The Matching Problem is in Quasi-NC

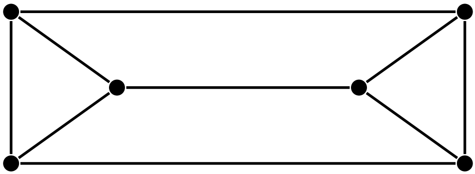
Ola Svensson and Jakub Tarnawski

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Institute for Advanced Study, 22.01.2018

Perfect matching problem

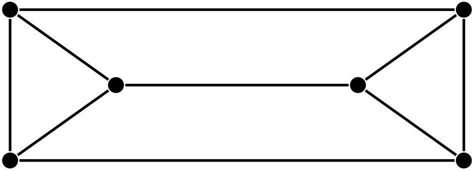
Given a graph, can we pair up all vertices using edges?



Perfect matching problem

Given a graph, can we pair up all vertices using edges?

very tough instance:
graph is non-bipartite!



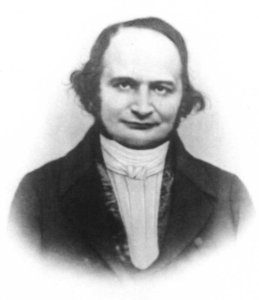
Perfect matching problem

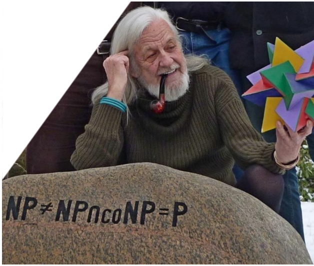
Given a graph, can we pair up all vertices using edges?

very tough instance:
graph is non-bipartite!

Perfect matching problem

Benchmark problem in computer science



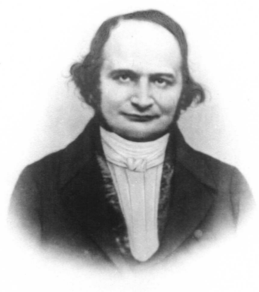


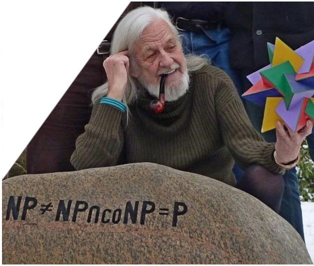
Perfect matching problem

Benchmark problem in computer science

Algorithms:

- bipartite: Jacobi [XIX century, weighted!]
- general: Edmonds [1965]
  - polynomial-time = efficient
- since then, tons of research
  and still active
- many models of computation:
  monotone circuits, extended formulations,
  parallel, streaming/sublinear, ...



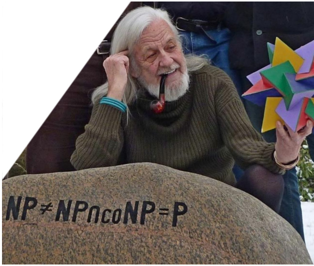


Perfect matching problem

Benchmark problem in computer science

Algorithms:

- bipartite: Jacobi [XIX century, weighted!]
- general: Edmonds [1965]
  - polynomial-time = efficient
- since then, tons of research
  and still active
- many models of computation:
  monotone circuits, extended formulations,
  parallel, streaming/sublinear, ...



Parallel complexity

Class $\mathcal{NC}$: problems that parallelize completely

poly n processors

poly log n time

Parallel complexity

Class $\mathcal{NC}$: problems that parallelize completely

poly n processors

poly log n time

Main open question: is matching in $\mathcal{NC}$?

Parallel complexity

Class $\mathcal{NC}$: problems that parallelize completely

poly n processors

It's in Randomized $\mathcal{NC}$

poly log n time

Main open question: is matching in $\mathcal{NC}$?

Parallel complexity

Matching is in RANDOMIZED $\mathcal{NC}$ [Lovász 1979]:
has randomized algorithm that uses:

- polynomially many processors
- polylog time

Parallel complexity

- Matching is in RANDOMIZED $\mathcal{NC}$ [Lovász 1979]:
  has randomized algorithm that uses:
  - polynomially many processors
  - polylog time
- Search version in RANDOMIZED $\mathcal{NC}$:
  - [Karp, Upfal, Wigderson 1986]
  - [Mulmuley, Vazirani, Vazirani 1987]

Parallel complexity

- Matching is in RANDOMIZED $\mathcal{NC}$ [Lovász 1979]:
  has randomized algorithm that uses:
  - polynomially many processors
  - polylog time
- Search version in RANDOMIZED $\mathcal{NC}$:
  - [Karp, Upfal, Wigderson 1986]
  - [Mulmuley, Vazirani, Vazirani 1987]

Can we derandomize efficient computation?

Parallel complexity

- Matching is in RANDOMIZED $\mathcal{NC}$ [Lovász 1979]:
  has randomized algorithm that uses:
  - polynomially many processors
  - polylog time
- Search version in RANDOMIZED $\mathcal{NC}$:
  - [Karp, Upfal, Wigderson 1986]
  - [Mulmuley, Vazirani, Vazirani 1987]

Can we derandomize efficient computation?

Can we derandomize one of these algorithms?

Is matching in $\mathcal{NC}$?

Is matching in $\mathcal{NC}$?

Yes, for restricted graph classes:

- bipartite regular [Lev, Pippenger, Valiant 1981]
- bipartite convex [Dekel, Sahni 1984]
- incomparability graphs [Kozen, Vazirani, Vazirani 1985]
- bipartite graphs with small number of perfect matchings [Grigoriev, Karpinski 1987]
- claw-free [Chrobak, Naor, Novick 1989]
- $K_{3,3}$-free (decision version) [Vazirani 1989]
- planar bipartite [Miller, Naor 1989]
- dense [Dahlhaus, Hajnal, Karpinski 1993]
- strongly chordal [Dahlhaus, Karpinski 1998]
- P_4-tidy [Parfenoff 1998]
- bipartite small genus [Mahajan, Varadarajan 2000]
- graphs with small number of perfect matchings [Agrawal, Hoang, Thierauf 2006]
- planar (search version) [Anari, Vazirani 2017]

Is matching in $\mathcal{NC}$?

Yes, for restricted graph classes:

- bipartite regular [Lev, Pippenger, Valiant 1981]
- bipartite convex [Dekel, Sahni 1984]
- incomparability graphs [Kozen, Vazirani, Vazirani 1985]
- bipartite graphs with small number of perfect matchings [Grigoriev, Karpinski 1987]
- claw-free [Chrobak, Naor, Novick 1989]
- $K_{3,3}$-free (decision version) [Vazirani 1989]
- planar bipartite [Miller, Naor 1989]
- dense [Dahlhaus, Hajnal, Karpinski 1993]
- strongly chordal [Dahlhaus, Karpinski 1998]
- P_4-tidy [Parfenoff 1998]
- bipartite small genus [Mahajan, Varadarajan 2000]
- graphs with small number of perfect matchings [Agrawal, Hoang, Thierauf 2006]
- planar (search version) [Anari, Vazirani 2017]

but not known for:

- **bipartite**

Theorem Fenner, Gurjar and Thierauf [2015]

Bipartite matching is in $\textsc{quasi-}\mathcal{NC}$

($n^{\text{poly} \log n}$ processors, poly log n time, deterministic)



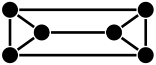
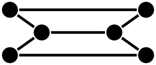
Theorem Fenner, Gurjar and Thierauf [2015]

Bipartite matching is in QUASI-$\mathcal{NC}$

($n^{\text{poly} \log n}$ processors, poly log n time, deterministic)



- Approach fails for non-bipartite graphs


much harder than


Theorem S. and Tarnawski [2017]

General matching is in $\textsc{quasi-}\mathcal{NC}$

($n^{\text{poly} \log n}$ processors, poly log n time, deterministic)

Theorem S. and Tarnawski [2017]

General matching is in $\textsc{quasi}\text{-}\mathcal{NC}$

($n^{\text{poly}\log n}$ processors, poly log n time, deterministic)

with quasi-polynomial
processors

Outline

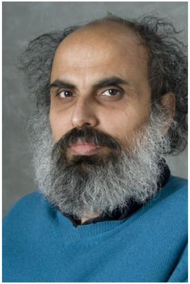
1. Basic approach for derandomization
2. Bipartite case [Fenner, Gurjar, Thierauf 2015]
3. Difficulties of general case & our approach

Basic approach for derandomization

Basic approach for derandomization

(Derandomize one of the randomized algorithms)

Algorithm of Mulmuley, Vazirani, Vazirani'87



Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random
2. Calculate determinant of Tutte matrix where X_e is replaced by $2^{w(e)}$

Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random
2. Calculate determinant of Tutte matrix where X_e is replaced by $\mathbf{2^{w(e)}}$

Important that w is polynomially bounded

Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random 
2. Calculate determinant of Tutte matrix where X_e is replaced by $\mathbf{2^{w(e)}}$

Important that w is polynomially bounded

Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random 
2. Calculate determinant of Tutte matrix where X_e is replaced by $\mathbf{2^{w(e)}}$

Important that w is polynomially bounded

Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random 
2. Calculate determinant of Tutte matrix where X_e is replaced by $\mathbf{2^{w(e)}}$

Important that w is polynomially bounded

Step 2 guaranteed to work if weight function w is **isolating**: unique min-weight matching

Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random 
2. Calculate determinant of Tutte matrix where X_e is replaced by $\mathbf{2^{w(e)}}$

Important that w is polynomially bounded

random sampling (Step 1)

Isolation Lemma:

Pr[w isolating] ≥ 0.9

Step 2 guaranteed to work if weight function w is **isolating**: unique min-weight matching

Algorithm of Mulmuley, Vazirani, Vazirani'87

Algorithm

1. For each edge e select weight $w(e) \in \{1, 2, \ldots, n^2\}$ at random 
2. Calculate determinant of Tutte matrix where X_e is replaced by $\mathbf{2^{w(e)}}$

Important that w is polynomially bounded

random sampling (Step 1)

Isolation Lemma:

Pr[w isolating] ≥ 0.9

something deterministic?

Construct isolating w in $\mathcal{NC}$?

