

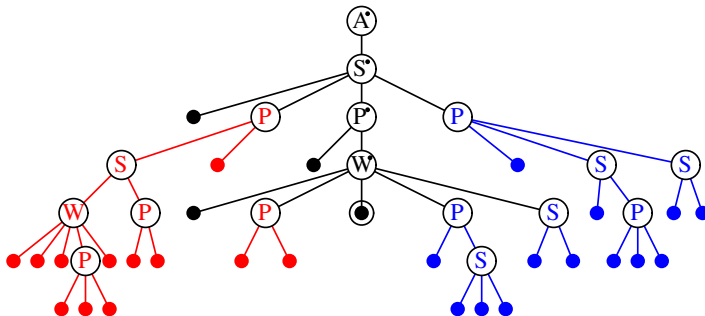
The challenge of linear-time Boltzmann sampling



Andrea Sportiello

CNRS and LIPN, Université Paris 13, Villetaneuse

AofA 2019, CIRM Luminy, June 24th 2019



Algorithms that work better when the size is not fixed...

Let $\mathcal{C} = \bigcup_N \mathcal{C}_N$ be a family of combinatorial structures,
where $C \in \mathcal{C}_N$ if its size $|C|$ is N .

Every \mathcal{C}_N is equipped with a measure μ_N (often just the uniform measure)

Your goal is to devise **algorithms** for **exactly sampling** from μ_N ,
which have the best possible **complexity in N**

It is often the case that you have a “**natural algorithm**” for sampling
from $\mu_{[\alpha]} := \sum_N \alpha_N \mu_N$ (with $\alpha_N \geq 0$ and $\sum_N \alpha_N = 1$)
and you are tempted to use the obvious algorithm

```
repeat
|  C ←  $\mu_{[\alpha]}$ 
until  $|C| = N$ ;
return C
```

Algorithms that work better when the size is not fixed...

Let $\mathcal{C} = \bigcup_N \mathcal{C}_N$ be a family of combinatorial structures,
where $C \in \mathcal{C}_N$ if its size $|C|$ is N .

Every \mathcal{C}_N is equipped with a measure μ_N (often just the uniform measure)

Your goal is to devise **algorithms** for **exactly sampling** from μ_N ,
which have the best possible **complexity in N**

It is often the case that you have a “**natural algorithm**” for sampling
from $\mu_{[\alpha]} := \sum_N \alpha_N \mu_N$ (with $\alpha_N \geq 0$ and $\sum_N \alpha_N = 1$)
and you are tempted to use the obvious algorithm

repeat


 | $C \leftarrow \mu_{[\alpha]}$

until $|C| = N$;

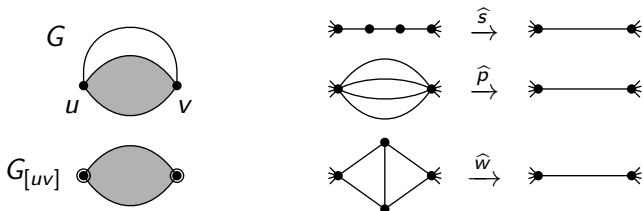
return C

Call this the “**Boltzmann case**”

Example: Boltzmann sampling of 2-terminal series-parallel graphs,
 or, more generally, of recursive minor-closed families of graphs.

Here we present the case of W_4 -free graphs,
 that is, graphs that do not contain  as a minor.

A graph G is in this class iff \exists (or \forall) edge $(uv) \in G$
 the 2-terminal graph $G_{[uv]}$ (rooted at u and v) can be reduced
 to one edge by series, parallel and 'wheatstone bridge' reductions:



In this case $\alpha_n = A(z)^{-1}[\zeta^n]A(\zeta)$, for $A(z)$ solving a certain
 equation, and $z < z^*$ root of a certain polynomial...
 (more details later on)

Algorithms that work better when the size is not fixed...

Now, let $\mathcal{C}_N = \bigcup_M \mathcal{C}_{N,M}$ be a family of combinatorial structures, where $C \in \mathcal{C}_{N,M}$ if $|C| = N$ and some statistics $m(C) \in \mathbb{Z}^d$ is equal to M .

Every $\mathcal{C}_{N,M}$ is equipped with a measure $\mu_{N,M}$

Again, you must devise **algorithms** for **exactly sampling** from $\mu_{N,M}$, which have the best possible **complexity in N**

You often have a **“natural algorithm”** for sampling from $\mu_{N,[\alpha]} := \sum_M \alpha_{N,M} \mu_{N,M}$ (with $\alpha_{N,M} \geq 0$ and $\sum_M \alpha_{N,M} = 1$ for all N) and you are tempted to use the obvious algorithm

```
repeat
  |  $C \leftarrow \mu_{N,[\alpha]}$ 
until  $m(C) = M$ ;
return  $C$ 
```

Algorithms that work better when the size is not fixed...

Now, let $\mathcal{C}_N = \bigcup_M \mathcal{C}_{N,M}$ be a family of combinatorial structures, where $C \in \mathcal{C}_{N,M}$ if $|C| = N$ and some statistics $m(C) \in \mathbb{Z}^d$ is equal to M .

Every $\mathcal{C}_{N,M}$ is equipped with a measure $\mu_{N,M}$

Again, you must devise **algorithms** for **exactly sampling** from $\mu_{N,M}$, which have the best possible **complexity in N**

You often have a **“natural algorithm”** for sampling from $\mu_{N,[\alpha]} := \sum_M \alpha_{N,M} \mu_{N,M}$ (with $\alpha_{N,M} \geq 0$ and $\sum_M \alpha_{N,M} = 1$ for all N) and you are tempted to use the obvious algorithm

repeat

 | $C \leftarrow \mu_{N,[\alpha]}$

until $m(C) = M$;

return C

Call this the “Bridge case”

Example: Area-weighted integer partitions in a $N \times M$ box,

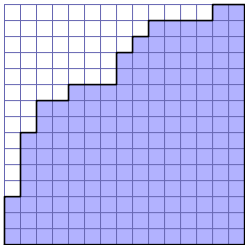
which can be represented as $(N + 1)$ -tuples (x_0, x_1, \dots, x_N) , associated to the vertical increments along the various columns:

$$\mu_{N,M}(x) \propto q^{\sum_k kx_k} \delta_{M, \sum_k x_k}$$

The natural problem, in which the δ -constraint is traded for a Lagrange multiplier ω , has measure

$$\mu_{N, [\alpha]}(x) \propto q^{\sum_k kx_k} \omega^{\sum_k x_k}$$

and the best you can do is tune ω in order to maximise $\alpha_{N,M}$.



Example: Area-weighted integer partitions in a $N \times M$ box,

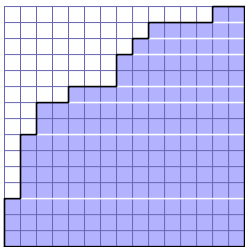
which can be represented as $(N + 1)$ -tuples (x_0, x_1, \dots, x_N) , associated to the vertical increments along the various columns:

$$\mu_{N,M}(x) \propto q^{\sum_k kx_k} \delta_{M, \sum_k x_k}$$

The natural problem, in which the δ -constraint is traded for a Lagrange multiplier ω , has measure

$$\mu_{N, [\alpha]}(x) \propto q^{\sum_k kx_k} \omega^{\sum_k x_k}$$

and the best you can do is tune ω in order to maximise $\alpha_{N,M}$.



$x = 3\ 4\ 2\ 0\ 1\ 0\ 0\ 2\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0$

In this case

$$\alpha_{N,M} = A(\omega)^{-1} [z^M] A(\omega z)$$

$$\text{with } A(\omega) = \prod_{k=0}^N \frac{1}{1 - \omega q^k}$$

Naïve complexity: Bridge case

Say that you are in the **Bridge case**,
and that sampling from $\mu_{N, [\alpha]}$ takes time $T_N \sim \tau N$.
Then of course the complexity is of order $T \sim \tau N / \alpha_{N, M}$.

Even in the “best realistic case” of Gaussian concentration of $m(C) \in \mathbb{Z}^d$ around M with variance (tensor spectrum) linear in N ,
this gives $T \sim N^{1+\frac{d}{2}}$

We would like to invent a better algorithm
in order to **kill the extra factor $N^{\frac{d}{2}}$** as much as we can.

This will be done, in a restricted family of models,
through the trick of **positive decomposition**,
which is in my “GASCom2018” paper.

Naïve complexity: Boltzmann case

Say that you are in the **Boltzmann case**,
that sampling from $\mu_{[\alpha]}$ takes time $T_n \sim \tau n$ when
the outcome $|C|$ is n , and $T_+ \sim \tau N$ when the sampling procedure
aborts with a certification that $|C|$ will be larger than N .
(this is achieved through **anticipated rejection**)

Then the complexity is of order $T \sim \frac{\tau N}{\alpha_N} \sum_n \alpha_n \min(n, N)$

Even in the “best realistic case” of $\alpha_n \sim z^n n^\gamma$, with $\gamma > -1$ and
 $z < 1$ that can be tuned at your will, this gives $T \sim N^2$

We would like to invent a better algorithm
in order to **kill the extra factor N** as much as we can.

This will “**almost**” be done, but only up to reach complexity $N^{\frac{3}{2}}$,
through the trick of **improved Hadamard product**,
which is in my “GASCom2016” paper.

Plan of the talk

It's now clear: I'm trying to 'sell' you **two recent tools** for improving the complexity of “**Boltzmann-like**” algorithms, that is, exact sampling algorithms which would be linear if it weren't for some size constraint, and are instead $T \sim N^{1+\gamma}$ because of the many repetitions necessary to get the desired size.

I propose you two main tools:

Boltzmann case: use the **improved Hadamard product** trick in wide generality, but the extra exponent only decreases to $\frac{\gamma}{2}$

Disclaimer, this is in part “under construction”!

Bridge case: in suitable circumstances, you can use the **positive decomposition** trick and remove the extra exponent

Before this, I want to discuss the **theoretical limit** to possible improvements, and a nice algorithm, due to Bacher, Bodini, Hollender and Lumbroso, that **can reach this limit** in a special case.

Shannon complexity bound

We have seen that the naïve algorithm has complexity $n^{1+\frac{d}{2}}$ in the Bridge case, and n^2 in the Boltzmann case.

This seems bad. But how bad exactly? How good can we possibly do?

Let us try to understand the **intrinsic minimal complexity** of a problem.

The **time complexity** is defined only up to a multiplicative constant, and with some degree of arbitrariness.

Instead, for the **random-bit complexity**, that is, the average number of random bits used for sampling an object of size N , also the overall constants do matter.

The intrinsic minimal random-bit complexity of an exact sampling problem is given by the **Shannon entropy** of the associated measure:

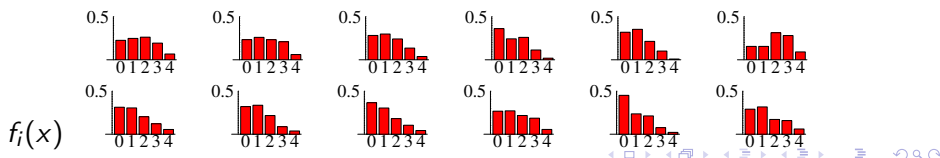
$$S_N = - \sum_{C \in \mathcal{C}_N} \mu_N(C) \ln \mu_N(C)$$

Shannon bound in the Bridge case

In this talk, we will only consider problems in the “Bridge case” of a special form:

$$\mathcal{C}_N = \{\mathbf{x}\} \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i,$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$$



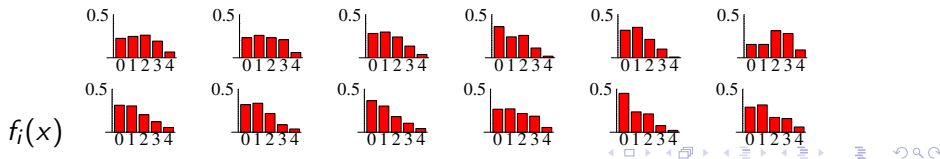
Shannon bound in the Bridge case

In this talk, we will only consider problems in the “Bridge case” of a special form:

$$\mathcal{C}_N = \{\mathbf{x}\} \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i, \quad \leftarrow \begin{array}{l} \text{random vector} \\ \text{of integers} \end{array}$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M} \quad \leftarrow \begin{array}{l} \text{NOT completely independent} \\ \text{(a single linear constraint)} \end{array}$$

↑
variables are NOT
identically distributed



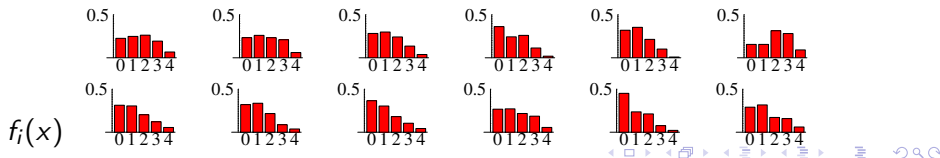
Shannon bound in the Bridge case

In this talk, we will only consider problems in the “Bridge case” of a special form:

$$\mathcal{C}_N = \{\mathbf{x}\} \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i, \quad \leftarrow \begin{array}{l} \text{random vector} \\ \text{of integers} \end{array}$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|, M} \quad \leftarrow \begin{array}{l} \text{completely independent:} \\ \text{the problem trivialises!} \end{array}$$

↑ variables are NOT identically distributed



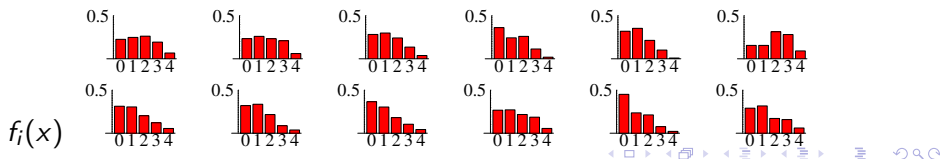
Shannon bound in the Bridge case

In this talk, we will only consider problems in the “Bridge case” of a special form:

$$\mathcal{C}_N = \{\mathbf{x}\} \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i, \quad \leftarrow \begin{array}{l} \text{random vector} \\ \text{of integers} \end{array}$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_{\times}(x_i) \times \delta_{|\mathbf{x}|,M} \quad \leftarrow \begin{array}{l} \text{NOT completely independent} \\ \text{(a single linear constraint)} \end{array}$$

↑
variables are identically distributed:
doable by using permutation symmetry [L. Devroye, 2012]



Shannon bound in the Bridge case

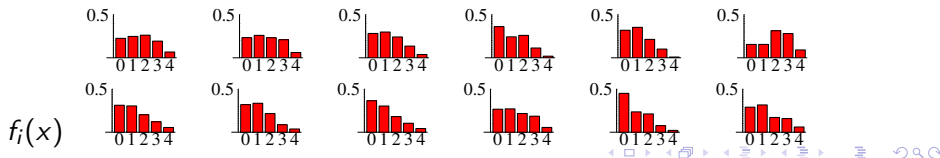
In this talk, we will only consider problems in the “Bridge case” of a special form:

$$\mathcal{C}_N = \{\mathbf{x}\} \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i, \quad \leftarrow \begin{array}{l} \text{random vector} \\ \text{of integers} \end{array}$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M} \quad \leftarrow \begin{array}{l} \text{NOT completely independent} \\ \text{(a single linear constraint)} \end{array}$$

↑ variables are NOT
identically distributed

We assume $\mathbb{E}(\sum_i x_i) = M$, so that $\mu_{N, [\alpha]}(\mathbf{x}) := \sum_M \mu_{N,M} \alpha_{N,M} = \prod_i f_i(x_i)$



Shannon bound in the Bridge case

So, what's the Shannon entropy of a measure

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}?$$

Simple fact 1: $S[\mu_{N,[\alpha]}] = \sum_i S[f_i] = \Theta(N)$

Simple fact 2:

$$\begin{aligned} S[\mu_{N,[\alpha]}] &= - \sum_m \sum_{\mathbf{x}:|\mathbf{x}|=m} \alpha_{N,m} \mu_{N,m}(\mathbf{x}) \ln(\alpha_{N,m} \mu_{N,m}(\mathbf{x})) \\ &= \mathbb{E}_{\alpha_N}(S(\mu_{N,m})) + S(\alpha_N) \end{aligned}$$

Note that, by CLT, $S(\alpha_N) = \Theta(\ln N)$.

