma|_S$.

(only need to worry about events intersecting both $S$ and $\bar{S}$.)

**Examples:**

The set of **all** variables is **unblocking**.

For independent sets, $S$ is **unblocking** if $\partial S$ are all unoccupied.
Resampling set

Given an assignment $\sigma$, we want $\text{Resample}(\sigma)$ to satisfy:

1. $\text{Resample}(\sigma)$ contains $\text{Bad}(\sigma)$;
2. $\text{Resample}(\sigma)$ is unblocking;
3. What is revealed has to be resampled.

$\text{Resample}(\sigma)$ can be found by a breadth-first search.

In the worst case we may resample all variables.
Given an assignment $\sigma$, we want $\text{Resample}(\sigma)$ to satisfy:

1. $\text{Resample}(\sigma)$ contains $\text{Bad}(\sigma)$;

2. $\text{Resample}(\sigma)$ is unblocking;

3. What is revealed has to be resampled.

$\text{Resample}(\sigma)$ can be found by a breadth-first search. In the worst case we may resample all variables.
Given an assignment $\sigma$, we want $\text{Resample}(\sigma)$ to satisfy:

1. $\text{Resample}(\sigma)$ contains $\text{Bad}(\sigma)$;

2. $\text{Resample}(\sigma)$ is unblocking;

3. What is revealed has to be resampled.

$\text{Resample}(\sigma)$ can be found by a breadth-first search. In the worst case we may resample all variables.
Given an assignment $\sigma$, we want $\text{Resample}(\sigma)$ to satisfy:

1. $\text{Resample}(\sigma)$ contains $\text{Bad}(\sigma)$;
2. $\text{Resample}(\sigma)$ is unblocking;
3. What is revealed has to be resampled.

$\text{Resample}(\sigma)$ can be found by a breadth-first search. In the worst case we may resample all variables.
Given an assignment \( \sigma \), we want \( \text{Resample}(\sigma) \) to satisfy:

1. \( \text{Resample}(\sigma) \) contains \( \text{Bad}(\sigma) \);
2. \( \text{Resample}(\sigma) \) is unblocking;
3. What is revealed has to be resampled.

\( \text{Resample}(\sigma) \) can be found by a breadth-first search.

In the worst case we may resample all variables.
Markov chain is a random walk in the solution space.
(The solution space has to be connected!)
PRS is a local search on the whole space.
Partial Rejection Sampling vs Markov chains

PRS is a local search on the whole space.
(Connectivity is not an issue.)
PRS is a local search on the whole space. (Uniformity is guaranteed by the bijection.)
Partial Rejection Sampling:
repeatedly resample the appropriately chosen $\text{Resample}(\sigma)$.

Theorem (G., Jerrum, Liu 17)
When PRS halts, its output is uniform.

Some applications beyond extremal instances:
• Weighted independent sets.
• $k$-CNF formulas.
Partial Rejection Sampling:
repeatedly resample the appropriately chosen \( \text{Resample}(\sigma) \).

Theorem (G., Jerrum, Liu 17)

When \( \text{PRS} \) halts, its output is uniform.

Some applications beyond extremal instances:

- Weighted independent sets.
- \( k \)-CNF formulas.
Partial Rejection Sampling:
repeatedly resample the appropriately chosen \texttt{Resample}(\sigma).

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

*When PRS halts, its output is uniform.*

Some applications beyond extremal instances:

- Weighted independent sets.
- \(k\)-CNF formulas.
1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
→ 1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
Sampling independent sets

1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
Sampling independent sets

1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
1. Randomize each vertex.

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

\[ \rightarrow 3. \text{Resample} = \text{Bad} \cup \partial \text{Bad}. \]

4. Resample \( \text{Resample} \).
   Check independence.

When the algorithm stops, it is a uniform independent set.
1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
1. Randomize each vertex.

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

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

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

When the algorithm stops, it is a uniform independent set.
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{Resample} = \textbf{Bad} \cup \partial \textbf{Bad}.

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

When the algorithm stops, it is a uniform independent set.
Set-up

Vertex weight $\lambda$. “Bad” events are occupied edges: $p = \left(\frac{\lambda}{1+\lambda}\right)^2$. Dependency graph is the line graph. $\Delta = 2d - 2$. 
Running time — independent sets

Set-up

Vertex weight $\lambda$. “Bad” events are occupied edges: $p = \left(\frac{\lambda}{1+\lambda}\right)^2$.
Dependency graph is the line graph. $\Delta = 2d - 2$.

