Counting Linear Extensions of Sparse Posets

Kustaa Kangas

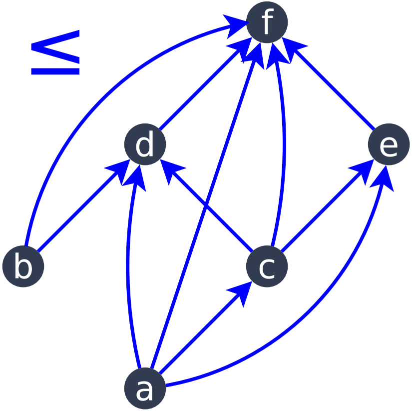
October 20, 2016

Papers

- Kustaa Kangas, Teemu Hankala, Teppo Niinimäki, and Mikko Koivisto. Counting linear extensions of sparse posets, IJCAI'16
- Eduard Eiben, Robert Ganian, Kustaa Kangas, and Sebastian Ordyniak. Counting linear extensions: Parameterizations by treewidth, ESA'16

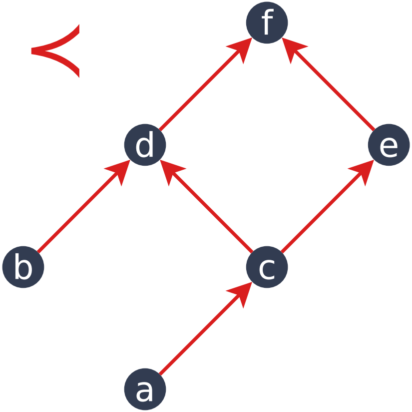
Partially ordered set (poset)

A finite set + a reflexive, antisymmetric, transitive relation



Partially ordered set (poset)

Cover relation / cover graph (transitive reduction)

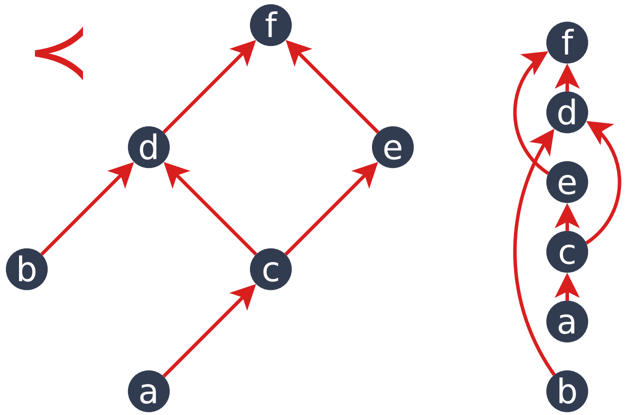


Linear extensions

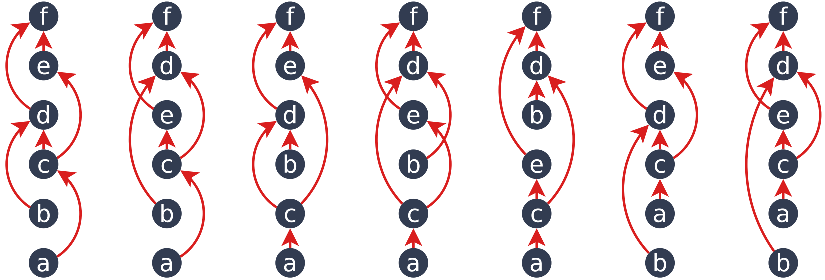
Linear order: all pairs are comparable

Linear extensions

A *linear extension* = order preserving permutation



Counting linear extensions



Determining the number of linear extensions of a given poset is #P-complete (Brightwell & Winkler, ’91)

(by reduction from #3SAT)

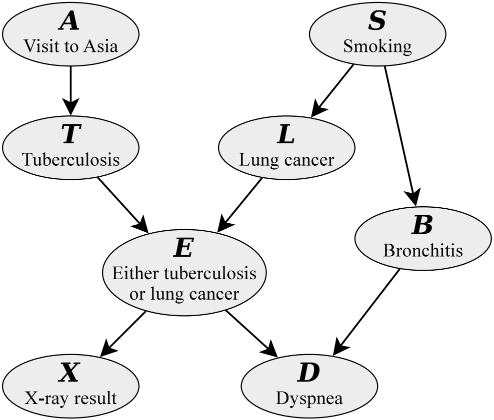
Motivation

Classic application: sorting

Other uses: preference reasoning, planning, convex rank tests, sequence analysis, ...

