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)

Φ: a k-CNF formula with degree d.

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

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

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

LLL only guarantees an exponentially small probability.

Φ: a k-CNF formula with degree d.

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

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

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

LLL only guarantees an exponentially small probability.

Φ: a k-CNF formula with degree d.

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

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

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

LLL only guarantees an exponentially small probability.

Moser-Tardos resampling algorithm

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 resampling algorithm

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 resampling algorithm

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.

Variable framework

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: $ep\Delta \leqslant 1$

p: probability of A_i Δ: # of dependent events of A_i

For k-CNF, $p = 2^{-k}$ and $\Delta \leqslant (d-1)k$.

Variable framework

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: $ep\Delta \leqslant 1$

p: probability of A_i Δ: # of dependent events of A_i

For k-CNF, $p = 2^{-k}$ and $\Delta \leqslant (d-1)k$.

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.

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.

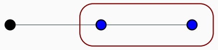
Question

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.

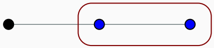
Question

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.

Wilson's "cycle-popping" algorithm

Goal: sample a uniform spanning tree with root r.

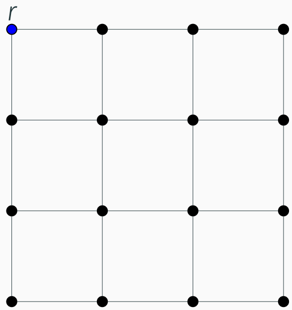
1. For each $v \neq r$, assign a random arrow from v to one of its neighbours.
2. While there is a (directed) cycle in the current graph, resample all vertices along all cycles.
3. Output.

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.

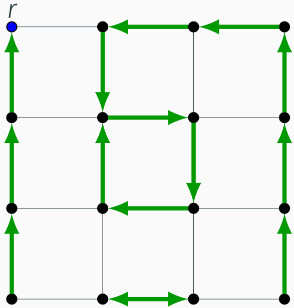
Wilson's "cycle-popping" algorithm

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.

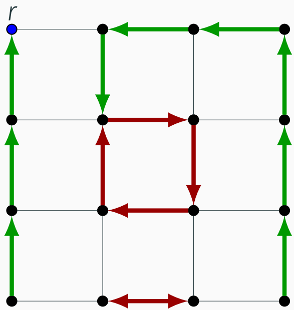
Wilson's "cycle-popping" algorithm

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.

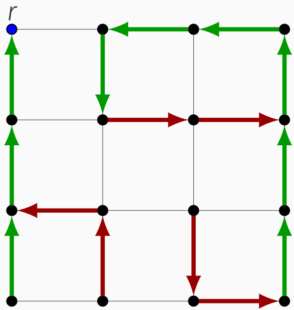
Wilson's "cycle-popping" algorithm

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.

$\rightarrow$ 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.

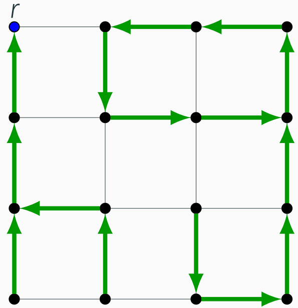
Wilson's "cycle-popping" algorithm

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.

$\rightarrow$ 2. While there is a (directed) cycle in the current graph, resample all vertices along all cycles.

3. Output.

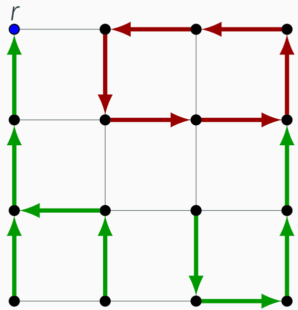


When this process stops, there is no cycle and it results in a spanning tree.

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.

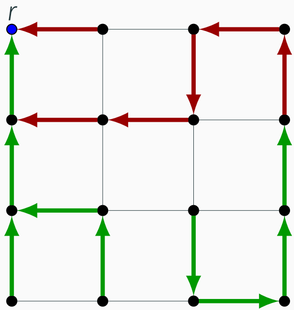
Wilson's "cycle-popping" algorithm

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.

$\rightarrow$ 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.

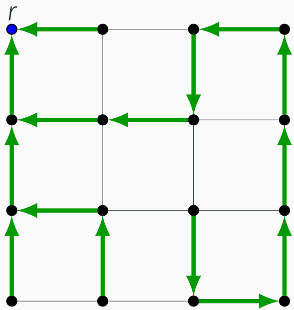
Wilson's "cycle-popping" algorithm

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.

$\rightarrow$ 2. While there is a (directed) cycle in the current graph, resample all vertices along all cycles.

