Accelerating Simulated Annealing for the Permanent and Combinatorial Counting Problems

Ivona Bezáková
Daniel Štefankovič
Vijay V. Vazirani
Eric Vigoda

Accelerating Simulated Annealing for the Permanent and Combinatorial Counting Problems

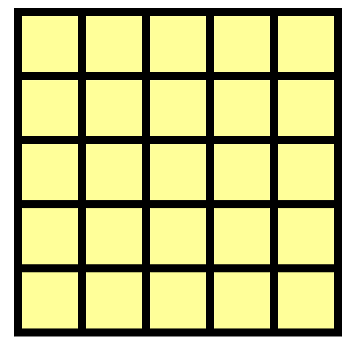
Talk outline:

1. The Permanent problem
2. Simulated annealing for the Permanent (MCMC algorithm by JSV '01)
3. New simulated annealing schedule

Permanent of an nxn matrix *A*

$$\mathrm{Per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

A =

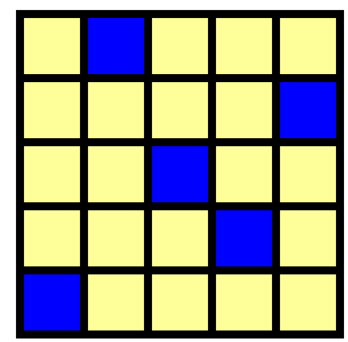


History & motivation:

- defined by Cauchy [1812]
- used in a variety of areas: statistical physics, statistics, vision, anonymization systems, …

Permanent of an nxn matrix *A*

$$\mathrm{Per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

A = 

π

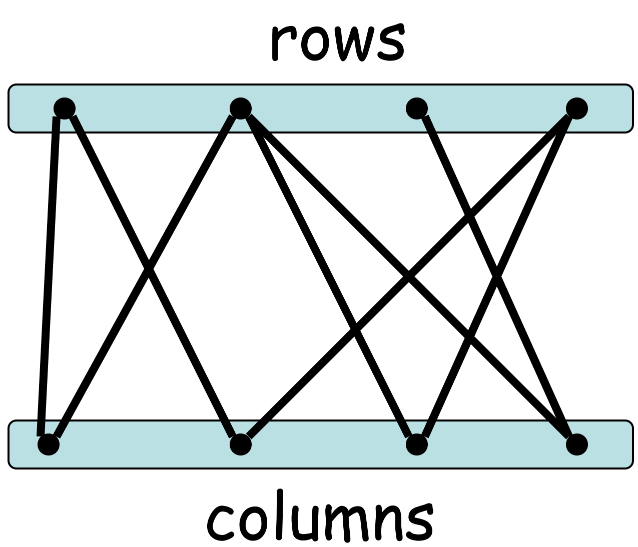
History & motivation:

- defined by Cauchy [1812]
- used in a variety of areas: statistical physics, statistics, vision, anonymization systems, …

Permanent of an nxn matrix A

$$\mathrm{Per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

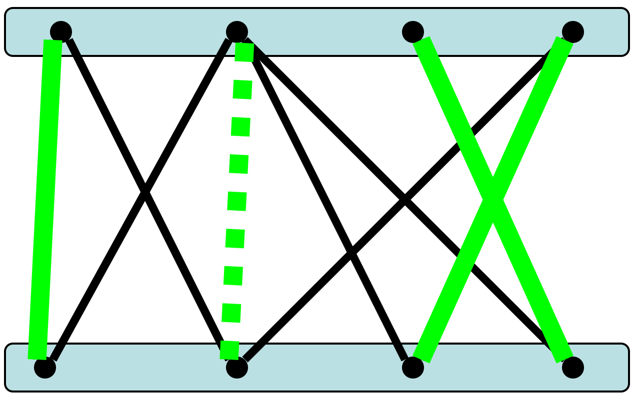
A – binary (entries 0 or 1):
adjacency matrix of a bipartite graph



Permanent of an nxn matrix A

$$\mathrm{Per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

A – binary (entries 0 or 1):
adjacency matrix of a bipartite graph

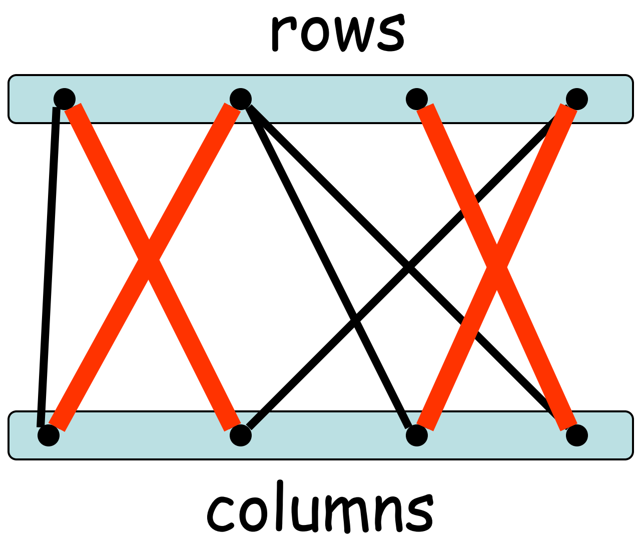


Permanent of an nxn matrix A

$$\text{Per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

A - binary (entries 0 or 1):
adjacency matrix of a bipartite graph

The permanent counts the number of perfect matchings.



Previous Work on the Permanent Problem

[Kasteleyn '67]
poly-time for planar graphs (bipartite or not)

[Valiant '79]
#P-complete for non-planar graphs

[Jerrum-Sinclair '89]
fpras for special graphs, e.g. the dense graphs, based on a Markov chain by Broder '88

[Jerrum-Sinclair-Vigoda '01 & '05]
$O^*(n^{26})$ fpras for any bipartite graph, later $O^*(n^{10})$

Our result:

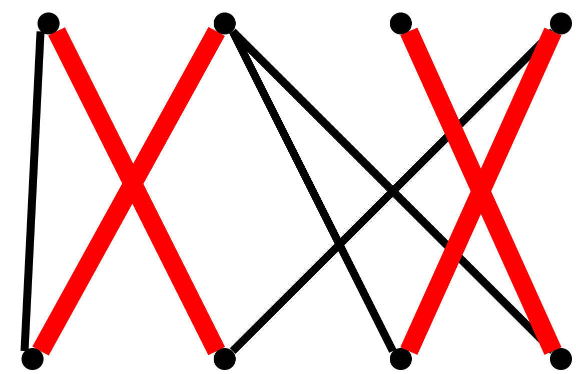
$$O^*(n^7)$$

Broder chain

uniform sampling of perfect matchings of a given graph

At a perfect matching:

