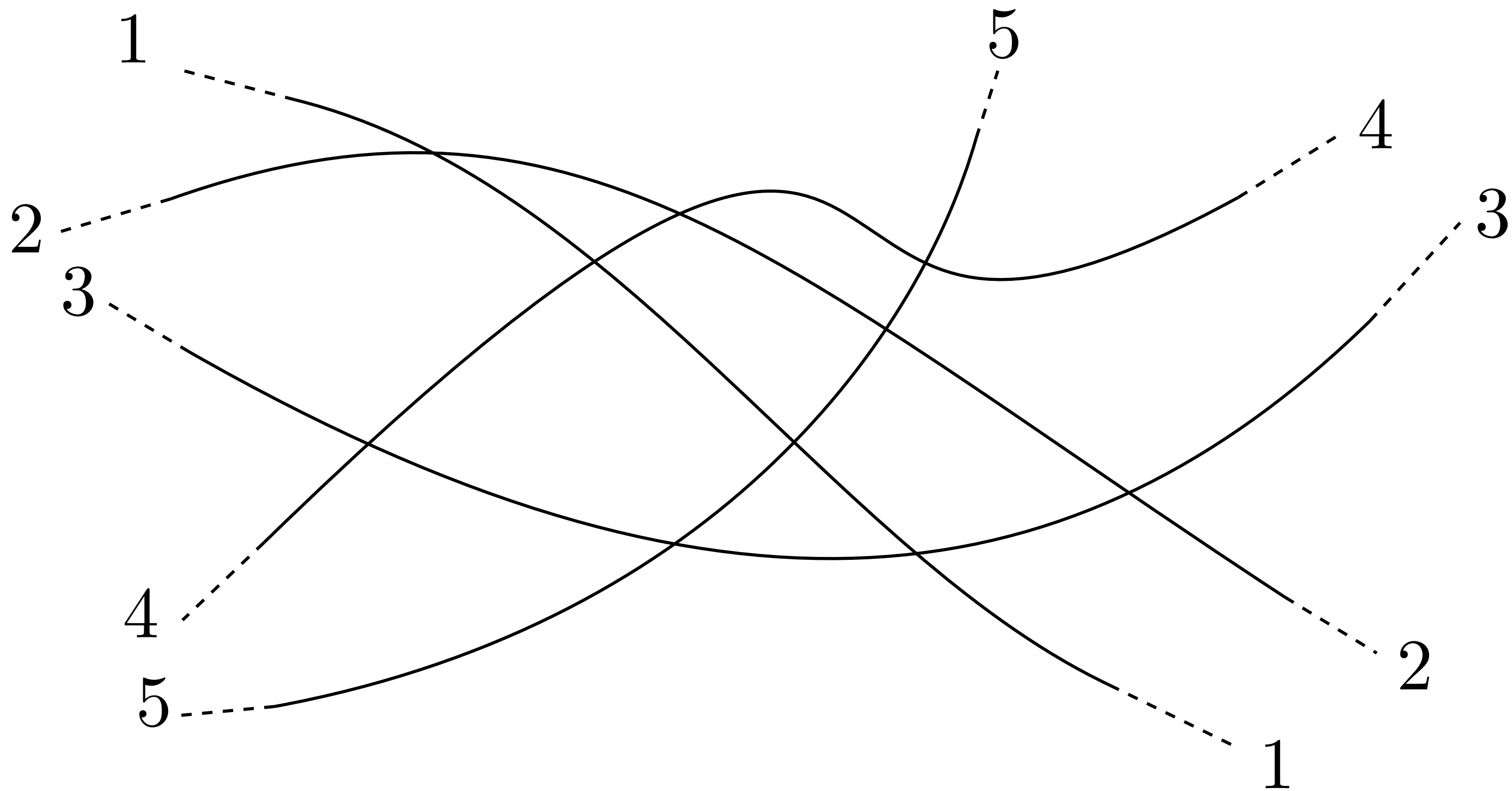
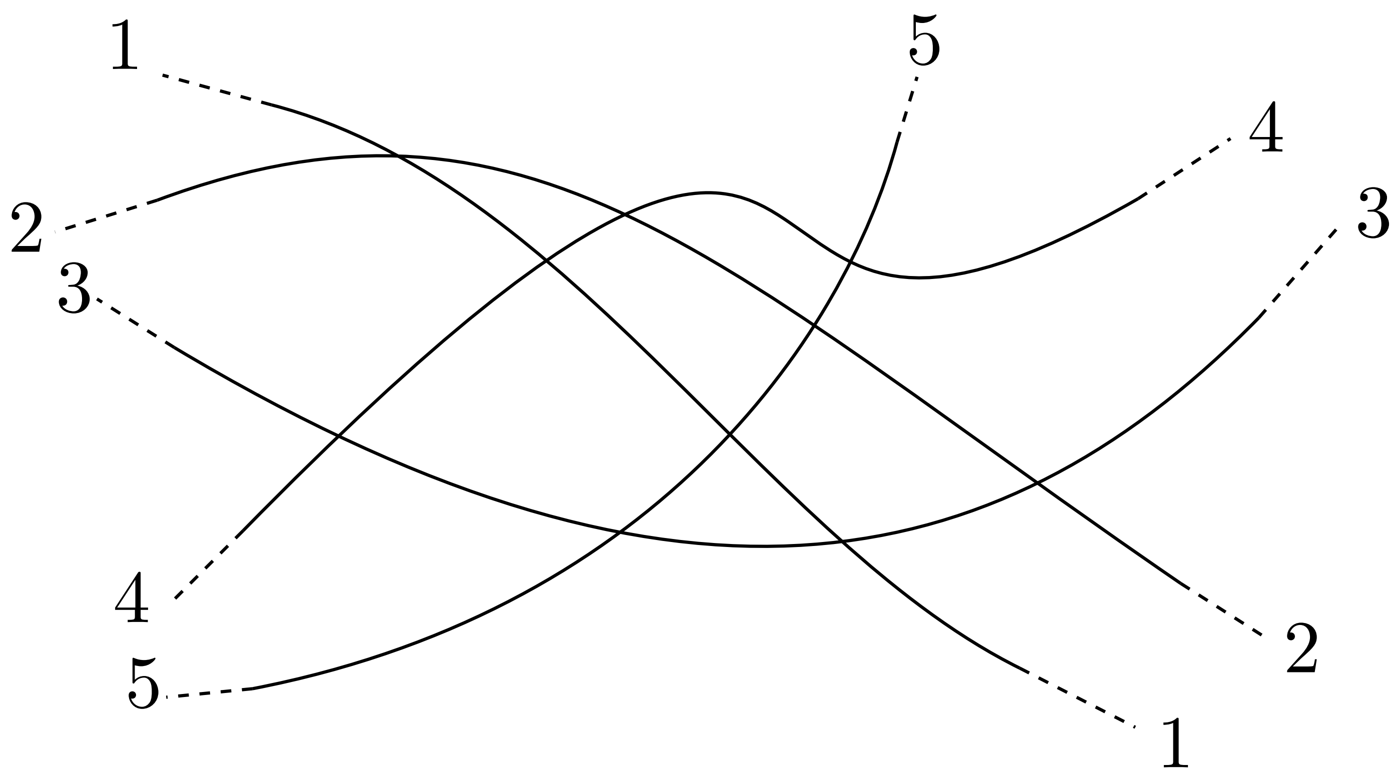
Enumeration and Counting of Pseudoline Arrangements

Günter Rote

Freie Universität Berlin



Pseudoline Arrangements



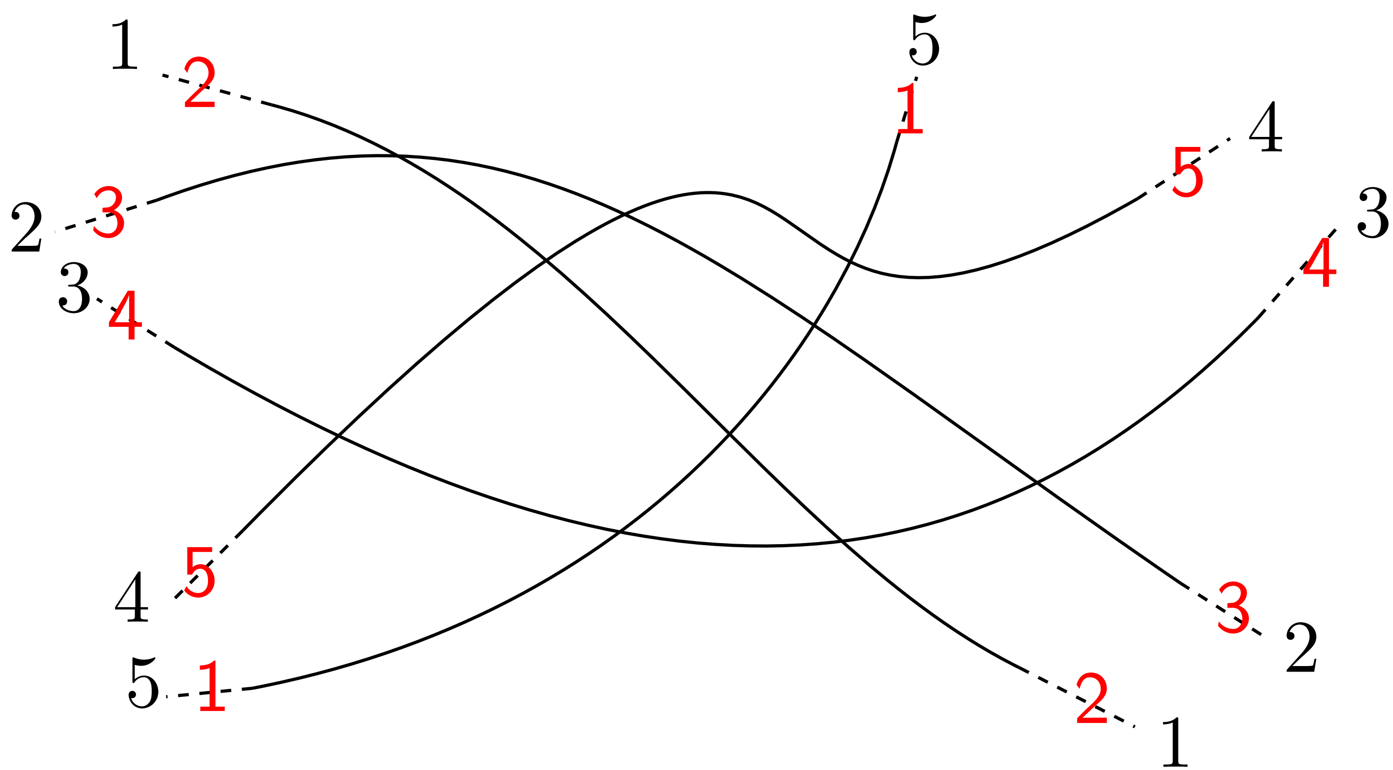
- n curves going to infinity
- Two curves intersect exactly once, and they cross.

- *simple* pseudoline arrangements: no multiple crossings
- x-monotone curves

Pseudoline Arrangements

Freie Universität Berlin

a *different* arrangement

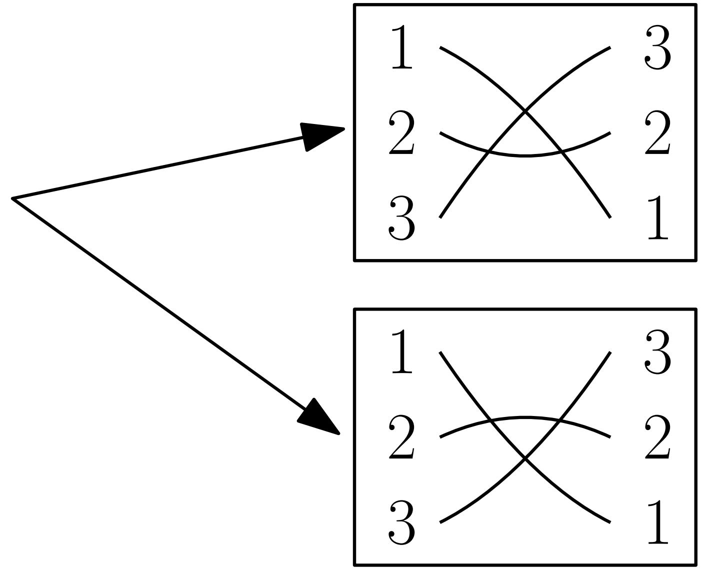


- n curves going to infinity
- Two curves intersect exactly once, and they cross.

- *simple* pseudoline arrangements: no multiple crossings
- x-monotone curves

How many pseudoline arrangements?

n	#PsA's with n pseudolines	
1	1	
2	1	
3	2	
4	8	
5	62	
6	908	
7	24698	
8	1232944	
9	112018190	
10	18410581880	
11	5449192389984	
12	2894710651370536	
13	2752596959306389652	
14	4675651520558571537540	[Yuma Tanaka, 2013]
15	14163808995580022218786390	[Yuma Tanaka, 2013]
16	7641307372577259323046193673 6	[G. Rote, 2021]



OEIS A006245

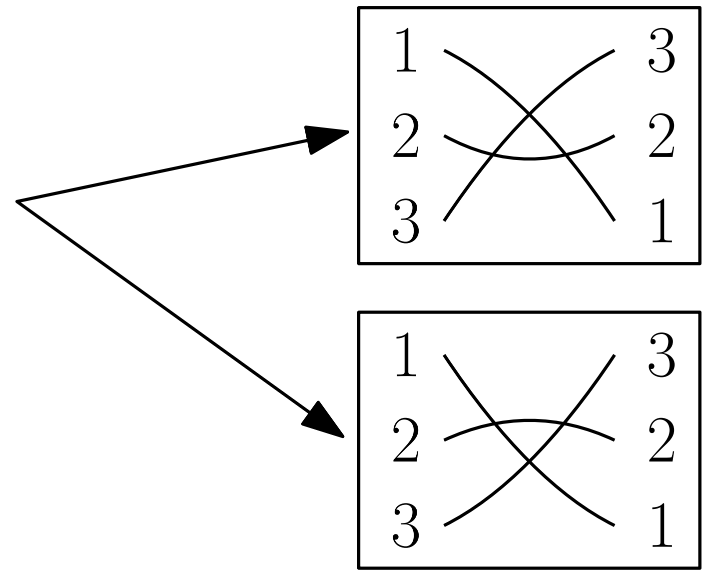
How many pseudoline arrangements?