Shannon bound in the Bridge case

So, what's the Shannon entropy of a measure

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}?$$

Simple fact 1: $S[\mu_{N,[\alpha]}] = \sum_i S[f_i] = \Theta(N)$

Simple fact 2:

$$\begin{aligned} S[\mu_{N,[\alpha]}] &= - \sum_m \sum_{\mathbf{x}:|\mathbf{x}|=m} \alpha_{N,m} \mu_{N,m}(\mathbf{x}) \ln(\alpha_{N,m} \mu_{N,m}(\mathbf{x})) \\ &= \mathbb{E}_{\alpha_N}(S(\mu_{N,m})) + S(\alpha_N) \end{aligned}$$

Note that, by CLT, $S(\alpha_N) = \Theta(\ln N)$.

Shannon bound in the Bridge case

So, what's the Shannon entropy of a measure

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}?$$

Simple fact 1: $S[\mu_{N,[\alpha]}] = \sum_i S[f_i] = \Theta(N)$

Simple fact 2:

$$\begin{aligned} S[\mu_{N,[\alpha]}] &= - \sum_m \sum_{\mathbf{x}:|\mathbf{x}|=m} \alpha_{N,m} \mu_{N,m}(\mathbf{x}) \ln(\alpha_{N,m} \mu_{N,m}(\mathbf{x})) \\ &= \mathbb{E}_{\alpha_N}(S(\mu_{N,m})) + S(\alpha_N) \end{aligned}$$

Note that, by CLT, $S(\alpha_N) = \Theta(\ln N)$.

Shannon bound in the Bridge case

A bit more subtle: By CLT, and considering a partition of the list (f_1, \dots, f_N) into two lists of size N_1 and $N_2 = N - N_1$, we have

$$\begin{aligned}\alpha_{N,m} S[\mu_{N,m}] &= - \sum_{m_1} \alpha_{N_1,m_1} \alpha_{N-N_1,m-m_1} \\ &\times \sum_{\substack{\mathbf{x}_1: |\mathbf{x}_1|=m_1 \\ \mathbf{x}_2: |\mathbf{x}_2|=m-m_1}} f_{N_1,m_1}(\mathbf{x}_1) f_{N-N_1,m-m_1}(\mathbf{x}_2) \ln (f_{N_1,m_1}(\mathbf{x}_1) f_{N-N_1,m-m_1}(\mathbf{x}_2)) \\ &= \sum_{m_1} \alpha_{N_1,m_1} \alpha_{N-N_1,m-m_1} (S[\mu_{N_1,m_1}] + S[\mu_{N-N_1,m-m_1}])\end{aligned}$$

so that the ansatz $S[f_{N,M+\delta}] - S[f_{N,M}] = \mathcal{O}(\delta^2/N)$
can be proven self-consistent.

These reasonings imply

$$S[f_{N,M}] = \left(\sum_i S[f_i] \right) \left(1 + \mathcal{O}\left(\frac{\ln N}{N}\right) \right)$$

Shannon bound in the Bridge case

A bit more subtle: By CLT, and considering a partition of the list (f_1, \dots, f_N) into two lists of size N_1 and $N_2 = N - N_1$, we have

$$\begin{aligned}\alpha_{N,m} S[\mu_{N,m}] &= - \sum_{m_1} \alpha_{N_1,m_1} \alpha_{N-N_1,m-m_1} \\ &\times \sum_{\substack{\mathbf{x}_1: |\mathbf{x}_1|=m_1 \\ \mathbf{x}_2: |\mathbf{x}_2|=m-m_1}} f_{N_1,m_1}(\mathbf{x}_1) f_{N-N_1,m-m_1}(\mathbf{x}_2) \ln (f_{N_1,m_1}(\mathbf{x}_1) f_{N-N_1,m-m_1}(\mathbf{x}_2)) \\ &= \sum_{m_1} \alpha_{N_1,m_1} \alpha_{N-N_1,m-m_1} (S[\mu_{N_1,m_1}] + S[\mu_{N-N_1,m-m_1}])\end{aligned}$$

so that the ansatz $S[f_{N,M+\delta}] - S[f_{N,M}] = \mathcal{O}(\delta^2/N)$ can be proven self-consistent.

These reasonings imply

$$S[f_{N,M}] = \left(\sum_i S[f_i] \right) \left(1 + \mathcal{O}\left(\frac{\ln N}{N}\right) \right)$$

Shannon bound in the Boltzmann case

In the Boltzmann case, the measure $\mu_{N,\alpha}$ is the one induced by the generating function $A_1(z)$, associated to the solution of a 'combinatorial specification' system

$$\begin{cases} A_1 = F_1(A_1, \dots, A_k, z) \\ \vdots \\ A_k = F_k(A_1, \dots, A_k, z) \end{cases}$$

From the positivity of the F_i 's coefficients, we can interpret $A_1(z)$ as a sum over suitable (coloured, weighted) trees, where the branching rules are described by the specification, and depend on z . Each tree corresponds to one configuration, exactly sampled from $\mu_{N,\alpha}$

As a result, any such measure corresponds to a **barely sub-critical Galton–Watson process**

Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

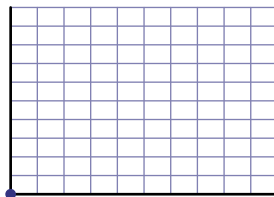
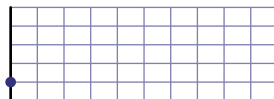
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

A



stack size: 1

obj. size: 0



Shannon bound in the Boltzmann case

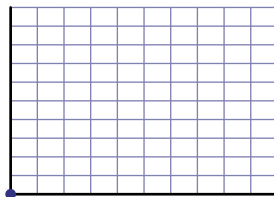
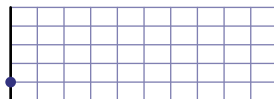
By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

A



stack size: 1 obj. size: 0

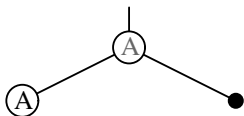


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

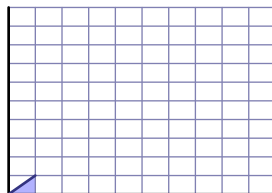
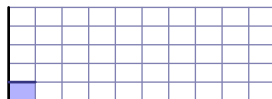
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

Az



stack size: 1

obj. size: 1

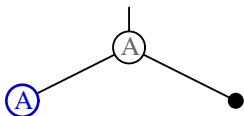


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

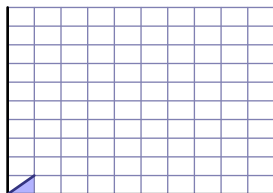
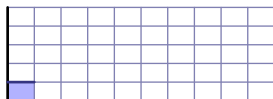
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

Az



stack size: 1

obj. size: 1

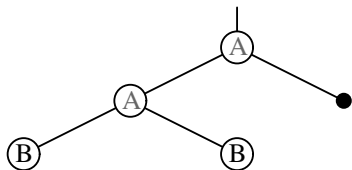


Shannon bound in the Boltzmann case

By randomising on the branching position,
any Galton–Watson process can be seen as a **rewriting system**,

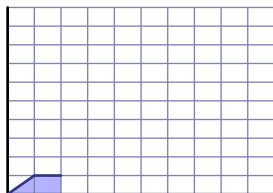
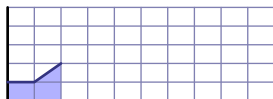
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

BBz



stack size: 2

obj. size: 1

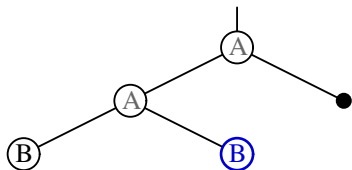


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

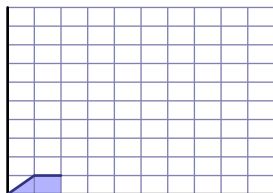
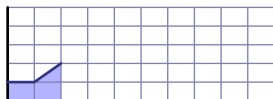
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

BBz



stack size: 2

obj. size: 1

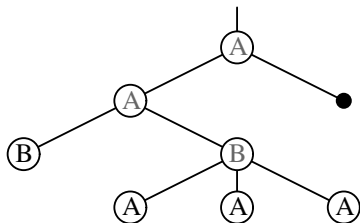


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

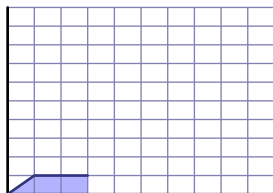
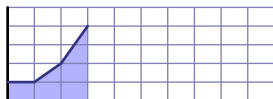
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

BAAAz



stack size: 4

obj. size: 1

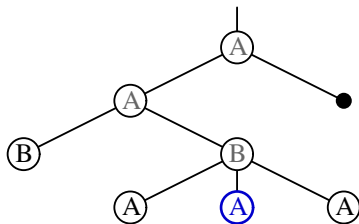


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

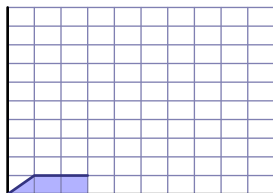
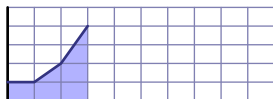
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BAAAz$



stack size: 4

obj. size: 1

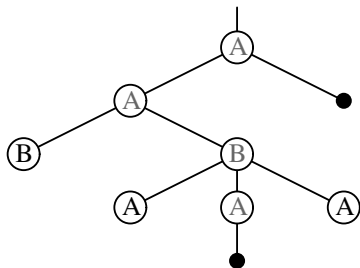


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

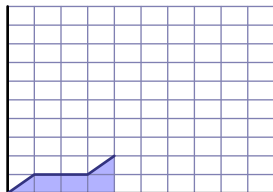
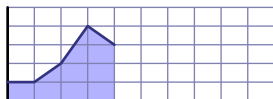
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BAzAz$



stack size: 3

obj. size: 2

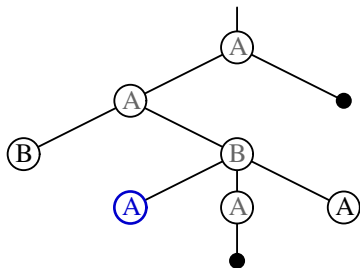


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

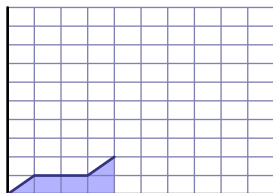
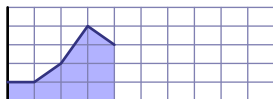
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BAzAz$



stack size: 3

obj. size: 2

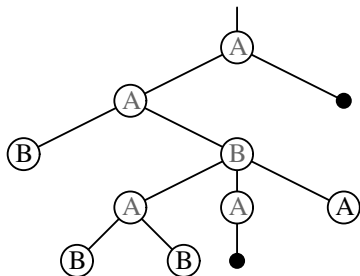


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

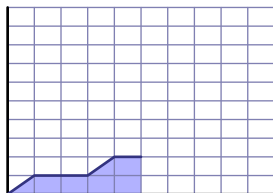
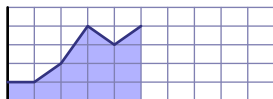
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

BBBzAz



stack size: 4

obj. size: 2

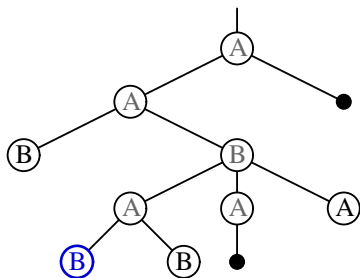


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

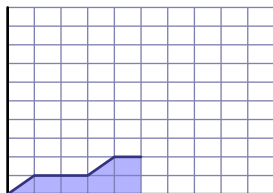
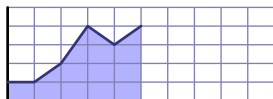
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BBzAz$



stack size: 4

obj. size: 2

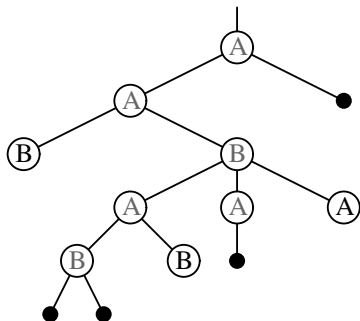


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

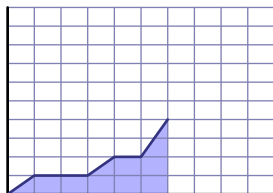
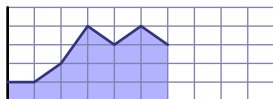
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BzzBzAz$



stack size: 3

obj. size: 4

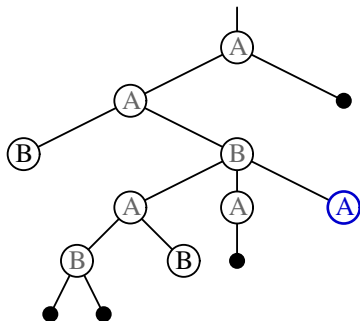


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

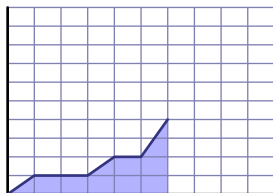
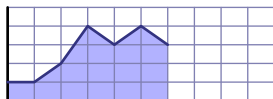
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BzzBzAz$



stack size: 3

obj. size: 4

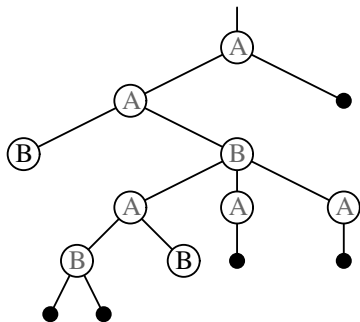


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

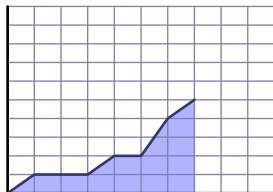
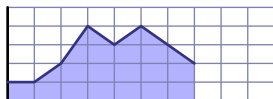
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