Suppose $k = |\text{Resample}_t|$.

Then $\mathbb{E}|\text{Bad}_{t+1}| \leq ep\Delta \cdot k$
Running time — independent sets

Set-up

Vertex weight $\lambda$. “Bad” events are occupied edges: $p = \left(\frac{\lambda}{1+\lambda}\right)^2$. Dependency graph is the line graph. $\Delta = 2d - 2$.

Suppose $k = |\text{Resample}_t|$.

Then $\mathbb{E}|\text{Bad}_{t+1}| \leq ep\Delta \cdot k$

1. Both $\text{Resample}_t$ and $\partial \text{Resample}_t$ are “dangerous”, and $|\partial \text{Resample}_t| \leq \Delta \cdot k$.

2. Under LLL condition, for any event $E$,

$$\Pr(E | \bigwedge \overline{A}_i) \leq e \Pr(E).$$
Set-up

Vertex weight $\lambda$. “Bad” events are occupied edges: $p = \left(\frac{\lambda}{1+\lambda}\right)^2$. Dependency graph is the line graph. $\Delta = 2d - 2$.

Suppose $k = |\text{Resample}_t|$.

Then $\mathbb{E}|\text{Bad}_{t+1}| \leq ep\Delta \cdot k \quad \Rightarrow \quad \mathbb{E}|\text{Resample}_{t+1}| \leq ep\Delta^2 \cdot k$. 
Running time — independent sets

Set-up

Vertex weight $\lambda$. “Bad” events are occupied edges: $p = \left( \frac{\lambda}{1+\lambda} \right)^2$. Dependency graph is the line graph. $\Delta = 2d - 2$.

Suppose $k = |\text{Resample}_t|$.

Then $\mathbb{E}|\text{Bad}_{t+1}| \leq ep\Delta \cdot k \quad \Rightarrow \quad \mathbb{E}|\text{Resample}_{t+1}| \leq ep\Delta^2 \cdot k$.

The resampling region shrinks if

$$ep\Delta^2 < 1 \quad \Leftrightarrow \quad \lambda = O(1/d)$$

(Recall that the local lemma requires $ep\Delta \leq 1$.)
Phase transition of independent sets

Sampling independent sets with weight $\lambda$ and maximum degree $d$:

- If $\lambda < \lambda_c(d) \approx \frac{e}{d}$, there is a deterministic, approximate, and polynomial-time algorithm [Weitz 06]. (Best randomized algorithm (based on Markov chains) has a worse range but $O(n \log n)$ running time.)

- If $\lambda > \lambda_c(d) \approx \frac{e}{d}$, it is $\text{NP}$-hard [Sly 10].

Our algorithm has linear expected running time if $\lambda \leq \frac{1}{2\sqrt{ed} - 1}$. The range is off by a constant, but it is fast, simple, exact, and distributed.
Phase transition of independent sets

Sampling independent sets with weight $\lambda$ and maximum degree $d$:

- If $\lambda < \lambda_c(d) \approx \frac{e}{d}$, there is a deterministic, approximate, and polynomial-time algorithm [Weitz 06]. (Best randomized algorithm (based on Markov chains) has a worse range but $O(n \log n)$ running time.)

- If $\lambda > \lambda_c(d) \approx \frac{e}{d}$, it is NP-hard [Sly 10].

Our algorithm has linear expected running time if $\lambda \leq \frac{1}{2\sqrt{ed}-1}$.

The range is off by a constant, but it is fast, simple, exact, and distributed.
 runaway time — general case

\[ \exists \text{ constant } C \text{ s.t. if } p\Delta^2 \geq C, \text{ then even approximate sampling is NP-hard.} \]

Hence we have to assume stronger conditions than \( ep\Delta \leq 1 \).

Independent sets are nice in that \text{Resample} is just \text{Bad} \cup \partial \text{Bad}. In general, \text{Resample} can expand more than one hop. Denote by \( r_{ij} \) the probability that \( A_i \) may expand to \( A_j \). Let \( r = \max\{r_{ij}\} \).

Theorem (G., Jerrum, Liu 17)

\[ \text{If } ep\Delta^2 \leq 1/6 \text{ and } er\Delta \leq 1/3, \text{ then } \mathbb{E}T = O(m). \]

The expected number of rounds is \( O(\log m) \).

The expected number of variable resamples is \( O(n \log m) \).

Our proof is a supermartingale argument on \(|\text{Resample}|\).

