Markov Chain Mixing Times And Applications III:

Conductance and Canonical Paths

Ivona Bezáková

(Rochester Institute of Technology)

Simons Institute for the Theory of Computing
Counting Complexity and Phase Transitions Bootcamp

January 27th, 2016

Outline

Other techniques for bounding the mixing time:

- conductance
- canonical paths
- canonical flows

Thanks to:

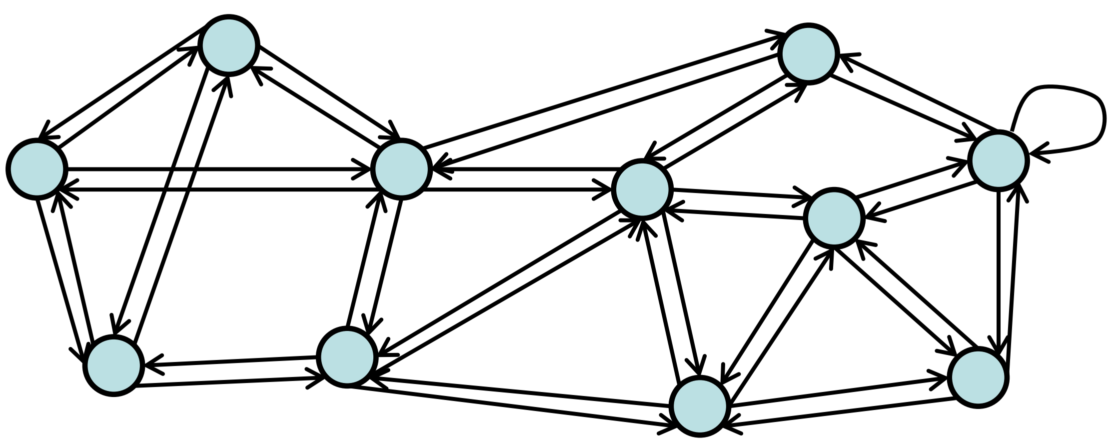
Bhatnagar, Diaconis, Dyer, Jerrum, Lawler, Müller, Randall, Sinclair, Sokal, Štefankovič, Stroock, Vazirani, Vigoda, …

Outline

Recall:

- Ergodic MC (Ω,P) => unique stationary distribution π
- Mixing time: $t_{mix}(\epsilon)$ = minimum t such that for every start state x, after t steps within ϵ of π

An ergodic reversible Markov chain (Ω,P):

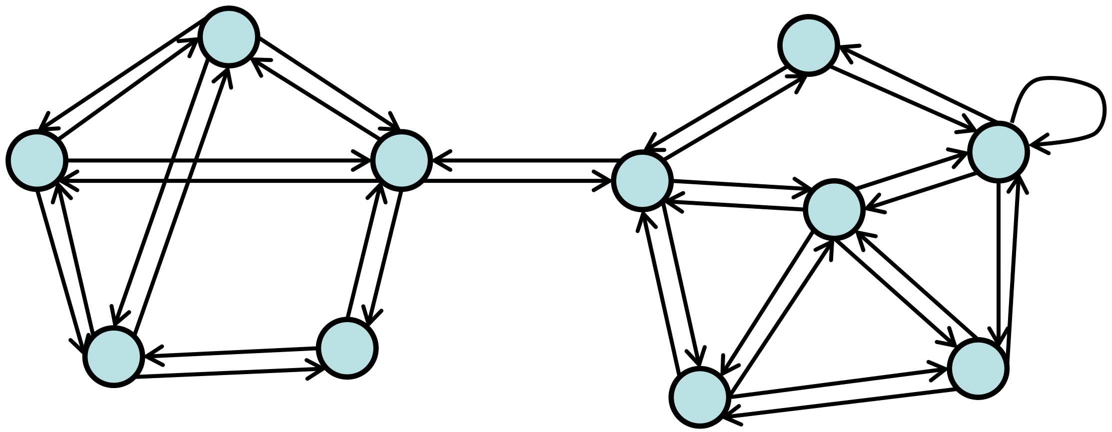


Outline

Recall:

- Ergodic MC (Ω,P) => unique stationary distribution π
- Mixing time: $t_{mix}(\epsilon)$ = minimum t such that for every start state x, after t steps within ϵ of π

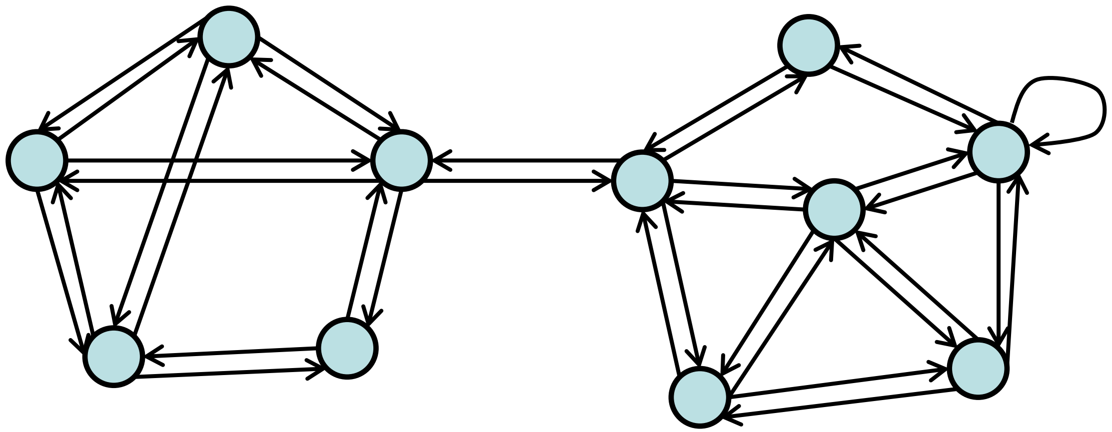
An ergodic reversible Markov chain (Ω,P):



Conductance

Def: For an ergodic reversible MC (Ω,P), its conductance is defined as:

$$\Phi = \min_{S: S \subseteq \Omega, \pi(S) \le 1/2} \frac{\sum_{x \in S, y \notin S} \pi(x) P(x, y)}{\pi(S)}$$

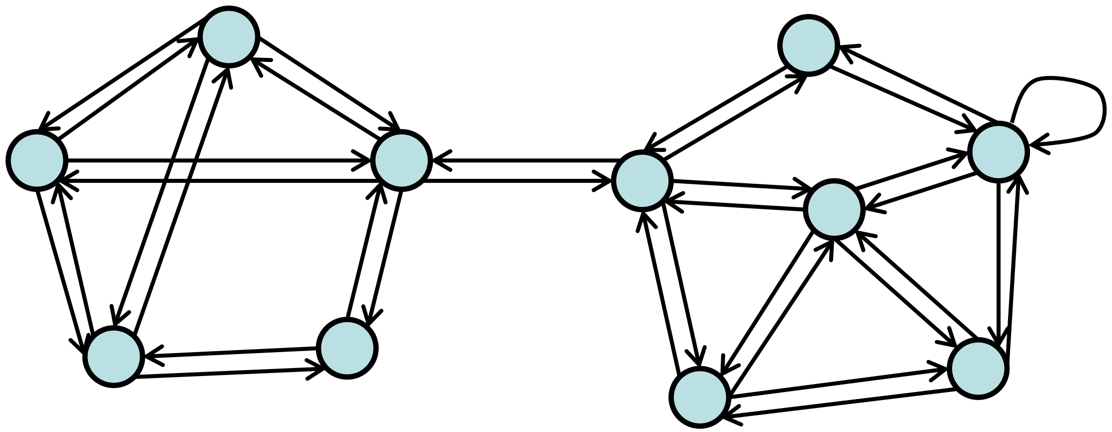


Conductance

Def: For an ergodic reversible MC (Ω, P), its conductance is defined as:

$$\Phi = \min_{S: S \subseteq \Omega, \pi(S) \le 1/2} \frac{\sum_{x \in S, y \notin S} \pi(x) P(x, y)}{\pi(S)}$$

Example: suppose π is uniform:

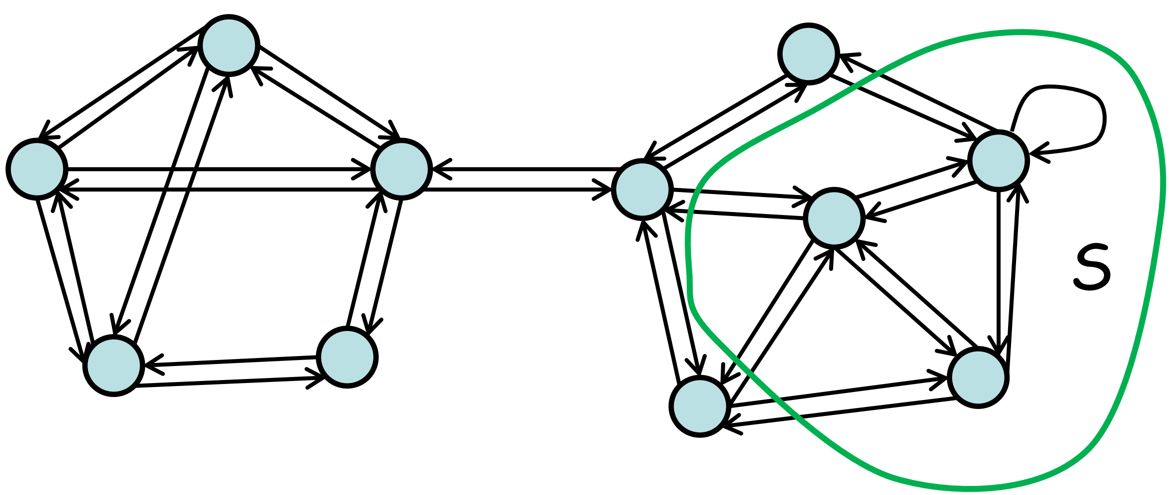


Conductance

Def: For an ergodic reversible MC (Ω,P), its conductance is defined as:

$$\Phi = \min_{S: S \subseteq \Omega, \pi(S) \le 1/2} \frac{\sum_{x \in S, y \notin S} \pi(x) P(x,y)}{\pi(S)}$$

Example: suppose π is uniform:



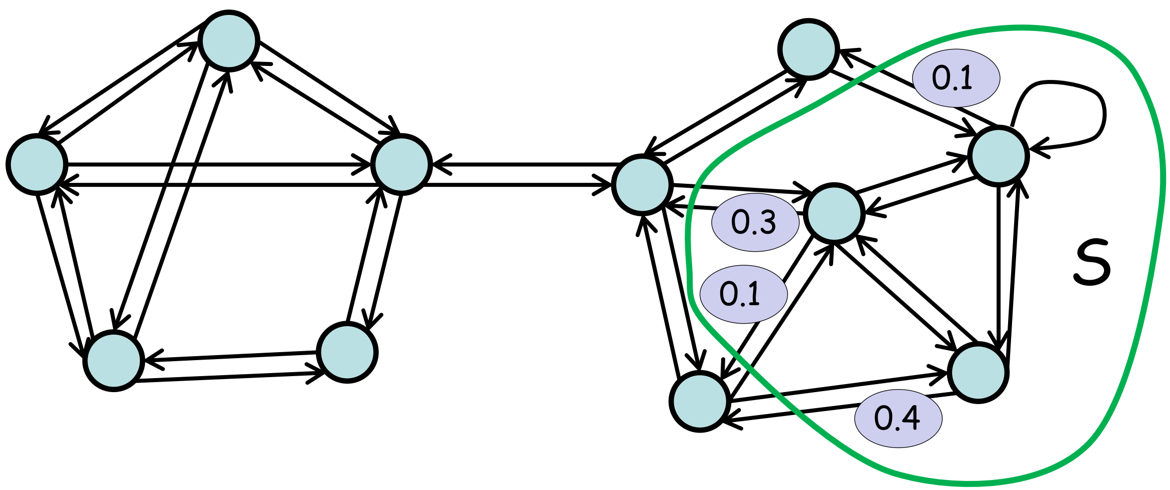
π(S) = 3/11 $\le$ 1/2

Conductance

Def: For an ergodic reversible MC (Ω,P), its conductance is defined as:

$$\Phi = \min_{S:S\subseteq\Omega,\pi(S)\le 1/2} \frac{\sum_{x\in S,y\notin S}\pi(x)P(x,y)}{\pi(S)}$$

Example: suppose π is uniform:



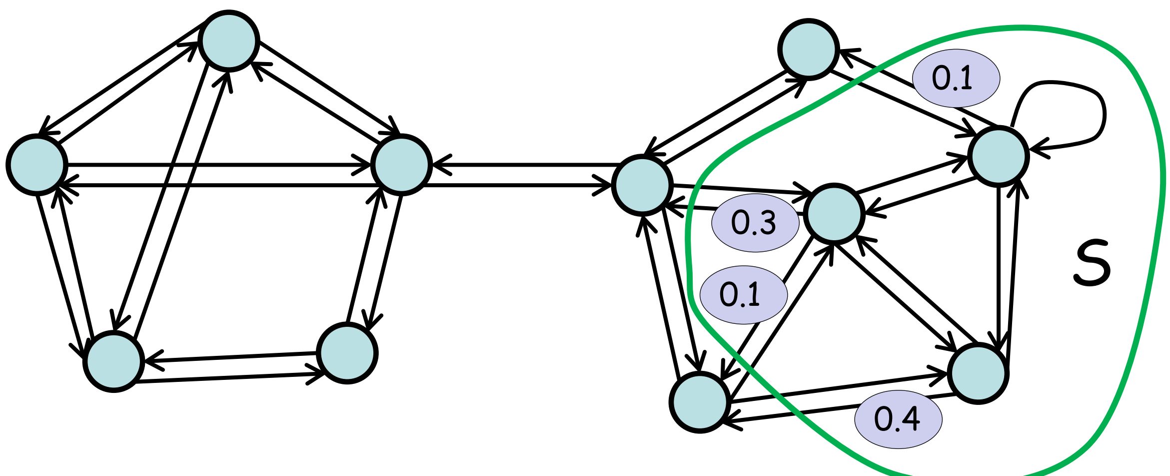
$\pi(S) = 3/11 \le 1/2$

Conductance

Def: For an ergodic reversible MC (Ω, P), its conductance is defined as:

$$\Phi = \min_{S:S\subseteq\Omega,\pi(S)\le 1/2} \boxed{\frac{\sum_{x\in S, y\notin S} \pi(x)P(x,y)}{\pi(S)}}$$

Example: suppose π is uniform: $\Phi_S = \dfrac{\frac{1}{11}0.1+\frac{1}{11}0.3+\frac{1}{11}0.1+\frac{1}{11}0.4}{\frac{3}{11}} = 0.3$



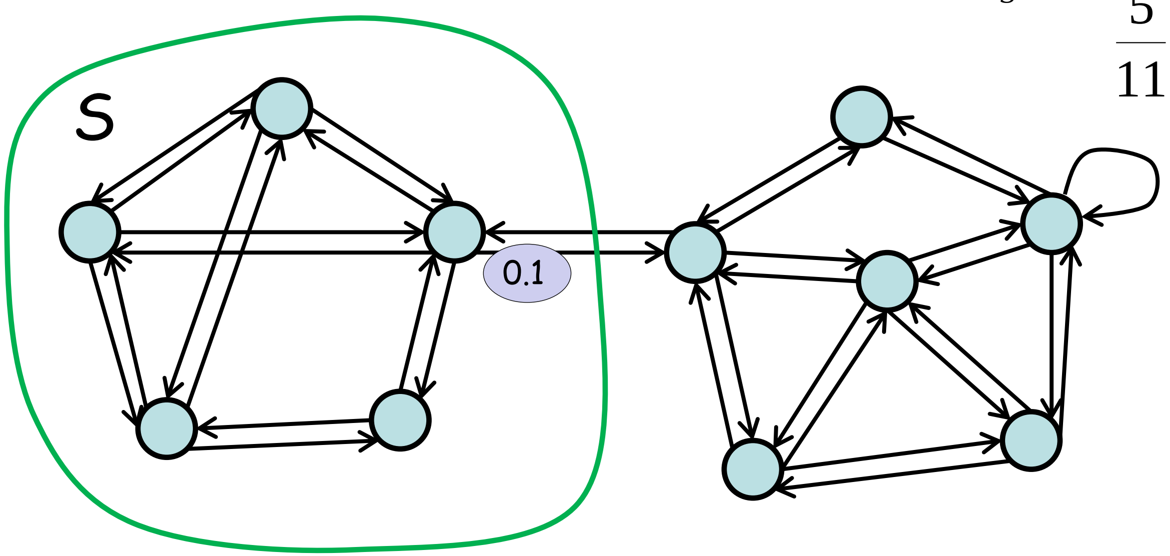
$\pi(S) = 3/11 \le 1/2$

Conductance

Def: For an ergodic reversible MC (Ω, P), its conductance is defined as:

$$\Phi = \min_{S: S\subseteq\Omega, \pi(S)\le 1/2} \boxed{\frac{\sum_{x\in S, y\notin S} \pi(x)P(x,y)}{\pi(S)}}$$

Example: suppose π is uniform: $\Phi_S = \dfrac{\frac{1}{11}0.1}{\frac{5}{11}} = 0.02$



$\pi(S) = 5/11 \le 1/2$

Conductance

Def: For an ergodic reversible MC (Ω,P), its conductance is defined as:

$$\Phi = \min_{S:S\subseteq\Omega,\pi(S)\leq 1/2} \frac{\sum_{x\in S, y\notin S} \pi(x)P(x,y)}{\pi(S)}$$

Thm: $\boxed{\Phi^2/2 \leq \textit{spectral gap} \leq 2\Phi}$

Recall:

Thm: For an ergodic MC, let λ_2 be the 2nd largest eigenvalue of P and $\pi_{\min} := \min_x \pi(x)$. Then

$$\frac{|\lambda_2|}{\textit{spectral gap}} \log\left(\frac{1}{2\varepsilon}\right) \leq t_{mix}(\varepsilon) \leq \frac{1}{\textit{spectral gap}} \log\left(\frac{1}{\varepsilon\pi_{\min}}\right)$$

[Jerrum-Sinclair, Diaconis-Stroock, Lawler-Sokal]

Conductance

Def: For an ergodic reversible MC (Ω,P), its conductance is defined as:

$$\Phi = \min_{S:S\subseteq\Omega,\pi(S)\le 1/2} \frac{\sum_{x\in S,y\notin S}\pi(x)P(x,y)}{\pi(S)}$$

Thm: $\boxed{\Phi^2/2 \le \textit{spectral gap} \le 2\Phi}$

Thm: For a lazy ergodic MC, where $\pi_{\min} := \min_x \pi(x)$:

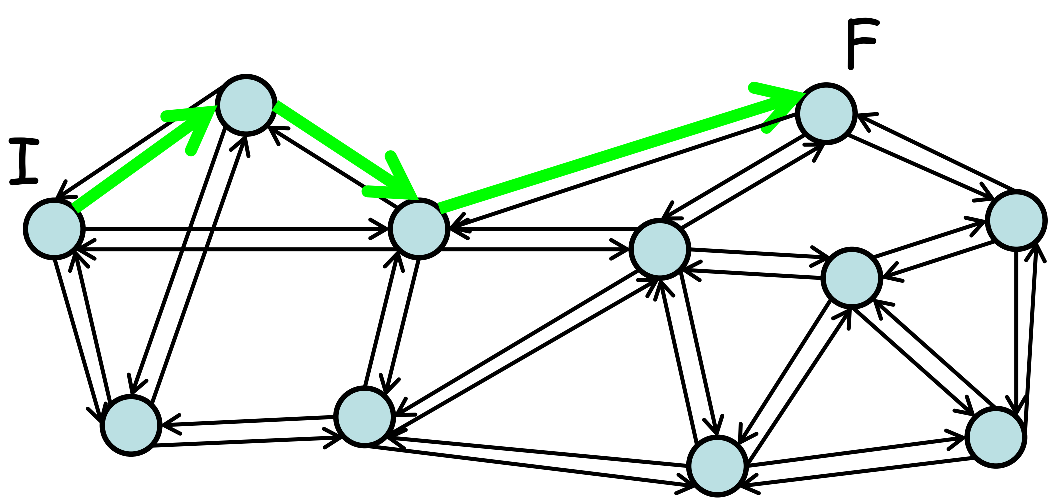
$$\boxed{\frac{1}{2}\left(\frac{1}{2\Phi}-1\right)\log\left(\frac{1}{2\varepsilon}\right) \le t_{mix}(\varepsilon) \le \frac{2}{\Phi^2}\log\left(\frac{1}{\varepsilon\pi_{\min}}\right)}$$

[Jerrum-Sinclair, Diaconis-Stroock, Lawler-Sokal]

Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F:

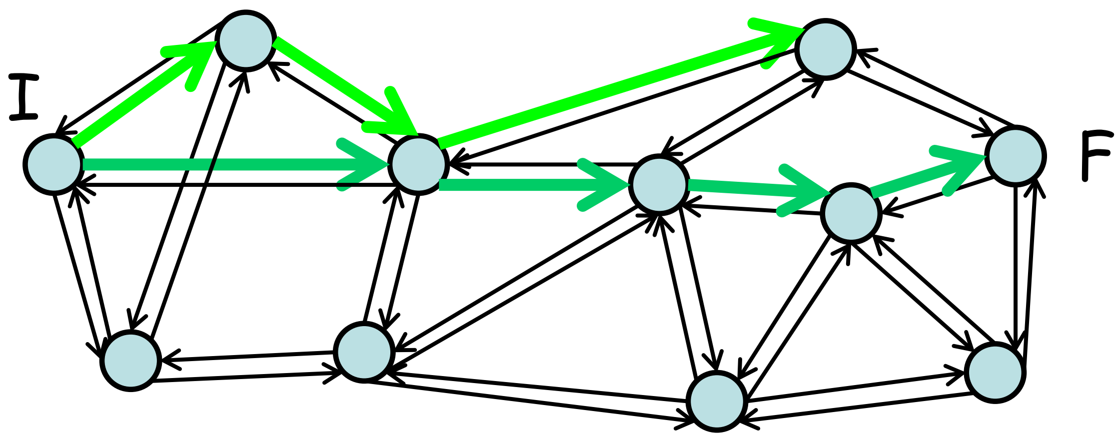


[Jerrum-Sinclair]

Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F ($|\Omega| \times |\Omega|$ paths)

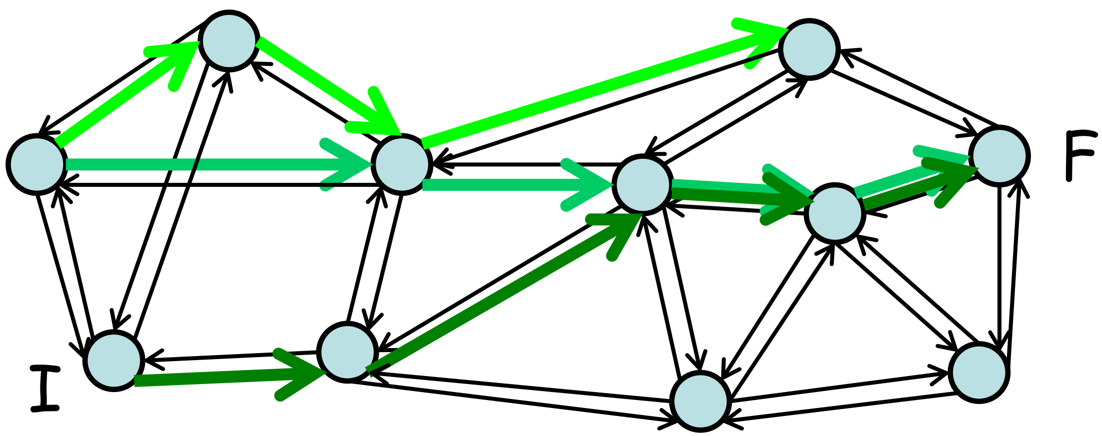


[Jerrum-Sinclair]

Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F ($|\Omega| \times |\Omega|$ paths)



[Jerrum-Sinclair]

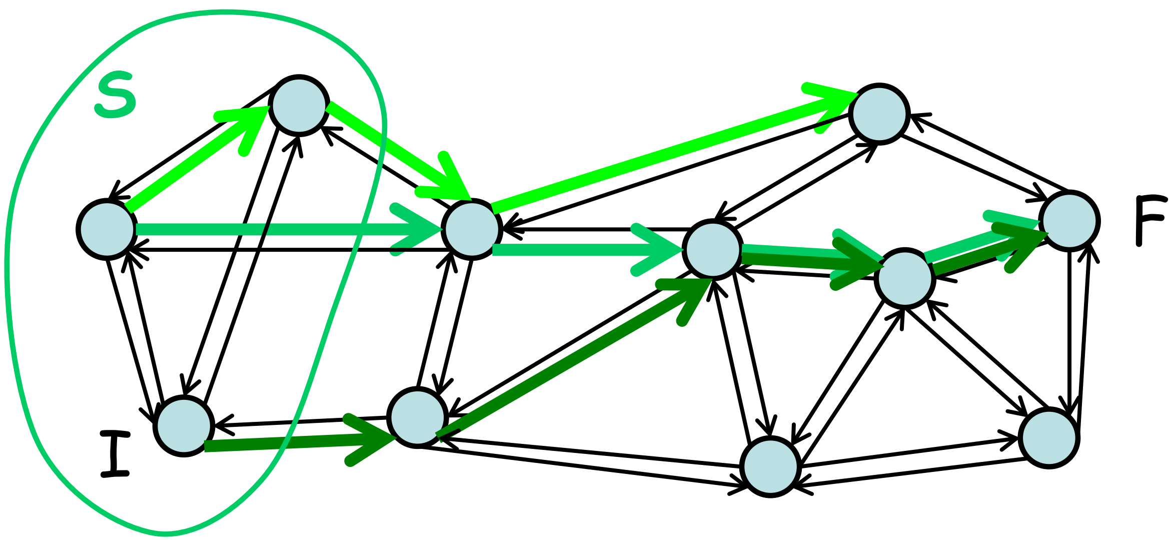
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F (|Ω|x|Ω| paths)

- Then, take any S:

$$\pi(S)\pi(\bar{S}) \quad = \sum_{I \in S,\, F \notin S} \pi(I)\pi(F)$$



[Jerrum-Sinclair]

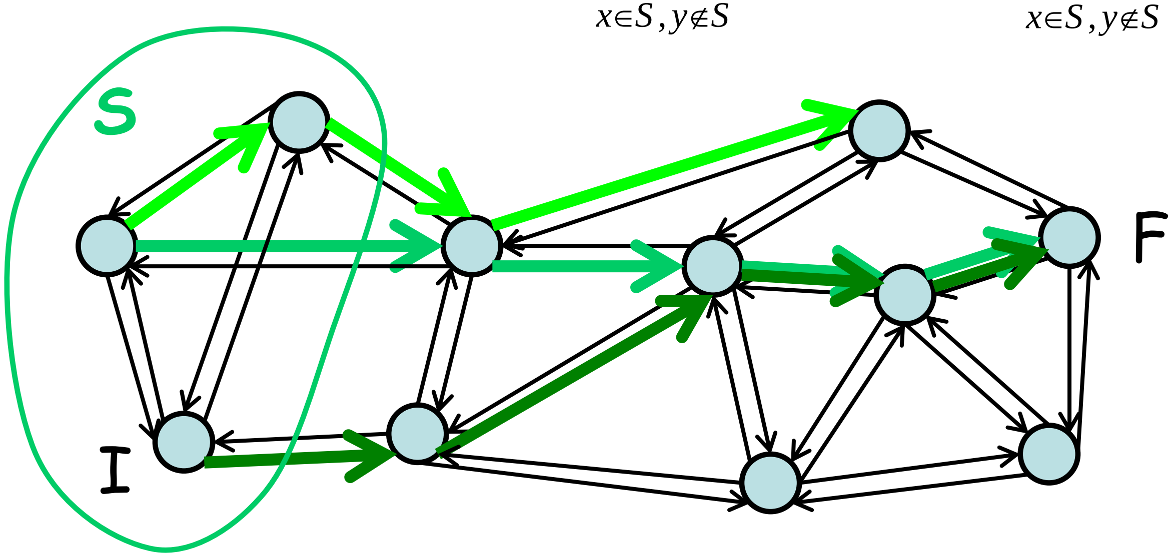
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F ($|\Omega| \times |\Omega|$ paths)

- Then, take any S:

$$\frac{\pi(S)\pi(\bar{S})}{\sum_{x\in S, y\notin S} \pi(x)P(x,y)} = \frac{\sum_{I\in S, F\notin S} \pi(I)\pi(F)}{\sum_{x\in S, y\notin S} \pi(x)P(x,y)}$$



[Jerrum-Sinclair]

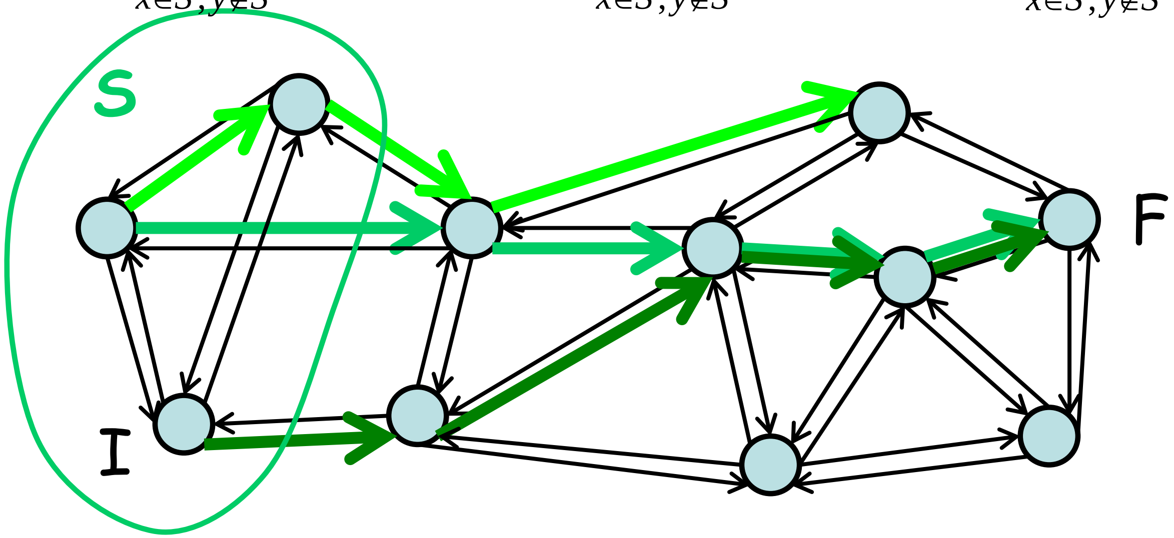
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F ($|\Omega| \times |\Omega|$ paths)

- Then, take any S:

$$\frac{\pi(S)/2}{\sum_{x\in S,\, y\notin S} \pi(x)P(x,y)} \le \frac{\pi(S)\pi(\bar{S})}{\sum_{x\in S,\, y\notin S} \pi(x)P(x,y)} = \frac{\sum_{I\in S,\, F\notin S} \pi(I)\pi(F)}{\sum_{x\in S,\, y\notin S} \pi(x)P(x,y)}$$



[Jerrum-Sinclair]

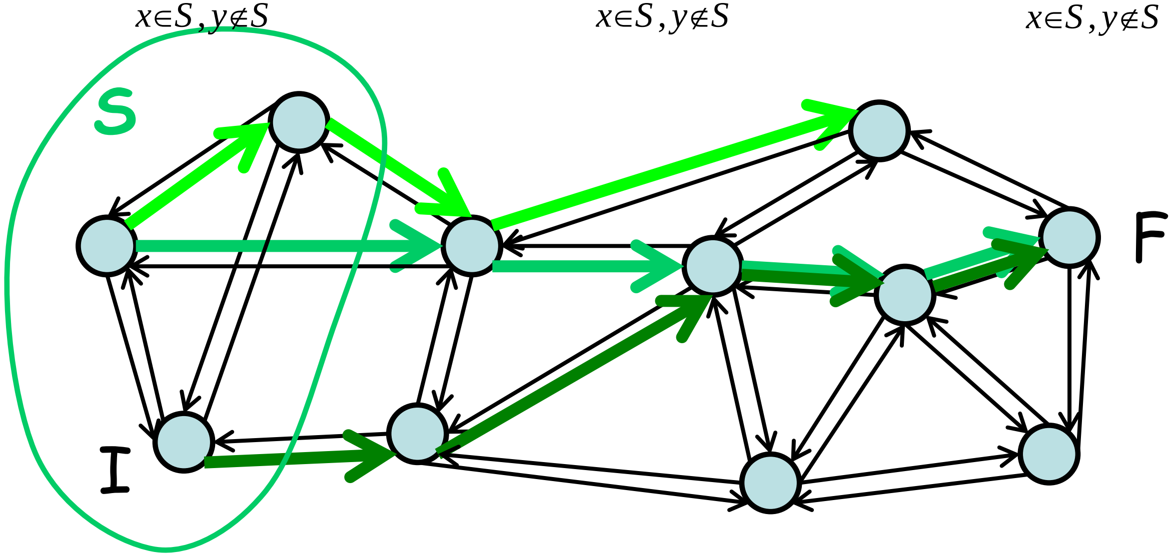
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F (|Ω|x|Ω| paths)

- Then, let S be the "smallest cut":

$$\frac{1}{2\Phi} = \frac{\pi(S)/2}{\sum_{x\in S, y\notin S}\pi(x)P(x,y)} \leq \frac{\pi(S)\pi(\bar{S})}{\sum_{x\in S, y\notin S}\pi(x)P(x,y)} = \frac{\sum_{I\in S, F\notin S}\pi(I)\pi(F)}{\sum_{x\in S, y\notin S}\pi(x)P(x,y)}$$



[Jerrum-Sinclair]

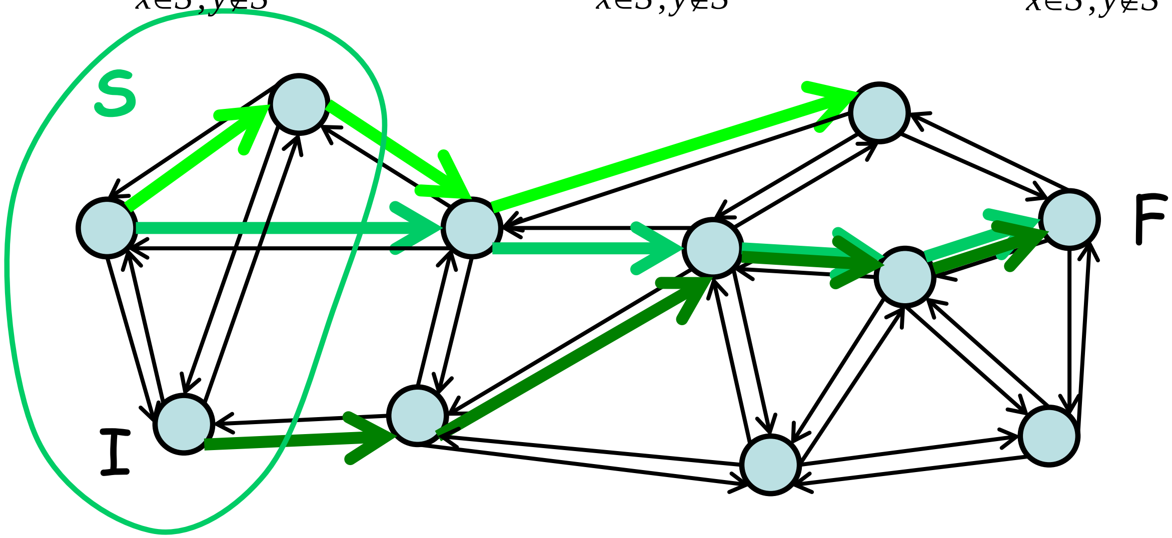
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F (|Ω|x|Ω| paths)

- Then, let S be the "smallest cut":

$$\frac{1}{2\Phi} = \frac{\pi(S)/2}{\sum\limits_{x\in S, y\notin S} \pi(x)P(x,y)} \leq \frac{\pi(S)\pi(\bar{S})}{\sum\limits_{x\in S, y\notin S} \pi(x)P(x,y)} = \frac{\sum\limits_{I\in S, F\notin S} \pi(I)\pi(F)}{\sum\limits_{x\in S, y\notin S} \pi(x)P(x,y)} \leq$$



$$\frac{\sum\limits_{x\in S, y\notin S} \; \sum\limits_{\substack{(I,F): I\in S, F\notin S,\\ (x,y)\text{ on } I\to F \text{ path}}} \pi(I)\pi(F)}{\sum\limits_{x\in S, y\notin S} \pi(x)P(x,y)}$$

[Jerrum-Sinclair]

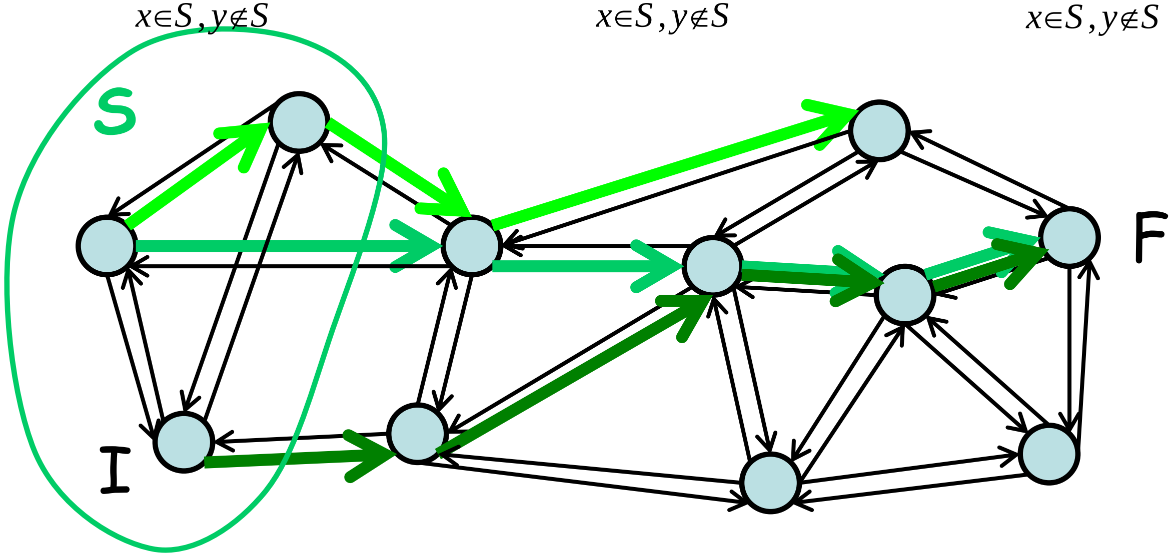
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F (|Ω|x|Ω| paths)

- Then, let S be the "smallest cut":

$$\frac{1}{2\Phi} = \frac{\pi(S)/2}{\sum_{x\in S, y\notin S} \pi(x)P(x,y)} \leq \frac{\pi(S)\pi(\bar{S})}{\sum_{x\in S, y\notin S} \pi(x)P(x,y)} = \frac{\sum_{I\in S,\, F\notin S} \pi(I)\pi(F)}{\sum_{x\in S, y\notin S} \pi(x)P(x,y)} \leq$$



$$\frac{\sum_{\substack{(I,F): I\in S,\, F\notin S, \\ (u,v) \text{ on } I\to F \text{ path}}} \pi(I)\pi(F)}{\pi(u)P(u,v)}$$

for some u in S, v not in S.