Step 2 guaranteed to work if weight function w is **isolating**: unique min-weight matching

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \text{poly}(n)$

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \text{poly}(n)$

The oblivious algorithm simply checks all weight functions in parallel

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. ~~For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$~~
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \text{poly}(n)$

The oblivious algorithm simply checks all weight functions in parallel

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. ~~For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$~~
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \text{poly}(n)$

The oblivious algorithm simply checks all weight functions in parallel

Easy even with $|\mathcal{W}^*| \leq 1$

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$
3. ~~The number of weight functions are polynomial~~ $\cancel{|\mathcal{W}^*| \leq \text{poly}(n)}$

The oblivious algorithm simply checks all weight functions in parallel

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \text{poly}(n)$
3. ~~The number of weight functions are polynomial~~ $\cancel{|\mathcal{W}^*| \leq \text{poly}(n)}$

The oblivious algorithm simply checks all weight functions in parallel

Easy, but best known bound on $|\mathcal{W}^*|$ is exponential in n

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \mathbf{n}^{\mathsf{poly}(\log \mathbf{n})}$
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \mathbf{n}^{\mathsf{poly}(\log \mathbf{n})}$

The oblivious algorithm simply checks all weight functions in parallel

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \mathbf{n}^{\mathsf{poly}(\log \mathbf{n})}$
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \mathbf{n}^{\mathsf{poly}(\log \mathbf{n})}$

The oblivious algorithm simply checks all weight functions in parallel

Thm[FGT'15]: $\mathcal{W}^*$ exists for bipartite graphs

Oblivious derandomization

Challenge: On input G, construct an isolating weight function in $\mathcal{NC}$

Oblivious challenge: On input n, construct a family $\mathcal{W}^*$ of weight functions that can be computed in $\mathcal{NC}$ such that

1. For any n-vertex graph, there is an isolating $w \in \mathcal{W}^*$
2. For $w \in \mathcal{W}^*$ and edge e, we have $w(e) \leq \mathbf{n}^{\mathbf{poly}(\log \mathbf{n})}$
3. The number of weight functions are polynomial $|\mathcal{W}^*| \leq \mathbf{n}^{\mathbf{poly}(\log \mathbf{n})}$

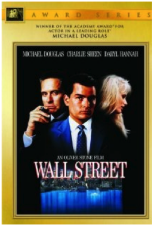
The oblivious algorithm simply checks all weight functions in parallel

Thm[FGT'15]: $\mathcal{W}^*$ exists for bipartite graphs

Thm[ST'17]: $\mathcal{W}^*$ exists for general graphs

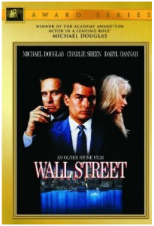
Bipartite case

[Fenner, Gurjar, Thierauf 2015]



Bipartite case

[Fenner, Gurjar, Thierauf 2015]



> "Greed is good. Greed is right. Greed works. Greed clarifies, cuts through and captures the essence of the evolutionary spirit."
>
> \- Gordon Gecko

Bipartite case

[Fenner, Gurjar, Thierauf 2015]

~~"Greed is good. Greed is right. Greed works. Greed clarifies, cuts through and captures the essence of the evolutionary spirit."~~

~~- Gordon Gecko~~

Bipartite case

[Fenner, Gurjar, Thierauf 2015]

Make progress step-by-step

Construct isolating function iteratively

Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

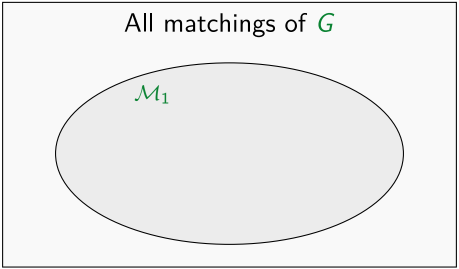
All matchings of G

Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

- Select $w_1 \in \mathcal{W}$ and let $\mathcal{M}_1$ be perfect matchings minimizing w_1

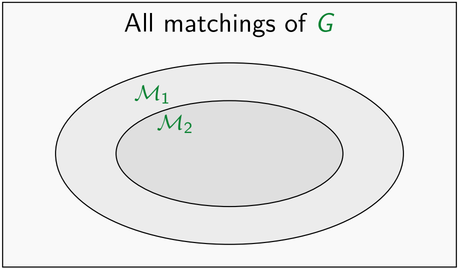


Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

- Select $w_1 \in \mathcal{W}$ and let $\mathcal{M}_1$ be perfect matchings minimizing w_1
- Select $w_2 \in \mathcal{W}$ and let $\mathcal{M}_2 \subseteq \mathcal{M}_1$ be PMs in $\mathcal{M}_1$ minimizing w_2



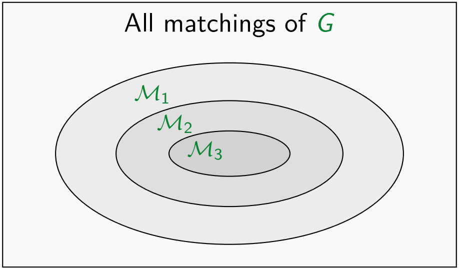
Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

- Select $w_1 \in \mathcal{W}$ and let $\mathcal{M}_1$ be perfect matchings minimizing w_1
- Select $w_2 \in \mathcal{W}$ and let $\mathcal{M}_2 \subseteq \mathcal{M}_1$ be PMs in $\mathcal{M}_1$ minimizing w_2
- Select $w_3 \in \mathcal{W}$ and let $\mathcal{M}_3 \subseteq \mathcal{M}_2$ be PMs in $\mathcal{M}_2$ minimizing w_3

$$\vdots$$



Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

- Select $w_1 \in \mathcal{W}$ and let $\mathcal{M}_1$ be perfect matchings minimizing w_1
- Select $w_2 \in \mathcal{W}$ and let $\mathcal{M}_2 \subseteq \mathcal{M}_1$ be PMs in $\mathcal{M}_1$ minimizing w_2
- Select $w_3 \in \mathcal{W}$ and let $\mathcal{M}_3 \subseteq \mathcal{M}_2$ be PMs in $\mathcal{M}_2$ minimizing w_3

$$\vdots$$

How many $w_1, \ldots, w_\ell \in \mathcal{W}$ necessary for $|\mathcal{M}_\ell| = 1$?

Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

- Select $w_1 \in \mathcal{W}$ and let $\mathcal{M}_1$ be perfect matchings minimizing w_1
- Select $w_2 \in \mathcal{W}$ and let $\mathcal{M}_2 \subseteq \mathcal{M}_1$ be PMs in $\mathcal{M}_1$ minimizing w_2
- Select $w_3 \in \mathcal{W}$ and let $\mathcal{M}_3 \subseteq \mathcal{M}_2$ be PMs in $\mathcal{M}_2$ minimizing w_3

$$\vdots$$

How many $w_1, \ldots, w_\ell \in \mathcal{W}$ necessary for $|\mathcal{M}_\ell| = 1$?

Thm [FGT'15]:

For any G, there is $w_1, \ldots, w_{\log_2(n)} \in \mathcal{W}$ so that $|\mathcal{M}_{\log_2(n)}| = 1$

Make progress step-by-step

Construct isolating function iteratively

Let $\mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$ be a polynomial set of simple weight functions

- Select $w_1 \in \mathcal{W}$ and let $\mathcal{M}_1$ be perfect matchings minimizing w_1
- Select $w_2 \in \mathcal{W}$ and let $\mathcal{M}_2 \subseteq \mathcal{M}_1$ be PMs in $\mathcal{M}_1$ minimizing w_2
- Select $w_3 \in \mathcal{W}$ and let $\mathcal{M}_3 \subseteq \mathcal{M}_2$ be PMs in $\mathcal{M}_2$ minimizing w_3

$$\vdots$$

How many $w_1, \ldots, w_\ell \in \mathcal{W}$ necessary for $|\mathcal{M}_\ell| = 1$?

Thm [FGT'15]:

For any G, there is $w_1, \ldots, w_{\log_2(n)} \in \mathcal{W}$ so that $|\mathcal{M}_{\log_2(n)}| = 1$

$$\Downarrow$$

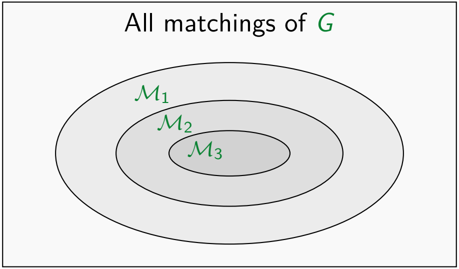
$$\mathcal{W}^* = \{n^{9(\log(n))} w_1 + n^{9(\log(n)-1)} w_2 + \cdots + 1 \cdot w_{\log(n)} : w_1, \ldots, w_{\log_2(n)} \in \mathcal{W}\}$$

gives oblivious quasi-polynomial derandomization

GOAL: For *any* n-vertex graph G, show that there is

$$w_1, \ldots, w_{\log n} \in \mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$$

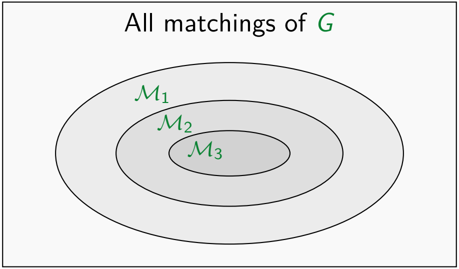
so that $|\mathcal{M}_{\log n}| = 1$



GOAL: For *any* n-vertex graph G, show that there is

$$w_1, \ldots, w_{\log n} \in \mathcal{W} = \{w_k : w_k(e_i) = 2^i \mod k \text{ for } k = 2, 3, \ldots, n^4\}$$

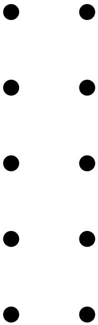
so that $|\mathcal{M}_{\log n}| = 1$



We need good progress measure

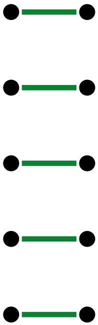
Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$



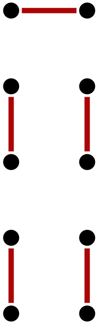
Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$



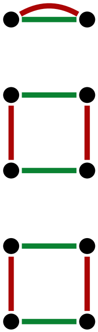
Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$



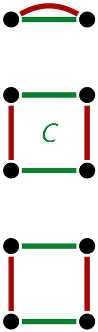
Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$
- symmetric difference $=$ alternating cycles



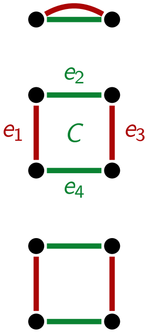
Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$
- symmetric difference $=$ alternating cycles
- in each cycle C, $w(M \cap C) = w(M' \cap C)$ (otherwise could get lighter matching)



Minimum perfect matchings of the same weight

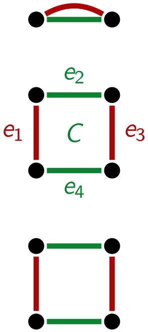
- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$
- symmetric difference $=$ alternating cycles
- in each cycle C, $w(M \cap C) = w(M' \cap C)$ (otherwise could get lighter matching)



$$w(e_1) + w(e_3) = w(e_2) + w(e_4)$$

Minimum perfect matchings of the same weight

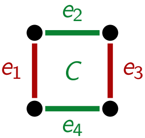
- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$
- symmetric difference
  $=$ alternating cycles
- in each cycle C,
  $w(M \cap C) = w(M' \cap C)$
  (otherwise could get lighter matching)
- define **discrepancy** of a cycle:
  $d_w(C) := w(e_1) - w(e_2) + w(e_3) - w(e_4)$



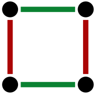
$$w(e_1) + w(e_3) = w(e_2) + w(e_4)$$

Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$
- symmetric difference $=$ alternating cycles
- in each cycle C, $w(M \cap C) = w(M' \cap C)$ (otherwise could get lighter matching)
- define **discrepancy** of a cycle: $d_w(C) := w(e_1) - w(e_2) + w(e_3) - w(e_4)$
- $d_w(C) = 0$



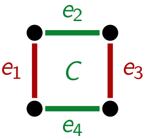
$$w(e_1) + w(e_3) = w(e_2) + w(e_4)$$



Minimum perfect matchings of the same weight

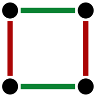
- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$

- symmetric difference $=$ alternating cycles



$$w(e_1) + w(e_3) = w(e_2) + w(e_4)$$

- in each cycle C, $w(M \cap C) = w(M' \cap C)$ (otherwise could get lighter matching)
- define **discrepancy** of a cycle: $d_w(C) := w(e_1) - w(e_2) + w(e_3) - w(e_4)$

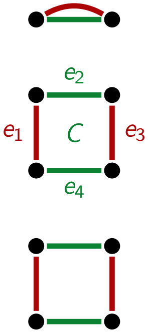


- $d_w(C) = 0$

If $(\forall C)\ d_w(C) \neq 0$, then w isolating!