- remove a random edge

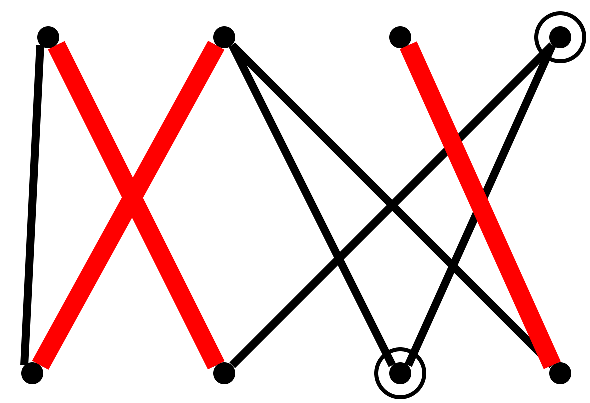


At a near-matching:

- pick a vertex at random
  - if a hole, try to match with the other hole
  - otherwise slide (if can)

Broder chain

uniform sampling of perfect matchings of a given graph



At a perfect matching:

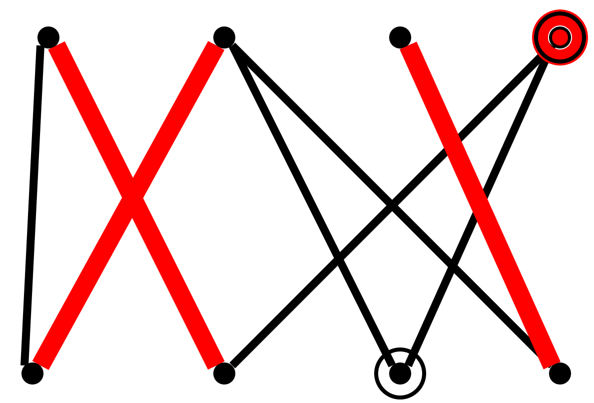
- remove a random edge

At a near-matching:

- pick a vertex at random
  - if a hole, try to match with the other hole
  - otherwise slide (if can)

Broder chain

uniform sampling of perfect matchings of a given graph



At a perfect matching:

- remove a random edge

At a near-matching:

- pick a vertex at random
  - if a hole, try to match with the other hole
  - otherwise slide (if can)

Broder chain

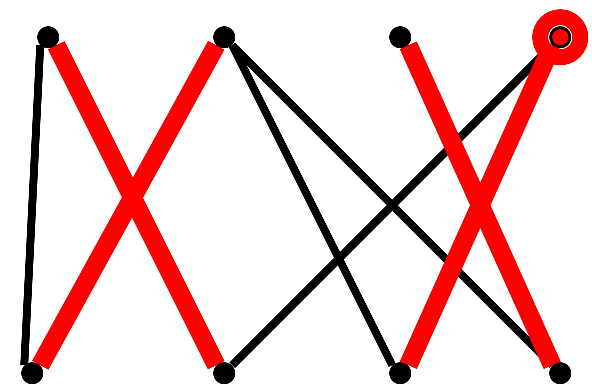
uniform sampling of perfect matchings of a given graph

At a perfect matching:

- remove a random edge

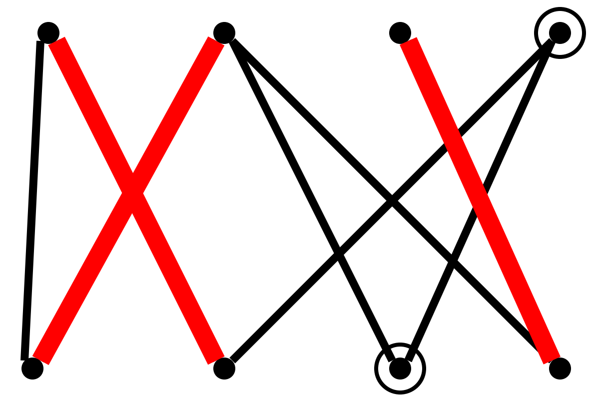
At a near-matching:

- pick a vertex at random
  - if a hole, try to match with the other hole
  - otherwise slide (if can)



Broder chain

uniform sampling of perfect matchings of a given graph



At a perfect matching:

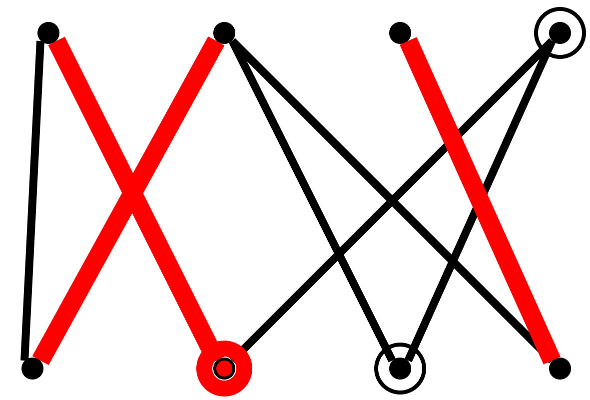
- remove a random edge

At a near-matching:

- pick a vertex at random
  - if a hole, try to match with the other hole
  - otherwise slide (if can)

Broder chain

uniform sampling of perfect matchings of a given graph



At a perfect matching:

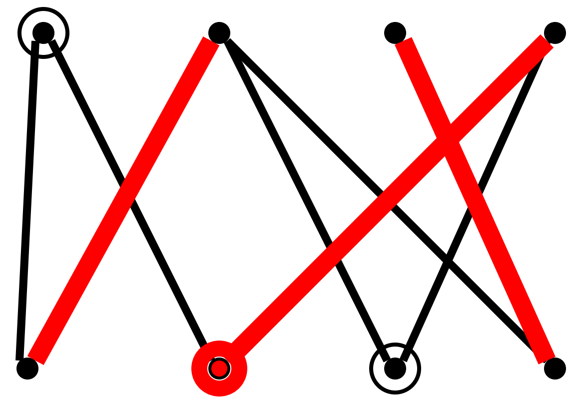
- remove a random edge

At a near-matching:

- pick a vertex at random
    - if a hole, try to match with the other hole
    - otherwise slide (if can)

Broder chain

uniform sampling of perfect matchings of a given graph



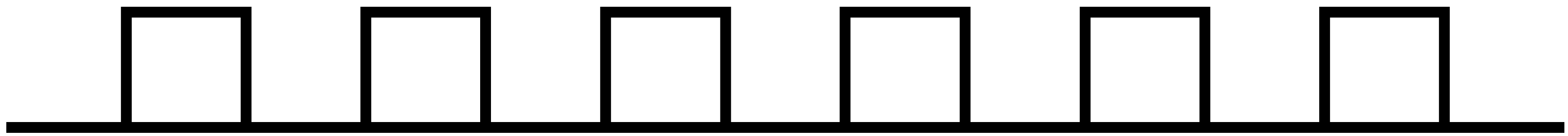
At a perfect matching:

- remove a random edge

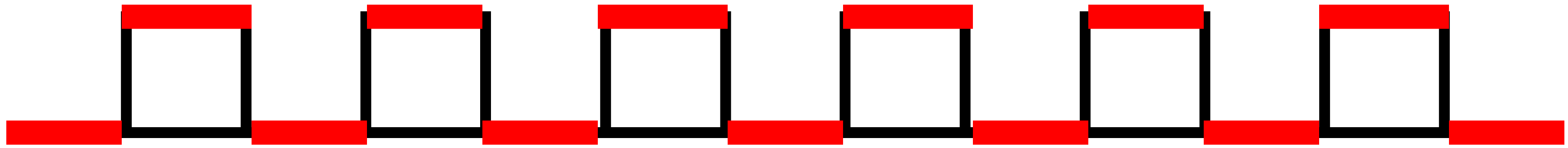
At a near-matching:

- pick a vertex at random
  - if a hole, try to match with the other hole
  - otherwise slide (if can)

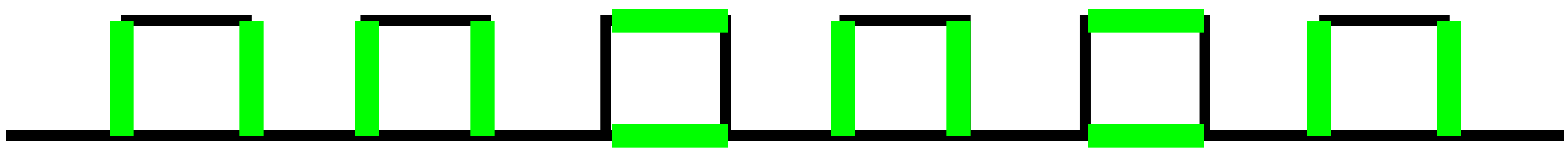
Does the Broder chain mix in polynomial time?



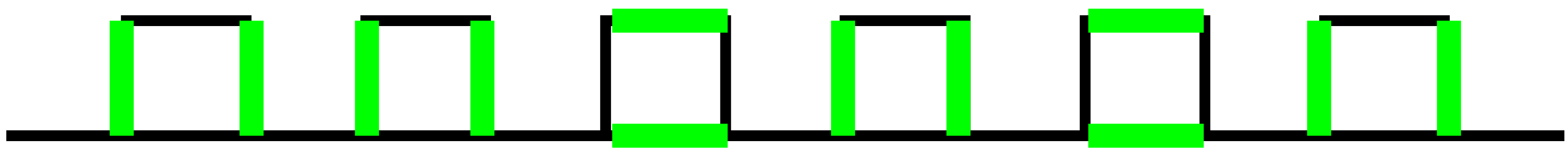
Does the Broder chain mix in polynomial time?

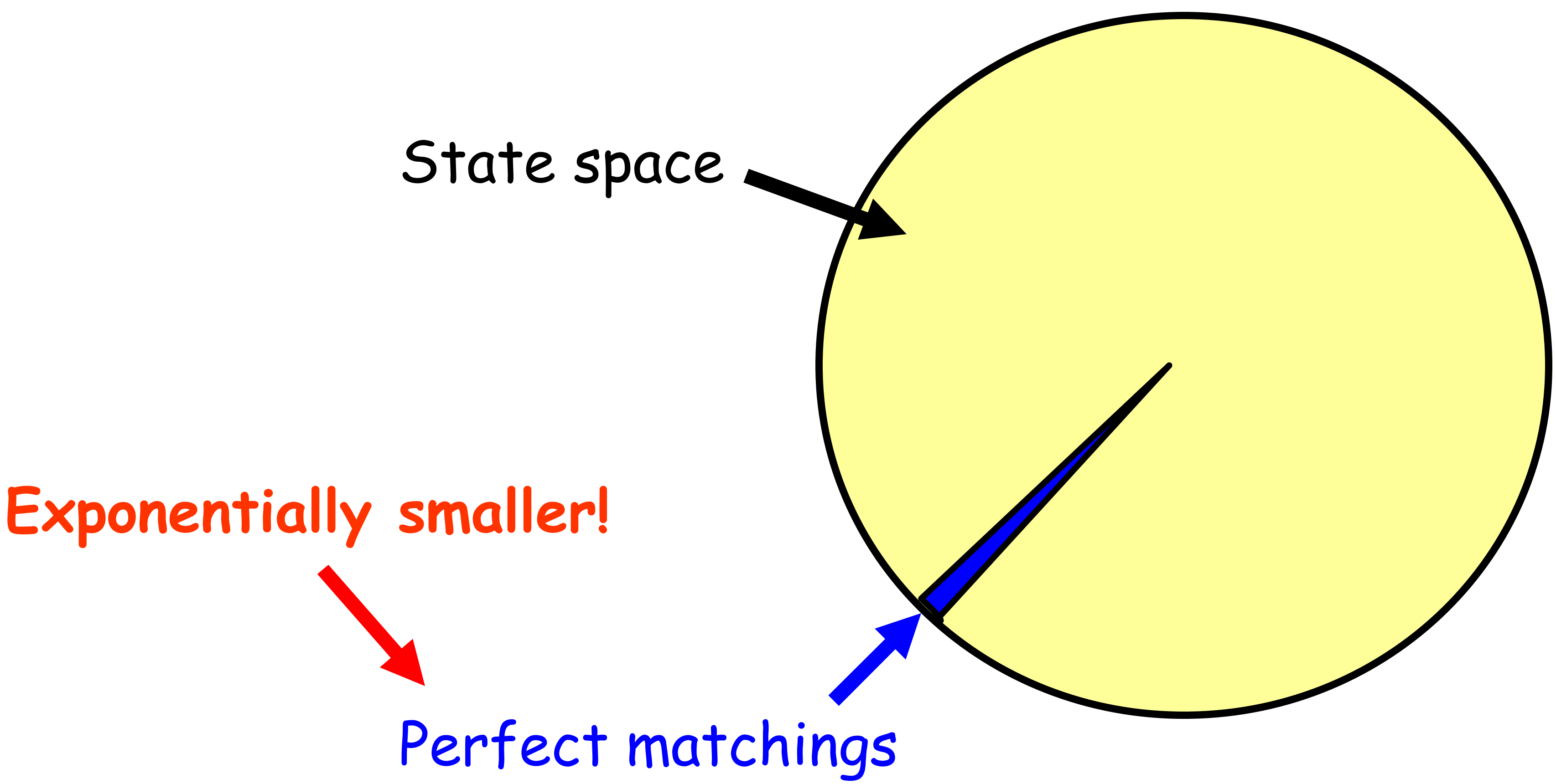


Does the Broder chain mix in polynomial time?