[Jerrum-Sinclair]

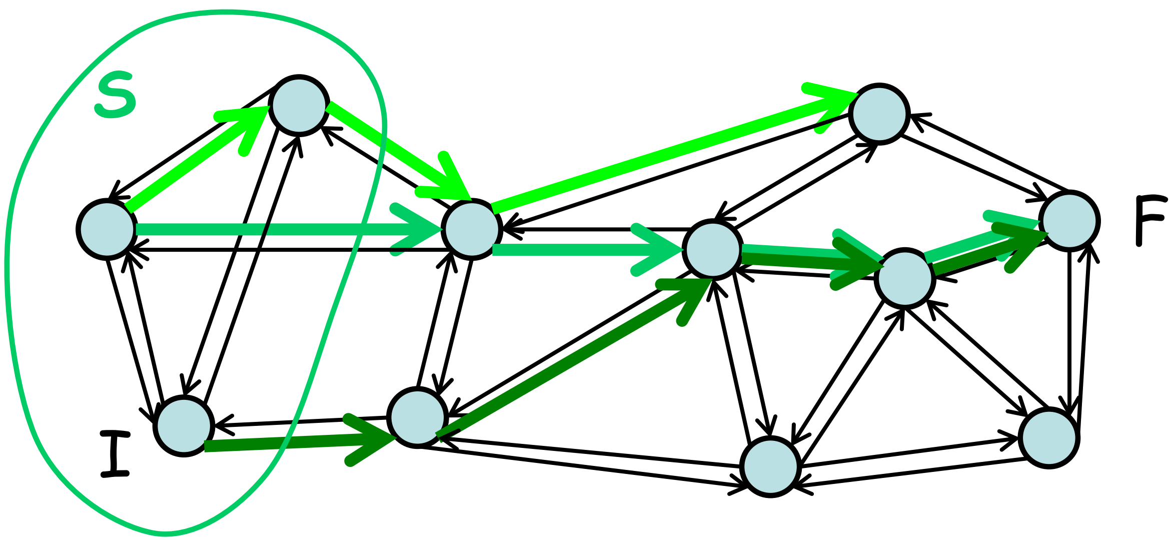
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F (|Ω|x|Ω| paths)

- Then, for some u in S, v not in S:

$$\frac{1}{2\Phi} \leq \frac{\displaystyle\sum_{\substack{(I,F):\, I\in S,\, F\notin S, \\ (u,v) \text{ on } I\to F \text{ path}}} \pi(I)\pi(F)}{\pi(u)P(u,v)}$$



Canonical Paths

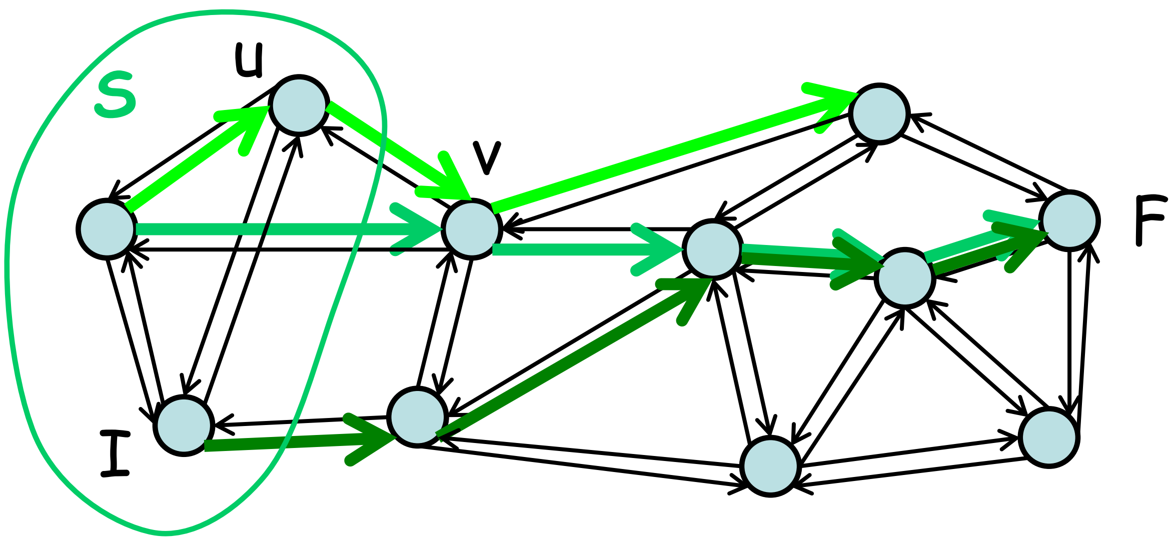
Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F ($|\Omega| \times |\Omega|$ paths)

- Then, for some u in S, v not in S:

$$\frac{1}{2\Phi} \leq \boxed{\frac{\displaystyle\sum_{\substack{(I,F):\, I \in S,\, F \notin S, \\ (u,v) \text{ on } I \to F \text{ path}}} \pi(I)\pi(F)}{\pi(u)P(u,v)}}$$

Congestion through transition (u,v)



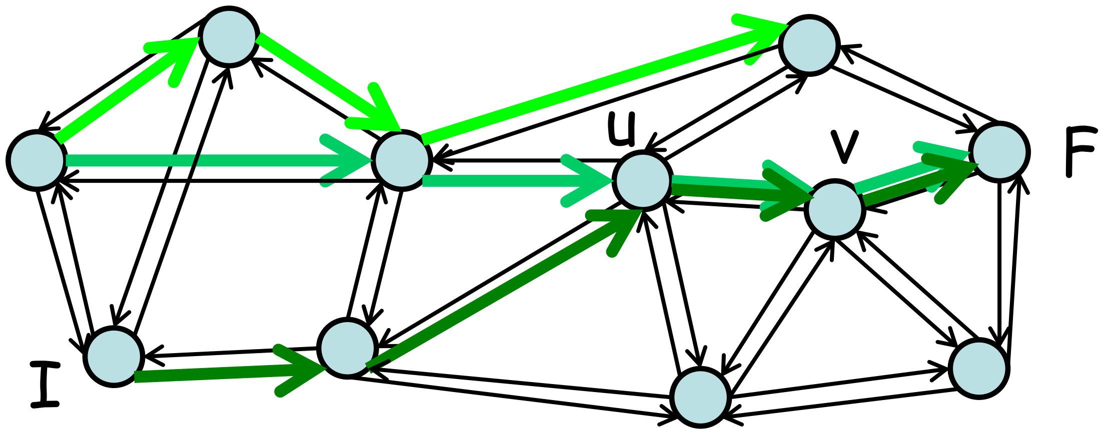
Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F (|Ω|x|Ω| paths)

Def: Congestion

$$\rho := \max_{u,v} \frac{1}{\pi(u)P(u,v)} \sum_{\substack{I \to F \text{ path} \\ \text{through } (u,v)}} \pi(I)\pi(F)(\text{length of } I \to F \text{ path})$$



Canonical Paths

Bounding the conductance:

- Find a path in the transition graph from every state I to every other state F ($|\Omega| \times |\Omega|$ paths)

Def: Congestion

$$\rho := \max_{u,v} \frac{1}{\pi(u)P(u,v)} \sum_{\substack{I \to F \text{ path} \\ \text{through } (u,v)}} \pi(I)\pi(F)(\text{length of } I \to F \text{ path})$$

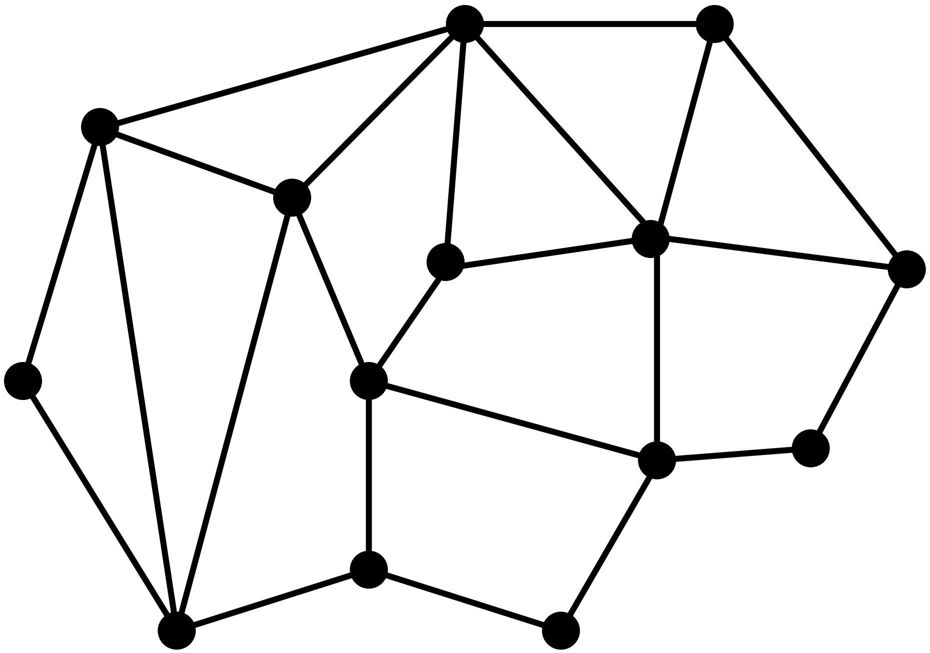
Thm [Sinclair]: For a lazy ergodic reversible MC:

$$t_{mix}(\varepsilon) \le 4\rho \ln\left(\frac{1}{\varepsilon\, \pi_{\min}}\right)$$

Matchings Revisited

Given an undirected graph $G=(V,E)$, a **matching** $M \subseteq E$ is a set of vertex disjoint edges. A matching is **perfect** if $|M|=n/2$, where n = # vertices (and m = # edges).

Example:

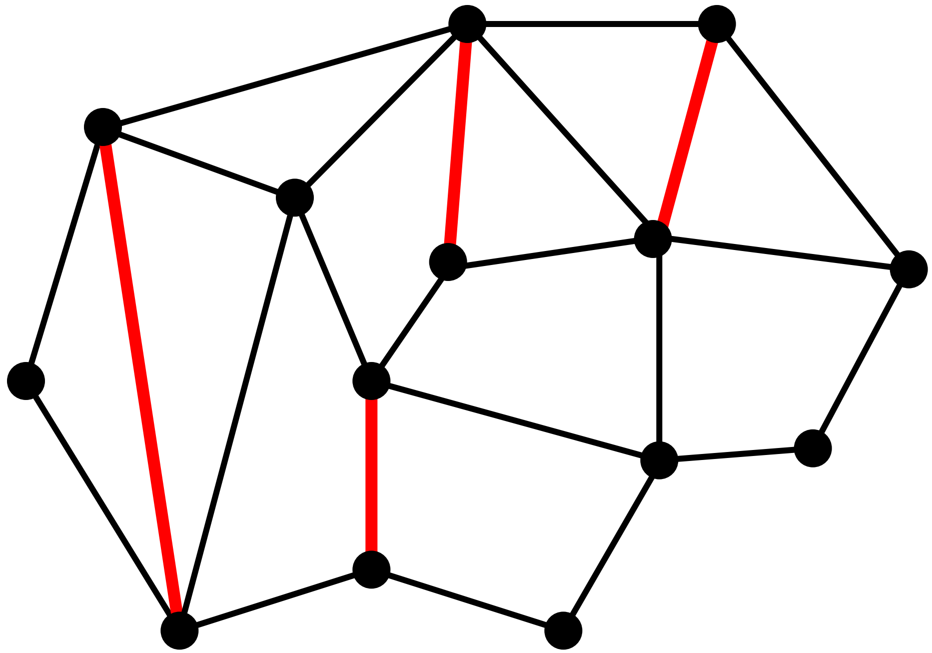


Matchings Revisited

Given an undirected graph $G=(V,E)$, a **matching** $M \subseteq E$ is a set of vertex disjoint edges. A matching is **perfect** if $|M|=n/2$, where n = # vertices (and m = # edges).

Example:

A matching

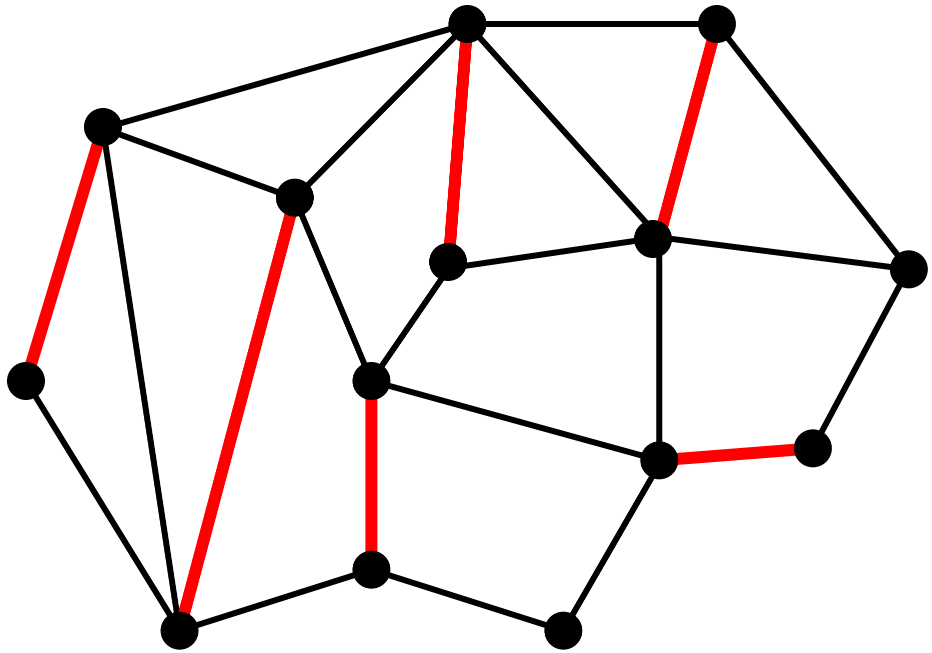


Matchings Revisited

Given an undirected graph $G=(V,E)$, a **matching** $M \subseteq E$ is a set of vertex disjoint edges. A matching is **perfect** if $|M|=n/2$, where n = # vertices (and m = # edges).

Example:

A perfect matching

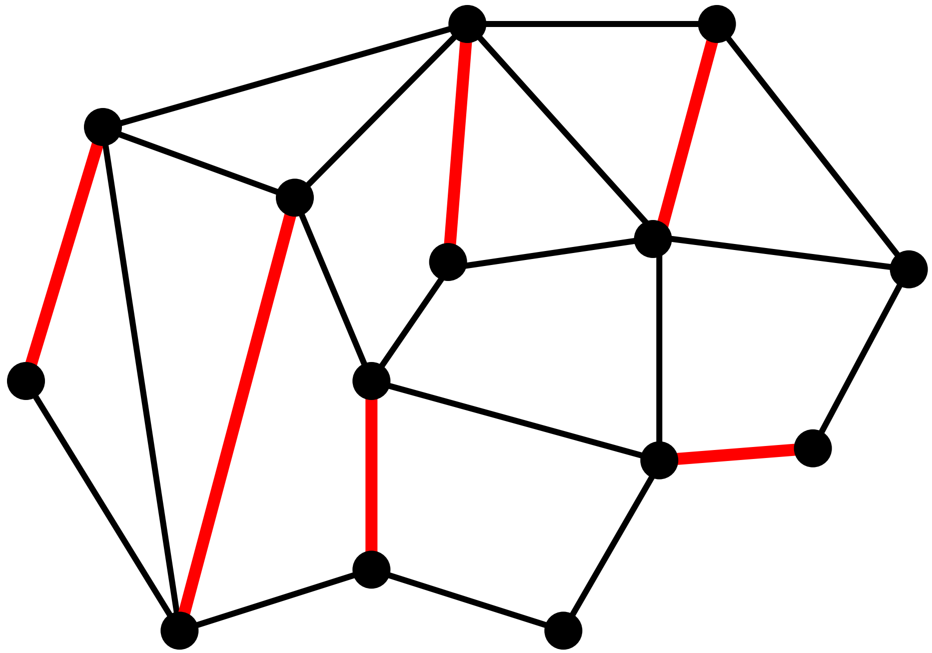


Matchings Revisited

Given an undirected graph $G=(V,E)$, a **matching** $M \subseteq E$ is a set of vertex disjoint edges. A matching is **perfect** if $|M|=n/2$, where n = # vertices (and m = # edges).

Example:

A perfect matching



Goal:

An FPRAS for

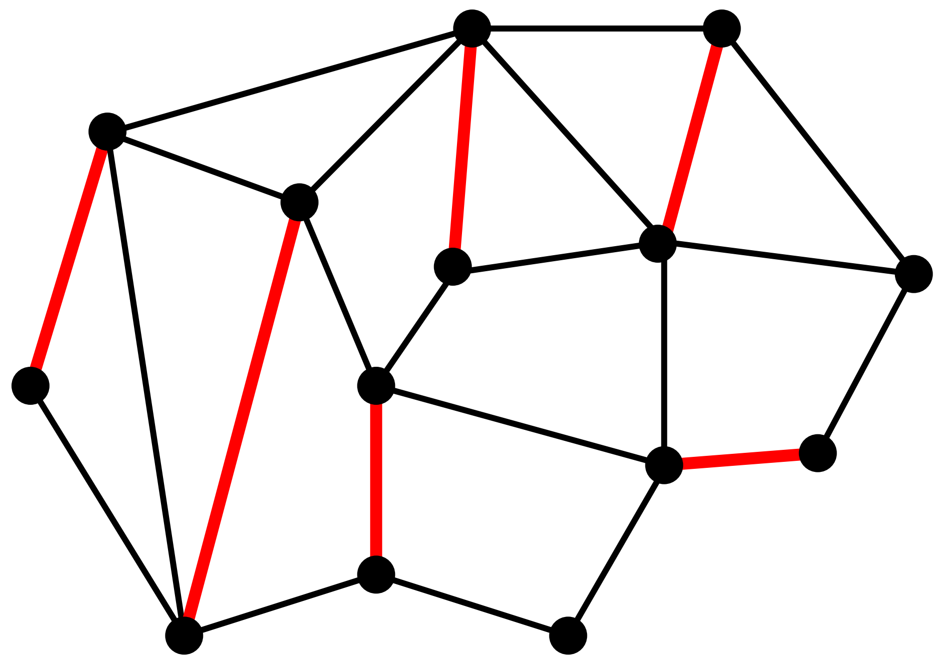
- # matchings
- # perfect matchings

Matchings Revisited

Given an undirected graph G=(V,E), a **matching** $M \subseteq E$ is a set of vertex disjoint edges. A matching is **perfect** if $|M|=n/2$, where n = # vertices (and m = # edges).

Example:

A perfect matching



Goal:

FPAUS (sampler)

An FPRAS for

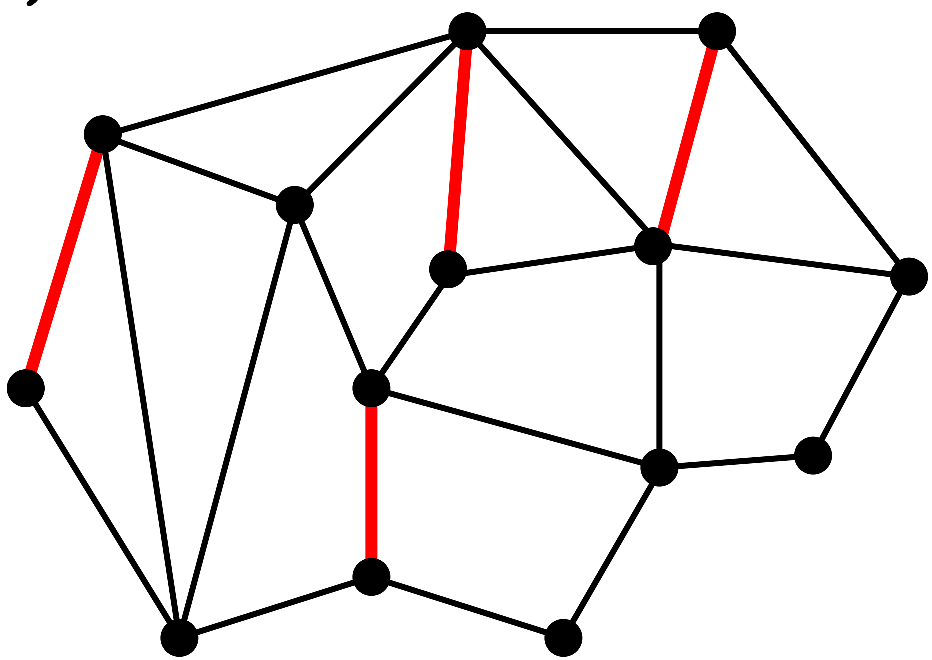
- # matchings
- # perfect matchings

A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

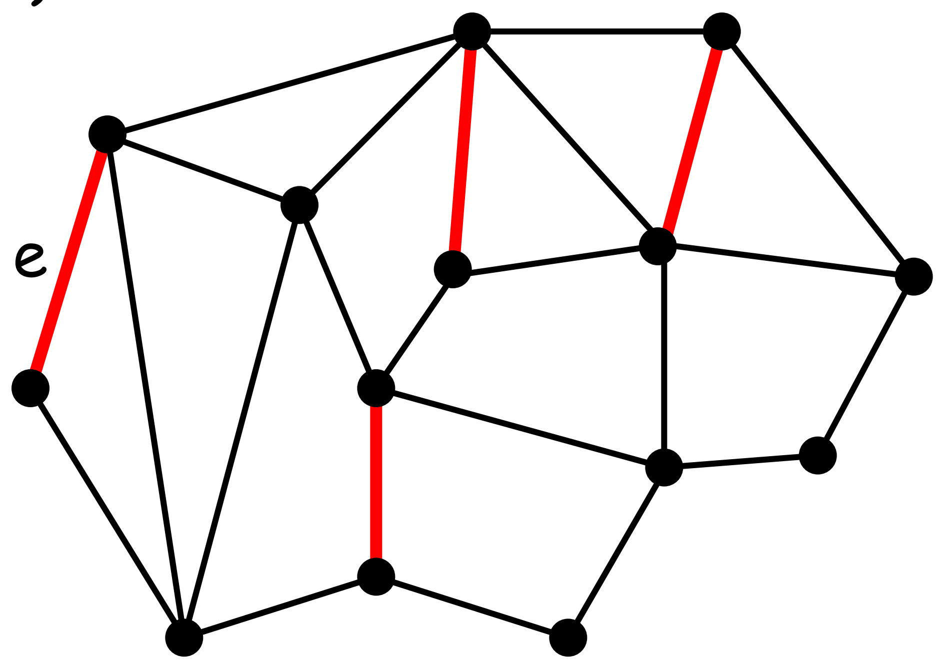
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

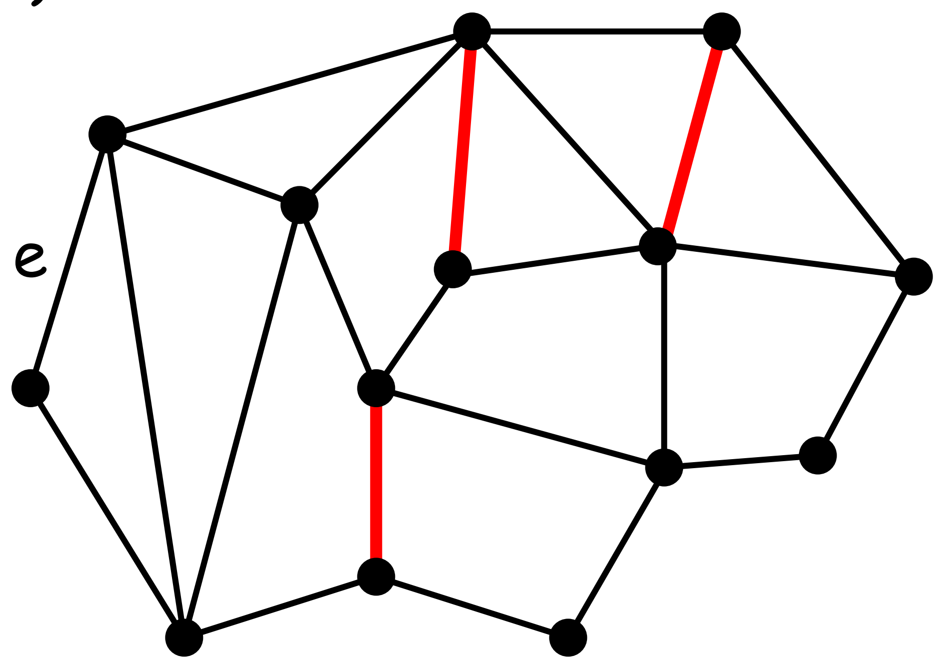


A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



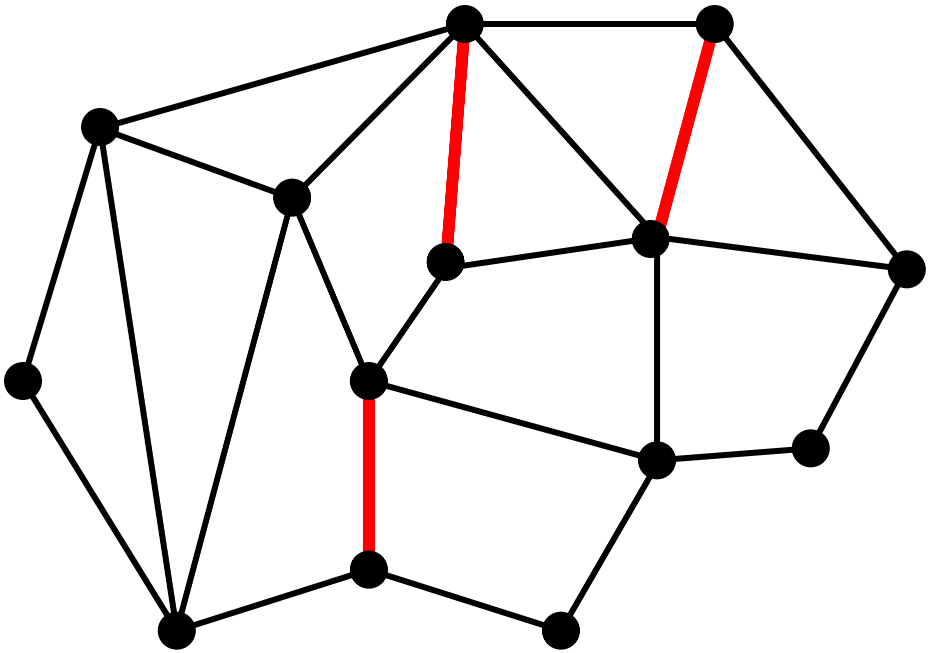
- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

A Markov Chain for Matchings

<u>Input</u>: a graph G

<u>State space Ω</u>: all matchings of G

<u>**Markov chain**</u> (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

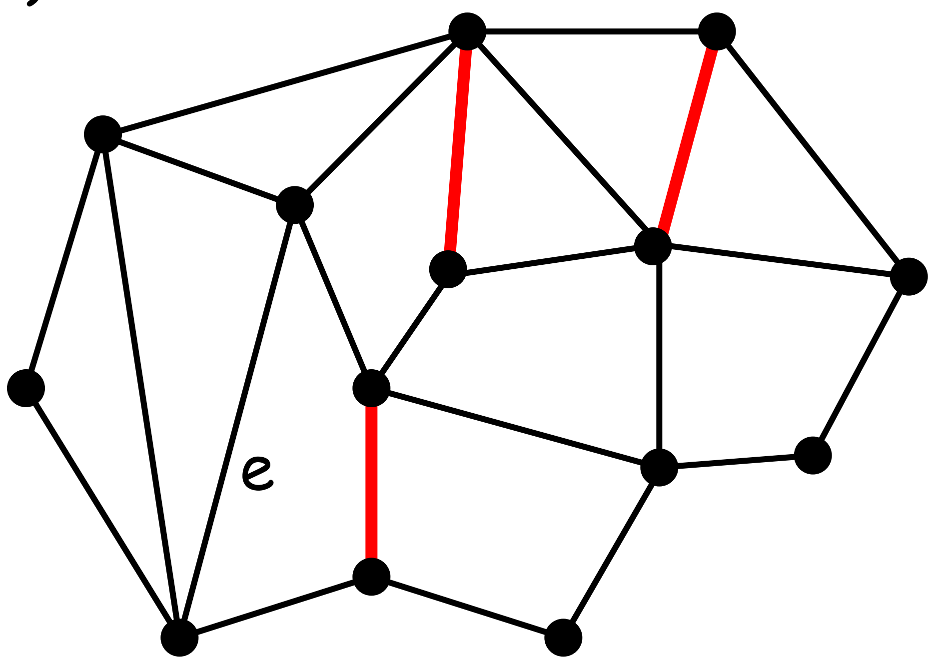
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



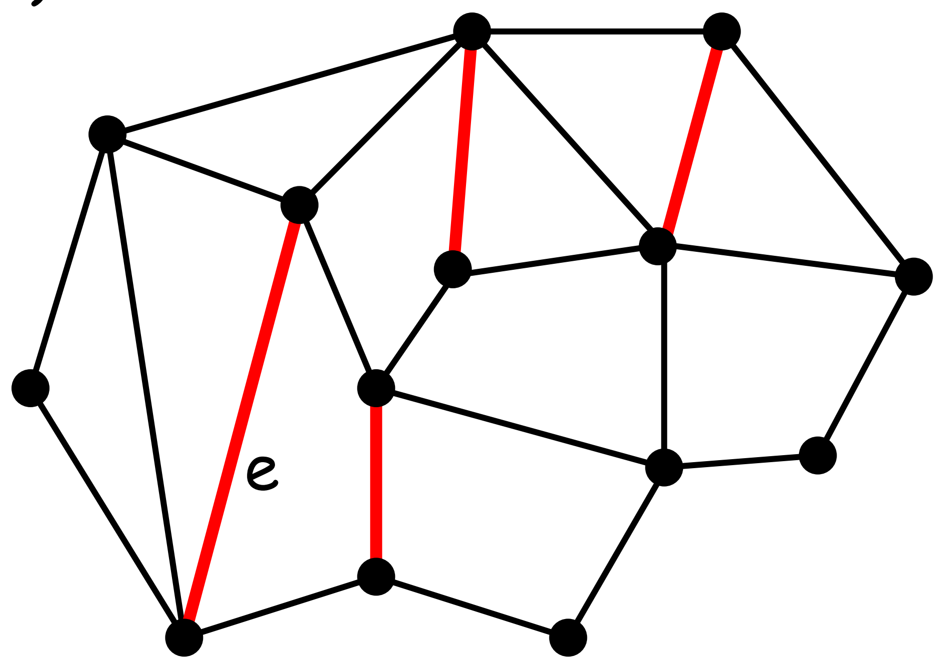
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

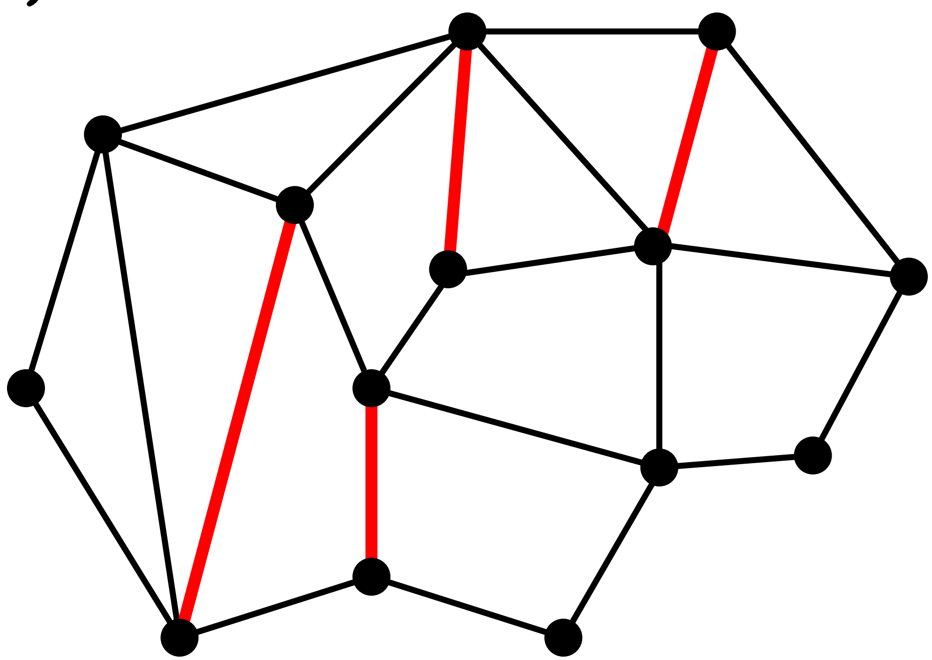


A Markov Chain for Matchings

<u>Input</u>: a graph G

<u>State space Ω</u>: all matchings of G

<u>Markov chain</u> (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

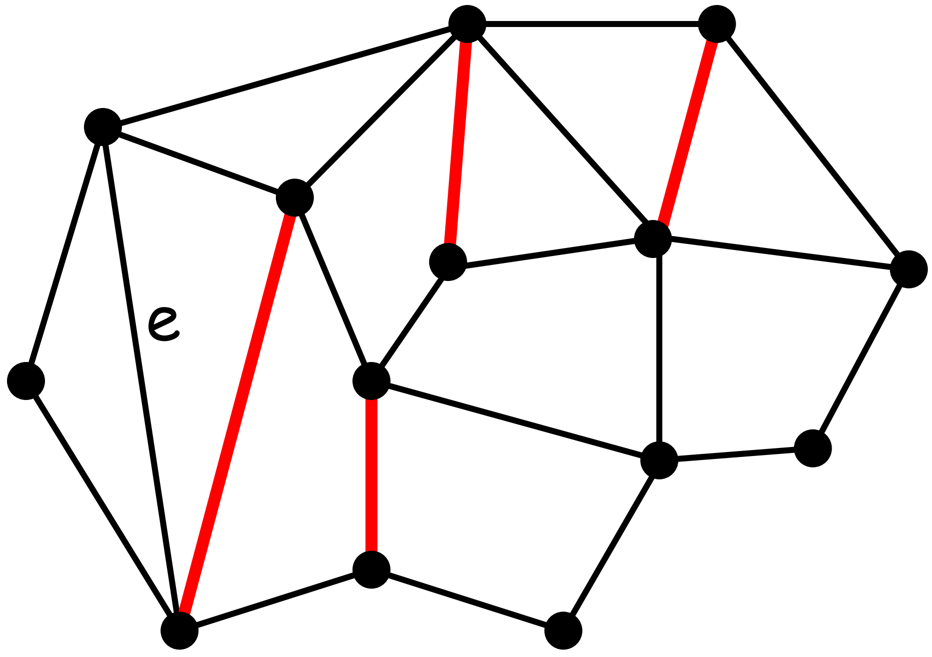
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



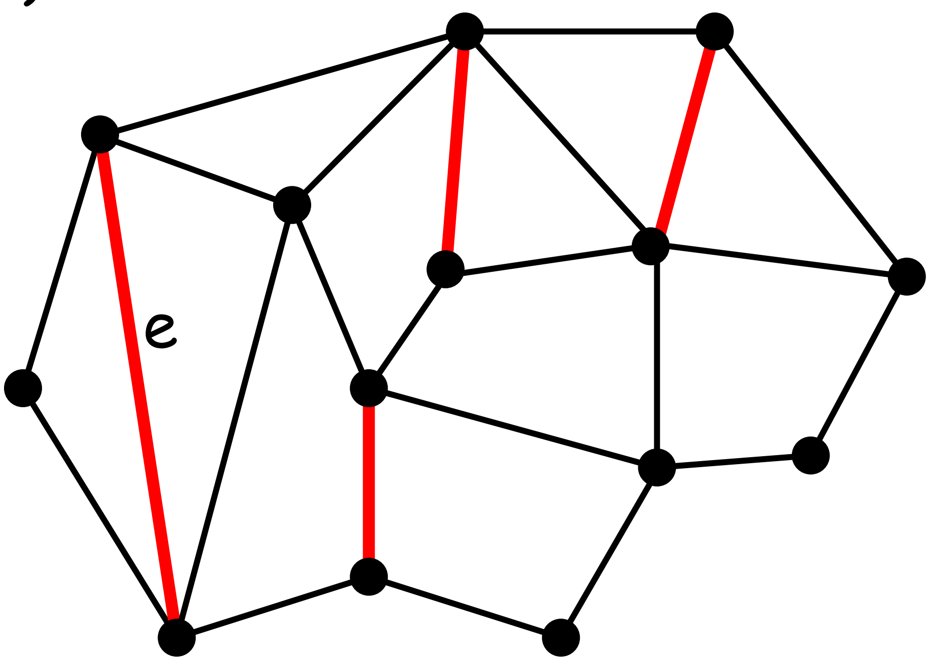
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

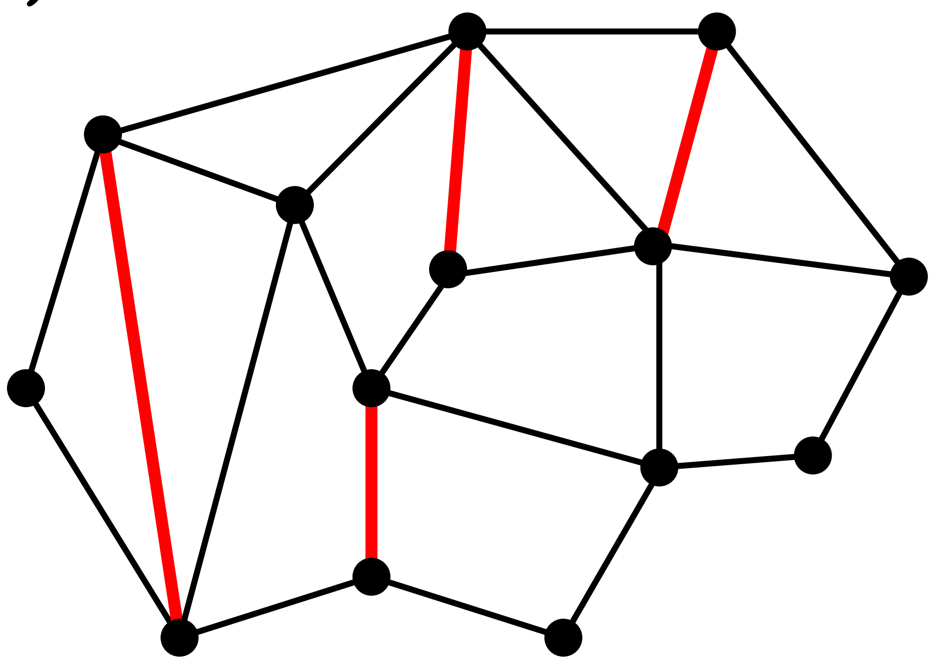


A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



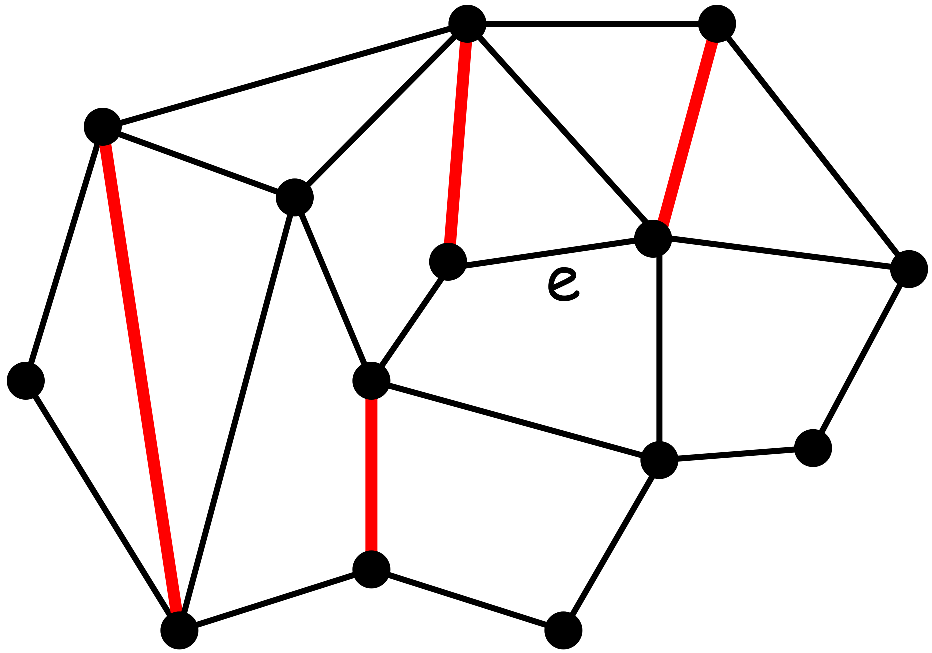
- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