BzzBzzz



stack size: 2

obj. size: 5

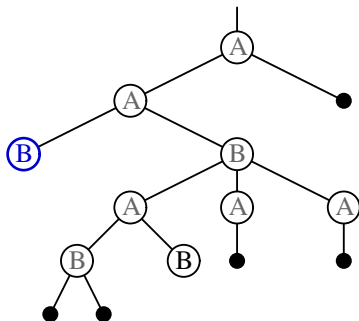


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

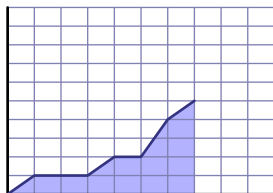
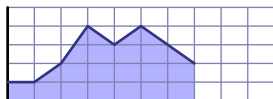
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

$BzzBzzz$



stack size: 2

obj. size: 5

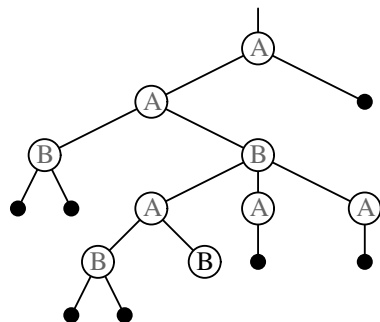


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

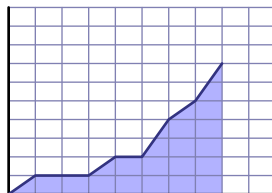
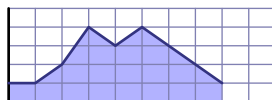
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

zzzzBzzz



stack size: 1

obj. size: 7

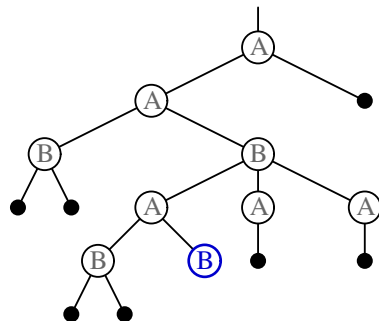


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

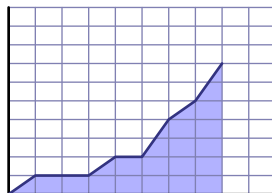
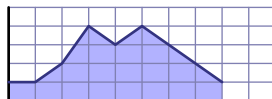
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

zzzzBzzz



stack size: 1

obj. size: 7

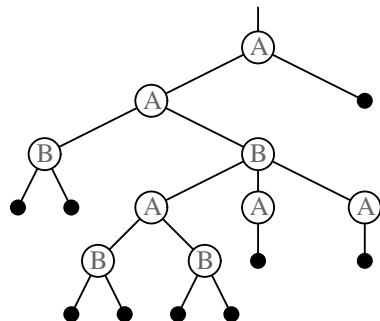


Shannon bound in the Boltzmann case

By randomising on the branching position, any Galton–Watson process can be seen as a **rewriting system**,

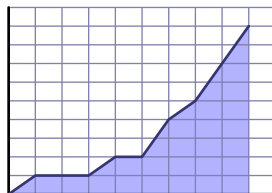
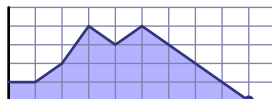
Example: for $\begin{cases} A = Az + B^2 + z \\ B = A^3 + z^2 \end{cases}$ we could get

zzzzzzzzzz



stack size: 0

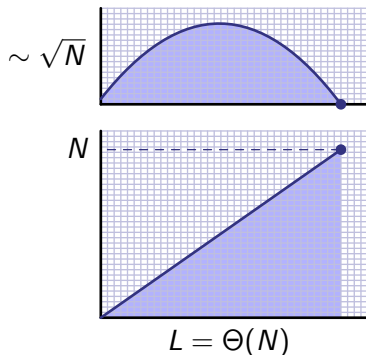
obj. size: 9



Shannon bound in the Boltzmann case

In the limit, on a successful run, the **stack size** profile is an **excursion**, while the **object size** profile is a **straight line**, both of length L

Call $n_1(t), \dots, n_k(t), s(t)$ the time evolution of the number of objects A_1, \dots, A_k in the stack, and of the object size.



Shannon bound in the Boltzmann case

In the limit, on a successful run, the **stack size** profile is an **excursion**, while the **object size** profile is a **straight line**, both of length L

Call $n_1(t), \dots, n_k(t), s(t)$ the time evolution of the number of objects A_1, \dots, A_k in the stack, and of the object size.

At linear order, these quantities satisfy equations of the form

$$\dot{n}_i = t^{-1} \sum_j n_j (-\delta_{ij} + \frac{A_i}{A_j} \frac{\partial}{\partial A_i} F_j) \Big|_{A_j \rightarrow A_j^*}^{z \rightarrow z^*} =: \sum_j M_{ij} n_j / t$$

M has a unique eigenvector (p_1, \dots, p_k) with all $p_i > 0$, $\sum_i p_i = 1$, and eigenvalue zero, all other eigenvalues are negative.

This is the eigenvalue/vector associated to the stack size excursion, which has height $\sim \sqrt{N}$, so it vanishes at linear order.

For the object size profile, we have

$$\dot{s} = t^{-1} \sum_j n_j \left(\frac{z}{A_j} \frac{\partial}{\partial z} F_j \right) \Big|_{A_j \rightarrow A_j^*}^{z \rightarrow z^*} \simeq \sum_j p_j \left(\frac{z}{A_j} \frac{\partial}{\partial z} F_j \right) \Big|_{A_j \rightarrow A_j^*}^{z \rightarrow z^*}$$

Shannon bound in the Boltzmann case

Note how the randomisation of the growth position has made the dynamics asymptotically homogeneous in time (that is, homogeneous up to $\mathcal{O}(N^{-\frac{1}{4}})$ fluctuations, when the stack size is far from the horizontal axis)

As a result, from the combinatorial specification, and the selection of the “good” critical point z^* , we can read the limit parameters p_i and \dot{s}

Any Galton–Watson branching on a type- A_i object corresponds to a combinatorial construction with an intrinsic Shannon entropy S_i

Just as in the Bridge case, having an excursion instead of a generic walk only reduces the entropy by a factor $\sim (\ln N)/N$

$$\begin{aligned} \text{This gives } L &= N/\dot{s} + \mathcal{O}(\sqrt{N}), \text{ and} \\ S &\simeq N \left(\frac{1}{\dot{s}} \sum_{i=1}^k p_i S_i \right) (1 + \mathcal{O}(N^{-\frac{1}{4}})), \\ &\text{which is our final result} \end{aligned}$$

Shannon complexity bound: summary

In conclusion, in our case of study, you have an **optimal** exact sampling algorithm if the time complexity is linear, and the random bit complexity $T_{\text{rand}}(N)$ is, up to corrections,

Bridge case: $\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$

$$T_{\text{rand}}(N, M) = \left(\sum_i S[f_i] \right) (1 + o(1))$$

Boltzmann case: $\{A_i = F_i(A_1, \dots, A_k, z)\}_{i=1, \dots, k}$

$$T_{\text{rand}}(N) = \left(\frac{N}{s} \sum_{i=1}^k p_i S_i \right) (1 + o(1))$$

I will not show any algorithm of mine that reaches optimality. But I will show you that optimality exists! By presenting you the “**mother of all** (Bridge-case exact sampling) **algorithms**”

Shannon complexity bound: summary

In conclusion, in our case of study, you have an **optimal** exact sampling algorithm if the time complexity is linear, and the random bit complexity $T_{\text{rand}}(N)$ is, up to corrections,

Bridge case: $\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$

$$T_{\text{rand}}(N, M) = \left(\sum_i S[f_i] \right) (1 + o(1))$$

Boltzmann case: $\{A_i = F_i(A_1, \dots, A_k, z)\}_{i=1, \dots, k}$

$$T_{\text{rand}}(N) = \left(\frac{N}{s} \sum_{i=1}^k p_i S_i \right) (1 + o(1))$$

I will not show any algorithm of mine that reaches optimality. But I will show you that optimality exists! By presenting you the “**mother of all** (Bridge-case exact sampling) **algorithms**”

BBHL algorithm: 'the mother of all algorithms'

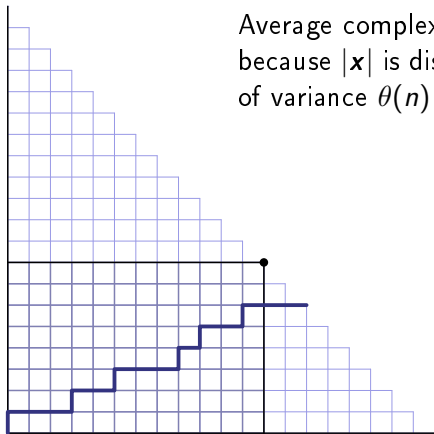
The following algorithm is hidden in a small corner of the paper
Bacher, Bodini, Hollender and Lumbroso,
MergeShuffle: A Very Fast, Parallel Random Permutation Algorithm
<https://arxiv.org/pdf/1508.03167>

The problem: exact sampling of strings in $\{\bullet, \circ\}^n$ with $\#\{\bullet\} = k$
BBHL solves it in linear time and optimal random-bit complexity.
(which is $T_{\text{rand}}(n) = n(-p \ln p - (1-p) \ln(1-p)) + o(n)$, with $p = \frac{k}{n}$)

BBHL algorithm: 'the mother of all algorithms'

First naïve idea: the “Bridge case” naïve approach. Sample n variables $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$, i.i.d. with Bern_p , restart if $|\mathbf{x}| \neq k$.

Average complexity: $\sim n^{\frac{3}{2}}$,
because $|\mathbf{x}|$ is distributed roughly as a Gaussian
of variance $\theta(n)$ and mean k .



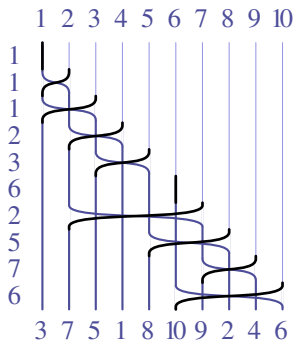
BBHL algorithm: 'the mother of all algorithms'

Second naïve idea: project down from Fisher–Yates

The Fisher–Yates algorithm samples a random permutation $\sigma \in \mathfrak{S}_n$ with optimal random-bit complexity: $T_{\text{rand}}(n) \simeq \ln n! \simeq n(\ln n - 1)$. It works by sampling $\mathbf{y} \in \{1\} \times \{1, 2\} \times \{1, 2, 3\} \times \dots \times \{1, \dots, n\}$, and doing as follows:

Then, 'projecting down' means $x_i = 1$ iff $\sigma^{-1}(i) \leq k$

Average complexity: $\sim n \ln n$, because, even if Fisher–Yates is optimal, the projection throws away most of the information



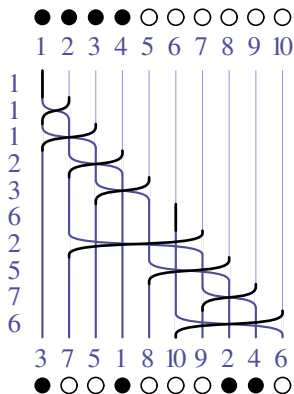
BBHL algorithm: 'the mother of all algorithms'

Second naïve idea: project down from Fisher–Yates

The Fisher–Yates algorithm samples a random permutation $\sigma \in \mathfrak{S}_n$ with optimal random-bit complexity: $T_{\text{rand}}(n) \simeq \ln n! \simeq n(\ln n - 1)$
It works by sampling $\mathbf{y} \in \{1\} \times \{1, 2\} \times \{1, 2, 3\} \times \dots \times \{1, \dots, n\}$, and doing as follows:

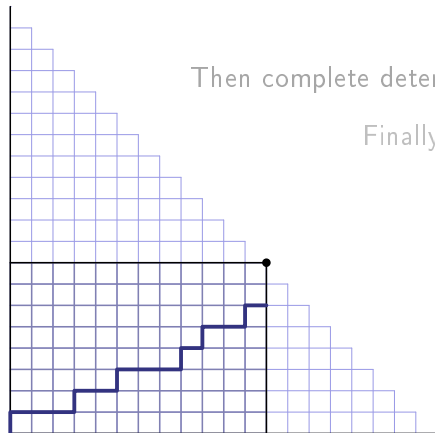
Then, 'projecting down' means
 $x_i = 1$ iff $\sigma^{-1}(i) \leq k$

Average complexity: $\sim n \ln n$,
because, even if Fisher–Yates is optimal, the projection throws away most of the information



BBHL algorithm: 'the mother of all algorithms'

The good idea: Sample the n variables $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$, i.i.d. with Bern_p , one by one up to when you have k entries $x_i = 1$, or $n - k$ entries $x_i = 0$.



Then complete deterministically with what is needed,

Finally, perform Fisher–Yates shufflings on these last added steps.

Average complexity:

$$T_{\text{rand}}(n) = S[\mu] + \mathcal{O}(\sqrt{n} \ln n)$$

because the final shuffles are just a few.

BBHL algorithm: 'the mother of all algorithms'

The good idea: Sample the n variables $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$, i.i.d. with Bern_p , one by one up to when you have k entries $x_i = 1$, or $n - k$ entries $x_i = 0$.