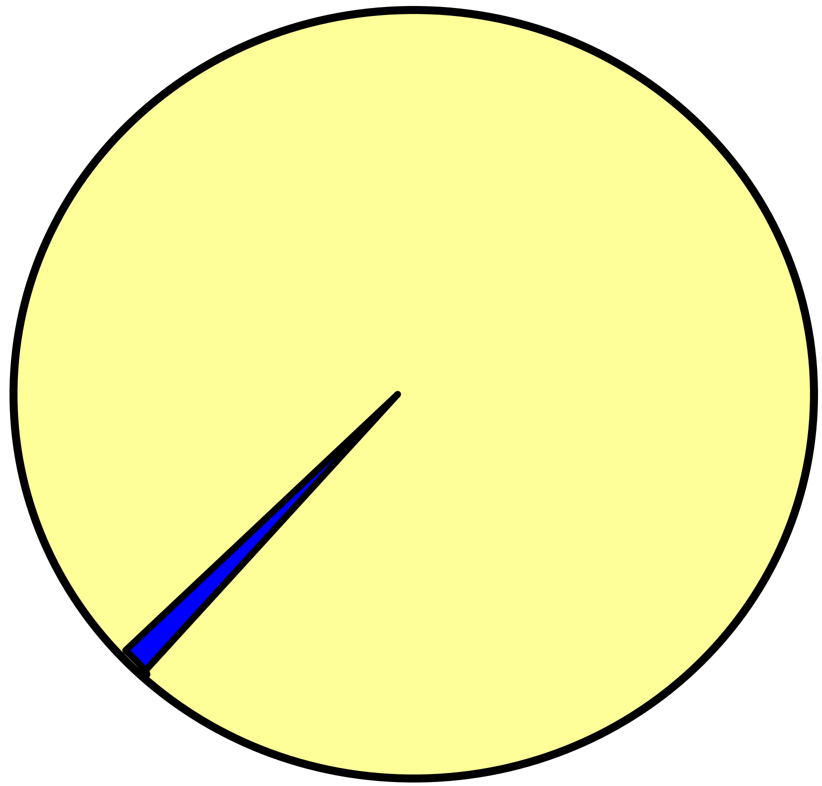


Does the Broder chain mix in polynomial time?



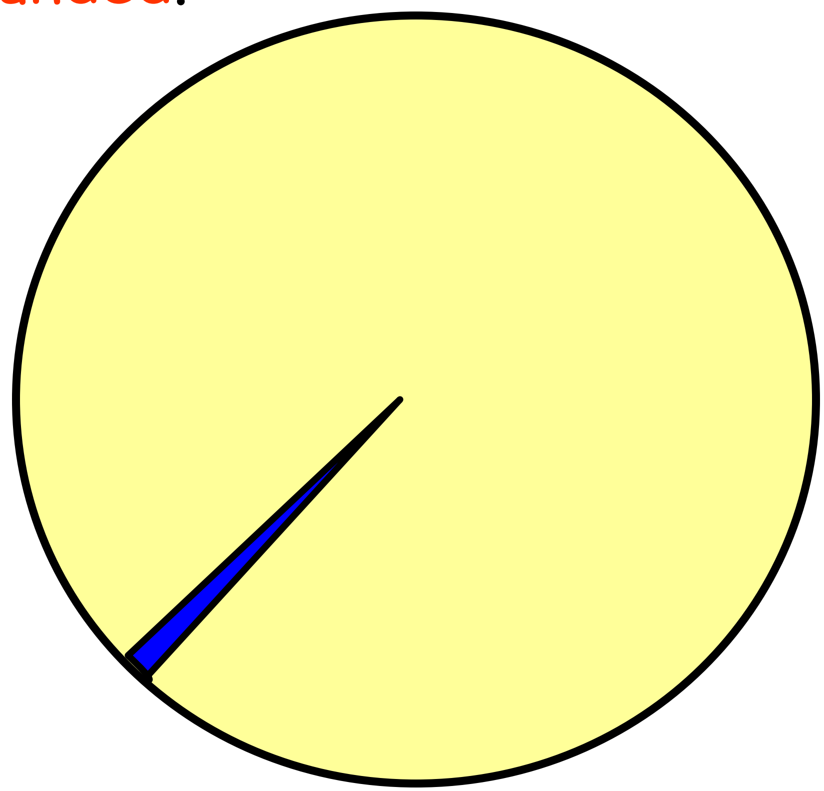


Theorem[JS]: Rapid mixing if perfect matchings polynomially related to near-perfect matchings.



Theorem[JS]: Rapid mixing if perfect matchings polynomially related to near-perfect matchings.

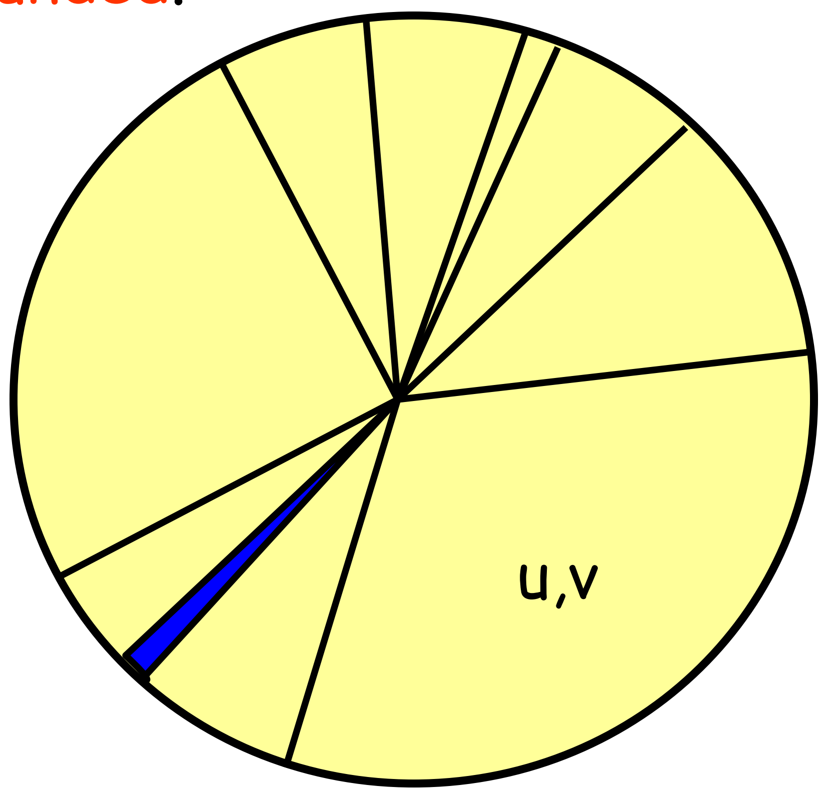
Idea[JSV]: Weight the states so that the weighted ratio is always polynomially bounded.