3. Output.



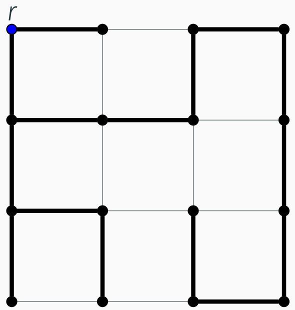
When this process stops, there is no cycle and it results in a spanning tree.

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.

$\rightarrow$ 3. Output.

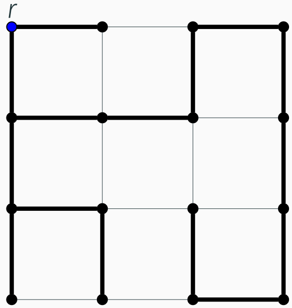


When this process stops, there is no cycle and it results in a spanning tree.

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

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

V corresponds to events;

$(i, j) \notin E \quad \Rightarrow \quad A_i$ and A_j are independent.

(In the variable framework, $\mathtt{var}(A_i) \cap \mathtt{var}(A_j) = \emptyset$.)

Then Δ is the maximum degree in G.

(Δ: max # of dependent events of A_i)

LLL condition: $ep\Delta \leqslant 1$.

Dependency Graph

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

V corresponds to events;

$(i, j) \notin E \quad \Rightarrow \quad A_i$ and A_j are independent.

(In the variable framework, $\mathtt{var}(A_i) \cap \mathtt{var}(A_j) = \emptyset$.)

Then Δ is the maximum degree in G.

(Δ: max # of dependent events of A_i)

LLL condition: $ep\Delta \leqslant 1$.

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.

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

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)

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)

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)

Resampling table

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

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

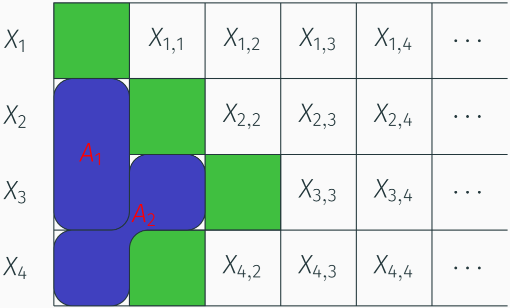
Change the future, not the past

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

X_1	$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$\cdots$
X_2	$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$\cdots$
X_3	$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$\cdots$
X_4	$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$\cdots$

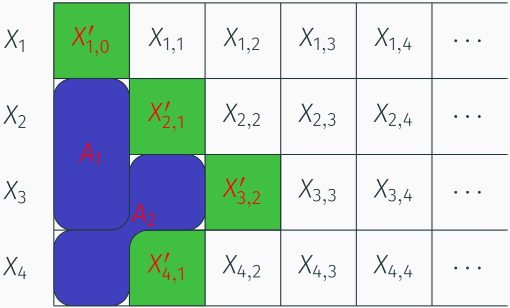
Change the future, not the past

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



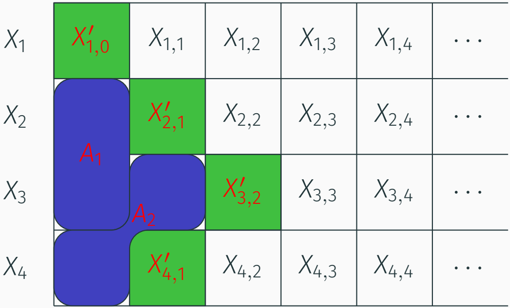
Change the future, not the past

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



Change the future, not the past

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



For any output σ and τ, there is a bijection between trajectories leading to σ and τ.

Running time of Moser-Tardos

Theorem (Kolipaka, Szegedy 11)

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

(Shearer's condition: $q_S \geqslant 0$ 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 of Moser-Tardos

Theorem (Kolipaka, Szegedy 11)

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

(Shearer's condition: $q_S \geqslant 0$ 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 $d \geqslant 3$.

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 $\widetilde{O}(n^2)$ for a smaller region. [Efthymiou, Hayes, Štefankovič, Vigoda, Yin 16]

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

Is there an algorithm that doesn't have Δ 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 $\widetilde{O}(n^2)$ for a smaller region. [Efthymiou, Hayes, Štefankovič, Vigoda, Yin 16]

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

Is there an algorithm that doesn't have Δ 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 $\widetilde{O}(n^2)$ for a smaller region. [Efthymiou, Hayes, Štefankovič, Vigoda, Yin 16]

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

Is there an algorithm that doesn't have Δ in the exponent?

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

Given G and $\mathbf{p}$, 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 $p_i \leqslant 2^{-d_i}$ (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: $\Pr(\text{perfect assignment}) = q_\emptyset(G, -\mathbf{p})$.

Given G and $\mathbf{p}$, 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 $p_i \leqslant 2^{-d_i}$ (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: $\Pr(\text{perfect assignment}) = q_\emptyset(G, -\mathbf{p})$.