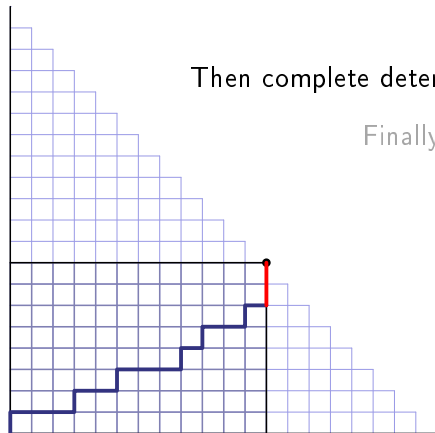
Then complete deterministically with what is needed,

Finally, perform Fisher–Yates shufflings on these last added steps.

Average complexity:

$$T_{\text{rand}}(n) = S[\mu] + \mathcal{O}(\sqrt{n} \ln n)$$

because the final shuffles are just a few.



BBHL algorithm: 'the mother of all algorithms'

The good idea: Sample the n variables $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$, i.i.d. with Bern_p , one by one up to when you have k entries $x_i = 1$, or $n - k$ entries $x_i = 0$.

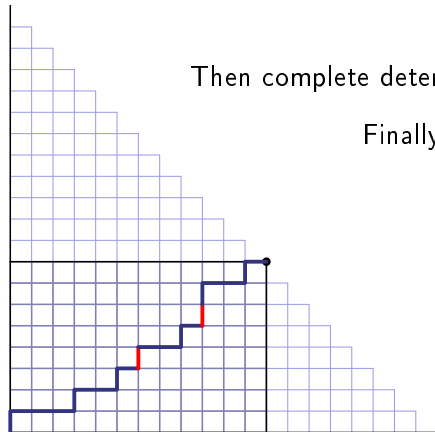
Then complete deterministically with what is needed,

Finally, perform Fisher–Yates shufflings on these last added steps.

Average complexity:

$$T_{\text{rand}}(n) = S[\mu] + \mathcal{O}(\sqrt{n} \ln n)$$

because the final shuffles are just a few.



BBHL algorithm: the code

Algorithm : BBHL shuffling algorithm

begin

$a = k, b = n - k, i = 0;$

repeat

$i++;$

$\nu_i \leftarrow \text{Bern}_\beta;$

if $\nu_i = 1$ **then** $a --$ **else** $b --$

until $a < 0$ **or** $b < 0$

complexity $\sim n;$

if $a < 0$ **then** $\bar{\nu} = 0$ **else** $\bar{\nu} = 1;$

for $j \leftarrow i$ **to** n **do**

$\nu_j = \bar{\nu};$

$h \leftarrow \text{RndInt}_j;$

swap ν_j and ν_h

complexity $\sim \sqrt{n} \ln n;$

return ν

end

Plan of the talk

I recall you that I'm trying to 'sell' you **two recent tools** for improving the complexity of “**Boltzmann-like**” algorithms, that is, exact sampling algorithms with complexity $T \sim N^{1+\gamma}$ because of the many repetitions necessary to get the desired size.

I propose you two main tools:

Boltzmann case: use the **improved Hadamard product** trick in wide generality, but the extra exponent only decreases to $\frac{\gamma}{2}$

Bridge case: in suitable circumstances, you can use the **positive decomposition** trick and remove the extra exponent

We are ready to go!

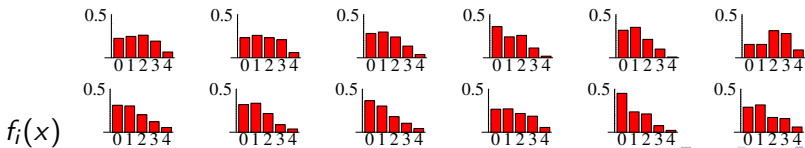
Let us start with the **Bridge case**
and the **positive decomposition** trick

Positive decomposition in one slide

$$\mathcal{C}_n = \{\mathbf{x}\}, \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$$

Problem: Assume that sampling from each distrib. f_i costs $\mathcal{O}(1)$. Find an algorithm that samples from the distribution $\mu_{N,M}$ in average linear time.

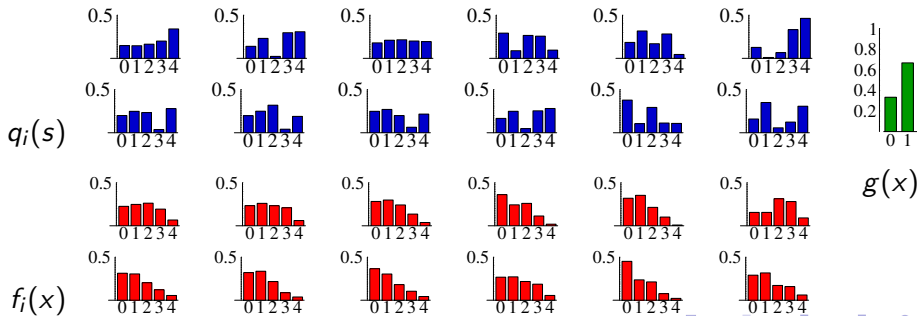


Positive decomposition in one slide

$$\mathcal{C}_n = \{\mathbf{x}\}, \quad \mathbf{x} = (x_1, \dots, x_N) \in \mathbb{N}^N, \quad |\mathbf{x}| := \sum_i x_i$$

$$\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$$

Our solution: **positive decomposition.** Assume that there exists $g(\mathbf{x}) \in \{\text{Bern}_b, \text{Pois}, \text{Geom}_b\}$, and $\{q_i(s)\}_{1 \leq i \leq n; s \in \mathbb{N}}$ real positive, such that $f_i(x) = \sum_s q_i(s) g^{*s}(\mathbf{x})$. Then our new algorithm does it!



Positive decomposition in two slides

Our new trick is based on the following ideas:

- ▶ Rejection algorithms have an extra factor in their complexity, on the scale of the inverse of the acceptance rate. In order to have the optimal complexity scaling, **you need the average acceptance rate to be $\Theta(1)$** , i.e. not to scale with the size n .
- ▶ **Positive decomposition** gives $f_i(x) = \sum_s q_i(s)g^{*s}(x)$.
As a result the measure $\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$ is a marginal of a measure **in two sets of variables**:
$$\mu_{N,M}(\mathbf{x}, \mathbf{s}) = \frac{1}{Z} \prod_{i=1}^N (q_i(s_i) g^{*s_i}(x_i)) \times \delta_{|\mathbf{x}|,M}.$$
- ▶ You can **first sample \mathbf{s}** , with measure $\mu_1(\mathbf{s}) = \prod_{i=1}^N q_i(s_i)$, then **accept this vector \mathbf{s}** with rate $a(\mathbf{s}) \propto g^{*|\mathbf{s}|}(M)$, and finally **sample \mathbf{x}** with measure $\mu_2(\mathbf{x} | \mathbf{s}) = \prod_{i=1}^N g^{*s_i}(x_i)$.
- ▶ **The acceptance rate is high because**, although $g^{*|\mathbf{s}|}(M) = \Theta(N^{-\frac{1}{2}})$, we have $g^{*|\mathbf{s}|}(M) / \max_N(g^{*N}(M)) = \Theta(1)$.

A reminder of the naïve algorithm in the Bridge case

Up to redefining the f_i 's, we can assume w.l.o.g. that $\mathbb{E}(|\mathbf{x}|) = \sum_i \mathbb{E}[f_i] = M$.
Assume that both M and $\sigma^2 := \sum_i \text{Var}[f_i]$ are $\Theta(N)$.

The 'Bridge case' rejection algorithm would give:

| | |
|---|----------------------------------|
| Algorithm : Naïve rejection sampling | <i>complexity</i> $\sim N^{3/2}$ |
|---|----------------------------------|

```
begin
  repeat
    |x| = 0;
    for i ← 1 to N do
      x_i ← f_i
      |x| += x_i;
    until |x| = M
    return (x_1, ..., x_N)
end
```

← complexity $\sim N$;

← complexity $\sim \sqrt{N}$;

The rejection paradigm

More generally, in any rejection algorithm,
you want to do exact sampling for $\mu(\mathbf{x})$,
when $\mu(\mathbf{x}) \propto \mu_0(\mathbf{x})a(\mathbf{x})$, with $a(\mathbf{x}) \in [0, 1]$,
supposing that you know how to sample from μ_0

Algorithm : Rejection sampling

$$T[\mu] \sim T[\mu_0] \mathbb{E}(a(\mathbf{x}))^{-1}$$

begin

repeat

$\mathbf{x} \leftarrow \mu_0$;

\leftarrow complexity $T[\mu_0]$

$\alpha \leftarrow \text{Bern}_{a(\mathbf{x})}$;

until $\alpha = 1$

\leftarrow complexity $\mathbb{E}(a(\mathbf{x}))^{-1}$;

return \mathbf{x}

end

The rejection paradigm for decomposed measures

Now assume $\mu(\mathbf{x}) \propto \sum_{\mathbf{y}} \mu_1(\mathbf{y}) \mu_2(\mathbf{x} | \mathbf{y}) a(\mathbf{y})$, with $a(\mathbf{y}) \in [0, 1]$, supposing that you know how to sample from μ_1 , and $\mu_2(\cdot | \mathbf{y})$

Algorithm : Rejection sampling for decomposed measures

begin

 repeat

$\mathbf{y} \leftarrow \mu_1$;

 ← sample a tentative \mathbf{y} with μ_1

$\alpha \leftarrow \text{Bern}_{a(\mathbf{y})}$

 until $\alpha = 1$

 ← accept \mathbf{y} with rate $a(\mathbf{y})$;

$\mathbf{x} \leftarrow \mu_2(\cdot | \mathbf{y})$;

 ← sample \mathbf{x} with $\mu_2(\cdot | \mathbf{y})$

 return \mathbf{x}

end

$$T = \frac{\sum_{\mathbf{y}} \mu_1(\mathbf{y}) (T_1(\mathbf{y}) + a(\mathbf{y}) T_2(\mathbf{y}))}{\sum_{\mathbf{y}} \mu_1(\mathbf{y}) a(\mathbf{y})} = \frac{\mathbb{E}(T_1 + a T_2)}{\mathbb{E}(a)} \leq \frac{T_1^{\max}}{\mathbb{E}(a)} + T_2^{\max}$$

The rejection paradigm for decomposed measures

Now assume $\mu(\mathbf{x}) \propto \sum_{\mathbf{y}} \mu_1(\mathbf{y}) \mu_2(\mathbf{x} | \mathbf{y}) a(\mathbf{y})$, with $a(\mathbf{y}) \in [0, 1]$, supposing that you know how to sample from μ_1 , and $\mu_2(\cdot | \mathbf{y})$

Algorithm : Rejection sampling for decomposed measures

begin

 repeat

$\mathbf{y} \leftarrow \mu_1$; \leftarrow complexity $T_1(\mathbf{y})$

$\alpha \leftarrow \text{Bern}_{a(\mathbf{y})}$

 until $\alpha = 1$ \leftarrow complexity $a(\mathbf{y})^{-1}$;

$\mathbf{x} \leftarrow \mu_2(\cdot | \mathbf{y})$; \leftarrow complexity $T_2(\mathbf{y})$

 return \mathbf{x}

end

$$T = \frac{\sum_{\mathbf{y}} \mu_1(\mathbf{y}) (T_1(\mathbf{y}) + a(\mathbf{y}) T_2(\mathbf{y}))}{\sum_{\mathbf{y}} \mu_1(\mathbf{y}) a(\mathbf{y})} = \frac{\mathbb{E}(T_1 + a T_2)}{\mathbb{E}(a)} \leq \frac{T_1^{\max}}{\mathbb{E}(a)} + T_2^{\max}$$

Positive decomposition provides a decomposed measure

Positive decomposition tells that, for all i ,
$$f_i(x) = \sum_s q_i(s) g^{*s}(x), \text{ with } q_i(s) \geq 0.$$

From the normalisation of the f_i 's and of g ,
it follows that also the $q_i(s)$ are probability distributions.

As a result the measure $\mu_{N,M}(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^N f_i(x_i) \times \delta_{|\mathbf{x}|,M}$
is a marginal of a measure in two sets of variables:

$$\mu_{N,M}(\mathbf{x}, \mathbf{s}) = \frac{1}{Z} \prod_{i=1}^N (q_i(s_i) g^{*s_i}(x_i)) \times \delta_{|\mathbf{x}|,M}.$$

This is exactly as in a decomposed measure, with correspondence

| | | | |
|---------------------|---|--|--|
| sample \mathbf{s} | with measure $\mu_1(\mathbf{s})$ | | $\mu_1(\mathbf{s}) = \prod_{i=1}^n q_i(s_i)$ |
| accept \mathbf{s} | with rate $a(\mathbf{s})$ | | $a(\mathbf{s}) \propto g^{* \mathbf{s} }(m)$ |
| sample \mathbf{x} | with measure $\mu_2(\mathbf{x} \mathbf{s})$ | | $\mu_2(\mathbf{x} \mathbf{s}) = \prod_{i=1}^n g^{*s_i}(x_i)$ |

Note: although μ_1 and μ_2 depend on the vector \mathbf{s} ,
the rate a only depends on $|\mathbf{s}| = \sum_i s_i$.

Increasing the acceptance rate

The crucial point is that the decomposition allows to **increase the acceptance rate!**

In the 'ordinary' rejection scheme, you accept \mathbf{x} iff a probabilistic event occurs (in our case, $|\mathbf{x}| = M$). If this probability is **intrinsically small** (in our case, $\Theta(N^{-1/2})$), there is nothing you can do.

In the rejection scheme for decomposed measures, the rate $a(\mathbf{s})$ is defined **up to a multiplicative factor**, as long as $\max_{\mathbf{s}} a(\mathbf{s}) \leq 1$.

Here, the obvious choice for $a(\mathbf{s})$ is $a(\mathbf{s}) = g^{*|\mathbf{s}|}(M)$, which is $\Theta(N^{-\frac{1}{2}})$.

However, we can push it up to $a(\mathbf{s}) = \frac{g^{*|\mathbf{s}|}(M)}{\max_n(g^{*n}(M))}$.

As we will see, with this choice $\mathbb{E}(a(\mathbf{s})) = \Theta(1)$.

How to sample from $\text{Bern}_{a(\mathbf{s})}$

This idea is not sufficient by itself. Even if you know in advance that, after maximisation, $\mathbb{E}(a(\mathbf{s})) = \Theta(1)$, you still have a problem:

sampling a Bernoulli rnd var with parameter $a(\mathbf{s})$ is difficult if you do not have an analytic expression for $a(\mathbf{s})$.

