# Polytope sampling algorithms and their applications to computer science and ecology

Matthieu Dien

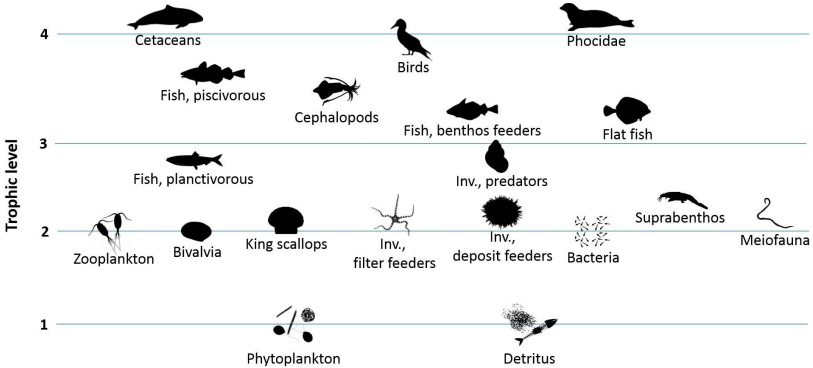
Université de Caen - GREYC - CNRS UMR 6072

28 June, 2023

# Outline

# Problem 1: food webs

Joint work with V. Girardin, T. Grente, N. Niquil and P. Regnault




(from "Cumulative effects of marine renewable energy and climate change on ecosystem properties: Sensitivity of ecological network analysis" by Nogues *et al.* )

# Problem 1: food webs

Joint work with V. Girardin, T. Grente, N. Niquil and P. Regnault

$$
\begin{aligned}
f_{i,j} &\geq 0, && ij \in E,\\
f_i &= \textstyle\sum_{j\in N^-(i)} f_{j,i}, && i \in S,\\
\textstyle\sum_{j\in N^+(i)} f_{i,j} - \sum_{j\in N^-(i)} f_{j,i} &= 0, && i \in S,\\
f_1 - f_{1,\mathrm{res}} - f_{1,\mathrm{det}} - c^+_{\mathrm{pro}} &\leq 0,\\
-f_1 + f_{1,\mathrm{res}} + f_{1,\mathrm{det}} + c^-_{\mathrm{pro}} &\leq 0,\\
-c^+_{\mathrm{res},i} f_i + f_{i,\mathrm{res}} &\leq 0, && i \in S',\\
c^-_{\mathrm{res},i} f_i - f_{i,\mathrm{res}} &\leq 0, && i \in S',\\
-c^+_{\mathrm{det},i} f_i + f_{i,\mathrm{det}} &\leq 0, && i \in S', i \neq \mathrm{PHY1}\\
c^-_{\mathrm{det},i} f_i - f_{i,\mathrm{det}} &\leq 0, && i \in S', i \neq \mathrm{PHY1}\\
-c^+_{\mathrm{det,phy}} P_{\mathrm{phy}} + f_{\mathrm{phy,det}} &\leq 0,\\
-f_i + f_{i,\mathrm{res}} + f_{i,\mathrm{det}} + c^-_{\mathrm{eff},i} f_i &\leq 0, && i \in S',\\
f_i - f_{i,\mathrm{res}} - f_{i,\mathrm{det}} - c^+_{\mathrm{bio},i} B_i &\leq 0, && i \in S',\\
-f_i + f_{i,\mathrm{res}} + f_{i,\mathrm{det}} + c^-_{\mathrm{bio},i} B_i &\leq 0, && i \in S',\\
f_{j,i} - c^+_{\mathrm{diet},i,j} f_i &\leq 0, && j \in S, i \in S',\\
-f_{j,i} + c^-_{\mathrm{diet},i,j} f_i &\leq 0, && j \in S, i \in S'.
\end{aligned}
$$

## Ecological interpretation

Each solution of the problem (*i.e.* a point of the underlying polytope) is possible scenario.

(equations from "Analysis of trophic networks: an optimisation approach" by Caputo *et al.*)

# Problem 2: synthesis of random tests for software

Joint work with V. Botbol, C. Dubois, A. Gotlieb, M. Pépin and G. Ziat

## Ocaml

A functionnal language with a powerful type system *e.g.*

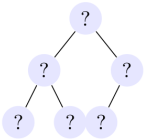
```
type t

type t Tree = Leaf of t
            | Node of t Tree list

type MTree = Leaf | UNode of UTree | BNode of BTree
and  UTree = MTree
and  BTree = MTree * MTree
```

# Problem 2: synthesis of random tests for software

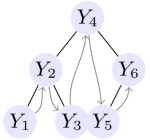
Joint work with V. Botbol, C. Dubois, A. Gotlieb, M. Pépin and G. Ziat

**Our contribution**

Add numerical constraints over the program's variables.



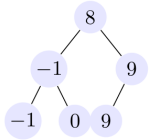

(a) Shape generation



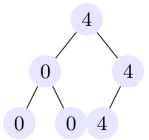

(b) Depth-first walk

$$X_1 \in -2..8, X_2 \in -3..5, X_3 \in -3..10,$$
$$X_4 \in -1..9, X_5 \in 0..7, X_6 \in 0..8,$$
$$\text{sort}((X_1,..,X_6),(Y_1,..,Y_6)),$$
$$Y_2 = Y_1 + Y_3, Y_5 = Y_6, Y_4 = Y_2 + Y_6$$

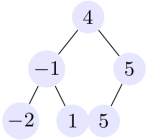
(c) Generating the corresponding CSP




(d) PRT first sol.



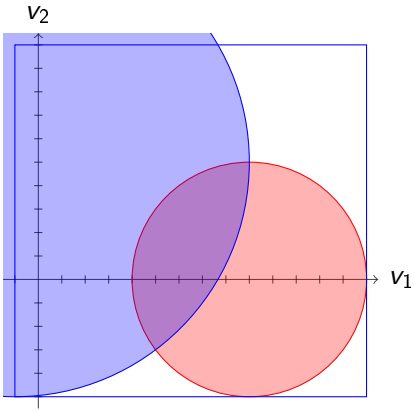

(e) PRT second sol.