Minimum perfect matchings of the same weight

- Consider min-weight perfect matchings M, M' with $w(M) = w(M')$
- symmetric difference $=$ alternating cycles
- in each cycle C, $w(M \cap C) = w(M' \cap C)$ (otherwise could get lighter matching)
- define **discrepancy** of a cycle: $d_w(C) := w(e_1) - w(e_2) + w(e_3) - w(e_4)$
- $d_w(C) = 0$



$$w(e_1) + w(e_3) = w(e_2) + w(e_4)$$

If $(\forall C)\ d_w(C) \neq 0$, then w isolating!

Progress: assign $\neq 0$ discrepancy to “many” cycles

Removing cycles

A graph may have exponentially many cycles $\Rightarrow$ seems hard to find w so that all of them have non-zero discrepancy

Removing cycles

A graph may have exponentially many cycles $\Rightarrow$ seems hard to find w so that all of them have non-zero discrepancy

Don't be greedy!

Old Lemma:

> **For any** collection of n^4 cycles, **some** $w \in \mathcal{W}$ assigns **all of them** $\neq 0$ discrepancy

Removing cycles

A graph may have exponentially many cycles $\Rightarrow$ seems hard to find w so that all of them have non-zero discrepancy

Don't be greedy!

Old Lemma:

> **For any** collection of n^4 cycles, **some** $w \in \mathcal{W}$ assigns **all of them** $\neq 0$ discrepancy

If $\leq n^4$ cycles in the graph: done!

Removing cycles

A graph may have exponentially many cycles $\Rightarrow$ seems hard to find w so that all of them have non-zero discrepancy

Don't be greedy!

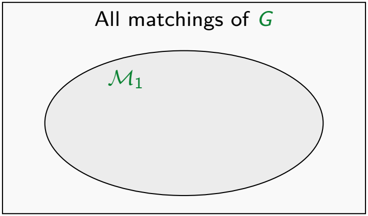
Old Lemma:

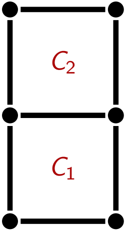
> **For any** collection of n^4 cycles, **some** $w \in \mathcal{W}$ assigns **all of them** $\neq 0$ discrepancy

If $\leq n^4$ cycles in the graph: done!

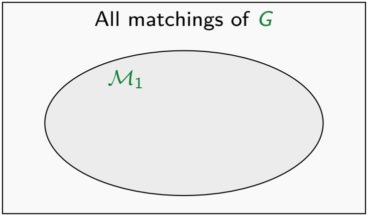
Not so easy, but we can cope with all **4-cycles**

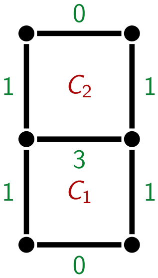
Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy



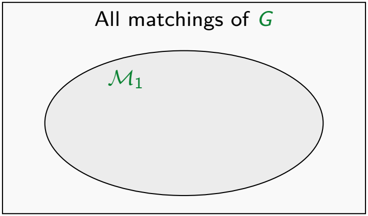


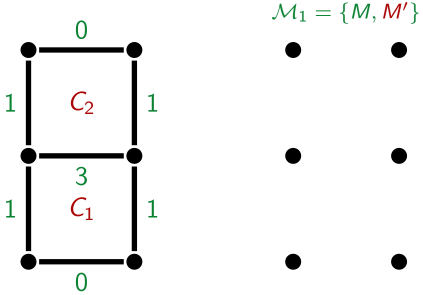
Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy



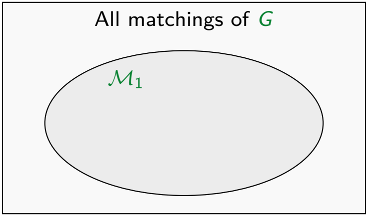


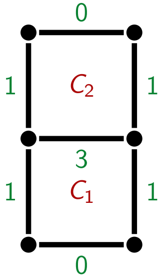
Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy



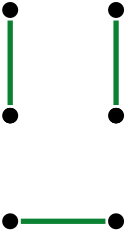


Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy

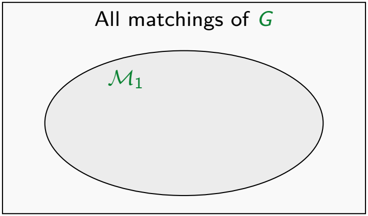


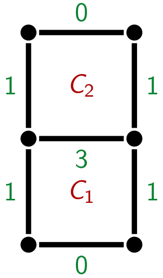


$\mathcal{M}_1 = \{M, M'\}$

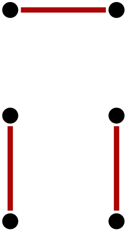


Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy

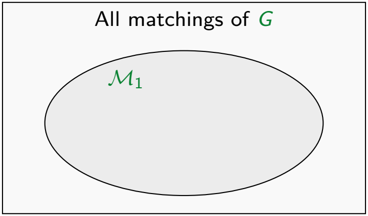


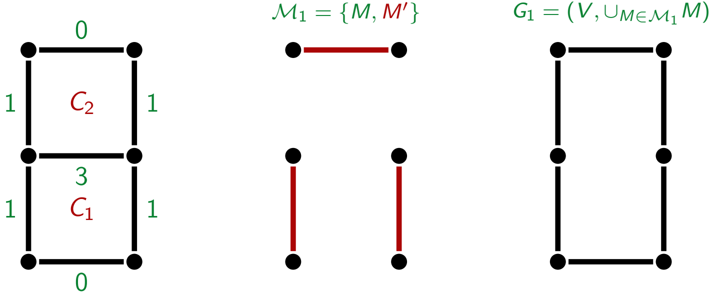


$\mathcal{M}_1 = \{M, M'\}$

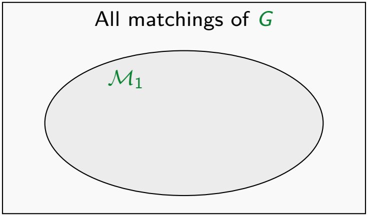


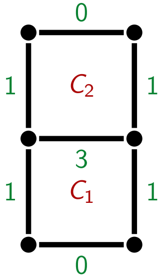
Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy



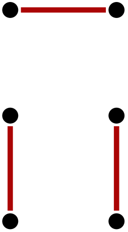


Select $w_1 \in \mathcal{W}$ so that all 4-cycles have $\neq 0$ discrepancy

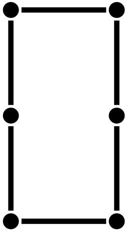




$\mathcal{M}_1 = \{M, M'\}$



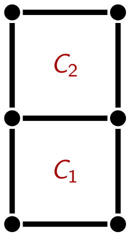
$G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$



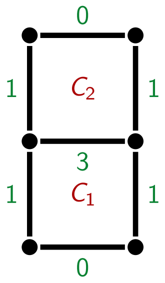
What can we say about the active subgraph G_1 that contains those edges that are in a min-weight perfect matching?

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

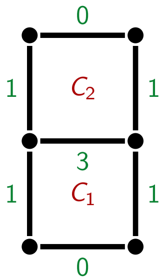
Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph



Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

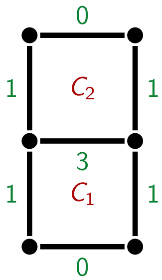


Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph



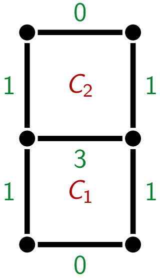
$d_w(C_1) = 1 \neq 0$
$d_w(C_2) = 1 \neq 0$

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

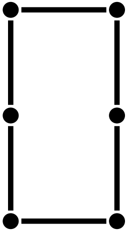


$$d_w(C_1) = 1 \neq 0$$
$$d_w(C_2) = 1 \neq 0$$
$$\Longrightarrow$$

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph



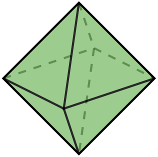
$$d_w(C_1) = 1 \neq 0$$
$$d_w(C_2) = 1 \neq 0$$
$$\Longrightarrow$$



Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):

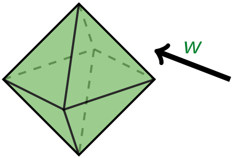


PM : perfect matching polytope (convex hull of matchings)

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):

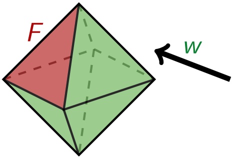


PM : perfect matching polytope (convex hull of matchings)

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):

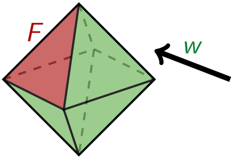


PM : perfect matching polytope (convex hull of matchings)

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):



PM : perfect matching polytope (convex hull of matchings)

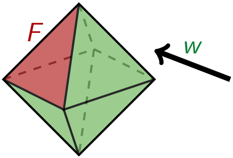
Bipartite PM

$$\begin{aligned} x(\delta(v)) &= 1 && \text{for every } v \in V \\ x_e &\geq 0 && \text{for every } e \in E \end{aligned}$$

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):



PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

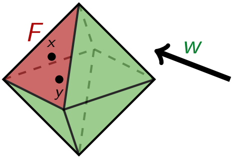
$$x(\delta(v)) = 1 \qquad \text{for every } v \in V$$
$$x_e \geq 0 \qquad \text{for every } e \in E$$

F is simply a subgraph

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):



PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

$$x(\delta(v)) = 1 \quad \text{for every } v \in V$$
$$x_e \geq 0 \quad \text{for every } e \in E$$

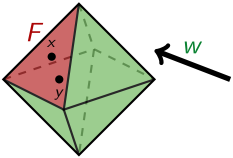
F is simply a subgraph

- What can we say about the weight of points in F?

Bipartite key property: Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph

Proof: Let $\mathcal{M}$ be the set of perfect matchings minimizing w

- Consider the convex hull of $\mathcal{M}$ (face F of the bipartite matching polytope):



PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

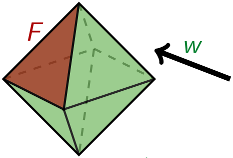
$x(\delta(v)) = 1$	for every $v \in V$
$x_e \geq 0$	for every $e \in E$

F is simply a subgraph

- What can we say about the weight of points in F?