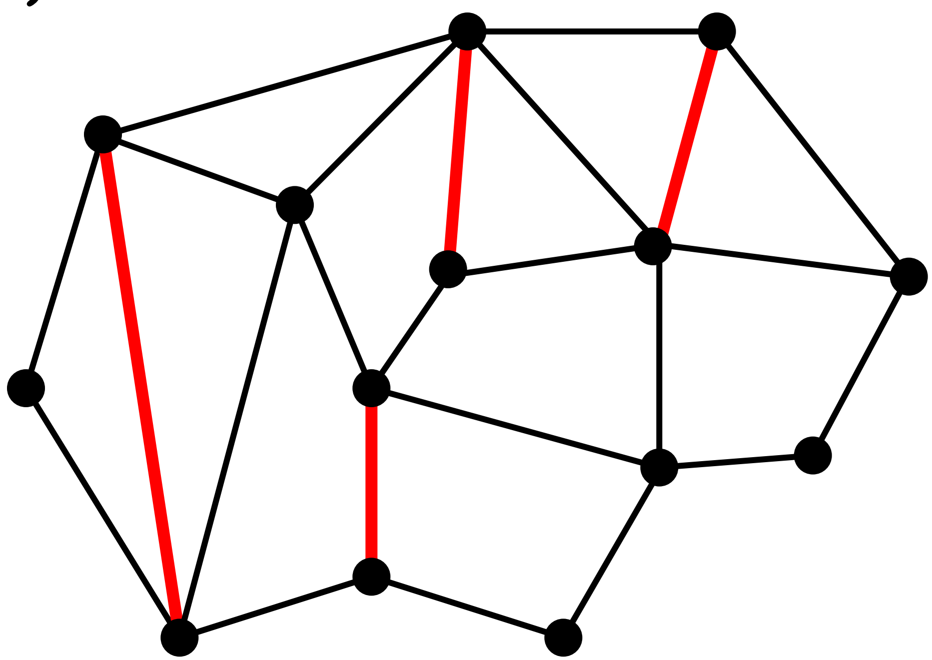
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



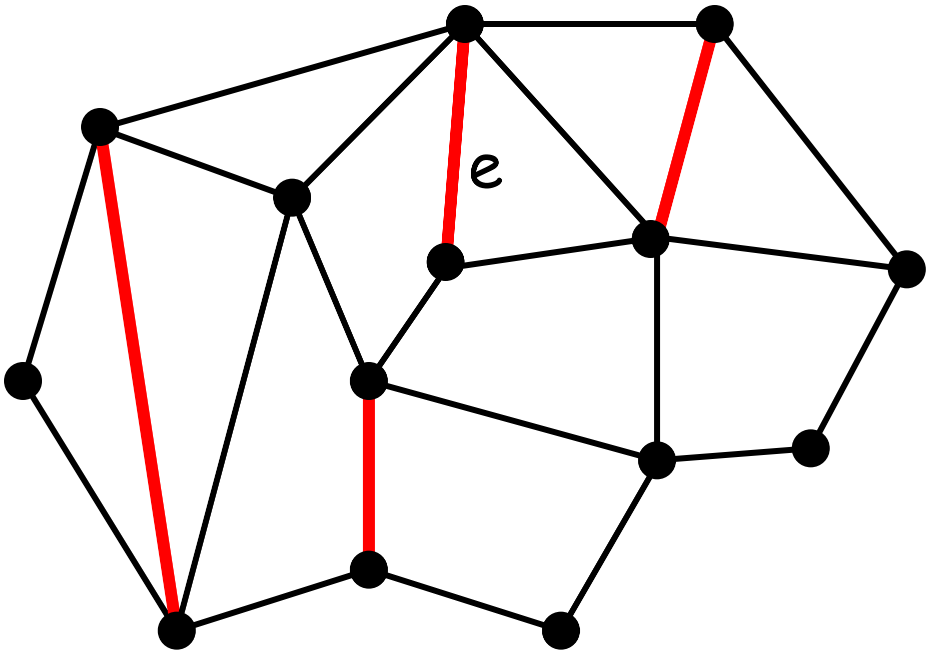
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



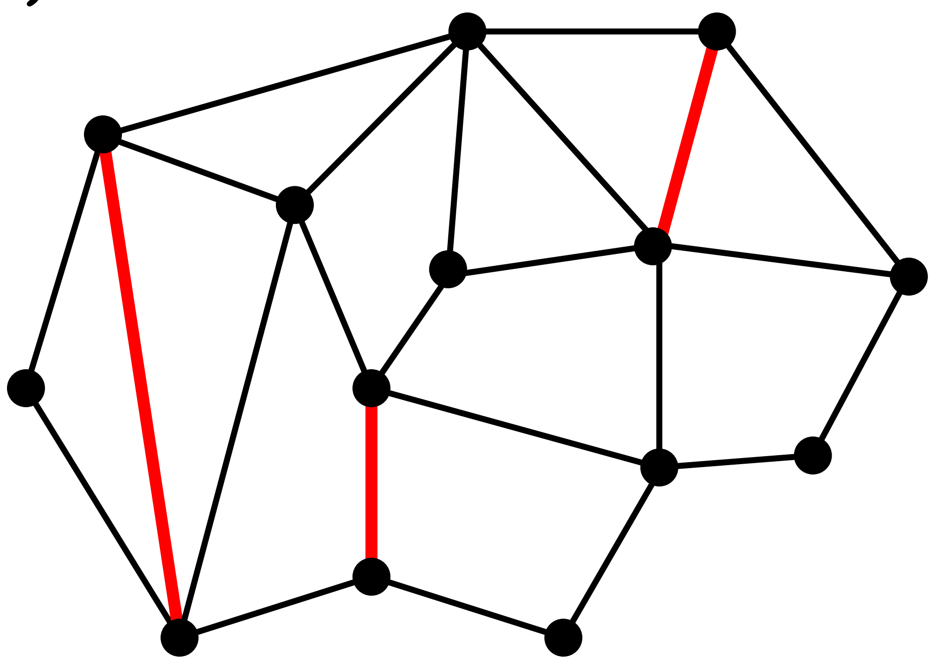
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



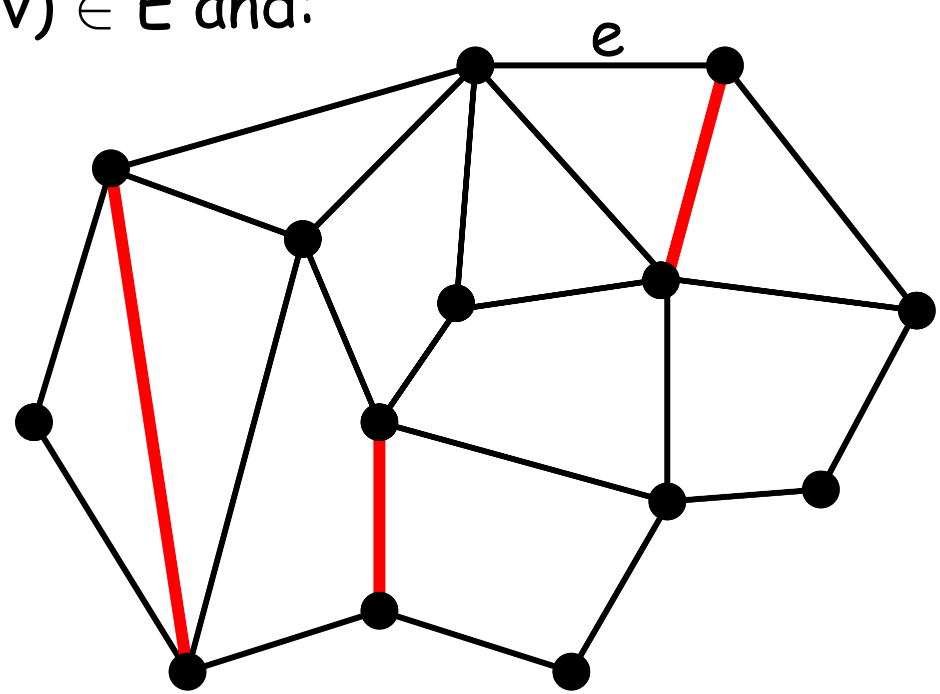
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



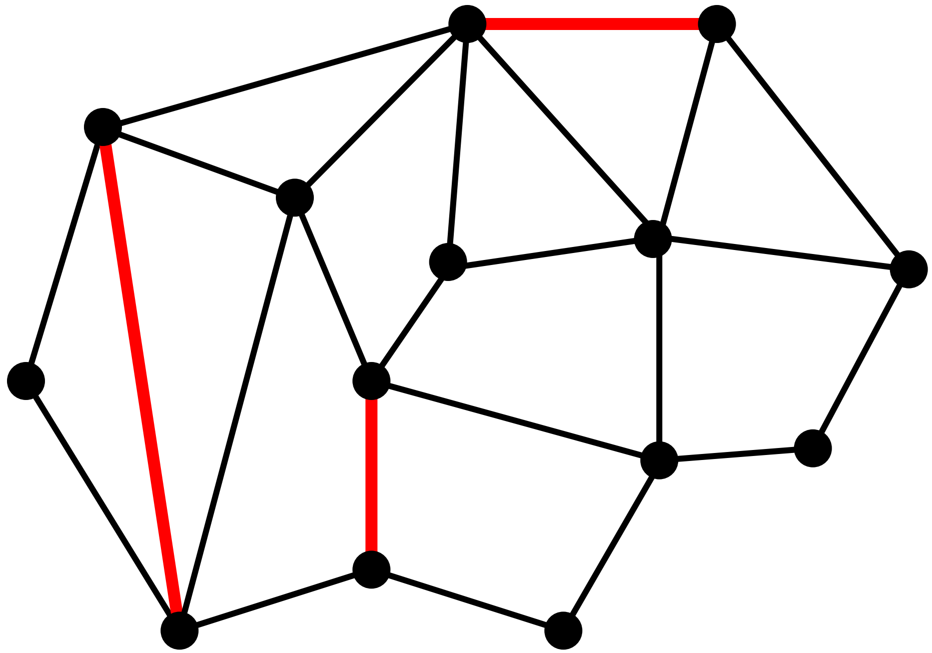
A Markov Chain for Matchings

<u>Input</u>: a graph G

<u>State space Ω</u>: all matchings of G

<u>**Markov chain**</u> (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



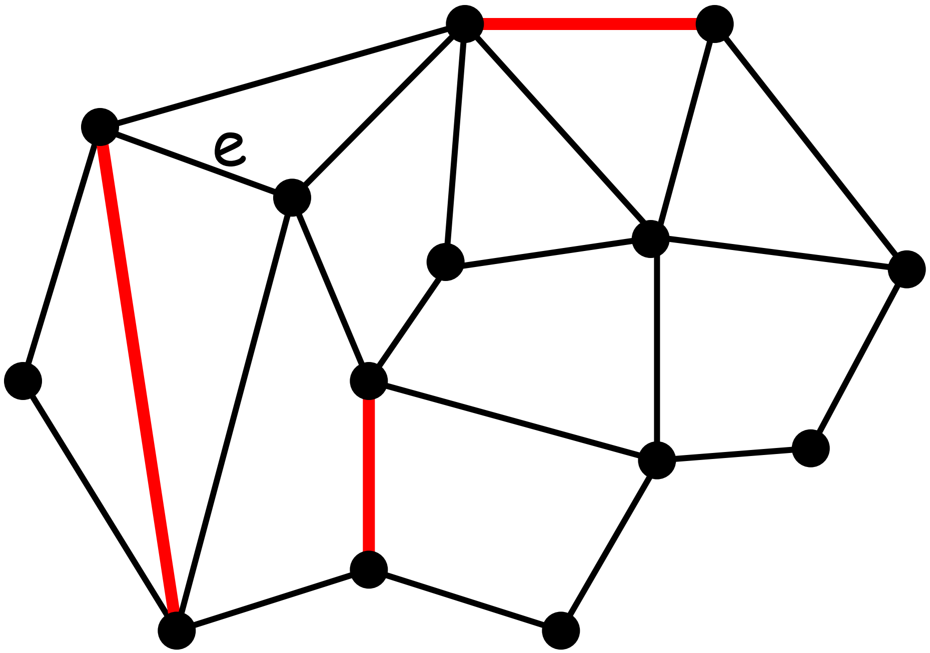
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

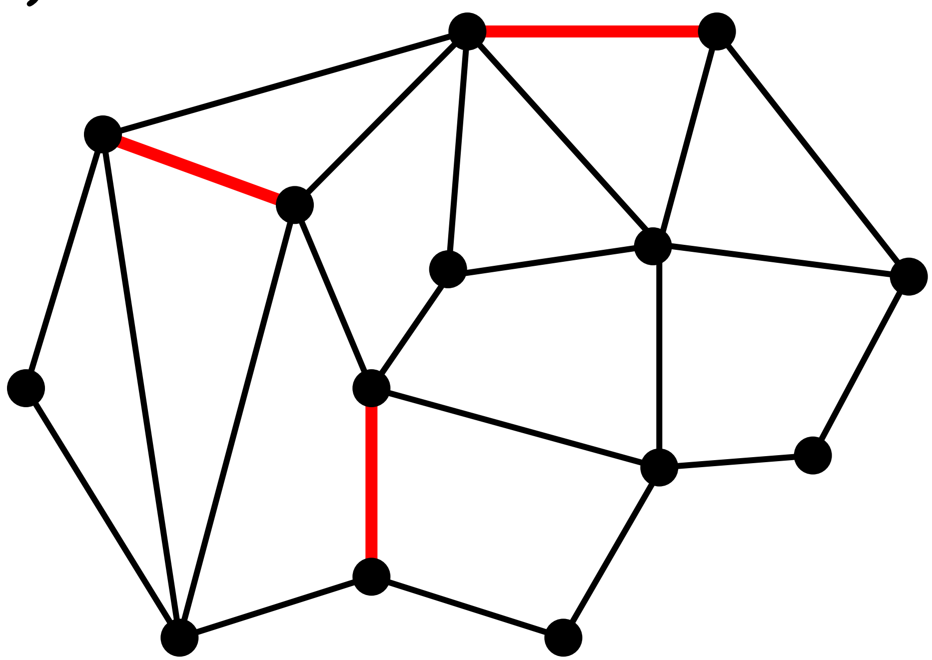


A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

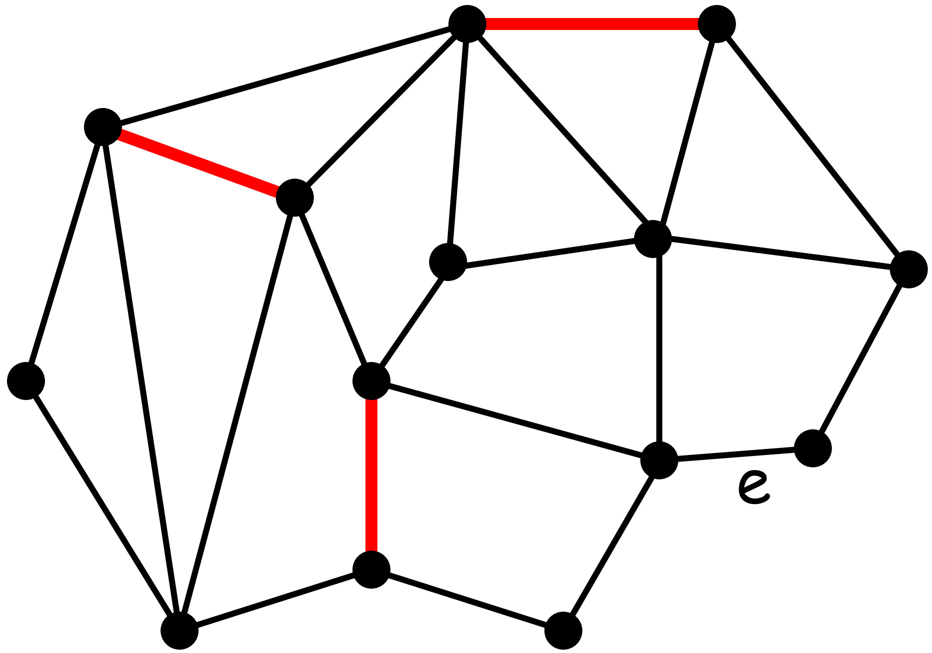
A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

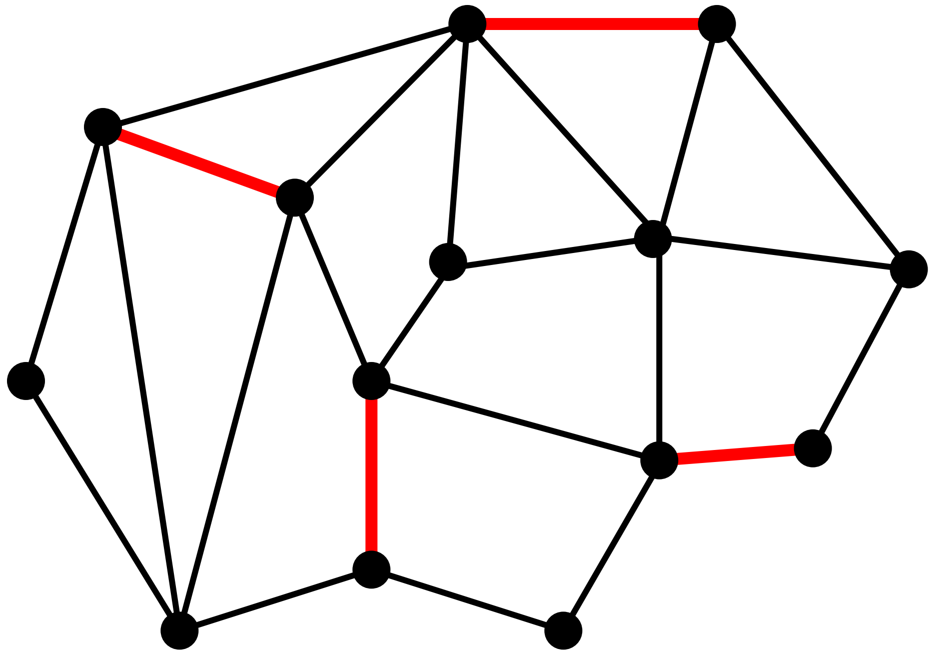


A Markov Chain for Matchings

Input: a graph G

State space Ω: all matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:



- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M

A Markov Chain for Matchings

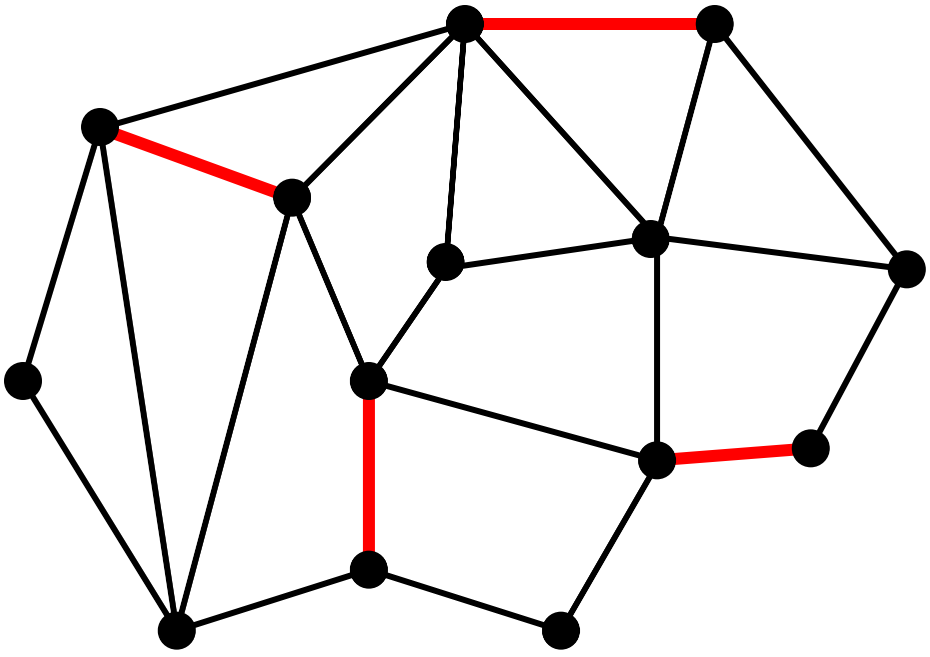
Technicality:
A **lazy** chain: with probability 1/2 stay in M, otherwise

Input: a graph G

State space Ω: all matchings of G

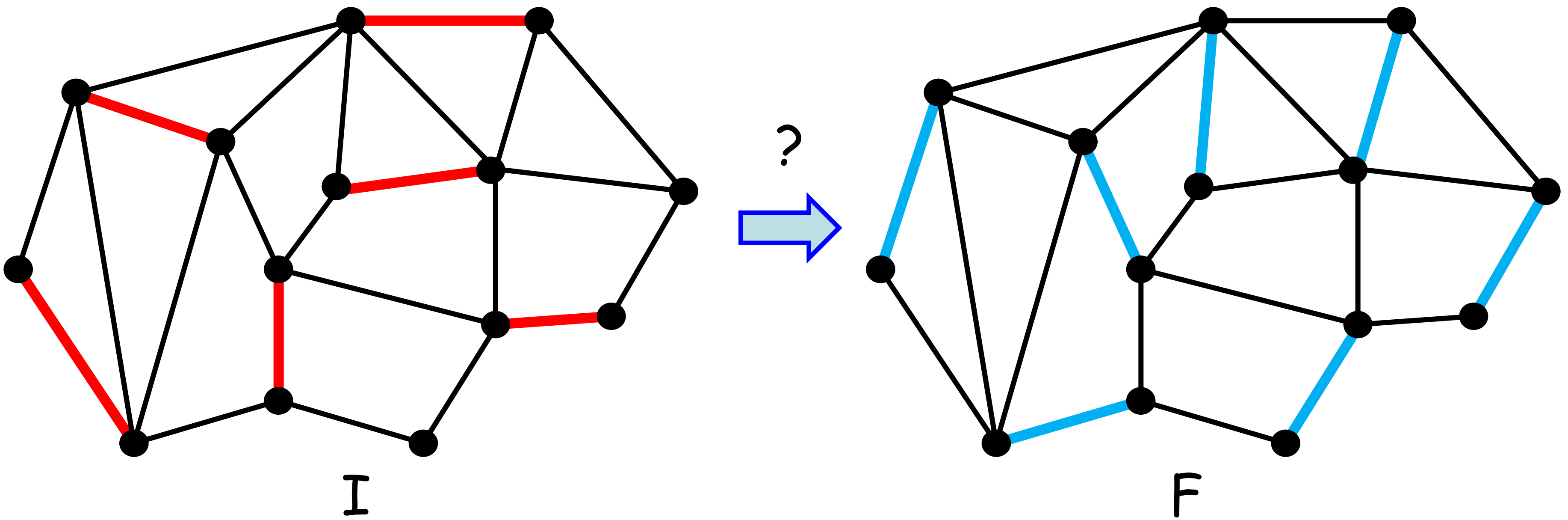
Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random edge $e=(u,v) \in E$ and:

- if $e \in M$, remove e from M
- if u,v are not covered by edges in M, add e to M
- if u is covered by edge $e' \in M$ and v is not covered by M, replace e' with e in M
- otherwise, stay in M



Canonical Paths

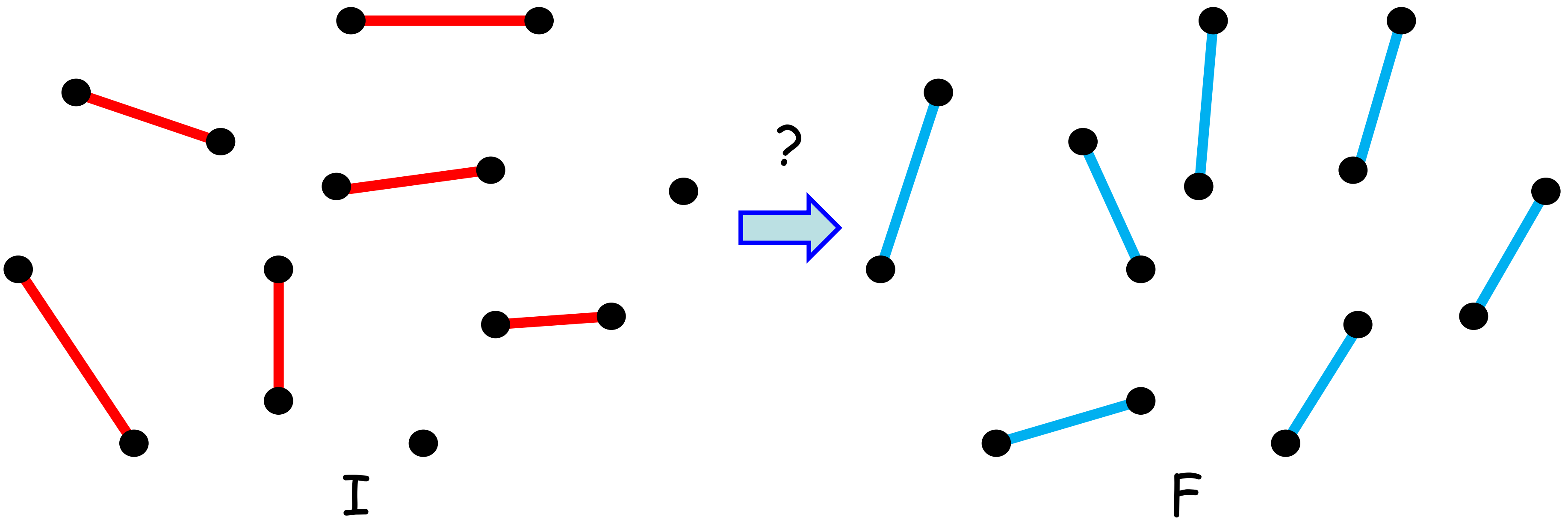
For any pair of matchings I,F, define a path from I to F in the transition graph:



[Jerrum-Sinclair]

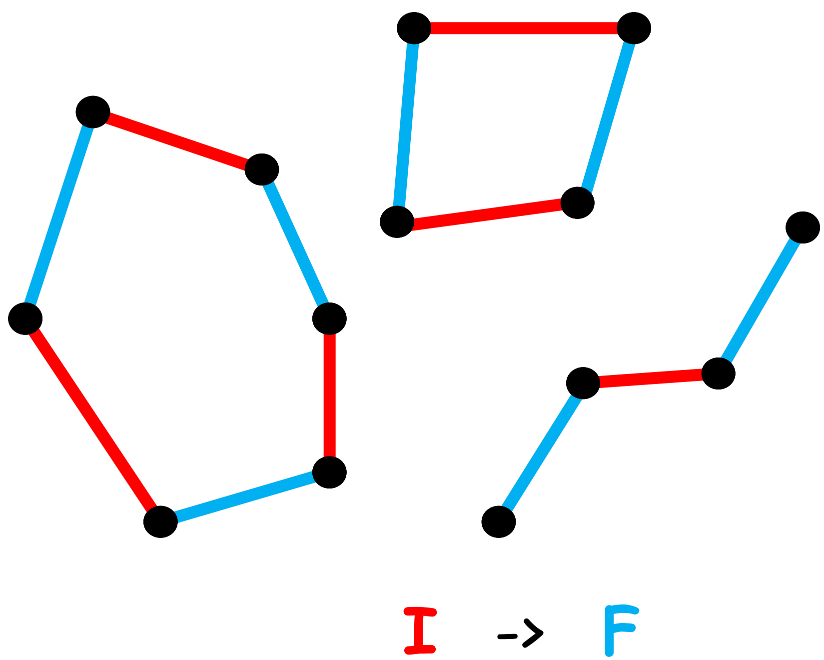
Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:

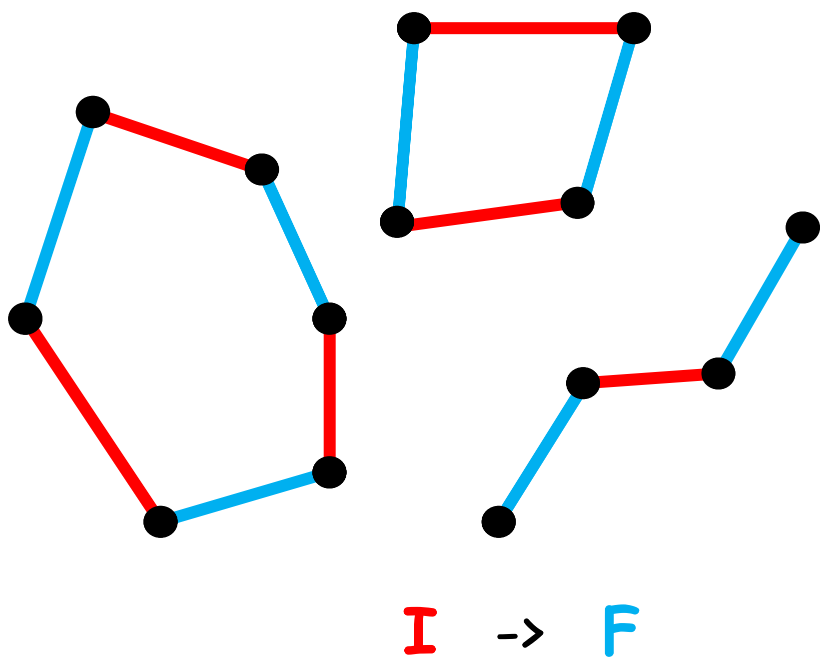


Going from red to blue:

- take I $\oplus$ F (sym. difference)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:

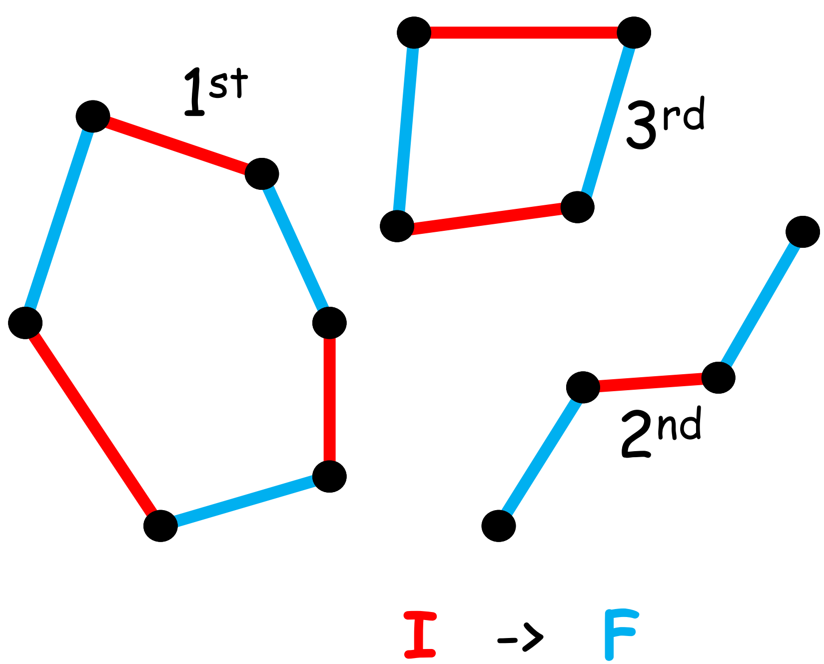


Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:

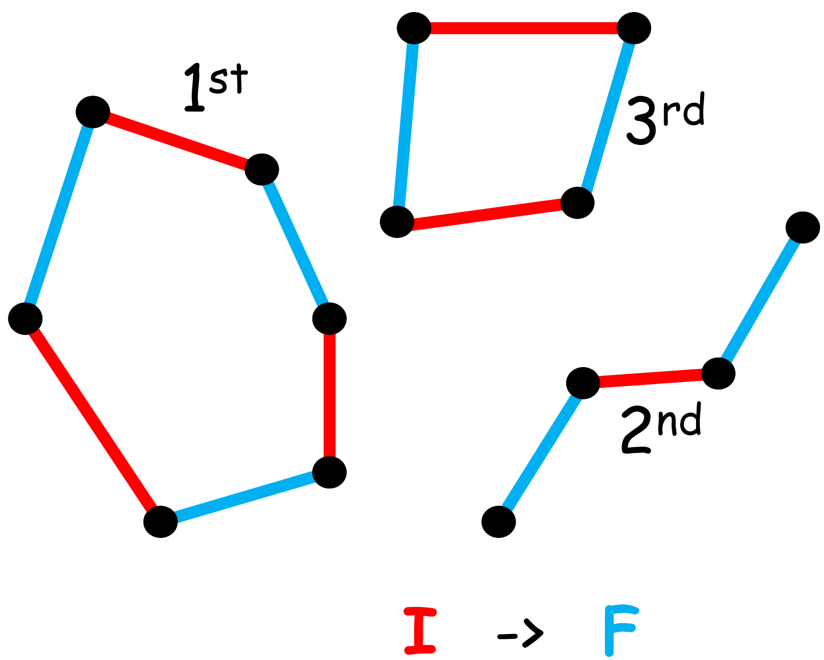


Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:

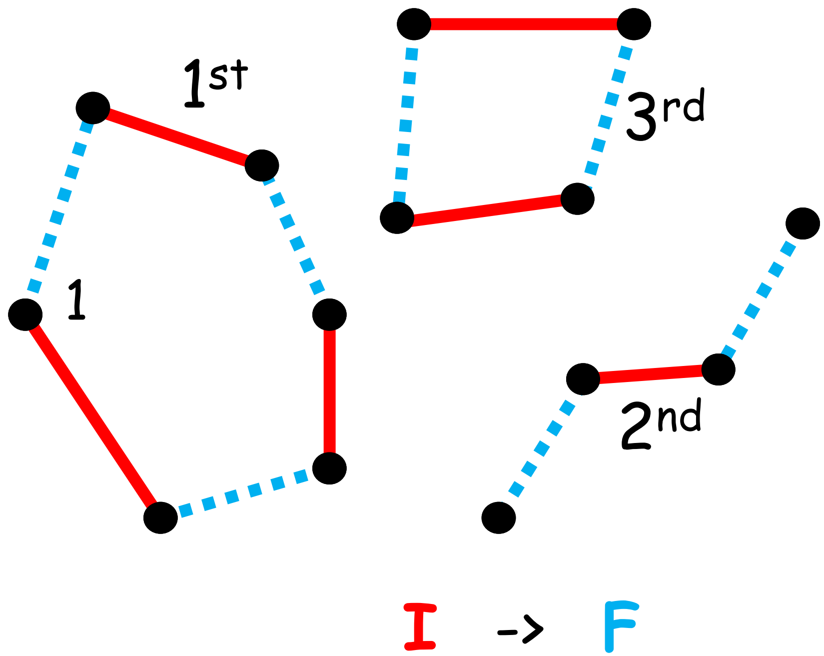


Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



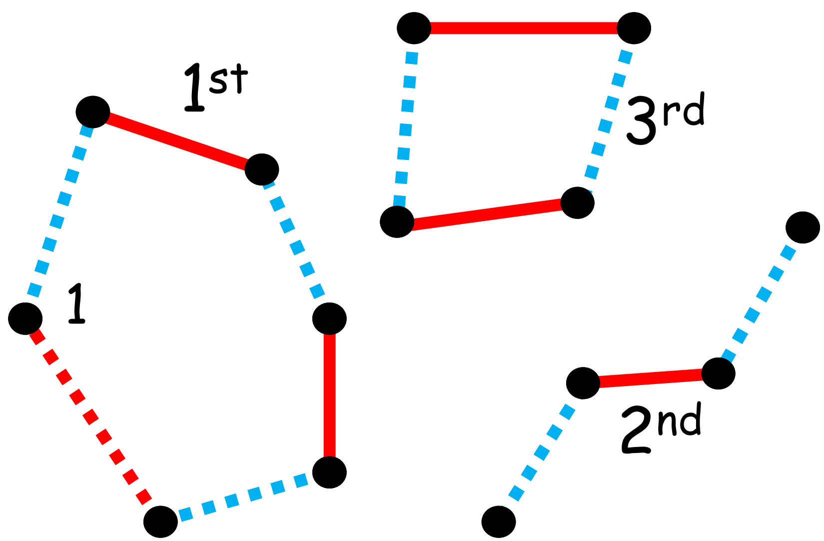
Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if cycle:
    - - remove lowest edge
    - - slide the rest
    - - add the last edge

(dashed edges:
not in the current matching)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



I -> F

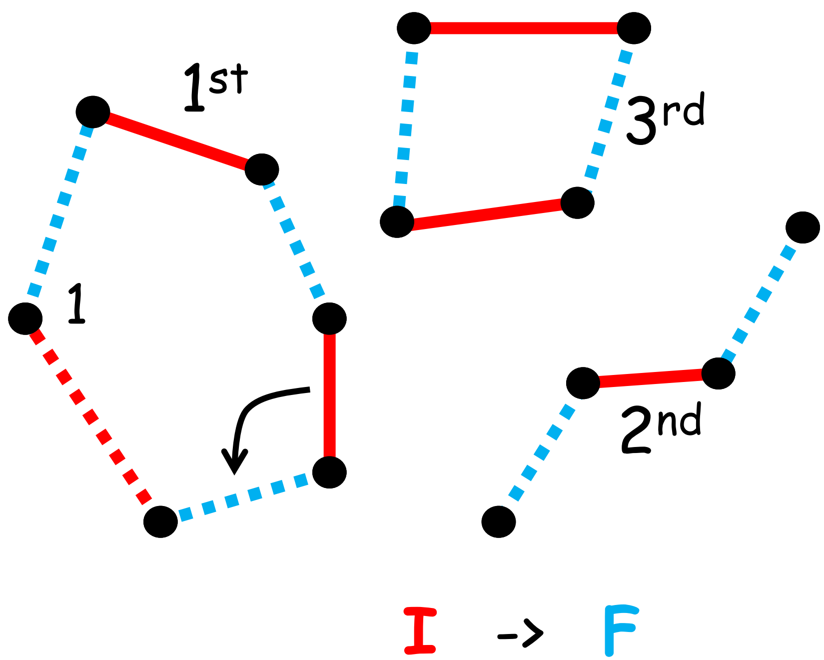
(dashed edges:
not in the current matching)

Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
    - -if cycle:
        - - remove lowest edge
        - - slide the rest
        - - add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