Given G and $\mathbf{p}$, 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,

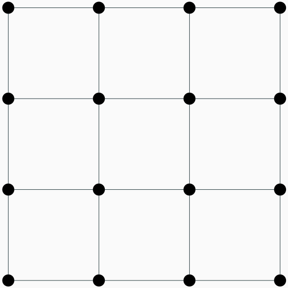
then we could use the uniform sampler (Moser-Tardos) to estimate $q_\emptyset$. With constant slack, Moser-Tardos runs in expected $O(n)$ time.

A simple construction exists if $p_i \leqslant 2^{-d_i}$ (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]

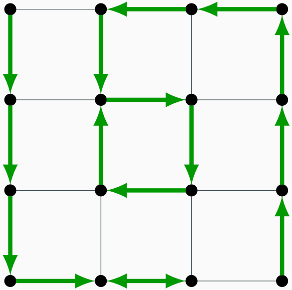
This approach does not work near Shearer's threshold. The situation is similar to the positive weight case, but for a different reason.

What else can we sample?



What else can we sample?

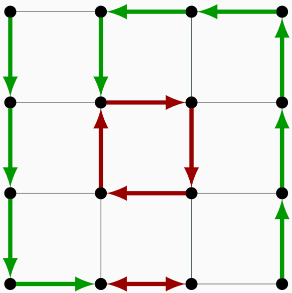
$\rightarrow$ **1.** For each v, assign a random arrow from v to one of its neighbours.



What else can we sample?

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

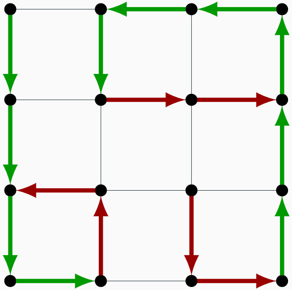
$\rightarrow$ 2. While there is a "small" cycle, resample all vertices along all cycles.



What else can we sample?

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

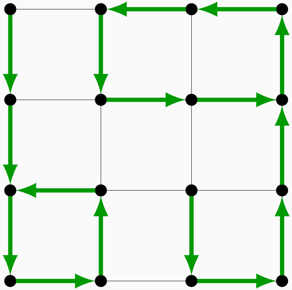
$\rightarrow$ 2. While there is a “small” cycle, resample all vertices along all cycles.



What else can we sample?

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

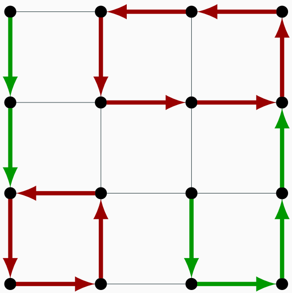
$\rightarrow$ 2. While there is a "small" cycle, resample all vertices along all cycles.



What else can we sample?

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

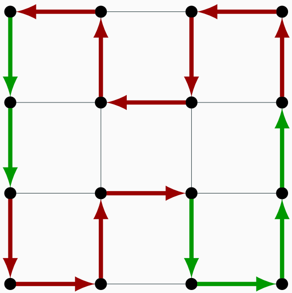
$\rightarrow$ 2. While there is a “small” cycle, resample all vertices along all cycles.



What else can we sample?

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

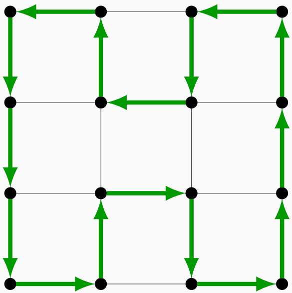
$\rightarrow$ 2. While there is a “small” cycle, resample all vertices along all cycles.



What else can we sample?

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

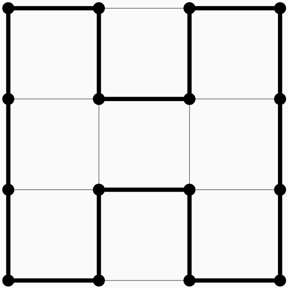
$\rightarrow$ 2. While there is a “small” cycle, resample all vertices along all cycles.



What else can we sample?

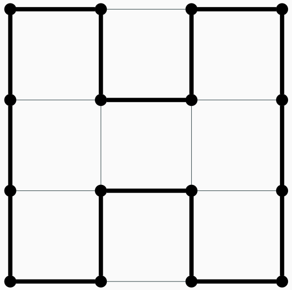
1. For each v, assign a random arrow from v to one of its neighbours.
2. While there is a "small" cycle, resample all vertices along all cycles.

$\rightarrow$ 3. Output.



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.

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 ($\delta = (1/2 + \varepsilon)n$), 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?

Beyond Extremal Instances

Partial Rejection Sampling

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

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

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

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

How to choose $\texttt{Resample}(\sigma)$ 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 σ — randomize all variables independently.
2. While σ is not perfect:
   choose an appropriate subset of events, `Resample`(σ);
   re-randomize all variables in `Resample`(σ).