The condition on \( r \) is necessary.
∃ constant C s.t. if $p\Delta^2 \geq C$, then even approximate sampling is $\textbf{NP}$-hard. Hence we have to assume stronger conditions than $ep\Delta \leq 1$.

Indenependent sets are nice in that $\textbf{Resample}$ is just $\textbf{Bad} \cup \partial \textbf{Bad}$. In general, $\textbf{Resample}$ can expand more than one hop. Denote by $r_{ij}$ the probability that $A_i$ may expand to $A_j$. Let $r = \max\{r_{ij}\}$.

Theorem (G., Jerrum, Liu 17)

If $ep\Delta^2 \leq 1/6$ and $er\Delta \leq 1/3$, then $E T = O(m)$.

The expected number of rounds is $O(\log m)$.

The expected number of variable resamples is $O(n \log m)$.

Our proof is a supermartingale argument on $|\textbf{Resample}|$.

The condition on $r$ is necessary.
Running time — general case

\[ \exists \text{ constant } C \text{ s.t. if } p\Delta^2 \geq C, \text{ then even approximate sampling is } \textbf{NP}-\text{hard. Hence we have to assume stronger conditions than } ep\Delta \leq 1. \]

Indenependent sets are nice in that \textbf{Resample} is just \textbf{Bad} \cup \partial\textbf{Bad}. In general, \textbf{Resample} can expand more than one hop. Denote by \( r_{ij} \) the probability that \( A_i \) may expand to \( A_j \). Let \( r = \max\{r_{ij}\} \).

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

If \( ep\Delta^2 \leq 1/6 \) and \( er\Delta \leq 1/3 \), then \( \mathbb{E} T = O(m) \).

The expected number of rounds is \( O(\log m) \).

The expected number of variable resamples is \( O(n \log m) \).

Our proof is a supermartingale argument on \(|\textbf{Resample}|\).

The condition on \( r \) is necessary.
∃ constant $C$ s.t. if $p\Delta^2 \geq C$, then even approximate sampling is \textbf{NP}-hard. Hence we have to assume stronger conditions than $ep\Delta \leq 1$.

Indenependent sets are nice in that \textbf{Resample} is just $\text{Bad} \cup \partial \text{Bad}$. In general, \textbf{Resample} can expand more than one hop. Denote by $r_{ij}$ the probability that $A_i$ may expand to $A_j$. Let $r = \max\{r_{ij}\}$.

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

*If $ep\Delta^2 \leq 1/6$ and $er\Delta \leq 1/3$, then $\mathbb{E}T = O(m)$.*

The expected number of rounds is $O(\log m)$.

The expected number of variable resamples is $O(n \log m)$.

Our proof is a supermartingale argument on $|\textbf{Resample}|$.

The condition on $r$ is necessary.
**Sampling \( k\)-CNF**

**NP-Hardness for sampling:**
- \( d \geq 3 \) — decision hardness for general formula
- \( d \geq 6, k = 2 \) (monotone formula) [Sly 10]
- \( d \geq 5 \cdot 2^{k/2} \) (monotone formula) [Bezáková, Galanis, Goldberg, G., Štefankovič 16]
  (LLL condition is \( d \leq \frac{2^k}{ek} \))

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

*PRS* has linear expected running time if \( d \leq \frac{1}{6e} \cdot 2^{k/2} \), and any two dependent clauses share at least \( \min\{\log dk, k/2\} \) variables.

**NP-hard even if** \( d \geq 5 \cdot 2^{k/2} \) and intersection \( = k/2 \) [BGGGŠ 16]
NP-Hardness for sampling:
- $d \geq 3$ — decision hardness for general formula
- $d \geq 6, k = 2$ (monotone formula) [Sly 10]
- $d \geq 5 \cdot 2^{k/2}$ (monotone formula) [Bezáková, Galanis, Goldberg, G., Štefankovič 16]
  (LLL condition is $d \leq \frac{2^k}{e^k}$)

Theorem (G., Jerrum, Liu 17)

*PRS* has linear expected running time if $d \leq \frac{1}{6e} \cdot 2^{k/2}$, and any two dependent clauses share at least $\min\{\log dk, k/2\}$ variables.