Motivation

Sampling Bayesian networks from a posterior distribution



Motivation

Markov Chain Monte Carlo

- States are DAGs
- Stationary distribution is the posterior

Order MCMC:

- States are linear orders
- Sample first an order, then a compatible DAG
- Faster mixing but requires bias correction via counting linear extensions of sampled DAGs

Known algorithms

Trivial solution: enumerate all orders (factorial time)

The best we can do is $O(2^n n)$ time for n elements

Polynomial time for special cases:

- Polytrees
- Series-parallel
- Bounded width
- Bounded decomposition diameter
- N-free orders of bounded activity

A fpras also exists

First paper

We give two algorithms, exploiting sparsity of the poset

1. recursion (exploiting low connectivity)
2. variable elimination (exploiting low treewidth)

First paper

We give two algorithms, exploiting sparsity of the poset

1. recursion (exploiting low connectivity)
2. variable elimination (exploiting low treewidth)

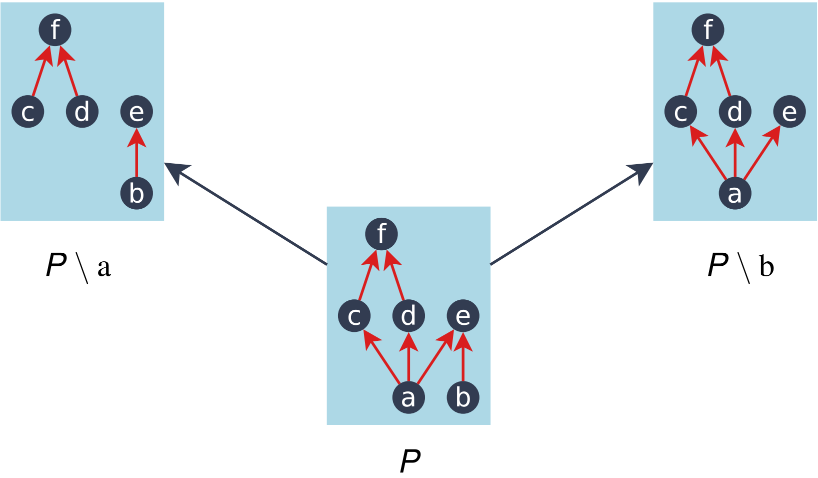
Recursive counting

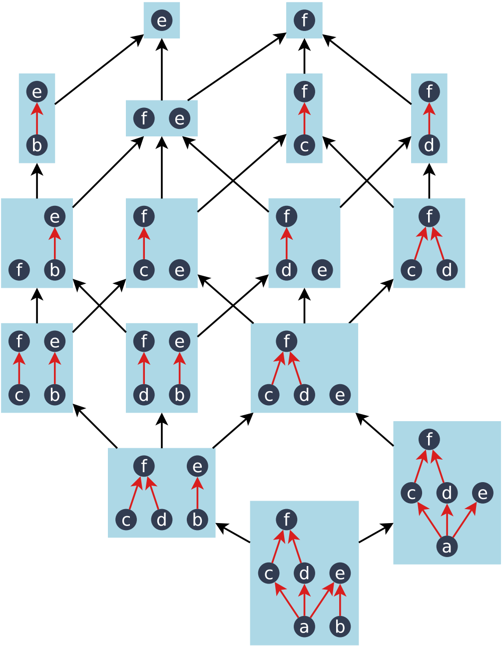
$\ell(P)$ = the number of linear extensions of poset P

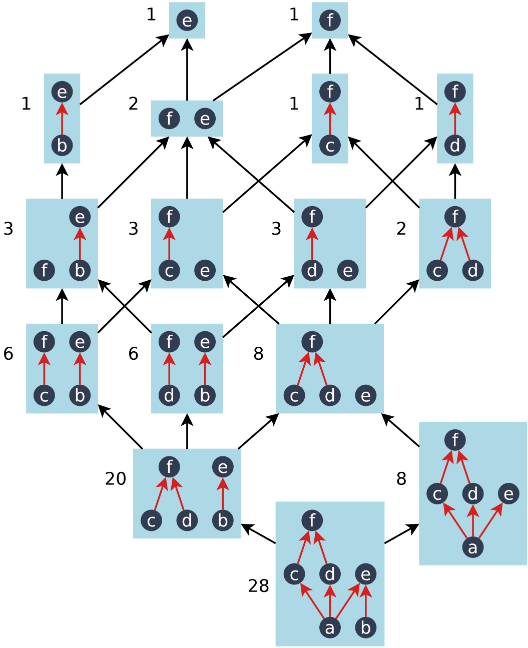
A simple observation:

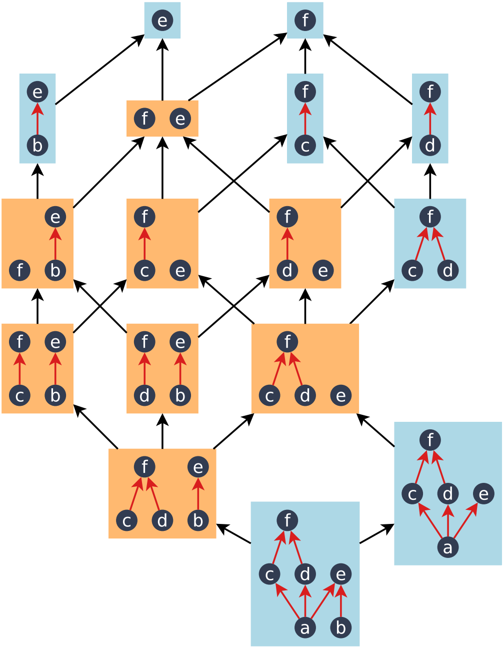
$$\ell(P) \;=\; \sum_{x \,\in\, \min(P)} \ell(P \setminus x)$$

Recursive counting




e
f
e
b
f
e
f
c
f
d
e
f
b
f
c
e
f
d
e
f
c
d
f
e
c
b
f
e
d
b
f
c
d
e
f
e
c
d
b
f
c
d
e
a
f
c
d
e
a
b


1
1
1
2
1
1
3
3
3
2
6
6
8
20
8
28

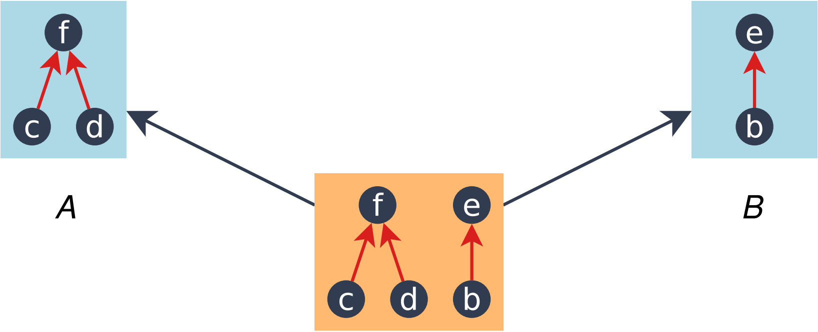

e
f
e b
f e
f c
f d
e f b
f c e
f d e
f c d
f e c b
f e d b
f c d e
f e c d b
f c d e a
f c d e a b

Recursive counting

If A and B partition P and are mutually disconnected, then

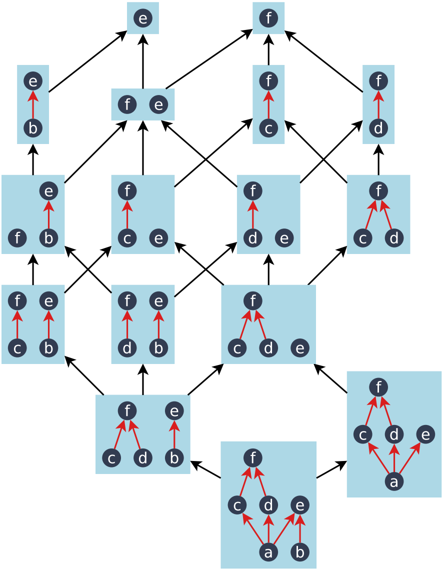
$$\ell(P) \;=\; \ell(A) \cdot \ell(B) \cdot \binom{|P|}{|A|}$$