Freie Universität Berlin

n	#PsA's with n pseudolines
1	1
2	1
3	2
4	8
5	62
6	908
7	24698
8	1232944
9	112018190
10	18410581880
11	5449192389984
12	2894710651370536
13	2752596959306389652
14	4675651520558571537540
15	14163808995580022218786390
16	764130737257725932304619367 36

$\geq 2^{0.2083n^2}$
[Dumitrescu, Mandal 2018]

$\leq 2^{0.6571n^2}$
[Felsner, Valtr 2012]

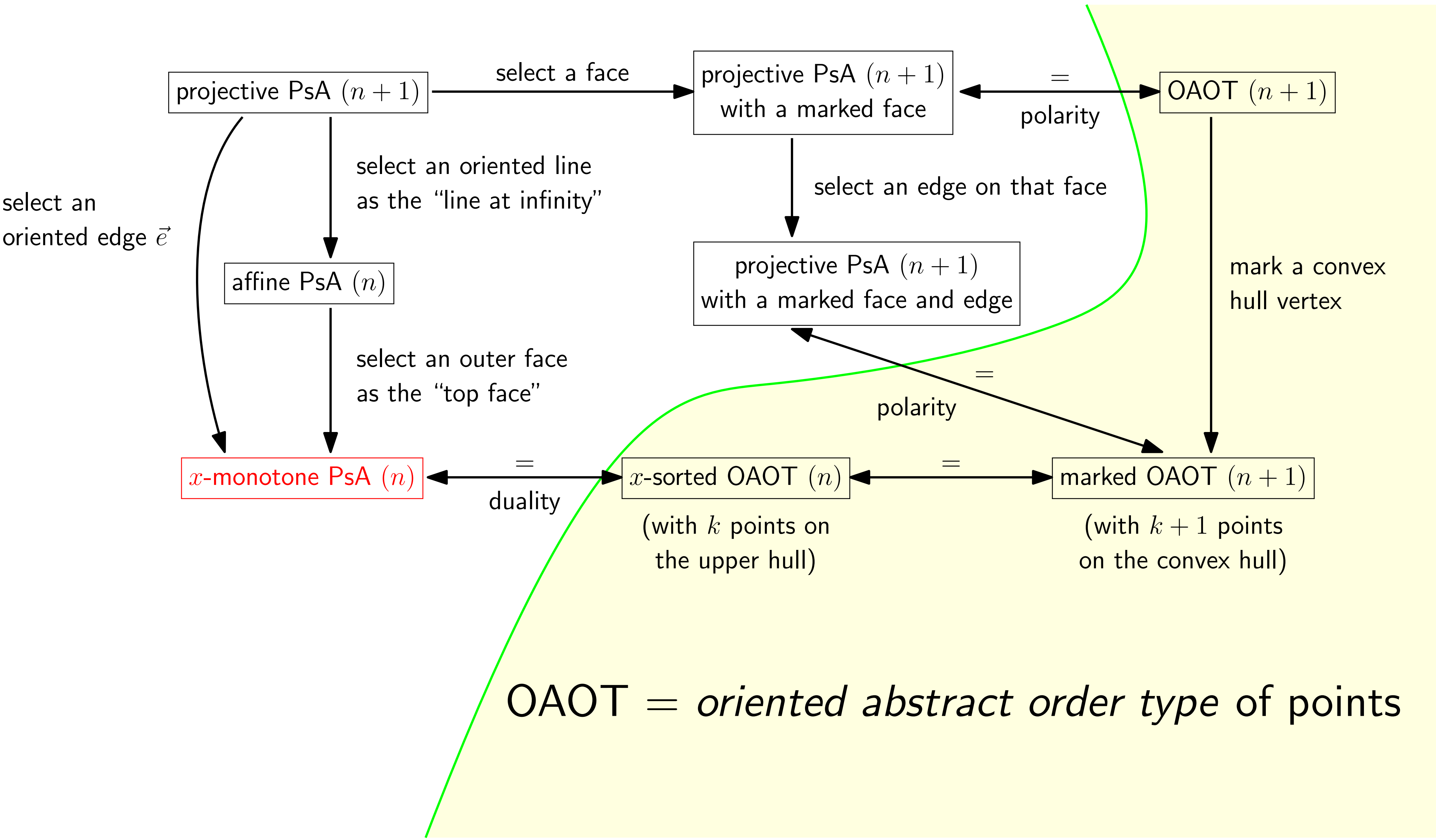


OEIS A006245

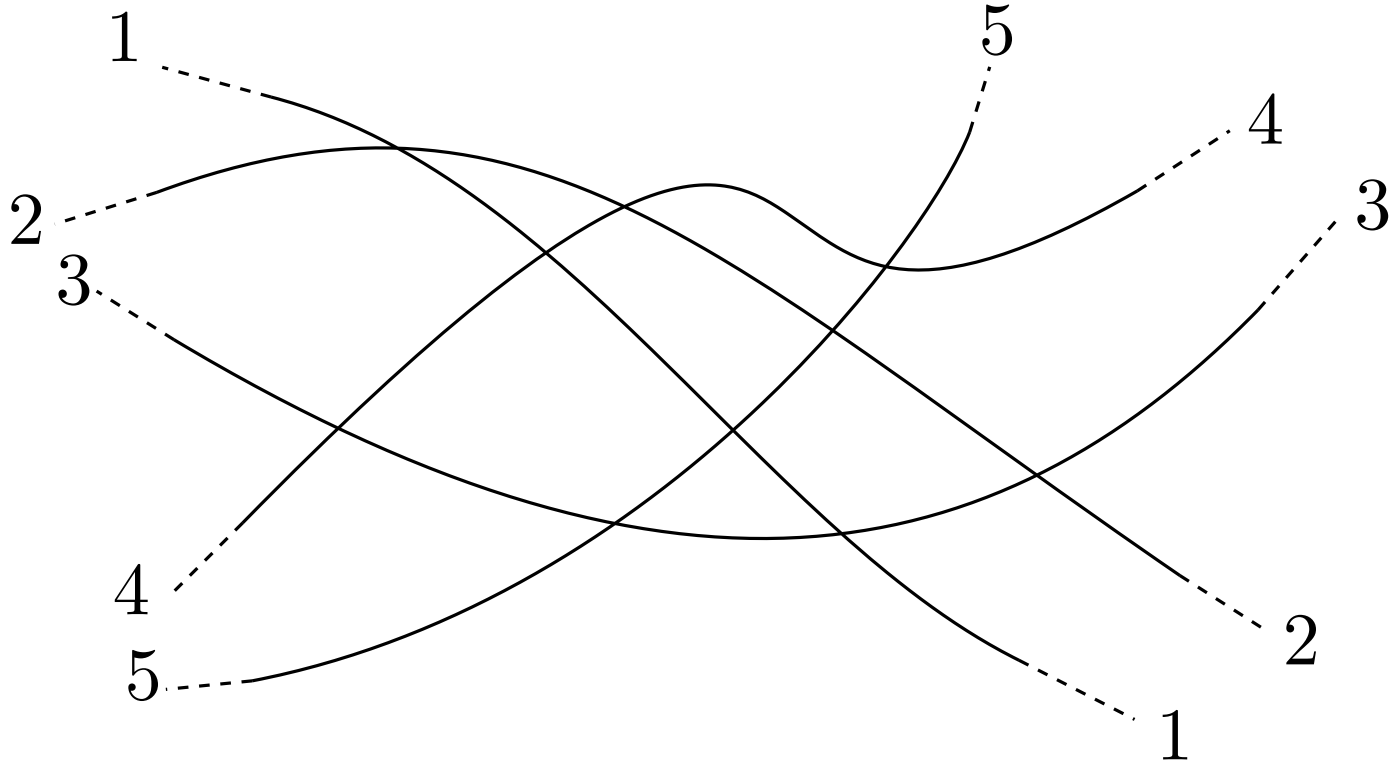
$n = 14, 15$: [Yuma Tanaka, 2013]

$n = 16$: [G. Rote, 2021]

