43ème École de Printemps d'Informatique Théorique (ÉPIT)

Graphes et surfaces : algorithmique, combinatoire et topologie

Speaker: Dimitrios Thilikos

CIRM, Marseille, May 09–13, 2016

Lectures:

- Monday 09/05/2016 - 11:45–12:30 (45'): **22** pages

  Complexity, Graphs, P vs. NP, NP-Hardness, Fixed-parameter tractability
- Monday 09/05/2016 - 16:00–17:30 (90'): **42** pages

  treewidth, dynamic programming
- Tuesday 10/05/2016 - 16:00–17:00 (60'): **26** pages

  Sphere-cut decompositions
- Thursday 10/05/2016 - 11:00–12:30 (90'): **31** pages

  Bidimensionality and subexponential algorithms
- Friday 10/05/2016 - 09:00–10:30 (90'): **14**+**20**=**34** pages

  Bidimensionality and Kernels + Irrelevant vertex technique

155 pages, **375** Minutes in total, 145 seconds per page (2.42 minutes per page)

Part 1, Monday 09/05/2016 - 11:45–12:30 (45')

Complexity,

Graphs,

P vs. NP,

NP-Hardness,

Fixed-parameter tractability

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

$2016n + 3016 = O(n)$: linear

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

$2016n + 3016 = O(n)$: linear

$23n^2 + 16n + 4 = O(n^2)$: quadratic

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

$2016n + 3016 = O(n)$: linear

$23n^2 + 16n + 4 = O(n^2)$: quadratic

$89n^2 + 6n^2 - 4n + 45676 = O(n^3)$: cubic

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

$2016n + 3016 = O(n)$: linear

$23n^2 + 16n + 4 = O(n^2)$: quadratic

$89n^2 + 6n^2 - 4n + 45676 = O(n^3)$: cubic

$5n^c = O(n^c) = n^{O(1)}$: polynomial

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

$2016n + 3016 = O(n)$: linear

$23n^2 + 16n + 4 = O(n^2)$: quadratic

$89n^2 + 6n^2 - 4n + 45676 = O(n^3)$: cubic

$5n^c = O(n^c) = n^{O(1)}$: polynomial

$n^3 4 \cdot 3^{\sqrt{n}} + n^{2016} = 2^{O(\sqrt{n})}$: subexponential

O-notation

$f(n) = O(g(n)) \iff$ there exist constants N and C such that

$$\forall n > N \quad f(n) \leq C \cdot g(n).$$

- Examples:

$2016 = O(1) = O(3)$: Constant

$4 \log_4 n + 5 \log_3 n = O(\log n)$: Logarithmic

$23 \log^5 n = O(\log^5 n) = \log^{O(1)} n$: polylogarithmic

$2016\sqrt{n} + 3016 = O(\sqrt{n})$: sublinear

$2016n + 3016 = O(n)$: linear

$23n^2 + 16n + 4 = O(n^2)$: quadratic

$89n^2 + 6n^2 - 4n + 45676 = O(n^3)$: cubic

$5n^c = O(n^c) = n^{O(1)}$: polynomial

$n^3 4 \cdot 3^{\sqrt{n}} + n^{2016} = 2^{O(\sqrt{n})}$: subexponential

$30^{7n+3} = 2^{O(n)}$: singly exponential

Problems and algorithms

Definition of an problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where Σ is an alphabet, typically $\Sigma = \{0, 1\}$

We look for a way to decide, given a $x \in \Sigma^*$, whether $x \in \Pi$.

Definition of an algorithm:

Problems and algorithms

Definition of an problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where Σ is an alphabet, typically $\Sigma = \{0, 1\}$

We look for a way to decide, given a $x \in \Sigma^*$, whether $x \in \Pi$.

Definition of an algorithm:

Muhammad ibn Musa al-Khwarizmi

Problems and algorithms

Definition of an problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where Σ is an alphabet, typically $\Sigma = \{0, 1\}$

We look for a way to decide, given a $x \in \Sigma^*$, whether $x \in \Pi$.

Definition of an algorithm:

Muhammad ibn Musa al-Khwarizmi

Alan Mathison Turing

Problems and algorithms

Definition of an problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where Σ is an alphabet, typically $\Sigma = \{0, 1\}$

We look for a way to decide, given a $x \in \Sigma^*$, whether $x \in \Pi$.

Definition of an algorithm:

Muhammad ibn Musa al-Khwarizmi

Alan Mathison Turing

Omitted!

Problems and algorithms

Definition of an problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where Σ is an alphabet, typically $\Sigma = \{0, 1\}$

We look for a way to decide, given a $x \in \Sigma^*$, whether $x \in \Pi$.

Definition of an algorithm:

Muhammad ibn Musa al-Khwarizmi

Alan Mathison Turing

Omitted! This journey is beautiful and long. We do not take it this week! ☹

Graphs

We mostly work on algorithms on graphs:

$V(G)$: vertices of G, $E(G)$: edges of G, $\mathcal{G}_{\text{all}}$: the set of all graphs

Graphs

We mostly work on algorithms on graphs:

$V(G)$: vertices of G, $E(G)$: edges of G, $\mathcal{G}_{\text{all}}$: the set of all graphs

A problem on graphs:

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?

Graphs

We mostly work on algorithms on graphs:

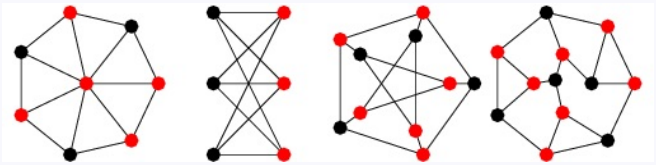
$V(G)$: vertices of G, $E(G)$: edges of G, $\mathcal{G}_{\text{all}}$: the set of all graphs

A problem on graphs:

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?



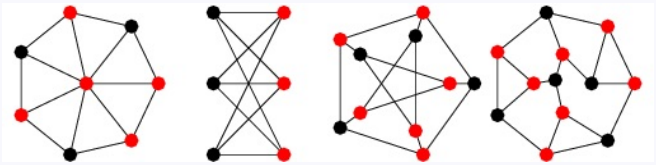
Graphs

We mostly work on algorithms on graphs:

$V(G)$: vertices of G, $E(G)$: edges of G, $\mathcal{G}_{\text{all}}$: the set of all graphs

A problem on graphs:

> VERTEX COVER
>
> *Instance:* A graph G and an integer $k \geq 0$.
>
> *Question:* $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?



Here:

Input: $x = \langle G, k \rangle$, i.e., x encodes the graph G and the integer k

Graphs

We mostly work on algorithms on graphs:

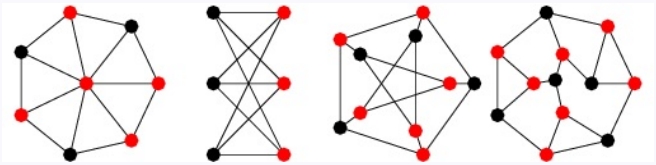
$V(G)$: vertices of G, $E(G)$: edges of G, $\mathcal{G}_{\text{all}}$: the set of all graphs

A problem on graphs:

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?



Here:

Input: $x = \langle G, k \rangle$, i.e., x encodes the graph G and the integer k

Problem: $\Pi_{\textbf{cv}} = \{\langle G, k \rangle \mid G \text{ has a vertex cover of size} \leq k\}$

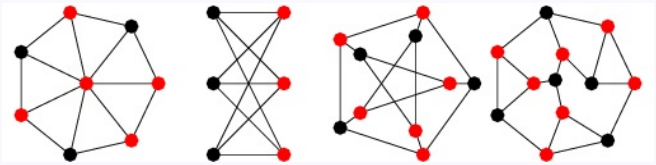
Graphs

We mostly work on algorithms on graphs:

$V(G)$: vertices of G, $E(G)$: edges of G, $\mathcal{G}_{\text{all}}$: the set of all graphs

A problem on graphs:

> VERTEX COVER
>
> *Instance:* A graph G and an integer $k \geq 0$.
>
> *Question:* $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?