For extremal instances, `Resample`(σ) is simply `Bad`(σ).

How to choose `Resample`(σ) to guarantee uniformity?

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

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

1. Initialize σ — randomize all variables independently.
2. While σ is not perfect:
   choose an appropriate subset of events, `Resample`(σ);
   re-randomize all variables in `Resample`(σ).

For extremal instances, `Resample`(σ) is simply `Bad`(σ).

How to choose `Resample`(σ) to guarantee uniformity?

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

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

1. Initialize σ — randomize all variables independently.
2. While σ is not perfect:
   choose an appropriate subset of events, `Resample`(σ);
   re-randomize all variables in `Resample`(σ).

For extremal instances, `Resample`(σ) is simply `Bad`(σ).

How to choose `Resample`(σ) 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 σ, 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 ∂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 σ, 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 ∂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 σ, 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 ∂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 σ, 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:

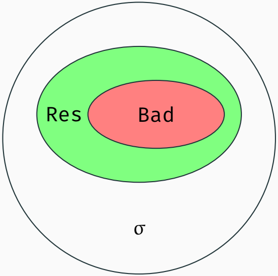
The set of all variables is unblocking.

For independent sets, S is unblocking if ∂S are all unoccupied.

Resampling set

Given an assignment σ, we want `Resample`(σ) to satisfy:

1. `Resample`(σ) contains `Bad`(σ);
2. `Resample`(σ) is unblocking;
3. What is revealed has to be resampled.



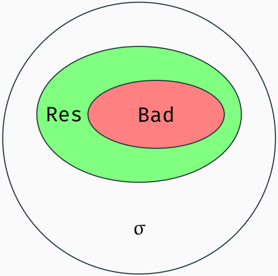
`Resample`(σ) can be found by a breadth-first search.

In the worst case we may resample all variables.

Resampling set

Given an assignment σ, we want Resample(σ) to satisfy:

1. Resample(σ) contains Bad(σ);
2. Resample(σ) is unblocking;
3. What is revealed has to be resampled.



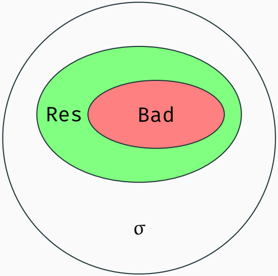
Resample(σ) can be found by a breadth-first search.

In the worst case we may resample all variables.

Resampling set

Given an assignment σ, we want `Resample`(σ) to satisfy:

1. `Resample`(σ) contains `Bad`(σ);
2. `Resample`(σ) is unblocking;
3. What is revealed has to be resampled.



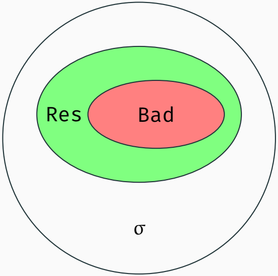
`Resample`(σ) can be found by a breadth-first search.

In the worst case we may resample all variables.

Resampling set

Given an assignment σ, we want `Resample`(σ) to satisfy:

1. `Resample`(σ) contains `Bad`(σ);
2. `Resample`(σ) is unblocking;
3. What is revealed has to be resampled.



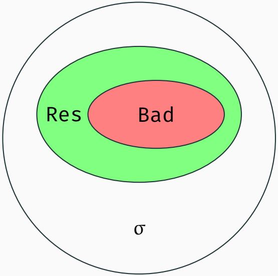
`Resample`(σ) can be found by a breadth-first search.

In the worst case we may resample all variables.

Resampling set

Given an assignment σ, we want Resample(σ) to satisfy:

1. Resample(σ) contains Bad(σ);
2. Resample(σ) is unblocking;
3. What is revealed has to be resampled.



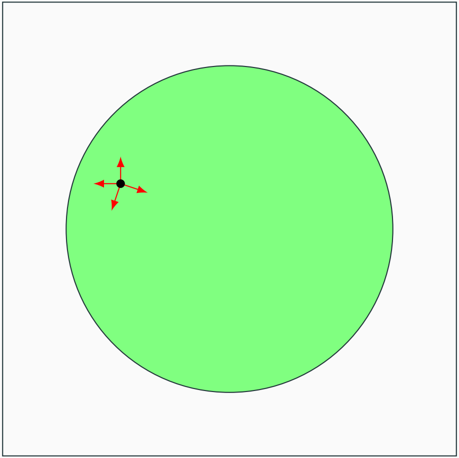
Resample(σ) can be found by a breadth-first search.

In the worst case we may resample all variables.