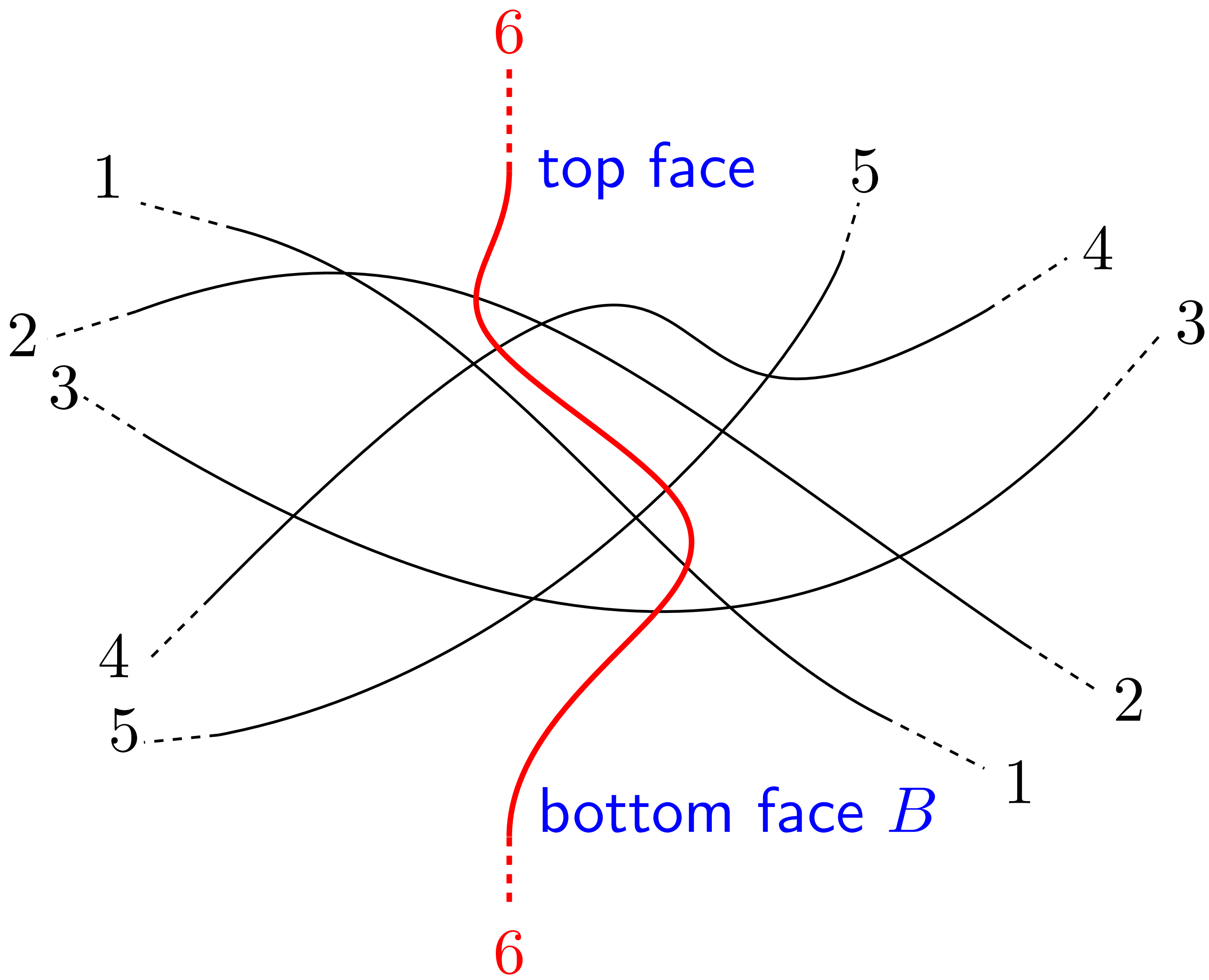
Related concepts



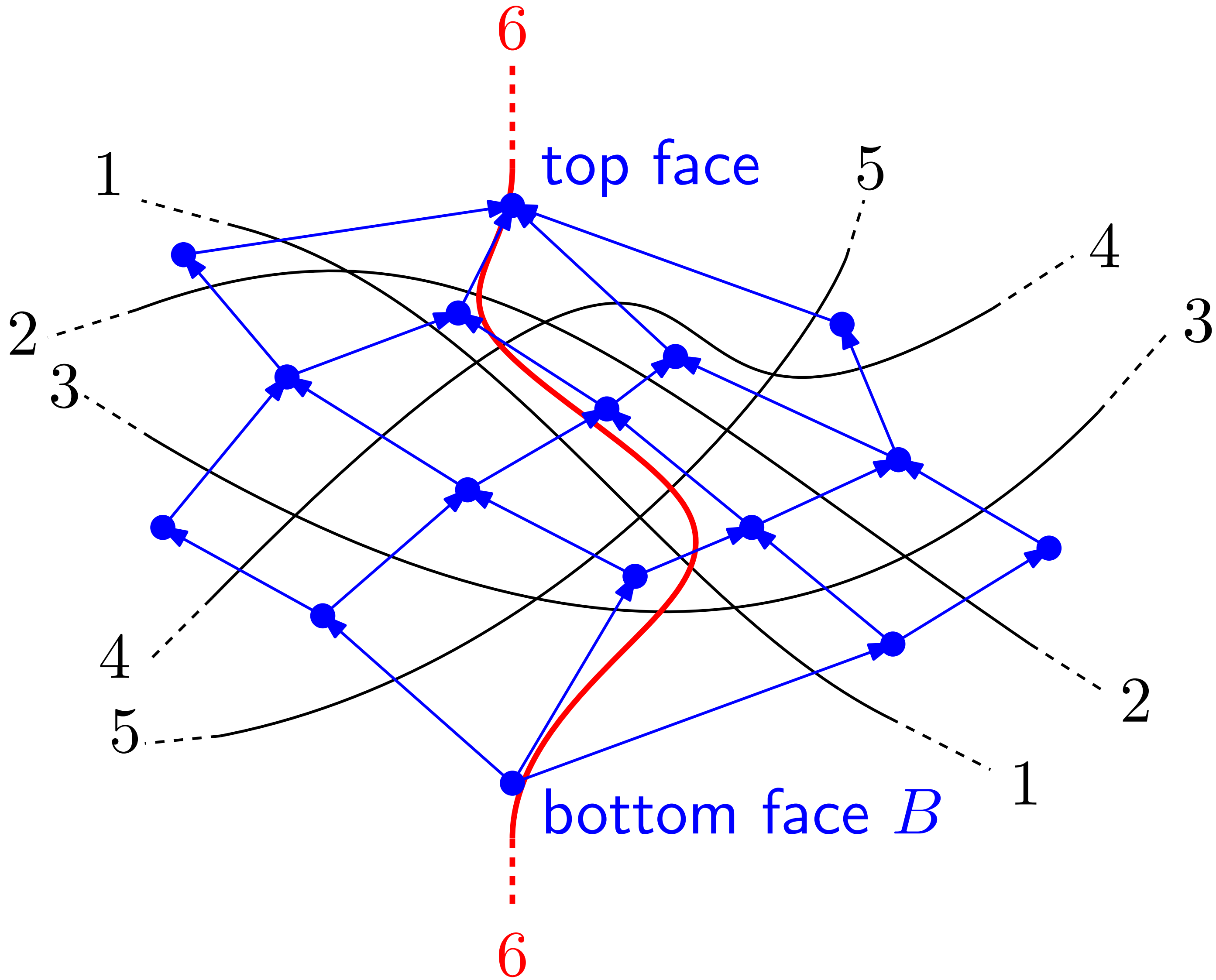
Inductive Enumeration of PsA's



Inductive Enumeration of PsA's

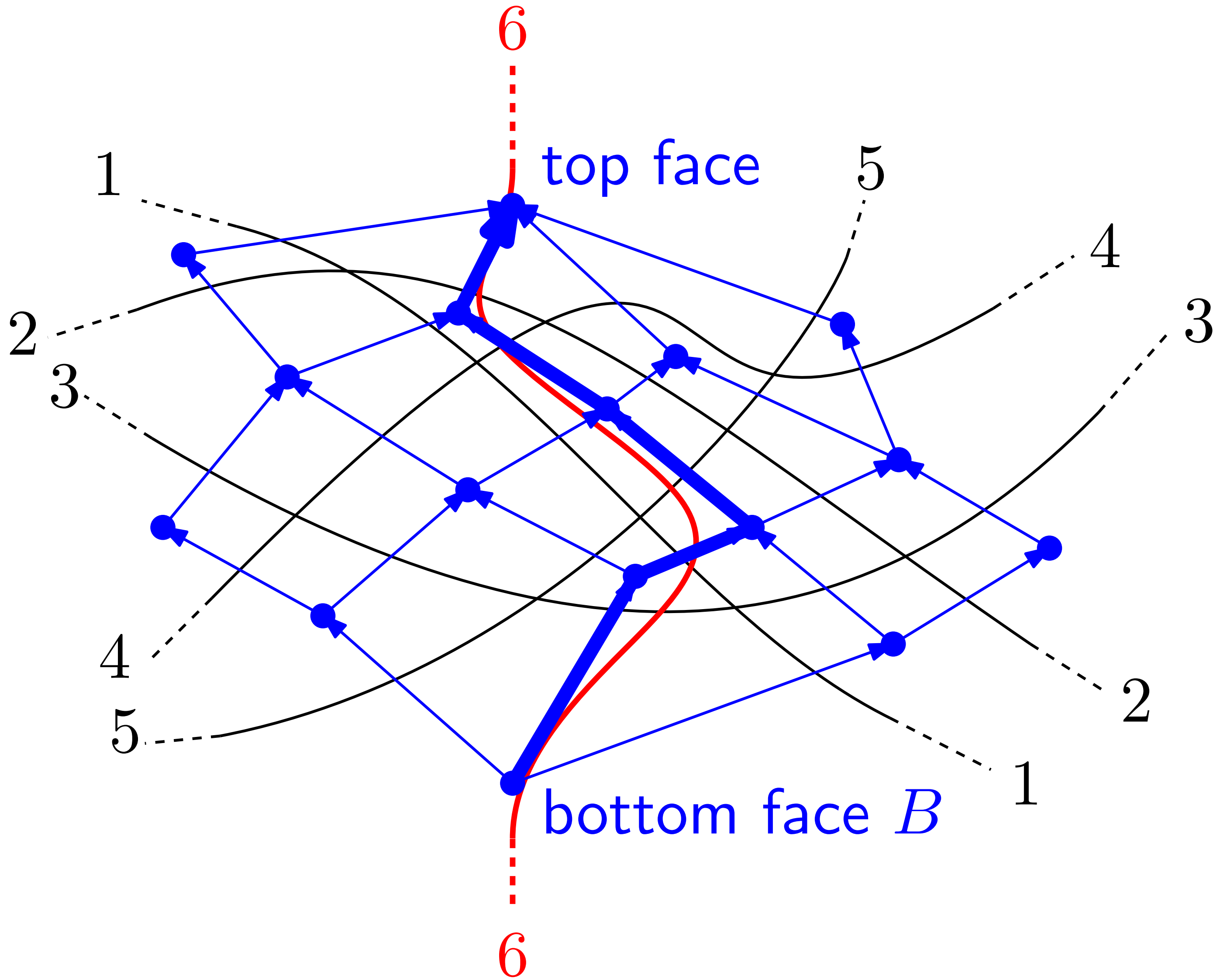


Inductive Enumeration of PsA's



pseudoline $n+1$ = path in the dual DAG

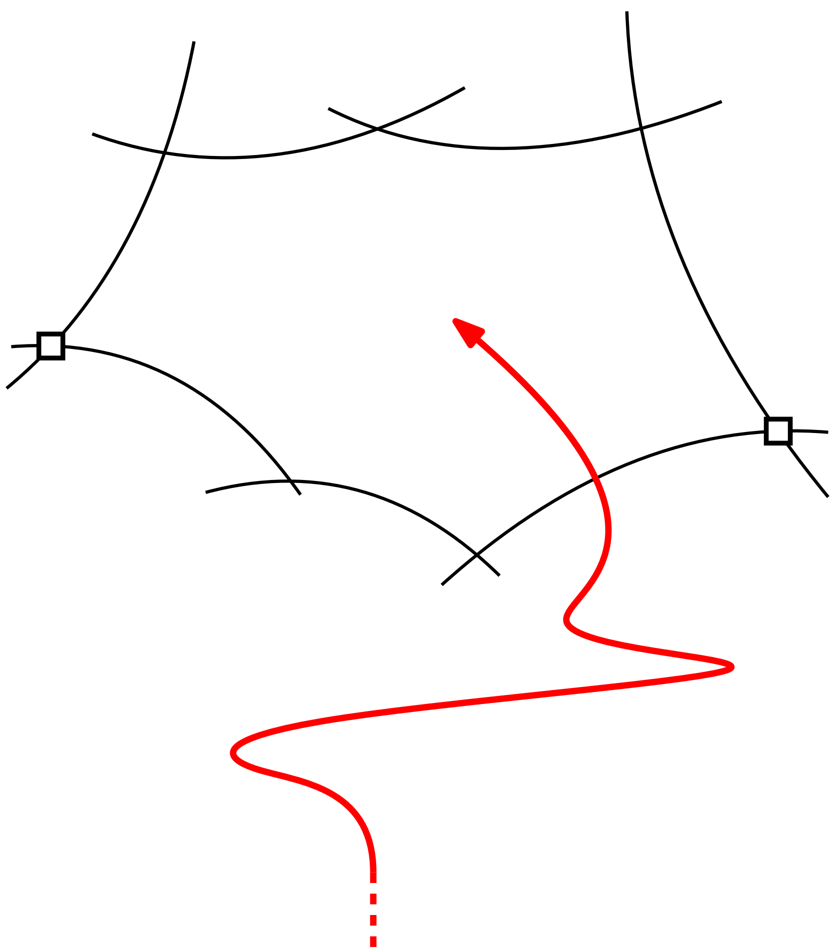
Freie Universität Berlin



pseudoline $n + 1$ = path in the dual DAG

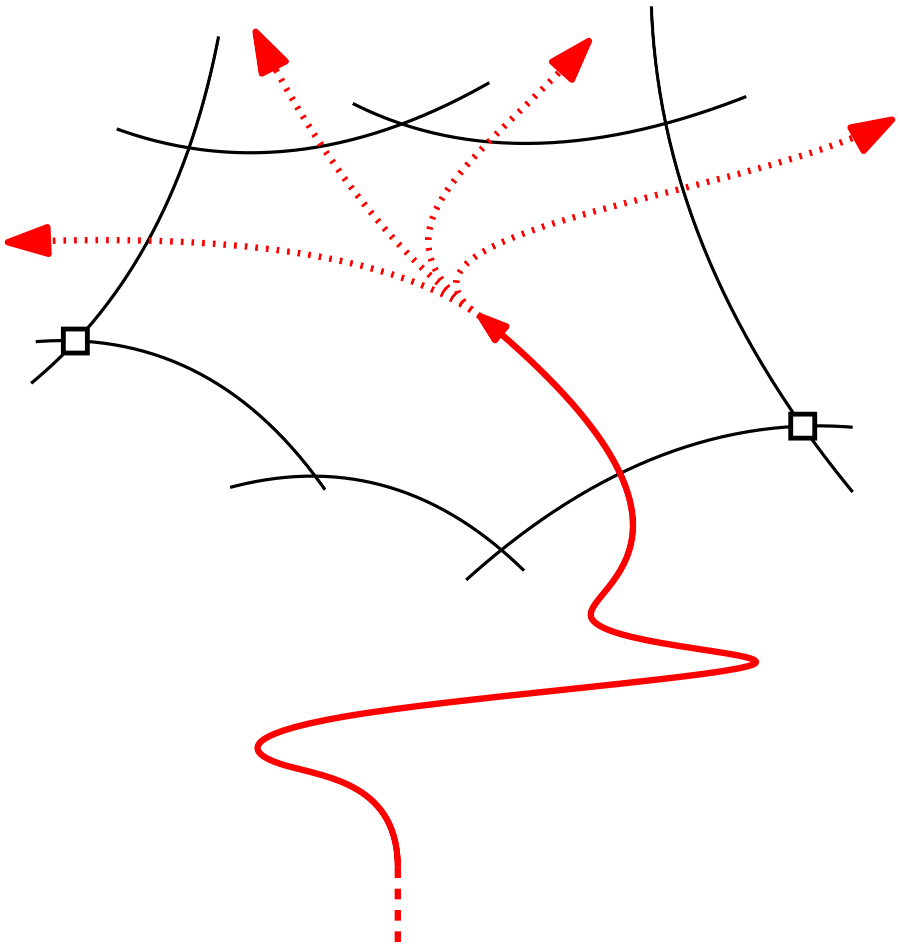
Inductive Enumeration of PsA's

Generation (enumeration) is straightforward. (No dead ends!)



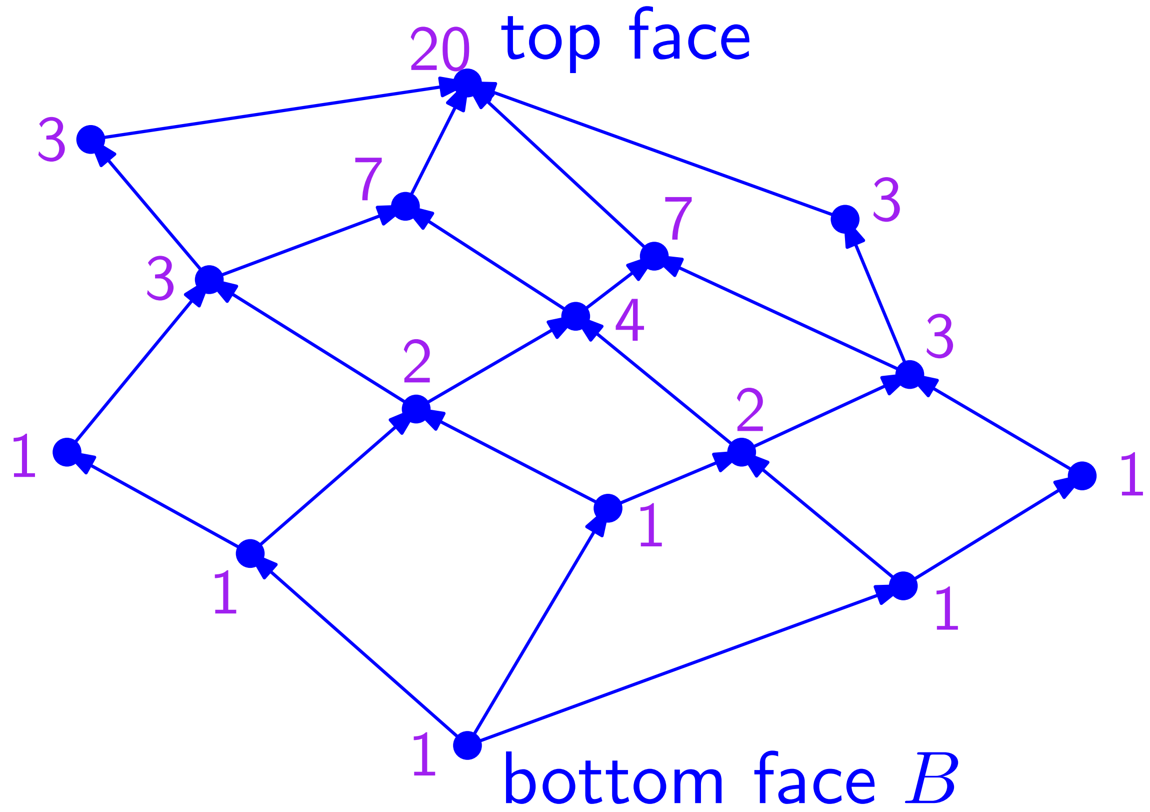
Inductive Enumeration of PsA's

Generation (enumeration) is straightforward. (No dead ends!)



Inductive Enumeration of PsA's

Counting is straightforward. (#paths from B in a DAG)



#paths $\leq 2.49^n$

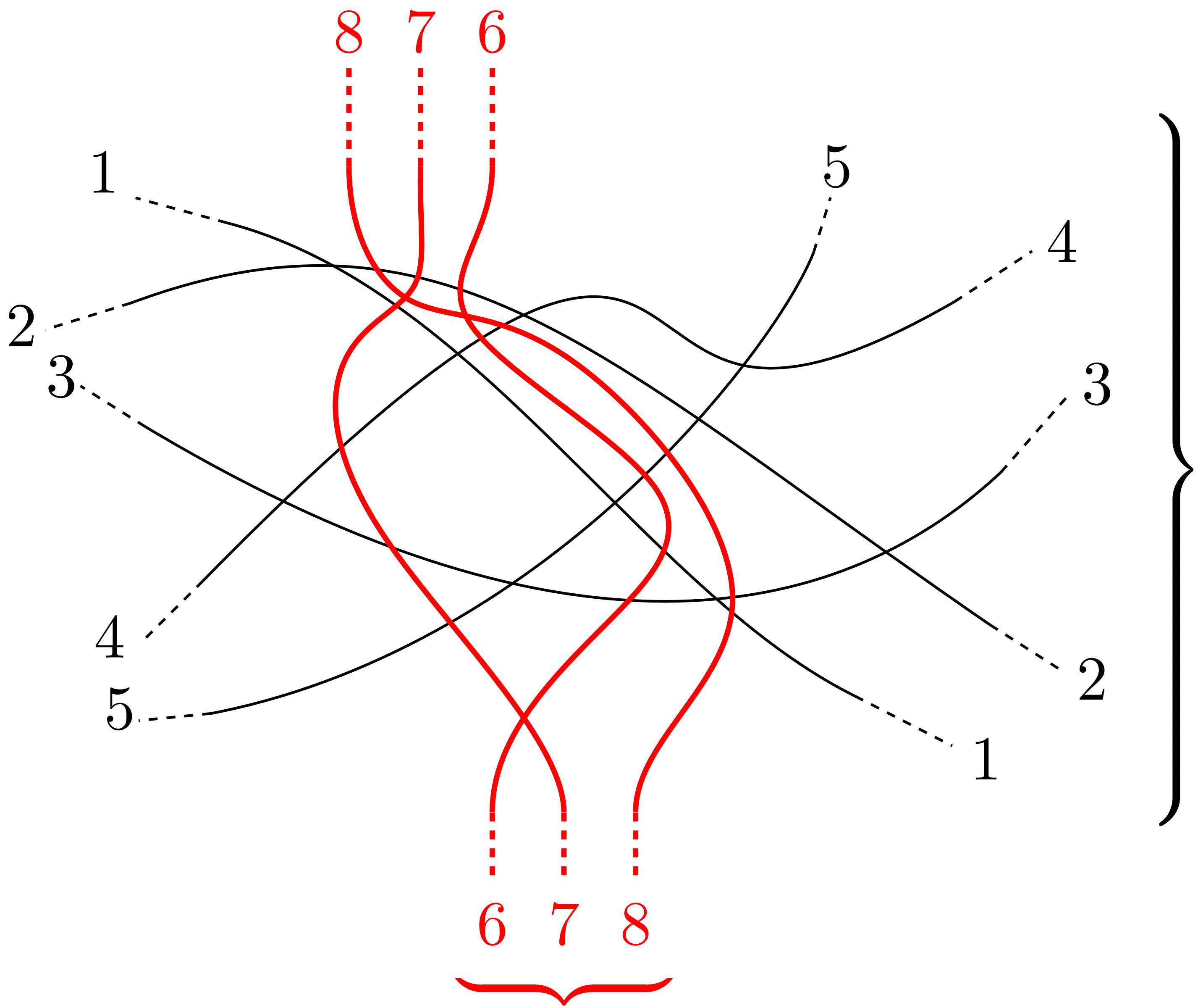
[Felsner, Valtr 2012]

#paths can be as large as 2.076^n.