It is not compulsory to have an analytic expression for $a(\mathbf{s})$ (just think to how the Monte Carlo algorithm:
 $x \leftarrow \text{Rnd}[0, 1]; y \leftarrow \text{Rnd}[0, 1];$ **return** $\text{sign}(1 - x^2 - y^2)$
samples $\text{Bern}_{\pi/4}$ without knowing $\pi \dots$)

however, it makes life easier, and in our case **we have it for free** if we choose the base function $g(x)$ for positive decomposition in the list $g(x) \in \{\text{Bern}_b, \text{Poiss}, \text{Geom}_b\}$

How to sample from $\text{Bern}_{a(s)}$

Example with Bernoulli (the other cases are similar)
(just write $a(s)$ for $a(\mathbf{s})$, with $s = |\mathbf{s}|$)

$$a(s) = \frac{g^{*s}(M)}{\max_n (g^{*n}(M))} = \frac{b^M (1-b)^{s-M} \binom{s}{M}}{\max_n (b^M (1-b)^{n-M} \binom{n}{M})}$$

The max is realised for $n = \bar{n} := \lfloor M/b \rfloor$, thus

$$a(s) = (1-b)^{s-\bar{n}} \frac{s! (\bar{n} - M)!}{\bar{n}! (s - M)!}$$

Good news 1: This is easily evaluated to high precision (i.e., calculating d binary digits has complexity $\ll 2^d$), so that the average cost of $\text{Bern}_{a(s)}$ is $\Theta(1)$.

Good news 2: For large M , and $b = \Theta(1)$, $a(s)$ converges to an un-normalised Gaussian centered around \bar{n} , and of variance $\Theta(M)$.

A rough evaluation of the complexity

Recall the basic steps in the rejection algo for our decomposed measure:

| | | | |
|---------------------|---|--|--|
| sample \mathbf{s} | with measure $\mu_1(\mathbf{s})$ | | $\mu_1(\mathbf{s}) = \prod_{i=1}^n q_i(s_i)$ |
| accept \mathbf{s} | with rate $a(\mathbf{s})$ | | $a(\mathbf{s}) = g^{*\mathbf{s}}(M)/g^{*\bar{n}}(M)$ |
| sample \mathbf{x} | with measure $\mu_2(\mathbf{x} \mathbf{s})$ | | $\mu_2(\mathbf{x} \mathbf{s}) = \prod_{i=1}^n g^{*s_i}(x_i)$ |

and that this algorithm has complexity

$$T \leq \frac{T_1^{\max}}{\mathbb{E}_{\mu_1}(a(\mathbf{s}))} + T_2^{\max} \quad \text{where } T_1^{\max}, T_2^{\max} = \Theta(n).$$

Under mild CLT hypotheses, the measure on $s = |\mathbf{s}|$ induced by $\mu_1(\mathbf{s})$ is a (normalised) Gaussian centered in \bar{n} , with variance $\sigma_1^2 N$, while $a(\mathbf{s})$ is an un-normalised Gaussian, centered in \bar{n} , with variance $\sigma_2^2 N$:

$$\mathbb{E}(a) \simeq \int dx \frac{1}{\sqrt{2\pi\sigma_1^2 N}} \exp \left[-\frac{x^2}{2N} \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right) \right] = \frac{\sigma_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}$$

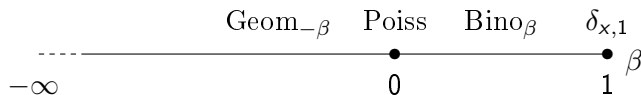
$$T \lesssim T_1^{\max} \sqrt{1 + (\sigma_1/\sigma_2)^2} + T_2^{\max} = \Theta(N)$$

The precise result

The three fundamental distributions

$$g_{\beta}^{*s}(r) = \begin{cases} \text{Bern}_{\beta}^{*s}(r) = \beta^r(1-\beta)^{s-r} \binom{s}{r} & \beta \in]0, 1[\\ \text{Pois}_s(r) = e^{-s} \frac{s^r}{r!} & \beta = 0 \\ \text{Geom}_{-\beta}^{*s}(r) = |\beta|^r(1+|\beta|)^{-s-r} \binom{s+r-1}{r} & \beta \in]-\infty, 0[\end{cases}$$

are such that g_{α}^{*s} has a **positive decomposition** in g_{β} iff $\alpha \leq \beta$.

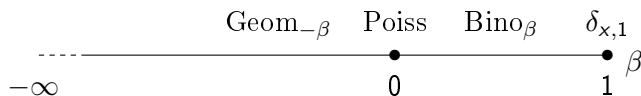


For the list of functions $\mathcal{F} = \{f_1, \dots, f_n\}$ in our measure,
call $\beta_{\min}(\mathcal{F})$ the smallest value of β
such that all the f_i 's have a positive decomposition in g_{β} .

The precise result

Then, the largest value for $\mathbb{E}(a)$ that can be achieved within our framework is

$$a_{\max}(\mathcal{F}) := \sqrt{1 - \beta_{\min}(\mathcal{F})} \cdot \sqrt{\frac{\sum_i \mathbb{E}[f_i]}{\sum_i \text{Var}[f_i]}}$$



A bonus surprise

If you have that $g(x)$ is Bern_b or Geom_b ,
then the BBHL algorithm at parameters $[n, m]$ can be used when
sampling from $\prod_i g^{*s_i}(x_i) \delta_{|x|, M}$, by rewriting $x_i = y_i^{(1)} + \dots + y_i^{(s_i)}$
Calling $s = \sum_i s_i$,

- ▶ $\text{BBHL}[s, M]$ can be used in the $g = \text{Bern}_b$ case, by identifying the outcome string of BBHL with the list of $y_i^{(j)}$'s.
- ▶ $\text{BBHL}[s + M - 1, M]$ can be used in the $g = \text{Geom}_b$ case, by identifying the lengths of runs of 0s in the outcome string of BBHL with the list of $y_i^{(j)}$'s.

The Poissonian case (more seldomly needed) can be dealt with a small algorithm that I invented, similar in spirit to BBHL

Examples of application

So, we have constructed our algorithm for the linear-time exact sampling of sum-constrained random variables, in the case in which they are *not* equally distributed.

However, you could just think:

*«who cares about not-equally-distributed variables?
After all, every time I wanted to generate walks, trees, etc.,
I always wanted equally-distributed variables... »*

The point is: examples of this sort may be **hidden beyond some smart bijection**, starting from more customary (and symmetric) problems.

This is well illustrated by two classical examples:

- Set partitions, and Stirling numbers of the second kind
- Permutations with m cycles, and Stirling numbers of the first kind

Set partitions, and Stirling numbers of the second kind

Call $\mathcal{S}_{n,m}^{\text{set}}$ the ensemble of partitions of a set with n (labelled) elements into m (unlabeled) non-empty subsets.

W.l.o.g. we can assume that the set has a total ordering.

Example, for $(n, m) = (28, 9)$, and the set

$\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, \alpha, \beta\}$

consider the partition

$\{\{a, g, t\}, \{b, d, m, o, \alpha\}, \{c, j, s, y\}, \{e, h, v\}, \{f, k, q\}, \{i, l, z\},$
 $\{n, p, u\}, \{r, \beta\}, \{w, x\}\}$

Set partitions, and Stirling numbers of the second kind

Call $\mathcal{S}_{n,m}^{\text{set}}$ the ensemble of partitions of a set with n (labelled) elements into m (unlabeled) non-empty subsets.

W.l.o.g. we can assume that the set has a total ordering.

Example, for $(n, m) = (28, 9)$, and the set

$\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, \alpha, \beta\}$

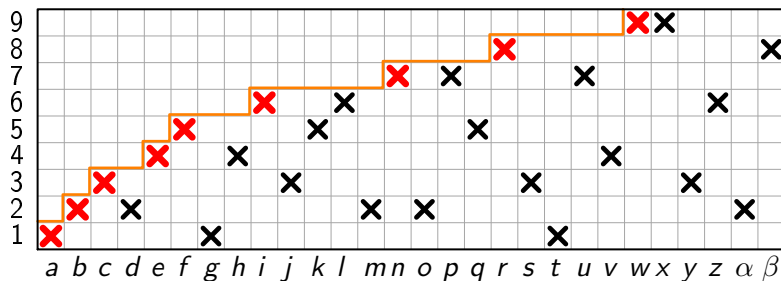
consider the partition

$\{\{a, g, t\}, \{b, d, m, o, \alpha\}, \{c, j, s, y\}, \{e, h, v\}, \{f, k, q\}, \{i, l, z\},$
 $\{n, p, u\}, \{r, \beta\}, \{w, x\}\}$

Although the sets are not labeled, they are canonically ordered, e.g. by their smallest element. As a result, we have a canonical incidence matrix T , with $T_{ij} = 1$ if the element j is in subset i .

Set partitions, and Stirling numbers of the second kind

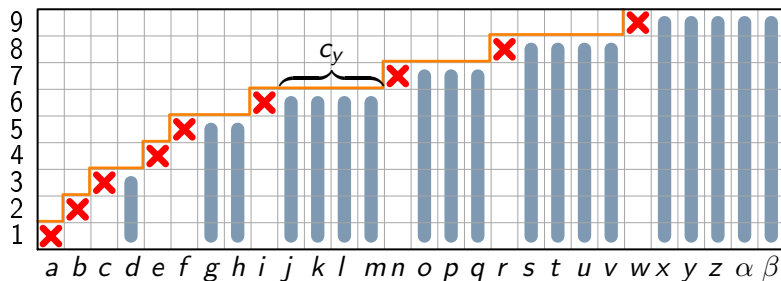
$$\{\{a, g, t\}, \{b, d, m, o, \alpha\}, \{c, j, s, y\}, \{e, h, v\}, \{f, k, q\}, \{i, l, z\}, \\ \{n, p, u\}, \{r, \beta\}, \{w, x\}\}$$



Call **backbone** $B(T)$ the list of smallest elements in the subsets, here $B = \{a, b, c, e, f, i, n, r, w\}$.

Set partitions, and Stirling numbers of the second kind

$$\{\{a, g, t\}, \{b, d, m, o, \alpha\}, \{c, j, s, y\}, \{e, h, v\}, \{f, k, q\}, \{i, l, z\}, \\ \{n, p, u\}, \{r, \beta\}, \{w, x\}\}$$



Call **backbone** $B(T)$ the list of smallest elements in the subsets, here $B = \{a, b, c, e, f, i, n, r, w\}$. The number of partitions T with $B(T) = B$ is the trivial product: $\prod_{y=1}^m y^{c_y}$, but the quantities c_y are **linearly constrained**: $\sum_y c_y = n - m$.

Set partitions, and Stirling numbers of the second kind

As a result, sampling **uniformly** set partitions in $\mathcal{S}_{n,m}$, which bijectively coincides to sampling uniformly the tableaux T , boils down to sampling the backbone B with the **non-uniform** measure

$$\mu_{n,m}(c_1, \dots, c_m) \propto \prod_{y=1}^m y^{c_y} \times \delta_{|c|, n-m}$$

This is exactly our framework! Introduce an appropriate Lagrange multiplier $\omega^{\sum_y c_y}$, in order to have $\mathbb{E}(|c|) = n - m$ (the good choice is the solution to the equation $\frac{n}{m} = -\frac{\ln(1-\omega)}{\omega}$)

The functions $f_y(c_y)$ are $\text{Geom}_{b_y}(c_y)$, with $b_y = \frac{\omega y}{n - \omega y}$

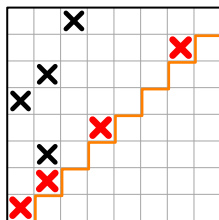
Now, Geom_a has a positive decomposition in terms of Bern_b

$$\text{Geom}_a(x) = \sum_s \text{Geom}_{\frac{a}{a+b}}(s) \text{Bern}_b^{*s}(x)$$

Choosing for simplicity $b = \frac{1}{2}$, our algorithm works, with an average acceptance rate $\mathbb{E}(a) = \sqrt{\frac{e^{-\theta} - 1 + \theta}{2(e^\theta - 1 - \theta)}} \quad (\omega = 1 - e^{-\theta})$

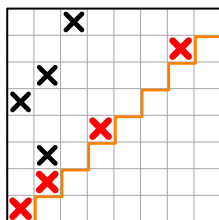
m -cycle permutations, and Stirling numbers of the 1st kind

Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



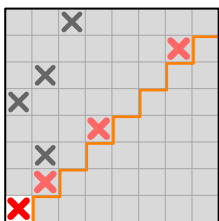
m -cycle permutations, and Stirling numbers of the 1st kind

Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



m -cycle permutations, and Stirling numbers of the 1st kind

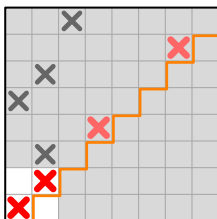
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((1))$$

m -cycle permutations, and Stirling numbers of the 1st kind

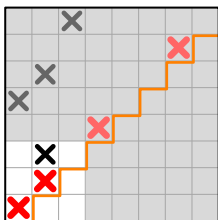
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((1)(2))$$

m -cycle permutations, and Stirling numbers of the 1st kind

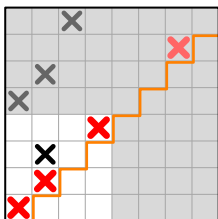
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((1)(23))$$

m -cycle permutations, and Stirling numbers of the 1st kind

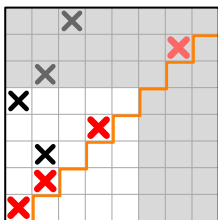
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((1)(23)(4))$$

m -cycle permutations, and Stirling numbers of the 1st kind

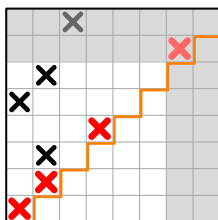
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((15)(23)(4))$$

m -cycle permutations, and Stirling numbers of the 1st kind

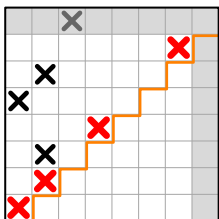
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((15)(263)(4))$$

m -cycle permutations, and Stirling numbers of the 1st kind

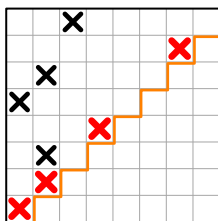
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((15)(263)(4)(7))$$

m -cycle permutations, and Stirling numbers of the 1st kind

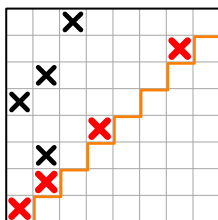
Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((15)(2638)(4)(7))$$

m -cycle permutations, and Stirling numbers of the 1st kind

Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$

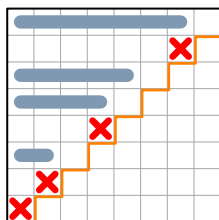


$$\sigma = ((15)(2638)(4)(7))$$

Call $B(\sigma) = \{0, 0, 1, 0, 1, 1, 0, 1\}$,
the indicator function of “black rows”
of $T(\sigma)$, the **backbone** of σ .

m -cycle permutations, and Stirling numbers of the 1st kind

Call $\mathcal{S}_{n,m}^{\text{cyc}}$ the set of permutations $\sigma \in \mathfrak{S}_n$ with m cycles.
Describe σ through the insertion table associated to its growth,
for example, for $\sigma = ((15)(2638)(4)(7))$



$$\sigma = ((15)(2638)(4)(7))$$

Call $B(\sigma) = \{0, 0, 1, 0, 1, 1, 0, 1\}$,
the indicator function of “black rows”
of $T(\sigma)$, the **backbone** of σ .