Theorem[JS]: Rapid mixing if perfect matchings polynomially related to near-perfect matchings.

Idea[JSV]: Weight the states so that the weighted ratio is always polynomially bounded.

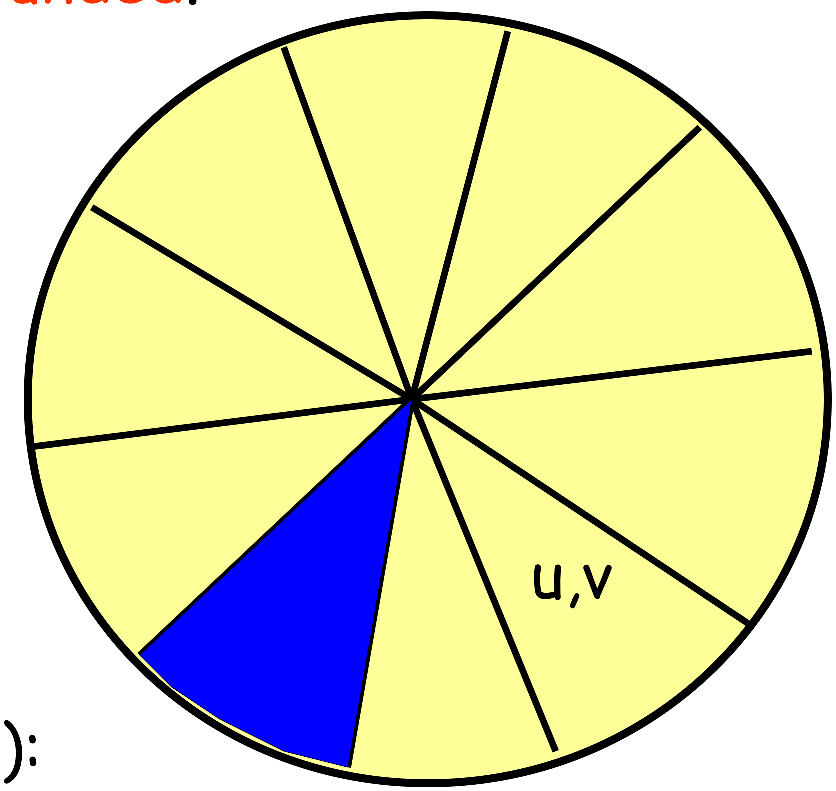
n^2+1 regions, very different weight



Theorem[JS]: Rapid mixing if perfect matchings polynomially related to near-perfect matchings.

Idea[JSV]: Weight the states so that the weighted ratio is always polynomially bounded.

n^2+1 regions, each about the same weight

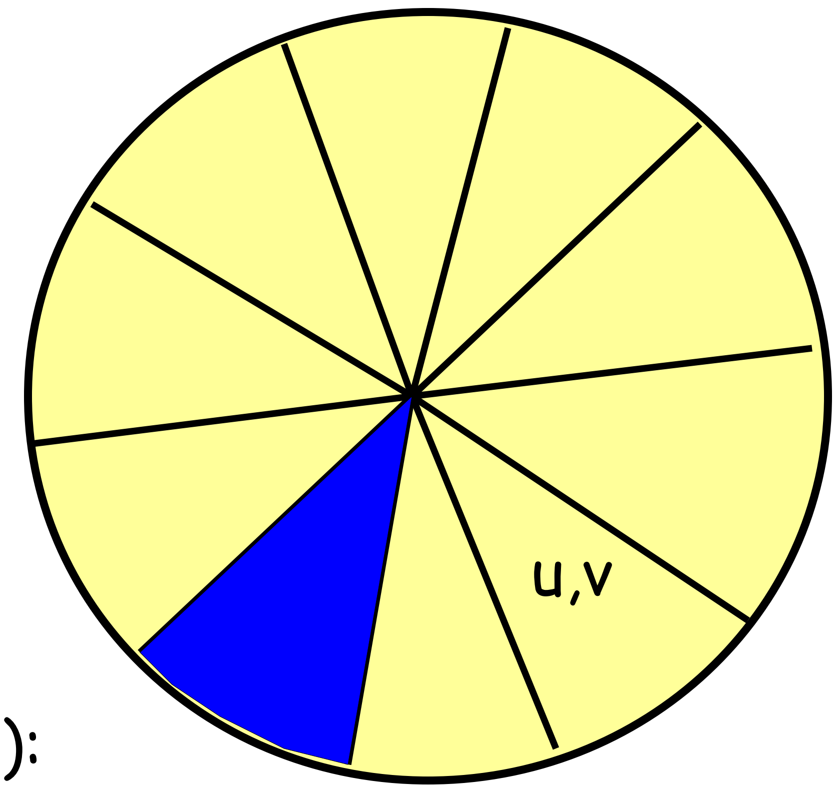


Ideal weights
(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

Good: A perfect matching sampled with prob. $1/(n^2+1)$

Bad: Computing ideal weights as hard as original problem?



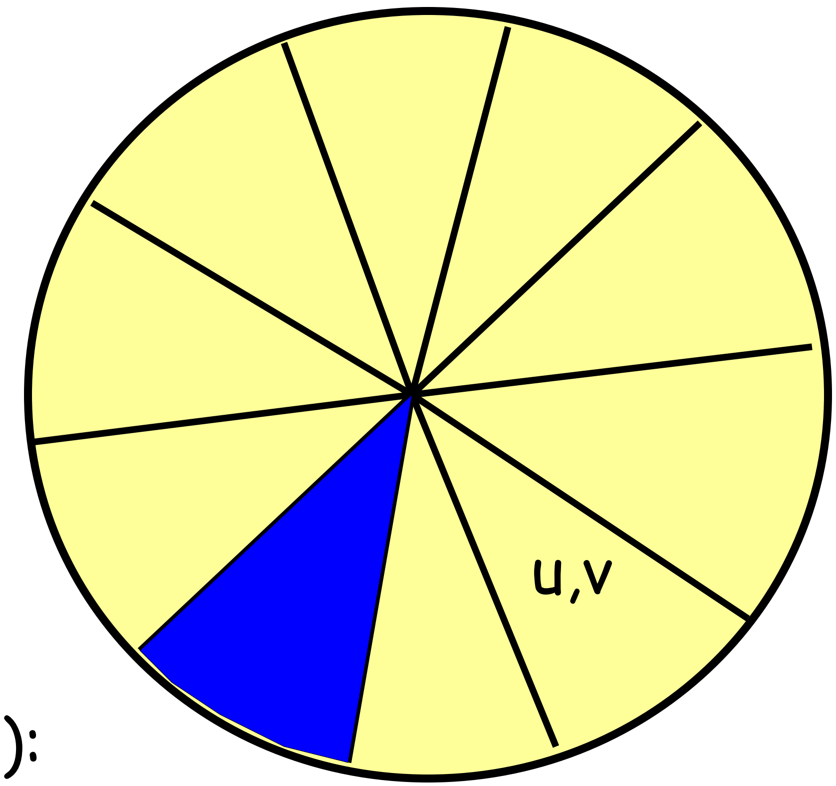
Ideal weights
(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

Good: A perfect matching sampled with prob. $1/(n^2+1)$

Bad: Computing ideal weights as hard as original problem?