[O. Bílka 2010]

pseudoline $n+1$ = path in the dual DAG

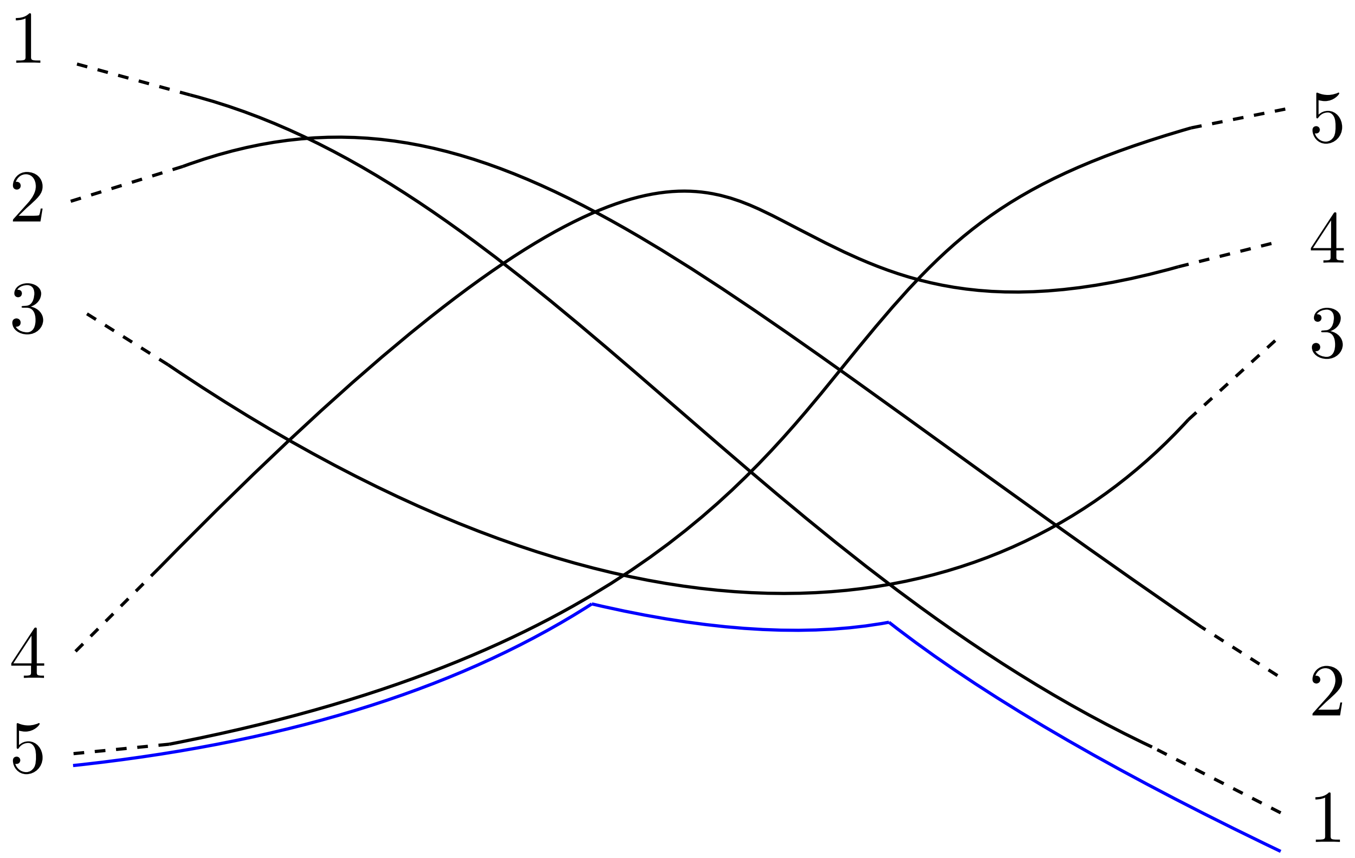
Threading several pseudolines at once



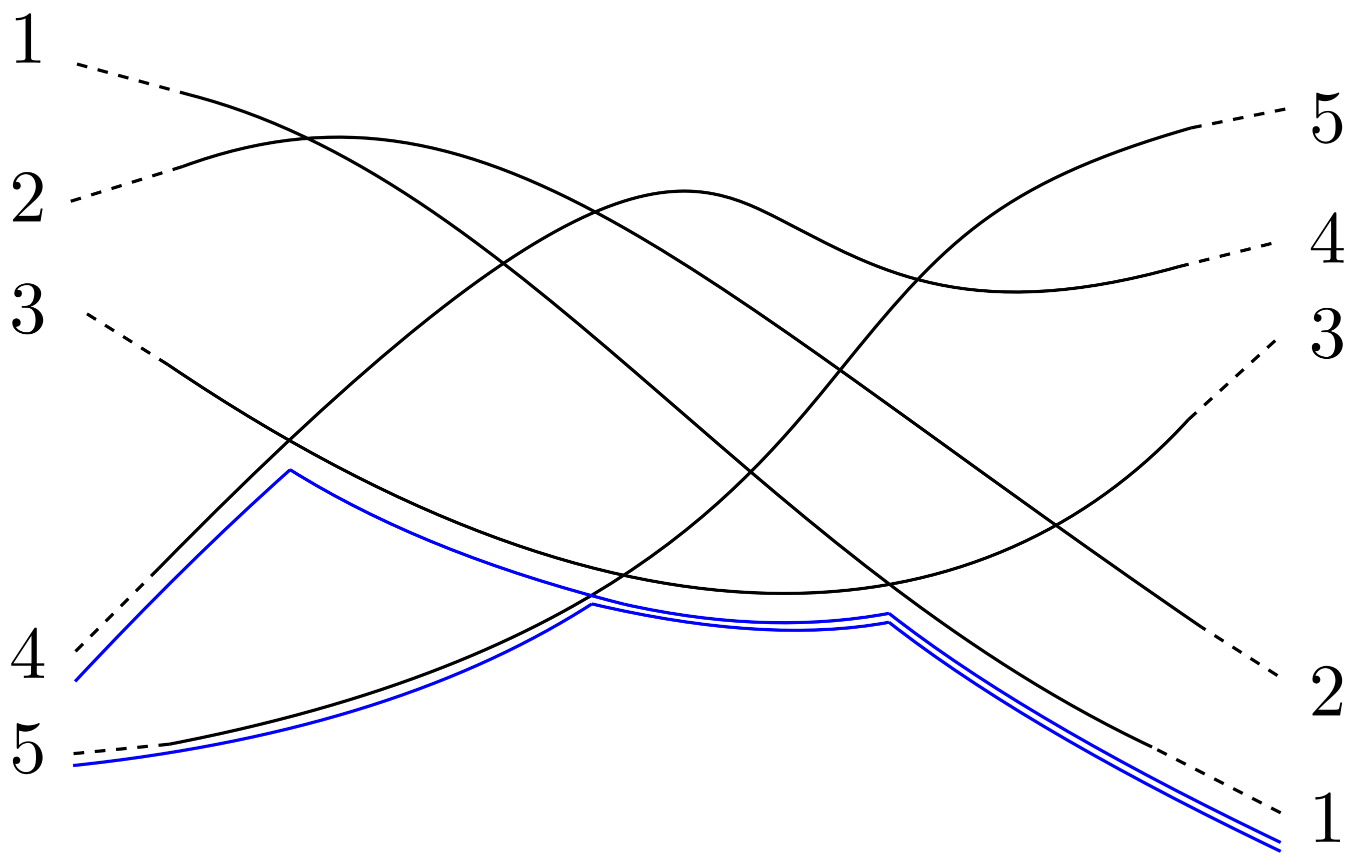
$k = 5$ *pseudolines*

$\ell = 3$ *strands*

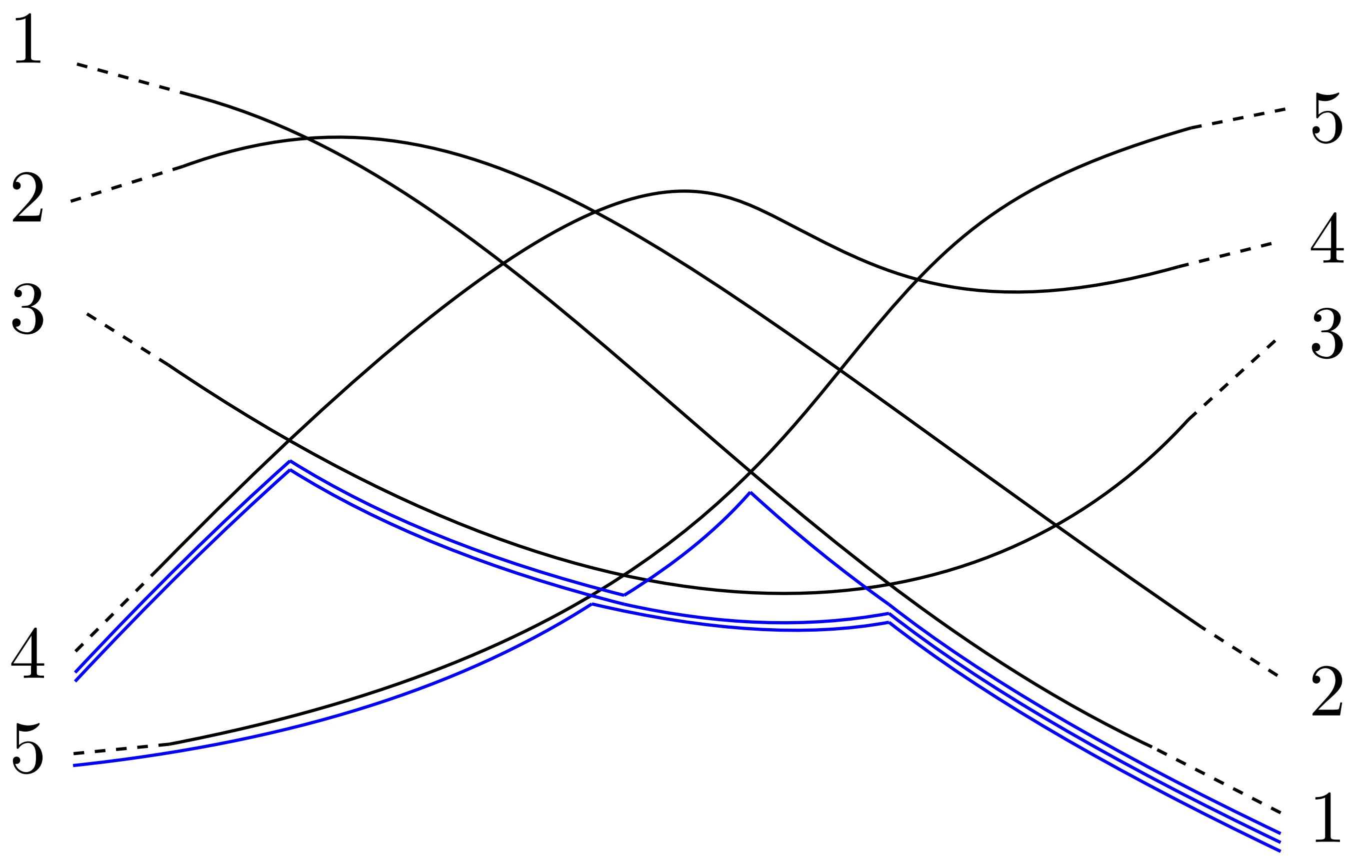
A sequence of ropes



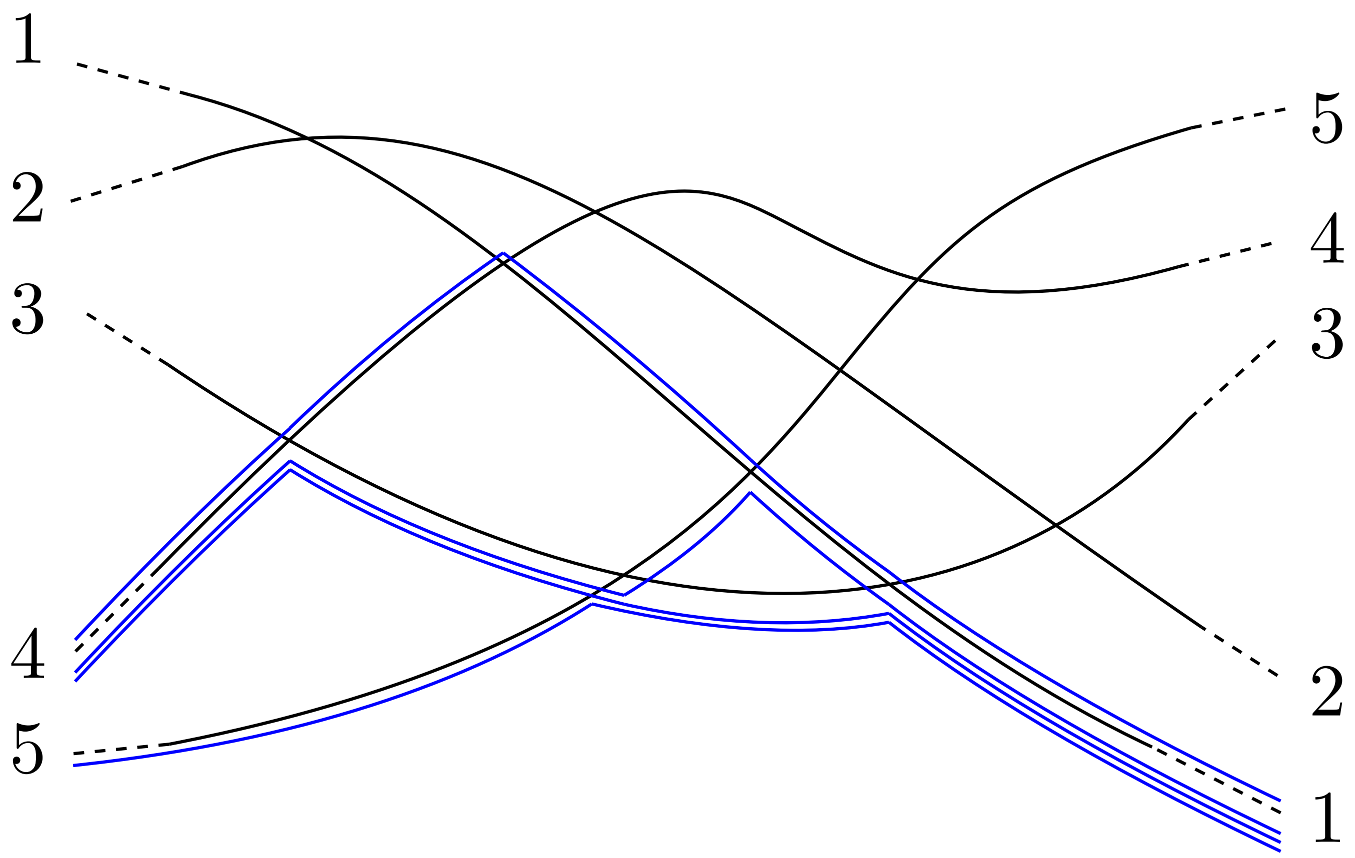
A sequence of ropes



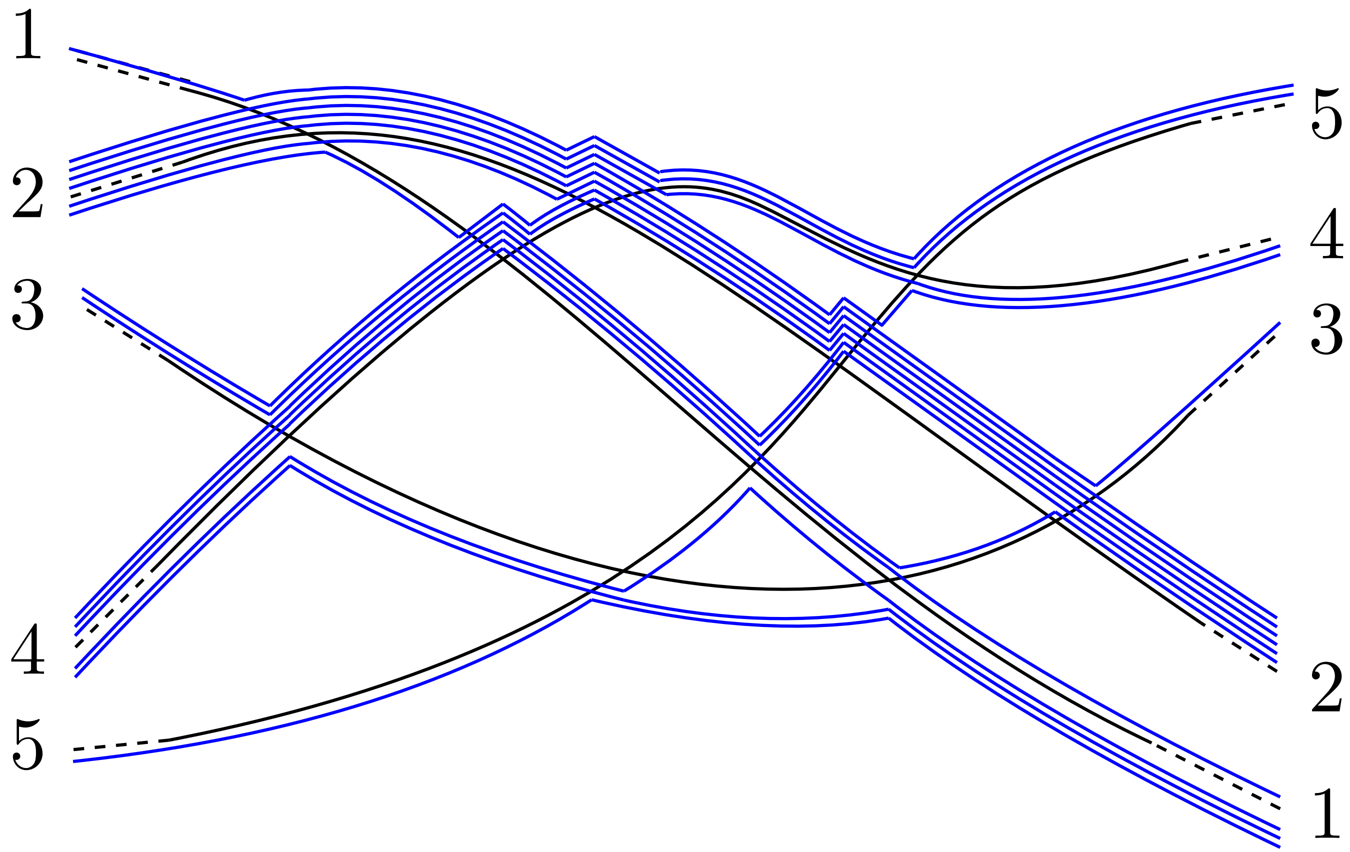
A sequence of ropes



A sequence of ropes



A sequence of ropes

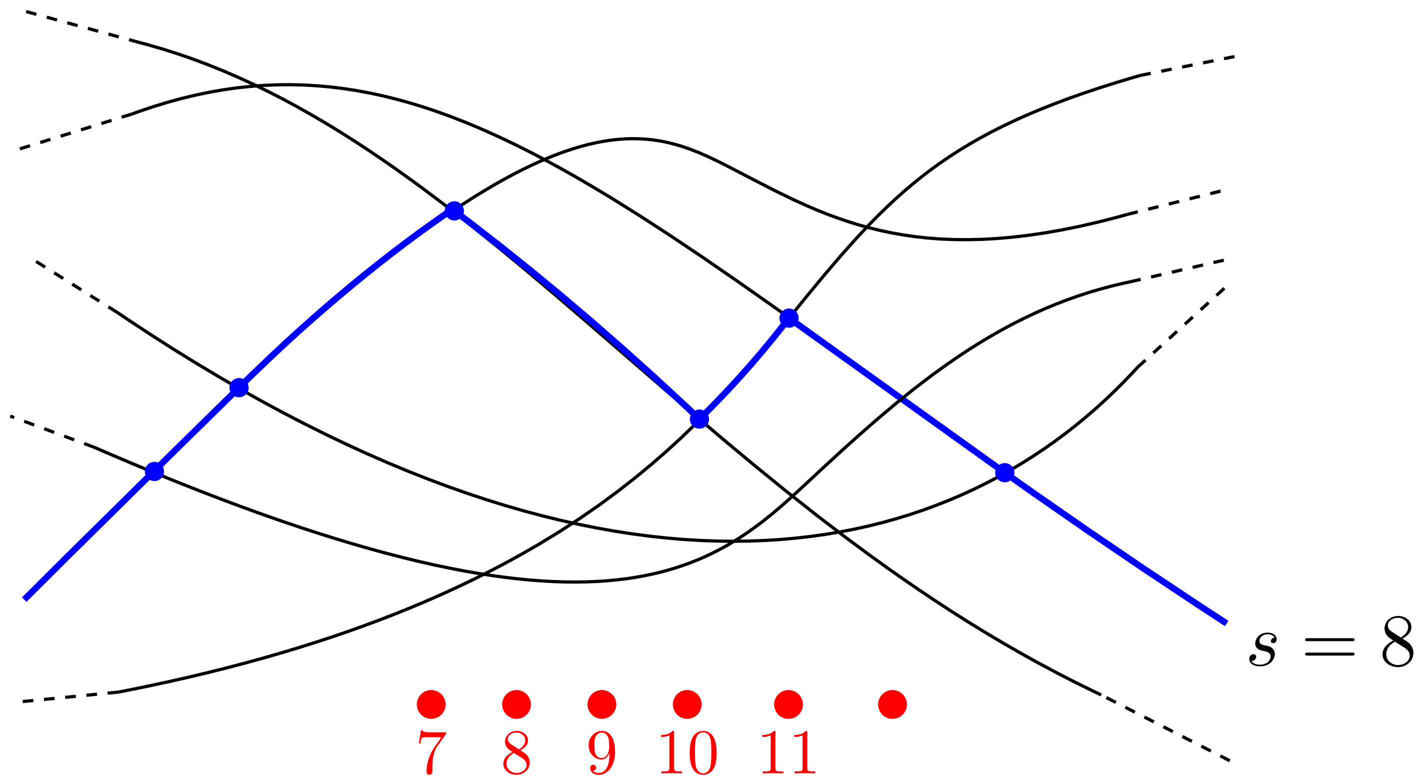