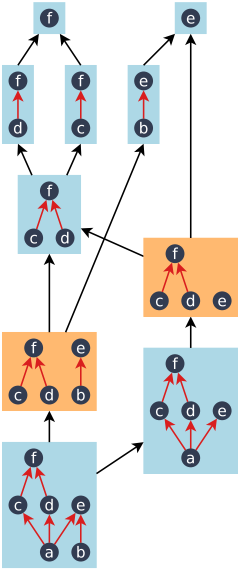
Recursive counting

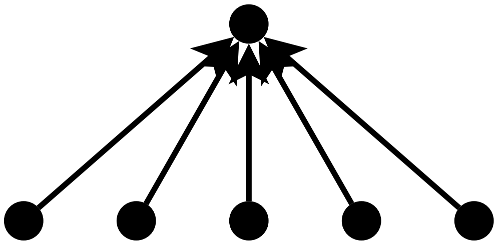


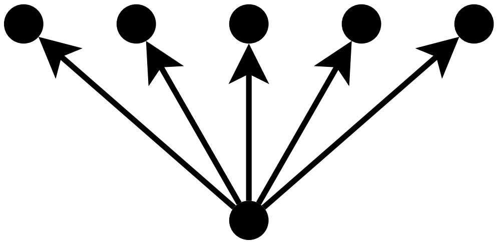
A B

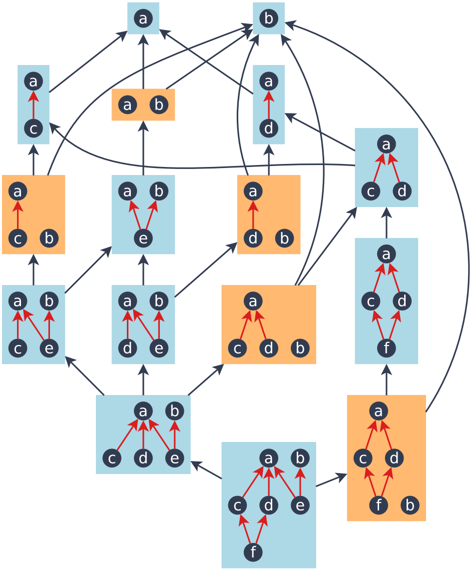
P

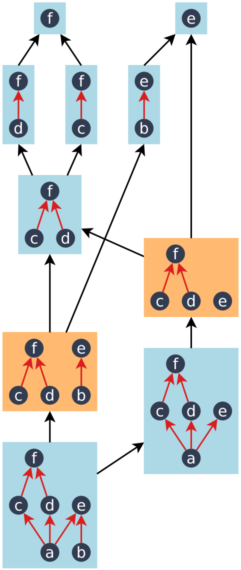












Recursive counting

Deciding whether to transpose is not trivial.

We consider two heuristics

1. Only count minimal and maximal elements
2. Estimate the size of subproblem space recursively

In practice both heuristics almost always make the better choice.

Recursive counting

$\ell(P)$ = the number of linear extensions of poset P

Rule 1 $\quad \ell(P) = \sum_{x \in \min(P)} \ell(P \setminus x)$

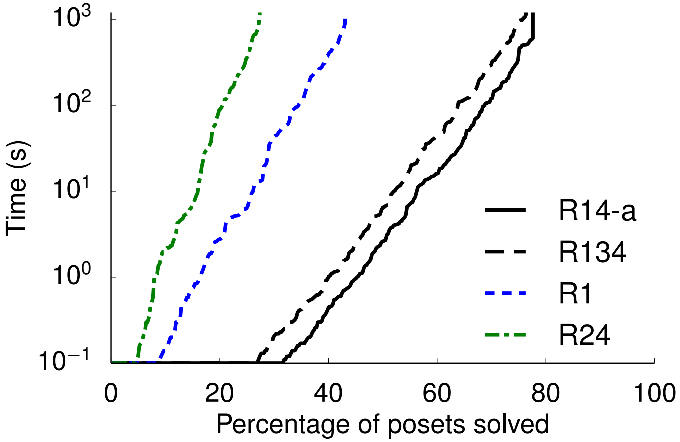
Rule 2 $\quad \ell(P) = \sum_{(D,U)} \ell(D) \cdot \ell(U)$