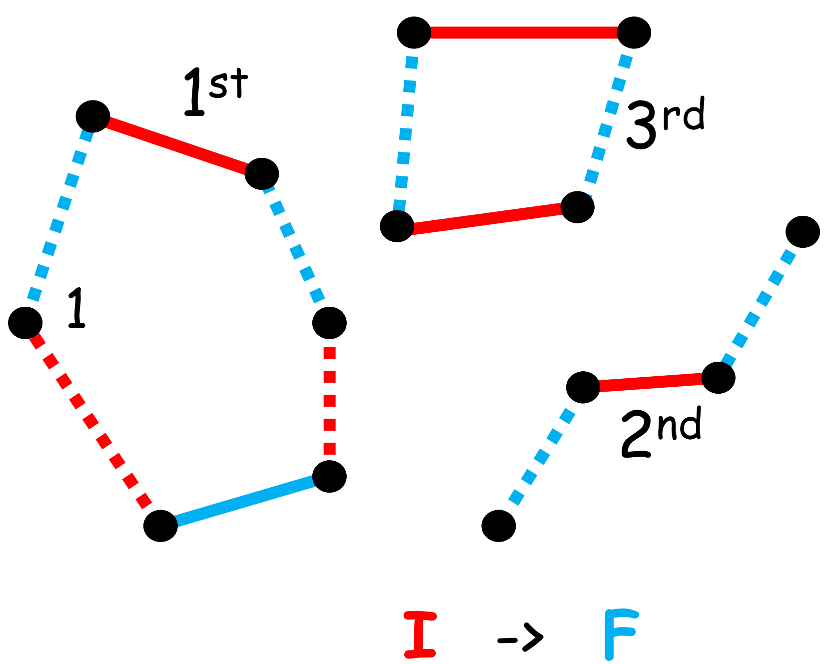
Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if cycle:
    - - remove lowest edge
    - - slide the rest
    - - add the last edge

(dashed edges:
not in the current matching)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



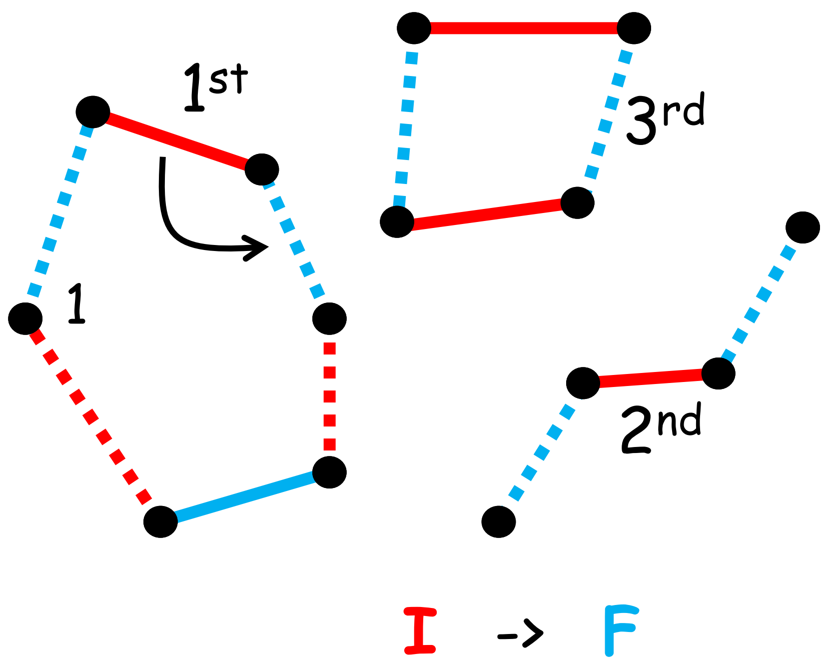
Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if cycle:
    - - remove lowest edge
    - - slide the rest
    - - add the last edge

(dashed edges:
not in the current matching)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



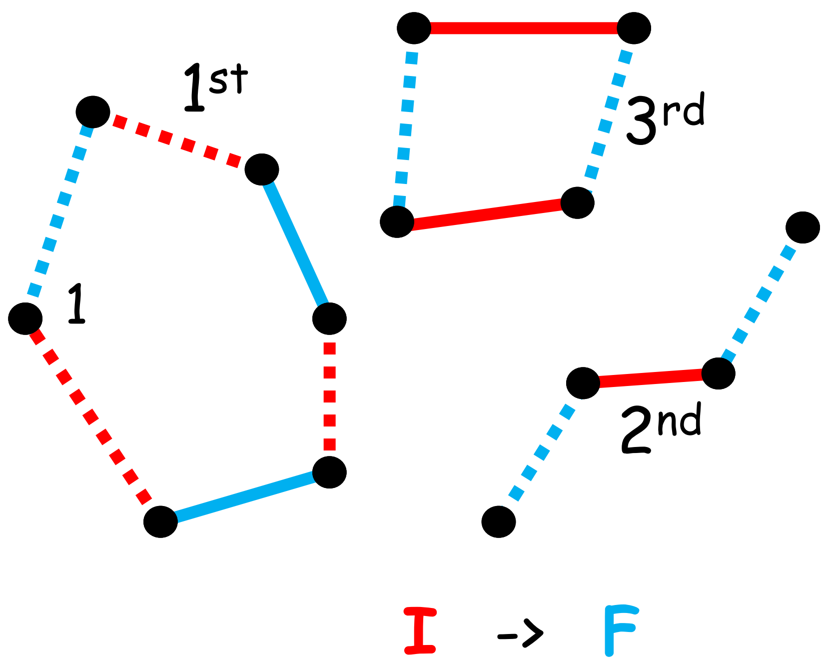
Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if cycle:
    - - remove lowest edge
    - - slide the rest
    - - add the last edge

(dashed edges:
not in the current matching)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



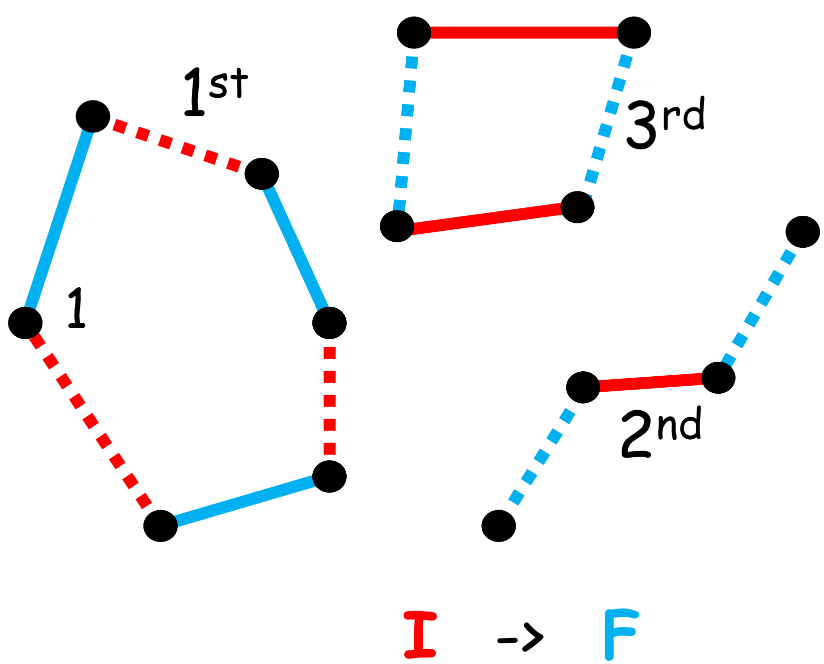
Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if cycle:
    - - remove lowest edge
    - - slide the rest
    - - add the last edge

(dashed edges:
not in the current matching)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



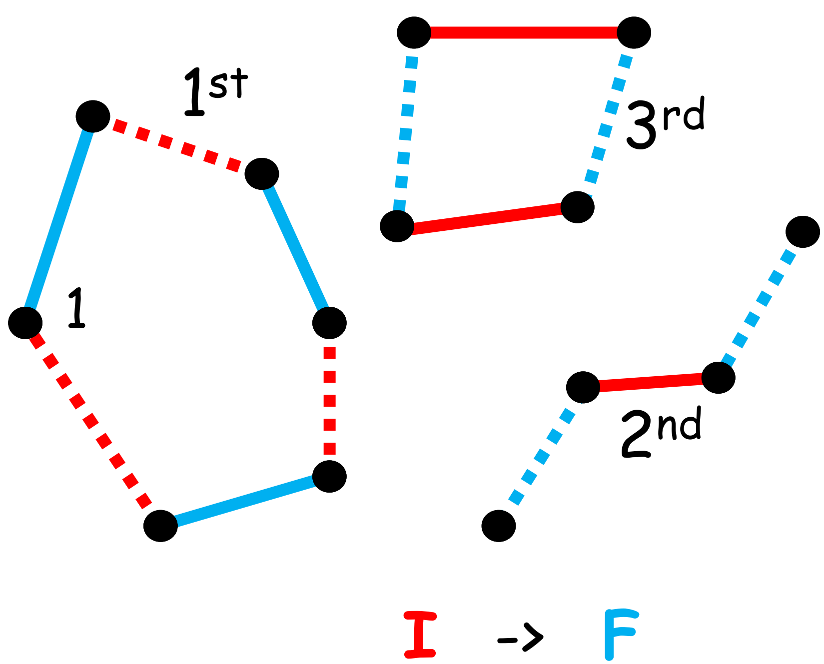
(dashed edges:
not in the current matching)

Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if cycle:
    - - remove lowest edge
    - - slide the rest
    - - add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



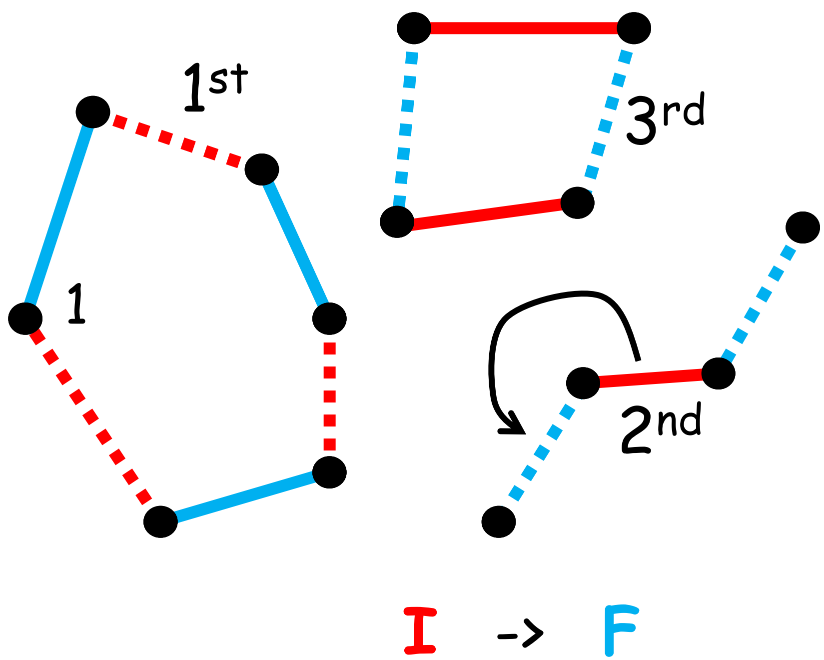
(dashed edges: not in the current matching)

Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



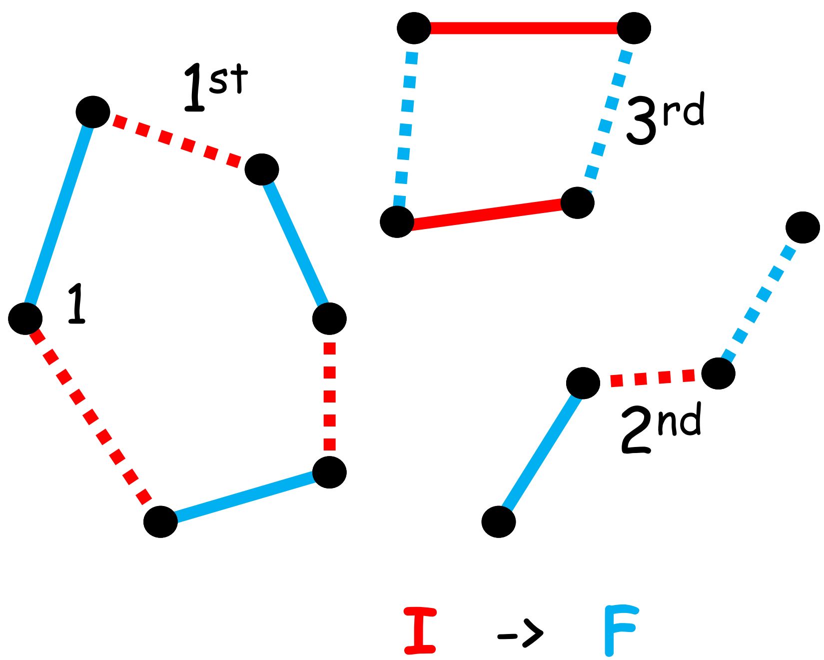
Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

(dashed edges: not in the current matching)

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



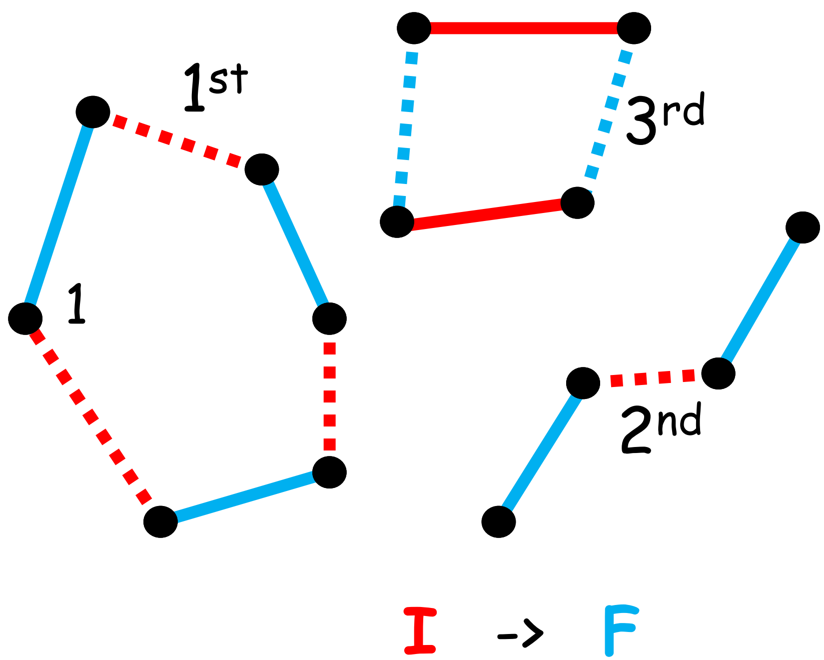
(dashed edges: not in the current matching)

Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



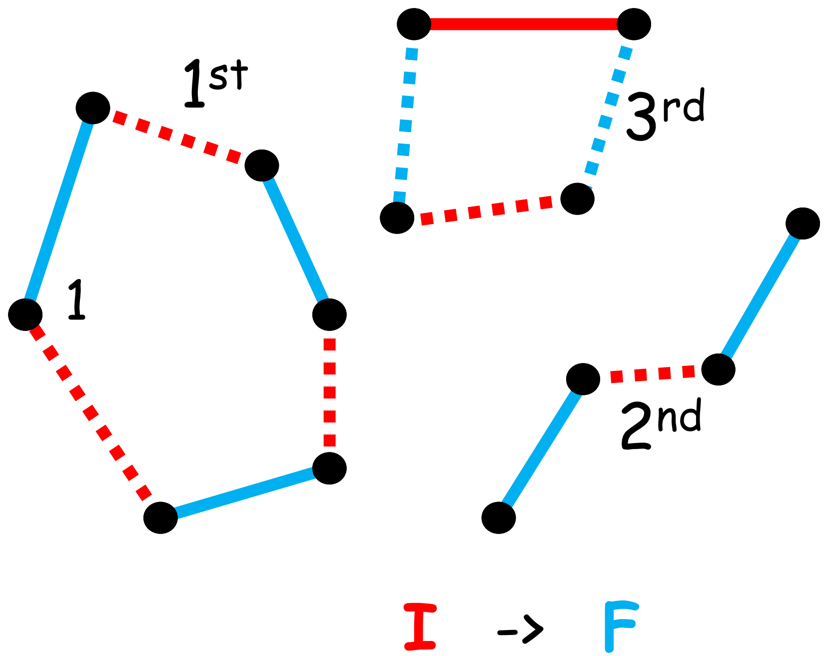
(dashed edges:
not in the current matching)

Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



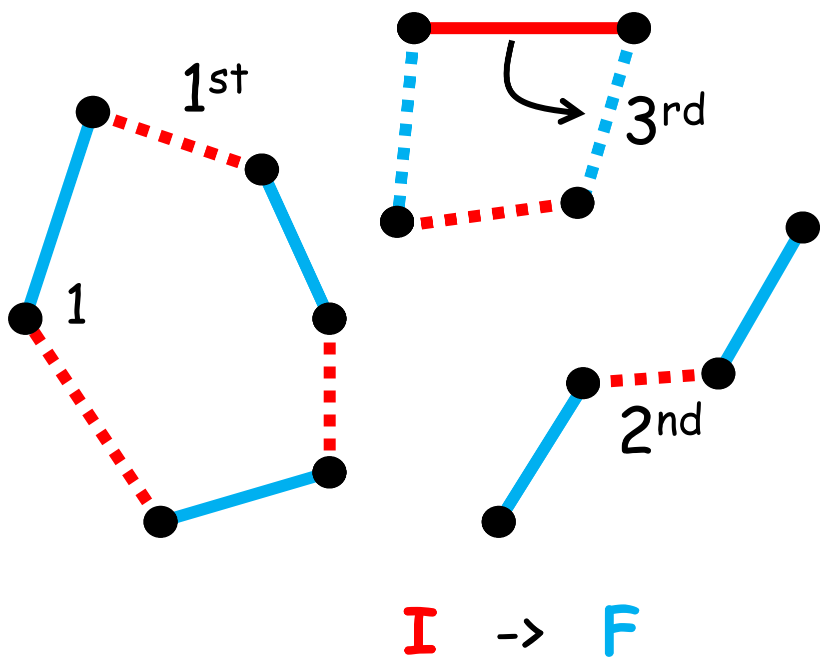
(dashed edges:
not in the current matching)

Going from red to blue:

- take $I \oplus F$ (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



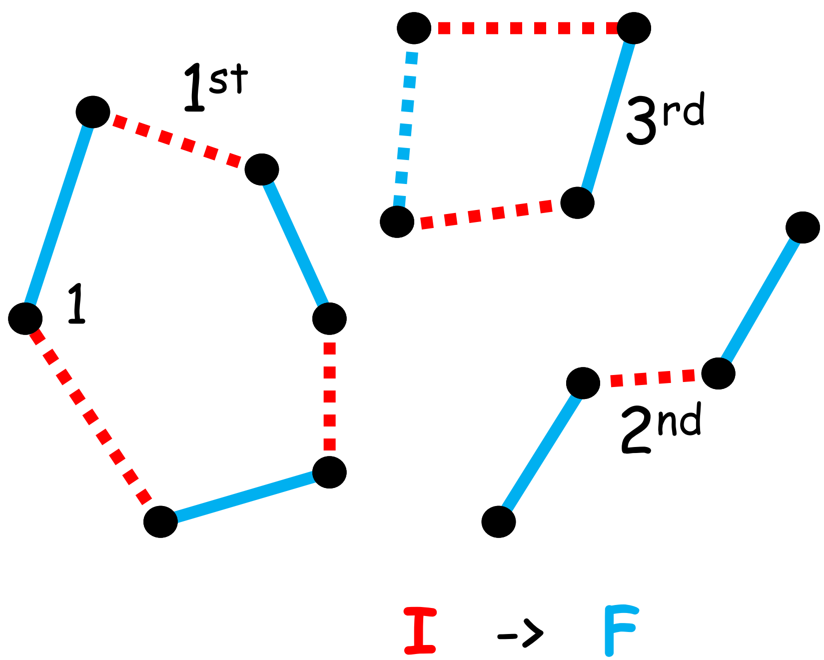
(dashed edges: not in the current matching)

Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



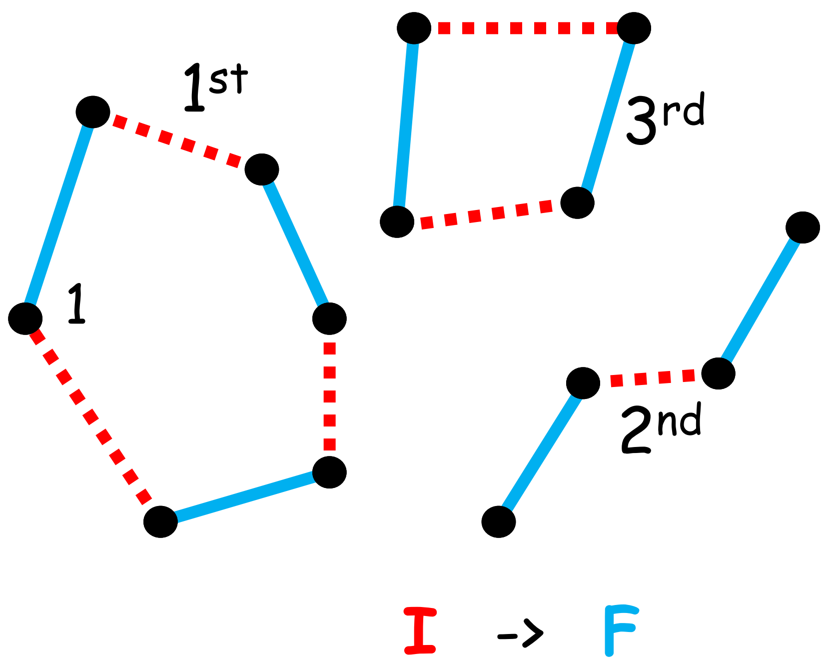
(dashed edges:
not in the current matching)

Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Canonical Paths

For any pair of matchings I,F, define a path from I to F in the transition graph:



(dashed edges:
not in the current matching)

Going from red to blue:

- take I $\oplus$ F (sym. difference)
- components are alternating cycles or paths
- order the components by the lowest vertex
- process components in order
  - -if path:
    - - if needed, remove lower end edge
    - - slide the rest
    - - if needed, add the last edge

Bounding the Congestion

Congestion through transition M->M':

$$\frac{1}{\pi(M)P(M,M')} \sum_{I\text{->}F \text{ path through } M\text{->}M'} \pi(I)\pi(F)\ \text{length}(I\text{->}F)$$

Since $\pi(M)=\pi(I)=\pi(F)=1/|\Omega|$ and $P(M,M')=1/(2m)$, and length(I->F)$\leq$n:

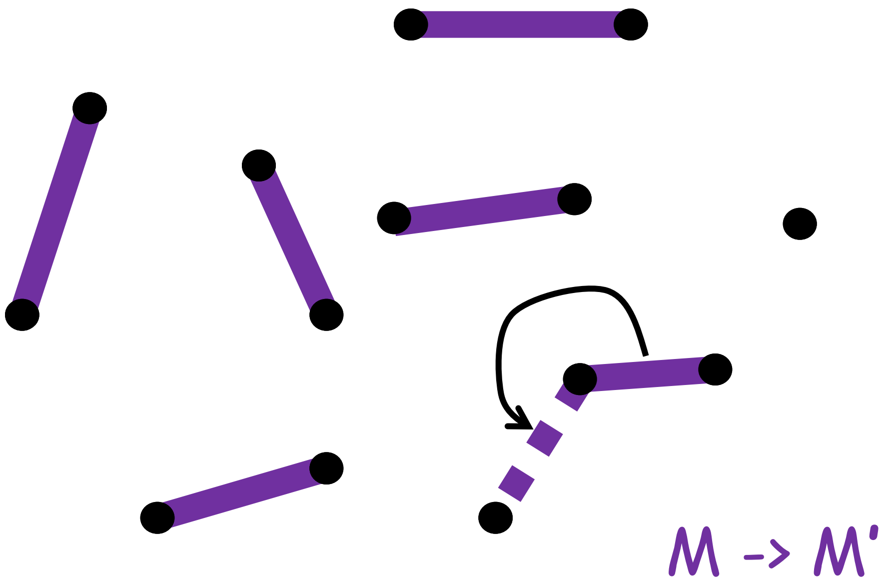
$$\leq \frac{2m}{|\Omega|} \sum_{I\text{->}F \text{ path through } M\text{->}M'} n$$

$$= \frac{2mn}{|\Omega|} (\#\ \text{canonical paths through } M\text{->}M')$$

Bounding the Congestion: Encoding

Let M->M' be a transition.

How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



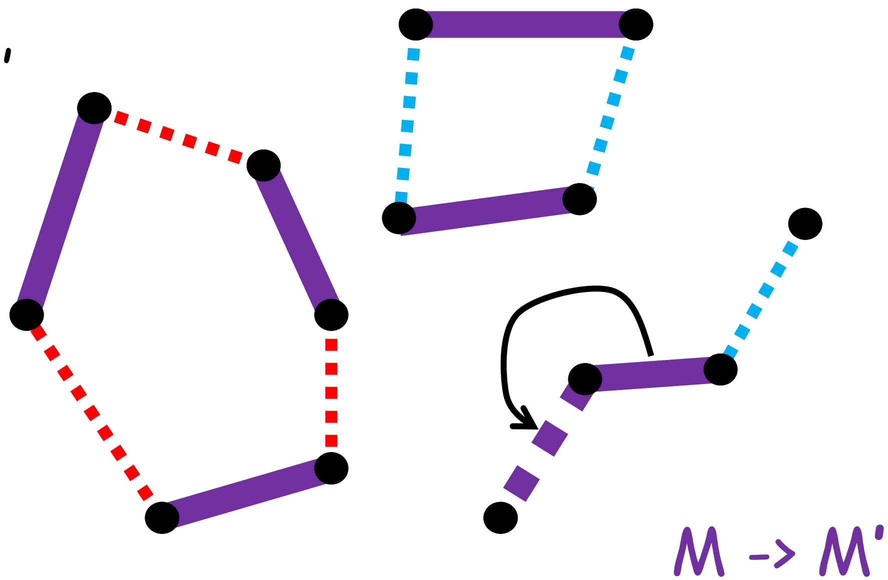
<u>Legend:</u>

- **purple**: transition

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|$poly(n)]

An I -> F path
through M->M'

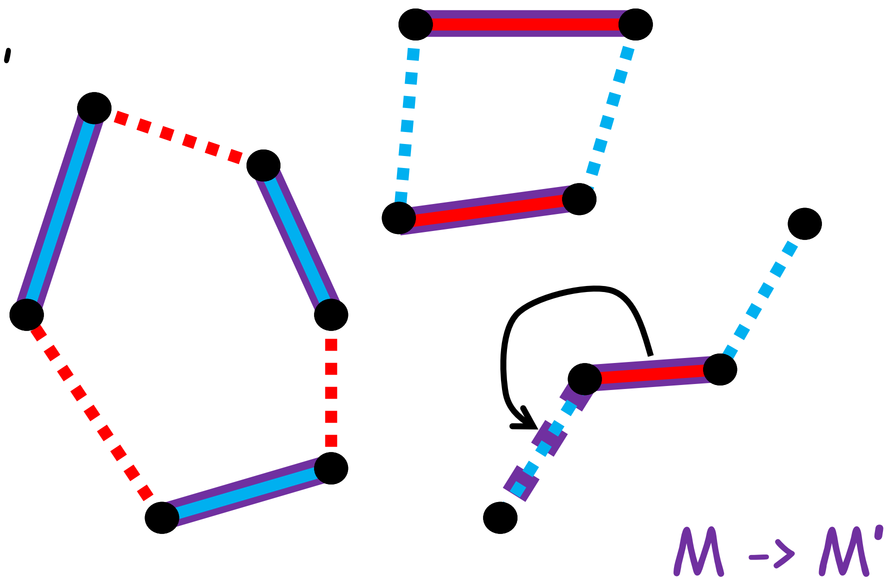


Legend:
- purple: transition
- red: initial matching
- blue: final matching

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]

An I -> F path
through M->M'

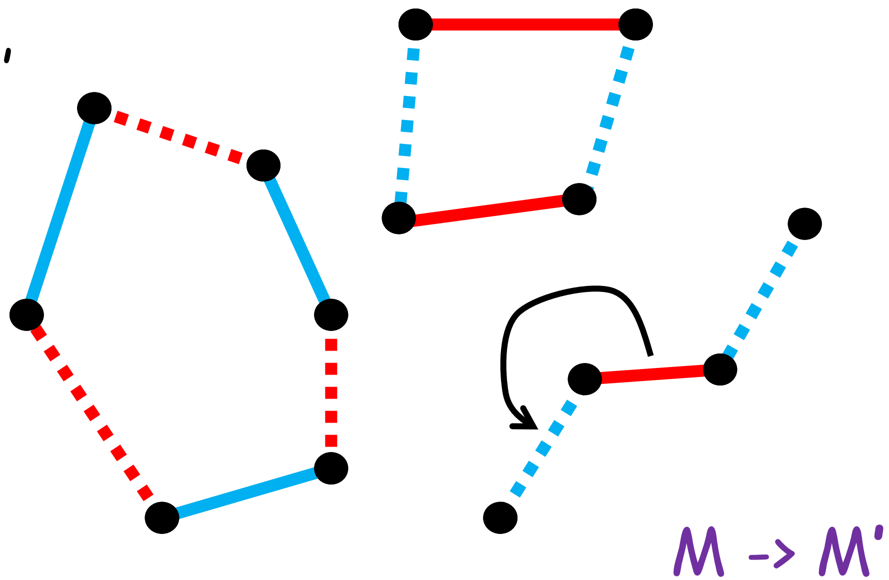


Legend:
- purple: transition
- red: initial matching
- blue: final matching

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\mathrm{poly}(n)$]

An **I** -> **F** path
through M->M'



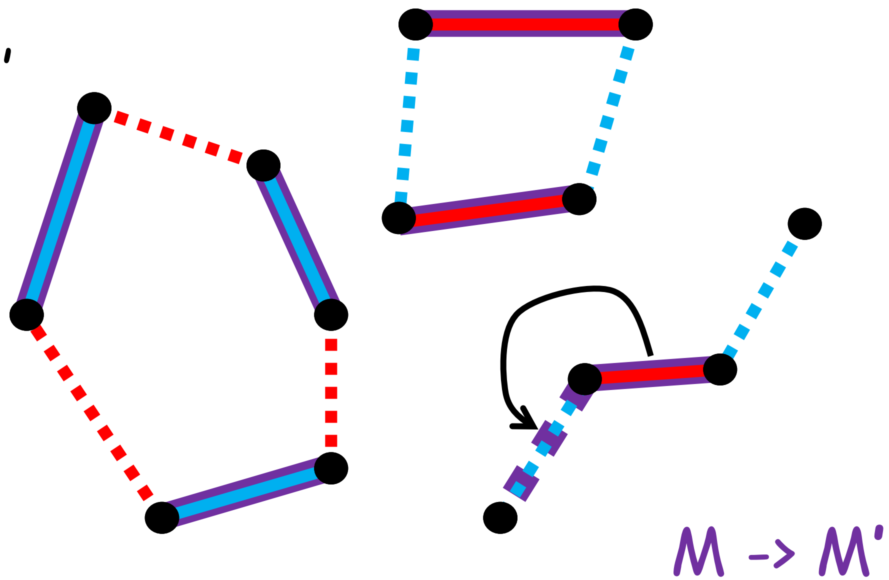
<u>Legend:</u>

- **purple**: transition
- **red**: initial matching
- **blue**: final matching

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]

An I -> F path
through M->M'

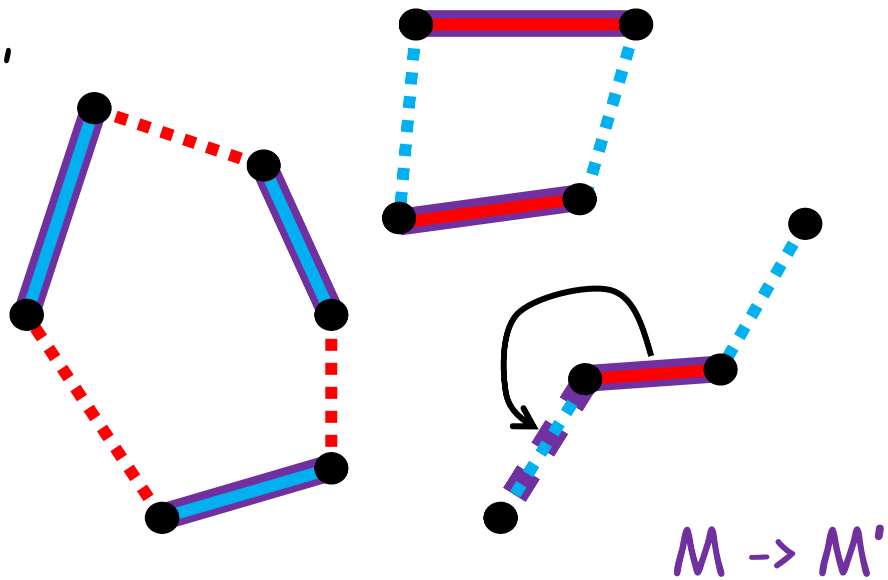


Legend:
- purple: transition
- red: initial matching
- blue: final matching

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]

An I -> F path
through M->M'



<u>Observation:</u>

$I \oplus F - (M \cup M')$
is a matching.

=> encoding E
(for I,F given M)

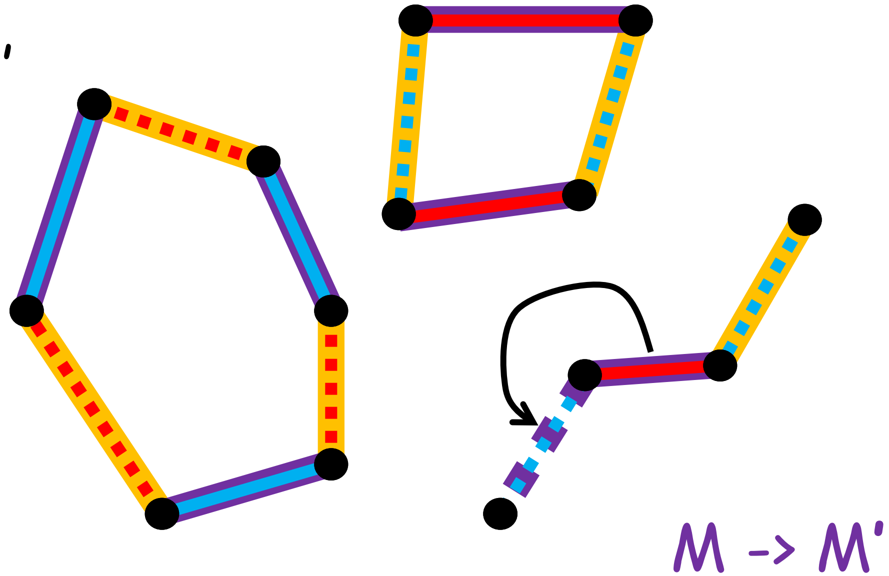
<u>Legend:</u>

- purple: transition
- red: initial matching
- blue: final matching

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\mathrm{poly}(n)$]

An I -> F path through M->M'



Observation:

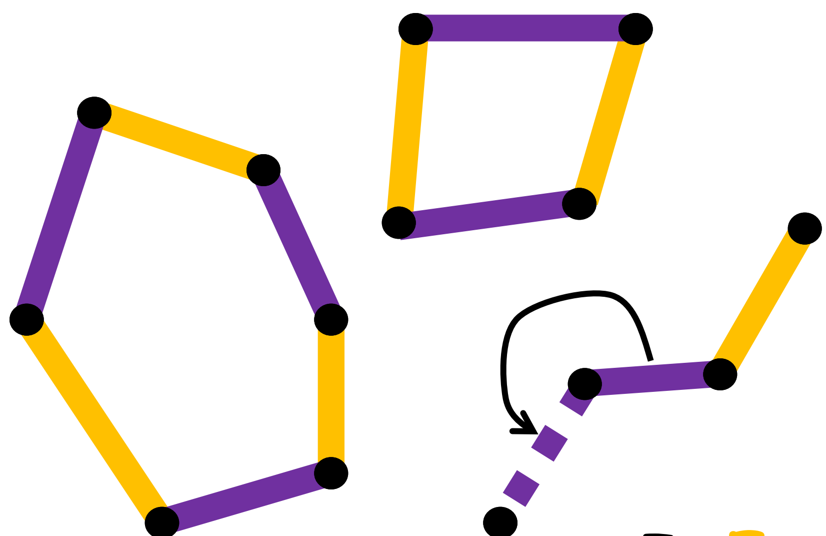
$I \oplus F - (M \cup M')$ is a matching.