  Every $x, y \in F$ have same weight: $\sum_e w(e)x_e = \sum_e w(e)y_e$

F is the convex hull of $\mathcal{M} \Rightarrow$ every $x, y \in F$ have same weight



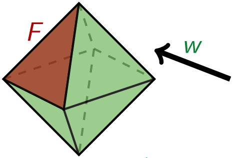
PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

$$x(\delta(v)) = 1 \qquad \text{for every } v \in V$$
$$x_e \geq 0 \qquad \text{for every } e \in E$$

F is simply a subgraph

F is the convex hull of $\mathcal{M} \Rightarrow$ every $x, y \in F$ have same weight



PM : perfect matching polytope (convex hull of matchings)

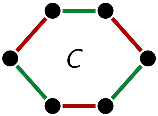
Bipartite PM

$$x(\delta(v)) = 1 \qquad \text{for every } v \in V$$
$$x_e \geq 0 \qquad \text{for every } e \in E$$

F is simply a subgraph

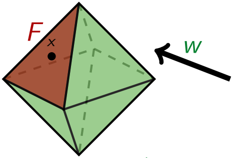
(edge set $\cup_{M \in \mathcal{M}} M$)

- Suppose active subgraph has cycle C of $\neq 0$ discepancy



w(green edges) $\neq$ w(red edges)

F is the convex hull of $\mathcal{M} \Rightarrow$ every $x, y \in F$ have same weight



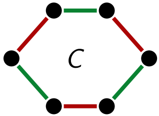
PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

$$x(\delta(v)) = 1 \qquad \text{for every } v \in V$$
$$x_e \geq 0 \qquad \text{for every } e \in E$$

F is simply a subgraph

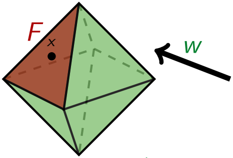
- Suppose active subgraph (edge set $\cup_{M \in \mathcal{M}} M$) has cycle C of $\neq 0$ discepancy



$w(\text{green edges}) \neq w(\text{red edges})$

- Let $x = \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \mathbf{1}_M$ be the mean of the face F

F is the convex hull of $\mathcal{M}$ $\Rightarrow$ every $x, y \in F$ have same weight



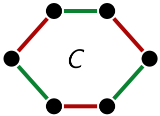
PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

$$x(\delta(v)) = 1 \qquad \text{for every } v \in V$$
$$x_e \geq 0 \qquad \text{for every } e \in E$$

F is simply a subgraph

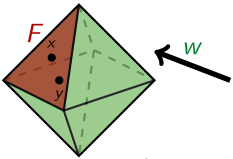
- Suppose active subgraph (edge set $\cup_{M \in \mathcal{M}} M$) has cycle C of $\neq 0$ discepancy



w(green edges) $\neq$ w(red edges)

- Let $x = \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \mathbf{1}_M$ be the mean of the face F
- Then $x_e > 0$ for every $e \in C$ (since support of x equals $\cup_{M \in \mathcal{M}} M$)

F is the convex hull of $\mathcal{M}$ $\Rightarrow$ every $x, y \in F$ have same weight



PM : perfect matching polytope (convex hull of matchings)

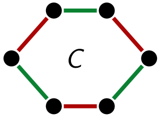
Bipartite PM

$$x(\delta(v)) = 1 \qquad \text{for every } v \in V$$
$$x_e \geq 0 \qquad \text{for every } e \in E$$

F is simply a subgraph

(edge set $\cup_{M \in \mathcal{M}} M$)

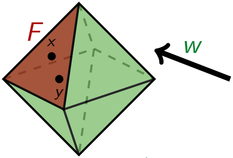
- Suppose active subgraph has cycle C of $\neq 0$ discepancy



w(green edges) $\neq$ w(red edges)

- Let $x = \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \mathbf{1}_M$ be the mean of the face F
- Then $x_e > 0$ for every $e \in C$ (since support of x equals $\cup_{M \in \mathcal{M}} M$)
- Increasing red edges while decreasing green maintain degrees

F is the convex hull of $\mathcal{M} \Rightarrow$ every $x, y \in F$ have same weight



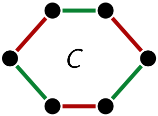
PM : perfect matching polytope (convex hull of matchings)

Bipartite PM

$x(\delta(v)) = 1$	for every $v \in V$
$x_e \geq 0$	for every $e \in E$

F is simply a subgraph

- Suppose active subgraph (edge set $\cup_{M \in \mathcal{M}} M$) has cycle C of $\neq 0$ discepancy



w(green edges) $\neq$ w(red edges)

- Let $x = \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \mathbf{1}_M$ be the mean of the face F
- Then $x_e > 0$ for every $e \in C$ (since support of x equals $\cup_{M \in \mathcal{M}} M$)
- Increasing red edges while decreasing green maintain degrees
- So we obtain a new point $y \in F$ of different weight; contradiction

The main ingredients

Old Lemma:

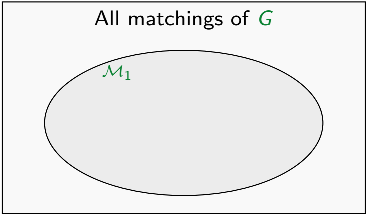
For any collection of n^4 cycles, **some** $w \in \mathcal{W}$ assigns **all of them** $\neq 0$ discrepancy

Bipartite key property:

Once we assign a cycle $\neq 0$ discrepancy,
it will disappear from the active subgraph

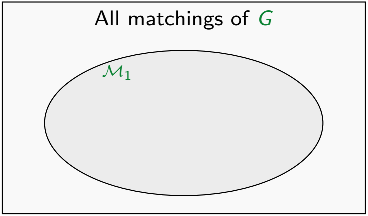
Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

A graph has at most n^4 cycles of length 4



Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

- Bipartite key property: $G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$ has no cycles of length ≤ 4

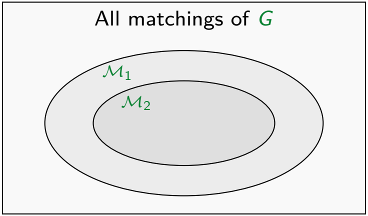


Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

- Bipartite key property: $G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$ has no cycles of length ≤ 4

Select $w_2 \in \mathcal{W}$ so that all ≤ 8-cycles in G_1 have $\neq 0$ discrepancy

A graph with no ≤ 4-cycles has at most n^4 cycles of length ≤ 8

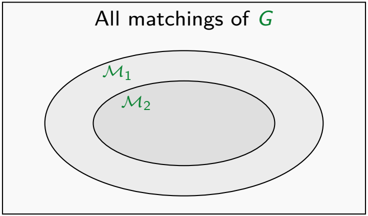


Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

- Bipartite key property: $G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$ has no cycles of length ≤ 4

Select $w_2 \in \mathcal{W}$ so that all ≤ 8-cycles in G_1 have $\neq 0$ discrepancy

- Bipartite key property: $G_2 = (V, \cup_{M \in \mathcal{M}_2} M)$ has no cycles of length ≤ 8



Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

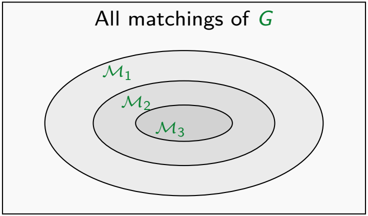
- Bipartite key property: $G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$ has no cycles of length ≤ 4

Select $w_2 \in \mathcal{W}$ so that all ≤ 8-cycles in G_1 have $\neq 0$ discrepancy

- Bipartite key property: $G_2 = (V, \cup_{M \in \mathcal{M}_2} M)$ has no cycles of length ≤ 8

Select $w_3 \in \mathcal{W}$ so that all ≤ 16-cycles in G_2 have $\neq 0$ discrepancy

A graph with no ≤ 8-cycles has at most n^4 cycles of length ≤ 16



Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

- Bipartite key property: $G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$ has no cycles of length ≤ 4

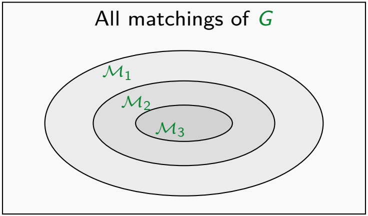
Select $w_2 \in \mathcal{W}$ so that all ≤ 8-cycles in G_1 have $\neq 0$ discrepancy

- Bipartite key property: $G_2 = (V, \cup_{M \in \mathcal{M}_2} M)$ has no cycles of length ≤ 8

Select $w_3 \in \mathcal{W}$ so that all ≤ 16-cycles in G_2 have $\neq 0$ discrepancy

- Bipartite key property: $G_3 = (V, \cup_{M \in \mathcal{M}_3} M)$ has no cycles of length ≤ 16

$$\vdots$$



Select $w_1 \in \mathcal{W}$ so that all 4-cycles in G have $\neq 0$ discrepancy

- Bipartite key property: $G_1 = (V, \cup_{M \in \mathcal{M}_1} M)$ has no cycles of length ≤ 4

Select $w_2 \in \mathcal{W}$ so that all ≤ 8-cycles in G_1 have $\neq 0$ discrepancy

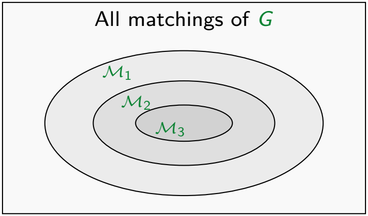
- Bipartite key property: $G_2 = (V, \cup_{M \in \mathcal{M}_2} M)$ has no cycles of length ≤ 8

Select $w_3 \in \mathcal{W}$ so that all ≤ 16-cycles in G_2 have $\neq 0$ discrepancy

- Bipartite key property: $G_3 = (V, \cup_{M \in \mathcal{M}_3} M)$ has no cycles of length ≤ 16

$$\vdots$$

$G_{\log n} = (V, \cup_{M \in \mathcal{M}_{\log n}} M)$ have no cycles so $|\mathcal{M}_{\log n}| = 1$ as required



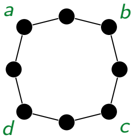
Final argument

A graph with no ≤ 4-cycles has at most n^4 cycles of length 8

Final argument

A graph with no ≤ 4-cycles has at most n^4 cycles of length 8

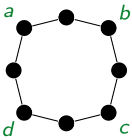
- Associate a signature (a, b, c, d) with each 8-cycle
  - a is the first vertex, b is the third vertex, c is the fifth vertex, d is the seventh vertex



Final argument

A graph with no ≤ 4-cycles has at most n^4 cycles of length 8

- Associate a signature (a, b, c, d) with each 8-cycle
  - a is the first vertex, b is the third vertex, c is the fifth vertex, d is the seventh vertex

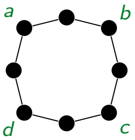


- Two cycles cannot have the same signature as that would imply a 4-cycle:

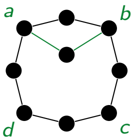
Final argument

A graph with no ≤ 4-cycles has at most n^4 cycles of length 8

- Associate a signature (a, b, c, d) with each 8-cycle
  - a is the first vertex, b is the third vertex, c is the fifth vertex, d is the seventh vertex



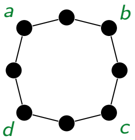
- Two cycles cannot have the same signature as that would imply a 4-cycle:



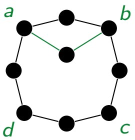
Final argument

A graph with no ≤ 4-cycles has at most n^4 cycles of length 8

- Associate a signature (a, b, c, d) with each 8-cycle
  - a is the first vertex, b is the third vertex, c is the fifth vertex, d is the seventh vertex



- Two cycles cannot have the same signature as that would imply a 4-cycle:

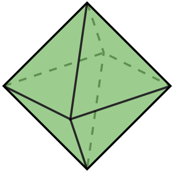


- So # 8-cycles is at most # signatures which is at most n^4

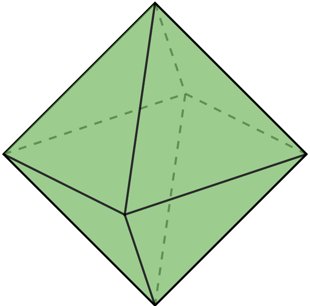
Some perspective

Polyhedral perspective

1

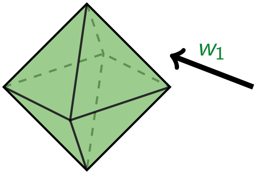


isolating in stages
=
decreasing sequence of faces

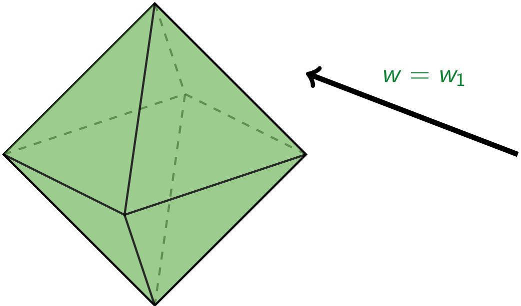


Polyhedral perspective

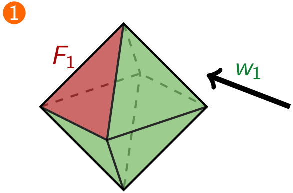
1



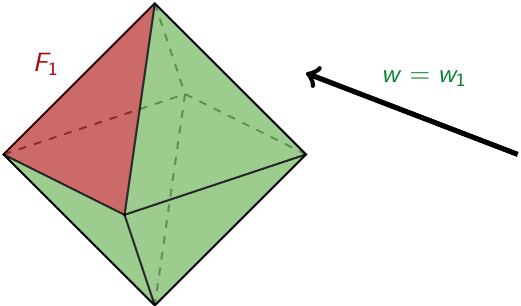
isolating in stages
=
decreasing sequence of faces