Here:

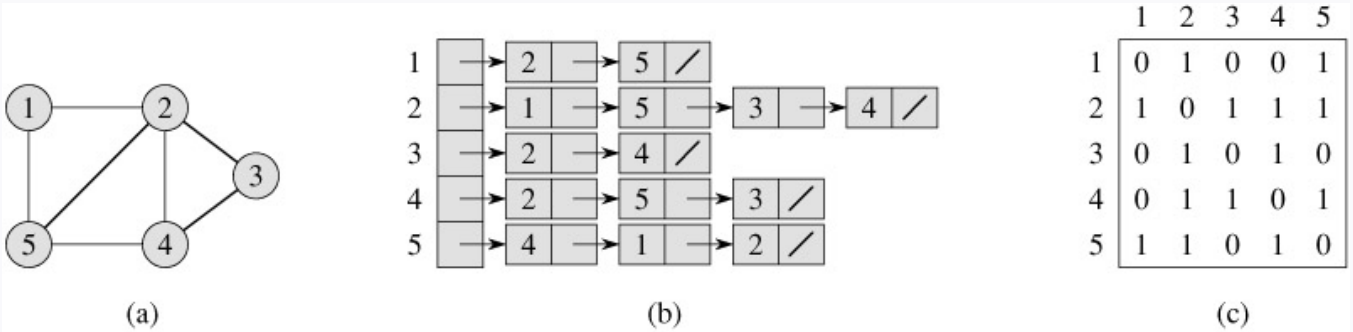
Input: $x = \langle G, k \rangle$, i.e., x encodes the graph G and the integer k

Problem: $\Pi_{\textbf{cv}} = \{\langle G, k \rangle \mid G \text{ has a vertex cover of size} \leq k\}$

Algorithm for VERTEX COVER: a ~~procedure~~ than receives as input $x = \langle G, k \rangle$ and outputs whether $x \in \Pi_{\textbf{vc}}$

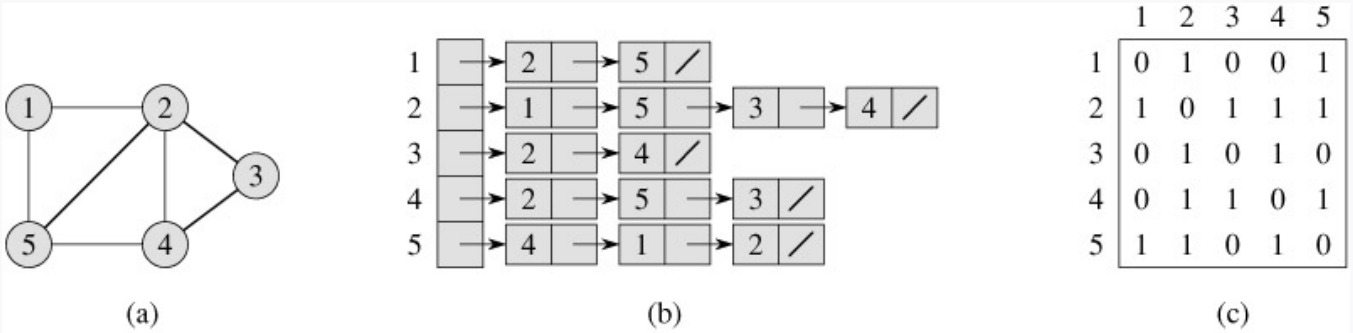
Data structures

Data structures for graphs: Adjacency list, Adjacency matrix



Data structures

Data structures for graphs: Adjacency list, Adjacency matrix

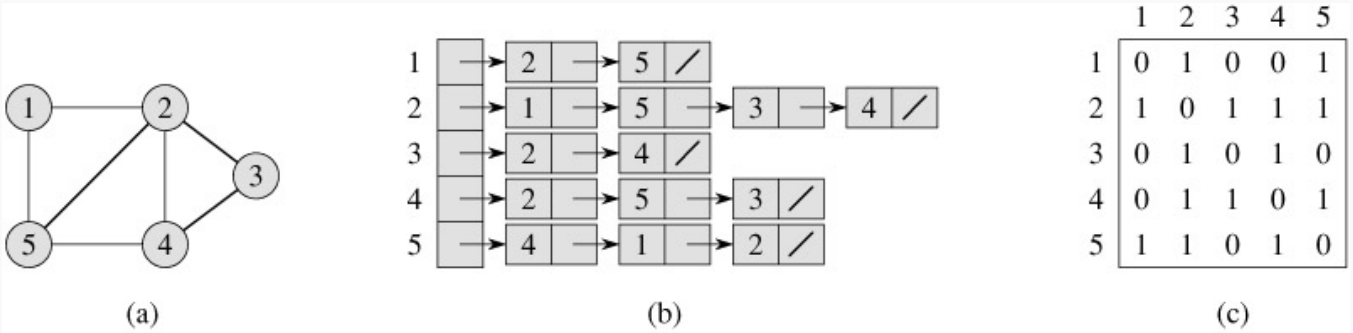


- Running time of a graph algorithm $=$

\# of elementary operations on the data structure that represents its input.

Data structures

Data structures for graphs: Adjacency list, Adjacency matrix



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

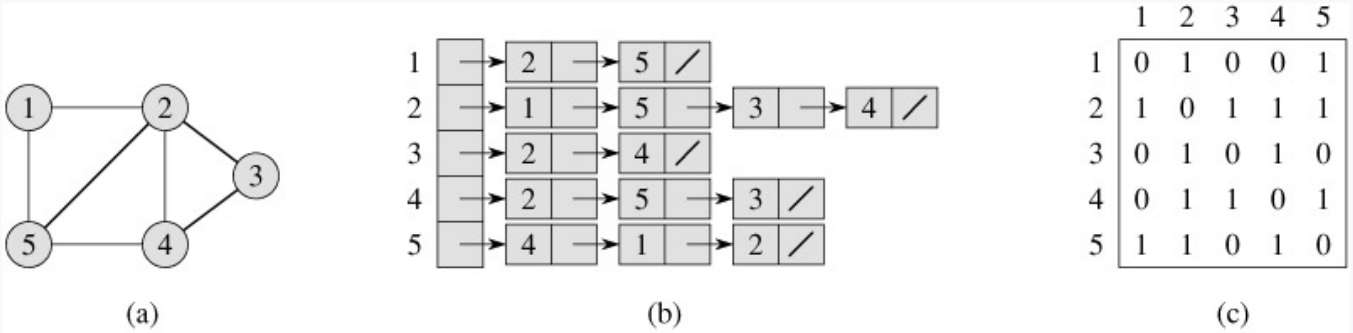
- Running time of a graph algorithm $=$

$\#$ of elementary operations on the data structure that represents its input.

Most graphs in this lectures are sparse: $|E(G)| = O(|V(G)|)$ (By Euler's formula)

Data structures

Data structures for graphs: Adjacency list, Adjacency matrix



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

- Running time of a graph algorithm $=$

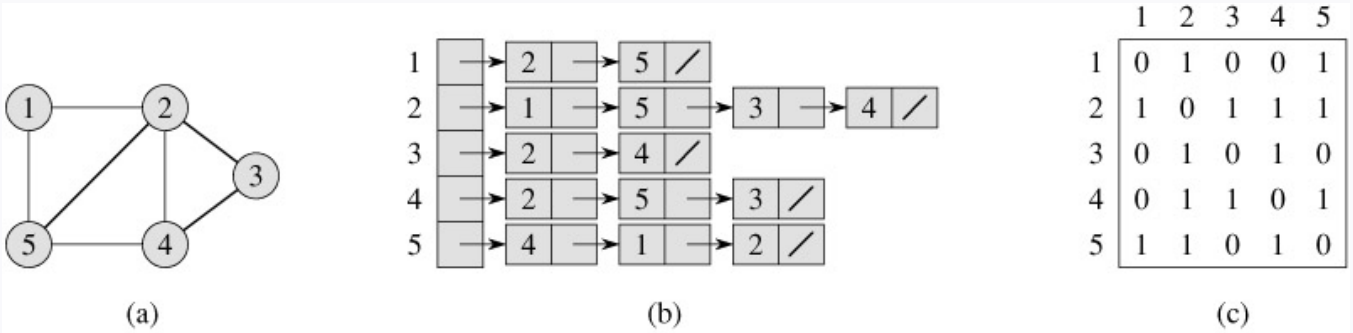
\# of elementary operations on the data structure that represents its input.

Most graphs in this lectures are sparse: $|E(G)| = O(|V(G)|)$ (By Euler's formula)

For this reason we prefer the Adjacency list data structure.

Data structures

Data structures for graphs: Adjacency list, Adjacency matrix



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

- Running time of a graph algorithm $=$

\# of elementary operations on the data structure that represents its input.