(f) PRT third sol.

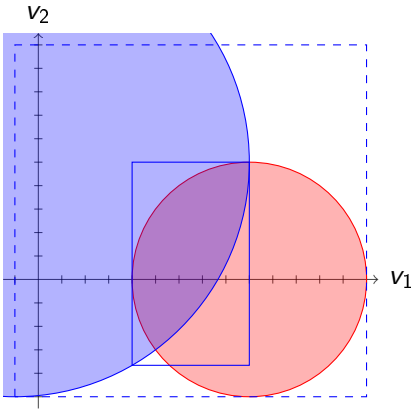
# Abstract solving method

```
1: function solve(D, C, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(D)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, C)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



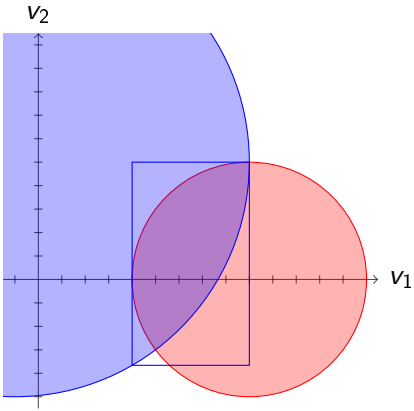
# Abstract solving method

```
1: function solve(D, C, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(D)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, C)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



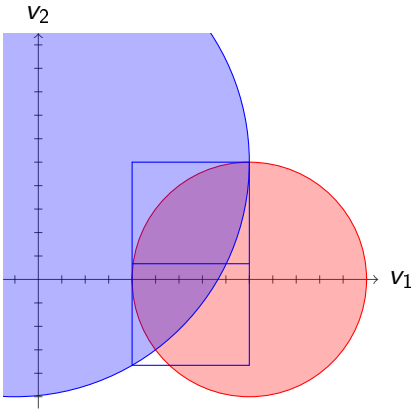
# Abstract solving method

```
1: function solve(𝒟, 𝒞, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(𝒟)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, 𝒞)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



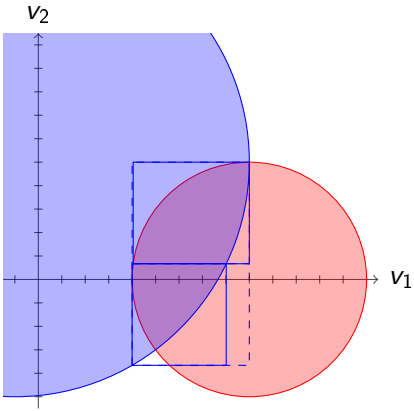
# Abstract solving method

```
1: function solve(𝒟, 𝒞, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(𝒟)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, 𝒞)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



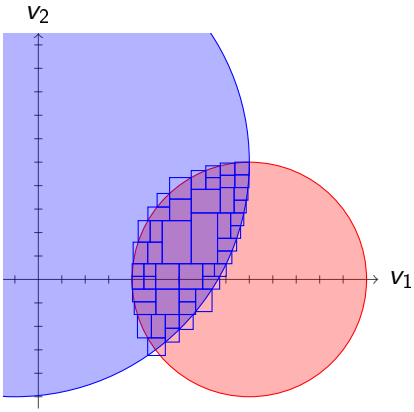
# Abstract solving method

```
1: function solve(𝒟, 𝒞, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(𝒟)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, 𝒞)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



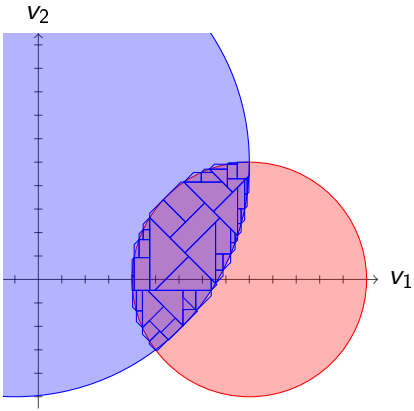
# Abstract solving method

```
1: function solve(𝒟, 𝒞, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(𝒟)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, 𝒞)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



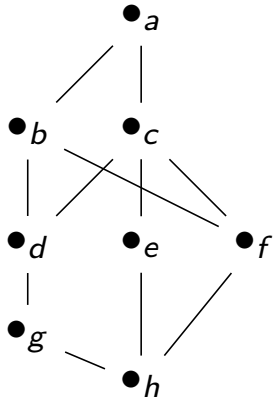
# Abstract solving method

```
1: function solve(𝒟, 𝒞, r)
2:     cover ← ∅                                  ▷ solutions
3:     explore ← ∅                                ▷ elements to explore
4:     e = init(𝒟)                                ▷ initialization
5:     push e in explore
6:     while explore ≠ ∅ do
7:         e ← pop(explore)
8:         e ← filter(e, 𝒞)
9:         if e ≠ ∅ then
10:            if τ(e) ≤ r then
11:                cover ← cover ∪ e
12:            else
13:                push ⊕(e) in explore
```



# Problem 3: linear extensions of posets

Joint work with O. Bodini, A. Genitrini and F. Peschanski



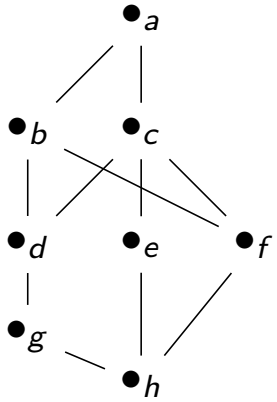

**Linear extensions**

- $a, b, c, d, e, f, g, h$
- $a, c, b, d, g, e, f, h$
- ...

**Total :** 27 linear extentions

# Problem 3: linear extensions of posets

Joint work with O. Bodini, A. Genitrini and F. Peschanski




## Linear extensions

- $a, b, c, d, e, f, g, h$
- $a, c, b, d, g, e, f, h$
- ...