Polyhedral perspective

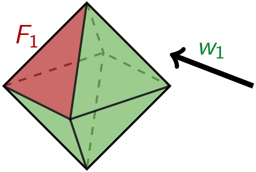


isolating in stages
=
decreasing sequence of faces

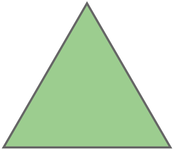


Polyhedral perspective

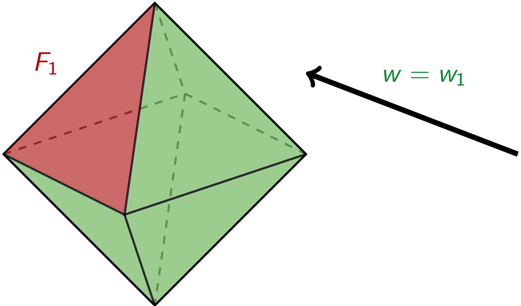
1



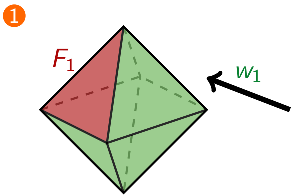
2

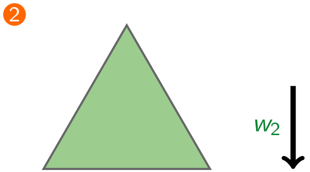


isolating in stages
=
decreasing sequence of faces



Polyhedral perspective

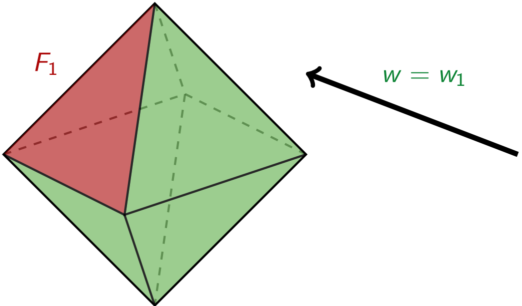




isolating in stages

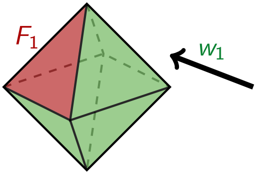
$=$

decreasing sequence of faces

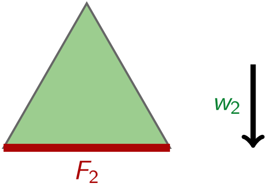


Polyhedral perspective

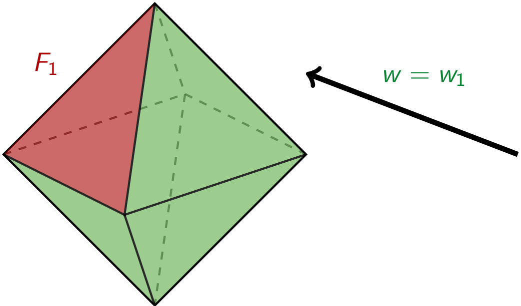
1



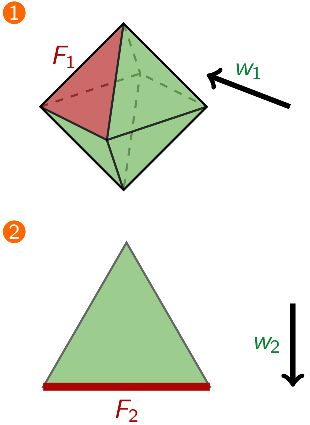
2



isolating in stages
=
decreasing sequence of faces



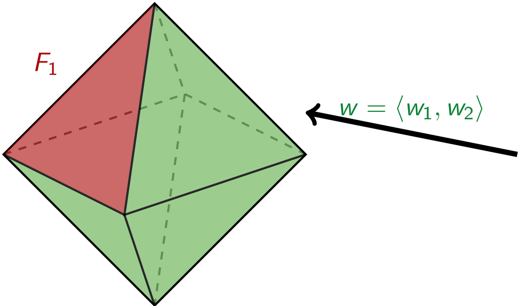
Polyhedral perspective



isolating in stages

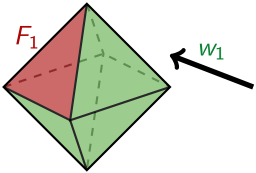
=

decreasing sequence of faces

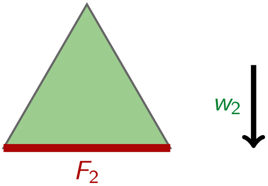


Polyhedral perspective

1



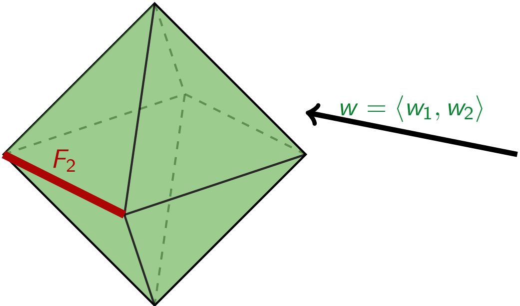
2



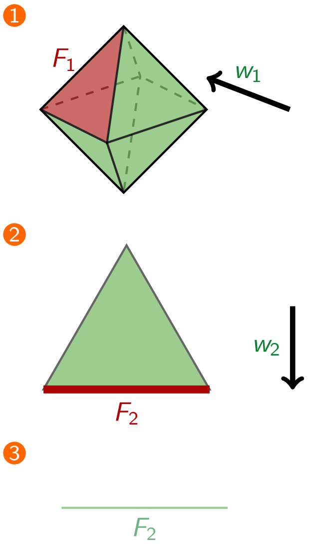
isolating in stages

=

decreasing sequence of faces



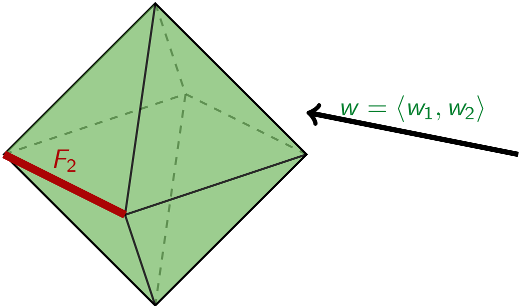
Polyhedral perspective



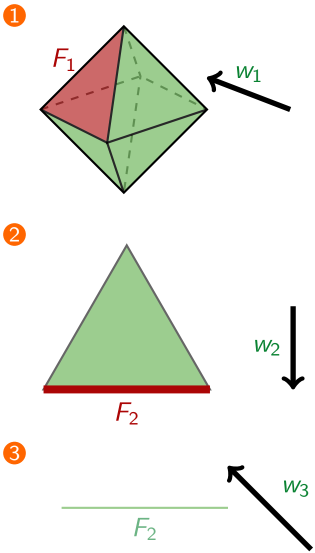
isolating in stages

=

decreasing sequence of faces



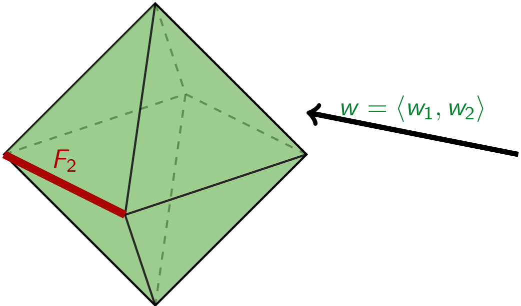
Polyhedral perspective



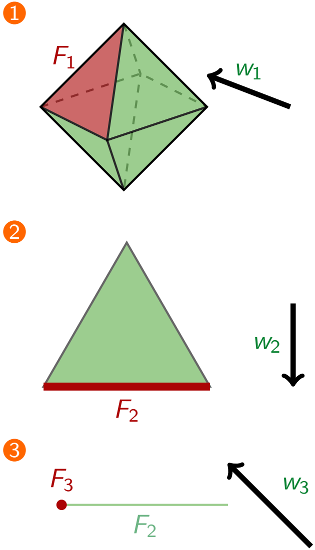
isolating in stages

=

decreasing sequence of faces



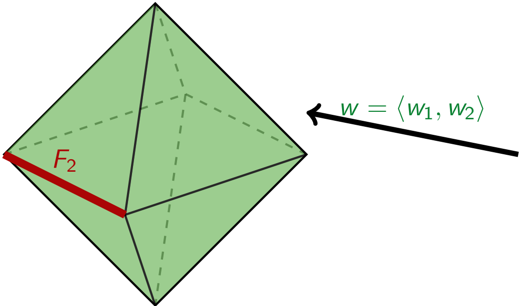
Polyhedral perspective



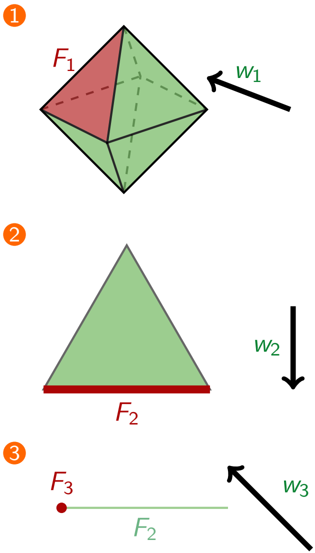
isolating in stages

=

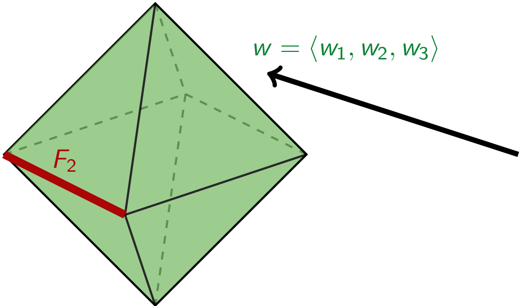
decreasing sequence of faces



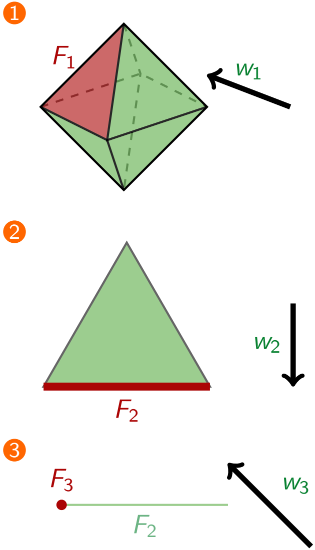
Polyhedral perspective



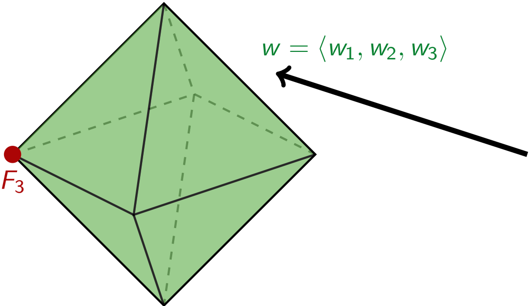
isolating in stages
=
decreasing sequence of faces