Solution: **Approximate** the ideal weights

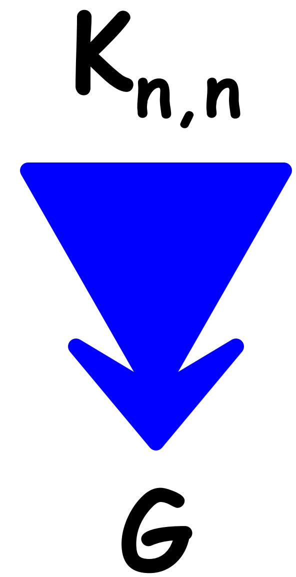


Ideal weights
(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

Simulated Annealing

Solution: **Approximate** the ideal weights



Start with an easy instance,

gradually get to the target instance.

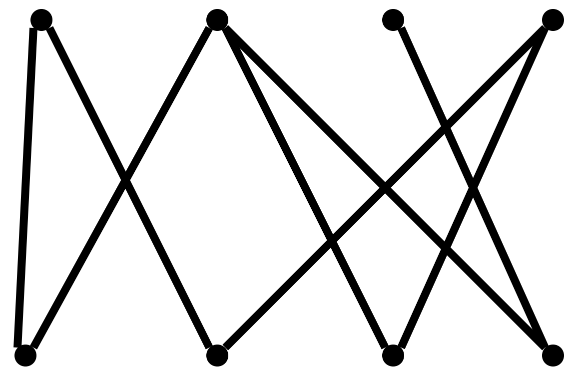
Ideal weights
(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

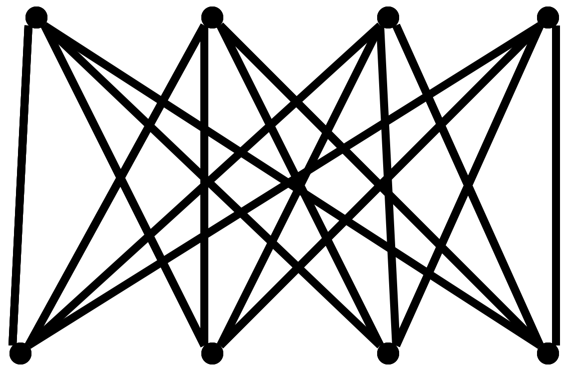


How?

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

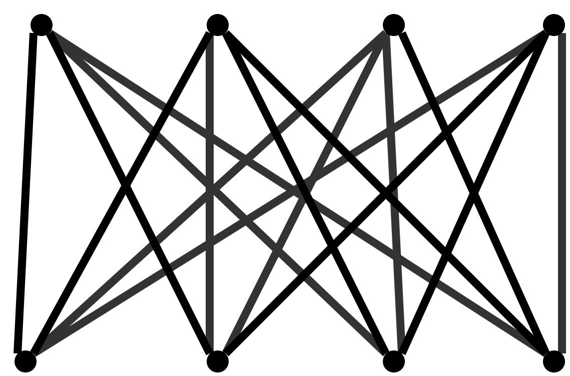


How?

- start with the complete graphs (weights easy to compute)

Ideal weights
(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



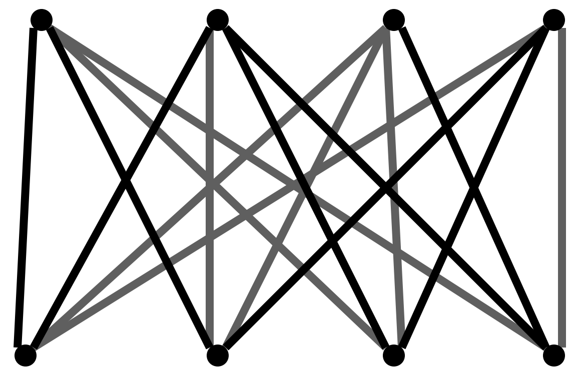
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



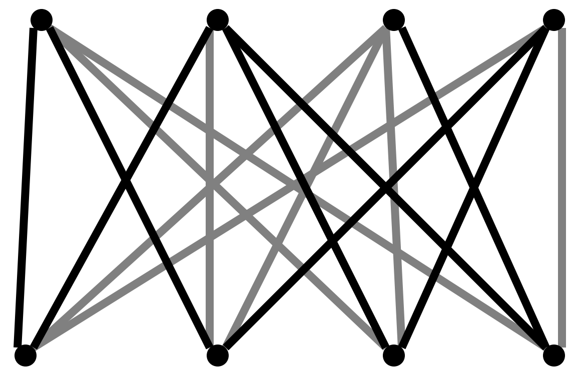
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



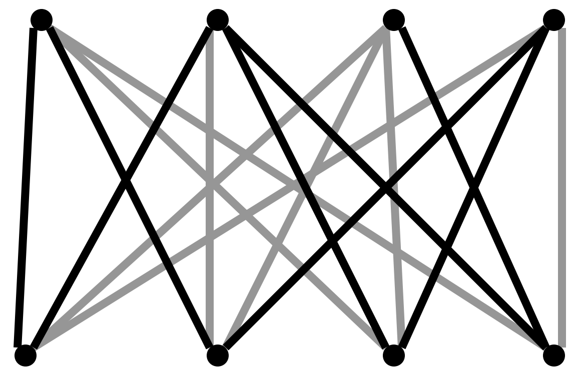
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



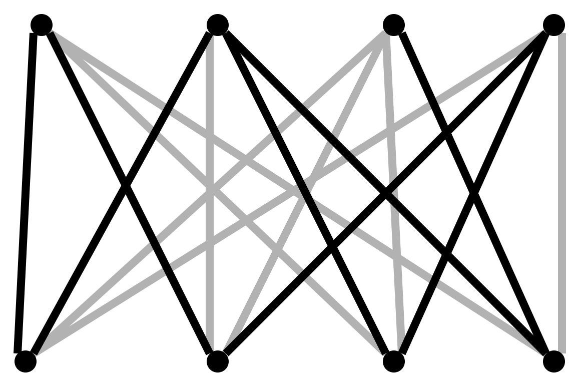
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