**Total :** 27 linear extentions

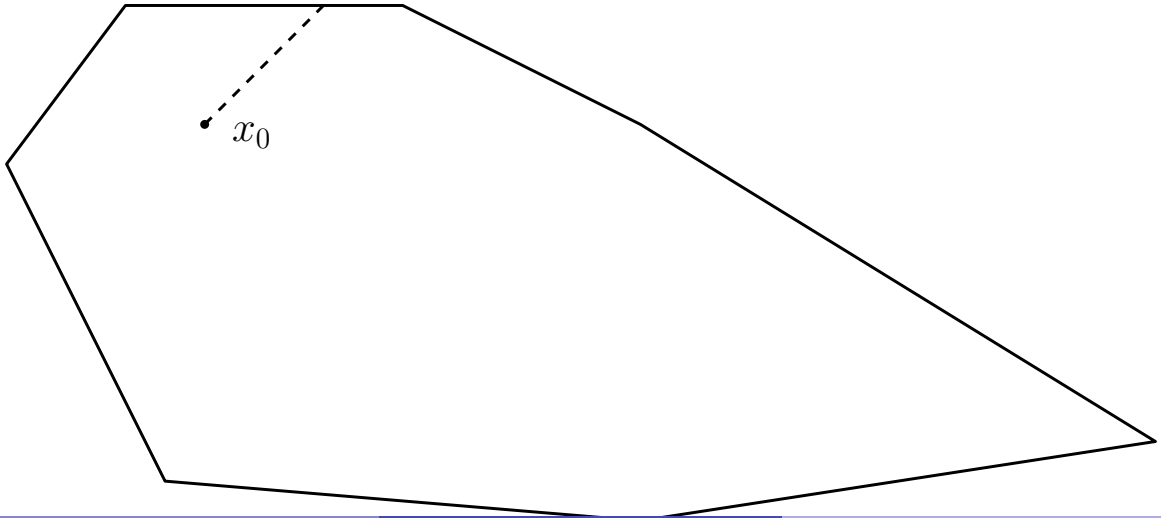
## Applications

- Concurrency theory
- Learning of Bayesian networks
- ...

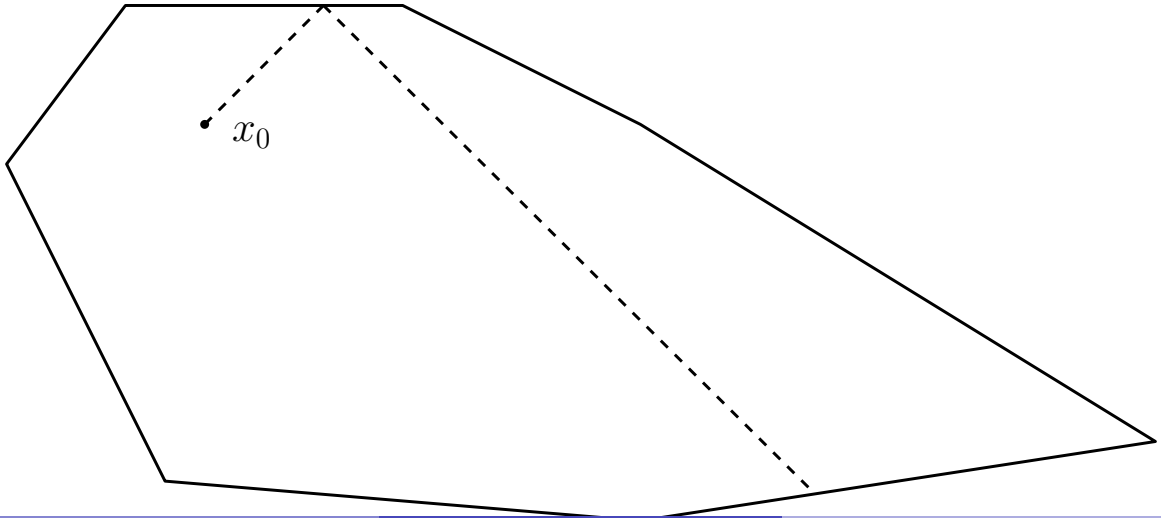
# Hit-and-Run

| **Markov Chain** | **Mixing Time** |
|---|---|
| Ball walk [Lovász'90] | $\mathcal{O}^*\left(n^3\right)$ after a warm start |
| *(Line)* Hit-and-run [Smith'84] | $\mathcal{O}^*\left(n^3\right)$ |
| Billiard walk [Polyak, Gryazina'14] | |
| *(Mirror)* Hit-and-Run [Van den Meersche *et al.* '09] | ? |