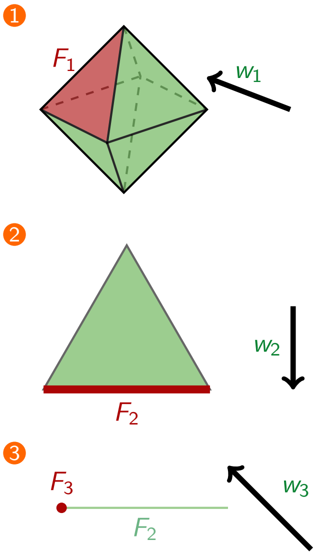
Polyhedral perspective



isolating in stages
=
decreasing sequence of faces



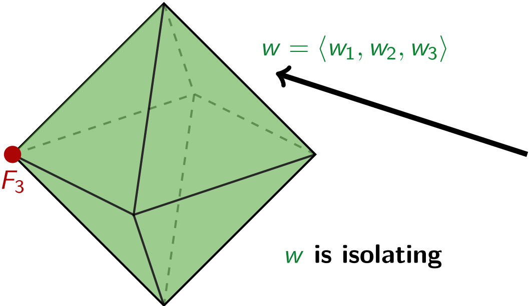
Polyhedral perspective



isolating in stages

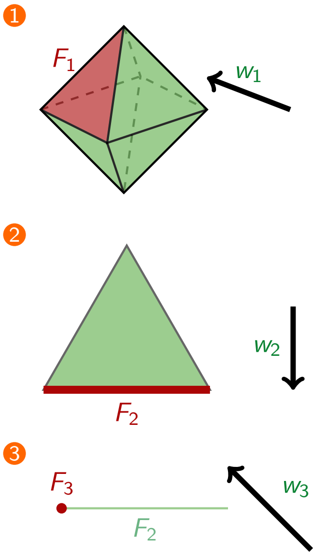
=

decreasing sequence of faces



w **is isolating**

Polyhedral perspective



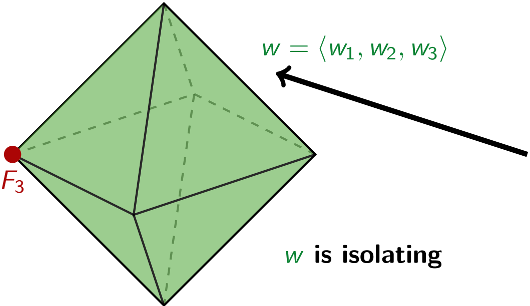
isolating in stages

=

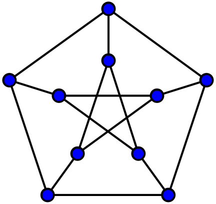
decreasing sequence of faces

Fast decrease due to bipartite matching polytope:

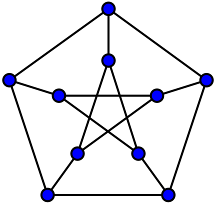
- every face is a subgraph
- Key property: girth doubles in every step



w **is isolating**



Difficulties of general case & our approach



Difficulties of general case & our approach

~~**Bipartite key property:** Once we assign a cycle $\neq 0$ discrepancy, it will disappear from the active subgraph~~

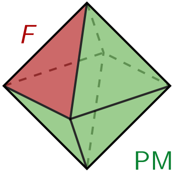
General graphs are "exponentially" harder

Edmonds [1965] Perfect matching polytope description on $x \in \mathbb{R}^E$:

- $x_e \geq 0$ for every edge e
- $x(\delta(v)) = 1$ for every vertex v

 $x(\delta(S)) \geq 1$ for every odd set S of vertices

($\delta(S)$ = edges crossing S)



General graphs are “exponentially” harder

Edmonds [1965] Perfect matching polytope description on $x \in \mathbb{R}^E$:

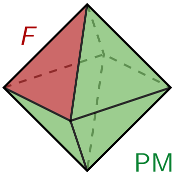
- $x_e \geq 0$ for every edge e
- $x(\delta(v)) = 1$ for every vertex v


$x(\delta(S)) \geq 1$ for every odd set S of vertices

($\delta(S)$ = edges crossing S)

So every face F is given as:

$$F = \{x \in \mathrm{PM} : x_e = 0 \quad \text{for some edges } e, \quad x(\delta(S)) = 1 \quad \text{for some odd sets } S\}$$



General graphs are "exponentially" harder

Edmonds [1965] Perfect matching polytope description on $x \in \mathbb{R}^E$:

- $x_e \geq 0$ for every edge e
- $x(\delta(v)) = 1$ for every vertex v

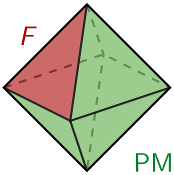
$x(\delta(S)) \geq 1$ for every odd set S of vertices

($\delta(S)$ = edges crossing S)

So every face F is given as:

$$F = \{x \in \text{PM} : x_e = 0 \quad \text{for some edges } e, \quad x(\delta(S)) = 1 \quad \text{for some odd sets } S\}$$

- In bipartite case:
  $F = \{x \in \text{PM} : \; x_e = 0 \;\; \text{for some edges } e\}$
  (F given by the active subgraph)
- Now, faces are exponentially harder
- Need $2^{\Omega(n)}$ inequalities [Rothvoss 2013]



General graphs are "exponentially" harder

Edmonds [1965] Perfect matching polytope description on $x \in \mathbb{R}^E$:

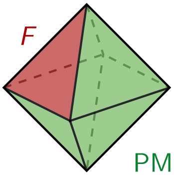
- $x_e \geq 0$ for every edge e
- $x(\delta(v)) = 1$ for every vertex v

$(\delta(S) = \text{edges crossing } S)$

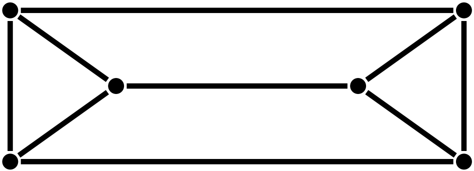
Girth does not make sense as progress measure and bipartite key property fails!

$x(\delta(S)) = 1$ for some odd sets $S\}$

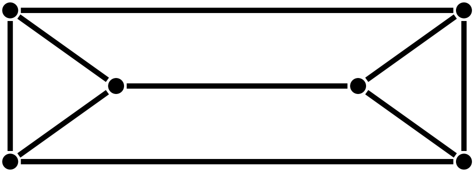
- In bipartite case:
  $F = \{x \in \text{PM} : \ x_e = 0 \ \text{ for some edges } e\}$
  (F given by the active subgraph)
- Now, faces are exponentially harder
- Need $2^{\Omega(n)}$ inequalities [Rothvoss 2013]



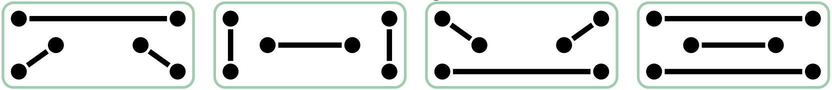
How bipartite key property fails



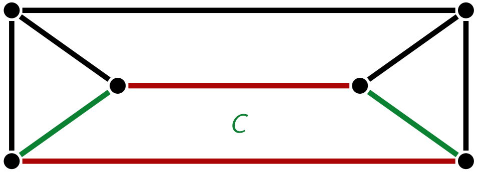
How bipartite key property fails



PM: convex hull of all four matchings:

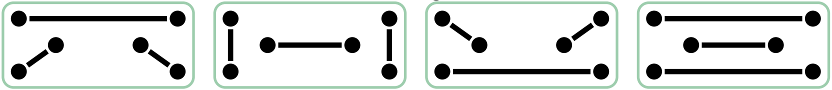


How bipartite key property fails

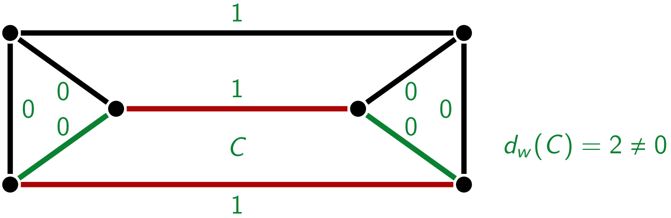


want:
$d_w(C) \neq 0$

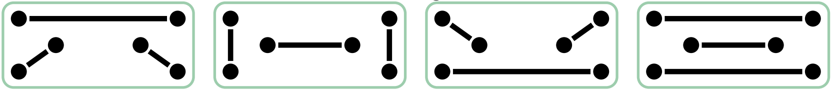
PM: convex hull of all four matchings:



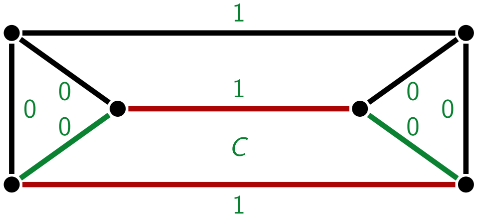
How bipartite key property fails



PM: convex hull of all four matchings:

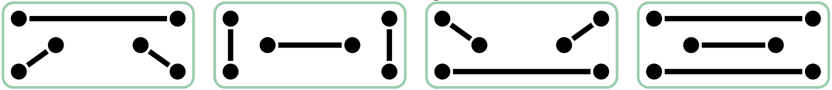


How bipartite key property fails

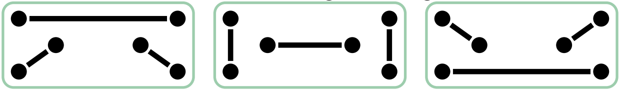


$d_w(C) = 2 \neq 0$

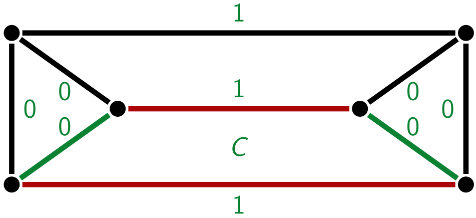
PM: convex hull of all four matchings:



F: convex hull of matchings of weight 1:

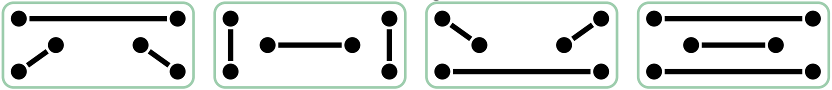


How bipartite key property fails

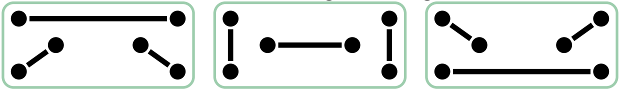


$d_w(C) = 2 \neq 0$

PM: convex hull of all four matchings:

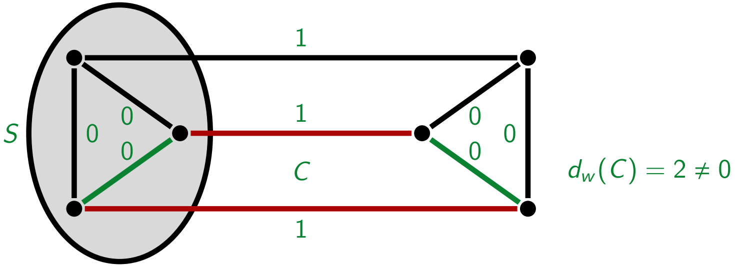


F: convex hull of matchings of weight 1:

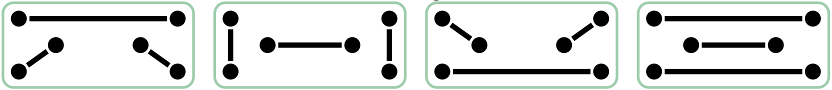


$F \subsetneq PM$ but still has all edges...