NP-hard even if $d \geq 5 \cdot 2^{k/2}$ and intersection $= k/2$ [BGGGŠ 16]
Sampling $k$-CNF

**NP-Hardness for sampling:**
- $d \geq 3$ — decision hardness for general formula
- $d \geq 6, k = 2$ (monotone formula) [Sly 10]
- $d \geq 5 \cdot 2^{k/2}$ (monotone formula) [Bezáková, Galanis, Goldberg, G., Štefankovič 16]
  (LLL condition is $d \leq \frac{2^k}{e^k}$)

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

*PRS has linear expected running time if $d \leq \frac{1}{6e} \cdot 2^{k/2}$, and any two dependent clauses share at least $\min\{\log dk, k/2\}$ variables.*

**NP-hard even if $d \geq 5 \cdot 2^{k/2}$ and intersection $= k/2$** [BGGGŠ 16]
### Sampling $k$-CNF

**NP-Hard if** $d \geq 3$ (decision); or $d \geq 6, k = 2$ (monotone) [Sly 10]; or $d \geq 5 \cdot 2^{k/2}$ (monotone) and intersection $= k/2$ [BGGGŠ 16].

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Condition</th>
<th>Restriction</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bubley, Dyer 97]</td>
<td>$d = 2$</td>
<td></td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Bordewich, Dyer, Karpinski 06]</td>
<td>$d \leq k - 2$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Liu, Lu 15]</td>
<td>$d \leq 5$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[BGGGŠ 16]</td>
<td>$d = 6, k = 3$ or $d \leq k$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[Hermon, Sly, Zhang 17]</td>
<td>$d \leq c_2^{k/2}$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Moitra 17]</td>
<td>$d \leq \tilde{O}(2^{k/60})$</td>
<td></td>
<td>Correlation decay + LP</td>
</tr>
<tr>
<td>[G., Jerrum, Liu 17]</td>
<td>$d \leq c_2^{k/2}$</td>
<td>Intersection $\geq \min{\log dk, k/2}$</td>
<td>PRS</td>
</tr>
</tbody>
</table>

All other methods are approximate, whereas PRS is exact.
Sampling $k$-CNF

**NP-Hard if** $d \geq 3$ (decision); or $d \geq 6, k = 2$ (monotone) \cite{Sly10}; or $d \geq 5 \cdot 2^{k/2}$ (monotone) and intersection $= k/2$ \cite{BGGGS16}.

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Condition</th>
<th>Restriction</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>\cite{BubleyDyer97}</td>
<td>$d = 2$</td>
<td></td>
<td>Markov chain</td>
</tr>
<tr>
<td>\cite{BordewichDyerKarpinski06}</td>
<td>$d \leq k - 2$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>\cite{LiuLu15}</td>
<td>$d \leq 5$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>\cite{BGGGS16}</td>
<td>$d = 6, k = 3$ or $d \leq k$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>\cite{HermonSlyZhang17}</td>
<td>$d \leq c2^{k/2}$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>\cite{Moitra17}</td>
<td>$d \leq \tilde{O}(2^{k/60})$</td>
<td></td>
<td>Correlation decay + LP</td>
</tr>
<tr>
<td>\cite{GJerrumLiu17}</td>
<td>$d \leq c2^{k/2}$</td>
<td>Intersection $\geq \min{\log dk, k/2}$</td>
<td>PRS</td>
</tr>
</tbody>
</table>

All other methods are approximate, whereas PRS is exact.
### Sampling $k$-CNF

**NP-Complete if** $d \geq 3$ (decision); or $d \geq 6$, $k = 2$ (monotone) [Sly 10]; or $d \geq 5 \cdot 2^{k/2}$ (monotone) and intersection $\geq k/2$ [BGGGŠ 16].

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Condition</th>
<th>Restriction</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bubley, Dyer 97]</td>
<td>$d = 2$</td>
<td></td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Bordewich, Dyer, Karpinski 06]</td>
<td>$d \leq k - 2$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Liu, Lu 15]</td>
<td>$d \leq 5$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[BGGGŠ 16]</td>
<td>$d = 6$, $k = 3$ or $d \leq k$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[Hermon, Sly, Zhang 17]</td>
<td>$d \leq c 2^{k/2}$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Moitra 17]</td>
<td>$d \leq \tilde{O}(2^{k/60})$</td>
<td></td>
<td>Correlation decay + LP</td>
</tr>
<tr>
<td>[G., Jerrum, Liu 17]</td>
<td>$d \leq c 2^{k/2}$</td>
<td>Intersection $\geq \min{\log dk, k/2}$</td>
<td>PRS</td>
</tr>
</tbody>
</table>

All other methods are approximate, whereas PRS is exact.
### Sampling $k$-CNF

NP-Hard if $d \geq 3$ (decision); or $d \geq 6$, $k = 2$ (monotone) [Sly 10]; or $d \geq 5 \cdot 2^{k/2}$ (monotone) and intersection $\geq k/2$ [BGGGŠ 16].