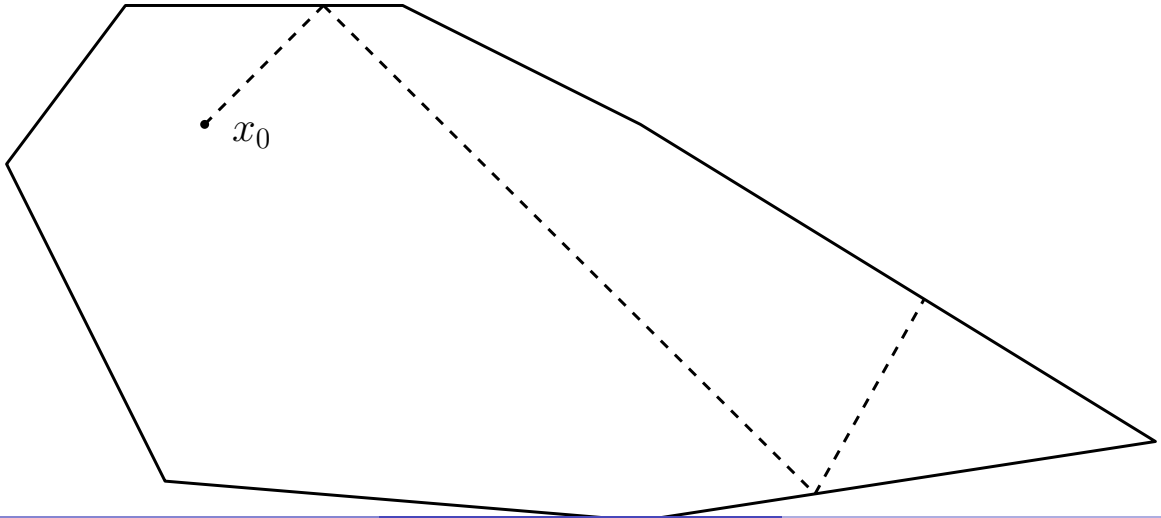
# Mirror Hit-and-Run



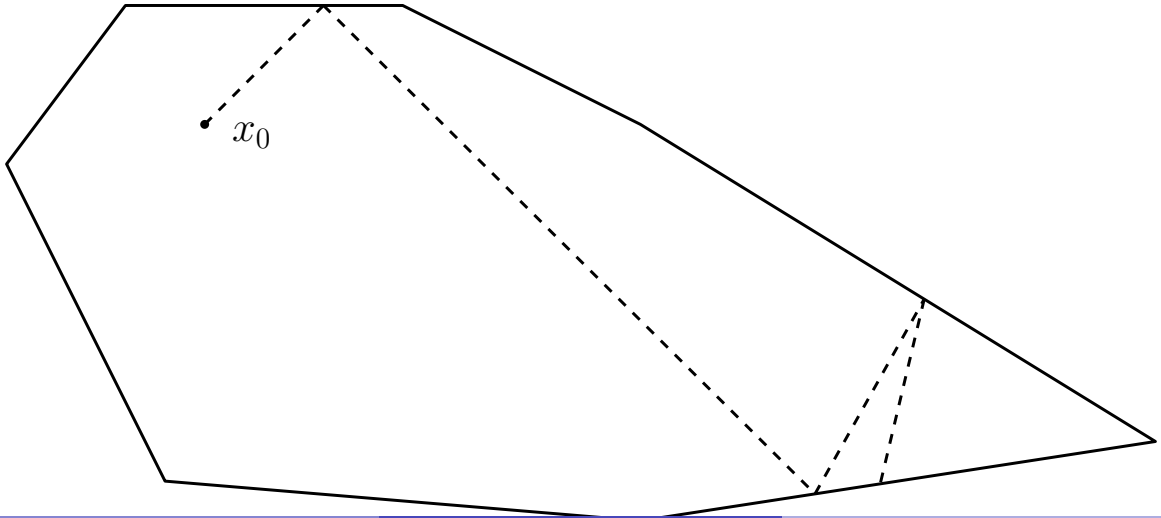
# Mirror Hit-and-Run



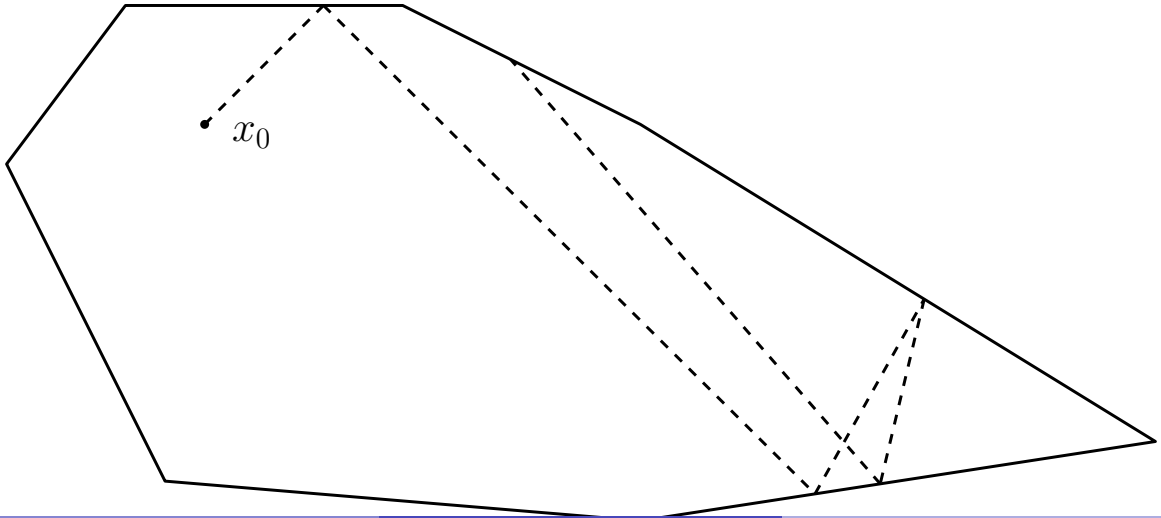
# Mirror Hit-and-Run



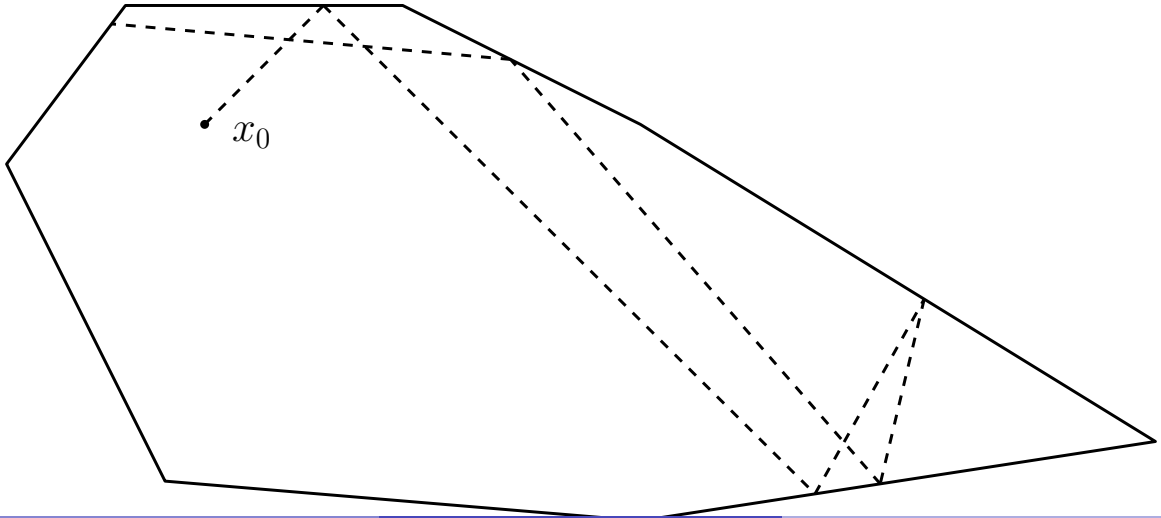
# Mirror Hit-and-Run



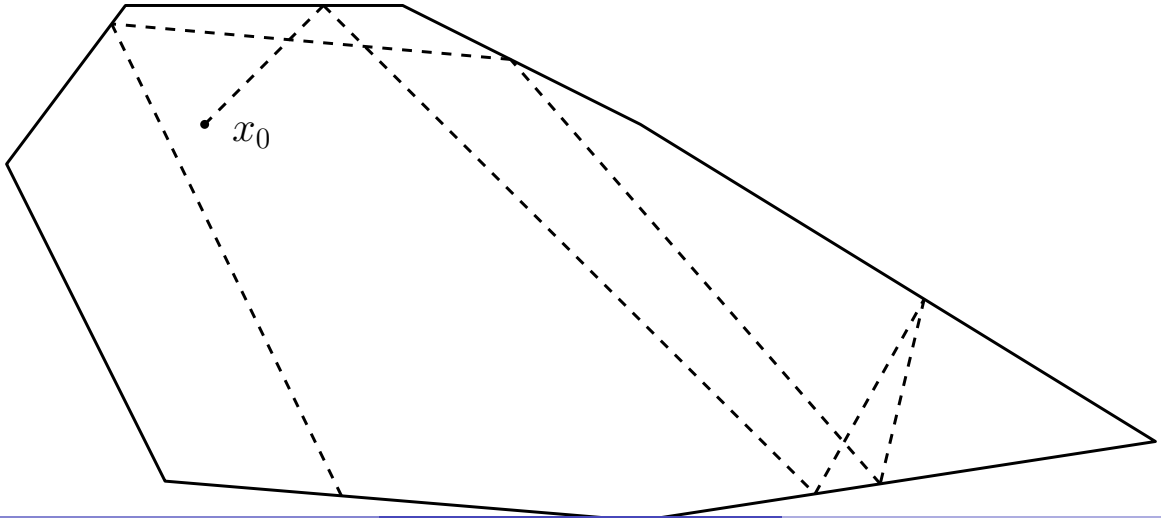
# Mirror Hit-and-Run



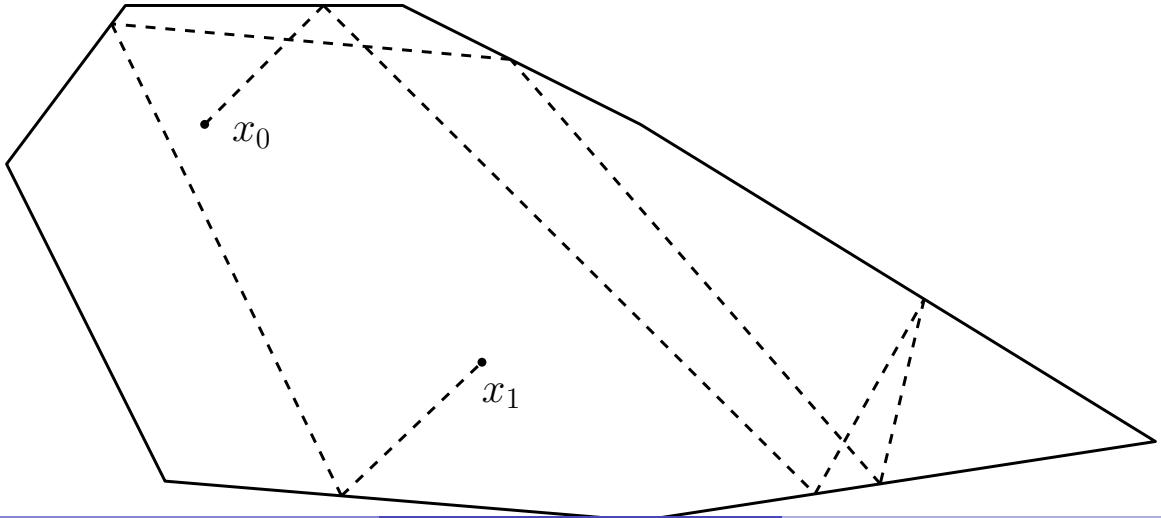
# Mirror Hit-and-Run



# Mirror Hit-and-Run



# Mirror Hit-and-Run



# The algorithm

- Find a point $x_0$ inside the polytope
- $i \leftarrow 0$
- While you want :
  - Sample a vector $\delta$ from a multidimensional Gaussian distribution $\mathcal{N}(0, \sigma)$
  - Throw $x_i$ in the direction $\delta$ and bounce when you meet a wall
  - The arrival point is $x_{i+1}$
  - Increment $i$
- Return $x_i$

# The algorithm

- Find a point $x_0$ inside the polytope
- $i \leftarrow 0$
- While you want :
  - Sample a vector $\delta$ from a multidimensional Gaussian distribution $\mathcal{N}(0, \sigma)$
  - Throw $x_i$ in the direction $\delta$ and bounce when you meet a wall
  - The arrival point is $x_{i+1}$
  - Increment $i$
- Return $x_i$

**Our job** (until now)

- A correct implementation of the algorithm in C with a R interface (for the biologists)
- An implementation of the tuning procedure to compute "a good" $\sigma$

# Open questions

- What is the mixing time of the "Mirror" Hit-and-run ?
- Can we exploit the geometry of food webs to accelerate the mixing ?
- Can we compute or estimate the makrginal distribution of each flow ?

# A particular case: Series-Parallel posets

## Definition

Let $P = (X_P, <_P)$ and $Q = (X_Q, <_Q)$ two posets, assuming $X_P \cap X_Q = \{\top, \bot\}$, i.e disjointness. The series and parallel operators are, respectively:

$$\begin{cases} P \odot Q = (X_P \cup X_Q, <_P \cup <_Q \cup (X_P \times X_Q)) \\ P \parallel Q = (X_P \cup X_Q, <_P \cup <_Q) \end{cases}$$

A (finite) poset is Series-Parallel (SP) if it can be obtained uniquely by (finite) compositions of series or parallel operators from singleton sets.