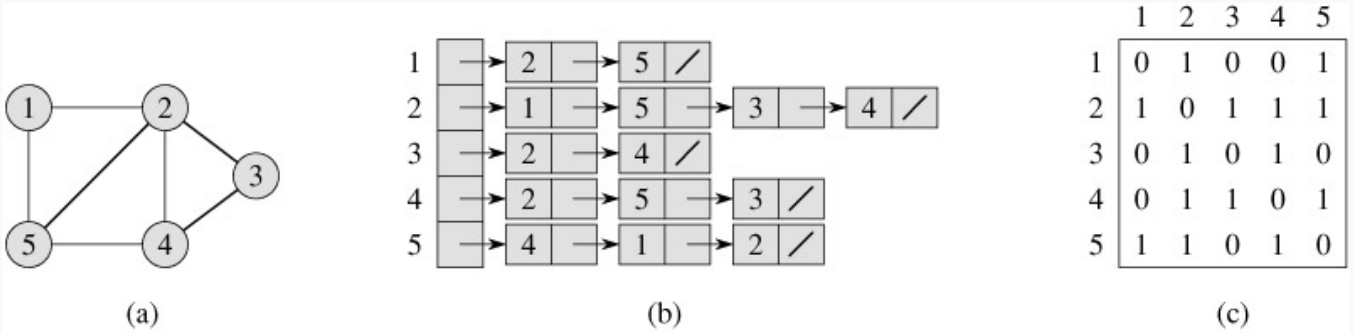
Most graphs in this lectures are sparse: $|E(G)| = O(|V(G)|)$ (By Euler's formula)

For this reason we prefer the Adjacency list data structure.

We also assume that arithmetic operations take $O(1)$ steps! 

Data structures

Data structures for graphs: Adjacency list, Adjacency matrix



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

- Running time of a graph algorithm $=$

\# of elementary operations on the data structure that represents its input.

Most graphs in this lectures are sparse: $|E(G)| = O(|V(G)|)$ (By Euler's formula)

For this reason we prefer the Adjacency list data structure.

We also assume that arithmetic operations take $O(1)$ steps! 

We measure the time complexity of a graph algorithm by as a function of $n = |V(G)|$

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G) \ |e \cap S| \geq 1}$?

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G) \ |e \cap S| \geq 1$?

S is the certificate.

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?

S is the certificate.

If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

Vertex Cover
Instance: A graph G and an integer $k \geq 0$.
Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1}$?

S is the certificate.

If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Clearly: P $\subseteq$ NP

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1}$?

S is the certificate.

If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Clearly: P $\subseteq$ NP $\qquad$ 10^6\$-question: is it correct that P $\neq$ NP?

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

Vertex Cover
Instance: A graph G and an integer $k \geq 0$.
Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1}$?

S is the certificate.

If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Clearly: P $\subseteq$ NP $\qquad$ 10^6\$-question: is it correct that P $\neq$ NP?

A problem Π is NP-hard if it is "as hard as"

all problems in NP

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

Vertex Cover
Instance: A graph G and an integer $k \geq 0$.
Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1}$?

S is the certificate.

If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Clearly: P $\subseteq$ NP $\qquad$ 10^6\$-question: is it correct that P $\neq$ NP?

A problem Π is NP-hard if it is "as hard as"

all problems in NP

A problem Π is NP-complete if it is NP-hard

and belongs in NP

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1}$?

S is the certificate.

If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Clearly: P $\subseteq$ NP $\qquad$ 10^6\$-question: is it correct that P $\neq$ NP?

A problem Π is NP-hard if it is "as hard as"

all problems in NP

A problem Π is NP-complete if it is NP-hard

and belongs in NP

VERTEX COVER is an NP-complete problem

Complexity classes

P: contains all problems that can be solved in $n^{O(1)}$ steps

NP: contains all problems that can be "certified" in $n^{O(1)}$ steps

Vertex Cover
Instance: A graph G and an integer $k \geq 0$.
Question: $\exists S \in V(G) : \boxed{|S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1}$?

S is the certificate.

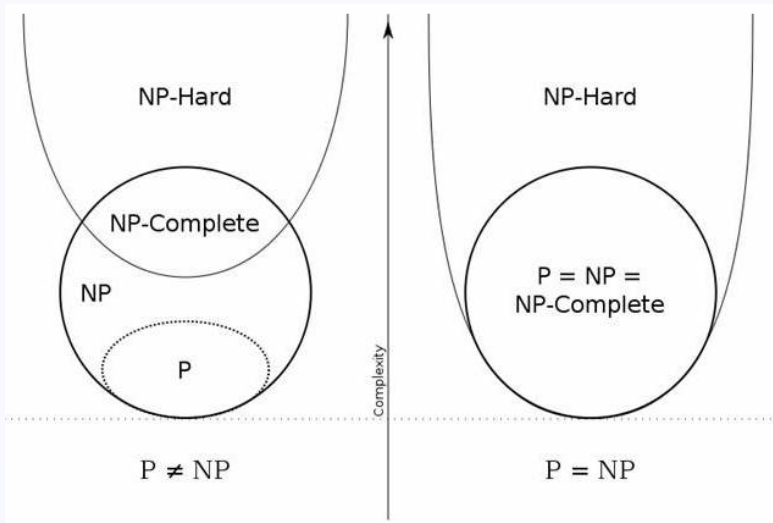
If we "guess" S then we can check whether it is a vertex cover of G in $n^{O(1)}$ steps.

Clearly: P $\subseteq$ NP $\qquad$ 10^6\$-question: is it correct that P $\neq$ NP?

A problem Π is NP-hard if it is "as hard as" all problems in NP

A problem Π is NP-complete if it is NP-hard and belongs in NP

Vertex Cover is an NP-complete problem



Parameterized Complexity

- Most of interesting problems are NP-hard!
- We care whether this situation can be "improved", taking into account structural characteristics of the input graph
  (for instance: topological properties).

Parameterized Complexity

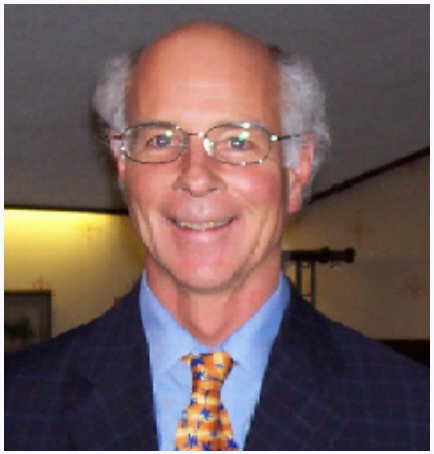
- Most of interesting problems are NP-hard!
- We care whether this situation can be "improved", taking into account structural characteristics of the input graph (for instance: topological properties).

We have to define what "improved" might mean here!

Parameterized Complexity

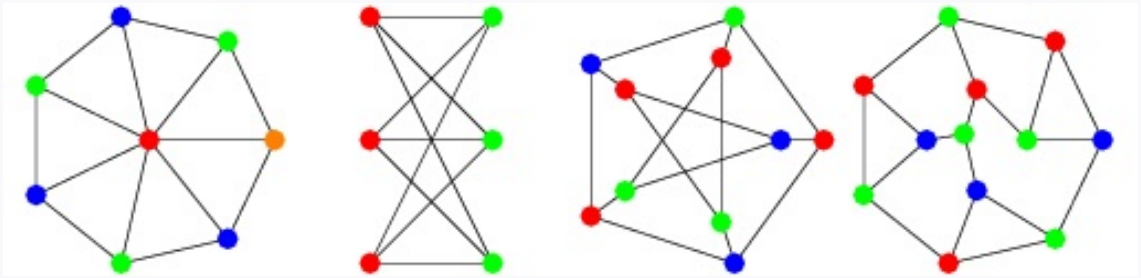
- Most of interesting problems are NP-hard!
- We care whether this situation can be "improved", taking into account structural characteristics of the input graph (for instance: topological properties).

We have to define what "improved" might mean here!



Parameterized complexity was introduced by Mike Fellows and Rod Downey proposed a way to refine the above landscape!