=> encoding E (for I,F given M)

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|$poly(n)]



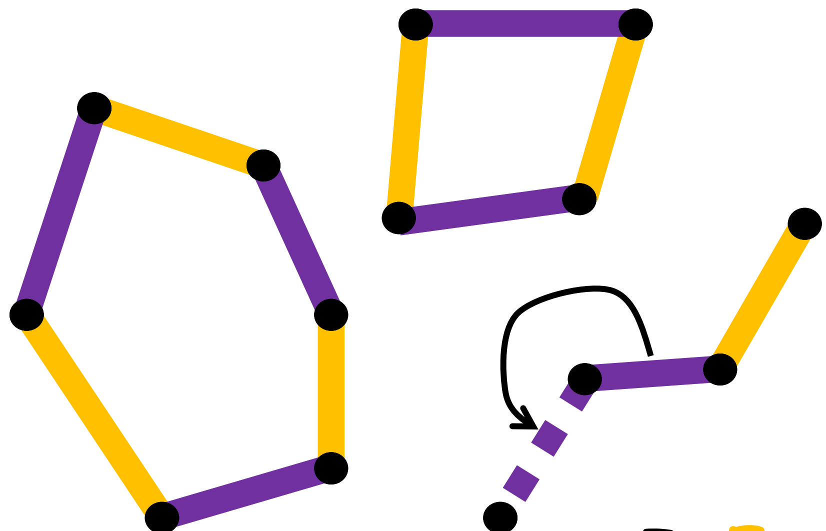
$E = I \oplus F - (M \cup M')$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Is E an "encoding"?
(Given M->M' and E, can reconstruct I,F?)
If yes, then # can.paths through M->M' is $\leq |\Omega|$

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



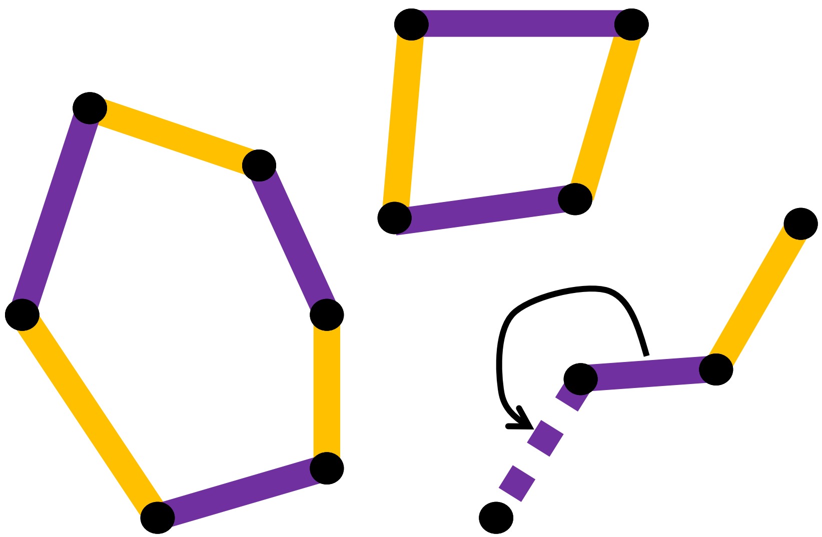
$E = I \oplus F - (M \cup M')$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Is E an "encoding"?
(Given M->M' and E, can reconstruct I,F?)
If yes, then # can.paths through M->M' is $\leq |\Omega|$

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



$E = I \oplus F - (M \cup M')$

Legend:

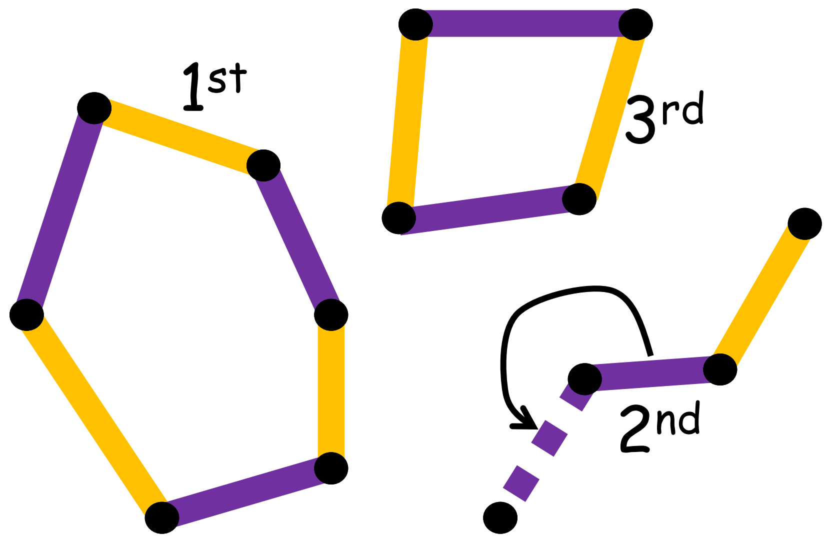
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Know:

- order of components
- currently working on 2nd
- 1st done, 3rd not yet
- current: done up to the transition

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



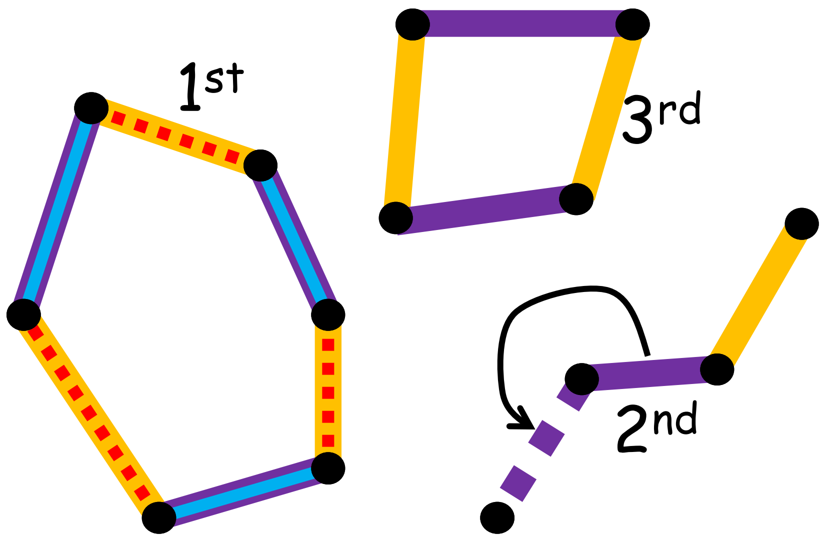
$E = I \oplus F - (M \cup M')$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Know:
- order of components
- currently working on 2nd
- 1st done, 3rd not yet
- current: done up to the transition

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



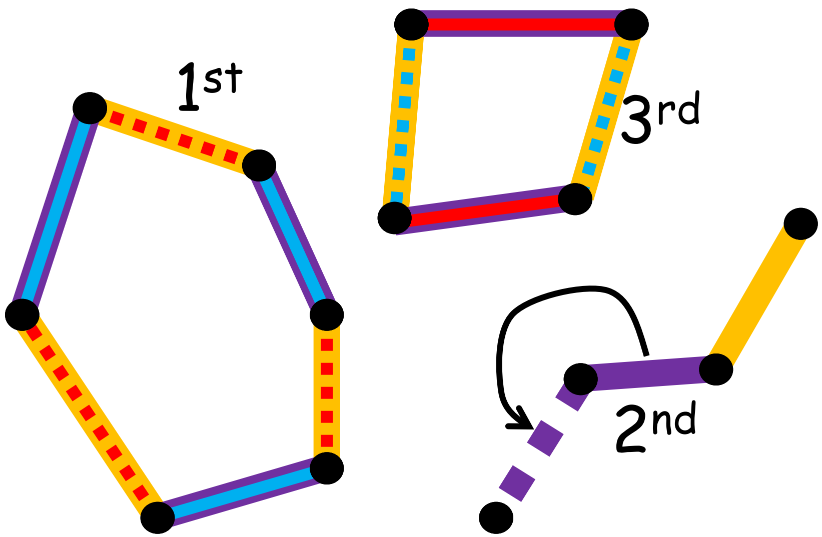
$E = I \oplus F - (M \cup M')$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Know:
- order of components
- currently working on 2nd
- 1st done, 3rd not yet
- current: done up to the transition

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



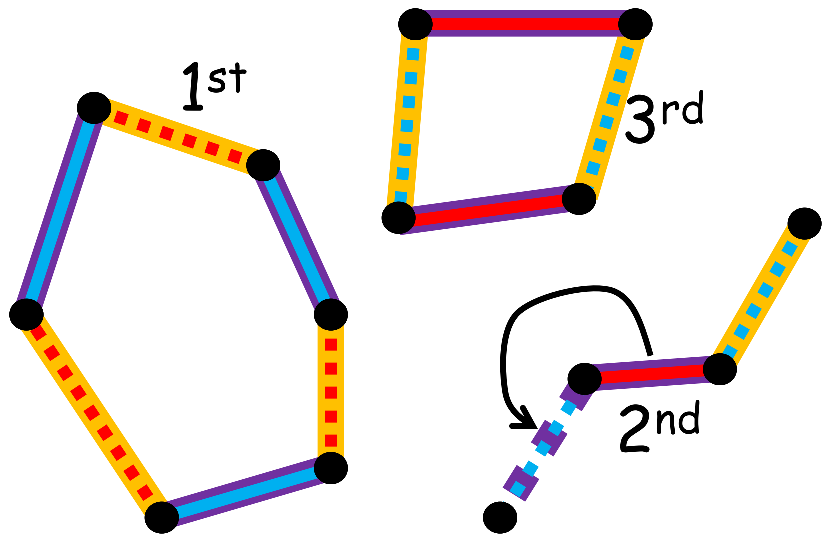
$E = I \oplus F - (M \cup M')$

Know:
- order of components
- currently working on 2nd
- 1st done, 3rd not yet
- current: done up to the transition

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



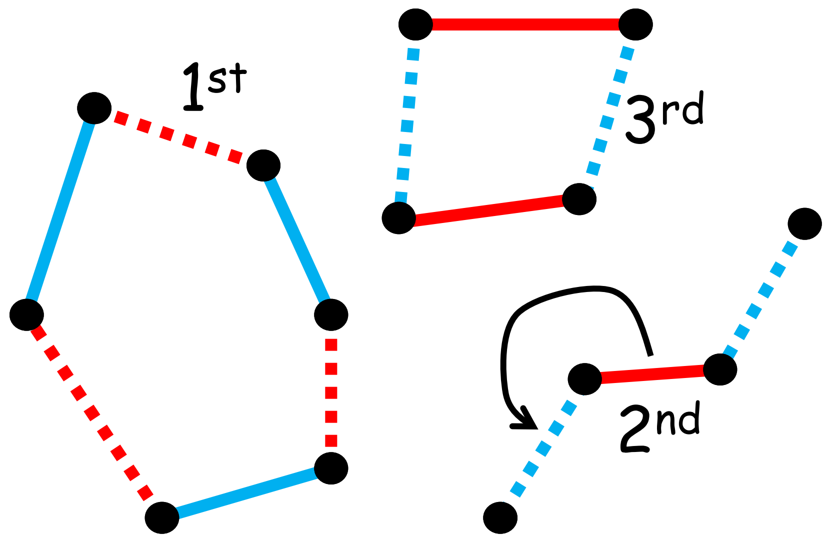
$E = I \oplus F - (M \cup M')$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Know:
- order of components
- currently working on 2nd
- 1st done, 3rd not yet
- current: done up to the transition

Bounding the Congestion: Encoding

Reconstructing I,F from E,M->M':



E = I ⊕ F - (M∪M')

<u>Legend:</u>

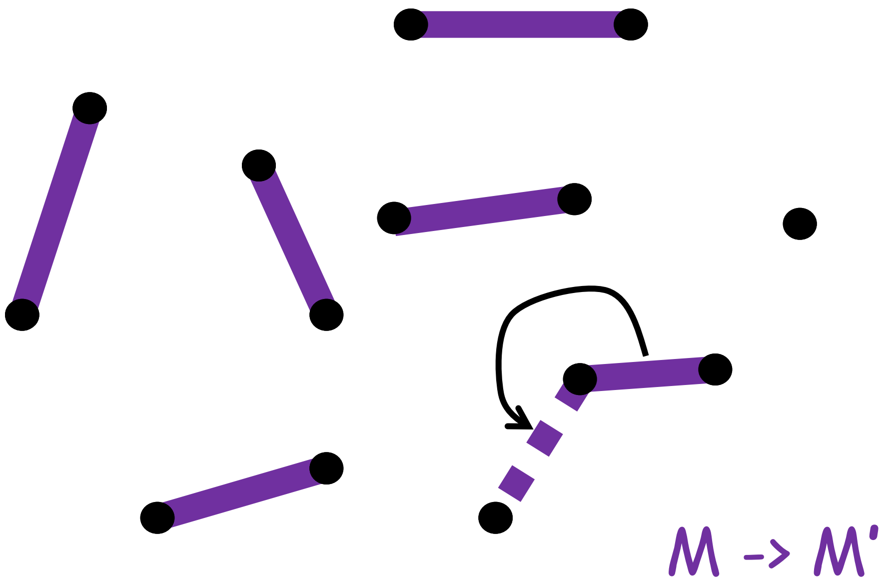
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Know:

- order of components
- currently working on 2nd
- 1st done, 3rd not yet
- current: done up to the transition

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]

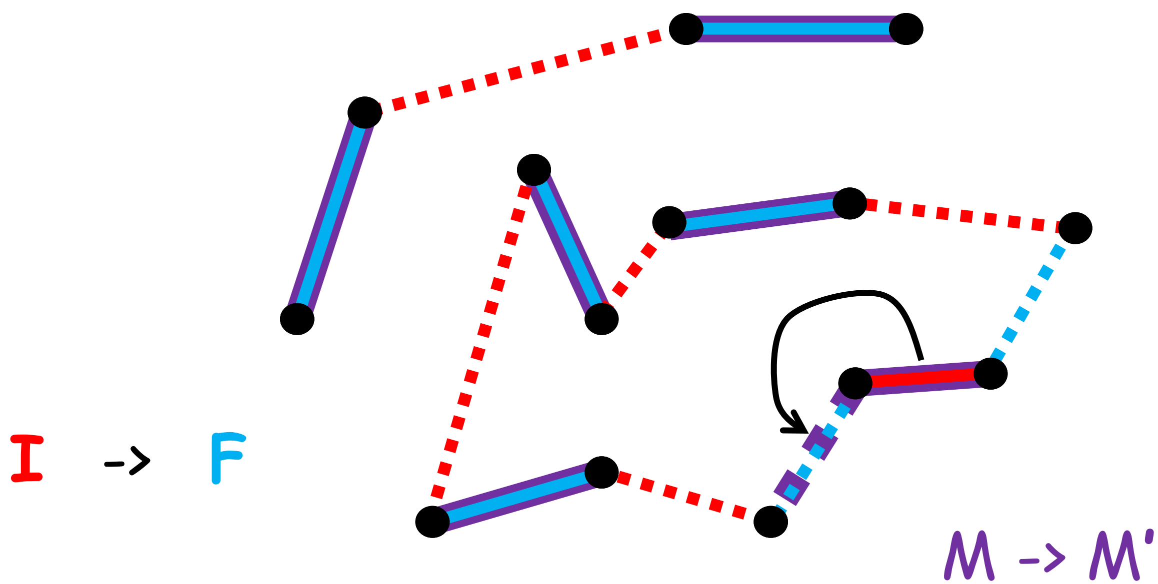


Legend:

- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

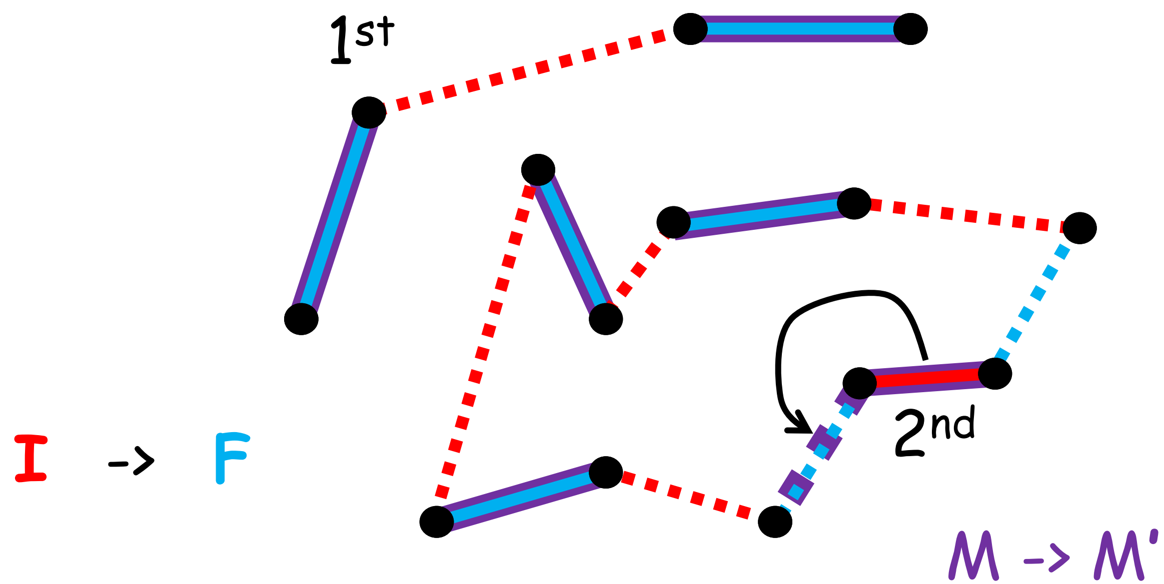
Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

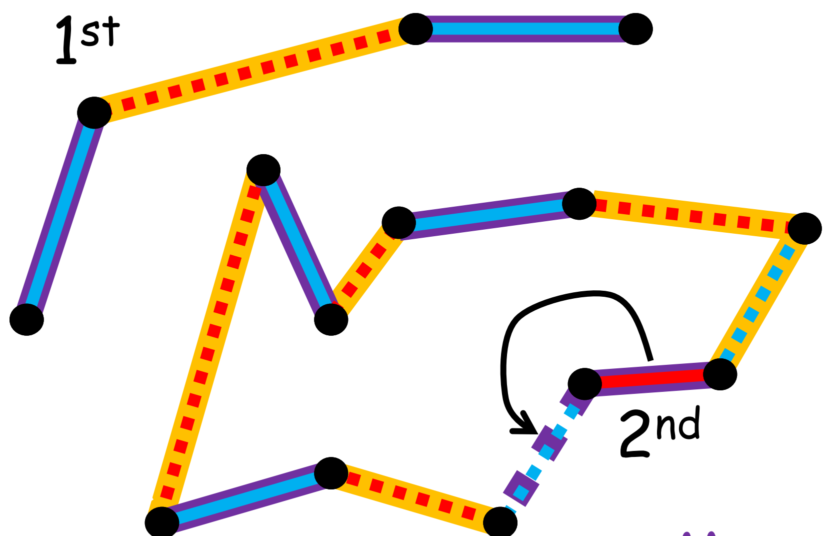
Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|$poly(n)]



$E = I \oplus F - (M \cup M')$

Not a matching…

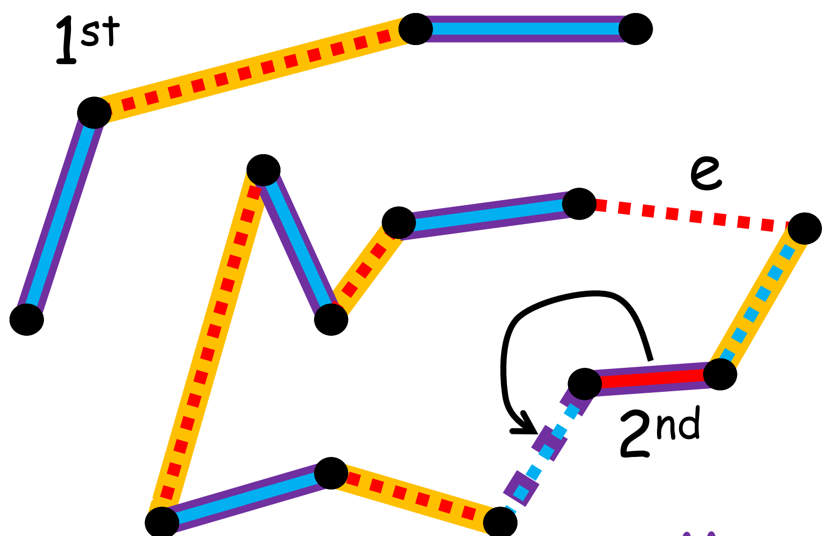
I -> F

M -> M'

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



I -> F

M -> M'

$E = I \oplus F - (M \cup M')$
$- \{e\}$

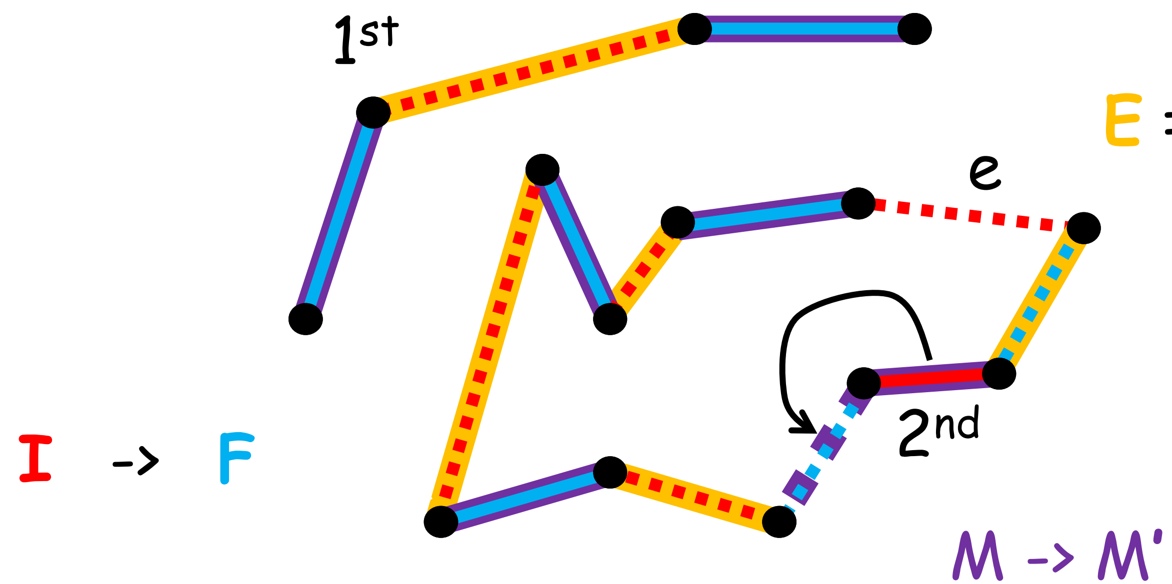
Now a matching:
$E \in \Omega$

<u>Legend:</u>

- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|$poly(n)]



$E = I \oplus F - (M \cup M') - \{e\}$

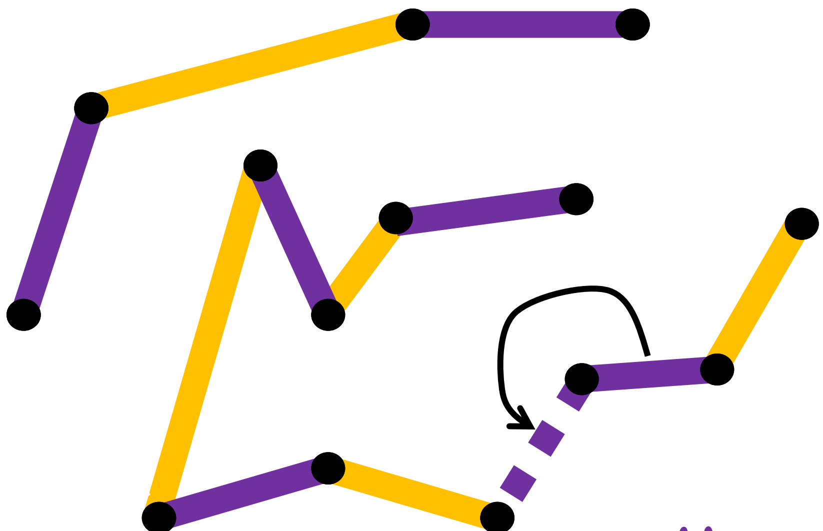
Now a matching:
$E \in \Omega$

Can "decode" E into I,F?

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



$E = I \oplus F - (M \cup M')$
$- \{e\}$

Now a matching:
$E \in \Omega$

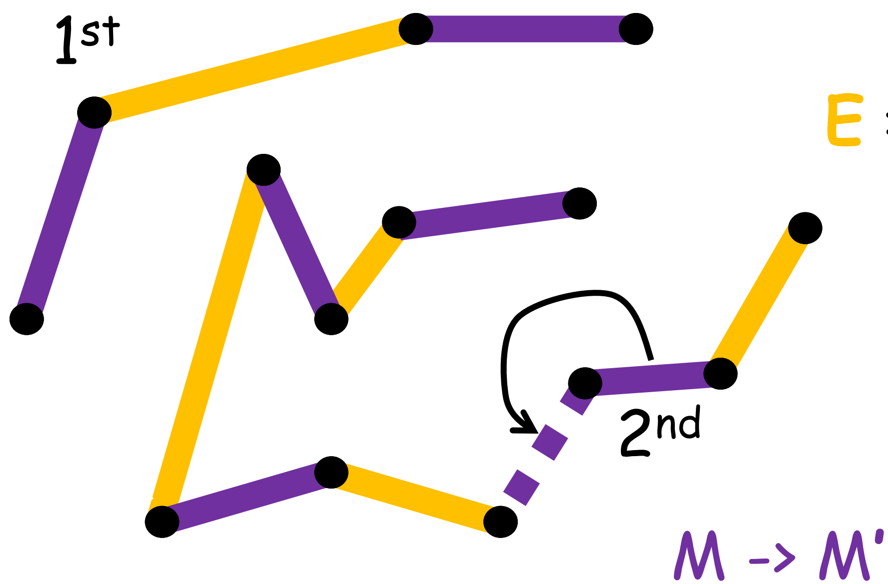
M -> M'

Can "decode" E into I,F?

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq$ $|\Omega|$poly(n)]



$E = I \oplus F - (M \cup M')$
$- \{e\}$

Now a matching:
$E \in \Omega$

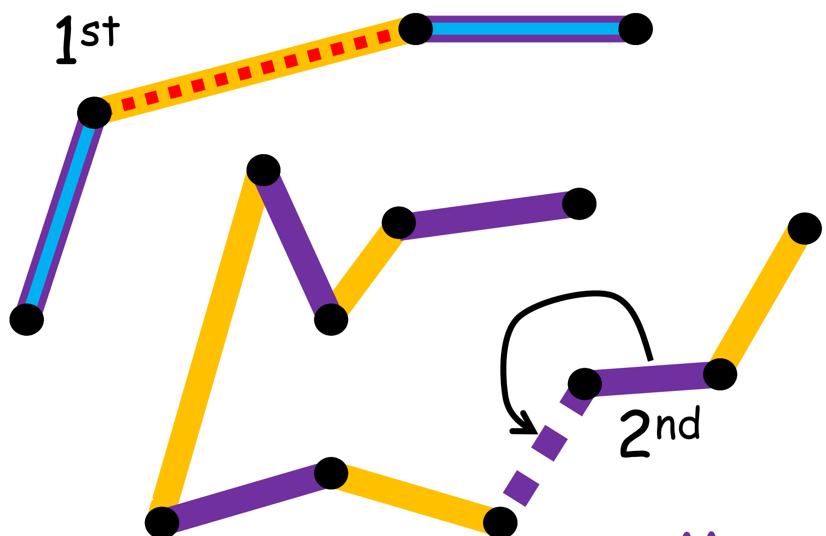
Can "decode" E into I,F?

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.

How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



$$E = I \oplus F - (M \cup M') - \{e\}$$

Now a matching: $E \in \Omega$

I -> F

M -> M'

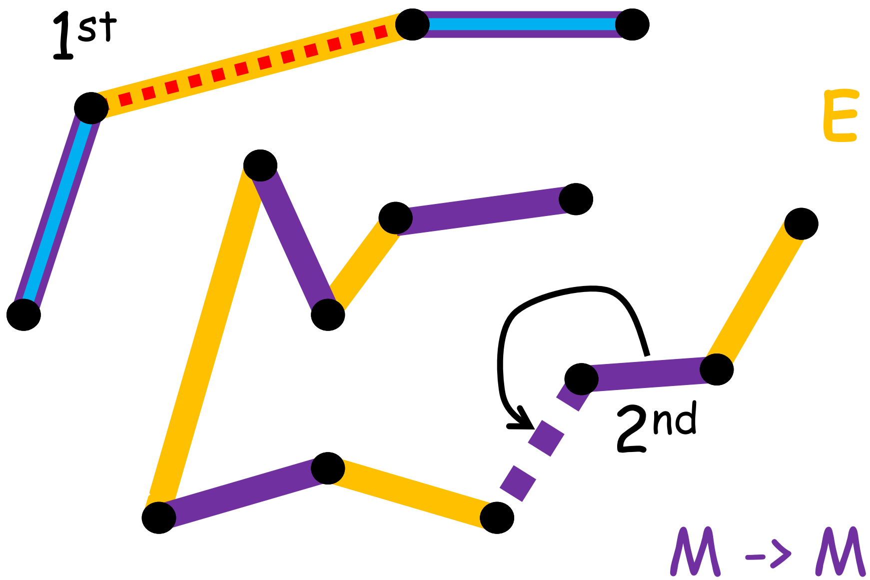
Can "decode" E into I,F?

<u>Legend:</u>

- **purple**: transition
- **red**: initial matching
- **blue**: final matching
- **orange**: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



$E = I \oplus F - (M \cup M')$
$- \{e\}$

Now a matching:
$E \in \Omega$

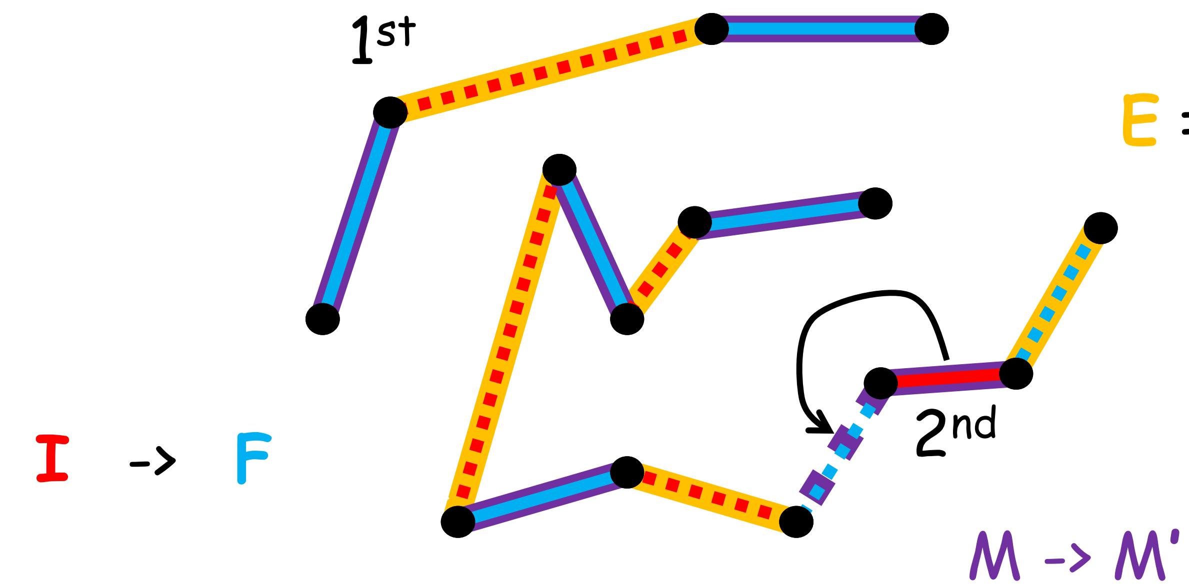
I -> F

Can "decode" E into I,F?
- Is 2nd component a path?

<u>Legend:</u>
- **purple**: transition
- **red**: initial matching
- **blue**: final matching
- **orange**: encoding

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\mathrm{poly}(n)$]



$E = I \oplus F - (M \cup M')$
$- \{e\}$

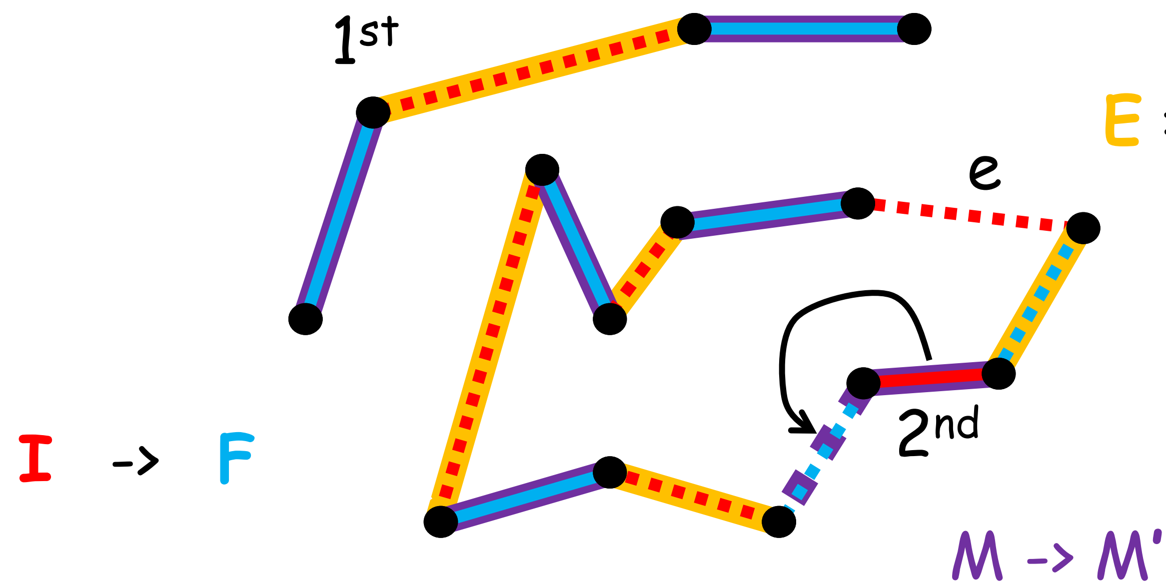
Now a matching:
$E \in \Omega$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Can "decode" E into I,F?
- Is 2nd component a path?

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|$poly(n)]



$E = I \oplus F - (M \cup M')$
$- \{e\}$

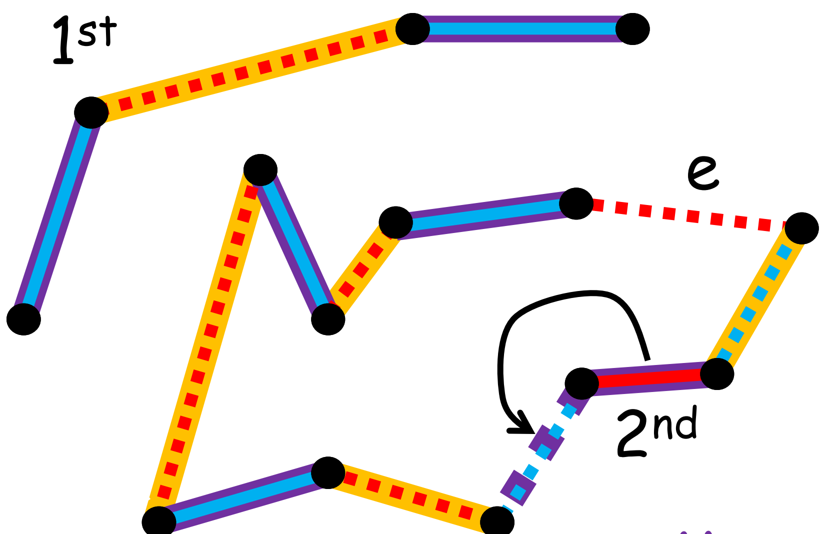
Now a matching:
$E \in \Omega$

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Can "decode" E into I,F?
- Is 2nd component a path?
- Or a cycle?

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|\text{poly}(n)$]



$E = I \oplus F - (M \cup M')$
$- \{e\}$

Now a matching:
$E \in \Omega$

I -> F

M -> M'

Legend:
- purple: transition
- red: initial matching
- blue: final matching
- orange: encoding

Can "decode" E into I,F?
- Is 2nd component a path?
- Or a cycle?