$\frac{k(k+1)}{2}$ ropes

Take a fixed sweep by a sequence of ropes.

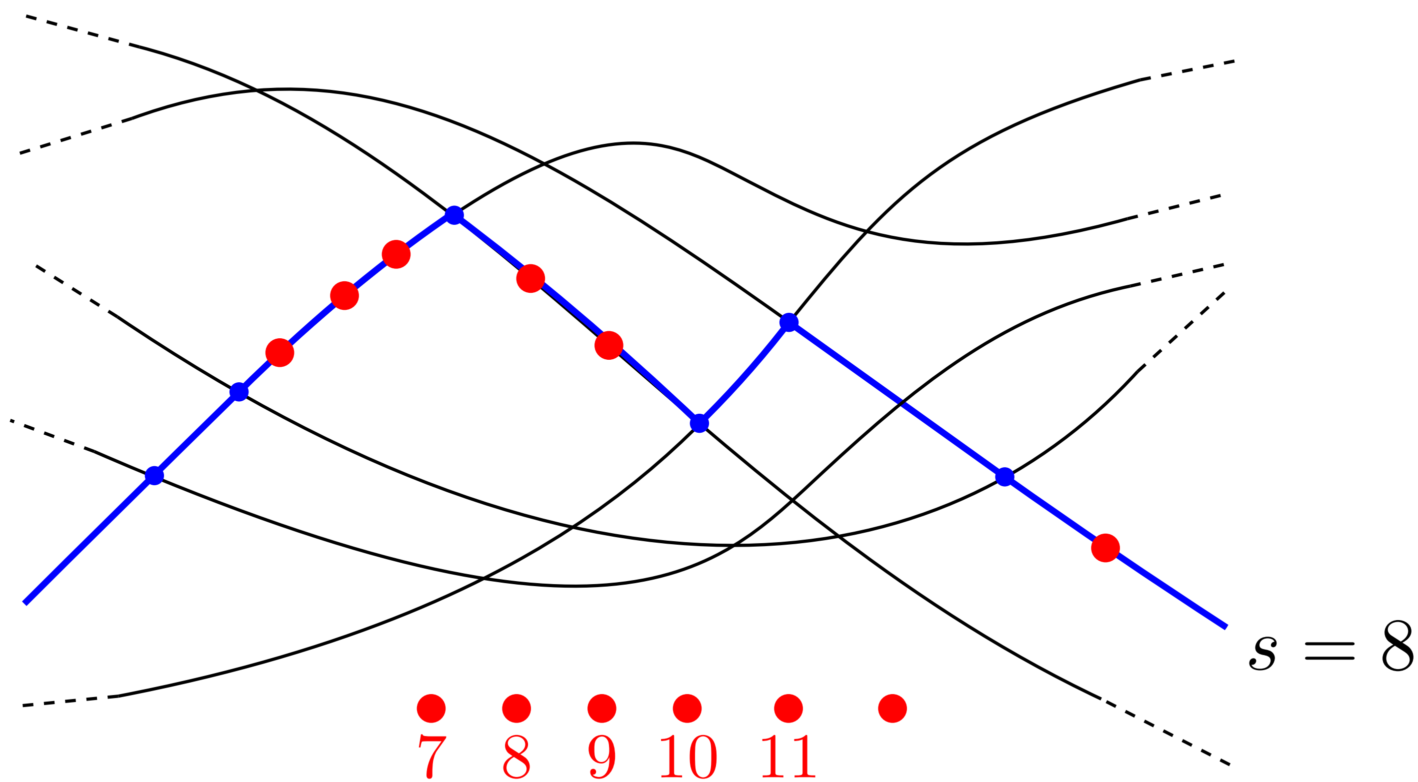
Dynamic programming

For each rope:
(s pieces)



Dynamic programming

For each rope:
(s pieces)



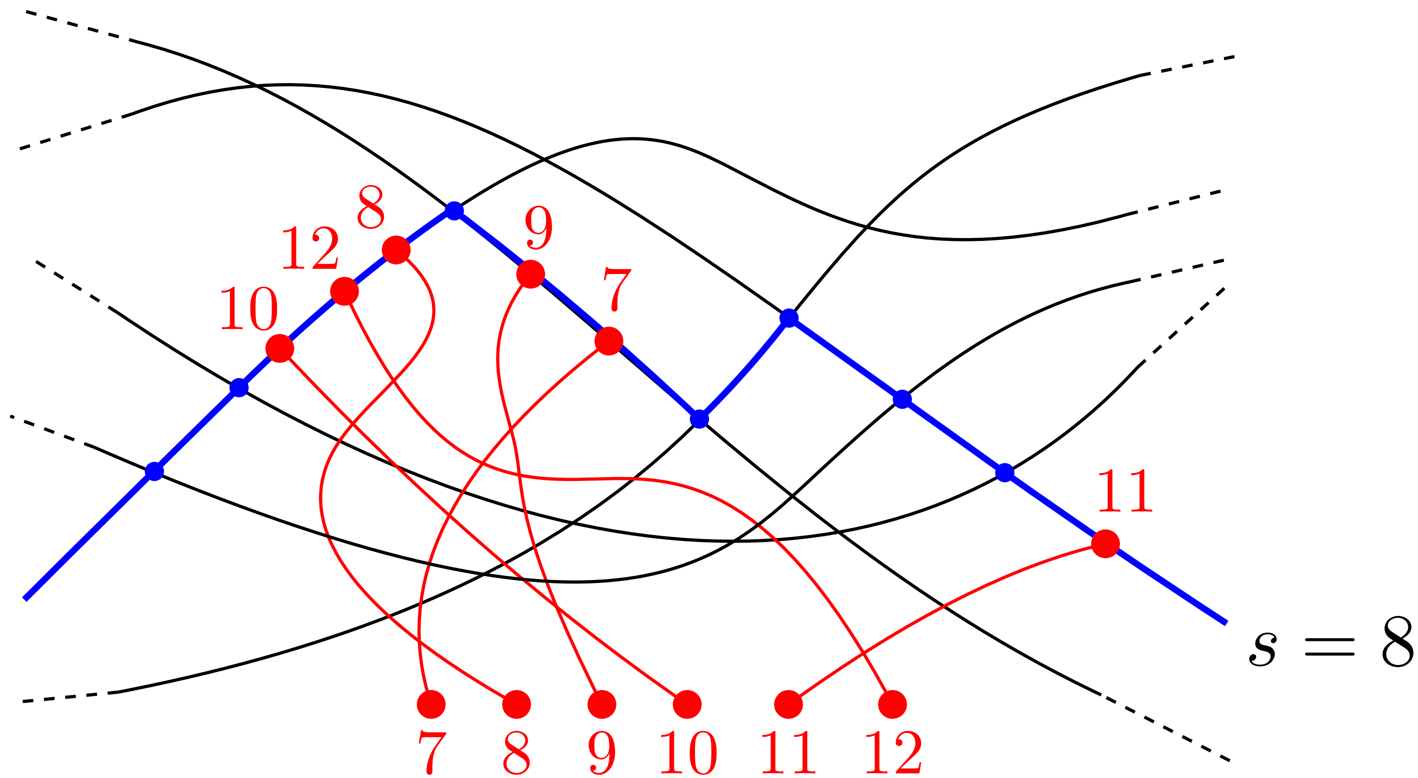
- For every distribution of the ℓ strands to the s pieces
- and for every permutation of the ℓ strands,

[$s(s+1)(s+2)\dots(s+\ell-1)$ entries]

store the number of possibilities to thread the ℓ strands from the bottom face to the rope.