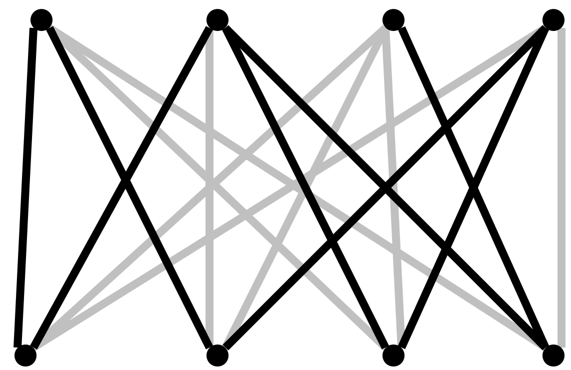
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



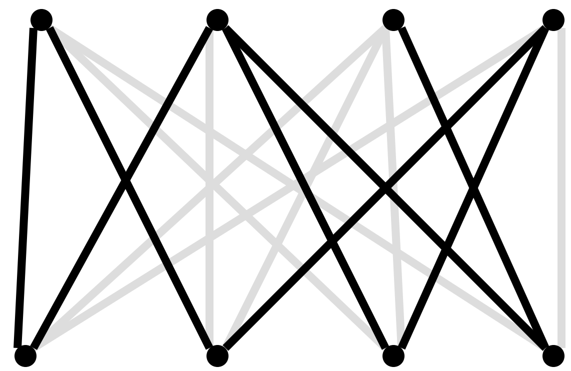
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



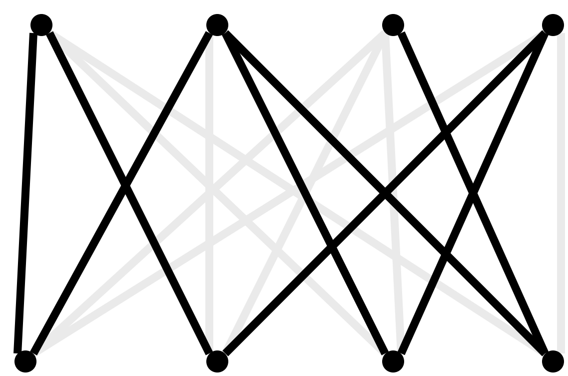
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



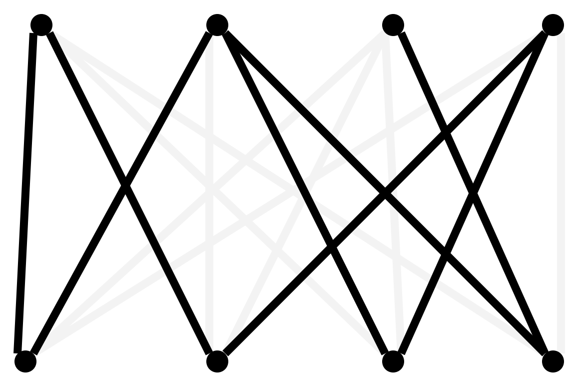
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)



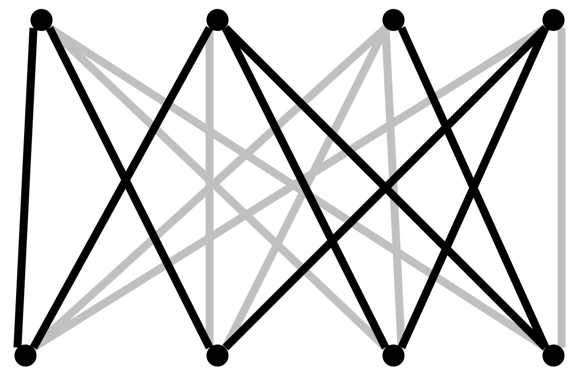
How?

- start with the complete graphs (weights easy to compute)
- fade away non-edges

Ideal weights

(for a matching with holes u,v):

(# perfects) / (# nears with holes u,v)

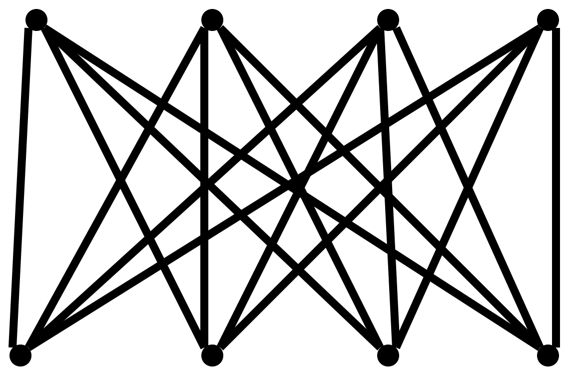


The edges have activities:

- 1 for a real edge
- $\lambda \in [0,1]$ for a non-edge

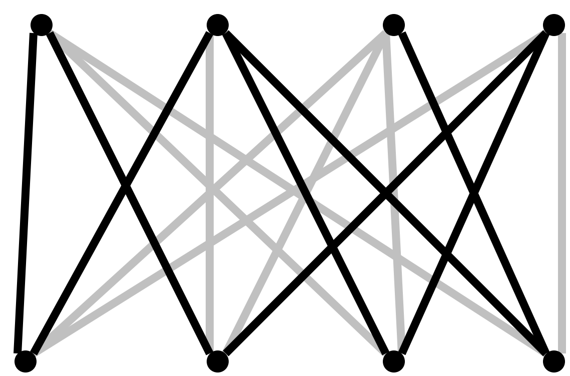
How activities help?

- start with $\lambda = 1$
- compute corresponding weights $n! \,/\, (n-1)!$

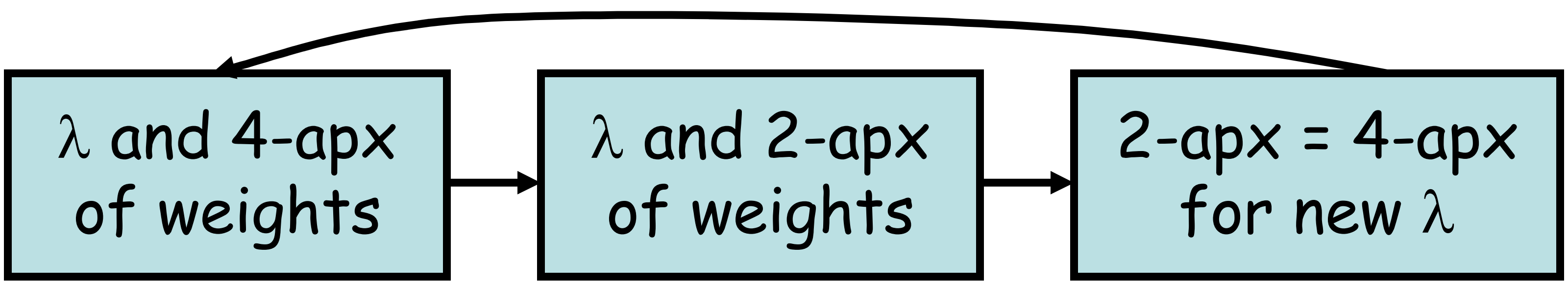


How activities help?