Three NP-complete problems

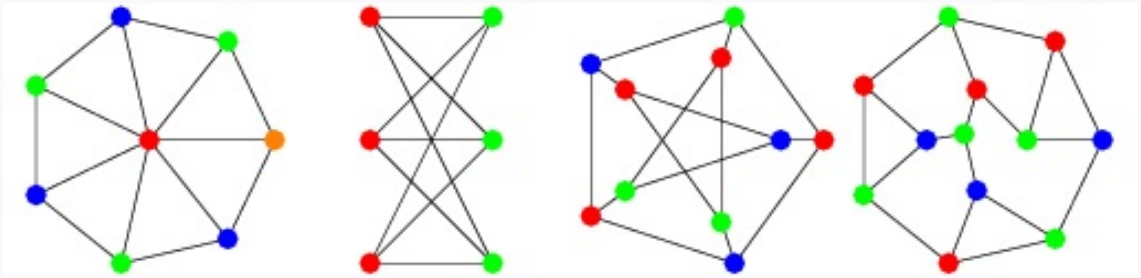


VERTEX COLORING

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists \sigma : V(G) \to \{1, \ldots, k\} : \forall \{v, u\} \in E(G)\ \sigma(v) \neq \sigma(u)$?

Three NP-complete problems



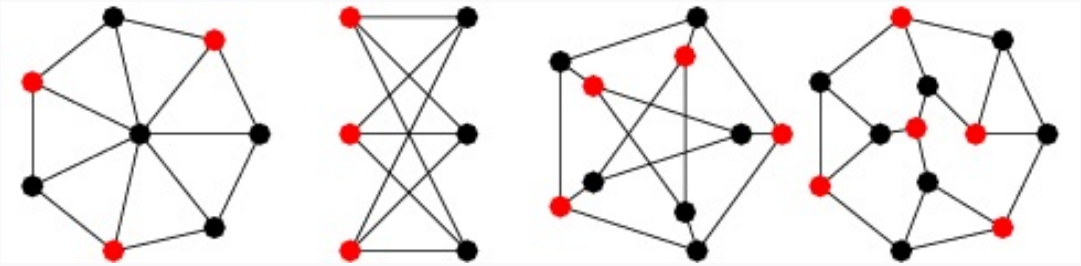
VERTEX COLORING

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists \sigma : V(G) \to \{1, \ldots, k\} : \forall \{v, u\} \in E(G)\ \sigma(v) \neq \sigma(u)$?

It can be solved in $O(n^2 \cdot k^n)$ steps

Three NP-complete problems

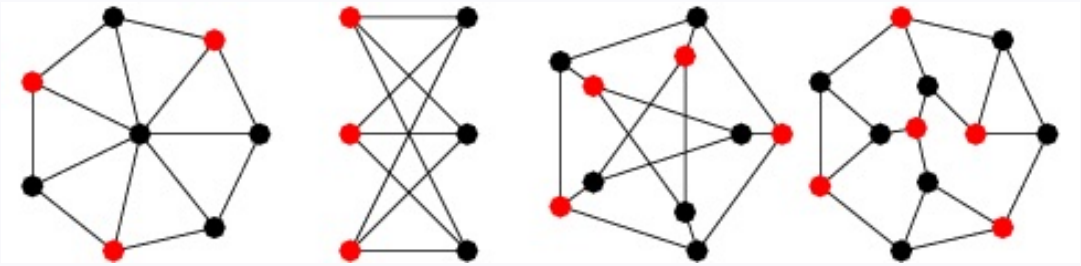


INDEPENDENT SET

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \geq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \leq 1$?

Three NP-complete problems



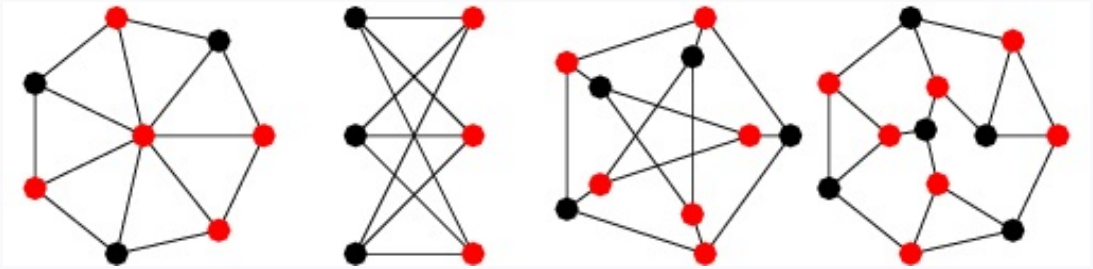
INDEPENDENT SET

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \geq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \leq 1$?

It can be solved in $O(n^{k+1})$ steps

Three NP-complete problems

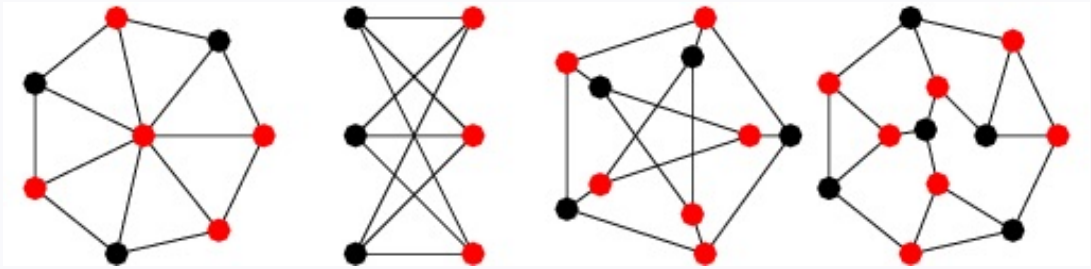


VERTEX COVER

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G) \; |e \cap S| \geq 1$?

Three NP-complete problems



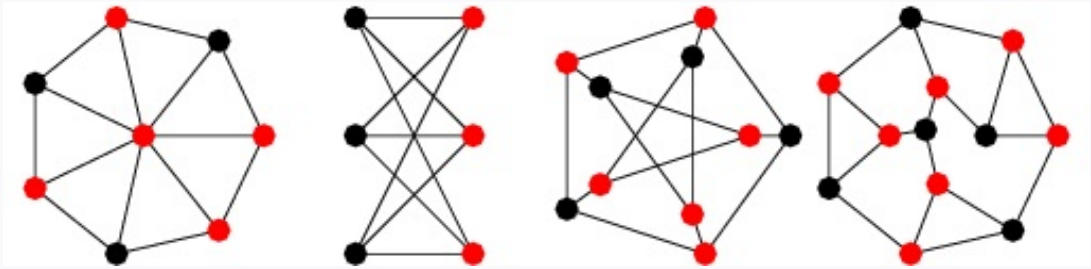
Vertex Cover

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G) \; |e \cap S| \geq 1$?

It can easily be solved in $O(2^k \cdot n)$ steps

Three NP-complete problems



Vertex Cover

Instance: A graph G and an integer $k \geq 0$.

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall e \in E(G)\ |e \cap S| \geq 1$?

It can easily be solved in $O(2^k \cdot n)$ steps

It can be solved in $O(1.2738^k + k \cdot n)$ steps [Chen, Kanj, Xia, 2010]

Comparisons

Summary:

Vertex Coloring	$O(n^2 \cdot k^n)$	
Independent Set	$O(n^{k+1})$	
Vertex Cover	$O(2^k \cdot n)$	

Different Interleavings between the parameter k and the main part n of the input.

Comparisons

Summary:

Vertex Coloring	$O(n^2 \cdot k^n)$	
Independent Set	$O(n^{k+1})$	
Vertex Cover	$O(2^k \cdot n)$	good

Different Interleavings between the parameter k and the main part n of the input.

Comparisons

Summary:

Vertex Coloring	$O(n^2 \cdot k^n)$	
Independent Set	$O(n^{k+1})$	bad
Vertex Cover	$O(2^k \cdot n)$	good

Different Interleavings between the parameter k and the main part n of the input.

Comparisons

Summary:

Vertex Coloring	$O(n^2 \cdot k^n)$	ugly
Independent Set	$O(n^{k+1})$	bad
Vertex Cover	$O(2^k \cdot n)$	good

Different Interleavings between the parameter k and the main part n of the input.

Comparison between $O(2^k \cdot n)$ and $O(n^{k+1})$

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2.500	5.625
$k = 3$	15.625	125.000	421.875
$k = 5$	390.625	6.250.000	31.640.623
$k = 10$	$1,9 \times 10^{12}$	$9,8 \times 10^{14}$	$3,7 \times 10^{16}$
$k = 20$	$1,8 \times 10^{26}$	$9,5 \times 10^{31}$	$2,1 \times 10^{35}$

The ratio $\frac{n^{k+1}}{2^k \cdot n}$ for several values of n and k.