Rule 3 $\quad \ell(P) = \prod_{i=1}^{k} \ell(S_i)$

Rule 4 $\quad \ell(P) = \ell(A) \cdot \ell(B) \cdot \binom{|P|}{|A|}$

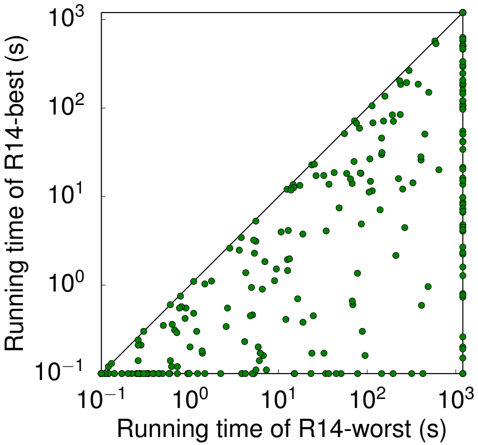
Recursive counting

Experiments on sparse posets for $n = 30, \ldots, 100$



Recursive counting

Experiments on sparse posets for $n = 30, \ldots, 100$



First paper

We give two algorithms, exploiting sparsity of the poset

1. recursion (exploiting low connectivity)
2. variable elimination (exploiting low treewidth)

Variable elimination

$$\sum_{a,b,c,d,e,f} \phi_1(a,b,d)\,\phi_2(a,c)\,\phi_3(b,c,e)\,\phi_4(d,f)$$

Variable elimination

$$\sum_{a,b,c,d,e} \phi_1(a,b,d)\, \phi_2(a,c)\, \phi_3(b,c,e) \sum_f \phi_4(d,f)$$

Variable elimination

$$\sum_{a,b,c,d,e} \phi_1(a,b,d)\, \phi_2(a,c)\, \phi_3(b,c,e)\, \lambda_1(d)$$

Variable elimination

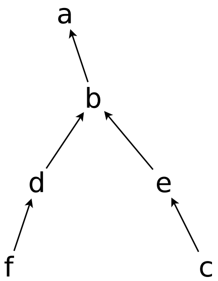
$$\sum_{a,b,c,e} \phi_2(a,c)\, \phi_3(b,c,e) \sum_{d} \phi_1(a,b,d)\, \lambda_1(d)$$

Variable elimination

$$\sum_{a,b,c,e} \phi_2(a,c)\,\phi_3(b,c,e)\,\lambda_2(a,b)$$

Variable elimination

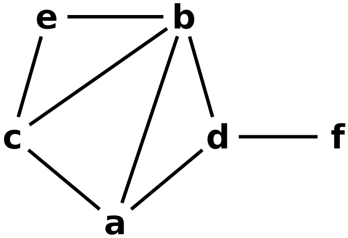
$$\sum_a \left(\sum_b \left(\left(\sum_e \phi_3(b,c,e) \left(\sum_c \phi_2(a,c) \right) \right) \left(\sum_d \phi_1(a,b,d) \left(\sum_f \phi_4(d,f) \right) \right) \right) \right)$$



Elimination order matters!

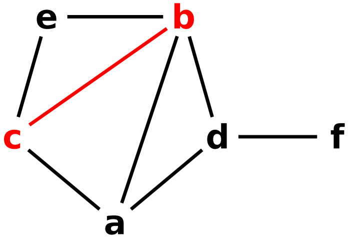
Variable elimination

$$\sum_{a,b,c,d,e,f} \phi_1(a,b,d)\,\phi_2(a,c)\,\phi_3(b,c,e)\,\phi_4(d,f)$$



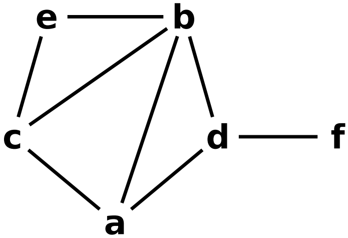
Variable elimination

$$\sum_{a,b,c,d,e,f} \phi_1(a,b,d)\,\phi_2(a,c)\,\phi_3(b,c,e)\,\phi_4(d,f)$$



Variable elimination

$$\sum_{a,b,c,d,e,f} \phi_1(a,b,d)\ \phi_2(a,c)\ \phi_3(b,c,e)\ \phi_4(d,f)$$



Polynomial time for bounded **treewidth**

Variable elimination

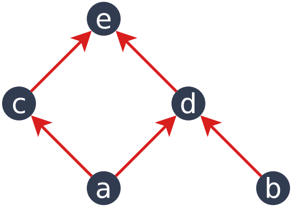
For every permutation $\sigma : P \to [n]$ define

$$\Phi(\sigma) = \prod_{x \prec y} [\sigma_x < \sigma_y]$$

Variable elimination

For every permutation $\sigma : P \to [n]$ define

$$\Phi(\sigma) = \prod_{x \prec y} [\sigma_x < \sigma_y]$$



$$\Phi(\sigma) = [\sigma_a < \sigma_c]\,[\sigma_a < \sigma_d]\,[\sigma_b < \sigma_d]\,[\sigma_c < \sigma_e]\,[\sigma_d < \sigma_e]$$