The number of σ 's with backbone
 $B = (x_1, \dots, x_n)$ is $\prod_y (y-1)^{x_y}$,
and we must have $|\mathbf{x}| = m$

Again, **this is exactly our framework!**
just with inhomogeneous Bernoulli variables,
instead of inhomogeneous Geometric variables.

Plan of the talk

So, I'm trying to 'sell' you **two recent tools** for improving the complexity of the “Boltzmann-like” problems I introduced.

In the “**Bridge case**”, if we have **positive decomposition** of the f_i 's in terms of a function g being Bern or Geom, I should have convinced you that my trick is 'optimal up to a factor'.

You shall be happy with this, unless you really search for rand-bit optimality (like in the nice BBHL algorithm).

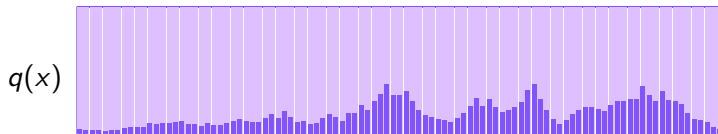
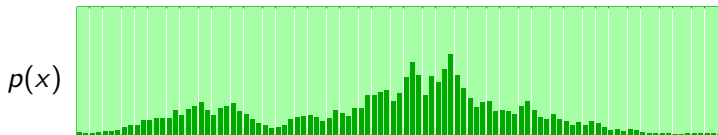
However, we are not always so lucky.

In particular, if we are in the “**Boltzmann case**”, we already start from a worse complexity ($\sim N^2$ because of fat tails), and we have much less tools in our hands...

It's time for me for trying to sell you the second tool:
the **improved Hadamard product** trick.

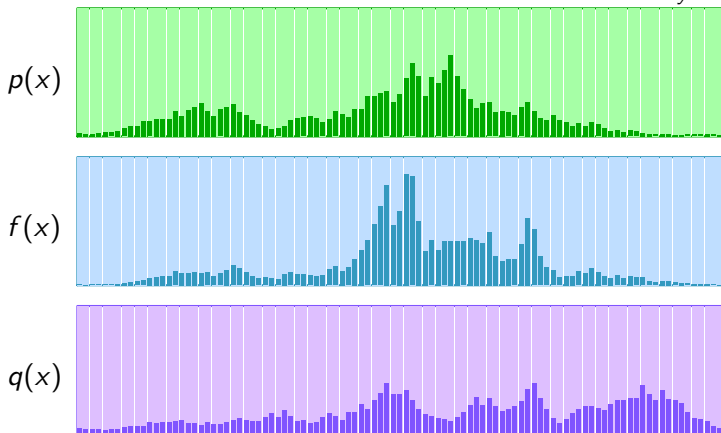
Sampling from the Hadamard product: the problem

We have two measures on \mathbb{Z} , $p(x)$ and $q(x)$.



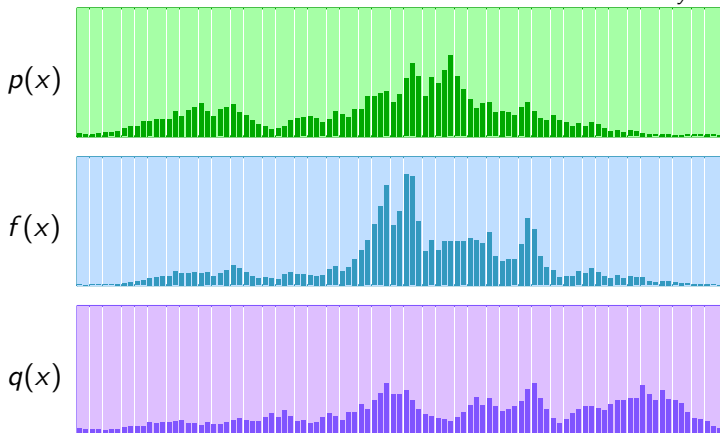
Sampling from the Hadamard product: the problem

We have two measures on \mathbb{Z} , $p(x)$ and $q(x)$. Let $f(x) = \frac{p(x)q(x)}{\sum_y p(y)q(y)}$



Sampling from the Hadamard product: the problem

We have two measures on \mathbb{Z} , $p(x)$ and $q(x)$. Let $f(x) = \frac{p(x)q(x)}{\sum_y p(y)q(y)}$



We have two black-box algorithms that sample from p and from q ,
and we want to sample from f

The obvious rejection algorithm

The rules of the game are clear. We have **no exploitable information** whatsoever on p and q . We only have the **black boxes**

The obvious rejection algorithm seems to be the only candidate:

Algorithm : Obvious rejection

begin

repeat

$x \leftarrow p$; $y \leftarrow q$

until $x = y$;

return x

end

Define the scalar product $(f, g) = \sum_x f(x)g(x)$.

The 'repeat' loop is repeated on average $1/(p, q)$ times.

If we have a **size parameter** n , this may be large,
and we want to make better

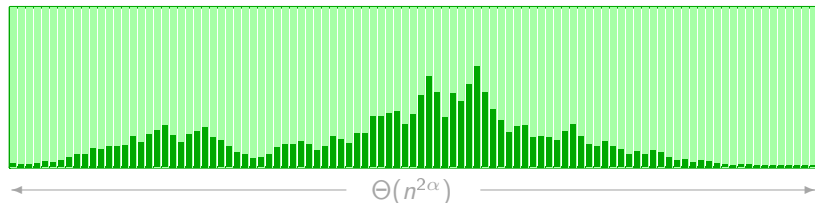
Scaling of the size parameter

Say that there exists a $\kappa \geq 3$ such that $\sum_x (p(x)^\kappa + q(x)^\kappa) \ll (p, q)$
(by Cauchy–Schwarz, it can't be $\kappa = 2$).

Then we want to bring down the complexity
from $\sim 1/(p, q)$ to $\sim 1/\sqrt{(p, q)}$ (maybe up to logs)

The typical case is $\kappa = 3$ above, and (p, p) , (p, q) , (q, q) are all $\Theta(n^{-2\alpha})$
(maybe up to logs)

Then, the complexity of the obvious rejection algorithm is $\Theta(n^{2\alpha})$
and we want to go down to $\Theta(n^\alpha \ln n)$ or $\Theta(n^\alpha)$



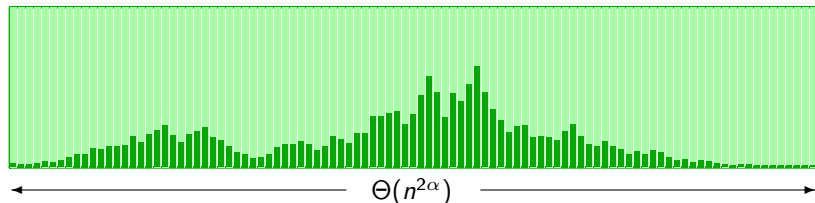
Scaling of the size parameter

Say that there exists a $\kappa \geq 3$ such that $\sum_x (p(x)^\kappa + q(x)^\kappa) \ll (p, q)$
(by Cauchy–Schwarz, it can't be $\kappa = 2$).

Then we want to bring down the complexity
from $\sim 1/(p, q)$ to $\sim 1/\sqrt{(p, q)}$ (maybe up to logs)

The typical case is $\kappa = 3$ above, and (p, p) , (p, q) , (q, q) are all $\Theta(n^{-2\alpha})$
(maybe up to logs)

Then, the complexity of the obvious rejection algorithm is $\Theta(n^{2\alpha})$
and we want to go down to $\Theta(n^\alpha \ln n)$ or $\Theta(n^\alpha)$



A complexity paradigm (you need this only if you care about logs)

In algorithm complexity you normally just count operations.

Nonetheless, when you have **black boxes**, it is wise to count separately operations and black-box queries, which are generally much more expensive than a single operation

Here we have two black boxes, for simplicity we will assume they have similar complexity

Nice notation: $\begin{cases} \text{👁️} & \text{for black-box queries} \\ \text{📀} & \text{for operations} \end{cases}$

Then, e.g., $\Theta(n^x \text{👁️} + n^y \text{📀})$ simplifies to $\Theta(n^x \text{👁️})$ if $x \geq y$, but stays as is if $x < y$, as we only know that $\text{👁️}/\text{📀} > 1$

The complexity of the **obvious rejection algorithm** is $\Theta(n^{2\alpha} \text{👁️})$
We want to **go down** to $\Theta(n^\alpha \text{👁️} + n^\alpha \ln(n) \text{📀})$

The naïve 'birthday paradox' algorithm

Let us consider the following algorithm

Algorithm : Birthday paradox, first try

begin

repeat

$(x_1, \dots, x_k) \leftarrow \boxed{p}$; $(y_1, \dots, y_k) \leftarrow \boxed{q}$

until $\exists! (i, j) \mid x_i = y_j$;

return x_i ;

end

Best hope: in each 'repeat' cycle

there are $\sim \text{Poiss}_{k^2(p,q)}$ pairs (i, j) such that $x_i = y_j$,

so if we tune $k \sim 1/\sqrt{(p, q)}$

the cycle costs $\Theta(k \text{ 📷 })$, and is repeated on average $\Theta(1)$ times

The complexity drops down to $\Theta(n^\alpha \text{ 📷 })$

An easy win?

An easy win?

An easy win?

An easy win? ... **No!**

An easy win?

An easy win? ... **No!**

besides a bunch of solvable minor issues
(is the number of good pairs really distributed as a Poissonian?)
(the 'wrong' naïve search for a good pair takes time k^2)
there is the **one big problem**:

An easy win?

An easy win? ... **No!**

besides a bunch of solvable minor issues
(is the number of good pairs really distributed as a Poissonian?)
(the 'wrong' naïve search for a good pair takes time k^2)
there is the **one big problem**:
The resulting probability distribution is **biased!**

An easy win? ... **No!**

besides a bunch of solvable minor issues
(is the number of good pairs really distributed as a Poissonian?)
(the 'wrong' naïve search for a good pair takes time k^2)
there is the **one big problem**:
The resulting probability distribution is biased!

The **average number** of good pairs (i, j) with $x_i = y_j = z$
is in fact **proportional** to $f(z)$, and **boosted** by a factor k^2 ,
which is good...

...but knowing that you have a **unique** good pair gives a **bias!**

However the whole idea remains valuable, because, if the average
number of good pairs is small, having no further pairs is 'normal',
so the bias is small, and maybe can be corrected with a
computationally-cheap trick.

Poissonisation of the naïve algorithm

Analysing the bias of the previous algorithm is complicated.
It gets easier if we 'Poissonise' k , i.e. we rather consider:

Algorithm : Birthday paradox, Poissonised

begin

repeat

 sample k_p and k_q with Pois_k ;

$(x_1, \dots, x_{k_p}) \leftarrow \boxed{p}$; $(y_1, \dots, y_{k_q}) \leftarrow \boxed{q}$

until $\exists! (i, j) \mid x_i = y_j$;

return x_i

end

Call $\nu_z = (a_z, b_z)$, with $a_z = \#\{i \mid x_i = z\}$ and $b_z = \#\{j \mid y_j = z\}$

The good fact of Poissonisation is that

the ν_z 's are independent random variables

so that we can easily analyse what is the probability of returning z

A formula for the bias

Fact: $\text{prob}(a_z, b_z) = \text{Poiss}_{kp(z)}(a_z)\text{Poiss}_{kq(z)}(b_z)$

thus the probability that there is a single pair for z is $\text{Poiss}_{kp(z)}(1)\text{Poiss}_{kq(z)}(1) = e^{-k(p(z)+q(z))} k^2 p(z)q(z)$

and the probability that there are no pairs for values $z' \neq z$ is $1 - (1 - \text{Poiss}_{kp(z')}(0))(1 - \text{Poiss}_{kq(z')}(0)) = e^{-k(p(z')+q(z'))} (e^{kp(z')} + e^{kq(z')} - 1)$

as a result, the probability of returning z is proportional to[†]

$$\frac{f(z)}{e^{kp(z)} + e^{kq(z)} - 1}$$

and we would be fine if we could devise a final rejection procedure for the spurious factor $\text{Bias}(z) := 1/(e^{kp(z)} + e^{kq(z)} - 1)$

[†] I.e., up to a factor, like $\prod_{z'} \psi(z')$, which is the same for all z , although possibly complicated to calculate.

A philosophical digression

So, we want to add a final procedure
for correcting the spurious factor $\text{Bias}(z) := 1/(e^{kp(z)} + e^{kq(z)} - 1)$.

In other words, for some $C > \max_z (e^{kp(z)} + e^{kq(z)} - 1)$,

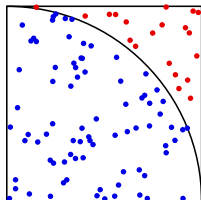
but still with $\sum_z f(z) \frac{e^{kp(z)} + e^{kq(z)} - 1}{C} = \Theta(1)$,

I want to sample a Bernoulli random variable of parameter $\frac{e^{kp(z)} + e^{kq(z)} - 1}{C}$

However, recall that **we have no analytic information on p and q**
(we only have the black boxes!)

Can we ever sample Bern_ξ without evaluating ξ ?

This is **not impossible a priori**,
just remember the famous algorithm
for $\text{Bern}_{\pi/4}$ that makes no use of π ...