Partial Rejection Sampling vs Markov chains

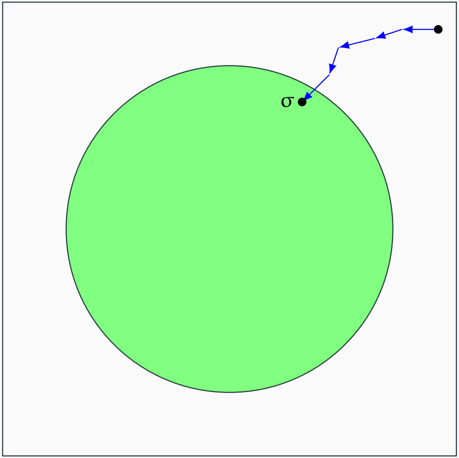
Markov chain is a random walk in the solution space.

(The solution space has to be connected!)



Partial Rejection Sampling vs Markov chains

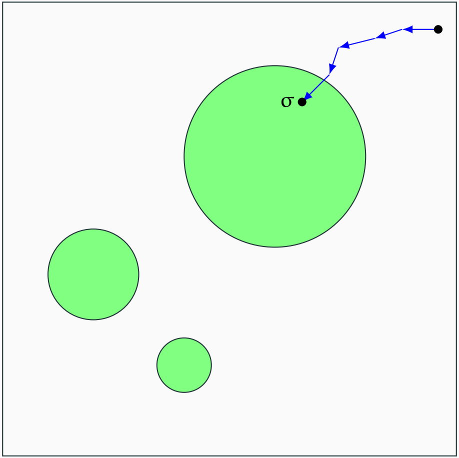
PRS is a local search on the whole space.



Partial Rejection Sampling vs Markov chains

PRS is a local search on the whole space.

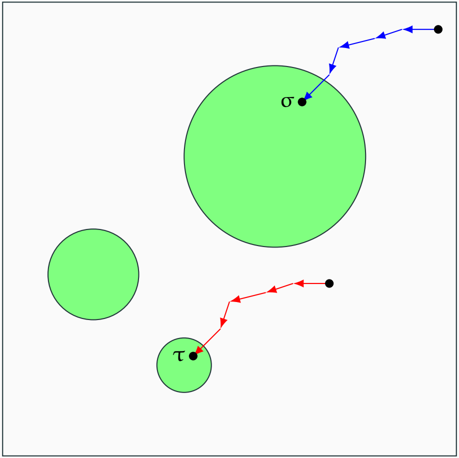
(Connectivity is not an issue.)



Partial Rejection Sampling vs Markov chains

PRS is a local search on the whole space.

(Uniformity is guaranteed by the bijection.)



Partial Rejection Sampling:
repeatedly resample the appropriately chosen `Resample`(σ).

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 `Resample`(σ).

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 `Resample`(σ).

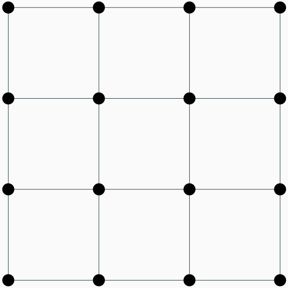
Theorem (G., Jerrum, Liu 17)
When PRS halts, its output is uniform.

Some applications beyond extremal instances:

- Weighted independent sets.
- k-CNF formulas.

Sampling independent sets

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



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

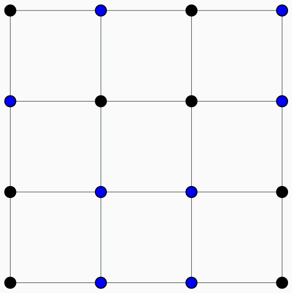
Sampling independent sets

$\rightarrow$ 1. Randomize each vertex.

2. Let `Bad` be the set of vertices whose connected component has size $\geqslant 2$.

3. `Resample` $=$ `Bad` $\cup\, \partial$`Bad`.

4. Resample `Resample`.
   Check independence.



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

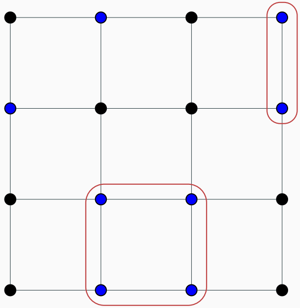
Sampling independent sets

1. Randomize each vertex.

$\rightarrow$ 2. Let **Bad** be the set of vertices whose connected component has size $\geqslant 2$.

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

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



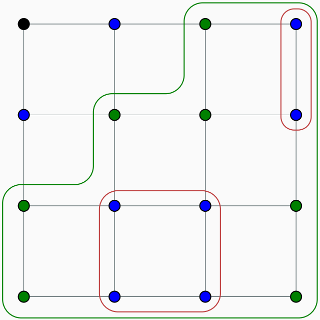
When the algorithm stops, it is a uniform independent set.