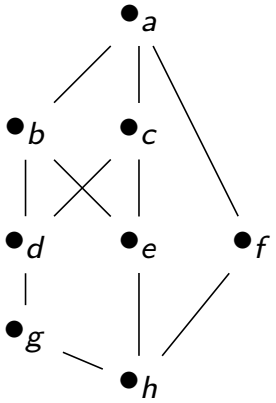
## Möhring's formula

$$\sharp\ell e(P \odot Q) = \sharp\ell e(P) \cdot \sharp\ell e(Q)$$
$$le(P \parallel Q) = \binom{|P|+|Q|}{|P|} \cdot \sharp\ell e(P) \cdot \sharp\ell e(Q)$$

## Random generation

A recursive and optimal algorithm in [Bodini, Dien, Genitrini, Peschanski, CSR'17]

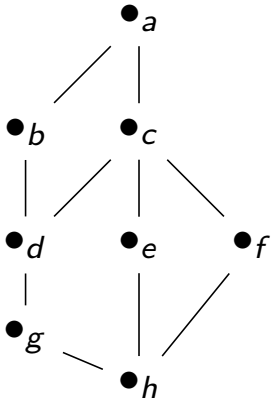
# A particular case: Series-Parallel posets



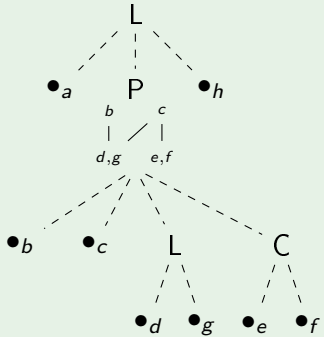

## Decomposition

$$\{a\} \odot \quad ((((\{b\} \parallel \{c\}) \odot ((\{d\} \odot \{g\}) \parallel \{e\})) \parallel \{f\}) \odot \{h\}$$

# A general framework: modular decomposition




## Modular decomposition tree



# Counting (and Sampling) with modular decomposition

## Ideas

- A bottom-up recursive scheme

# Counting (and Sampling) with modular decomposition

## Ideas

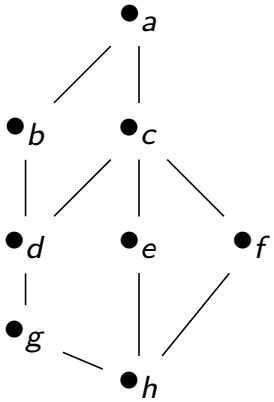
- A bottom-up recursive scheme
- For **L**inear (Series) and **C**omplete (Parallel) nodes, the Möhring's formula do the job

# Counting (and Sampling) with modular decomposition

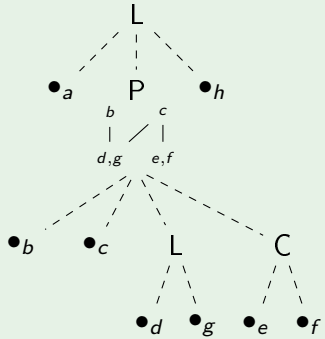
## Ideas

- A bottom-up recursive scheme
- For **L**inear (Series) and **C**omplete (Parallel) nodes, the Möhring's formula do the job
- For **P**rime nodes, use your favourite black-box algorithm, then proceed by substitution

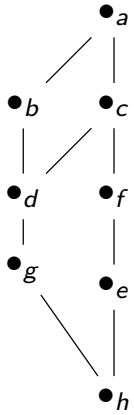
# Example



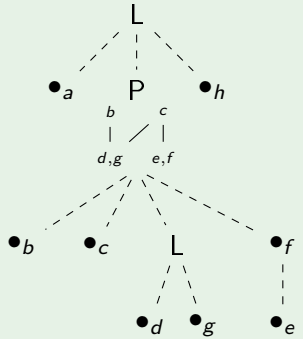

## Modular decomposition tree



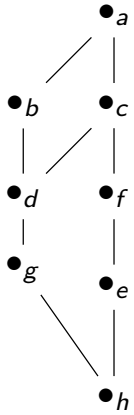
# Example



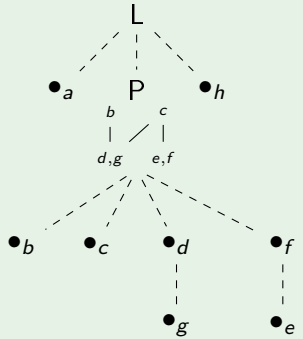

## Modular decomposition tree



# Example




## Modular decomposition tree



# Example

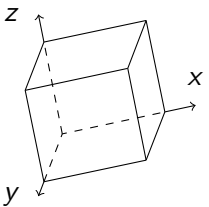
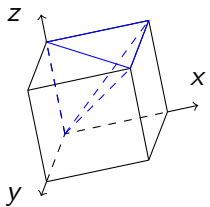
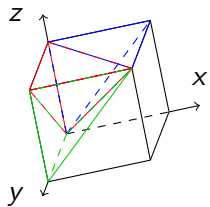
Finish with "black-box" sampling

$$\bullet_a \text{ — } \bullet_c \text{ — } \bullet_b \text{ — } \bullet_d \text{ — } \bullet_f \text{ — } \bullet_e \text{ — } \bullet_g \text{ — } \bullet_h$$

## Black-box

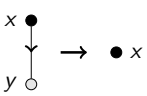
- Enumerate all linear extensions [Pruesse, Ruskey'94] (constant armotized time)
- Use your favorite MCMC method
- Check the next slides