A first solution with static lists

Algorithm : Hadamard product, with static lists

repeat

sample k_p and k_q with Poiss_k ;

$(x_1, \dots, x_{k_p}) \leftarrow \boxed{p}$; $(y_1, \dots, y_{k_q}) \leftarrow \boxed{q}$;

sort the lists above, produce the list of $\{(z, a_z, b_z)\}$;

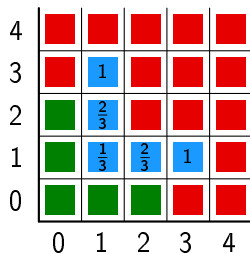
if $N_{\bullet} = 0$ **and** $N_{\bullet} = 1$ **then**

$\nu \leftarrow a_z + b_z - 1$ for the only $z \in \Omega_{\bullet}$;

$\text{win} \leftarrow \text{Bern}_{\nu/3}$;

until $\text{win} = \text{true}$;

return z ;



Why this algorithm is unbiased

Let us first see why this algorithm is correct

Call for short $\phi(z) = e^{-k(p(z)+q(z))}$

The algorithm may return z only if
no $z' \neq z$ arrives out of the Ω_\bullet region,

each z' makes a factor

$$\phi(z') \left(1 + kp(z') + \frac{(kp(z'))^2}{2} + kq(z') + \frac{(kq(z'))^2}{2} \right)$$

Then, it shall also happen that z falls within the Ω_\bullet region
this makes a factor

$$k^2 p(z)q(z) \phi(z) \left(1 + \frac{kp(z)}{2} + \frac{(kp(z))^2}{6} + \frac{kq(z)}{2} + \frac{(kq(z))^2}{6} \right)$$

We would have won if the two factors in parenthesis
were proportional. But **they are not**. For this reason
we put a final **Bernoulli rejection**, that corrects for the factorials

| | | | | | |
|---|---|---|---|---|---|
| 4 | ■ | ■ | ■ | ■ | ■ |
| 3 | ■ | ■ | ■ | ■ | ■ |
| 2 | ■ | ■ | ■ | ■ | ■ |
| 1 | ■ | ■ | ■ | ■ | ■ |
| 0 | ■ | ■ | ■ | ■ | ■ |
| | 0 | 1 | 2 | 3 | 4 |

Why this algorithm is unbiased

Let us first see why this algorithm is correct

Call for short $\phi(z) = e^{-k(p(z)+q(z))}$

The algorithm may return z only if
no $z' \neq z$ arrives out of the Ω_\bullet region,

each z' makes a factor

$$\phi(z') \left(1 + kp(z') + \frac{(kp(z'))^2}{2} + kq(z') + \frac{(kq(z'))^2}{2} \right)$$

Then, it shall also happen that z falls within the Ω_\bullet region
this makes a factor

$$k^2 p(z)q(z) \phi(z) \left(\frac{1}{3} \cdot 1 + \frac{2}{3} \cdot \frac{kp(z)}{2} + 1 \cdot \frac{(kp(z))^2}{6} + \frac{2}{3} \cdot \frac{kq(z)}{2} + 1 \cdot \frac{(kq(z))^2}{6} \right)$$

We would have won if the two factors in parenthesis
were proportional. But **they are not**. For this reason
we put a final **Bernoulli rejection**, that corrects for the factorials

| | | | | | |
|---|---|---------------|---------------|---|---|
| 4 | ■ | ■ | ■ | ■ | ■ |
| 3 | ■ | 1 | ■ | ■ | ■ |
| 2 | ■ | $\frac{2}{3}$ | ■ | ■ | ■ |
| 1 | ■ | $\frac{1}{3}$ | $\frac{2}{3}$ | 1 | ■ |
| 0 | ■ | ■ | ■ | ■ | ■ |
| | 0 | 1 | 2 | 3 | 4 |

Why this algorithm is fast

We have taken the apparently risky choice of making a very large rejection region Ω_{\bullet} . If any z falls here, we restart no matter what. Does this affect the complexity in an important way, or the estimates from the naïve birthday heuristics are still valid?

One run costs on average $2k \text{ (eye)} + k \ln(k) \text{ (stack)}$ (for sorting),
i.e. already the sought $\Theta(n^\alpha \text{ (eye)} + n^\alpha \ln(n) \text{ (stack)})$,
so we win **iff the acceptance rate of one run is $\Theta(1)$**

This rate is easily deduced from the previous calculation, to be

$$\left(\sum_z \frac{1}{3} k^2 p(z) q(z) \right) \prod_w \phi(w) \left(1 + kp(w) + \frac{(kp(w))^2}{2} + kq(w) + \frac{(kq(w))^2}{2} \right)$$
$$=: \frac{1}{3} k^2 (p, q) \prod_w \psi(w) \quad \dots \text{we claim that } \prod_w \psi(w) \simeq e^{-k^2(p, q)}$$

Why this algorithm is fast

If this is true, the average complexity can be made as small as

$$\begin{aligned} \overline{\text{complexity}} + \Theta(\ln n) &= \min_k \frac{2k}{\frac{1}{3}k^2(p, q)e^{-k^2(p, q)}} \\ &= \frac{1}{\sqrt{(p, q)}} \min_x \frac{2\sqrt{x}}{\frac{x}{3}\exp(-x)} = \frac{1}{\sqrt{(p, q)}} \min_x \frac{6e^x}{\sqrt{x}} = \frac{6\sqrt{2e}}{\sqrt{(p, q)}} \end{aligned}$$

The reason why this is true is that

$$\frac{1}{\psi(w)} = 1 + \frac{k^2 p(w)q(w) + \frac{k^3}{3!}(p(w)^3 + q(w)^3) + \dots}{1 + kp(w) + \frac{(kp(w))^2}{2} + kq(w) + \frac{(kq(w))^2}{2}}$$

$$\begin{aligned} \text{thus } \prod_w \psi(w)^{-1} &\simeq \prod_w (1 + k^2 p(w)q(w)) \simeq \\ &\prod_w \exp(k^2 p(w)q(w)) = \exp(k^2(p, q)) \end{aligned}$$

Why this algorithm is fast

If this is true, the average complexity can be made as small as

$$\begin{aligned} \overbrace{\text{complexity}}^{\text{eye}} + \Theta(\ln n) \underbrace{\text{stack}} &= \min_k \frac{2k}{\frac{1}{3}k^2(p, q)e^{-k^2(p, q)}} \\ &= \frac{1}{\sqrt{(p, q)}} \min_x \frac{2\sqrt{x}}{\frac{x}{3}\exp(-x)} = \frac{1}{\sqrt{(p, q)}} \min_x \frac{6e^x}{\sqrt{x}} = \frac{6\sqrt{2e}}{\sqrt{(p, q)}} \end{aligned}$$

The reason why this is true is that

$$\frac{1}{\psi(w)} = 1 + \frac{k^2 p(w)q(w) + \frac{k^3}{3!}(p(w)^3 + q(w)^3) + \dots}{1 + kp(w) + \frac{(kp(w))^2}{2} + kq(w) + \frac{(kq(w))^2}{2}}$$

$$\begin{aligned} \text{thus } \prod_w \psi(w)^{-1} &\simeq \prod_w (1 + k^2 p(w)q(w)) \simeq \\ &\prod_w \exp(k^2 p(w)q(w)) = \exp(k^2(p, q)) \end{aligned}$$

Why this algorithm is fast

If this is true, the average complexity can be made as small as

$$\begin{aligned} \overbrace{\text{complexity}}^{\text{eye}} + \Theta(\ln n) \underbrace{\text{stack}} &= \min_k \frac{2k}{\frac{1}{3}k^2(p, q)e^{-k^2(p, q)}} \\ &= \frac{1}{\sqrt{(p, q)}} \min_x \frac{2\sqrt{x}}{\frac{x}{3}\exp(-x)} = \frac{1}{\sqrt{(p, q)}} \min_x \frac{6e^x}{\sqrt{x}} = \frac{6\sqrt{2e}}{\sqrt{(p, q)}} \end{aligned}$$

The reason why this is true is that

$$\frac{1}{\psi(w)} = 1 + \frac{k^2 p(w)q(w) + \frac{k^3}{3!}(p(w)^3 + q(w)^3) + \dots}{1 + kp(w) + \frac{(kp(w))^2}{2} + kq(w) + \frac{(kq(w))^2}{2}}$$

$$\begin{aligned} \text{thus } \prod_w \psi(w)^{-1} &\simeq \prod_w (1 + k^2 p(w)q(w)) \simeq \\ &\prod_w \exp(k^2 p(w)q(w)) = \exp(k^2(p, q)) \end{aligned}$$

Why this algorithm is fast

If this is true, the average complexity can be made as small as

$$\begin{aligned} \overline{\text{complexity}} + \Theta(\ln n) &= \min_k \frac{2k}{\frac{1}{3}k^2(p, q)e^{-k^2(p, q)}} \\ &= \frac{1}{\sqrt{(p, q)}} \min_x \frac{2\sqrt{x}}{\frac{x}{3}\exp(-x)} = \frac{1}{\sqrt{(p, q)}} \min_x \frac{6e^x}{\sqrt{x}} = \frac{6\sqrt{2e}}{\sqrt{(p, q)}} \end{aligned}$$

The reason why this is true is that

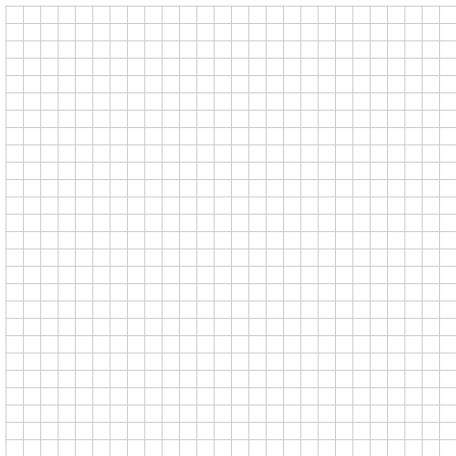
negligible by
our assumption on
 $\sum_z (p(z)^3 + q(z)^3)$

$$\frac{1}{\psi(w)} = 1 + \frac{k^2 p(w)q(w) + \frac{k^3}{3!}(p(w)^3 + q(w)^3) + \dots}{1 + kp(w) + \frac{(kp(w))^2}{2} + kq(w) + \frac{(kq(w))^2}{2}}$$

$$\begin{aligned} \text{thus } \prod_w \psi(w)^{-1} &\simeq \prod_w (1 + k^2 p(w)q(w)) \simeq \\ &\prod_w \exp(k^2 p(w)q(w)) = \exp(k^2(p, q)) \end{aligned}$$

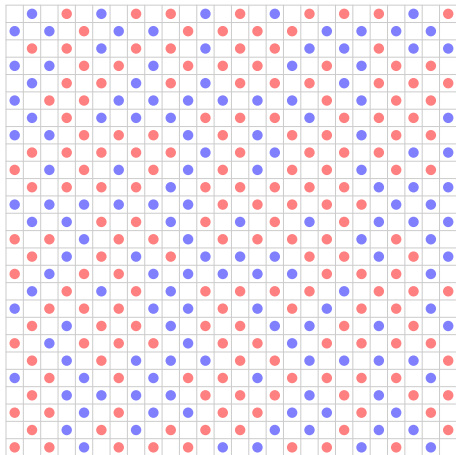
One application: random bridges in random media

Consider a bistochastic digraph (a graph with uniform in- and out-degree) which has some 'translational invariance on average' (I choose here a model of plaquettes i.i.d. oriented clockwise or counterclockwise)



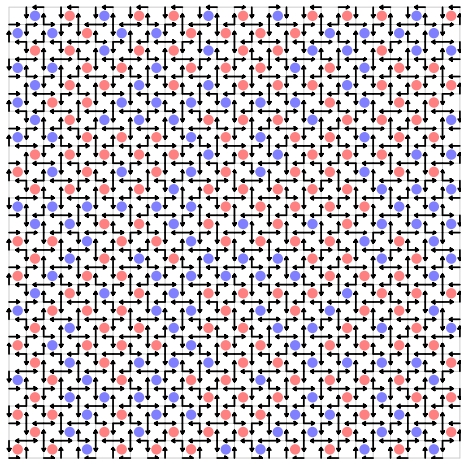
One application: random bridges in random media

Consider a bistoochastic digraph (a graph with uniform in- and out-degree) which has some 'translational invariance on average' (I choose here a model of plaquettes i.i.d. oriented clockwise or counterclockwise)



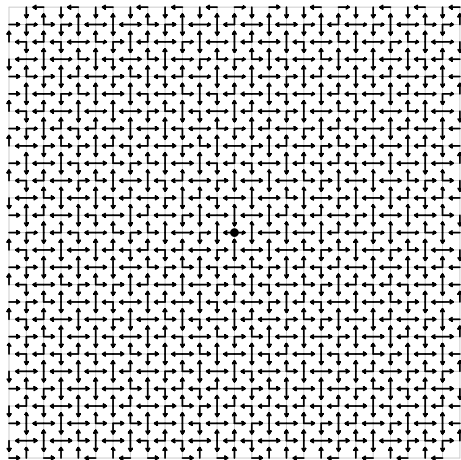
One application: random bridges in random media

Consider a bistochastic digraph (a graph with uniform in- and out-degree) which has some 'translational invariance on average' (I choose here a model of plaquettes i.i.d. oriented clockwise or counterclockwise)



One application: random bridges in random media

Consider a bistochastic digraph (a graph with uniform in- and out-degree) which has some 'translational invariance on average' (I choose here a model of plaquettes i.i.d. oriented clockwise or counterclockwise)

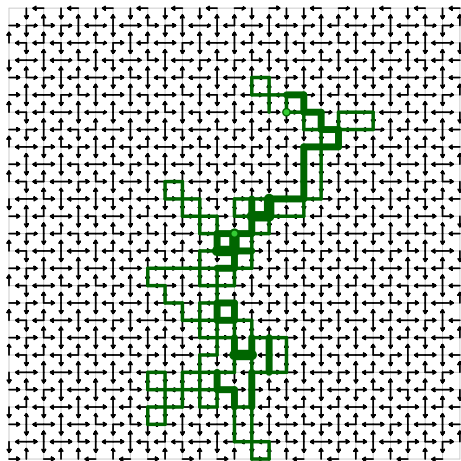


Now we have a **random** instance of a graph.