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Condition</th>
<th>Restriction</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bubley, Dyer 97]</td>
<td>$d = 2$</td>
<td></td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Bordewich, Dyer, Karpinski 06]</td>
<td>$d \leq k - 2$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Liu, Lu 15]</td>
<td>$d \leq 5$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[BGGGŠ 16]</td>
<td>$d = 6$, $k = 3$ or $d \leq k$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[Hermon, Sly, Zhang 17]</td>
<td>$d \leq c 2^{k/2}$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Moitra 17]</td>
<td>$d \leq \tilde{O}(2^{k/60})$</td>
<td></td>
<td>Correlation decay + LP</td>
</tr>
<tr>
<td>[G., Jerrum, Liu 17]</td>
<td>$d \leq c 2^{k/2}$</td>
<td>Intersection $\geq \min{\log dk, k/2}$</td>
<td>PRS</td>
</tr>
</tbody>
</table>

All other methods are approximate, whereas PRS is exact.
## Sampling $k$-CNF

NP-Hard if $d \geq 3$ (decision); or $d \geq 6$, $k = 2$ (monotone) [Sly 10]; or $d \geq 5 \cdot 2^{k/2}$ (monotone) and intersection $= k/2$ [BGGGŠ 16].

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Condition</th>
<th>Restriction</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bubley, Dyer 97]</td>
<td>$d = 2$</td>
<td></td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Bordewich, Dyer, Karpinski 06]</td>
<td>$d \leq k - 2$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Liu, Lu 15]</td>
<td>$d \leq 5$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[BGGGŠ 16]</td>
<td>$d = 6$, $k = 3$ or $d \leq k$</td>
<td>monotone</td>
<td>Correlation decay</td>
</tr>
<tr>
<td>[Hermon, Sly, Zhang 17]</td>
<td>$d \leq c 2^{k/2}$</td>
<td>monotone</td>
<td>Markov chain</td>
</tr>
<tr>
<td>[Moitra 17]</td>
<td>$d \leq \tilde{O}(2^{k/60})$</td>
<td></td>
<td>Correlation decay + LP</td>
</tr>
<tr>
<td>[G., Jerrum, Liu 17]</td>
<td>$d \leq c 2^{k/2}$</td>
<td>Intersection $\geq \min{\log dk, k/2}$</td>
<td>PRS</td>
</tr>
</tbody>
</table>

All other methods are approximate, whereas PRS is exact.
Concluding remarks
• For extremal instances, Moser-Tardos is uniform, with expected running time $\frac{\text{# “near-perfect” assignments}}{\text{# “perfect” assignments}}$.

• For general instances, we need to carefully choose a resampling set to ensure uniformity.

• The expected running time is linear if $p\Delta^2 = O(1)$ and $r\Delta = O(1)$. 
• For extremal instances, Moser-Tardos is uniform, with expected running time \( \frac{\# \text{“near-perfect” assignments}}{\# \text{“perfect” assignments}} \).

• For general instances, we need to carefully choose a resampling set to ensure uniformity.

• The expected running time is linear if \( p\Delta^2 = O(1) \) and \( r\Delta = O(1) \).
For extremal instances, Moser-Tardos is uniform, with expected running time $\frac{\# \text{“near-perfect” assignments}}{\# \text{“perfect” assignments}}$.

For general instances, we need to carefully choose a resampling set to ensure uniformity.

The expected running time is linear if $p\Delta^2 = O(1)$ and $r\Delta = O(1)$. 
Sampling threshold under LLL?

Existence threshold \([\text{Erdős, Lovász 75}]\)

\[ \approx \frac{1}{e\Delta} \]
Sampling threshold under LLL?

Searching threshold $[\text{Moser, Tardos 10}]$

$$\approx \frac{1}{e\Delta}$$
Sampling threshold under LLL?

\[ O(1/\Delta^2) \approx \frac{1}{e\Delta} \]
Open problems

• $O(n^c)$ algorithm for the independence polynomial with negative weights?

• Can we sample Hamiltonian cycles exactly and efficiently in some interesting graph families?

• How to remove the side condition on intersections?
  • Where is the transition threshold for $k$-CNF of degree $d$?

• Beyond the variable model - resampling permutations???
Thank you!