- start with $\lambda = 1$
- compute corresponding weights $n! / (n-1)!$



Repeat until $\lambda < 1/n!$



Running Time [JSV]

Thm: The (λ,hole-weights)-Broder chain mixes in time $O^*(n^6)$.

We need:

$O^*(n^6)$	per sample
$O^*(n^2)$	samples (boosting from 4-apx to 2-apx)
$O^*(n^2)$	λ-decrements (phases)
$O^*(n^{10})$	total to get a 2-apx of the ideal weights

Running Time [BŠVV]

Thm: The (λ,hole-weights)-Broder chain mixes in time ~~$O^*(n^6)$~~ $O^*(n^4)$.

We need:

$O^*(n^4)$	~~$O^*(n^6)$~~	per sample
$O^*(n^2)$	$O^*(n^2)$	samples (boosting from 4-apx to 2-apx)
$O^*(n)$	~~$O^*(n^2)$~~	λ-decrements (phases)
$O^*(n^7)$	~~$O^*(n^{10})$~~	total to get a 2-apx of the ideal weights

Reformulation of the problem

Promise: a set of polynomials of degree n such that

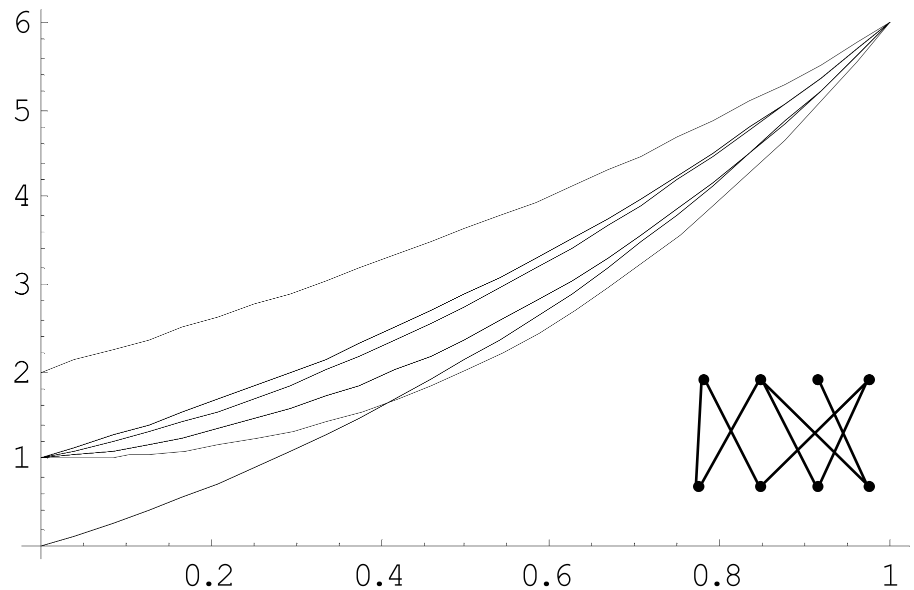
- polynomials have a low-degree term
- non-negative integer coefficients sum to $\leq$ n!

Goal: λ-sequence (from 1 to 1/n!) such that

for every polynomial ratio of consecutive values $\leq$ 2

Tricky part:

Don't know the coefficients!

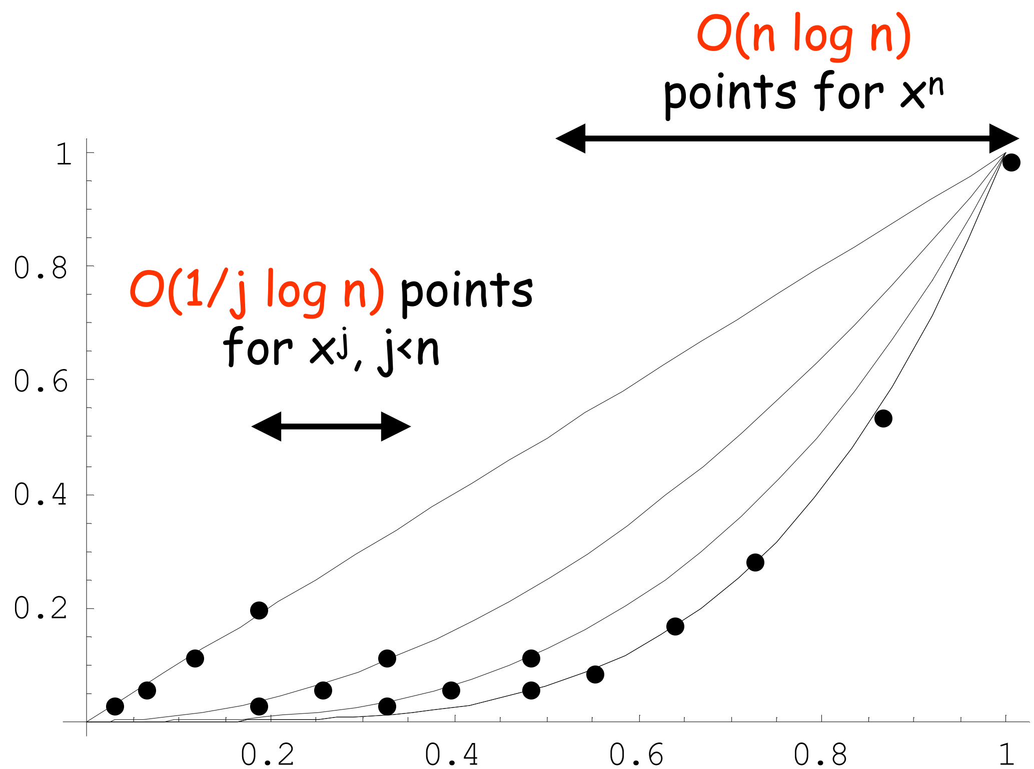


Intuition

The worst case is the set of polynomials x^j, $j=1,\ldots,n$

Problem: no low-degree terms and x^n "dominates"

Fix: if the value of some x^j drops below $1/n!$, ignore the polynomial



TOTAL:

$O(n \log^2 n)$ points

Conclusions

- new cooling schedule: a blackbox, applicable to other problems
- improved analysis of the weighted Broder chain
- interest of practical comunity

Open Problems

- other applications of the cooling schedule
- faster mixing result
- do we need n^2 weights?
- non-bipartite graphs