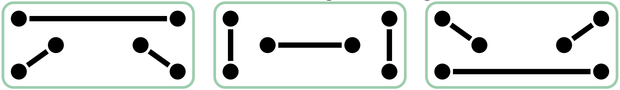
How bipartite key property fails



PM: convex hull of all four matchings:



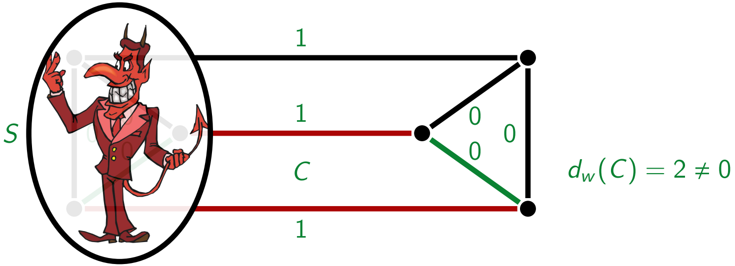
F: convex hull of matchings of weight 1:



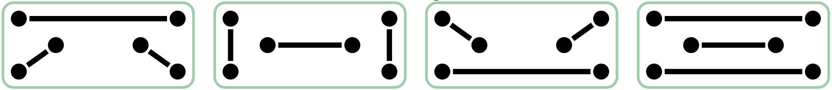
$F \subsetneq PM$ but still has all edges... 😢

$F = \{x \in PM : x(\delta(S)) = 1\}$

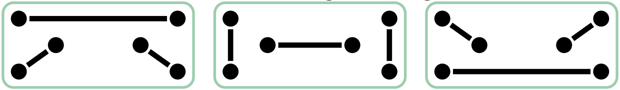
How bipartite key property fails



PM: convex hull of all four matchings:



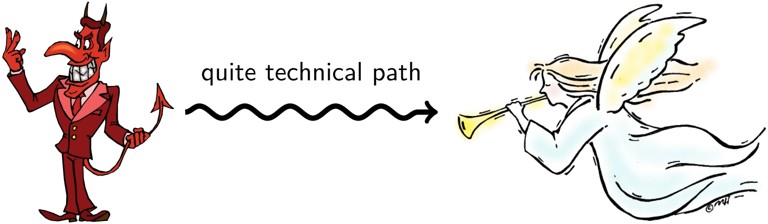
F: convex hull of matchings of weight 1:



$F \subsetneq PM$ but still has all edges... 

$F = \{x \in PM : x(\delta(S)) = 1\}$

quite technical path



Main ingredients:

- Laminar family of tight constraints (at most $2n-1$ constraints instead of exponential)
- Tight cut constraints decompose the instance

  $\Rightarrow$ divide-and-conquer approach

Laminarity

Every face F is given as:

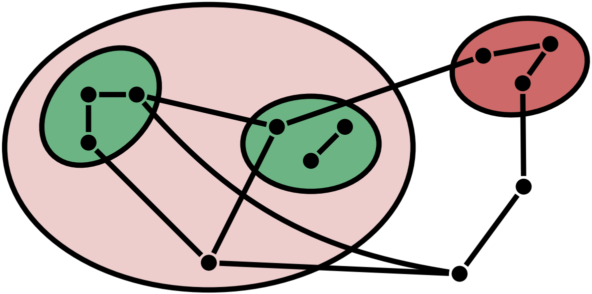
$$\begin{aligned} F = \{x \in \mathrm{PM} : x_e &= 0 && \text{for some edges } e, \\ x(\delta(S)) &= 1 && \text{for some odd sets } S\} \end{aligned}$$

Laminarity

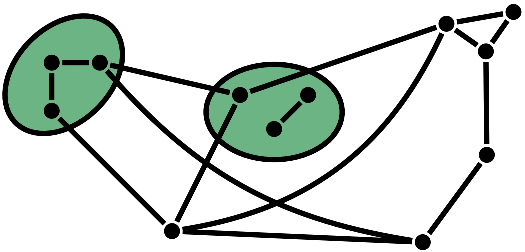
Every face F is given as:

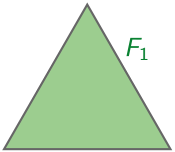
$$F = \{x \in \mathrm{PM} : x_e = 0 \quad \text{for some edges } e, \\ x(\delta(S)) = 1 \quad \text{for some odd sets } S\}$$

Great news: "some" can be chosen to be a laminar family!



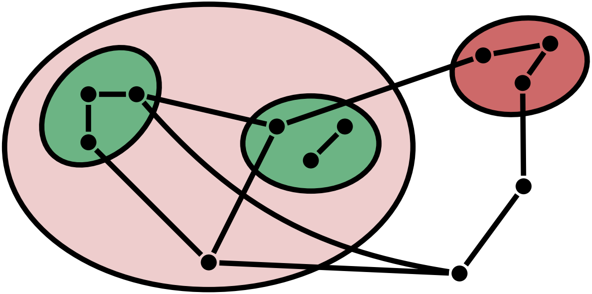
Laminarity

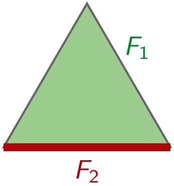




face $\sim$ (edge subset, laminar family)

Laminarity

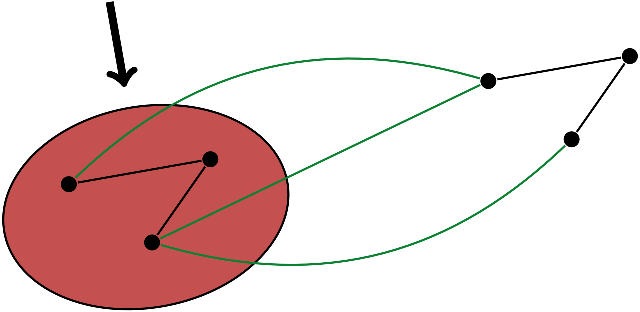




face $\sim$ (edge subset, laminar family)

Tight odd cuts decomposes instance

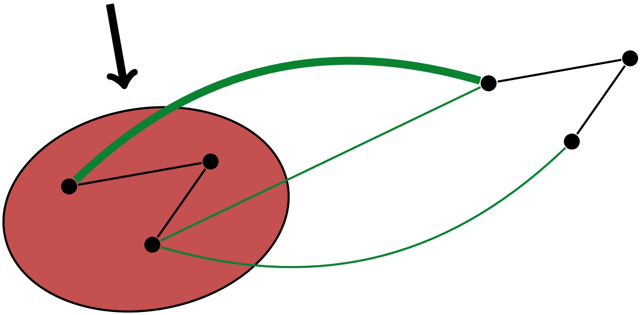
exactly one edge crossing



- once we fix a boundary edge...

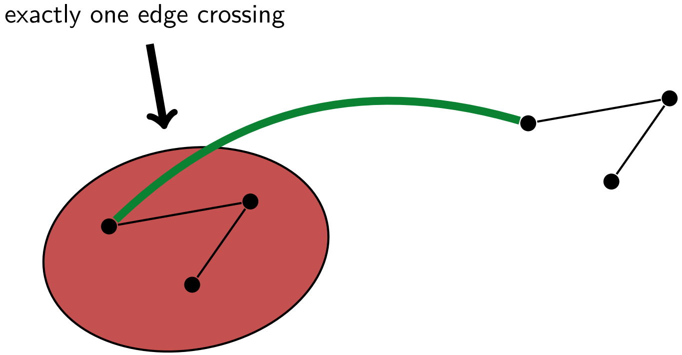
Tight odd cuts decomposes instance

exactly one edge crossing



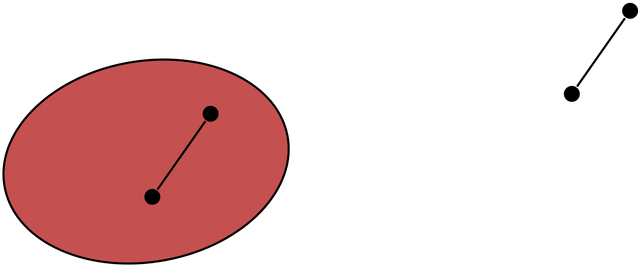
- once we fix a boundary edge...

Tight odd cuts decomposes instance



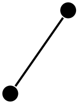
- once we fix a boundary edge...

Tight odd cuts decomposes instance



- once we fix a boundary edge...
- ... the instance decomposes into two **independent** ones

Tight odd cuts decomposes instance

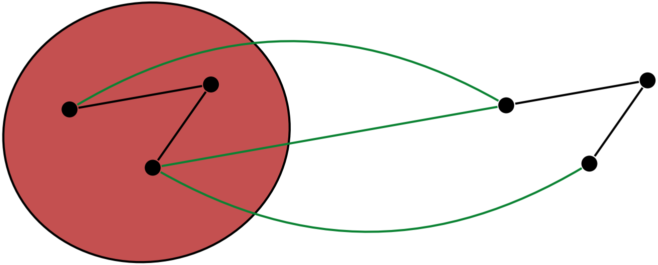


- once we fix a boundary edge...
- ... the instance decomposes into two **independent** ones

Divide & conquer

Between friends: cycles that do not cross tight odd sets behave like in the bipartite case and can thus be removed

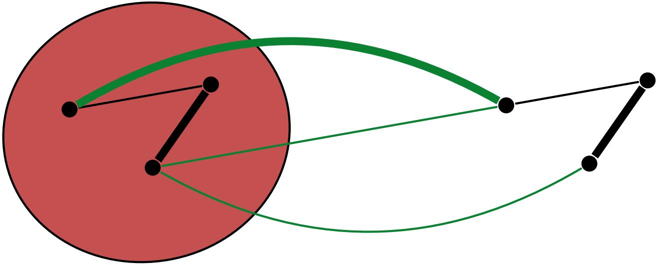
Simplest case: only one tight odd set



Divide & conquer

Between friends: cycles that do not cross tight odd sets behave like in the bipartite case and can thus be removed

Simplest case: only one tight odd set

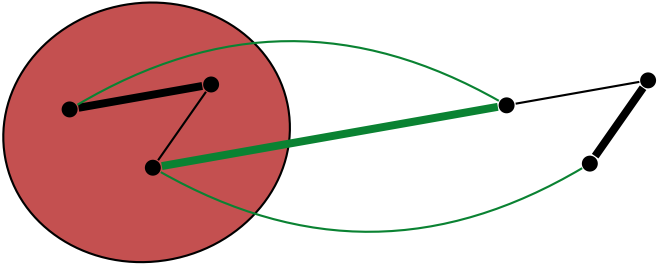


- then every boundary edge determines entire matching

Divide & conquer

Between friends: cycles that do not cross tight odd sets behave like in the bipartite case and can thus be removed

Simplest case: only one tight odd set

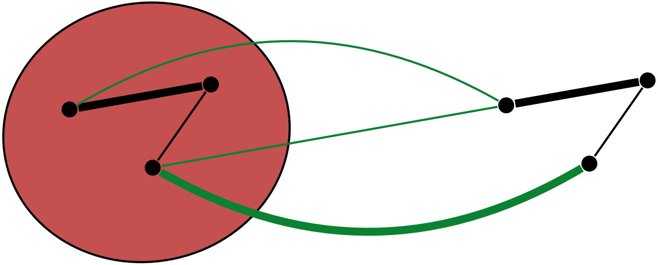


- then every boundary edge determines entire matching

Divide & conquer

Between friends: cycles that do not cross tight odd sets behave like in the bipartite case and can thus be removed