# Order polytope [Stanley'86]

| partial order | $x \quad y \quad z$ | $z \leftarrow x \leftarrow y$ | $z \leftarrow x \quad y$ |
|---|---|---|---|
| volume |   $3! \int_0^1 \int_0^1 \int_0^1 \mathrm{d}z\ \mathrm{d}x\ \mathrm{d}y = 6$ |   $3! \int_0^1 \int_0^y \int_0^x \mathrm{d}z\ \mathrm{d}x\ \mathrm{d}y = 1$ |   $3! \int_0^1 \int_0^1 \int_0^x \mathrm{d}z\ \mathrm{d}x\ \mathrm{d}y = 3$ |

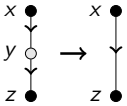
# BITS decomposition

(B)ottom



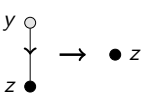

$\Psi' = \int_x^1 \Psi.dy$

(I)ntermediate



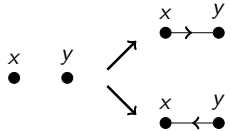

$\Psi' = \int_x^z \Psi.dy$

(T)op




$\Psi' = \int_0^z \Psi.dy$

(S)plit

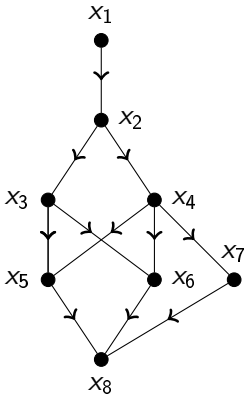



$\Psi' = \Psi_{x \prec y} + \Psi_{y \prec x}$

**Theorem**

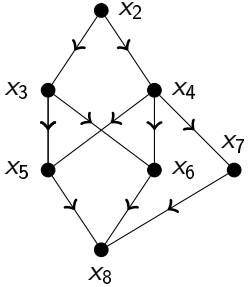
- BITS decomposition computes the volume of **any** poset.
- The number of symbolic integration may be **exponential** in the size of the poset.
- The number of symbolic integration is **linear** in the size of posets **without (S)plit**.

# Example



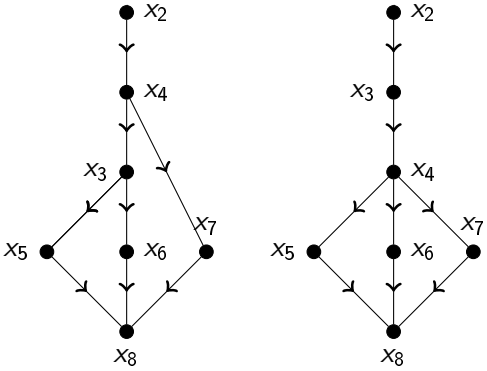

$$\Psi = 1$$

# Example



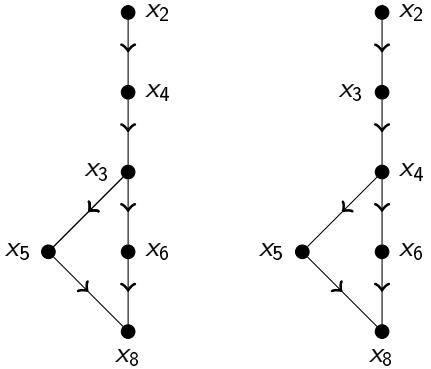

$$\Psi' = \int_0^{x_2} \Psi \mathrm{d}x_1$$

# Example




$$\Psi'' = \Psi'_{x_3 \prec x_4} + \Psi'_{x_4 \prec x_3}$$

# Example




$$\Psi''' = \int_{x_4}^{x_8} \Psi''_{x_4 \prec x_3} \mathrm{d}x_7 + \int_{x_4}^{x_8} \Psi''_{x_3 \prec x_4} \mathrm{d}x_7$$

## Example

$$
\begin{aligned}
\Psi &= \int_0^{x_2} \Psi' \mathrm{d}x_1 \\
&= \int_0^{x_2} \left( \Psi''_{\{x_4 \prec x_3\}} + \Psi''_{\{x_3 \prec x_4\}} \right) \mathrm{d}x_1 \\
&= \int_{x_3}^{x_8} \int_0^{x_2} \Psi'''_{\{x_4 \prec x_3\}} \mathrm{d}x_1 \mathrm{d}x_5 + \int_{x_4}^{x_8} \int_0^{x_2} \Psi'''_{\{x_3 \prec x_4\}} \mathrm{d}x_1 \mathrm{d}x_5 \\
&= \int_{x_4}^{x_8} \int_{x_3}^{x_8} \int_0^{x_2} \Psi^{(4)}_{\{x_4 \prec x_3\}} \mathrm{d}x_1 \mathrm{d}x_5 \mathrm{d}x_7 + \int_{x_4}^{x_8} \int_{x_4}^{x_8} \int_0^{x_2} \Psi^{(4)}_{\{x_3 \prec x_4\}} \mathrm{d}x_1 \mathrm{d}x_5 \mathrm{d}x_7 \\
&= \int_0^1 \int_{x_2}^1 \int_{x_4}^1 \int_{x_3}^1 \int_{x_6}^1 \int_{x_4}^{x_8} \int_{x_3}^{x_8} \int_0^{x_2} \mathrm{d}x_1 \mathrm{d}x_5 \mathrm{d}x_7 \mathrm{d}x_8 \mathrm{d}x_6 \mathrm{d}x_3 \mathrm{d}x_4 \mathrm{d}x_2 \\
&\quad + \int_0^1 \int_{x_2}^1 \int_{x_3}^1 \int_{x_4}^1 \int_{x_6}^1 \int_{x_4}^{x_8} \int_{x_4}^{x_8} \int_0^{x_2} \mathrm{d}x_1 \mathrm{d}x_5 \mathrm{d}x_7 \mathrm{d}x_8 \mathrm{d}x_6 \mathrm{d}x_3 \mathrm{d}x_4 \mathrm{d}x_2 \\
&= \frac{8+6}{8!} = \frac{14}{8!}.
\end{aligned}
$$

# Random generation

## Discrete to continuous

random linear extension $\simeq$ random simplex $\simeq$ random point of the polytope

# Random generation

## Key idea

If $f$ is a continuous function from $[a, b]$ such that

$$\int_a^b f(x)\,\mathrm{d}x = C,$$

then $x \mapsto \frac{f(x)}{C}$ is the density of a random variable over $[a, b]$.