Variable elimination

For every permutation $\sigma : P \to [n]$ define

$$\Phi(\sigma) = \prod_{x \prec y} [\sigma_x < \sigma_y]$$

Then,

$$\Phi(\sigma) = \begin{cases} 1, \text{ if } \sigma \text{ is a linear extension,} \\ 0, \text{ otherwise.} \end{cases}$$

Variable elimination

For every permutation $\sigma : P \to [n]$ define

$$\Phi(\sigma) = \prod_{x \prec y} [\sigma_x < \sigma_y]$$

Then,

$$\Phi(\sigma) = \begin{cases} 1, \text{ if } \sigma \text{ is a linear extension,} \\ 0, \text{ otherwise.} \end{cases}$$

As a consequence

$$\ell(P) = \sum_{\substack{\sigma : P \to [n] \\ \text{bijection}}} \Phi(\sigma)$$

Variable elimination

$$\ell(P) = \sum_{\substack{\sigma : P \to [n] \\ \text{bijection}}} \Phi(\sigma)$$

Variable elimination

$$\ell(P) = \sum_{\substack{\sigma : P \to [n] \\ \text{bijection}}} \Phi(\sigma)$$

Can't apply variable elimination because of the bijectivity constraint.

Variable elimination

$$
\begin{aligned}
\ell(P) &= \sum_{\substack{\sigma : P \to [n] \\ \text{bijection}}} \Phi(\sigma) \\
&= \sum_{X \subseteq [n]} (-1)^{n-|X|} \sum_{\sigma : P \to X} \Phi(\sigma) \\
&= \sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} \sum_{\sigma : P \to [k]} \Phi(\sigma)
\end{aligned}
$$

Inclusion–exclusion principle

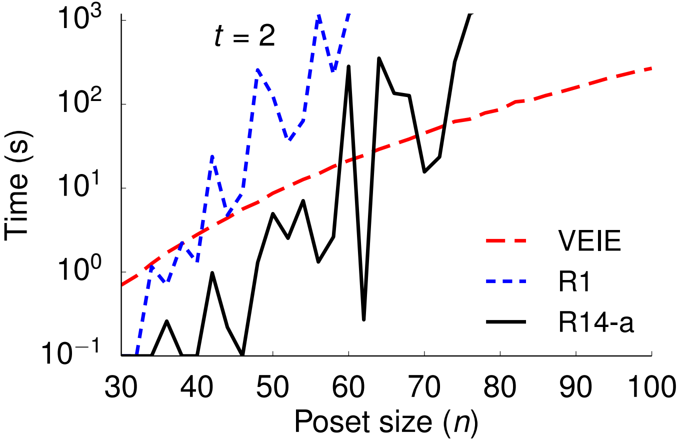
Variable elimination

$$\begin{aligned}
\ell(P) &= \sum_{\substack{\sigma : P \to [n] \\ \text{bijection}}} \Phi(\sigma) \\
&= \sum_{X \subseteq [n]} (-1)^{n-|X|} \sum_{\sigma : P \to X} \Phi(\sigma) \\
&= \sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} \sum_{\sigma : P \to [k]} \Phi(\sigma)
\end{aligned}$$

$O(n^{t+4})$ time for treewidth t

Variable elimination

VEIE: Variable elimination via inclusion–exclusion



Parameterized complexity

Let n be input size and k an additional numerical parameter of the input.

- **XP**: problems solvable in time $n^{f(k)}$
- **FPT**: problems solvable in time $f(k) \cdot n^{O(1)}$.

Problems in **FPT** are called *fixed-parameter tractable*.

Parameterized complexity

Results: Counting linear extensions is...

- **W**[1]-hard when parameterized by the treewidth of the cover graph
- in **FPT** when parameterized by the treewidth of the *incomparability graph*

A **W**[1]-hard problem is not in **FPT** unless **FPT** $=$ **W**[1].

Summary

- **Recursion:** often fast in practice
- **Variable elimination:** polynomial time for bounded treewidth

Thank you!