Sampling independent sets

1. Randomize each vertex.
2. Let Bad be the set of vertices whose connected component has size $\geqslant 2$.

$\rightarrow$ 3. Resample $=$ Bad $\cup\, \partial$Bad.

4. Resample Resample.
   Check independence.

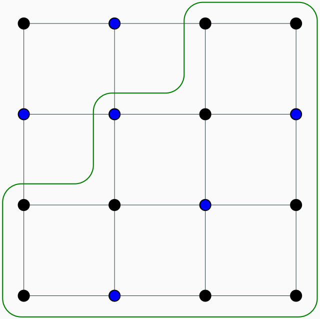


When the algorithm stops, it is a uniform independent set.

Sampling independent sets

1. Randomize each vertex.
2. Let $\texttt{Bad}$ be the set of vertices whose connected component has size $\geqslant 2$.
3. $\texttt{Resample} = \texttt{Bad} \cup \partial\texttt{Bad}$.

$\rightarrow$ 4. Resample $\texttt{Resample}$.
Check independence.

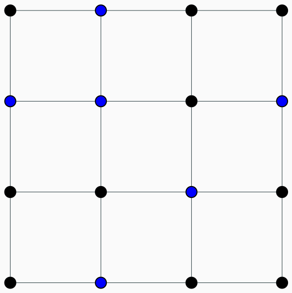


When the algorithm stops, it is a uniform independent set.

Sampling independent sets

1. Randomize each vertex.
2. Let $\texttt{Bad}$ be the set of vertices whose connected component has size $\geqslant 2$.
3. $\texttt{Resample} = \texttt{Bad} \cup \partial\texttt{Bad}$.

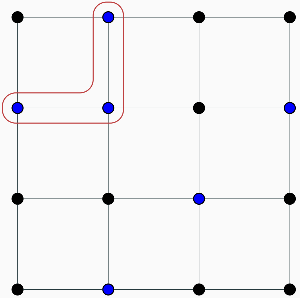
$\rightarrow$ 4. Resample $\texttt{Resample}$.
Check independence.



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

Sampling independent sets

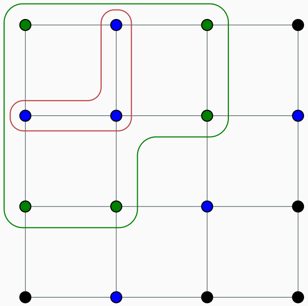
1. Randomize each vertex.
2. $\rightarrow$ Let $\texttt{Bad}$ be the set of vertices whose connected component has size $\geqslant 2$.
3. $\texttt{Resample} = \texttt{Bad} \cup \partial\texttt{Bad}$.
4. Resample $\texttt{Resample}$.
   Check independence.



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

Sampling independent sets

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

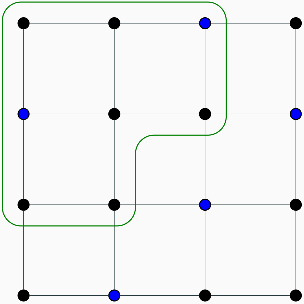


When the algorithm stops, it is a uniform independent set.

Sampling independent sets

1. Randomize each vertex.
2. Let $\texttt{Bad}$ be the set of vertices whose connected component has size $\geqslant 2$.
3. $\texttt{Resample} = \texttt{Bad} \cup \partial\texttt{Bad}$.

$\rightarrow$ 4. Resample $\texttt{Resample}$.
Check independence.

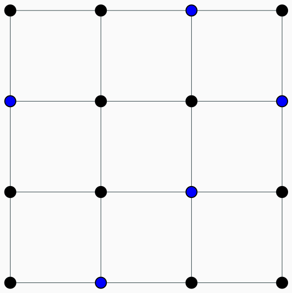


When the algorithm stops, it is a uniform independent set.

Sampling independent sets

1. Randomize each vertex.
2. Let Bad be the set of vertices whose connected component has size $\geqslant 2$.
3. $\texttt{Resample} = \texttt{Bad} \cup \partial\texttt{Bad}$.

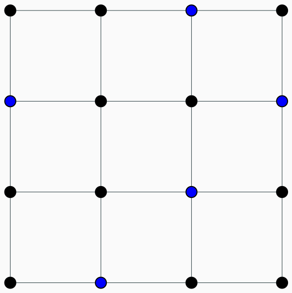
$\rightarrow$ 4. Resample $\texttt{Resample}$.
Check independence.