On top of this, we will consider **random walks** starting at the origin

One application: random bridges in random media

Consider a bistochastic digraph (a graph with uniform in- and out-degree) which has some 'translational invariance on average' (I choose here a model of plaquettes i.i.d. oriented clockwise or counterclockwise)

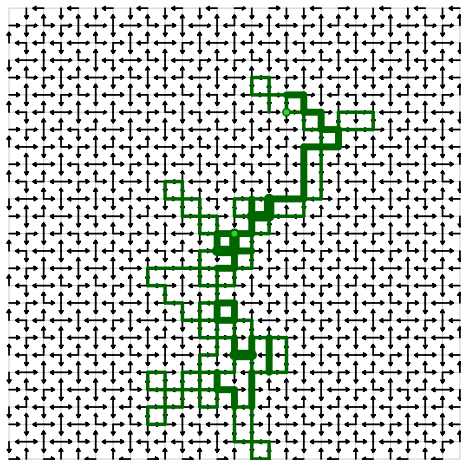


Now we have a **random** instance of a graph.

On top of this, we will consider **random walks** starting at the origin

One application: random bridges in random media

Consider a bistochastic digraph (a graph with uniform in- and out-degree) which has some 'translational invariance on average' (I choose here a model of plaquettes i.i.d. oriented clockwise or counterclockwise)

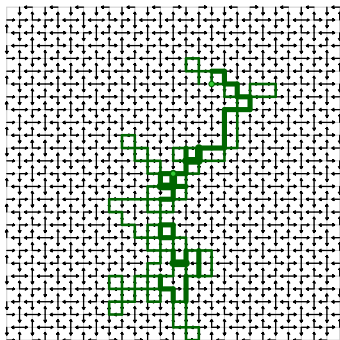


Now we have a **random** instance of a graph.

On top of this, we will consider **random walks** starting at the origin

Do you have a good algorithm for uniformly sampling walks $W_{\mathbf{0} \rightarrow \mathbf{0}}^{(n)}$, of length $2n$, **starting and arriving at the origin?**

One application: random walks in random media



Sampling random walks of length $2n$, with no prescribed endpoint, is trivially linear...

but the endpoint \mathbf{x} is a random variable, probably at distance $\sim \sqrt{n}$, and the probability that $\mathbf{x} = \mathbf{0}$ is roughly $n^{-\frac{D}{2}}$

Naïve rejection algorithm: complexity $n^{1+\frac{D}{2}}$

Now, grow the first half of the path, of length n , up to its endpoint \mathbf{x} (which has distribution $p(\mathbf{x})$), and grow the last half of the path, again starting from the origin, and walking on the digraph with reversed orientation (this has distribution $q(\mathbf{x})$)

If you use the algorithm described here, with these p and q , the complexity goes from $n^{1+\frac{D}{2}}$ to $n^{1+\frac{D}{4}}$

A more complicated problem

Now suppose that you have a **family** of Hadamard product problems, that is, for some set \mathcal{C} , you want to sample x from

$$f(x) = \frac{\sum_{c \in \mathcal{C}} g_c p_c(x) q_c(x)}{\sum_{c, y} g_c p_c(y) q_c(y)}$$

(with $\sum_c g_c = 1$, and $\sum_y p_c(y) = \sum_y q_c(y) = 1$ for all c)

Our winning strategy was to sample x with a probability of the form

$$k^2 p(x) q(x) \prod_y \psi(y)$$

where the k^2 factor is the “birthday paradox” enhancement, and the $\prod_y \psi(y)$ is an irrelevant factor, of order 1 when $k^2(p, q)$ is of order 1.

However, if we just do the same in this more general framework, the $\prod_y \psi(y)$ factor **depends on \mathcal{C}** , and is not irrelevant anymore. . .

Can we correct its contribution?

A more complicated problem

If we just apply the strategy above, we sample a measure of the like

$$f_{\text{alg}}(x) \propto \sum_{c \in \mathcal{C}} g_c \frac{1}{3} k^2 p_c(x) q_c(x) \exp(-k^2(p_c, q_c) + \dots)$$

We have a nice variant of the Hadamard product algorithm, which uses “incremental lists” instead of “static” ones.

This works with continuous time and Poisson clocks, and the probability of terminating at time in $[t, t + dt]$ is

$$f_{\text{alg}}(x, t) dt \propto \sum_{c \in \mathcal{C}} g_c \frac{1}{3} t p_c(x) q_c(x) \exp(-t^2(p_c, q_c) + t^3 \dots) dt$$

that is, integrating over time,

$$f_{\text{alg}}(x) \sim \frac{1}{3} \sum_{c \in \mathcal{C}} g_c \frac{p_c(x) q_c(x)}{(p_c, q_c) + \dots}$$

this other bias factor is a bit easier to correct...

A more complicated problem

Summary: the “incremental lists” variant of the Hadamard product algorithm gives a measure

$$f_{\text{alg}}(x) \sim \frac{1}{3} \sum_{c \in \mathcal{C}} g_c \frac{p_c(x) q_c(x)}{(p_c, q_c) + \dots}$$

instead of the desired $f(x) \propto \sum_{c \in \mathcal{C}} g_c p_c(x) q_c(x)$.

We can correct this bias, **up to error terms that we still have to fix**, by first doing the Birthday paradox strategy (at given c with a static parameter k such that we know that $k^2 \max_c (p_c, q_c) \lesssim 1$), and then perform the incremental list run at c only if we have found some “good pair”. This gives a probability of the form

$$\begin{aligned} f_{\text{alg}}(x) &\sim \frac{1}{3} \sum_{c \in \mathcal{C}} g_c \frac{p_c(x) q_c(x)}{(p_c, q_c) + \dots} \left[1 - \prod_y (e^{-k p_c(y)} + e^{-k q_c(y)} - e^{-k(p_c(y) + q_c(y))}) \right] \\ &\sim \frac{1}{3} \sum_{c \in \mathcal{C}} g_c p_c(x) q_c(x) \frac{k^2 (p_c, q_c) + \dots}{(p_c, q_c) + \dots} \propto \sum_{c \in \mathcal{C}} g_c p_c(x) q_c(x) (1 + \dots) \end{aligned}$$

Boltzmann sampling of specificable structures

Let us just pretend that we can solve this 'small' $(1 + \dots)$ problem, and see what this would imply at the level of applications...

Recall the framework of 'combinatorial specification' systems:

$$\begin{cases} A_1 = F_1(A_1, \dots, A_k, z) \\ \vdots \\ A_k = F_k(A_1, \dots, A_k, z) \end{cases}$$

By the introduction of marked objects
(a useful trick in order to modify the tail exponent of the distribution),
you get associated equations for the marked classes

$$\begin{cases} A'_1 = \sum_i A'_i F_{1,i}(A_1, \dots, A_k, z) + F_{1,0}(A_1, \dots, A_k, z) \\ \vdots \\ A'_k = \sum_i A'_i F_{k,i}(A_1, \dots, A_k, z) + F_{k,0}(A_1, \dots, A_k, z) \end{cases}$$

Boltzmann sampling of specifiable structures

Now, redefine $A_i = \frac{1}{2}(A_i^{\text{red}} + A_i^{\text{blue}})$ in the A'_i equations, and perform the Galton–Watson exploration of the A'_i ‘spine’ only, up to when you hit the marked object.

At this point you have typically produced:

- ▶ $\nu_i^{\text{red}} \sim \sqrt{N}$ red branches A_i^{red} (for the various $1 \leq i \leq k$),
- ▶ $\nu_i^{\text{blue}} \sim \sqrt{N}$ blue branches A_i^{blue} ,
- ▶ $n_0 \sim \sqrt{N}$ elementary units,
- ▶ a multiplicative combinatorial factor g .


Call c the datum of $\{\nu_i^{\text{red}}, \nu_i^{\text{blue}}, n_0, g\}$ and

$$p_c(x) = [z^x] \prod_i (A_i^{\text{red}}(z))^{\nu_i^{\text{red}}}$$

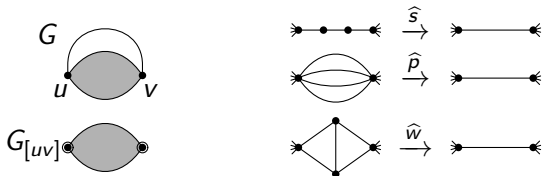
$$q_c(x) = [z^{N-n_0(c)-x}] \prod_i (A_i^{\text{blue}}(z))^{\nu_i^{\text{blue}}}$$

Then we are exactly in the framework above!

Graphs with no W_4 minor

Let's choose the example of W_4 -free graphs,
that is, graphs that do not contain  as a minor.

These graphs, rooted at one edge, are in bijection
with 2-terminal graphs $G_{[uv]}$ that can be reduced to one edge
by series, parallel and 'wheatstone bridge' reductions:



Crucially, for any such graph, **the decomposition tree is unique!**

Thus counting the graphs is like counting these trees,
which gives rise to a system of equations
(just like series-parallel graphs, but a bit more complicated)

Graphs with no W_4 minor

Introduce the combinatorial classes:

- ▶ z is the one-edge trivial class;
- ▶ A is all the (2-terminal) graphs we want;
- ▶ S graphs consist of 2 or more blocks in series in the outermost layer;
- ▶ P graphs consist of 2 or more blocks;
- ▶ W graphs consist of one Wheatstone bridge.

This gives rise to Galton–Watson trees in which:

- ▶ all and only the leaves are z ;
- ▶ A is only at the root, has a single child in the list $\{S, P, W, z\}$;
- ▶ S and P do not have children with same label, and have degree at least 2. A P node of degree d comes with a symmetry factor $1/d!$;
- ▶ W has degree 5 and symmetry factor $1/2$.

Graphs with no W_4 minor

Introduce the combinatorial classes:

- ▶ z is the one-edge trivial class;
- ▶ A is all the (2-terminal) graphs we want;
- ▶ S graphs consist of 2 or more blocks in series in the outermost layer;
- ▶ P graphs consist of 2 or more blocks;
- ▶ W graphs consist of one Wheatstone bridge.

This gives rise to Galton–Watson trees in which:

- ▶ all and only the leaves are z ;
- ▶ A is only at the root, has a single child in the list $\{S, P, W, z\}$;
- ▶ S and P do not have children with same label, and have degree at least 2. A P node of degree d comes with a symmetry factor $1/d!$;
- ▶ W has degree 5 and symmetry factor $1/2$.

Graphs with no W_4 minor

$$\begin{cases} A = S + P + W + z \\ S = (A - S)^2 / (1 - (A - S)) \\ P = \exp(A - P) - (1 + (A - P)) \\ W = A^5 / 2 \end{cases}$$

$$\begin{cases} A = S + P + \mathcal{W}(A) + z & \mathcal{W}(A) = A^5/2 \\ S = (A - S)^2 / (1 - (A - S)) \\ P = \exp(A - P) - (1 + (A - P)) \end{cases}$$

Graphs with no W_4 minor

$$\begin{cases} A = S + P + \mathcal{W}(A) + z & \mathcal{W}(A) = A^5/2 \\ S = A^2/(1+A) \\ P = A - \ln(1+A) \end{cases}$$

$$z = \ln(1+A) - \frac{A^2}{1+A} - \mathcal{W}(A)$$

(A_c, z_c) is the position of the first point of slope zero, on the branch starting in $(0, 0)$ with slope 1 (this holds for any polynomial $\mathcal{W}(A)$, i.e. any recursive family of graphs). Thus A_c is a root of the polynomial $1 - A - A^2 - (1+A)^2 \mathcal{W}'(A) = 0$. So we 'know' the branching rates of the associated critical Galton–Watson process.

Let's go back to the 'combinatorial' system...

$$\begin{cases} A = S + P + \mathcal{W}(A) + z \\ S = \sum_{k \geq 2} (A - S)^k \\ P = \sum_{k \geq 2} \frac{1}{k!} (A - P)^k \end{cases}$$

With one marking we get

$$\begin{cases} A' = S' + P' + A' \mathcal{W}'(A) + 1 \\ S' = (A' - S') \sum_{k \geq 1} (k+1)(A - S)^k \\ P' = (A' - P') \sum_{k \geq 1} \frac{1}{k!} (A - P)^k \end{cases}$$

Graphs with no W_4 minor

A typical example of the 'spine' of the tree following the X' class:

$$\left\{ \begin{array}{l} A = S + P + \mathcal{W}(A) + z \\ S = \sum_{k \geq 2} (A - S)^k \\ P = \sum_{k \geq 2} \frac{1}{k!} (A - P)^k \end{array} \right. \quad \left\{ \begin{array}{l} A' = S' + P' + A' \mathcal{W}'(A) + 1 \\ S' = (A' - S') \sum_{k \geq 1} (k + 1) (A - S)^k \\ P' = (A' - P') \sum_{k \geq 1} \frac{1}{k!} (A - P)^k \end{array} \right.$$

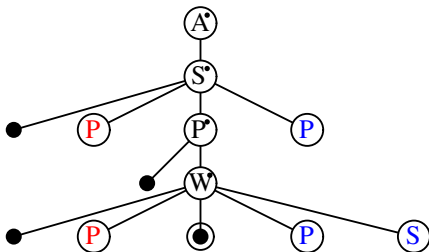
(the colours tell how to implement the Hadamard product trick)

Graphs with no W_4 minor

A typical example of the 'spine' of the tree following the X' class:

$$\begin{cases} W = \frac{1}{2}(S+P+W+z)^5 \\ S = \sum_{k \geq 2} (P+W+z)^k \\ P = \sum_{k \geq 2} \frac{1}{k!} (S+W+z)^k \end{cases} \quad \begin{cases} W' = \frac{5}{2}(S'+P'+W'+1)(S+P+W+z)^4 \\ S' = (P'+W'+1) \sum_{k \geq 1} (k+1)(P+W+z)^k \\ P' = (S'+W'+1) \sum_{k \geq 1} \frac{1}{k!} (S+W+z)^k \end{cases}$$

(the colours tell how to implement the Hadamard product trick)



Graphs with no W_4 minor

A typical example of the 'spine' of the tree following the X' class:

$$\begin{cases} W = \frac{1}{2}(S+P+W+z)^5 \\ S = \sum_{k \geq 2} (P+W+z)^k \\ P = \sum_{k \geq 2} \frac{1}{k!} (S+W+z)^k \end{cases} \quad \begin{cases} W' = \frac{5}{2}(S'+P'+W'+1)(S+P+W+z)^4 \\ S' = (P'+W'+1) \sum_{k \geq 1} (k+1)(P+W+z)^k \\ P' = (S'+W'+1) \sum_{k \geq 1} \frac{1}{k!} (S+W+z)^k \end{cases}$$

(the colours tell how to implement the Hadamard product trick)