# Random generation

## Key idea

If $f$ is a continuous function from $[a, b]$ such that

$$\int_a^b f(x)\,\mathrm{d}x = C,$$

then $x \mapsto \frac{f(x)}{C}$ is the density of a random variable over $[a, b]$.

## Inversion method

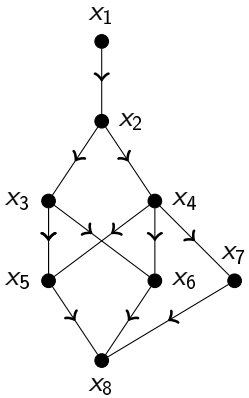
Let $U$ be a r.v. of uniform law over $[a, b]$, then the solution $Y$ of

$$\int_a^Y f(x)\,\mathrm{d}x = U,$$

is a r.v. of density $f$.

# Example

$$\Psi = \Psi_{\{x_4 \prec x_3\}} + \Psi_{\{x_3 \prec x_4\}} = \frac{8+6}{8!}$$



# Example

## Volume

$$\Psi_{\{x_4 \prec x_3\}} = \int_0^1 \int_{x_2}^1 \int_{x_4}^1 \int_{x_3}^1 \int_{x_6}^1 \int_{x_4}^{x_8} \int_{x_3}^{x_8} \int_0^{x_2} \mathrm{d}x_1 \mathrm{d}x_5 \mathrm{d}x_7 \mathrm{d}x_8 \mathrm{d}x_6 \mathrm{d}x_3 \mathrm{d}x_4 \mathrm{d}x_2$$

$$= \int_0^1 \frac{1}{453600} (x_2^6 - 6\,x_2^5 + 15\,x_2^4 - 20\,x_2^3 + 15\,x_2^2 - 6\,x_2 + 1) x_2 \ \mathrm{d}x_2$$

## Equation of inversion

$$\int_0^{x_2} \frac{1}{90} (x_2^6 - 6\,x_2^5 + 15\,x_2^4 - 20\,x_2^3 + 15\,x_2^2 - 6\,x_2 + 1) x_2 \ \mathrm{d}x_2 = 0.13$$

## Sampled point

$$x_2 = 0.08$$

# Example

## Equation of inversion

$$\int_{x_2}^{x_4} -\frac{1}{15}\,(x_4^5 - 5\,x_4^4 + 10\,x_4^3 - 10\,x_4^2 + 5\,x_4 - 1)x_2 \;\mathrm{d}x_4 = 0.84$$

## Sampled point

$$x_2 = 0.08$$
$$x_4 = 0.32$$

# Example

## Equation of inversion

$$\int_{x_4}^{x_3} -\frac{1}{12}\left(x_3^4 - 6\,x_3^2 - 2\,(2\,x_3^3 - 6\,x_3^2 + 3\,x_3)x_4 - 2\,(3\,x_3 - 2)x_4 + 8\,x_3 - 3)x_2\right)\,\mathrm{d}x_3 = 0.76$$

## Sampled point

$$x_2 = 0.08$$
$$x_4 = 0.32$$
$$x_3 = 0.54$$

# Example

## Equation of inversion

$$\int_{x_3}^{x_6} -\frac{1}{6}\left(6\,x_3x_4x_6 - 3\,(x_3+x_4)x_6^2 + 2\,x_6^3 - 3\,(2\,x_3-1)x_4 + 3\,x_3 - 2\right)x_2\;\mathrm{d}x_6 = 0.25$$

## Sampled point

$$x_2 = 0.08$$
$$x_4 = 0.32$$
$$x_3 = 0.54$$
$$x_6 = 0.62$$

# Example

## Sampled point

$$x_2 = 0.08$$
$$x_4 = 0.32$$
$$x_3 = 0.54$$
$$x_6 = 0.62$$
$$x_8 = 0.89$$
$$x_7 = 0.58$$
$$x_5 = 0.76$$
$$x_1 = 0.06$$

## Linear extension

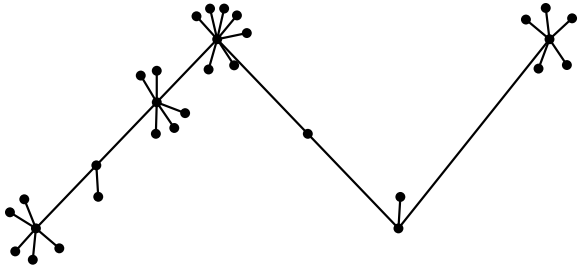
$$x_1 \prec x_2 \prec x_4 \prec x_3 \prec x_7 \prec x_6 \prec x_5 \prec x_8$$

## Some algorithmic aspects

- We can characterize some "good" orders of BIT-decomposition with a parameter called the *BIT-width*
  - If the BIT-width is bounded by $k$ then the counting algorithm runs in $\mathcal{O}(n^{k+1})$
  - Posets of BIT-width 1 are caterpillar posets

# Some algorithmic aspects

- We can characterize some "good" orders of BIT-decomposition with a parameter called the *BIT-width*
  - If the BIT-width is bounded by $k$ then the counting algorithm runs in $\mathcal{O}(n^{k+1})$
  - Posets of BIT-width 1 are caterpillar posets
- Going back to modular decomposition
  - If the prime nodes lives in a finite sets of patterns then the algorithm is polynomial
  - $\Rightarrow$ The counting of linear extensions is $\mathcal{O}(n)$ for $N$-extendible posets (related to $P_4$-extendible graphs)



# Open problems

## Conjectures

The following decision problem is NP-complete:

**BIT-WIDTH**

**Input:** a BIT-decomposable poset $P$, an integer $k \geq 3$

**Output:** $w(P) \leq k$

The following counting problem is $\sharp P$-complete:

**BIT-MODULAR-COUNT**

**Input:** a BIT-modular poset $P$

**Output:** $\sharp \ell e(P)$

## Other questions

- How to evaluate the integral formula fastly ?
- Is a greedy algorithm a good approximation to compute the BIT-order ?