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

Sampling independent sets

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



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

Running time — independent sets

Set-up

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

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

Then $\mathbb{E}\,|\mathtt{Bad}_{t+1}| \leqslant ep\Delta \cdot k$

Set-up

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

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

Then $\mathbb{E}\,|\texttt{Bad}_{t+1}| \leqslant ep\Delta \cdot k$

1. Both $\texttt{Resample}_t$ and $\partial\texttt{Resample}_t$ are "dangerous", and $|\partial\texttt{Resample}_t| \leqslant \Delta \cdot k$.
2. Under LLL condition, for any event E,
$$\Pr(E \mid \bigwedge \overline{A_i}) \leqslant e \Pr(E).$$

Set-up

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

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

Then $\mathbb{E}\,|\texttt{Bad}_{t+1}| \leqslant ep\Delta \cdot k \quad \Rightarrow \quad \mathbb{E}\left|\texttt{Resample}_{t+1}\right| \leqslant ep\Delta^2 \cdot k$.

Running time — independent sets

Set-up

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

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

Then $\mathbb{E}\,|\texttt{Bad}_{t+1}| \leqslant ep\Delta \cdot k \quad \Rightarrow \quad \mathbb{E}\left|\texttt{Resample}_{t+1}\right| \leqslant ep\Delta^2 \cdot k$.

The resampling region shrinks if

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

(Recall that the local lemma requires $ep\Delta \leqslant 1$.)

Phase transition of independent sets

Sampling independent sets with weight λ 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 $\lambda \leqslant \frac{1}{2\sqrt{e}d-1}$.

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 λ 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 $\lambda \leqslant \frac{1}{2\sqrt{e}d-1}$.

The range is off by a constant, but it is fast, simple, exact, and distributed.

Running time — general case

$\exists$ constant C s.t. if $p\Delta^2 \geqslant C$, then even approximate sampling is **NP**-hard. Hence we have to assume stronger conditions than $ep\Delta \leqslant 1$.

Indenependent sets are nice in that `Resample` is just `Bad` $\cup\, \partial$`Bad`. In general, `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 $ep\Delta^2 \leqslant 1/6$ and $er\Delta \leqslant 1/3$, 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 |`Resample`|.

The condition on r is necessary.

Running time — general case

$\exists$ constant C s.t. if $p\Delta^2 \geqslant C$, then even approximate sampling is **NP**-hard. Hence we have to assume stronger conditions than $ep\Delta \leqslant 1$.

Indenepependent sets are nice in that `Resample` is just `Bad` $\cup\, \partial$`Bad`. In general, `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 $ep\Delta^2 \leqslant 1/6$ and $er\Delta \leqslant 1/3$, 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 $|$`Resample`$|$.

The condition on r is necessary.

Running time — general case

$\exists$ constant C s.t. if $p\Delta^2 \geqslant C$, then even approximate sampling is **NP**-hard. Hence we have to assume stronger conditions than $ep\Delta \leqslant 1$.

Indenependent sets are nice in that `Resample` is just `Bad` $\cup\, \partial$`Bad`. In general, `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 $ep\Delta^2 \leqslant 1/6$ and $er\Delta \leqslant 1/3$, 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 |`Resample`|.

The condition on r is necessary.

Running time — general case

$\exists$ constant C s.t. if $p\Delta^2 \geqslant C$, then even approximate sampling is **NP**-hard. Hence we have to assume stronger conditions than $ep\Delta \leqslant 1$.

Indenependent sets are nice in that `Resample` is just `Bad` $\cup\, \partial$`Bad`. In general, `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 $ep\Delta^2 \leqslant 1/6$ and $er\Delta \leqslant 1/3$, 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 |`Resample`|.

The condition on r is necessary.

Sampling k-CNF

NP-Hardness for sampling:

$d \geqslant 3$ — decision hardness for general formula
$d \geqslant 6, k = 2$ (monotone formula) [Sly 10]
$d \geqslant 5 \cdot 2^{k/2}$ (monotone formula) [Bezáková, Galanis, Goldberg, G., Štefankovič 16]
(LLL condition is $d \leqslant \frac{2^k}{ek}$.)

Theorem (G., Jerrum, Liu 17)

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

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

NP-Hardness for sampling:

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

Theorem (G., Jerrum, Liu 17)

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

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

NP-Hardness for sampling:

$d \geqslant 3$ — decision hardness for general formula

$d \geqslant 6, k = 2$ (monotone formula) [Sly 10]

$d \geqslant 5 \cdot 2^{k/2}$ (monotone formula) [Bezáková, Galanis, Goldberg, G., Štefankovič 16]

(LLL condition is $d \leqslant \frac{2^k}{ek}$.)

Theorem (G., Jerrum, Liu 17)

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

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

Sampling k-CNF

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

Ref.	Condition	Restriction	Method
[Bubley, Dyer 97]	$d = 2$		Markov chain
[Bordewich, Dyer, Karpinski 06]	$d \leqslant k - 2$	monotone	Markov chain
[Liu, Lu 15]	$d \leqslant 5$	monotone	Correlation decay
[BGGGŠ 16]	$d = 6, k = 3$ or $d \leqslant k$	monotone	Correlation decay
[Hermon, Sly, Zhang 17]	$d \leqslant c2^{k/2}$	monotone	Markov chain
[Moitra 17]	$d \leqslant \widetilde{O}(2^{k/60})$		Correlation decay + LP
[G., Jerrum, Liu 17]	$d \leqslant c2^{k/2}$	Intersection $\geqslant \min\{\log dk, k/2\}$	PRS

All other methods are approximate, whereas PRS is exact.