How the parameters appear?

VLIf design: In VLSI chip construction, the number of circuit layers is no more than 10. While the problem is, in general, NP-complete, when we fix the number of layers, it becomes tractable.

How the parameters appear?

VLIf design: In VLSI chip construction, the number of circuit layers is no more than 10. While the problem is, in general, NP-complete, when we fix the number of layers, it becomes tractable.

Computational Biology: in general, many problems in DNA chain reconstruction are intractable. In the majority of the cases, real instances have special properties (e.g., bounded treewidth or pathwidth – by 11) that facilitate the design of efficient algortihms.

How the parameters appear?

Robotics: The number of degrees of freedom in motion planning problems are not more than 10. While these problems are NP-complete in general, the become tractable taking into account this natural restriction.

How the parameters appear?

Robotics: The number of degrees of freedom in motion planning problems are not more than 10. While these problems are NP-complete in general, the become tractable taking into account this natural restriction.

Compilers: One of the main tasks of a compiler for the language ML is the compatibility checking of type declarations of the program. It is known that the general problem is EXP-complete. However, in real cases, the implementations work well as there is an algorithm with complexity $O(2^k \cdot n)$, where n is the size of the program and k is the depth of its type declarations. As, normally, $k \leq 10$, the problem can be considered tractable.

Parameterized problems

Given an alphabet Σ,

Parameterized problems

Given an alphabet Σ,

(1) A *parameterization* of Σ^* is a recursive function $\kappa : \Sigma^* \to \mathbb{N}$

Parameterized problems

Given an alphabet Σ,

(1) A *parameterization* of Σ^* is a recursive function $\kappa : \Sigma^* \to \mathbb{N}$

(2) A *parameterized problem* (with respect to Σ) is a pair (Π, κ) where $\Pi \subseteq \Sigma^*$ and κ is a parameterization of Σ^*.

Examples

A parameterization of Independent Set can be defined as $\kappa(G, k) = k$.

Examples

A parameterization of Independent Set can be defined as $\kappa(G, k) = k$.

We can do the same with all the problems that have some integer in their instances, such as Vertex Coloring and Vertex Cover.

That way, we define the parameterized problems p-Vertex Coloring and p-Vertex Cover.

Examples

A parameterization of Independent Set can be defined as $\kappa(G, k) = k$.

We can do the same with all the problems that have some integer in their instances, such as Vertex Coloring and Vertex Cover.

That way, we define the parameterized problems p-Vertex Coloring and p-Vertex Cover.

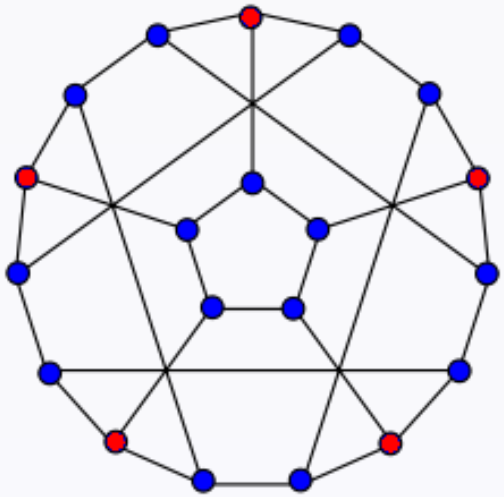
Other parameterizations of the above problems can be defined as

$\kappa(G, k) = \Delta(G)$ or

$\kappa(G, k) = \mathbf{genus}(G)$

$\kappa(G, k) = \Delta(G) + k$

Some parameterized problems



p-Dominating Set

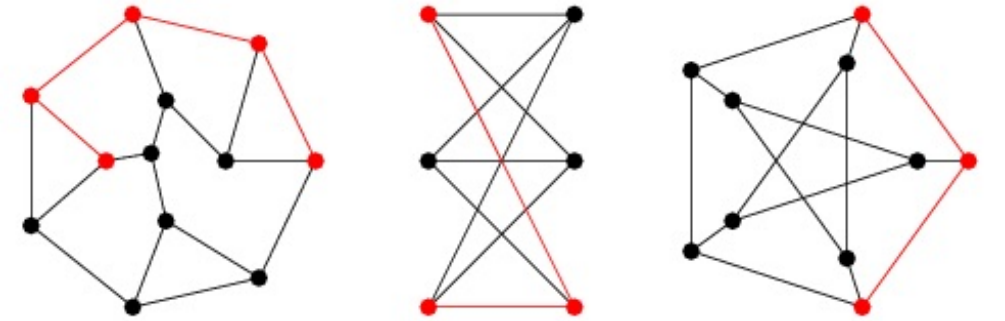
Instance: A graph G and an integer $k \geq 0$

Parameter: k,

Question:

$\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall v \in V(G) - S \; \exists u \in S \; \{v, u\} \in E(G)$?

Some parameterized problems



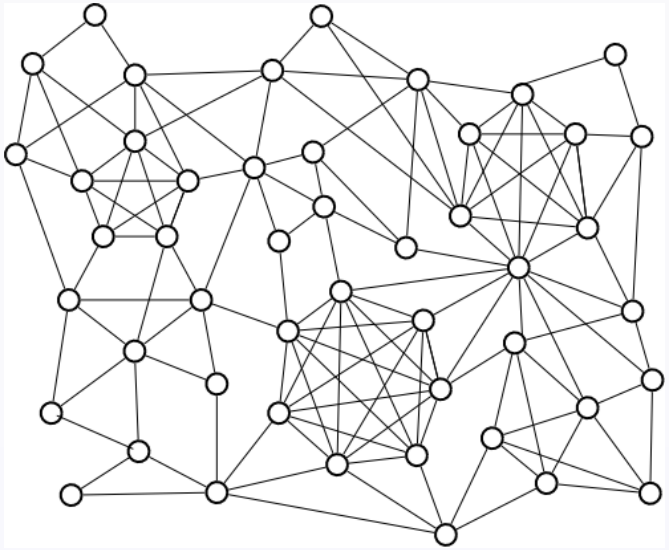
p-PATH

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: Does G contain a path of length k?

Some parameterized problems



p-Clique

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: $\exists S \in V(G) : |S| \leq k \quad \wedge \quad \forall v, u \in S \ \{v, u\} \in E(G)$?

More parameterized problems

p-Steiner Tree

Instance: A graph G, $S \subseteq V(G)$, $k \in \mathbb{N}$.

Parameter: k

Question: $\exists R \in V(G) : |R| \leq k$, $R \cap S = \emptyset$, $G[S \cup R]$ is connected?

Here $\kappa(G, S, k) = k$

More parameterized problems

p-Steiner Tree

Instance: A graph G, $S \subseteq V(G)$, $k \in \mathbb{N}$.

Parameter: k

Question: $\exists R \in V(G) : |R| \leq k$, $R \cap S = \emptyset$, $G[S \cup R]$ is connected?

Here $\kappa(G, S, k) = k$

p'-Steiner Tree

Instance: A graph G, $S \subseteq V(G)$, $k \in \mathbb{N}$.

Parameter: $|S|$

Question: $\exists R \in V(G) : |R| \leq k$, $R \cap S = \emptyset$, $G[S \cup R]$ is connected?

Here $\kappa(G, S, k) = |S|$

The class FPT

Given an alphabet Σ and a parameterization $\kappa : \Sigma^* \to \mathbb{N}$,

The class FPT

Given an alphabet Σ and a parameterization $\kappa : \Sigma^* \to \mathbb{N}$,

(a) An algorithm A is a FPT-*algorithm with respect to* κ if there is a function computable $f : \mathbb{N} \to \mathbb{N}$ and a polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$, the algorithm A requires

$$\leq f(\kappa(x)) \cdot p(|x|) \text{ steps}$$

The class FPT

Given an alphabet Σ and a parameterization $\kappa : \Sigma^* \to \mathbb{N}$,

(a) An algorithm A is a FPT-*algorithm with respect to* κ if there is a function computable $f : \mathbb{N} \to \mathbb{N}$ and a polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$, the algorithm A requires

$$\leq f(\kappa(x)) \cdot p(|x|) \text{ steps}$$

(b) A parameterized problem (L, κ) is *fixed parameter tractable* if there exists an FPT-algoritm with respect to κ that decides L.