Simplest case: only one tight odd set

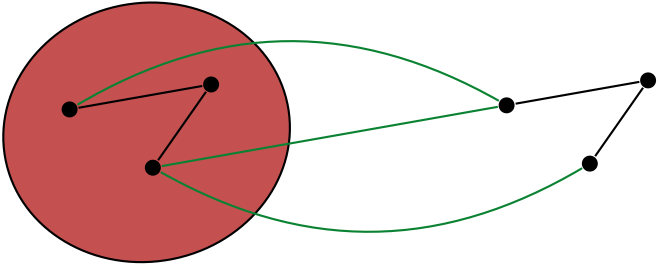


- then every boundary edge determines entire matching

Divide & conquer

Between friends: cycles that do not cross tight odd sets behave like in the bipartite case and can thus be removed

Simplest case: only one tight odd set

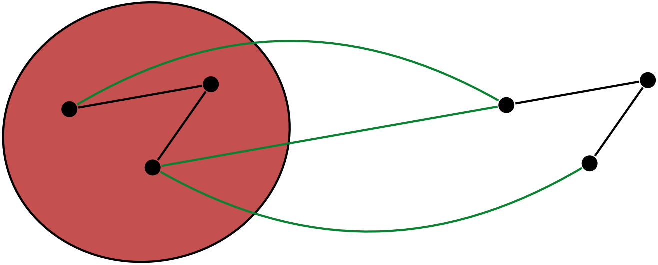


- then every boundary edge determines entire matching
- so: at most n^2 perfect matchings

Divide & conquer

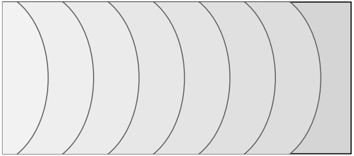
Between friends: cycles that do not cross tight odd sets behave like in the bipartite case and can thus be removed

Simplest case: only one tight odd set

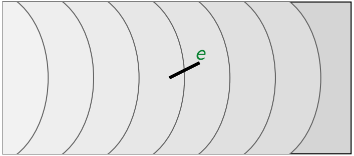


- then every boundary edge determines entire matching
- so: at most n^2 perfect matchings
- some $w \in \mathcal{W}$ will give them different weights

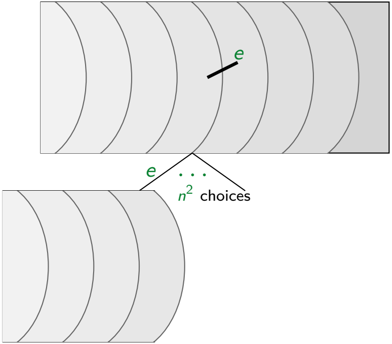
Divide & conquer: chain case



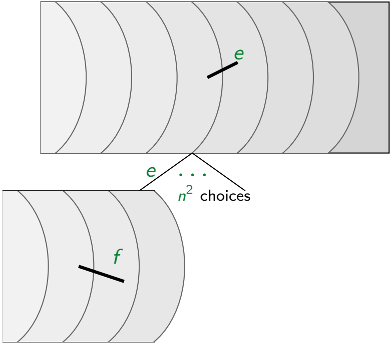
Divide & conquer: chain case



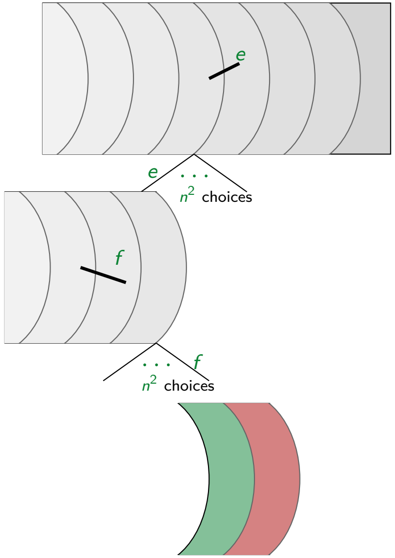
Divide & conquer: chain case



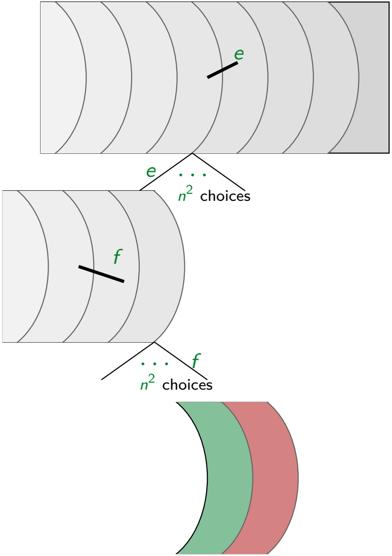
Divide & conquer: chain case



Divide & conquer: chain case

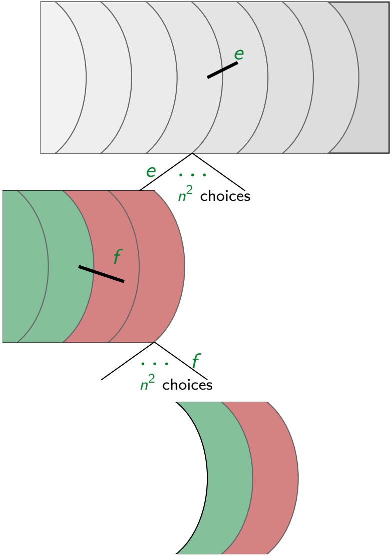


Divide & conquer: chain case



Instance where both sides of the cut are isolated,
one $w \in \mathcal{W}'$ makes the whole subinstance isolated

Divide & conquer: chain case



Now instance where both sides of the cut are isolated,
one $w \in \mathcal{W}'$ makes the whole subinstance isolated

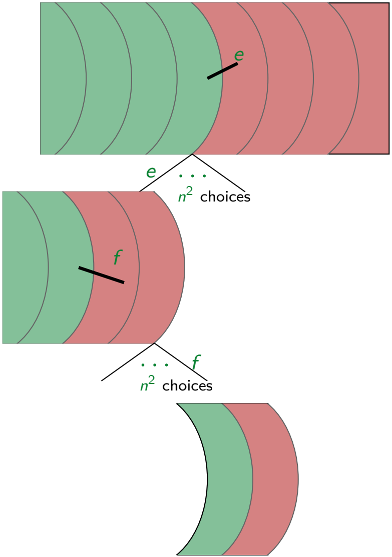
Instance where both sides of the cut are isolated,
one $w \in \mathcal{W}'$ makes the whole subinstance isolated

Divide & conquer: chain case

Now instance where both sides of the cut are isolated,
one $w \in \mathcal{W}'$ makes the **whole instance isolated :)**

Now instance where both sides of the cut are isolated,
one $w \in \mathcal{W}'$ makes the whole subinstance isolated

Instance where both sides of the cut are isolated,
one $w \in \mathcal{W}'$ makes the whole subinstance isolated



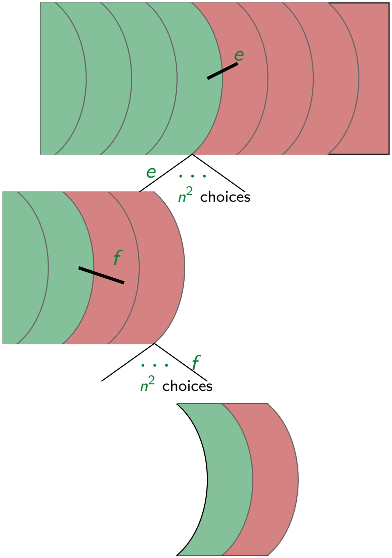
Divide & conquer: chain case

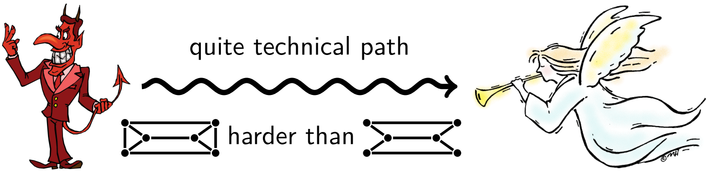
As before we isolate the whole instance in $O(\log n)$ phases

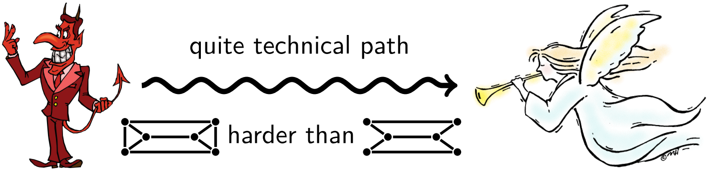
Now instance where both sides of the cut are isolated, one $w \in \mathcal{W}'$ makes the **whole instance isolated :)**

Now instance where both sides of the cut are isolated, one $w \in \mathcal{W}'$ makes the whole subinstance isolated

Instance where both sides of the cut are isolated, one $w \in \mathcal{W}'$ makes the whole subinstance isolated

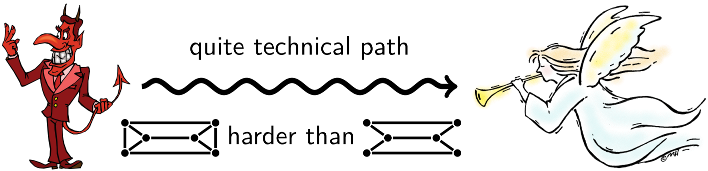



quite technical path
harder than



Carefully selected progress measure allows us to reduce laminar case to

- Removing cycles similar to bipartite case
- The chain case (divide-and-conquer)

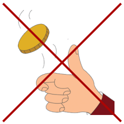


Carefully selected progress measure allows us to reduce laminar case to

- Removing cycles similar to bipartite case
- The chain case (divide-and-conquer)

Theorem S. and Tarnawski [2017]

General matching is in QUASI-$\mathcal{NC}$



with quasi-polynomial # processors

Future work

- go down to $\mathcal{NC}$
  - even for bipartite graphs
  - ✓ for planar graphs: [Anari, Vazirani 2017]

Future work

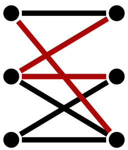
- go down to $\mathcal{NC}$
  - even for bipartite graphs
  - ✓ for planar graphs: [Anari, Vazirani 2017]

- derandomize Isolation Lemma in other cases (any efficiently solvable $\{0, 1\}$ polytope?)
  - ✓ matroid intersection: [Gurjar, Thierauf 2017]
  - ✓ totally unimodular polytopes: [Gurjar, Thierauf, Vishnoi 2017]

Future work

- go down to $\mathcal{NC}$
  - even for bipartite graphs
  - ✓ for planar graphs: [Anari, Vazirani 2017]
- derandomize Isolation Lemma in other cases (any efficiently solvable $\{0, 1\}$ polytope?)
  - ✓ matroid intersection: [Gurjar, Thierauf 2017]
  - ✓ totally unimodular polytopes: [Gurjar, Thierauf, Vishnoi 2017]

Exact Matching Problem



Given: graph with some edges red, number k.
Is there a perfect matching with exactly k red edges?

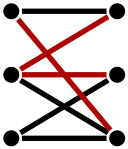
- randomized complexity: even RANDOMIZED $\mathcal{NC}$
- deterministic complexity: is it in $\mathcal{P}$?

Future work

- go down to $\mathcal{NC}$
  - even for bipartite graphs
  - ✓ for planar graphs: [Anari, Vazirani 2017]

- derandomize Isolation Lemma in other cases (any efficiently solvable $\{0,1\}$ polytope?)
  - ✓ matroid intersection: [Gurjar, Thierauf 2017]
  - ✓ totally unimodular polytopes: [Gurjar, Thierauf, Vishnoi 2017]

Exact Matching Problem



Given: graph with some edges red, number k.
Is there a perfect matching with exactly k red edges?

- randomized complexity: even RANDOMIZED $\mathcal{NC}$
- deterministic complexity: is it in $\mathcal{P}$?

Thank you!