Redefine encoding:
$E' := (E,0)$
$E' := (E,1)$
$\in \Omega \times \{0,1\}$

Bounding the Congestion: Encoding

Let M->M' be a transition.
How many canonical paths go through it ? [Want $\leq |\Omega|$poly(n)]

- For the sliding transition: $\leq 2|\Omega|$

- Need to analyze the add and remove transitions

<u>Bound on the congestion:</u>

$$\rho \leq \frac{2mn}{|\Omega|} (\text{\# can. paths through M->M'}) = \frac{2mn}{|\Omega|} 2|\Omega| = 4mn$$

<u>Mixing time:</u> $t_{mix}(\epsilon) = O(mn \log(1/(\epsilon\pi_{min})))$
$= O^*(mn^2)$ [O* - ignore polylog]

<u>FPRAS:</u> $O(T(n,m,\epsilon/(6m))\ m^2/\epsilon^2) = O^*(m^3n^2/\epsilon^2)$

A Markov Chain for Perfect Matchings

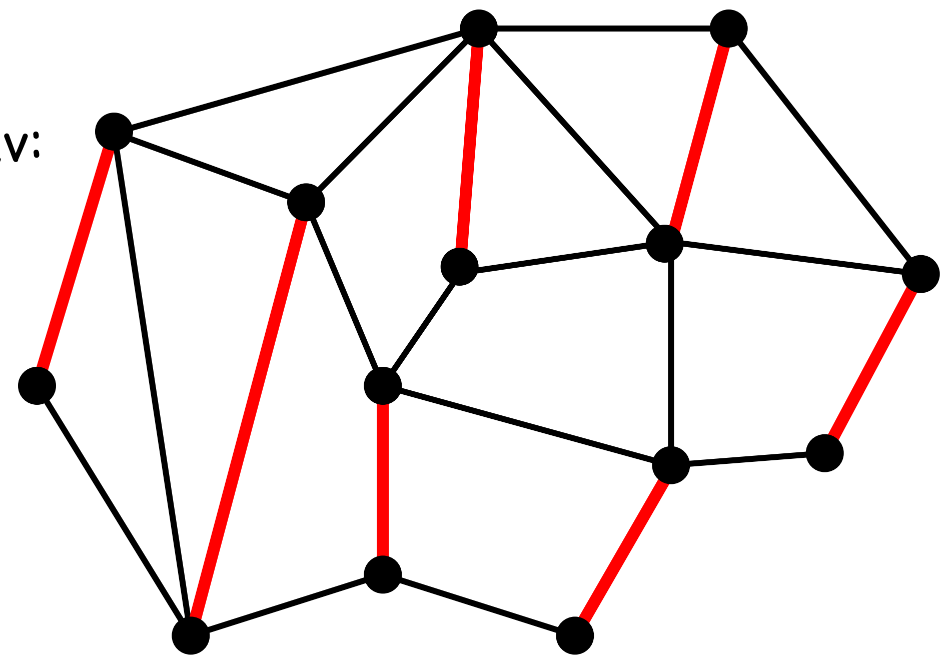
Input: a graph G

Exactly 2 vertices not matched

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if w=u or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Input: a graph G

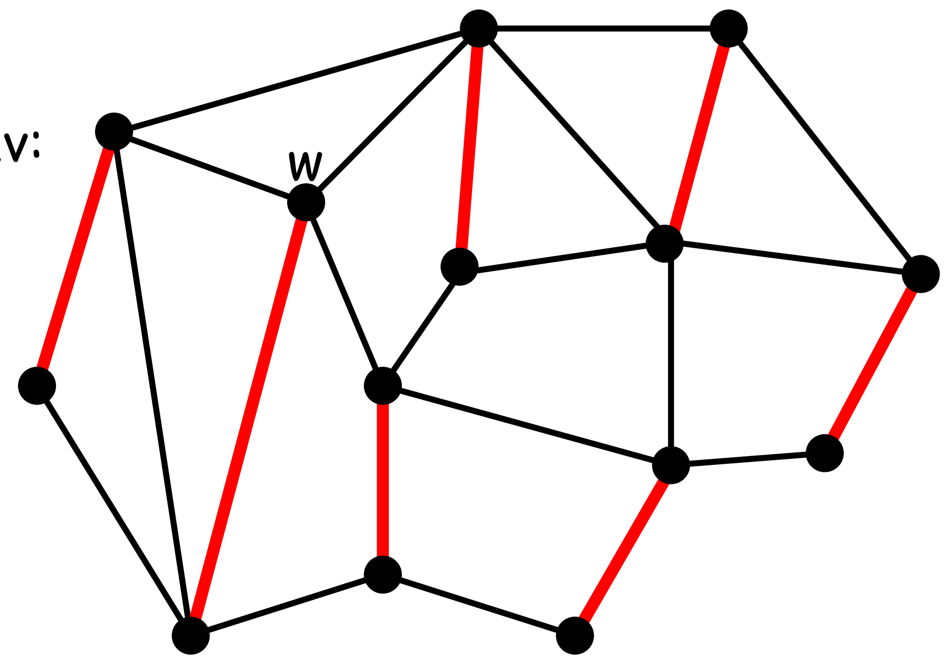
Exactly 2 vertices not matched

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):

Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if $w=u$ or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

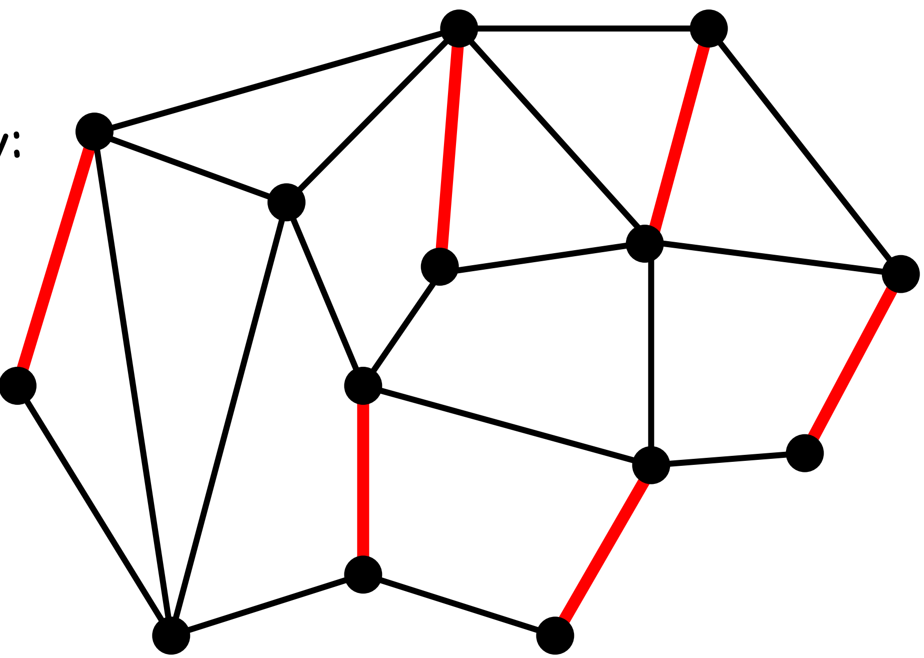
<u>Input</u>: a graph G

Exactly 2 vertices not matched

<u>State space Ω</u>: all perfect and near-perfect matchings of G

<u>Markov chain</u> (slide chain):
Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
    - if $w=u$ or v, add (u,v) if can
    - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Exactly 2 vertices not matched

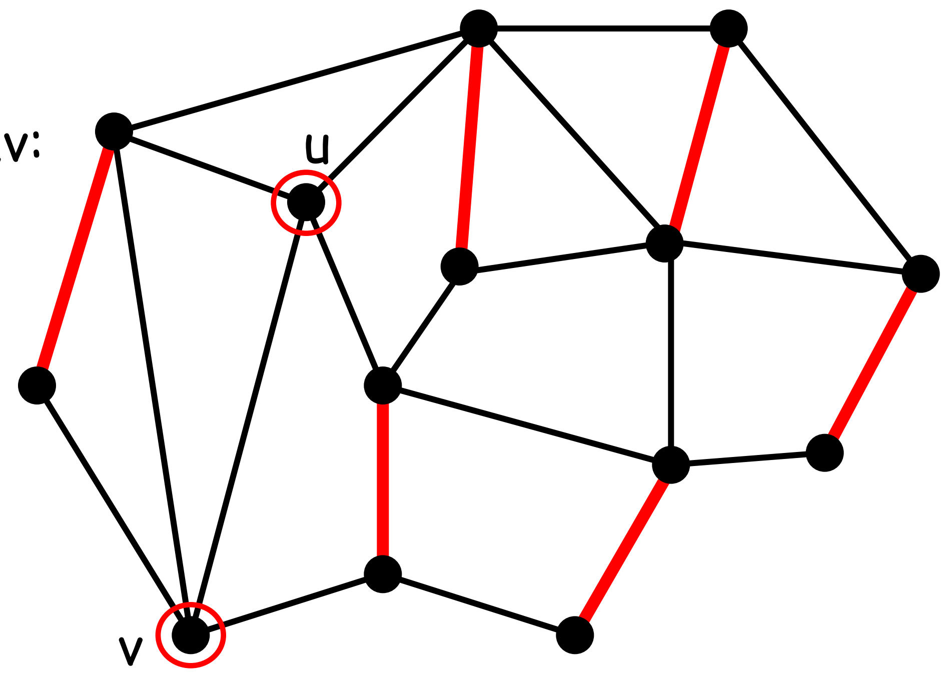
Input: a graph G

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):

Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if $w=u$ or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Input: a graph G

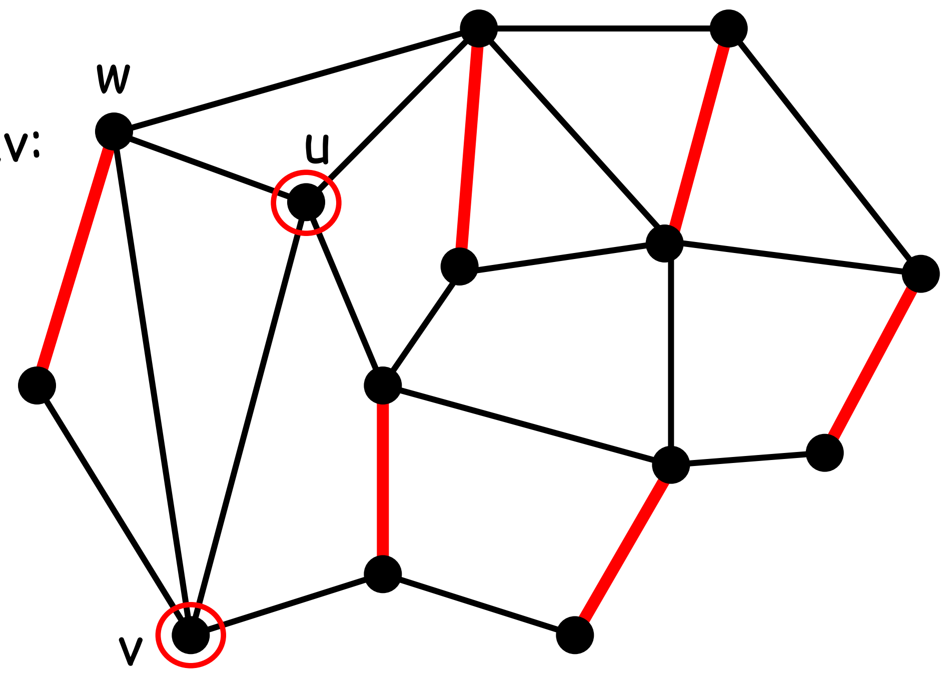
Exactly 2 vertices not matched

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):

Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if w=u or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Input: a graph G

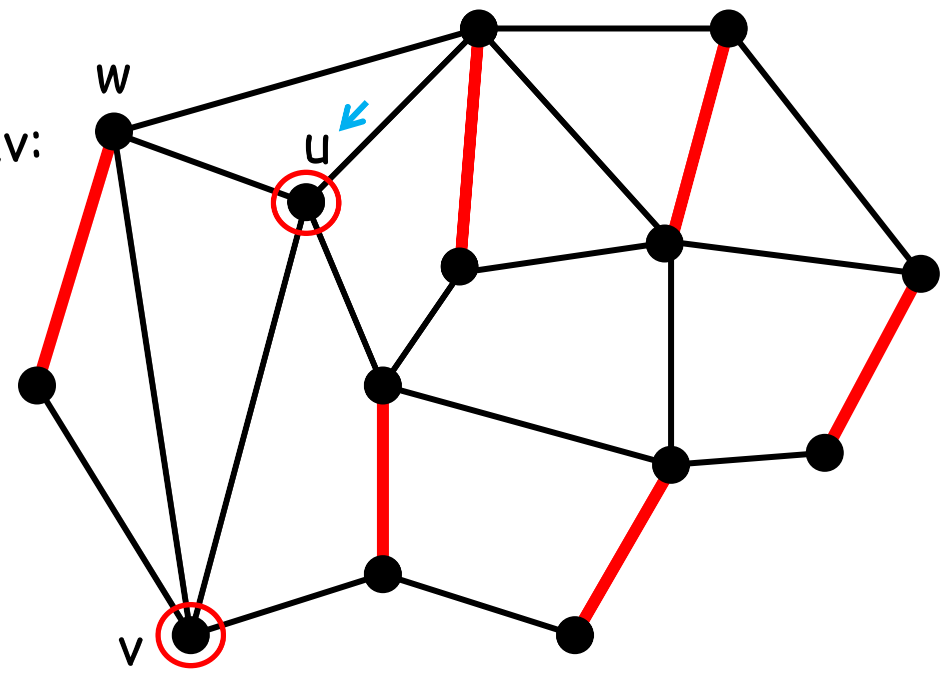
Exactly 2 vertices not matched

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):

Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if $w=u$ or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Exactly 2 vertices not matched

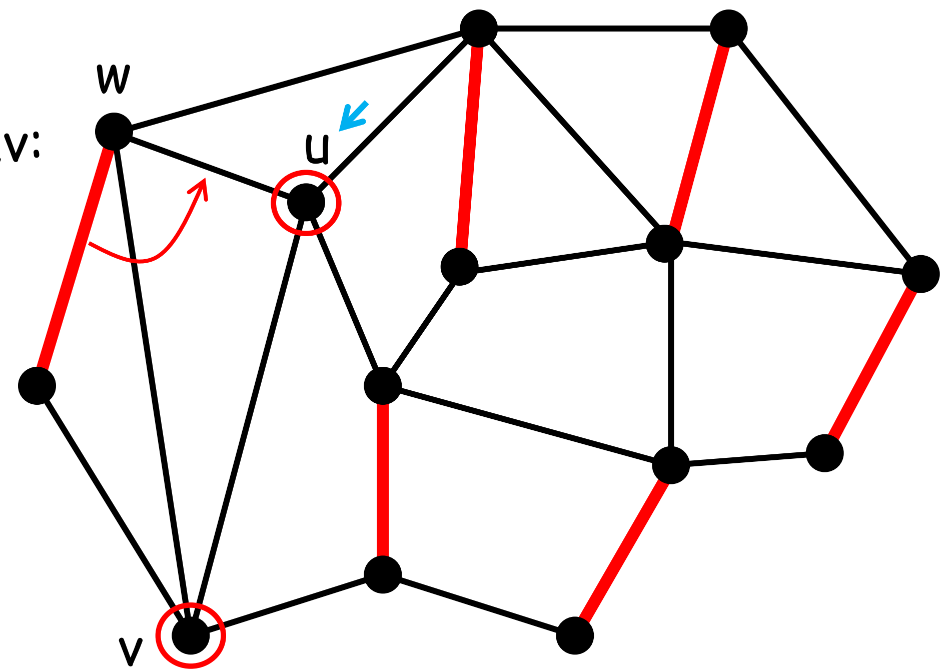
Input: a graph G

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):

Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if $w=u$ or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

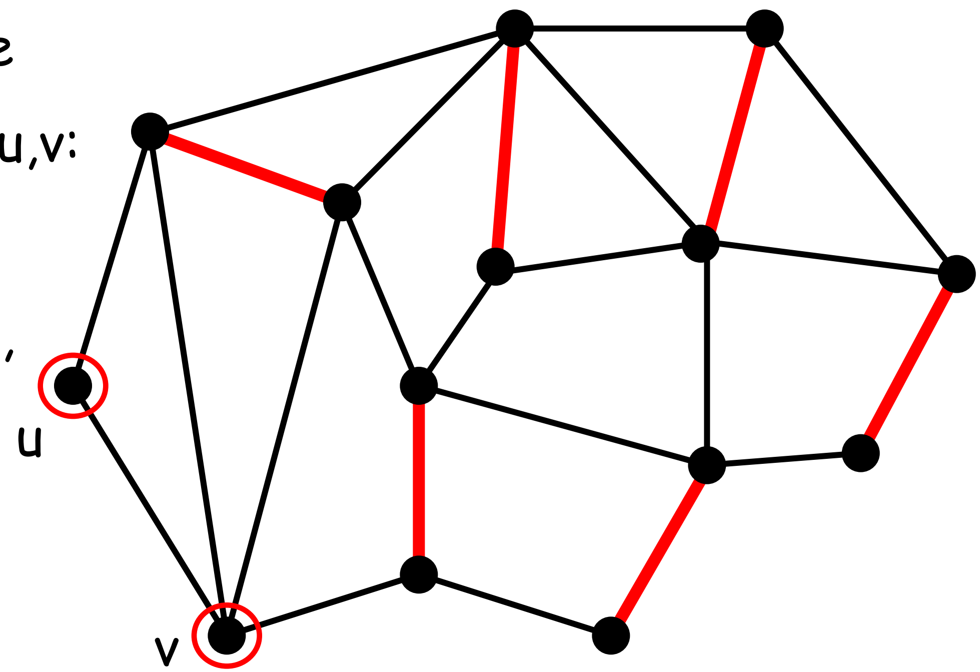
<u>Input</u>: a graph G

Exactly 2 vertices not matched

<u>State space Ω</u>: all perfect and near-perfect matchings of G

<u>Markov chain</u> (slide chain):
Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
    - if w=u or v, add (u,v) if can
    - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Input: a graph G

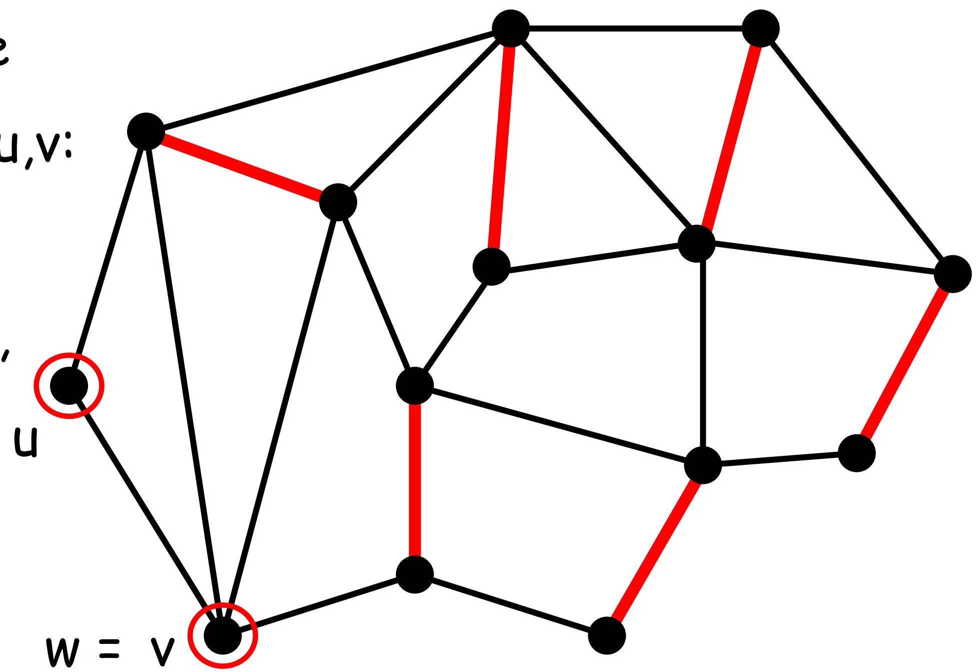
Exactly 2 vertices not matched

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):

Let M be the current matching, we get the next state by choosing a random vertex w and:

- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if $w=u$ or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

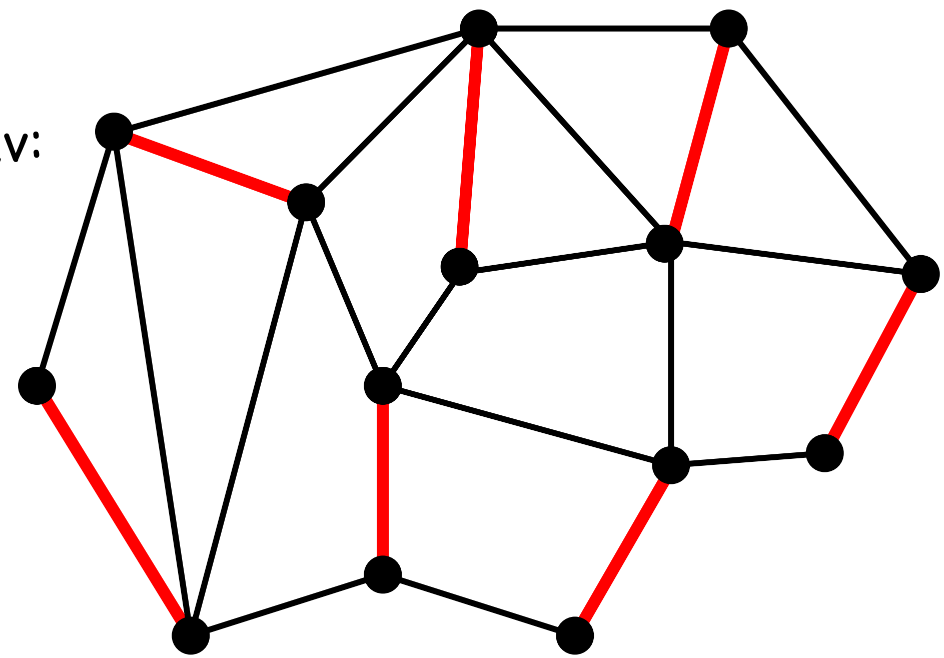
Input: a graph G

Exactly 2 vertices not matched

State space Ω: all perfect and near-perfect matchings of G

Markov chain (slide chain):
Let M be the current matching, we get the next state by choosing a random vertex w and:

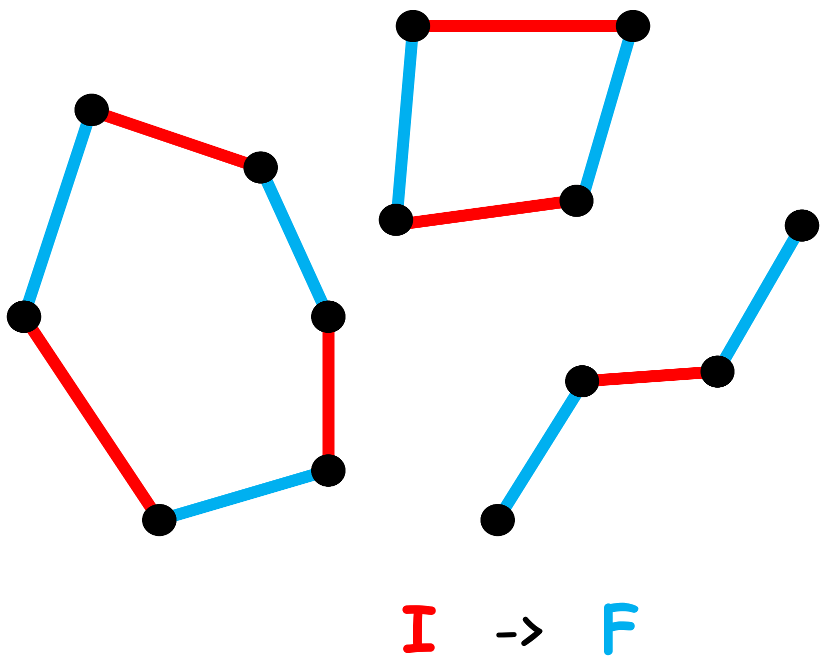
- if M is perfect: remove w's edge
- if M is near-perfect with holes u,v:
  - if $w=u$ or v, add (u,v) if can
  - else, randomly choose u or v, replace w's current edge by (u,w) or (v,w)



A Markov Chain for Perfect Matchings

Analysis of this MC:

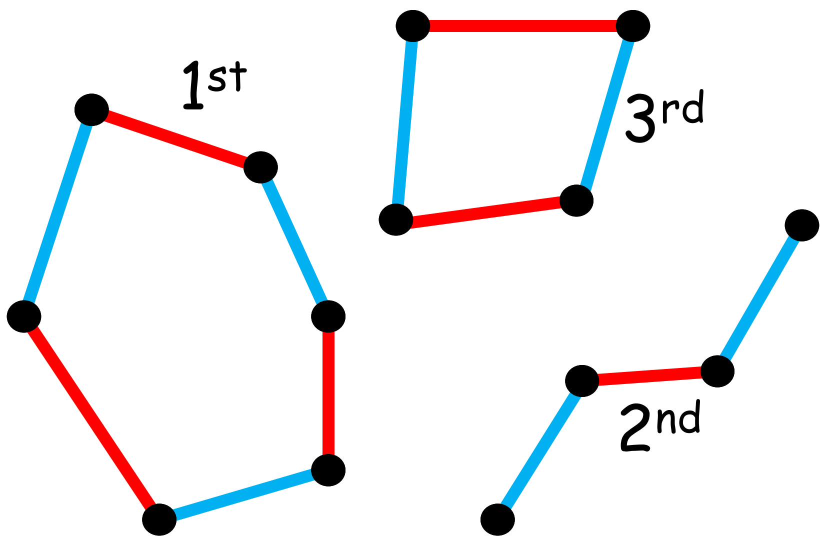
- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last



A Markov Chain for Perfect Matchings

Analysis of this MC:

- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last

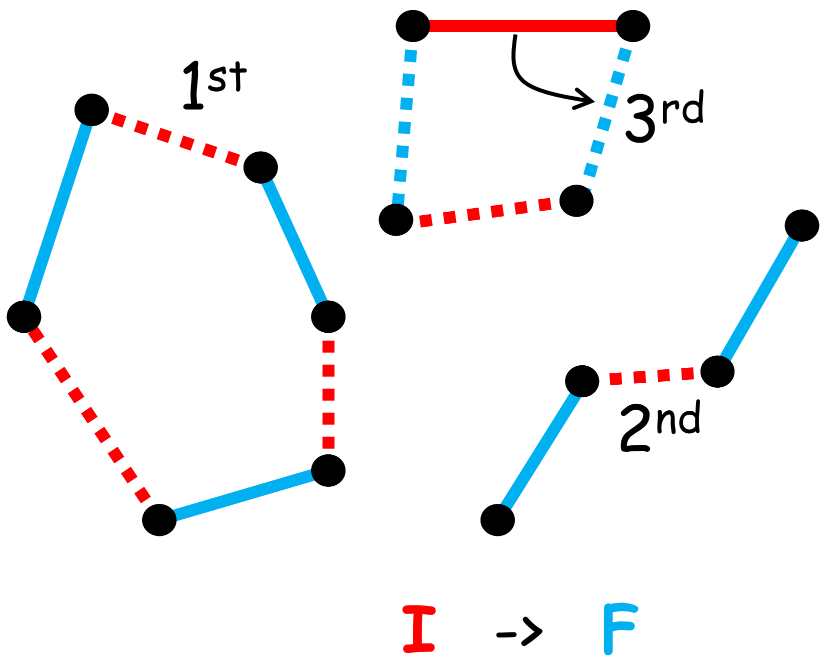


I -> F

A Markov Chain for Perfect Matchings

<u>Analysis of this MC:</u>

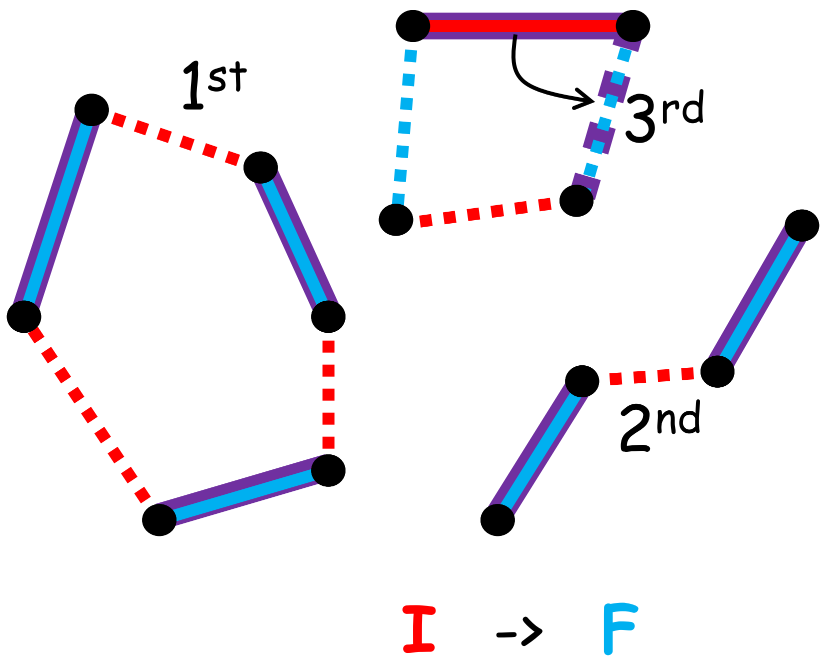
- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last



A Markov Chain for Perfect Matchings

Analysis of this MC:

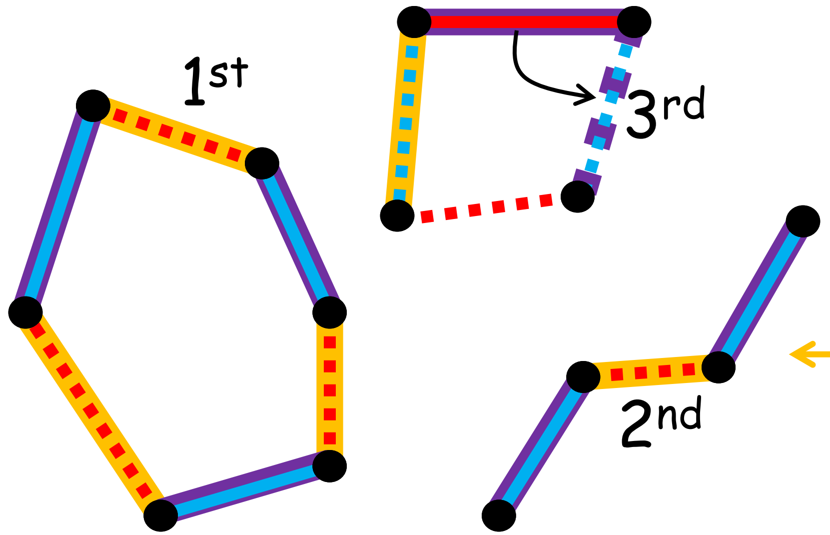
- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last



A Markov Chain for Perfect Matchings

Analysis of this MC:

- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last



Want to process last, otherwise 4 holes -> not in Ω

I -> F

A Markov Chain for Perfect Matchings

<u>Analysis of this MC:</u>

- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last
- for near to near:
  go through a random perfect matching

  (Instead of canonical paths, split into a flow.)

A Markov Chain for Perfect Matchings

Analysis of this MC:

- canonical paths as before for perfect to perfect
- for near to perfect:
  exactly one alternating path – process it last
- for near to near:
  go through a random perfect matching

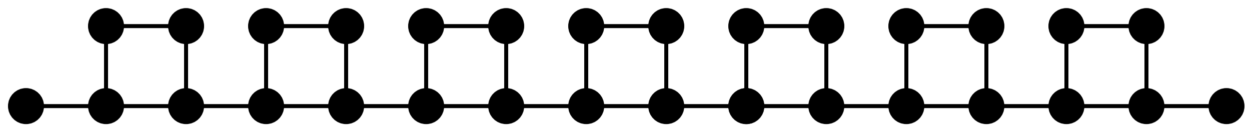
  (Instead of canonical paths, split into a flow.)

Mixing time: $t_{mix}(\epsilon) = O^*(n^3\,(\#nears/\#perfects))$

Polynomial if # near-perfect / # perfect matchings is polynomial…
E.g. for dense graphs: every vertex of degree > n/2.

A Markov Chain for Perfect Matchings

What if improve mixing time analysis?

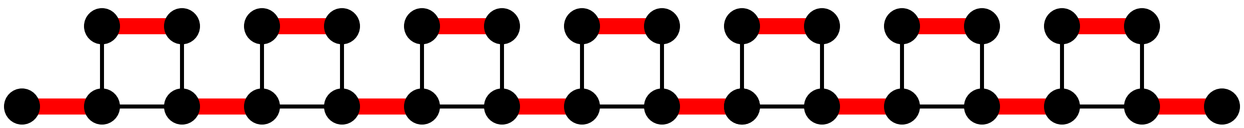


Mixing time: $t_{mix}(\epsilon) = O^*(n^3\,(\#nears/\#perfects))$

Polynomial if # near-perfect / # perfect matchings is polynomial…
E.g. for dense graphs: every vertex of degree > n/2.

A Markov Chain for Perfect Matchings

What if improve mixing time analysis?



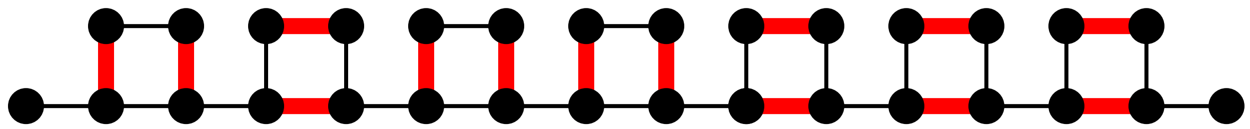
Just one perfect matching…

Mixing time: $t_{mix}(\epsilon) = O^*(n^3\,(\#nears/\#perfects))$

Polynomial if # near-perfect / # perfect matchings is polynomial…
E.g. for dense graphs: every vertex of degree > n/2.

A Markov Chain for Perfect Matchings

<u>What if improve mixing time analysis?</u>



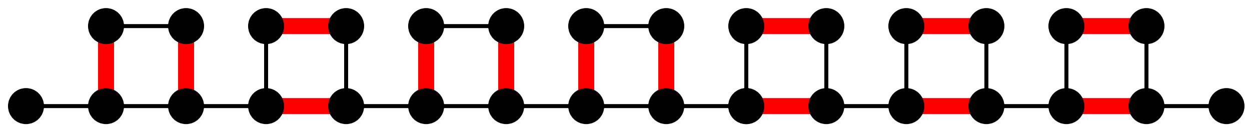
Just one perfect matching... But exponentially many nears!

<u>Mixing time:</u> $t_{mix}(\epsilon) = O^*(n^3\,(\#nears/\#perfects))$

Polynomial if # near-perfect / # perfect matchings is polynomial...
E.g. for dense graphs: every vertex of degree > n/2.

A Markov Chain for Perfect Matchings

What if improve mixing time analysis?



Just one perfect matching... But exponentially many nears!

This MC good only if #nears/#perfects polynomial...

Mixing time: $t_{mix}(\epsilon) = O^*(n^3\,(\#nears/\#perfects))$

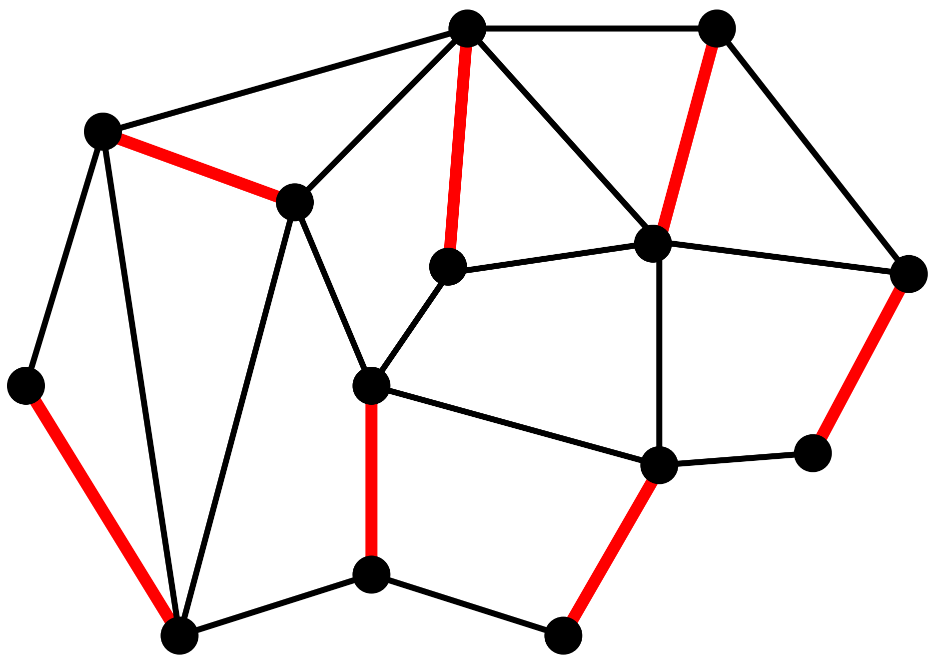
Polynomial if # near-perfect / # perfect matchings is polynomial...
E.g. for dense graphs: every vertex of degree > n/2.

Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

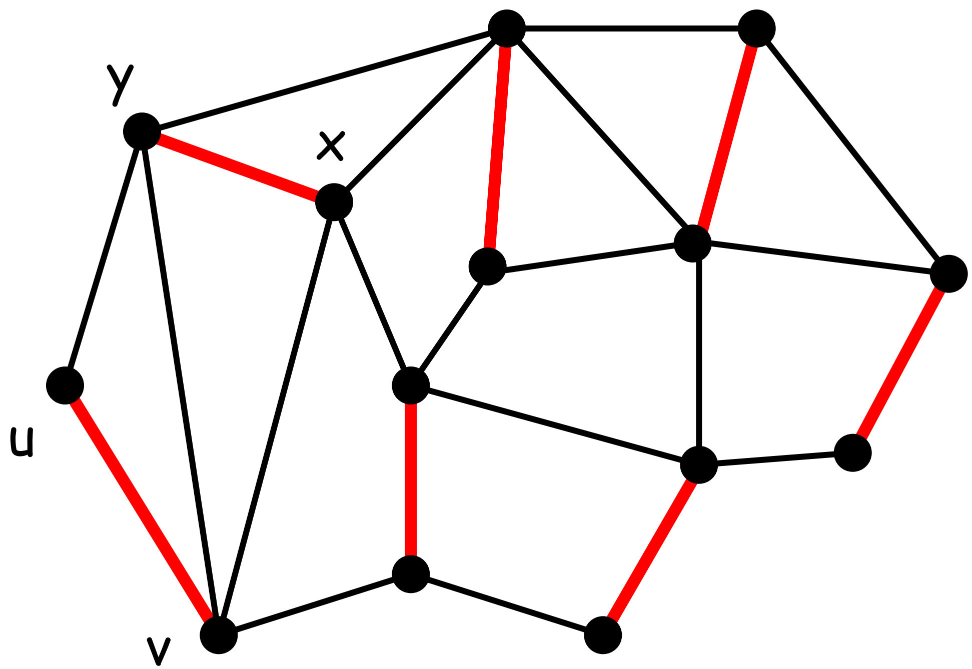


Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

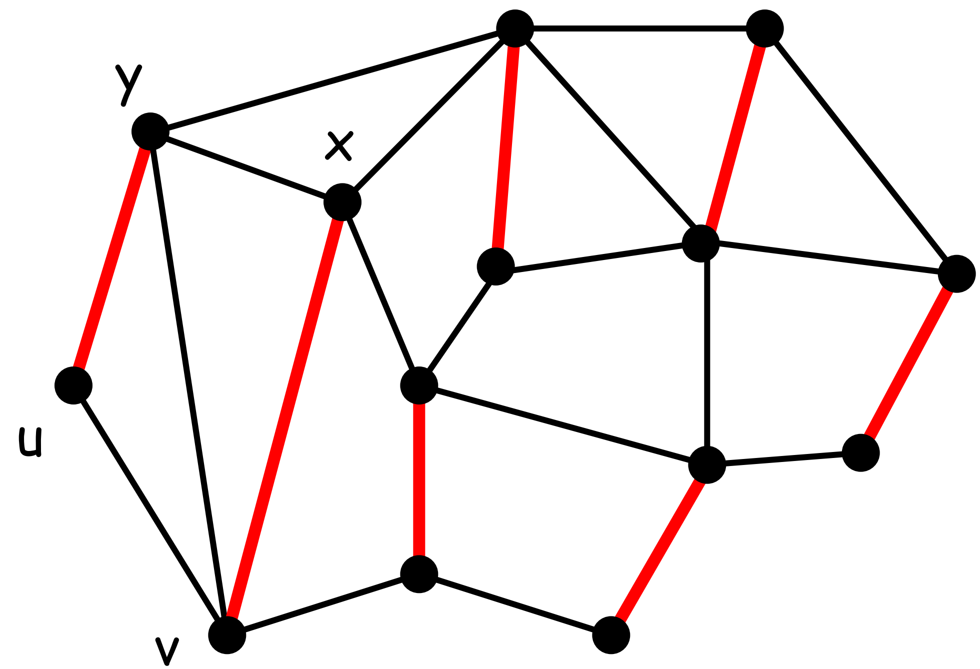


Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

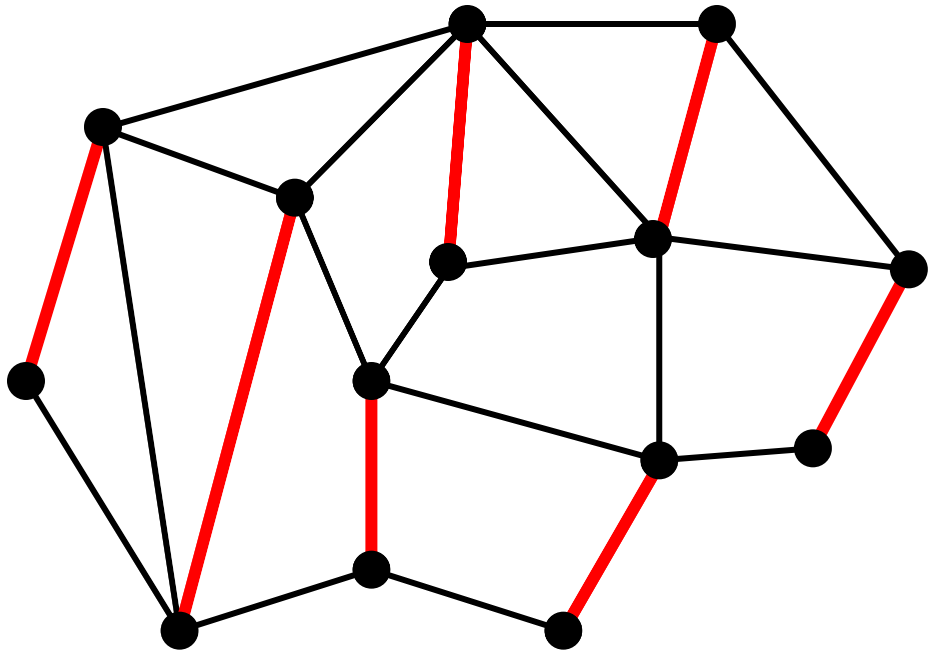


Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

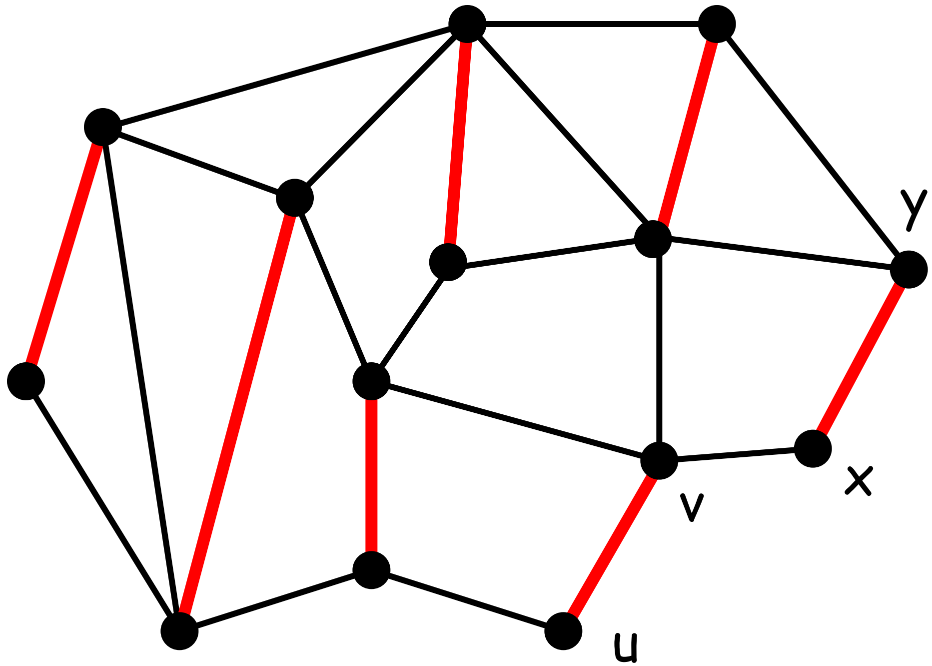


Another MC for Perfect Matchings

<u>Input</u>: a graph G

<u>State space Ω</u>: all perfect matchings of G

<u>Markov chain</u> (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

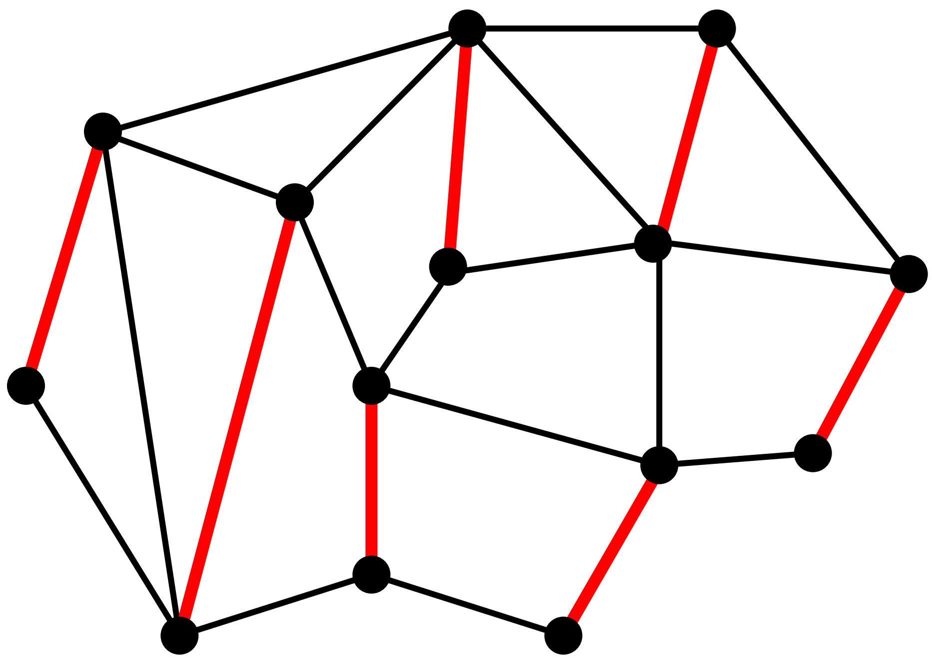


Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

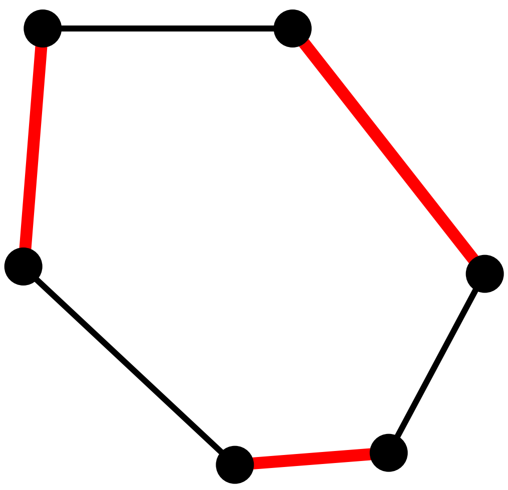


Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.