The class FPT

Given an alphabet Σ and a parameterization $\kappa : \Sigma^* \to \mathbb{N}$,

(a) An algorithm A is a FPT-*algorithm with respect to* κ if there is a function computable $f : \mathbb{N} \to \mathbb{N}$ and a polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$, the algorithm A requires

$$\leq f(\kappa(x)) \cdot p(|x|) \text{ steps}$$

(b) A parameterized problem (L, κ) is *fixed parameter tractable* if there exists an FPT-algoritm with respect to κ that decides L.

- We then say that $(L, \kappa) \in$ FPT or, more precisely, f-FPT

The class FPT

Given an alphabet Σ and a parameterization $\kappa : \Sigma^* \to \mathbb{N}$,

(a) An algorithm A is a FPT-*algorithm with respect to* κ if there is a function computable $f : \mathbb{N} \to \mathbb{N}$ and a polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$, the algorithm A requires

$$\leq f(\kappa(x)) \cdot p(|x|) \text{ steps}$$

(b) A parameterized problem (L, κ) is *fixed parameter tractable* if there exists an FPT-algoritm with respect to κ that decides L.

- We then say that $(L, \kappa) \in$ FPT or, more precisely, f-FPT
- The function f is called *parameterized dependence* of the running time of the FPT-algoritm

An algorithm for Vertex Cover:

We set up a search tree with depth depending *only* on the parameter k.

[Bounded Search Tree Method]

An algorithm for Vertex Cover:

We set up a search tree with depth depending *only* on the parameter k.

[Bounded Search Tree Method]

---

algvc(G, k)

1. If $|E(G)| = 0$, then return "YES"

2. If $k = 0$, then return "NO"

3. choose (arbitrarily) an edge $e = \{v, u\} \in E(G)$ and

return algvc$(G - v, k - 1) \bigvee$ algvc$(G - u, k - 1)$

An algorithm for Vertex Cover:

We set up a search tree with depth depending *only* on the parameter k.

[Bounded Search Tree Method]

---

algvc(G, k)

1. If $|E(G)| = 0$, then return "YES"

2. If $k = 0$, then return "NO"

3. choose (arbitrarily) an edge $e = \{v, u\} \in E(G)$ and

return algvc$(G - v, k - 1) \bigvee$ algvc$(G - u, k - 1)$

Recursive calls: 2, Depth of the recursion: k,

Time in the leaves of the recursion: $O(n)$ steps

An algorithm for Vertex Cover:

We set up a search tree with depth depending *only* on the parameter k.

[Bounded Search Tree Method]

---

algvc(G, k)

1. If $|E(G)| = 0$, then return "YES"

2. If $k = 0$, then return "NO"

3. choose (arbitrarily) an edge $e = \{v, u\} \in E(G)$ and

return algvc$(G - v, k - 1) \bigvee$ algvc$(G - u, k - 1)$

Recursive calls: 2, Depth of the recursion: k,

Time in the leaves of the recursion: $O(n)$ steps

Total time: $O(2^k \cdot n)$ steps.

An algorithm for Vertex Cover:

We set up a search tree with depth depending *only* on the parameter k.

[Bounded Search Tree Method]

---

algvc(G, k)

1. If $|E(G)| = 0$, then return "YES"

2. If $k = 0$, then return "NO"

3. choose (arbitrarily) an edge $e = \{v, u\} \in E(G)$ and

return algvc$(G - v, k - 1) \bigvee$algvc$(G - u, k - 1)$

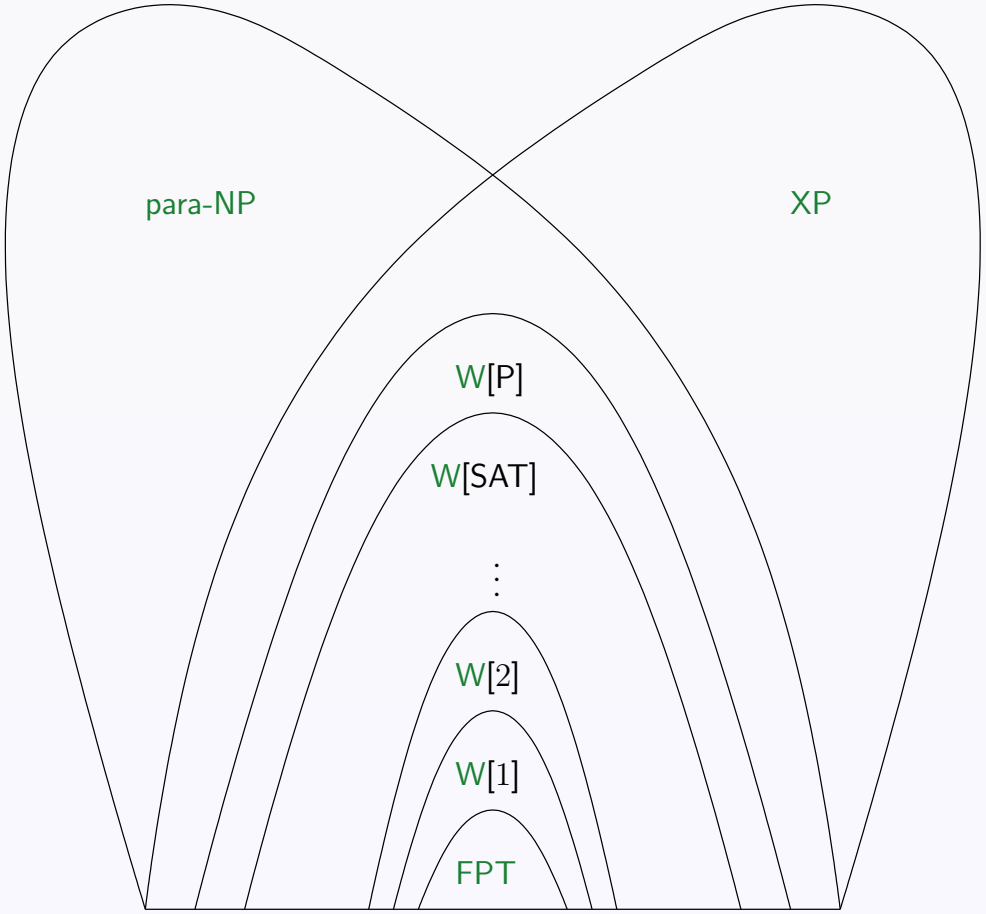
Recursive calls: 2, Depth of the recursion: k,

Time in the leaves of the recursion: $O(n)$ steps

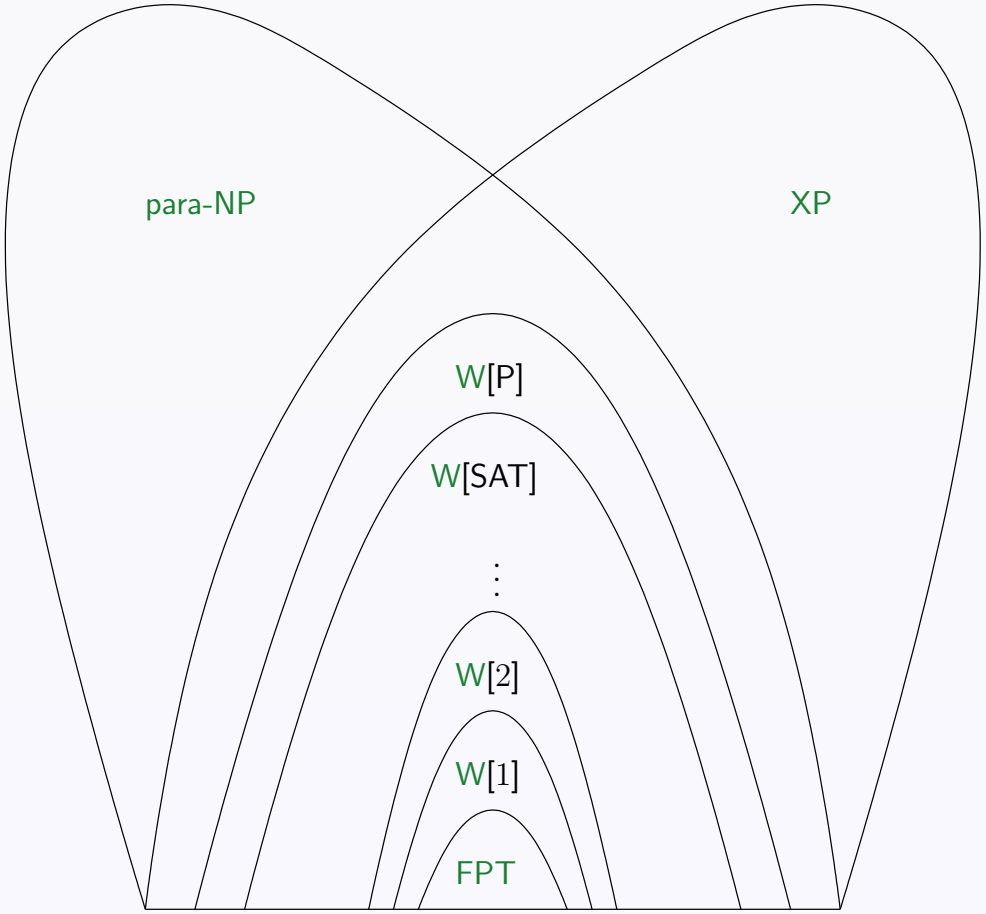
Total time: $O(2^k \cdot n)$ steps.