Dynamic programming

For each rope:
(s pieces)

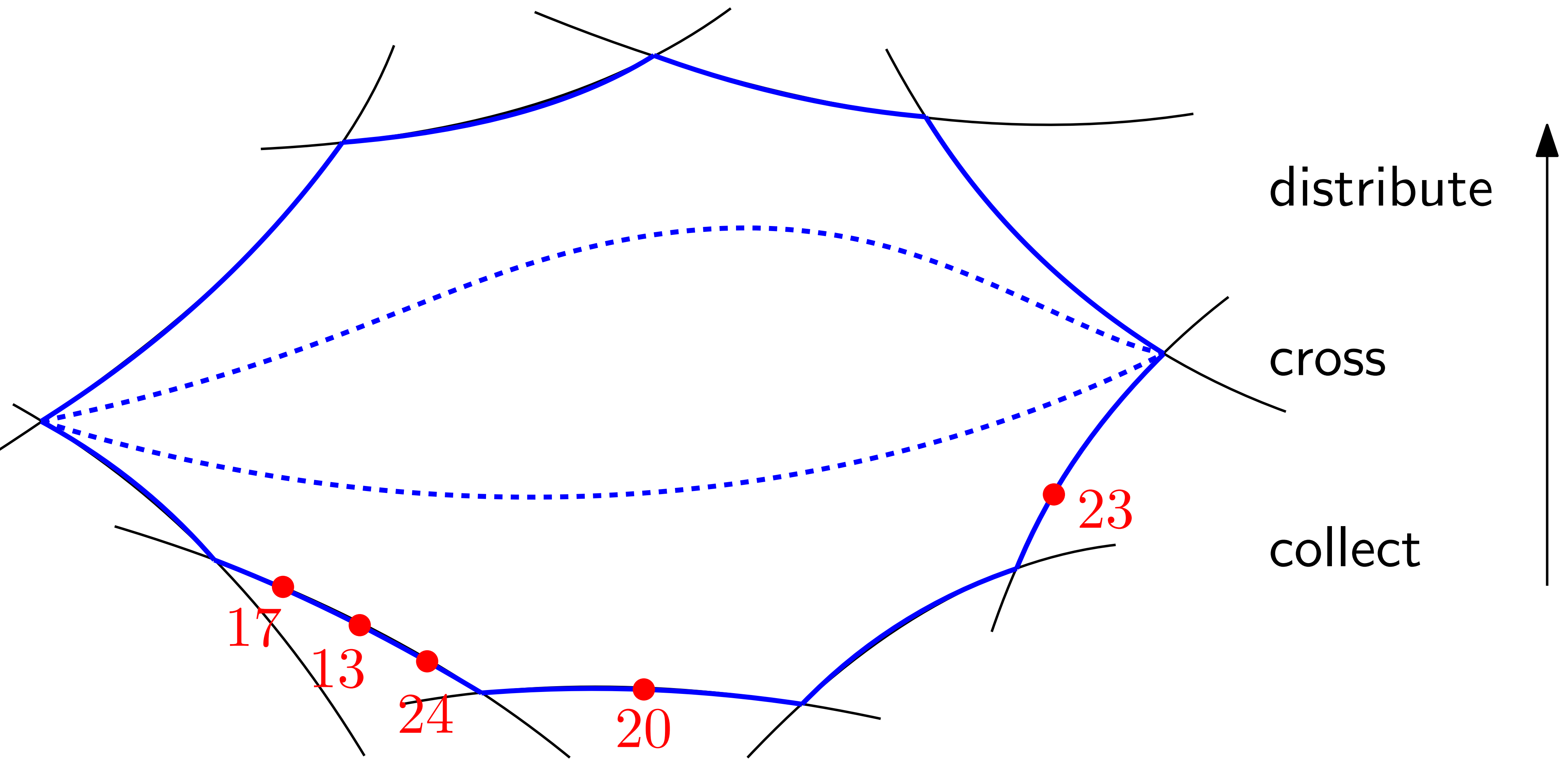


- For every distribution of the ℓ strands to the s pieces
- and for every permutation of the ℓ strands,

[$s(s+1)(s+2)\dots(s+\ell-1)$ entries]

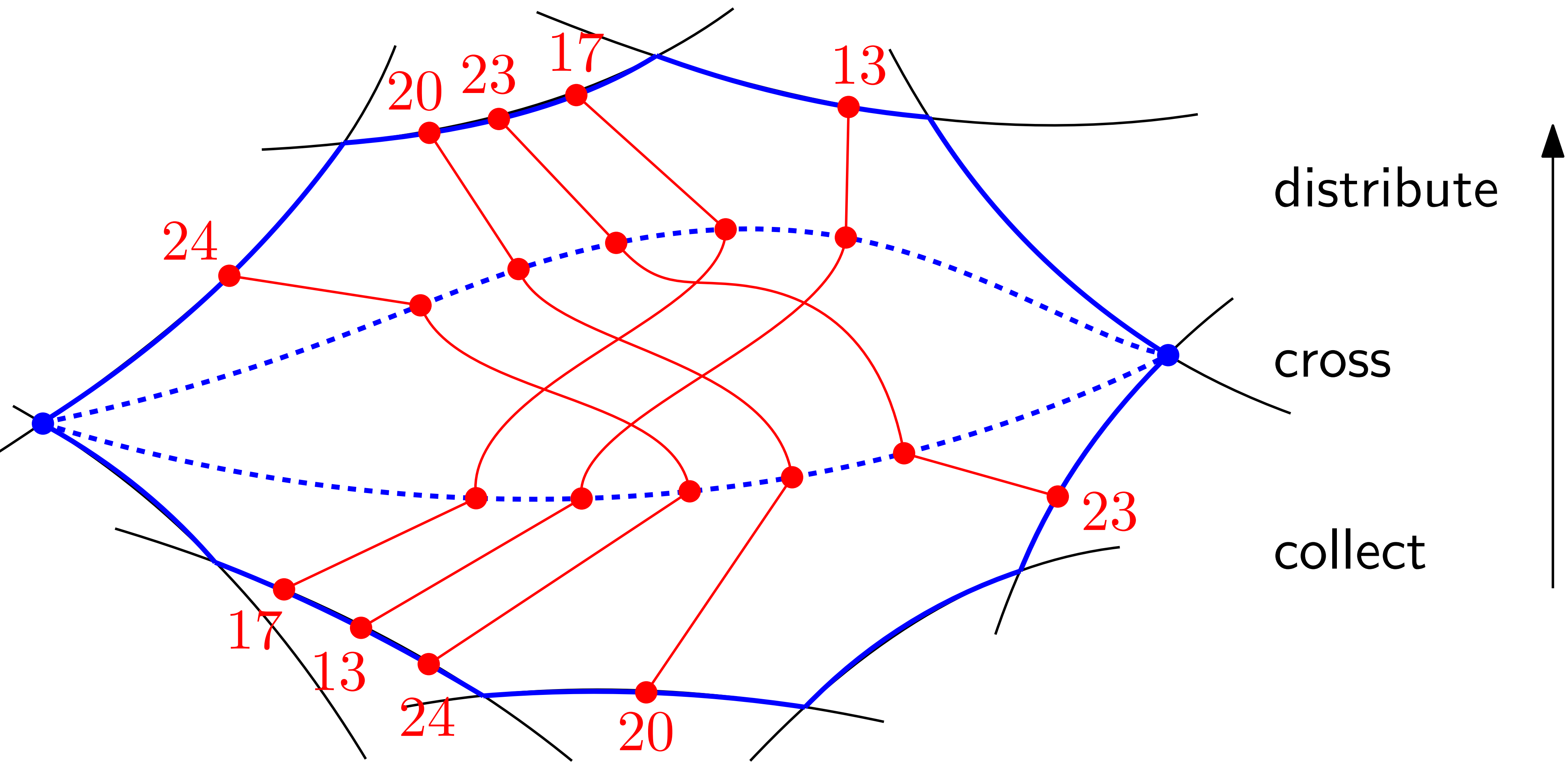
store the number of possibilities to thread the ℓ strands from the bottom face to the rope.

Advancing the rope across a face



What is the contribution to the next rope?

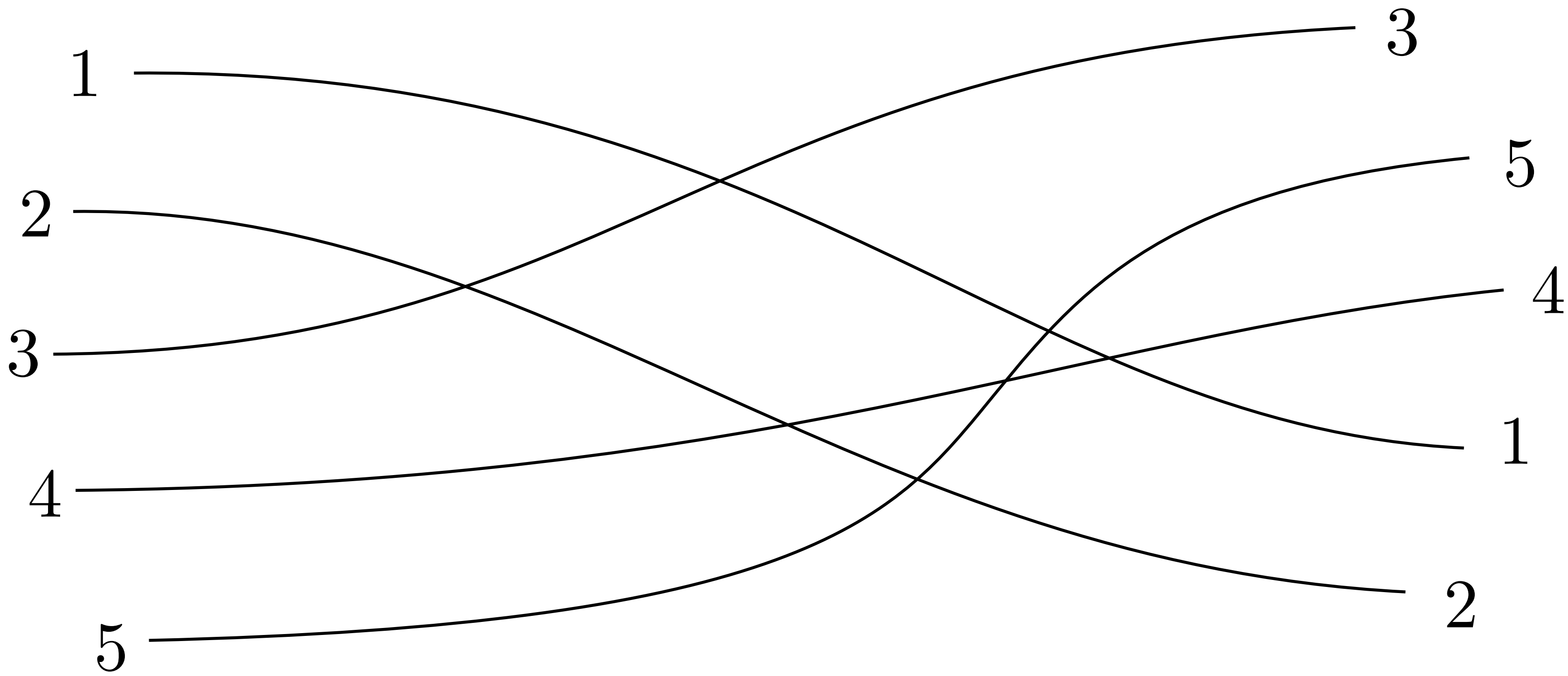
Advancing the rope across a face



What is the contribution to the next rope?

PARTIAL pseudoline arrangements

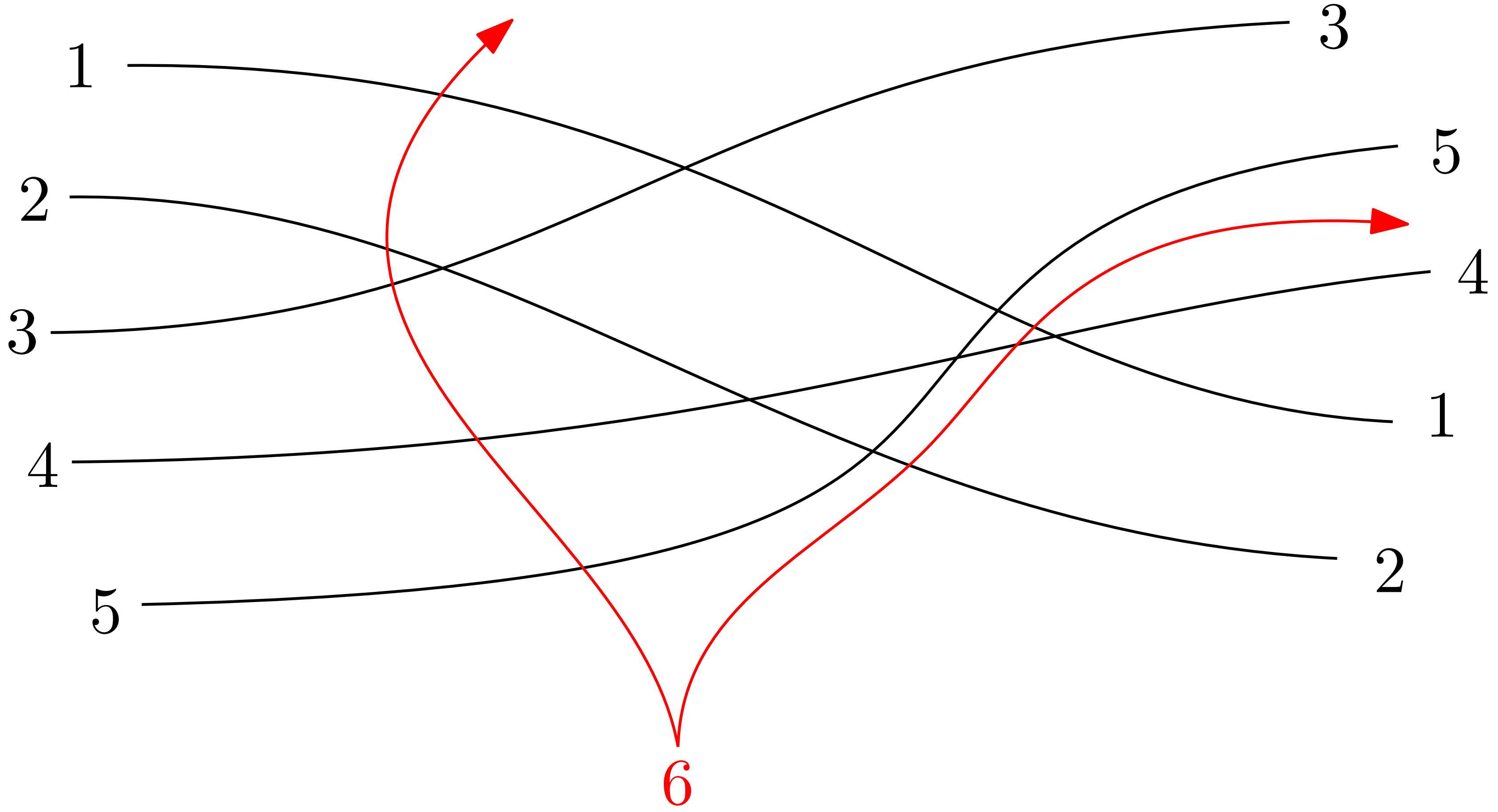
Pseudolines may not cross at all.



PARTIAL pseudoline arrangements

Freie Universität Berlin

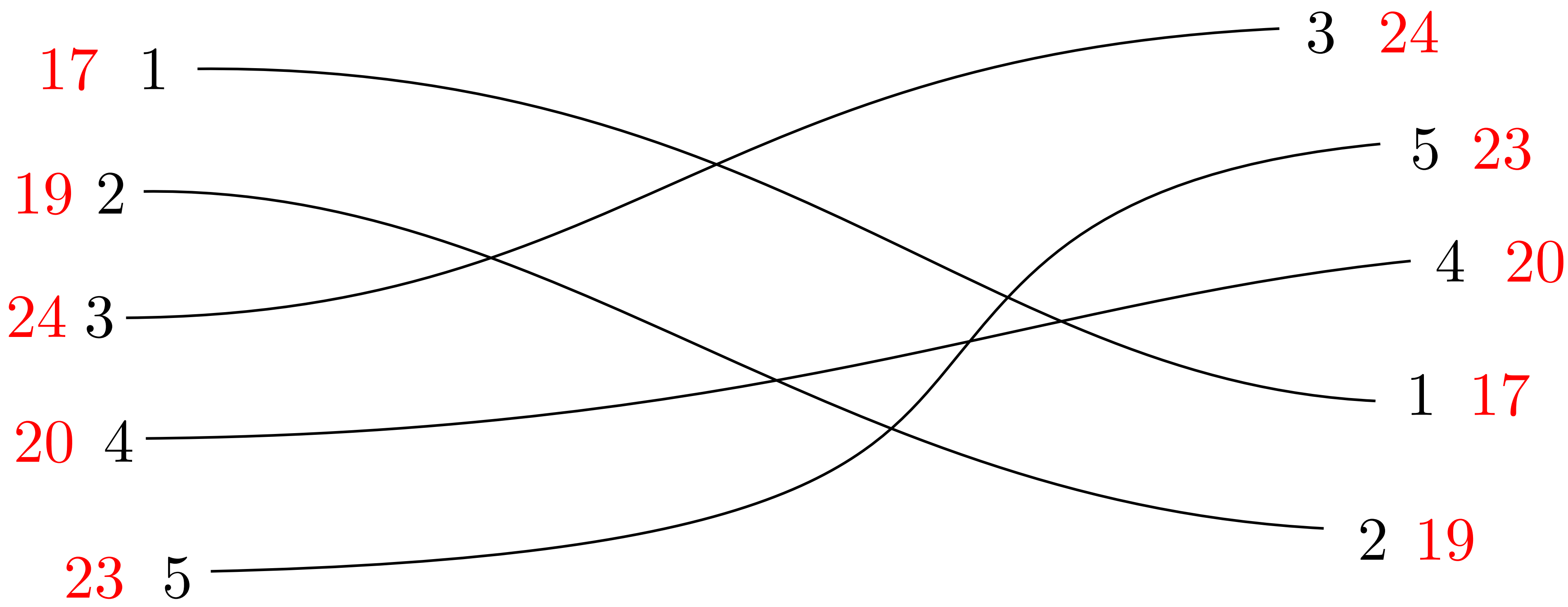
Pseudolines may not cross at all.