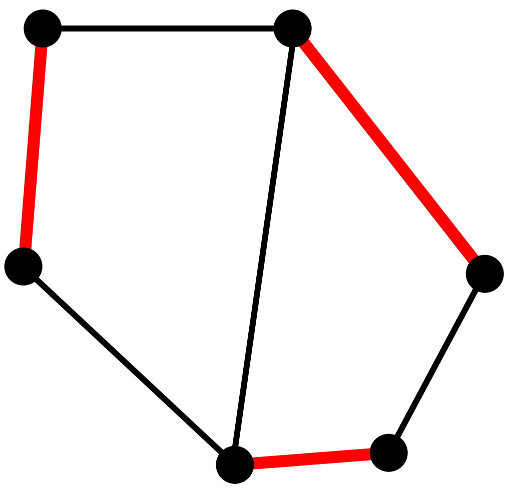
Symmetric but state space disconnected…

Another MC for Perfect Matchings

Input: a graph G

State space Ω: all perfect matchings of G

Markov chain (swap chain):
Let M be the current matching, choose two random edges (u,v) and (x,y) in M, replace them with (u,y) and (v,x) if can.

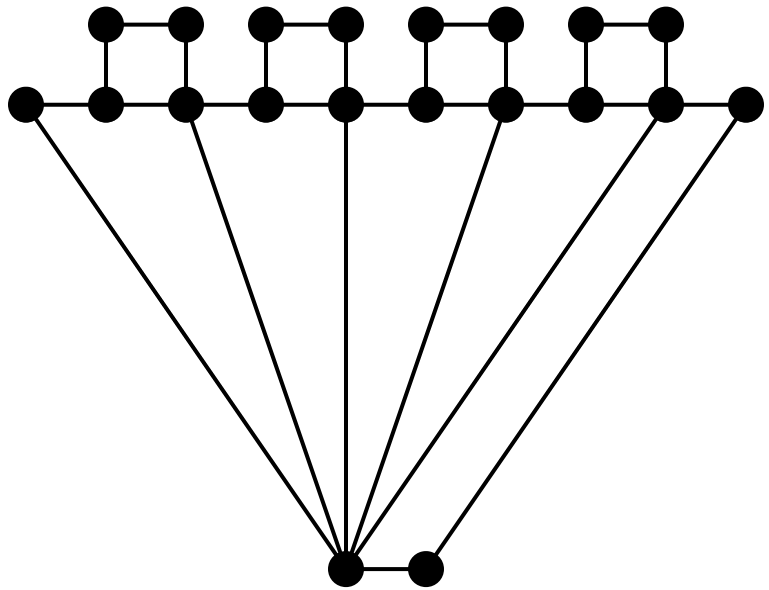


What if have an instance with connected state space:

Does it then mix rapidly?

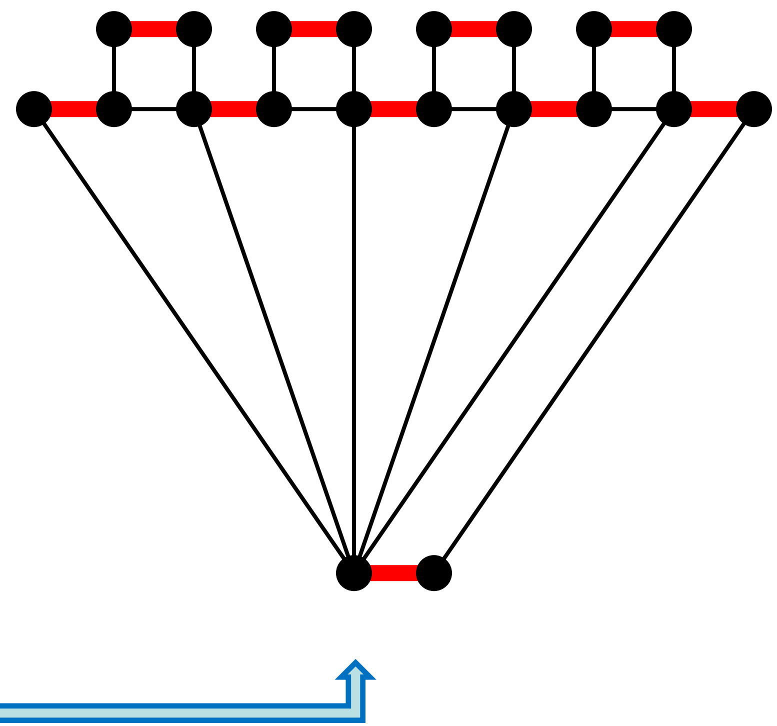
Another MC for Perfect Matchings

Consider this instance (family of instances):



Another MC for Perfect Matchings

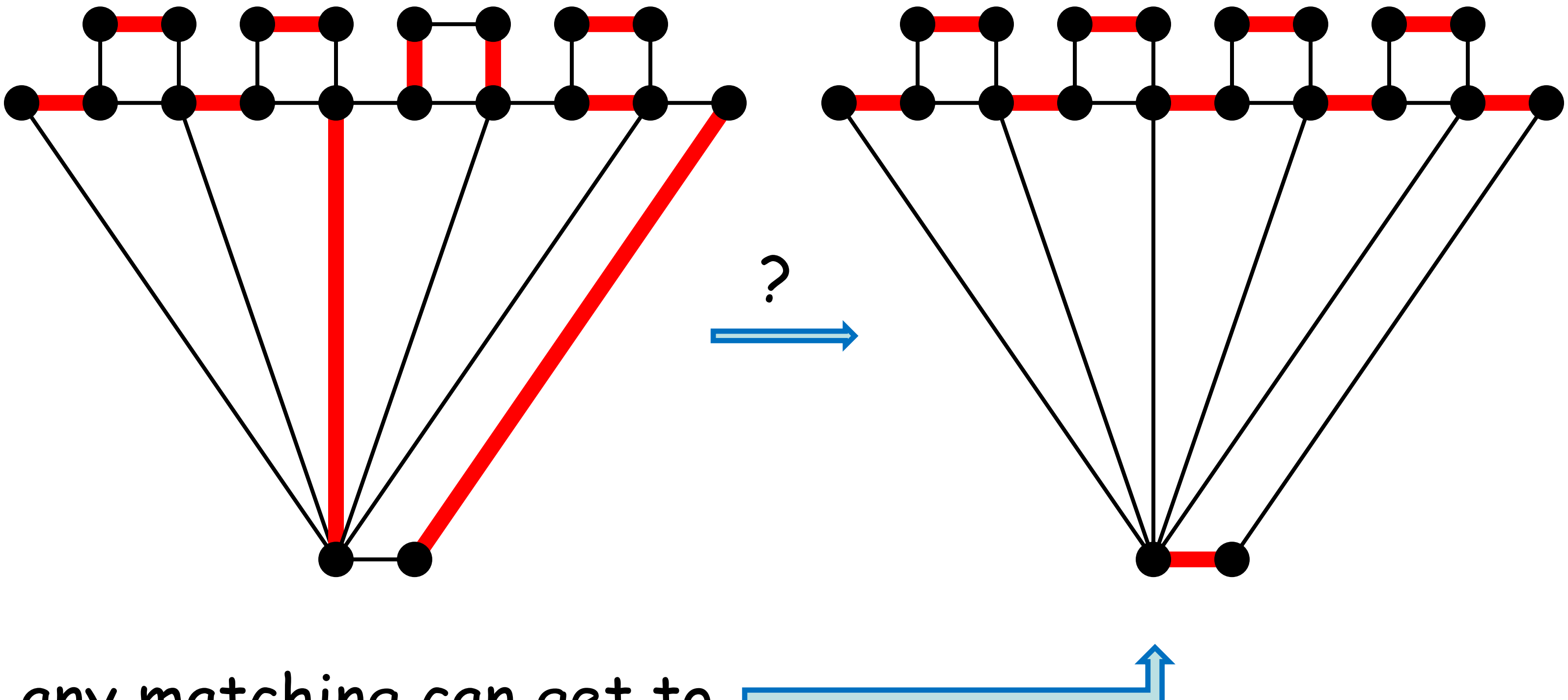
Consider this instance (family of instances):



From any matching can get to

Another MC for Perfect Matchings

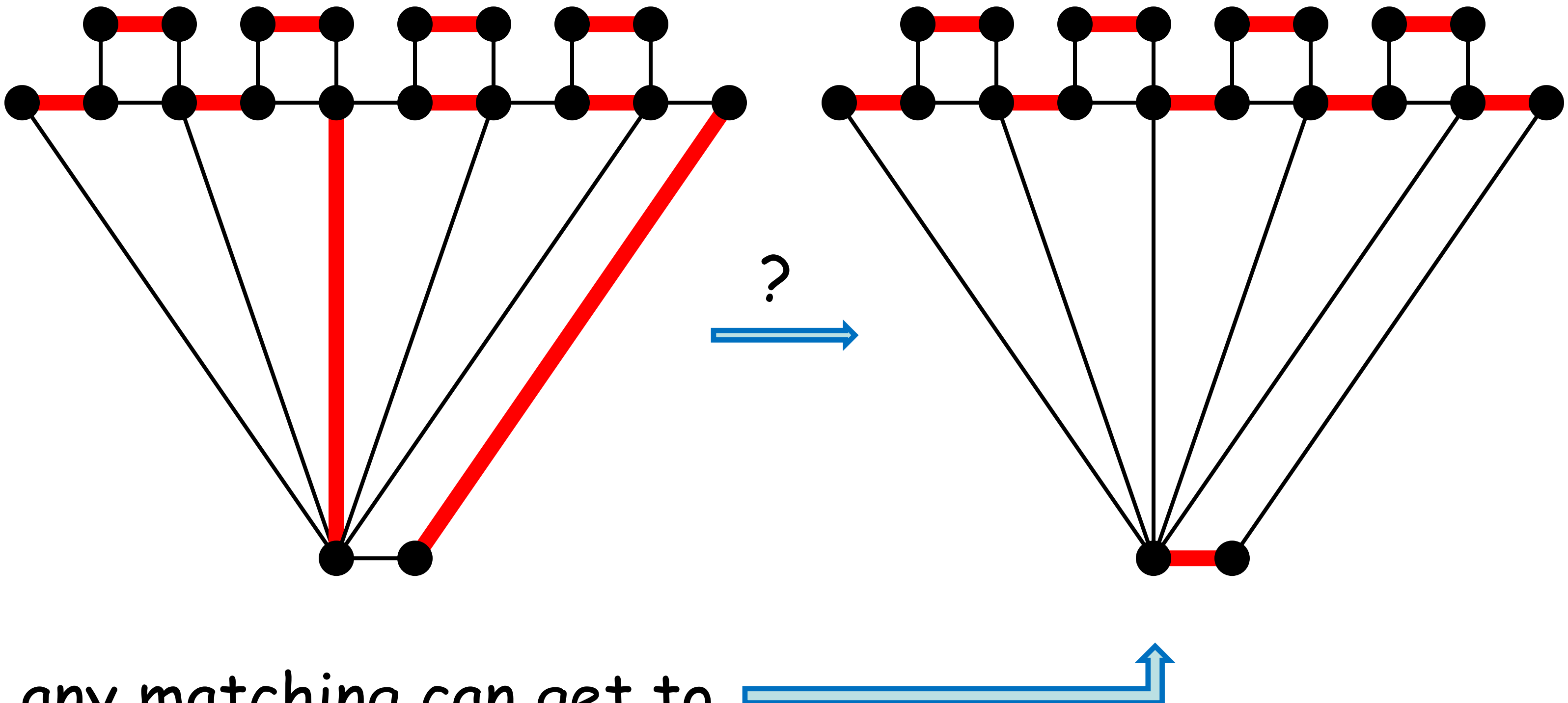
Consider this instance (family of instances):



From any matching can get to

Another MC for Perfect Matchings

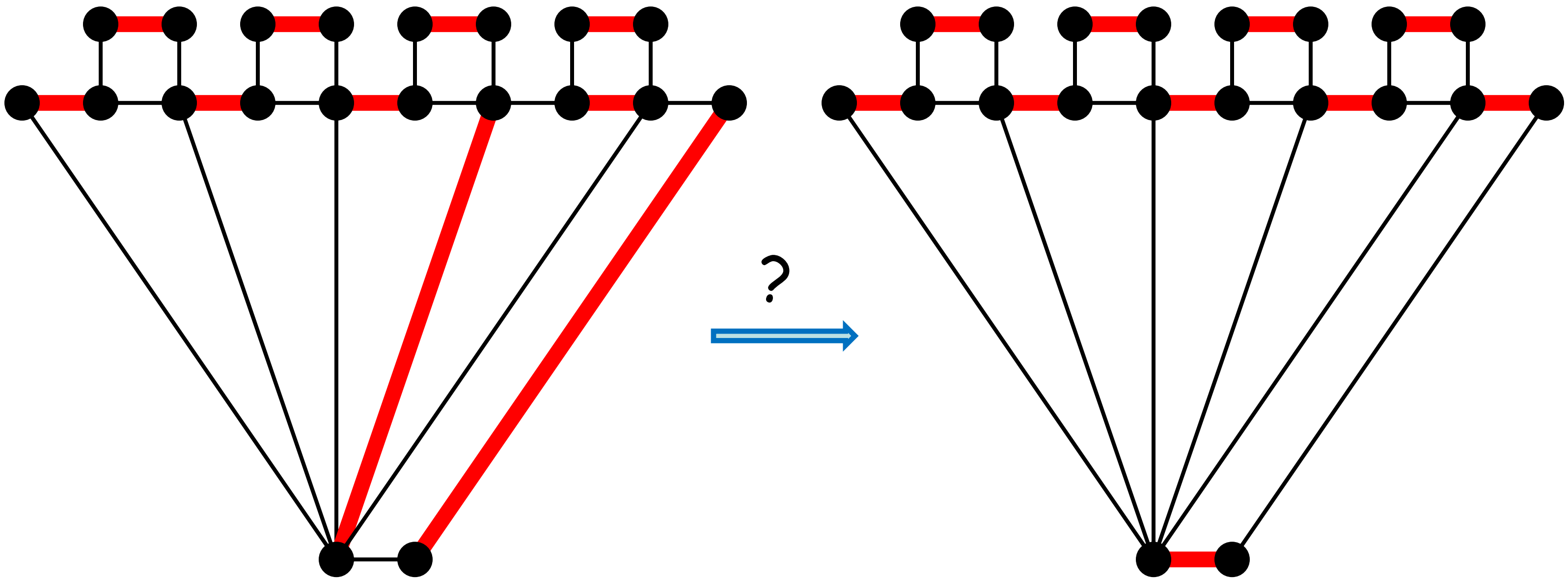
Consider this instance (family of instances):



From any matching can get to

Another MC for Perfect Matchings

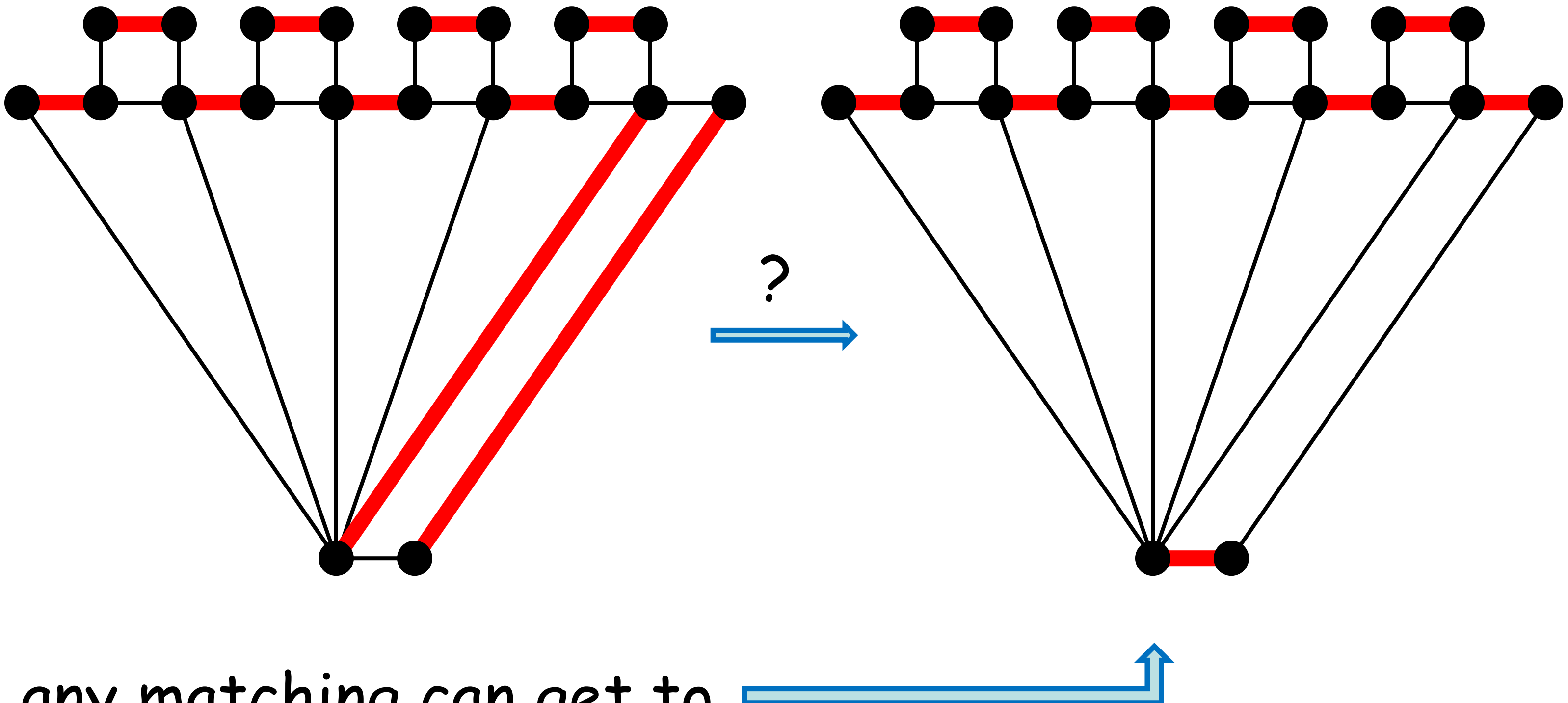
Consider this instance (family of instances):



From any matching can get to

Another MC for Perfect Matchings

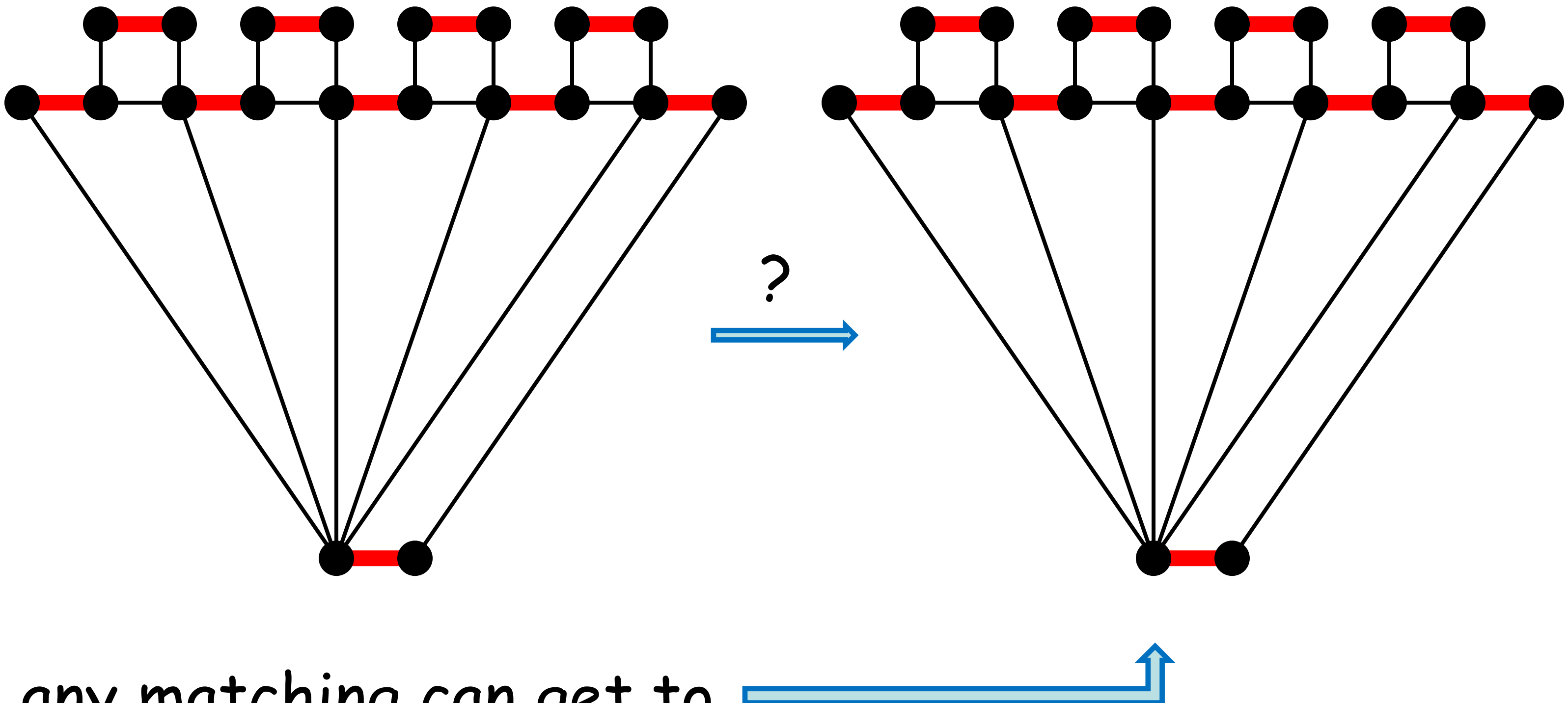
Consider this instance (family of instances):



From any matching can get to

Another MC for Perfect Matchings

Consider this instance (family of instances):

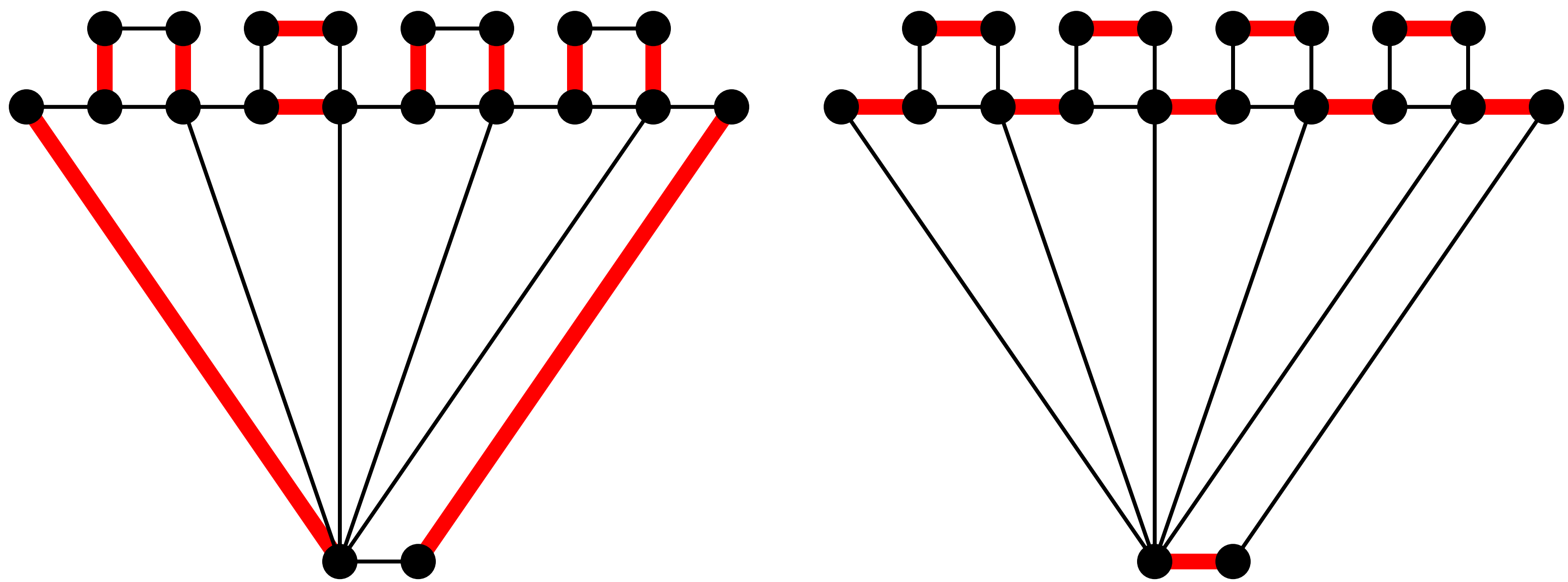


From any matching can get to

-> State space is connected

Another MC for Perfect Matchings

Consider this instance (family of instances):

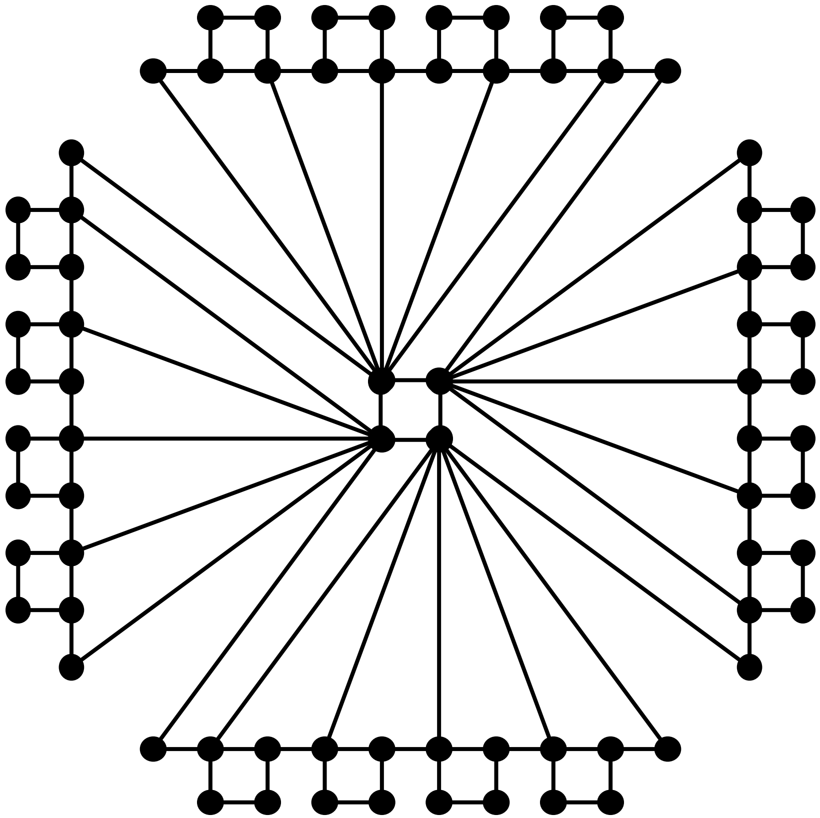


matchings that do not use the bottom edge: $\geq 2^{n/4-1}$

matchings that use the bottom edge: 1

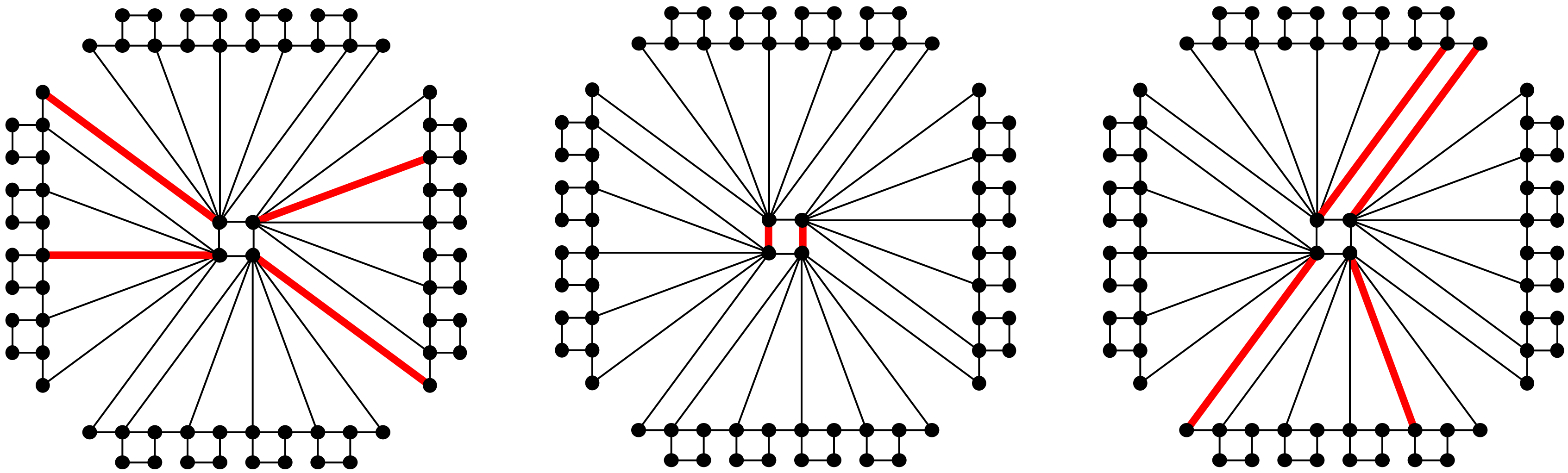
Another MC for Perfect Matchings

Consider this instance (family of instances):



Another MC for Perfect Matchings

Consider this instance (family of instances):

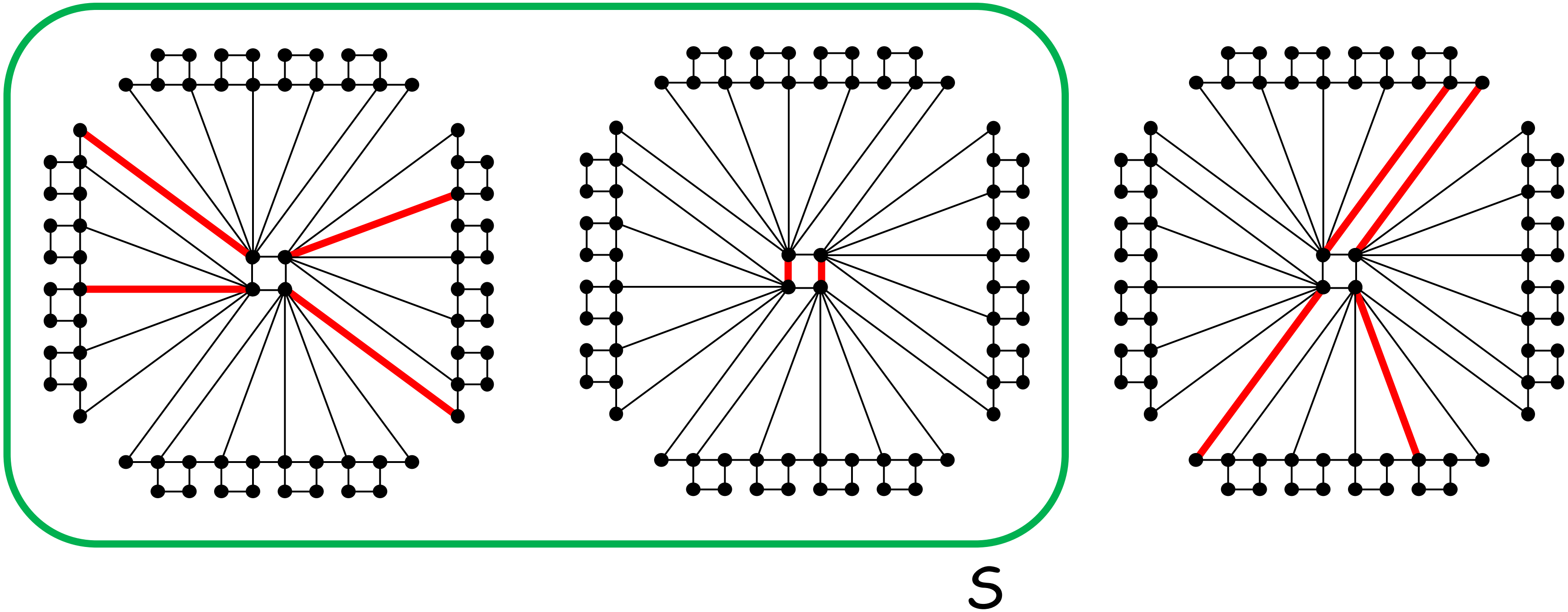


Conductance:

$$\Phi := \min_{S:S\subseteq\Omega,\pi(S)\le 1/2} \frac{\sum_{x\in S, y\notin S} \pi(x)P(x,y)}{\pi(S)} \le \frac{\frac{1}{|\Omega|}\frac{1}{2n(n-1)}}{1/2} \le \frac{1}{2^{n/2-1}n(n-1)}$$

Another MC for Perfect Matchings

Consider this instance (family of instances):

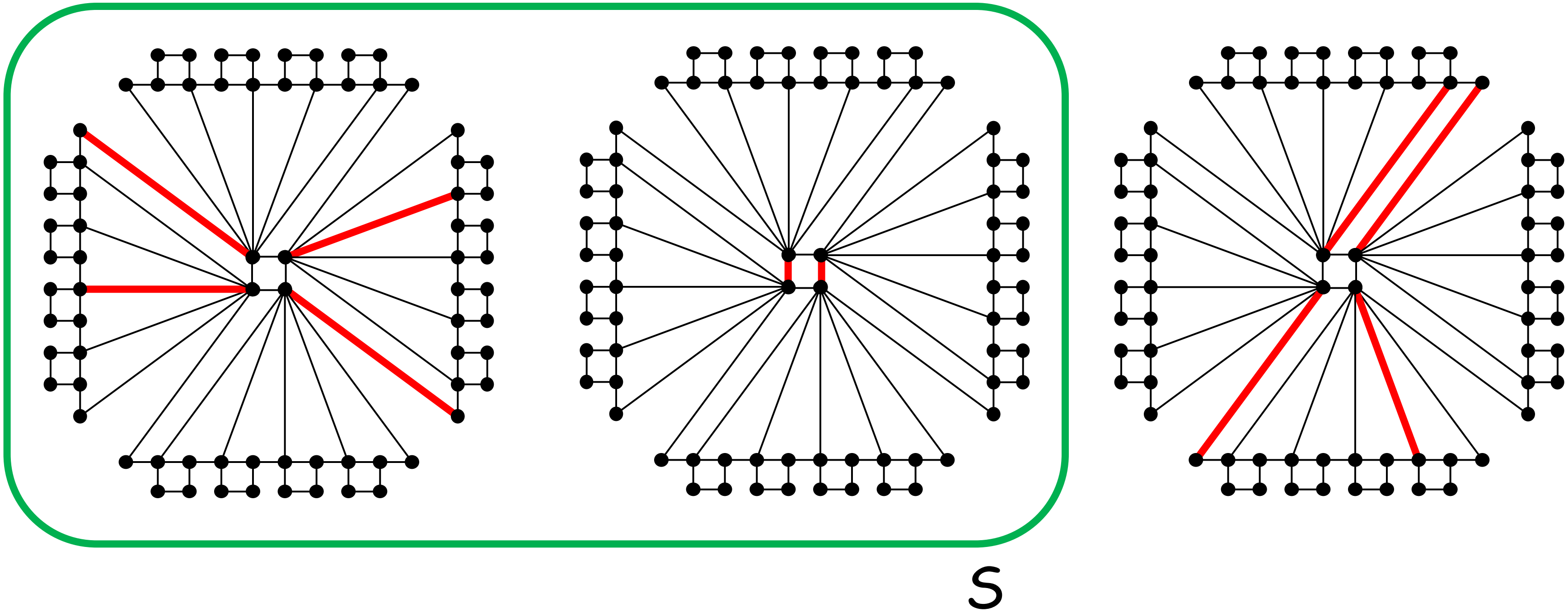


Conductance:

$$\Phi := \min_{S:S\subseteq\Omega,\pi(S)\le 1/2} \frac{\sum_{x\in S,y\notin S}\pi(x)P(x,y)}{\pi(S)} \le \frac{\frac{1}{|\Omega|}\frac{1}{2n(n-1)}}{1/2} \le \frac{1}{2^{n/2-1}n(n-1)}$$