Sampling k-CNF

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

Ref.	Condition	Restriction	Method
[Bubley, Dyer 97]	$d = 2$		Markov chain
[Bordewich, Dyer, Karpinski 06]	$d \leqslant k - 2$	monotone	Markov chain
[Liu, Lu 15]	$d \leqslant 5$	monotone	Correlation decay
[BGGGŠ 16]	$d = 6, k = 3$ or $d \leqslant k$	monotone	Correlation decay
[Hermon, Sly, Zhang 17]	$d \leqslant c2^{k/2}$	monotone	Markov chain
[Moitra 17]	$d \leqslant \widetilde{O}(2^{k/60})$		Correlation decay + LP
[G., Jerrum, Liu 17]	$d \leqslant c2^{k/2}$	Intersection $\geqslant \min\{\log dk, k/2\}$	PRS

All other methods are approximate, whereas PRS is exact.

Sampling k-CNF

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

Ref.	Condition	Restriction	Method
[Bubley, Dyer 97]	$d = 2$		Markov chain
[Bordewich, Dyer, Karpinski 06]	$d \leqslant k - 2$	monotone	Markov chain
[Liu, Lu 15]	$d \leqslant 5$	monotone	Correlation decay
[BGGGŠ 16]	$d = 6, k = 3$ or $d \leqslant k$	monotone	Correlation decay
[Hermon, Sly, Zhang 17]	$d \leqslant c2^{k/2}$	monotone	Markov chain
[Moitra 17]	$d \leqslant \widetilde{O}(2^{k/60})$		Correlation decay + LP
[G., Jerrum, Liu 17]	$d \leqslant c2^{k/2}$	Intersection $\geqslant \min\{\log dk, k/2\}$	PRS

All other methods are approximate, whereas PRS is exact.

Sampling k-CNF

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

Ref.	Condition	Restriction	Method
[Bubley, Dyer 97]	$d = 2$		Markov chain
[Bordewich, Dyer, Karpinski 06]	$d \leqslant k - 2$	monotone	Markov chain
[Liu, Lu 15]	$d \leqslant 5$	monotone	Correlation decay
[BGGGŠ 16]	$d = 6, k = 3$ or $d \leqslant k$	monotone	Correlation decay
[Hermon, Sly, Zhang 17]	$d \leqslant c2^{k/2}$	monotone	Markov chain
[Moitra 17]	$d \leqslant \widetilde{O}(2^{k/60})$		Correlation decay + LP
[G., Jerrum, Liu 17]	$d \leqslant c2^{k/2}$	Intersection $\geqslant \min\{\log dk, k/2\}$	PRS

All other methods are approximate, whereas PRS is exact.

Sampling k-CNF

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

Ref.	Condition	Restriction	Method
[Bubley, Dyer 97]	$d = 2$		Markov chain
[Bordewich, Dyer, Karpinski 06]	$d \leqslant k - 2$	monotone	Markov chain
[Liu, Lu 15]	$d \leqslant 5$	monotone	Correlation decay
[BGGGŠ 16]	$d = 6, k = 3$ or $d \leqslant k$	monotone	Correlation decay
[Hermon, Sly, Zhang 17]	$d \leqslant c2^{k/2}$	monotone	Markov chain
[Moitra 17]	$d \leqslant \widetilde{O}(2^{k/60})$		Correlation decay + LP
[G., Jerrum, Liu 17]	$d \leqslant c2^{k/2}$	Intersection $\geqslant \min\{\log dk, k/2\}$	PRS

All other methods are approximate, whereas PRS is exact.

Concluding remarks

Summary

- 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 $r\Delta = O(1)$.

Summary

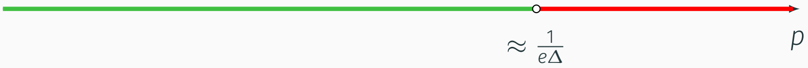
- For extremal instances, Moser-Tardos is uniform, with expected running time $\frac{\text{\# ``near-perfect'' assignments}}{\text{\# ``perfect'' assignments}}$.
- For general instances, we need to carefully choose a resampling set to ensure uniformity.
- The expected running time is linear if $p\Delta^2 = O(1)$ and $r\Delta = O(1)$.

Summary

- 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 [Erdős, Lovász 75]



Sampling threshold under LLL?

Sampling threshold under LLL?

Sampling threshold?

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!