Enumeration is as easy as for full PsA's.

PARTIAL pseudoline arrangements

Pseudolines may not cross at all.



Preprocessing: $\rightarrow$ $\ell! \times \ell!$ table (sparse!)

Algorithm summary

For each PsA of k pseudolines:

- Compute a sweep by ropes
- For each rope:
  - For each distribution and permutation of the ℓ strands:
    - Compute the contributions to the next rope, and accumulate them.

Some implementation details

- PYTHON, with `scipy` for large arrays of 32/64-bit integers
- modular arithmetic, using 2^{64} plus two 30-bit moduli
- $n = 16 = k + \ell = 7 + 9$. Large memory!
  256 GBytes is enough; 128 GBytes sometimes failed.
- easy to parallelize:
  a large number (24,698) of independent tasks
- total CPU time: about 5.5 months, using various workstations of different speeds
- CPU time for $n = 15 = 6 + 9$ (exploiting symmetry): 6 h.
  By contrast*: PYTHON without `scipy` took 50 CPU days.
- There is also a version in C (using `CWEB`) for the task of *enumerating* PsA's.

The required rope length

- Every arrangement requires $\geq n+1$ pieces (for $n \geq 3$).
- can always do with $\leq 2n-2$ pieces. (greedy sweep)
- Some arrangements require $\lfloor \frac{7n}{4} \rfloor - 1$ pieces.
  (This is the true maximum for $n \leq 9$.)
- NP-hard? (homotopy height, cutwidth)

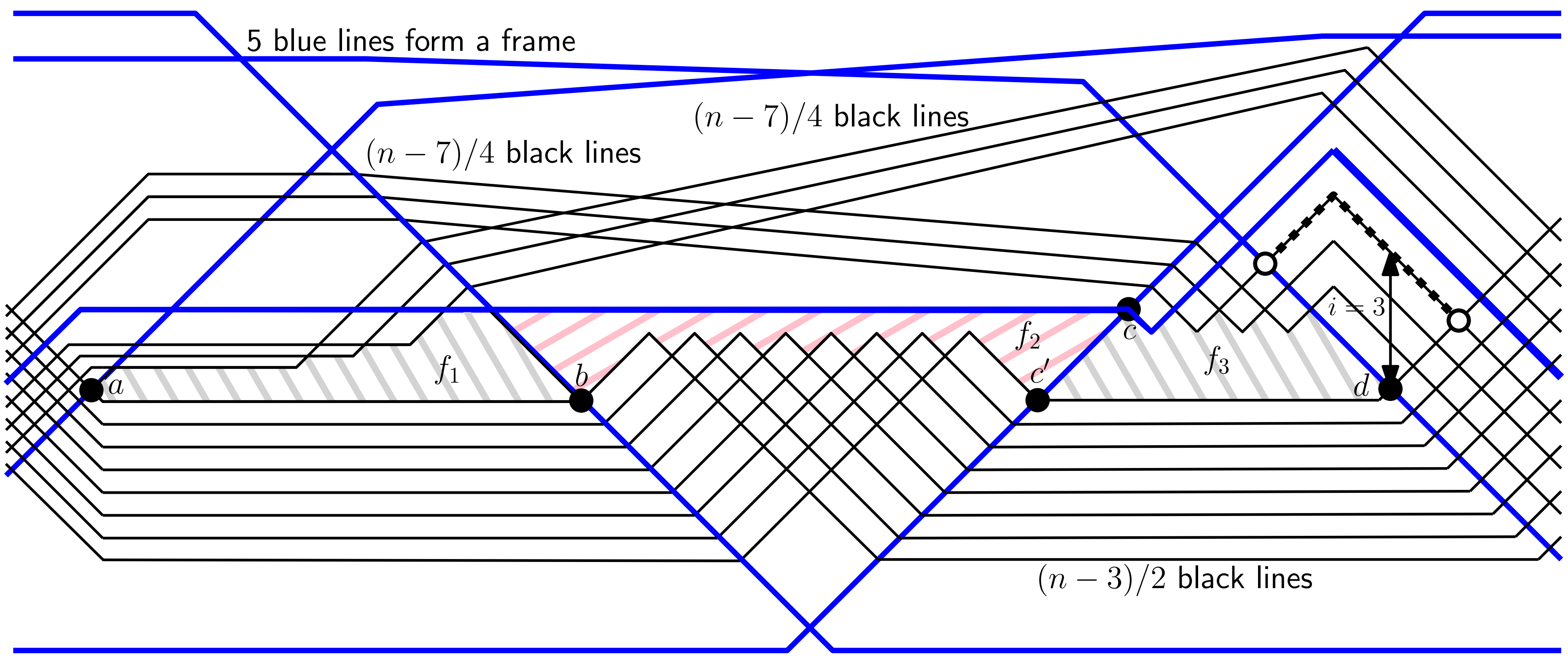
[Biedl, Chambers, Kostitsyna, Rote, 2020, unpublished, + this week]

The required rope length

Freie Universität Berlin

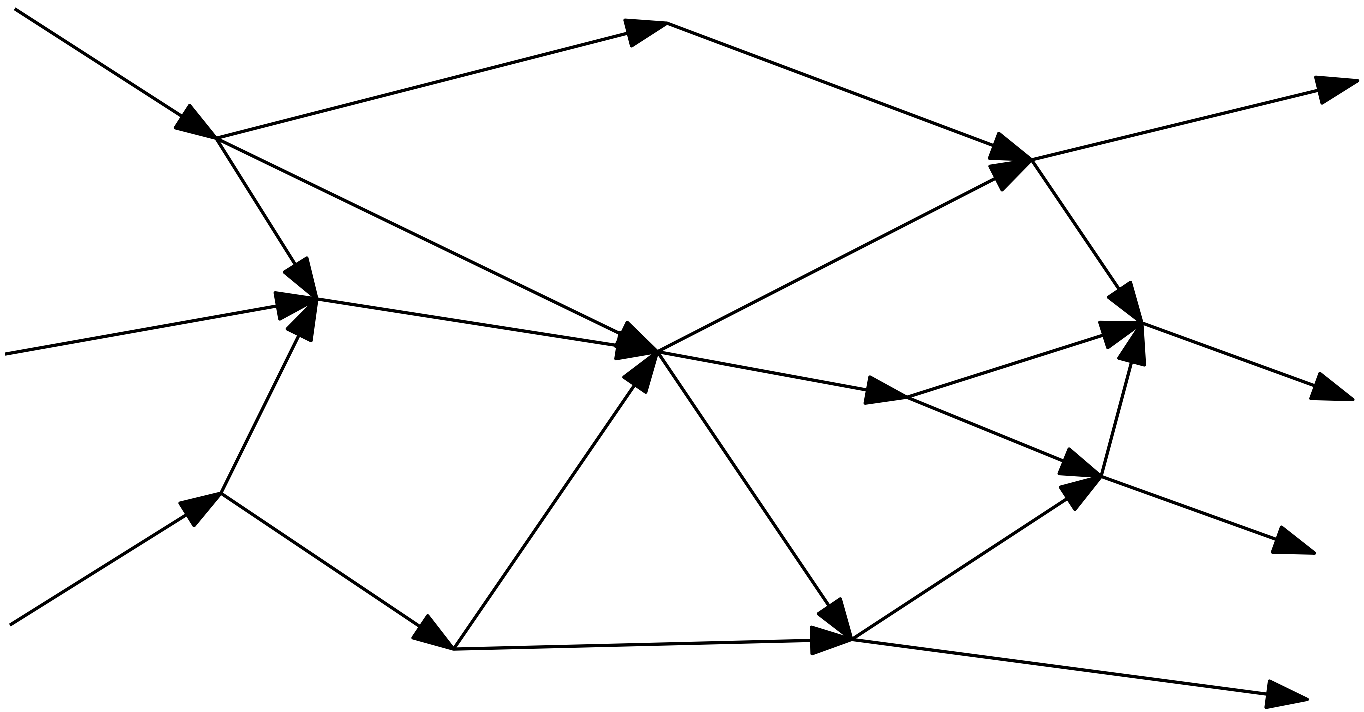
- Every arrangement requires $\geq n+1$ pieces (for $n \geq 3$).
- can always do with $\leq 2n-2$ pieces. (greedy sweep)
- Some arrangements require $\lfloor \frac{7n}{4} \rfloor - 1$ pieces.
  (This is the true maximum for $n \leq 9$.)
- NP-hard? (homotopy height, cutwidth)

[Biedl, Chambers, Kostitsyna, Rote, 2020, unpublished, + this week]



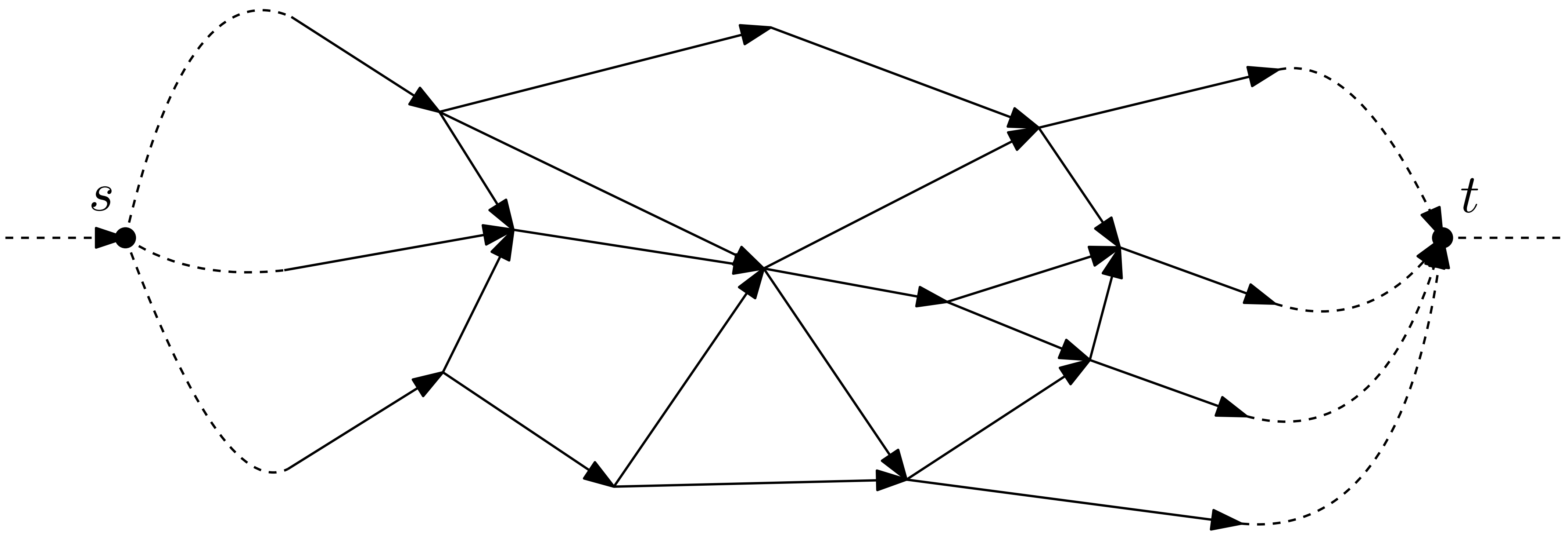
The required rope length

This is really about *bipolar orientations* (s-t-planar DAGs):



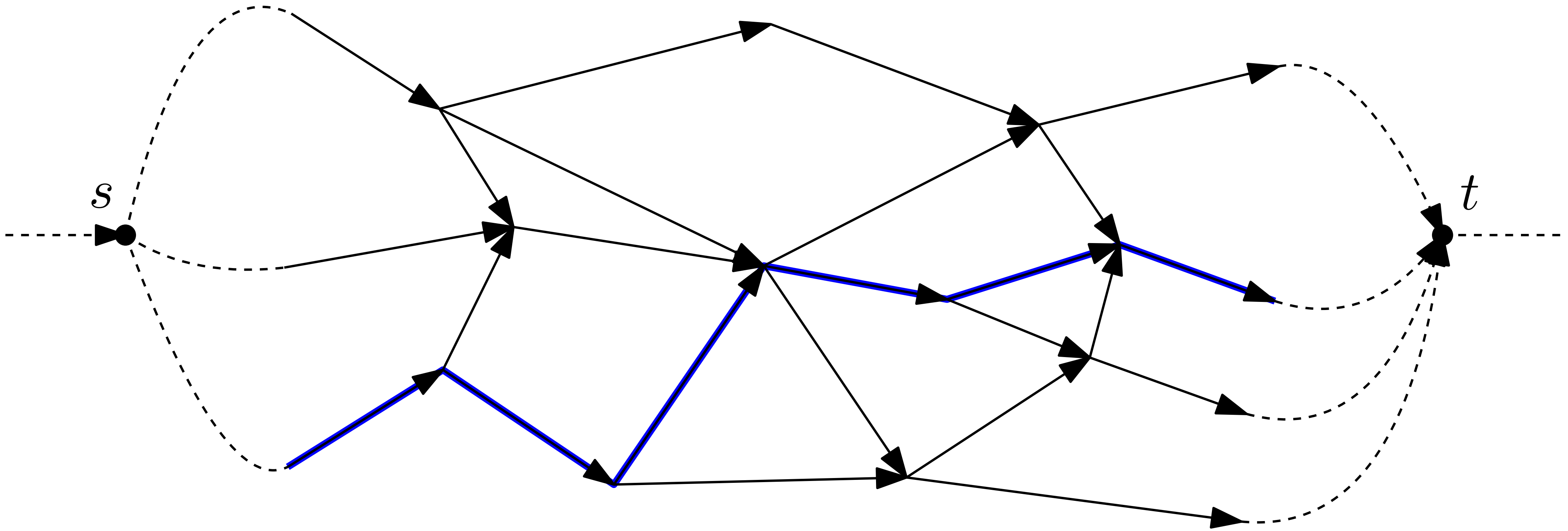
The required rope length

This is really about *bipolar orientations* (s-t-planar DAGs):