Another MC for Perfect Matchings

Consider this instance (family of instances):

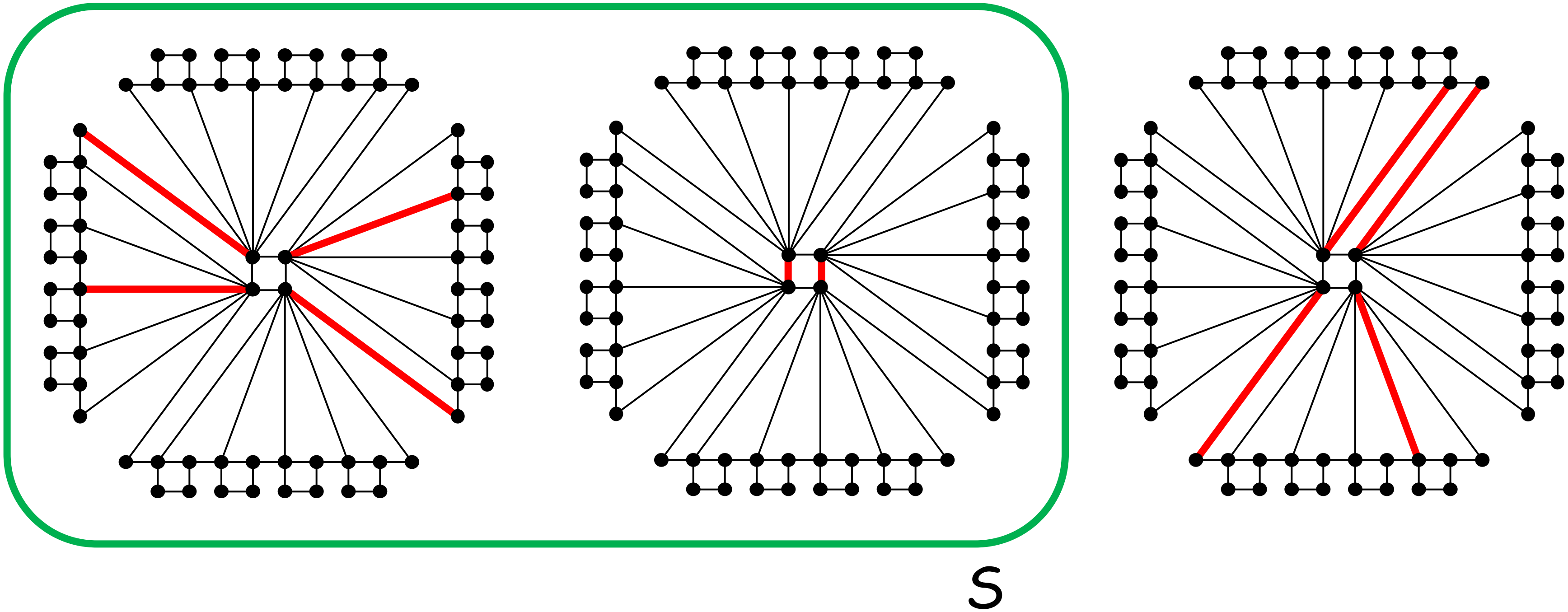


Conductance:

$$t_{mix}(\varepsilon) \geq \frac{1}{2}\left(\frac{1}{2\Phi}-1\right)\log\left(\frac{1}{2\varepsilon}\right) \geq \left(2^{n/2-3}n(n-1)-\frac{1}{2}\right)\log\left(\frac{1}{2\varepsilon}\right)$$

Another MC for Perfect Matchings

Consider this instance (family of instances):



Conductance:

More on this chain:
Dyer-Jerrum-Müller

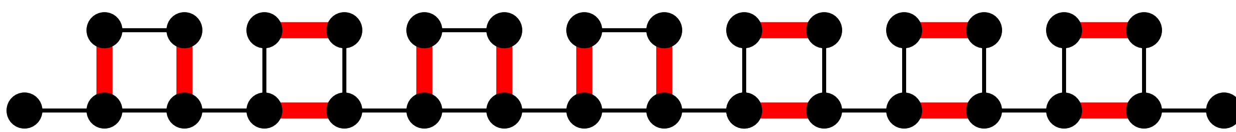
$$t_{mix}(\varepsilon) \geq \frac{1}{2}\left(\frac{1}{2\Phi}-1\right)\log\left(\frac{1}{2\varepsilon}\right) \geq \left(2^{n/2-3}n(n-1)-\frac{1}{2}\right)\log\left(\frac{1}{2\varepsilon}\right)$$

Back to the Sliding Chain: Permanent

<u>What if improve mixing time analysis?</u>

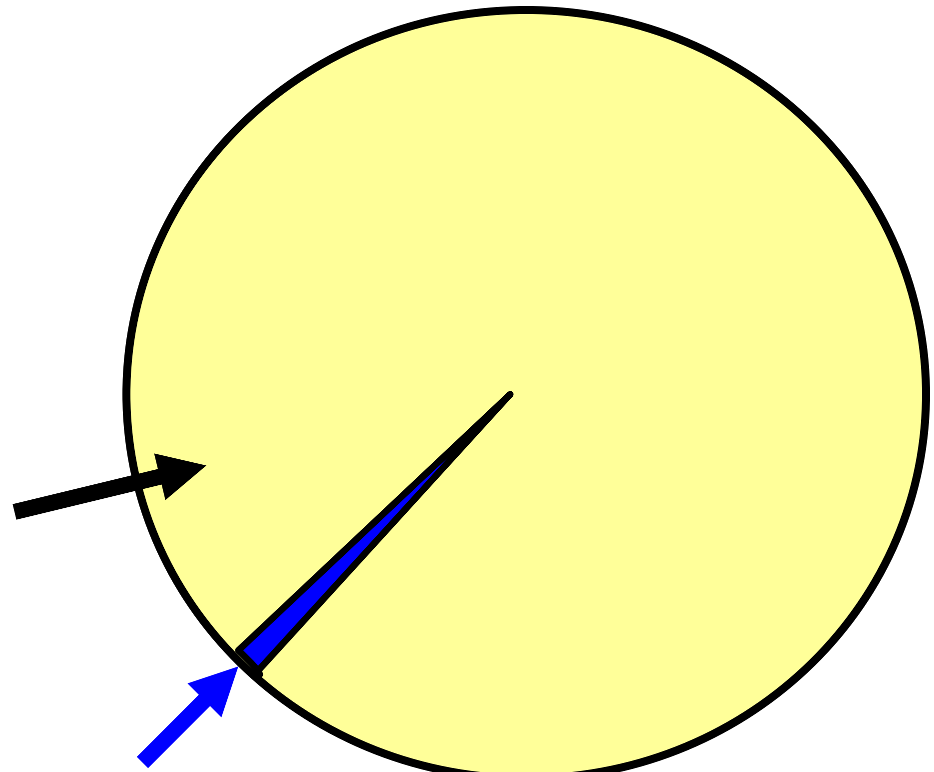
$$\boxed{\mathrm{Per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}}$$

Counts perfect matchings in <u>bipartite</u> graphs



Just one perfect matching…

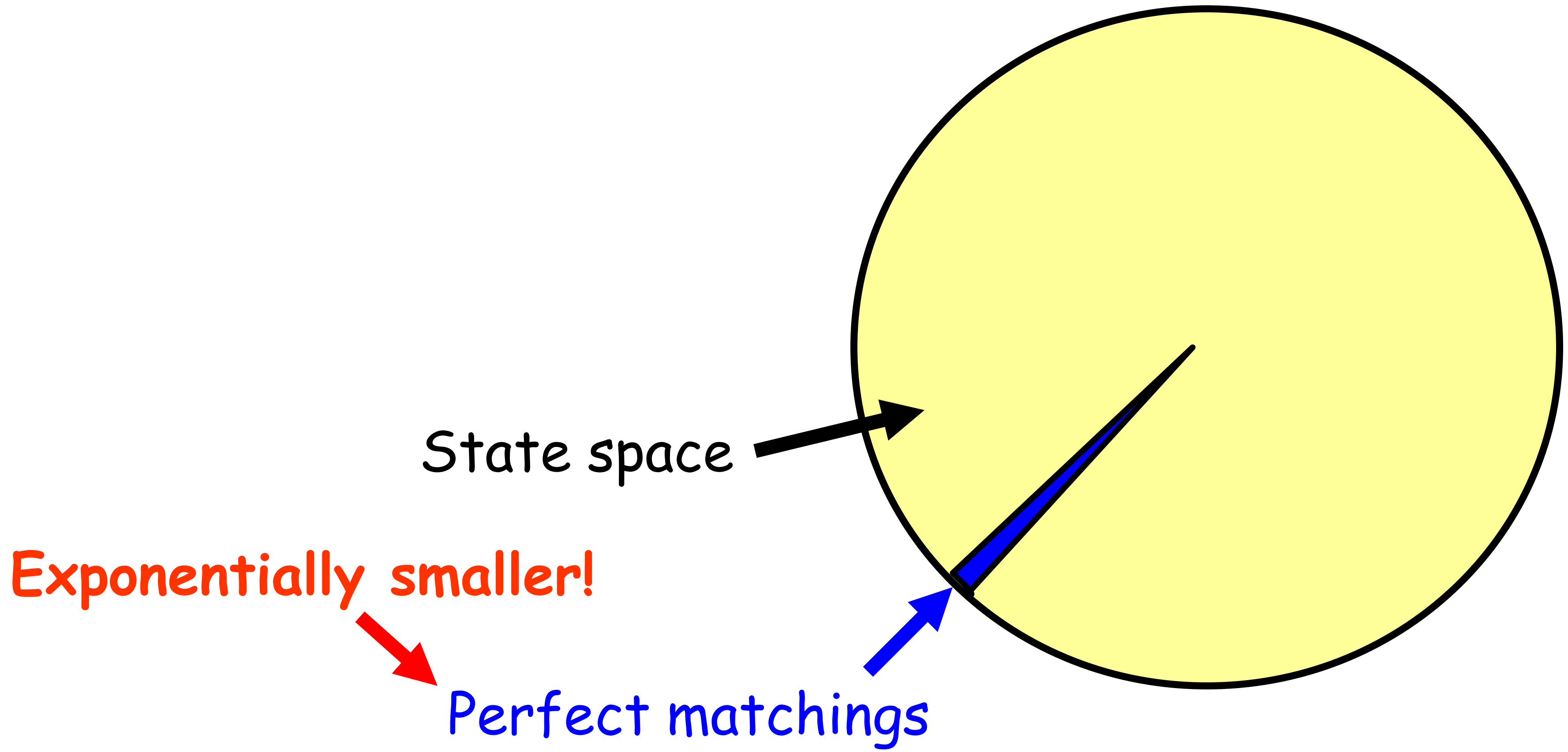
But exponentially many nears!



Back to the Sliding Chain: Permanent

Idea [Jerrum-Sinclair-Vigoda]:

Change the weights of the states
(change stationary distribution).

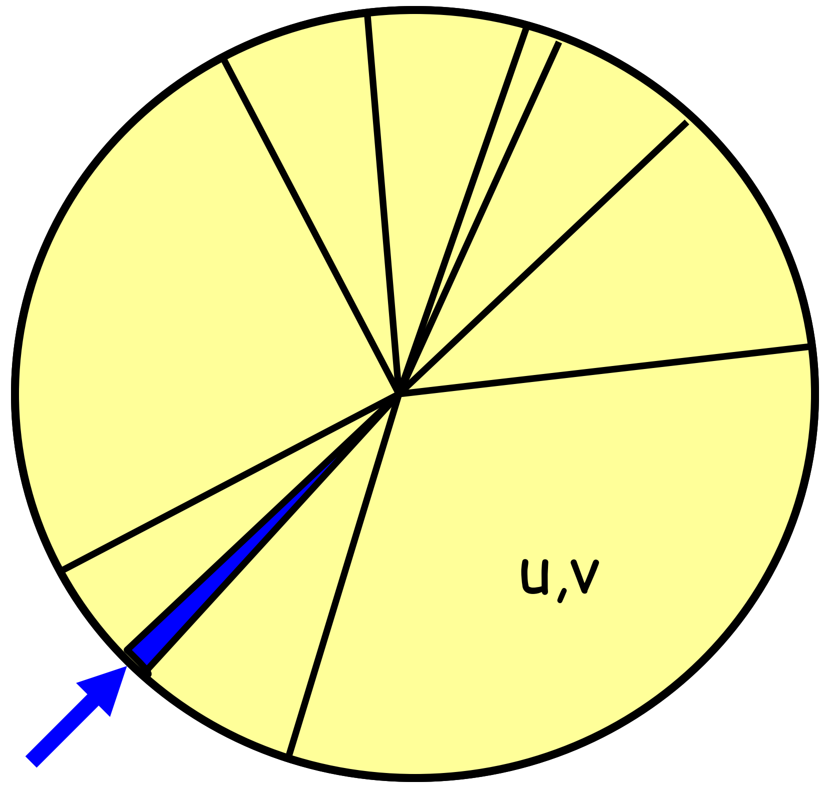


Back to the Sliding Chain: Permanent

Idea [Jerrum-Sinclair-Vigoda]:

Change the weights of the states
(change stationary distribution).

n^2+1 regions,
very different
weight



Exponentially smaller!

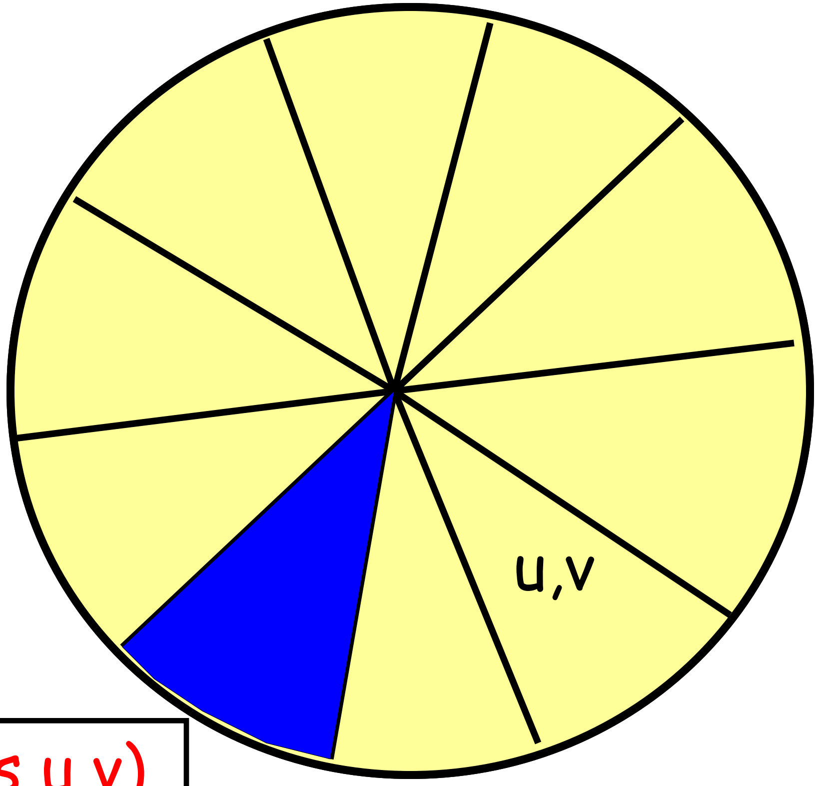
Perfect matchings

Back to the Sliding Chain: Permanent

<u>Idea</u> [Jerrum-Sinclair-Vigoda]:

Change the weights of the states
(change stationary distribution).

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



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

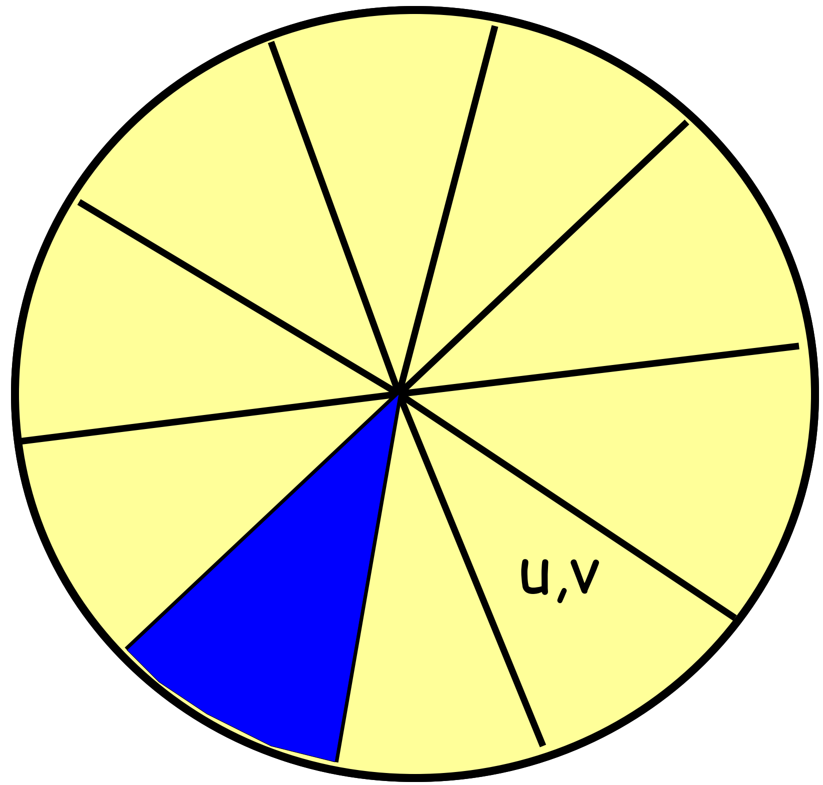
(# perfects) / (# nears with holes u,v)

Back to the Sliding Chain: Permanent

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

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

How to compute ???



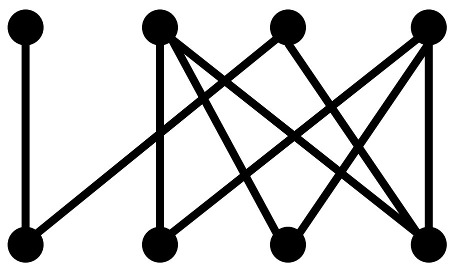
Back to the Sliding Chain: Permanent

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

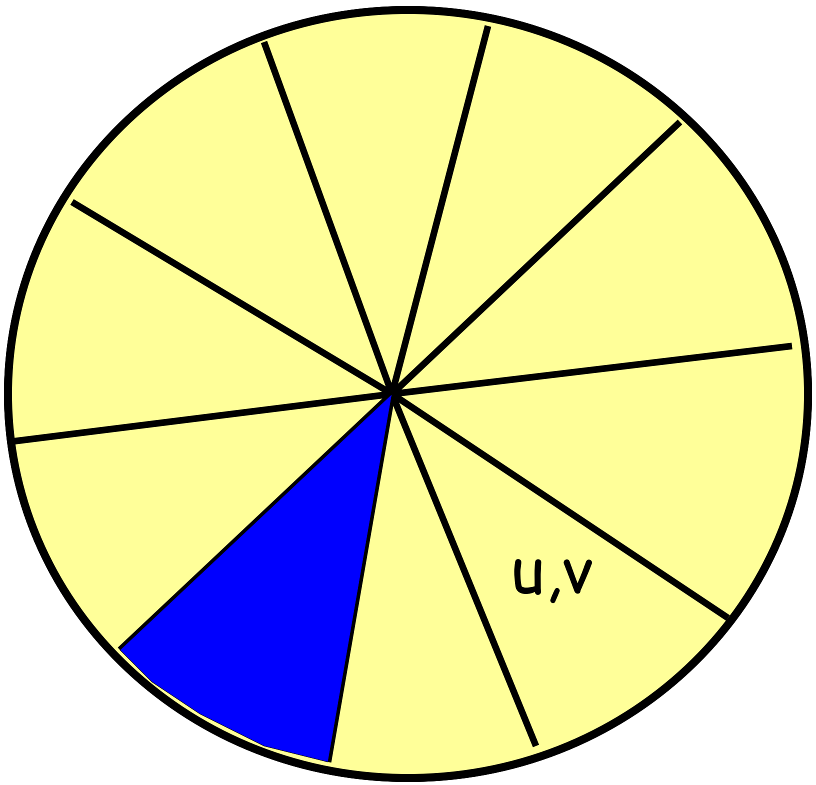
(# perfects) / (# nears with holes u,v)

How to compute ???

Approximate:
start with an easy graph,
gradually get to the target graph



target



Back to the Sliding Chain: Permanent

Ideal weights

(for a matching with holes u,v):

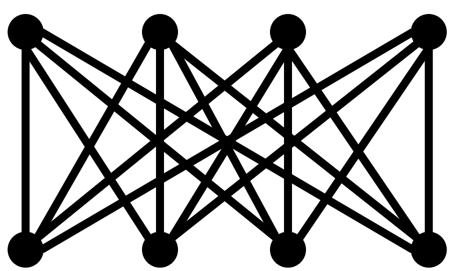
(# perfects) / (# nears with holes u,v)

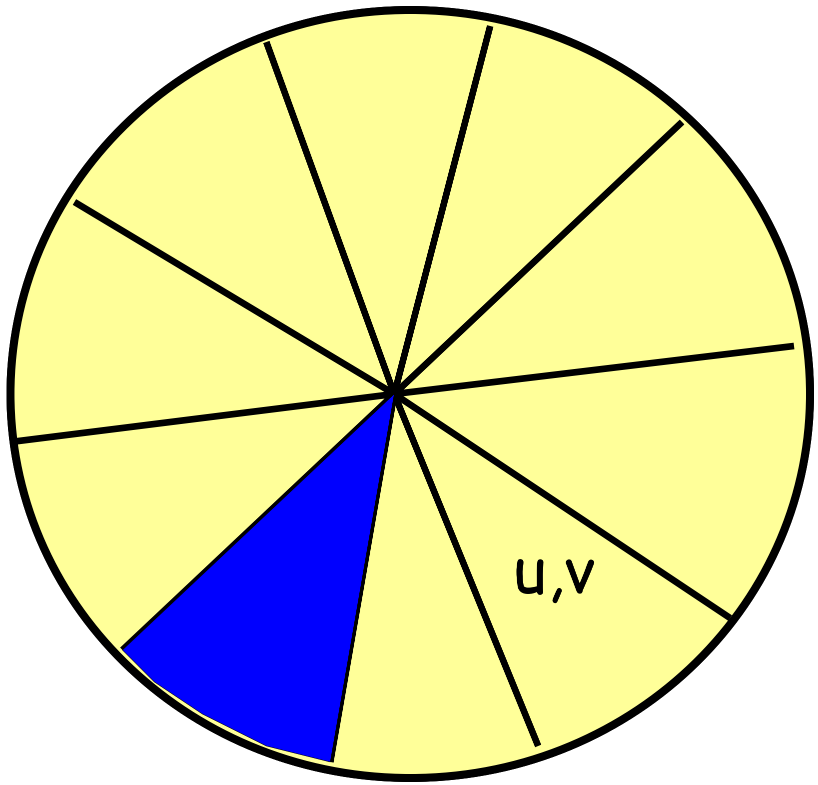
How to compute ???

<u>Approximate:</u>

start with an easy graph,

gradually get to the target graph





Back to the Sliding Chain: Permanent

Ideal weights

(for a matching with holes u,v):

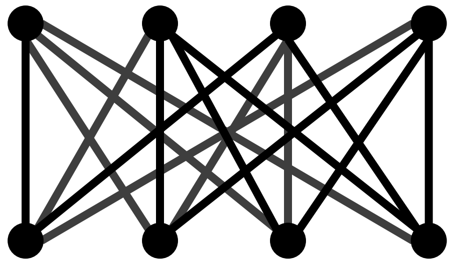
(# perfects) / (# nears with holes u,v)

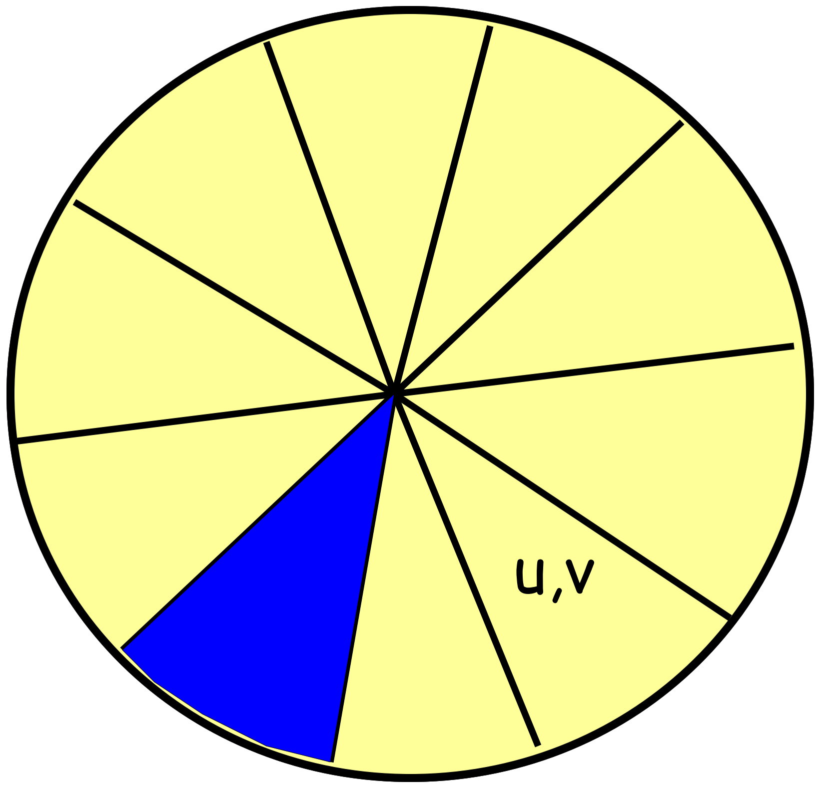
How to compute ???

<u>Approximate:</u>

start with an easy graph,

gradually get to the target graph





Back to the Sliding Chain: Permanent

Ideal weights

(for a matching with holes u,v):

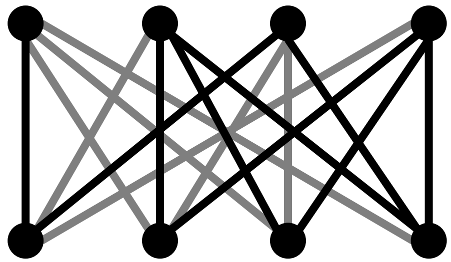
(# perfects) / (# nears with holes u,v)

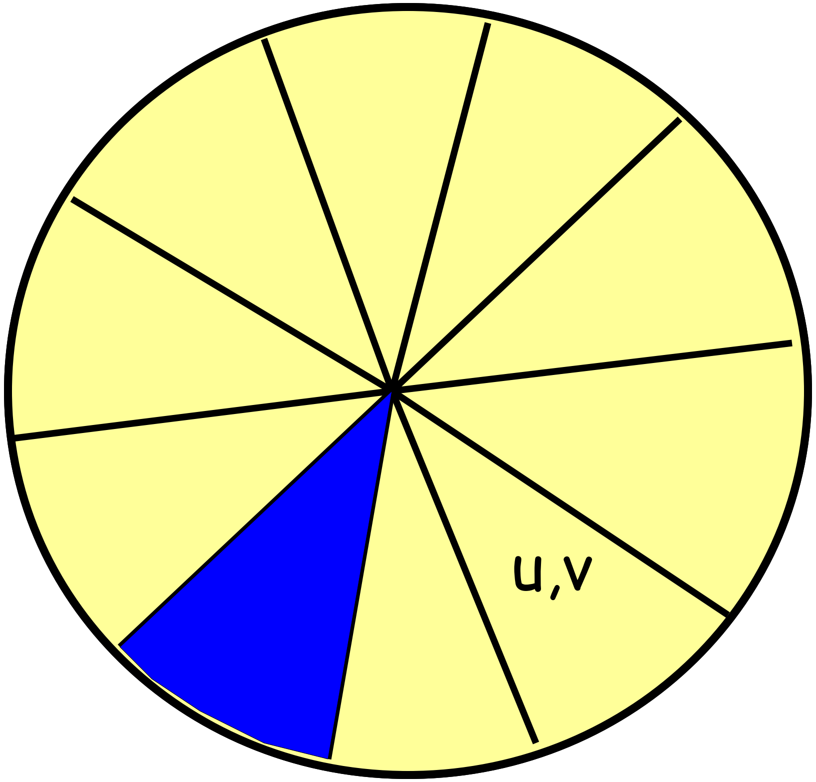
How to compute ???

<u>Approximate:</u>

start with an easy graph,

gradually get to the target graph





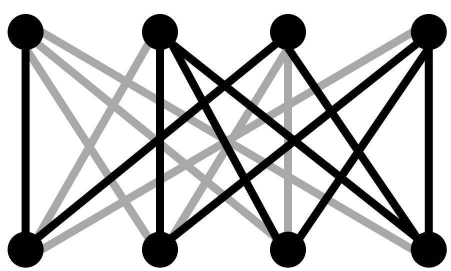
Back to the Sliding Chain: Permanent

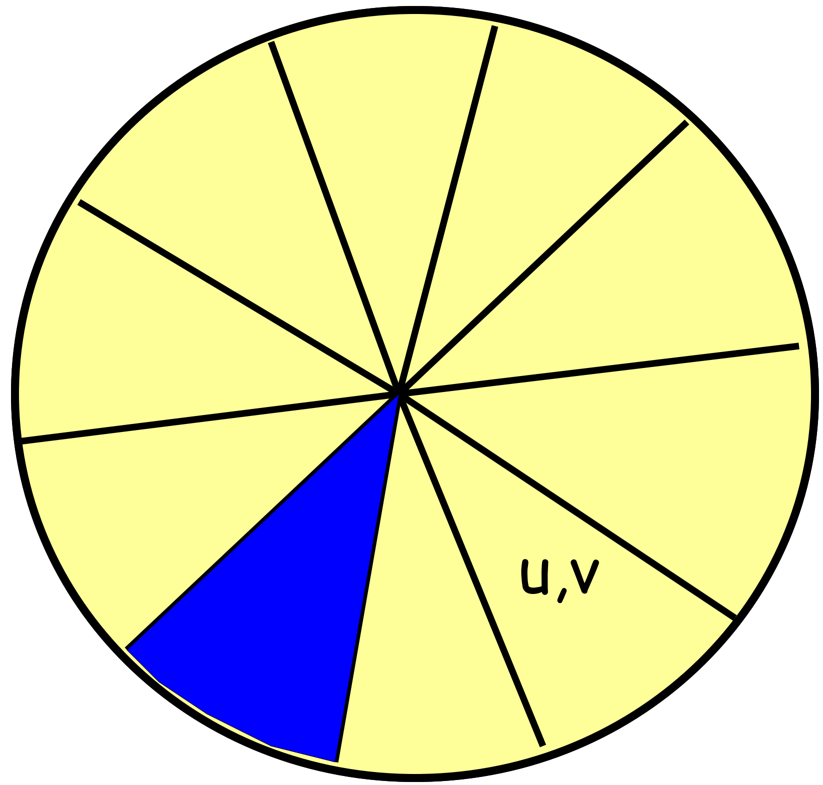
Ideal weights
(for a matching with holes u,v):

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

How to compute ???

<u>Approximate:</u>
start with an easy graph,
gradually get to the target graph





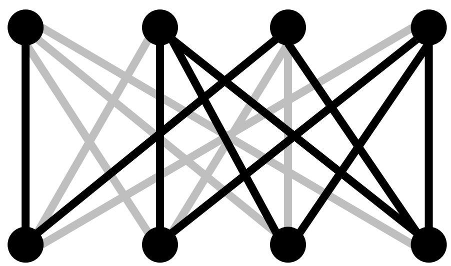
Back to the Sliding Chain: Permanent

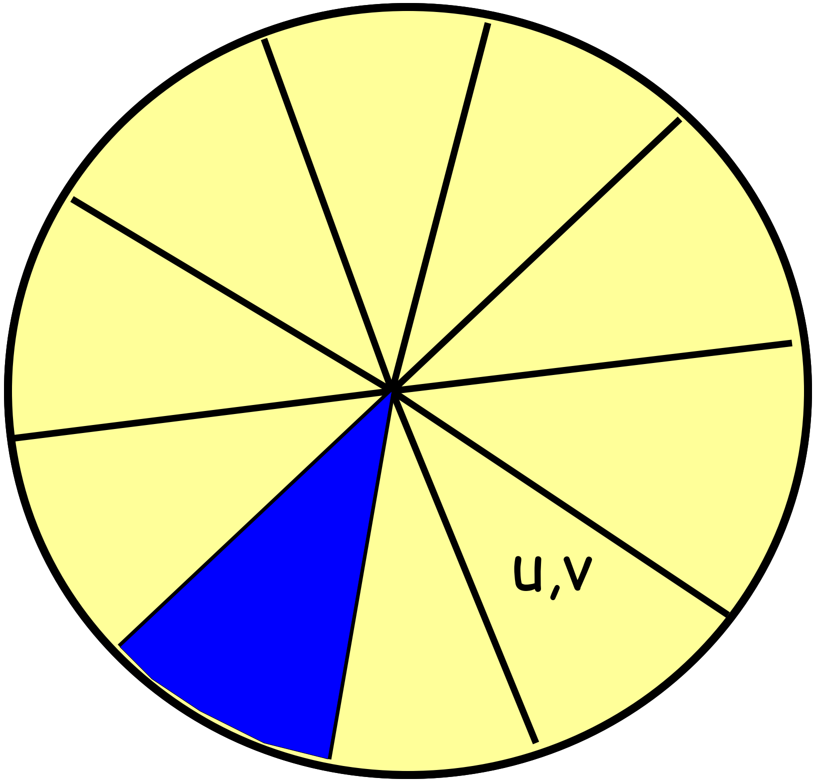
Ideal weights
(for a matching with holes u,v):

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

How to compute ???

<u>Approximate:</u>
start with an easy graph,
gradually get to the target graph





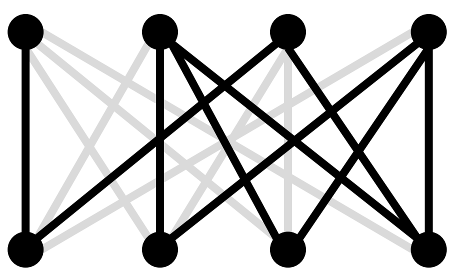
Back to the Sliding Chain: Permanent

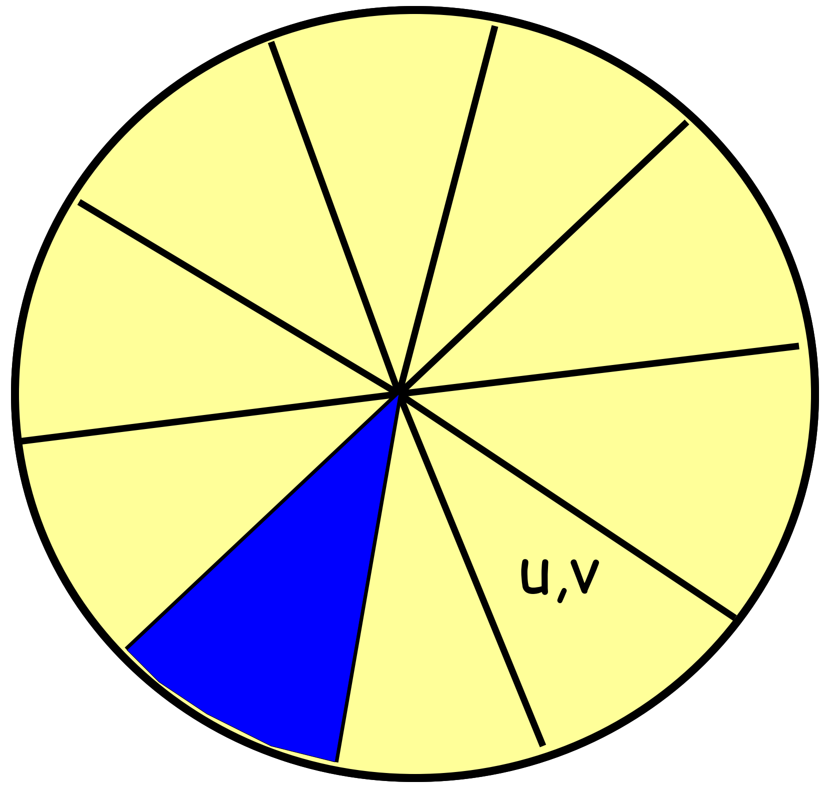
Ideal weights
(for a matching with holes u,v):

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

How to compute ???

<u>Approximate:</u>
start with an easy graph,
gradually get to the target graph





Back to the Sliding Chain: Permanent

Ideal weights

(for a matching with holes u,v):

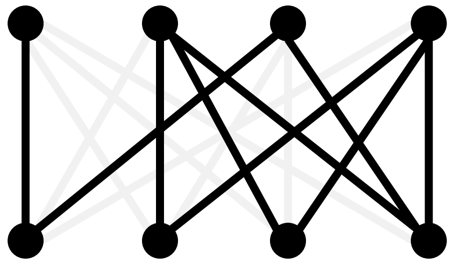
(# perfects) / (# nears with holes u,v)

Edge weights:

- 1 for edge
- λ for non-edge

- Start with λ=1:

#perfect/#nears = n!/(n-1)!



Back to the Sliding Chain: Permanent

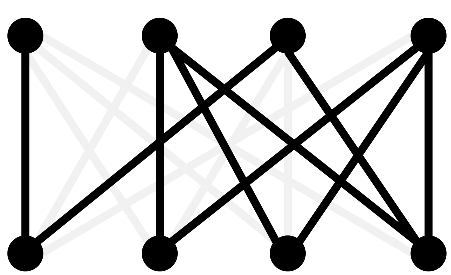
Ideal weights

(for a matching with holes u,v):

λ(perfects) / λ(nears with holes u,v)

Edge weights:

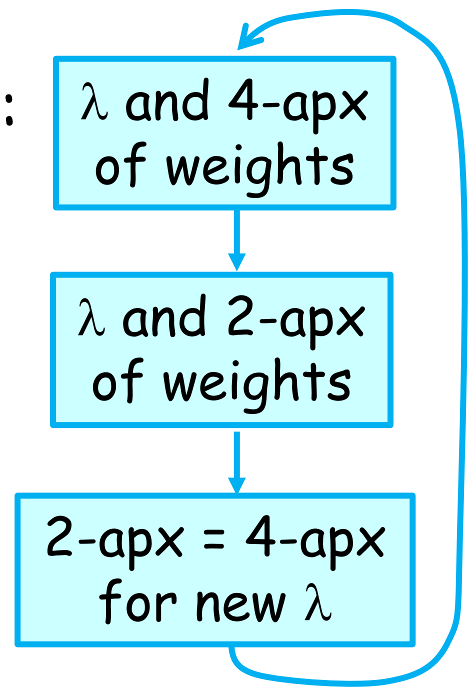
- 1 for edge
- λ for non-edge



- Start with λ=1:

  #perfect/#nears = n!/(n-1)!

- Repeat until λ < 1/n!:



Back to the Sliding Chain: Permanent

Thm [Jerrum-Sinclair-Vigoda]:

FPRAS for the permanent.

OPEN PROBLEM:

counting perfect matchings in non-bipartite graphs

$\boxed{\lambda(\text{perfects}) \,/\, \lambda(\text{nears with holes u,v})}$