Therefore, p-Vertex Cover$\in 2^{O(k)}$-FPT.

Panorama of Parameterized complexity classes

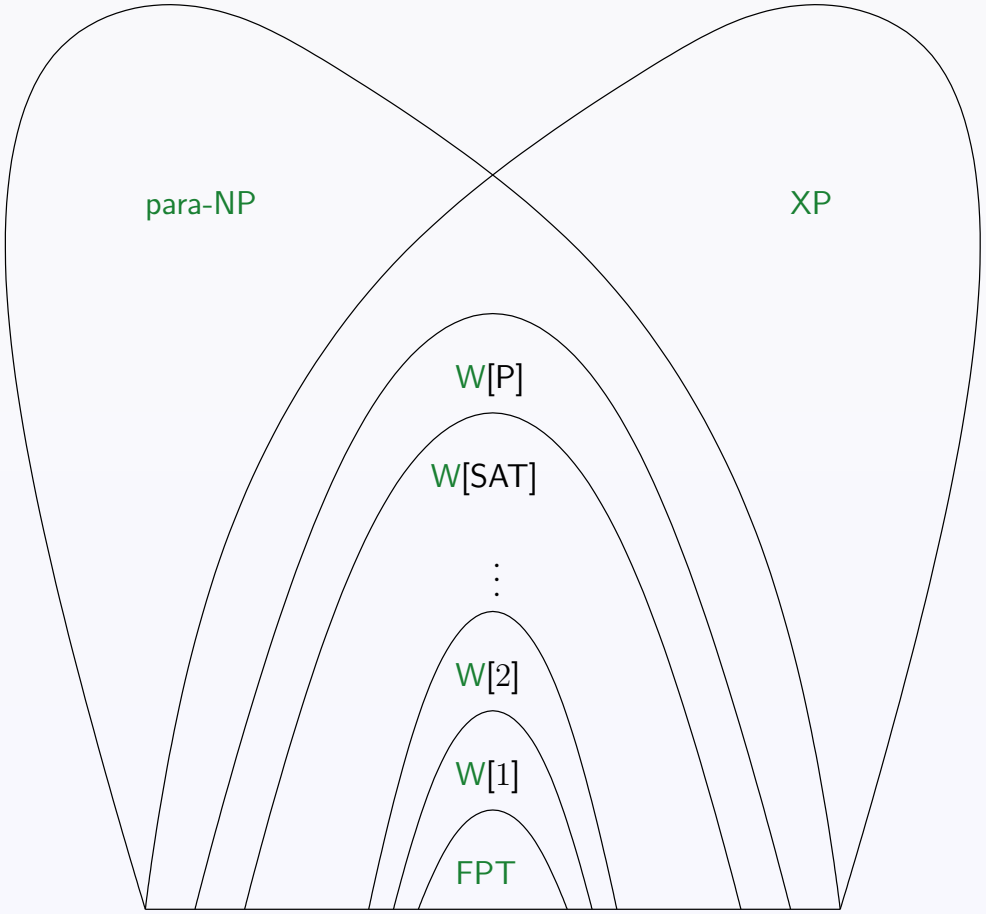


Panorama of Parameterized complexity classes



p-Vertex Cover: FPT

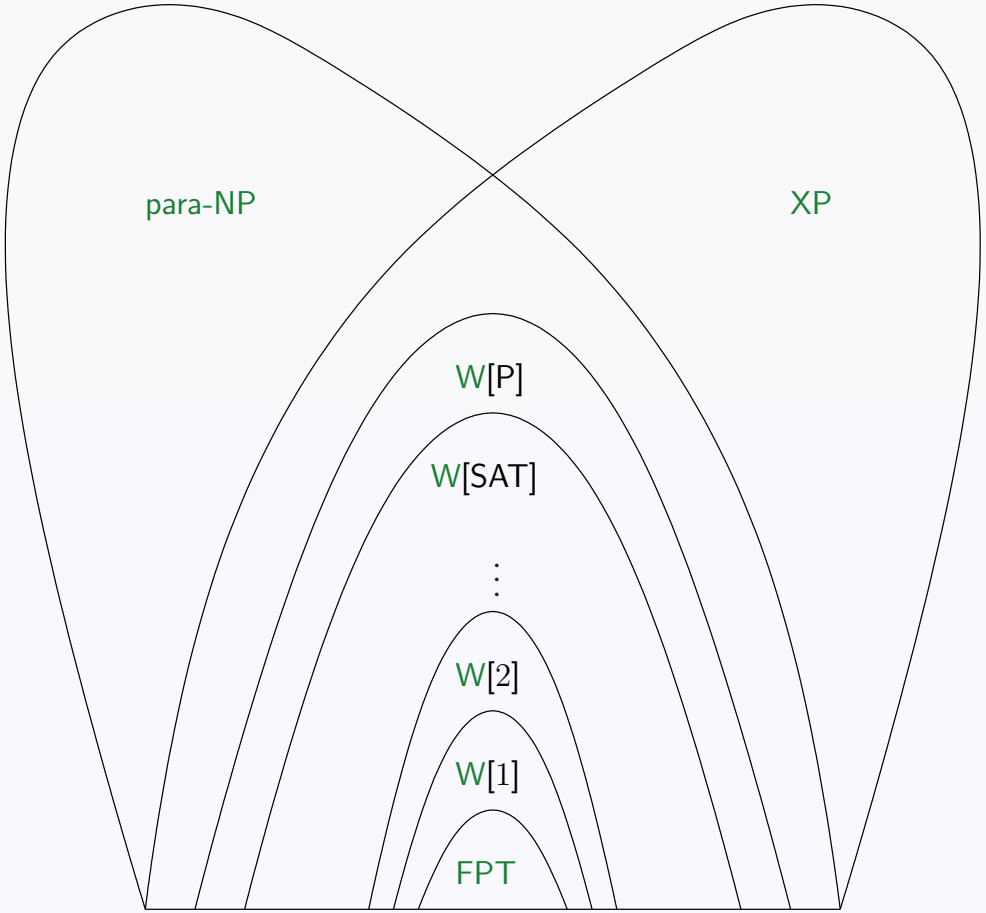
Panorama of Parameterized complexity classes