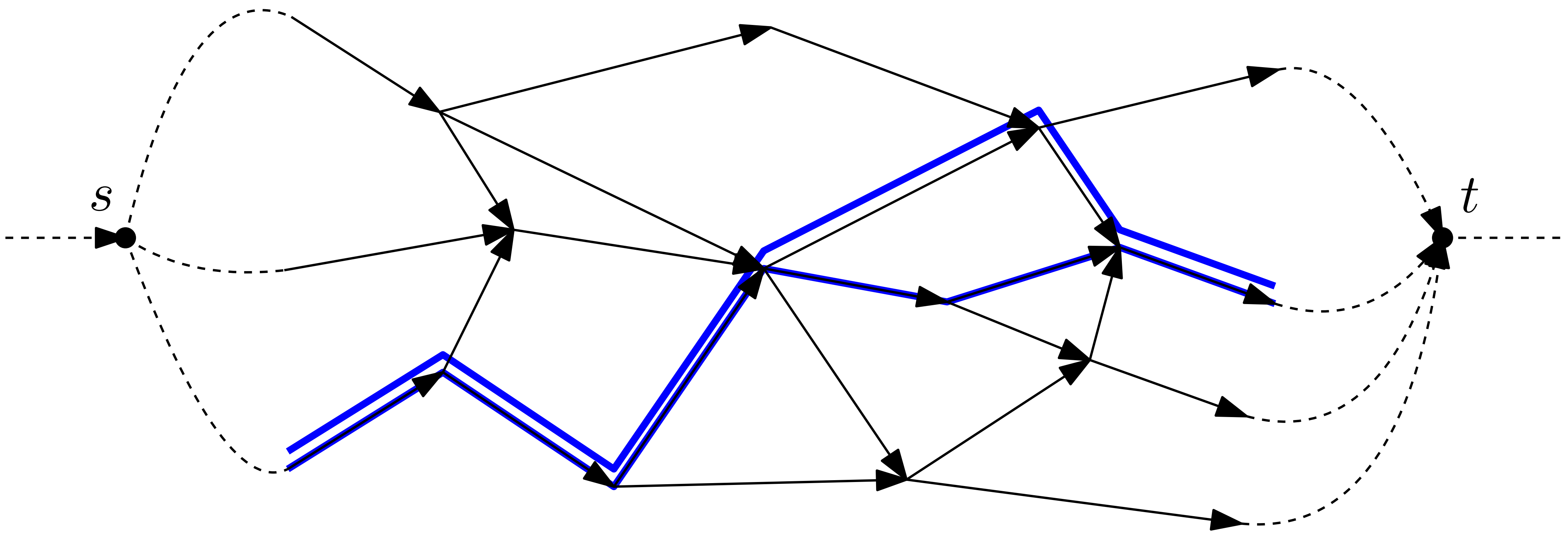
The required rope length

This is really about *bipolar orientations* (s-t-planar DAGs):



The required rope length

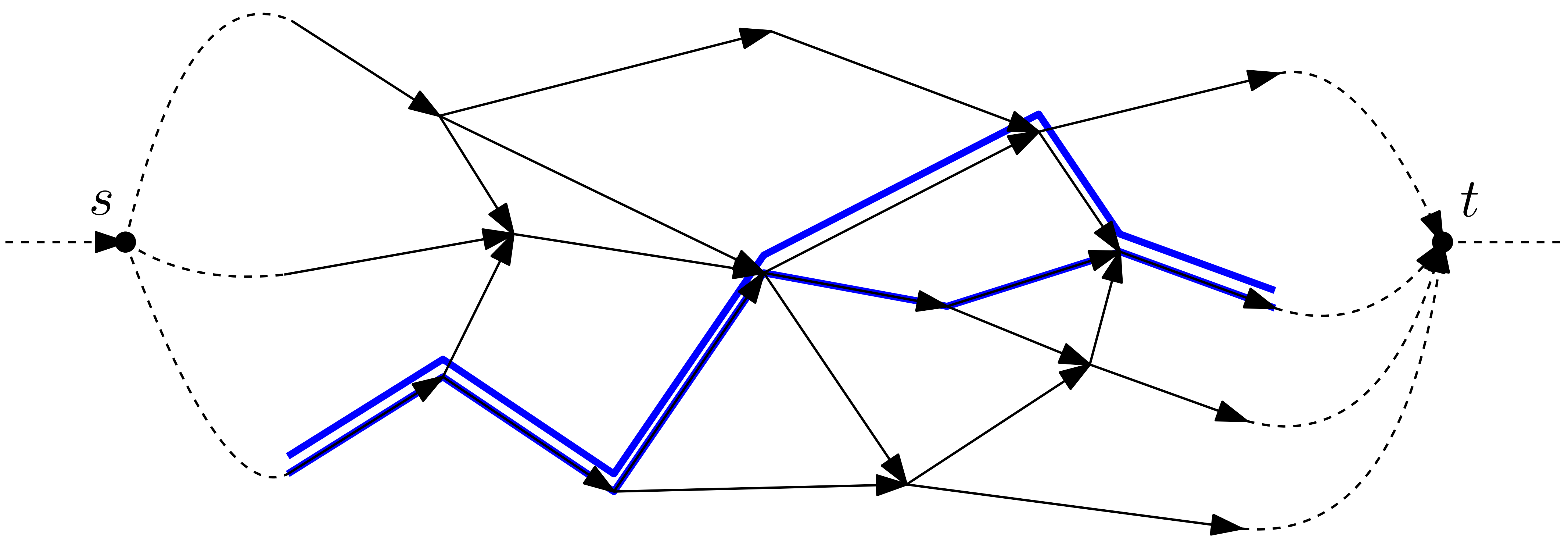
This is really about *bipolar orientations* (s-t-planar DAGs):



The required rope length

Freie Universität Berlin

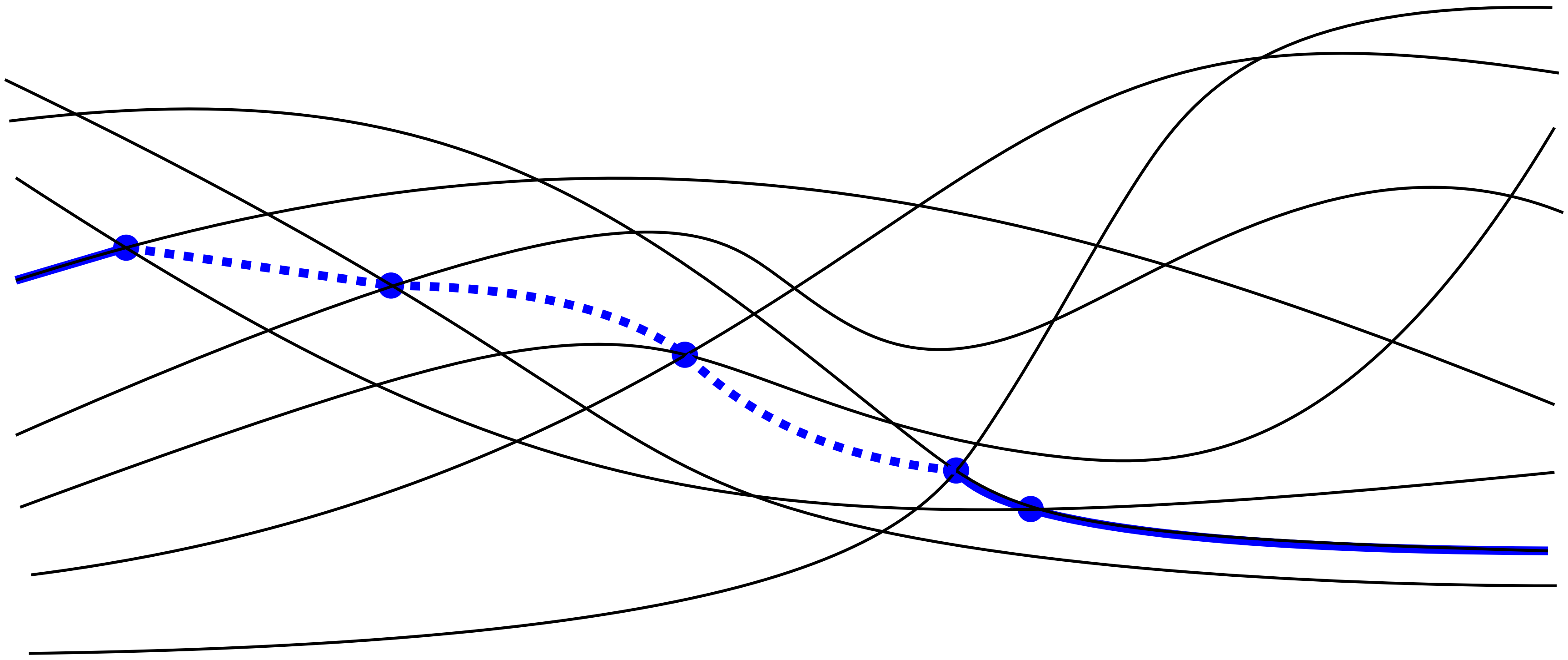
This is really about *bipolar orientations* (s-t-planar DAGs):



“leftmost-first” greedy sweep

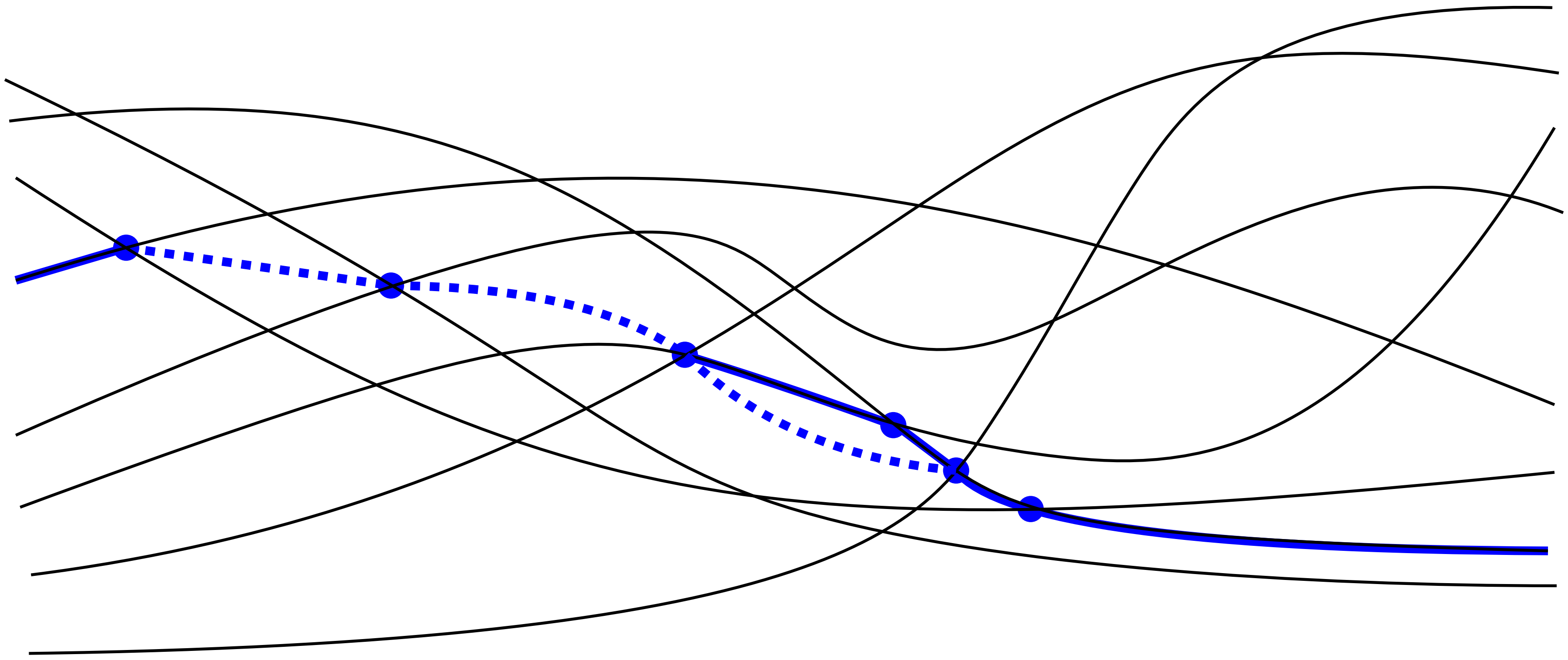
$\rightarrow$ coordinated simultaneous primal-dual sweep

What really matters in practice



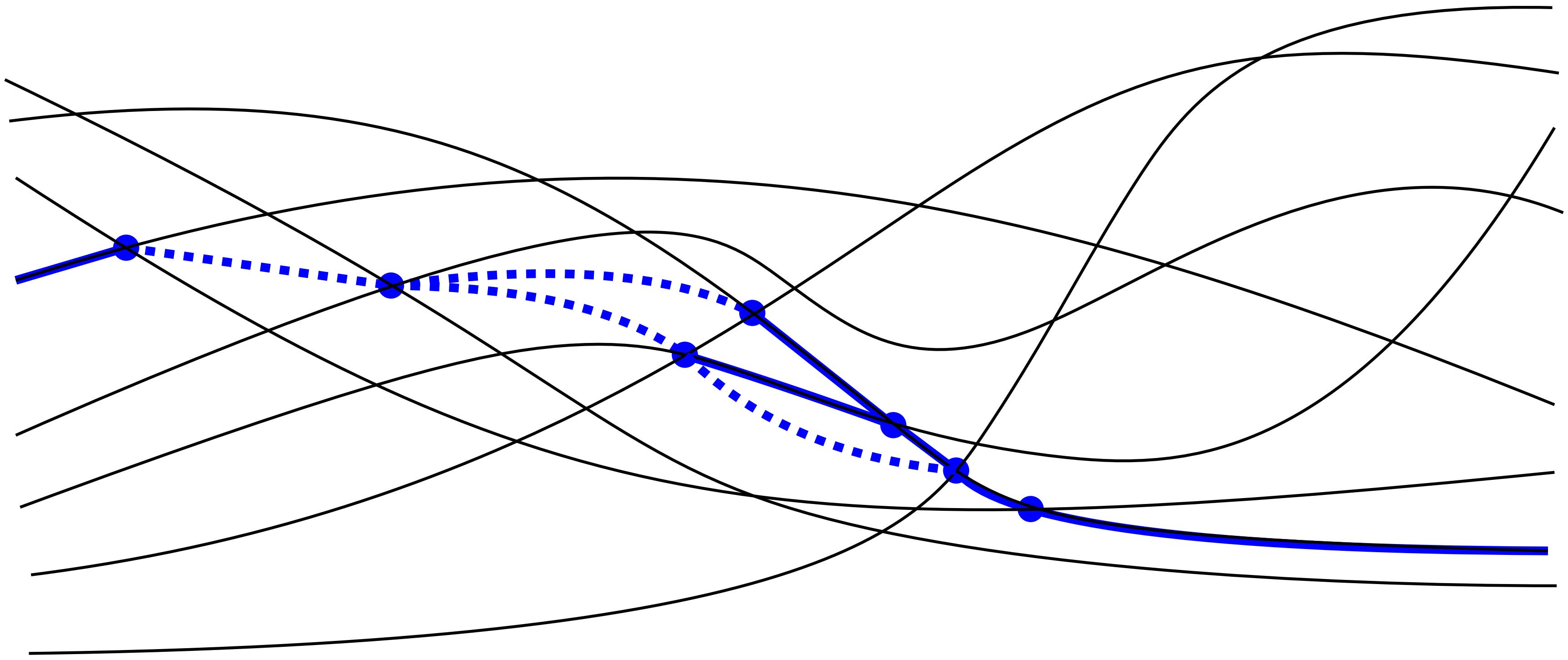
- several *distribute* steps simultaneously, followed by *collect*s
- *cross* steps separately

What really matters in practice



- several *distribute* steps simultaneously, followed by *collect*s
- *cross* steps separately

What really matters in practice



- several *distribute* steps simultaneously, followed by *collect*s
- *cross* steps separately