p-Vertex Cover: FPT

p-Path: FPT

Panorama of Parameterized complexity classes

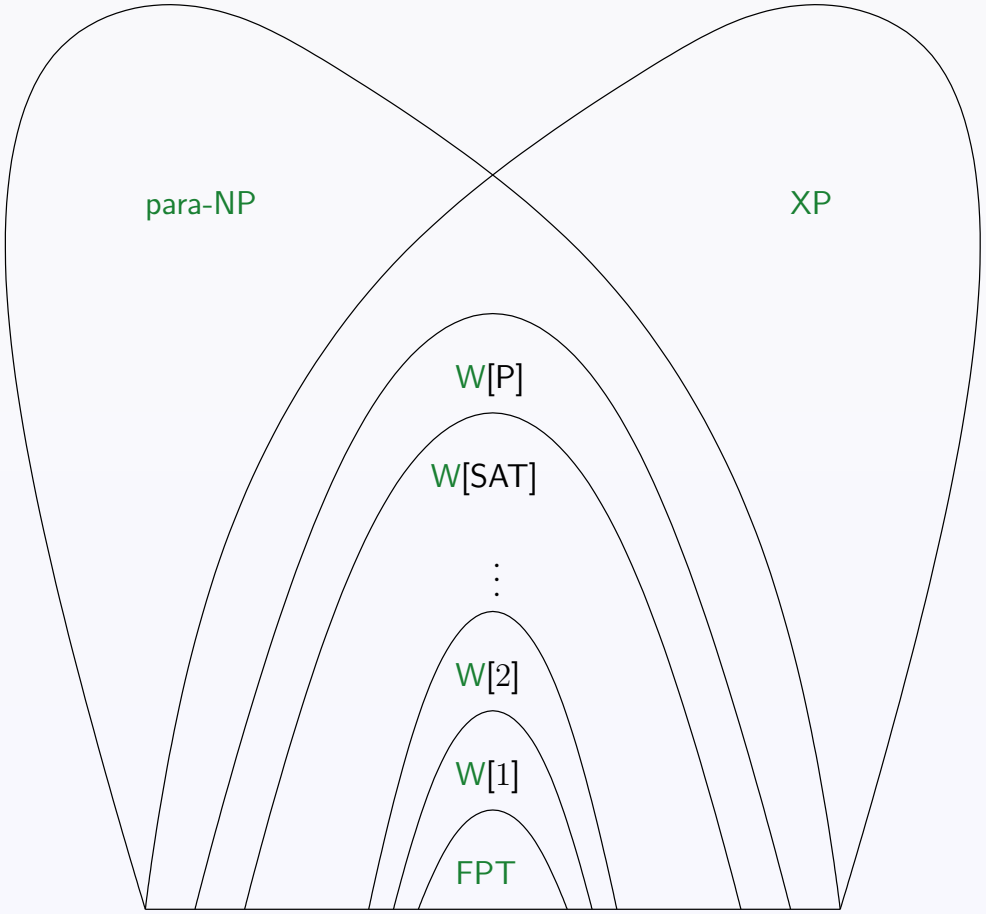


p-Vertex Cover: FPT

p-Path: FPT

p'-Steiner Tree: FPT

Panorama of Parameterized complexity classes



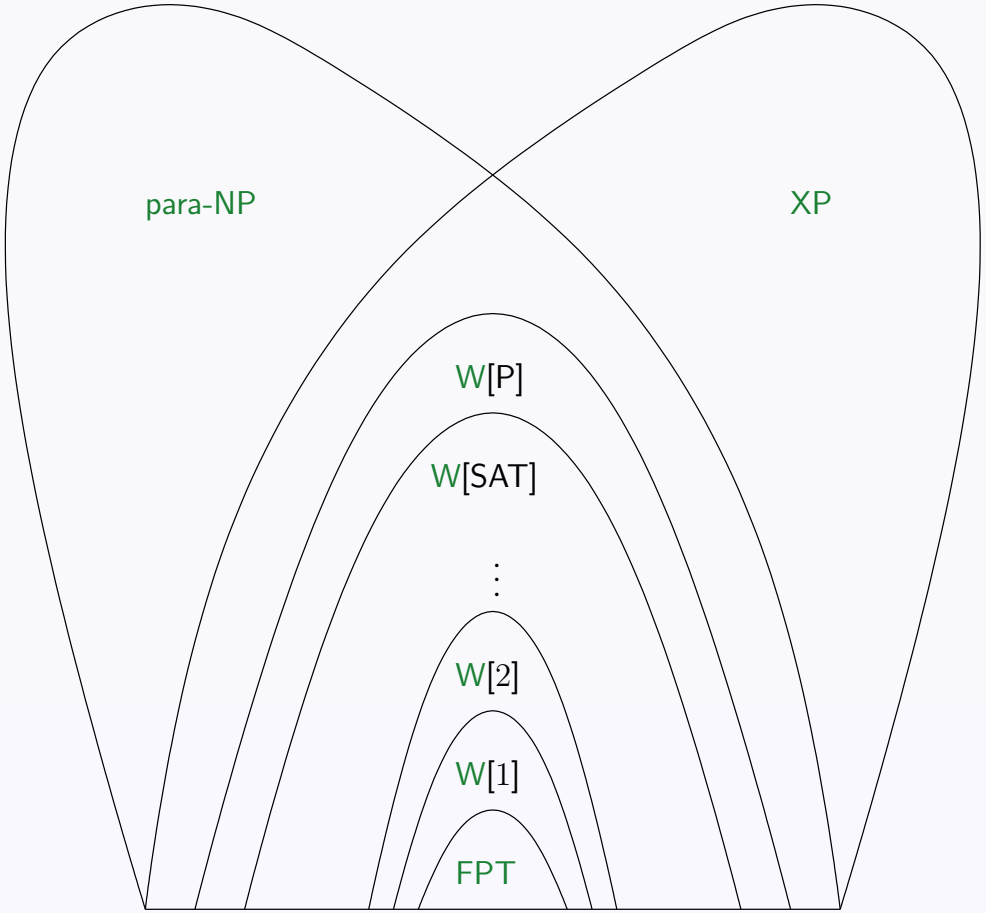
p-Vertex Cover: FPT

p-Path: FPT

p'-Steiner Tree: FPT

p-Clique: W[1]-complete

Panorama of Parameterized complexity classes



p-Vertex Cover: FPT

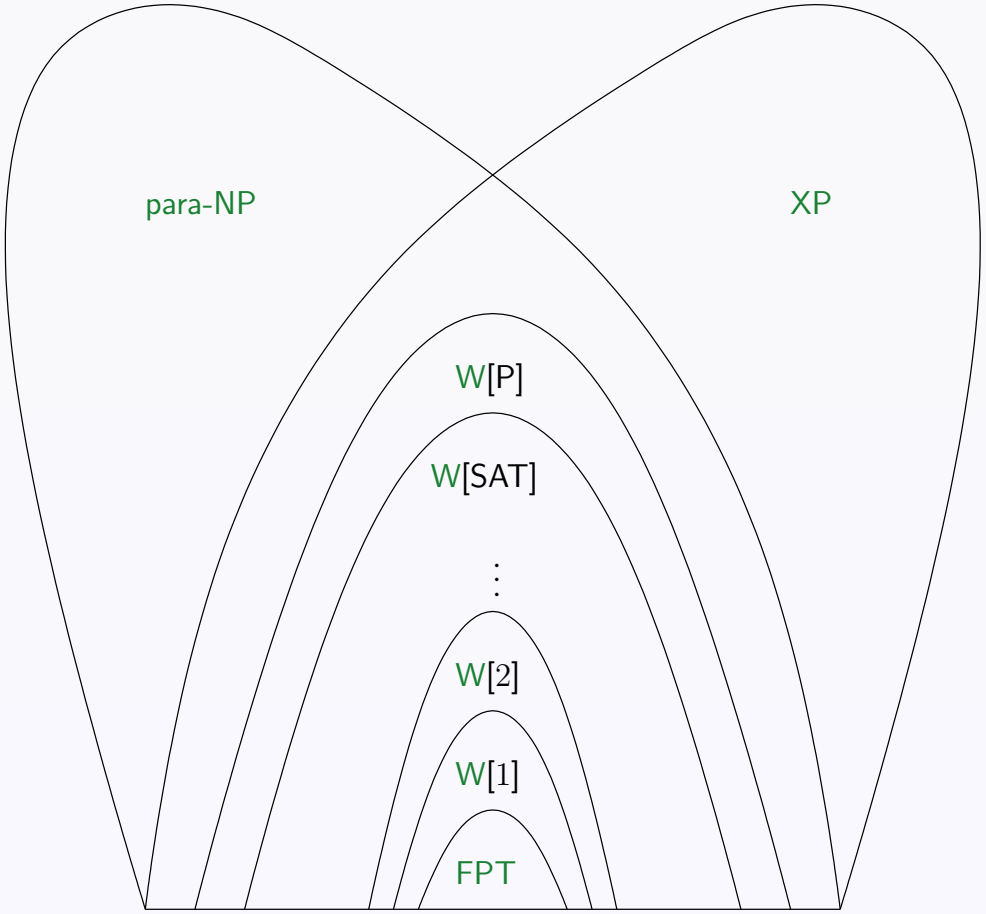
p-Path: FPT

p'-Steiner Tree: FPT

p-Clique: W[1]-complete

p-Independent Set: W[1]-complete

Panorama of Parameterized complexity classes



p-Vertex Cover: FPT

p-Path: FPT

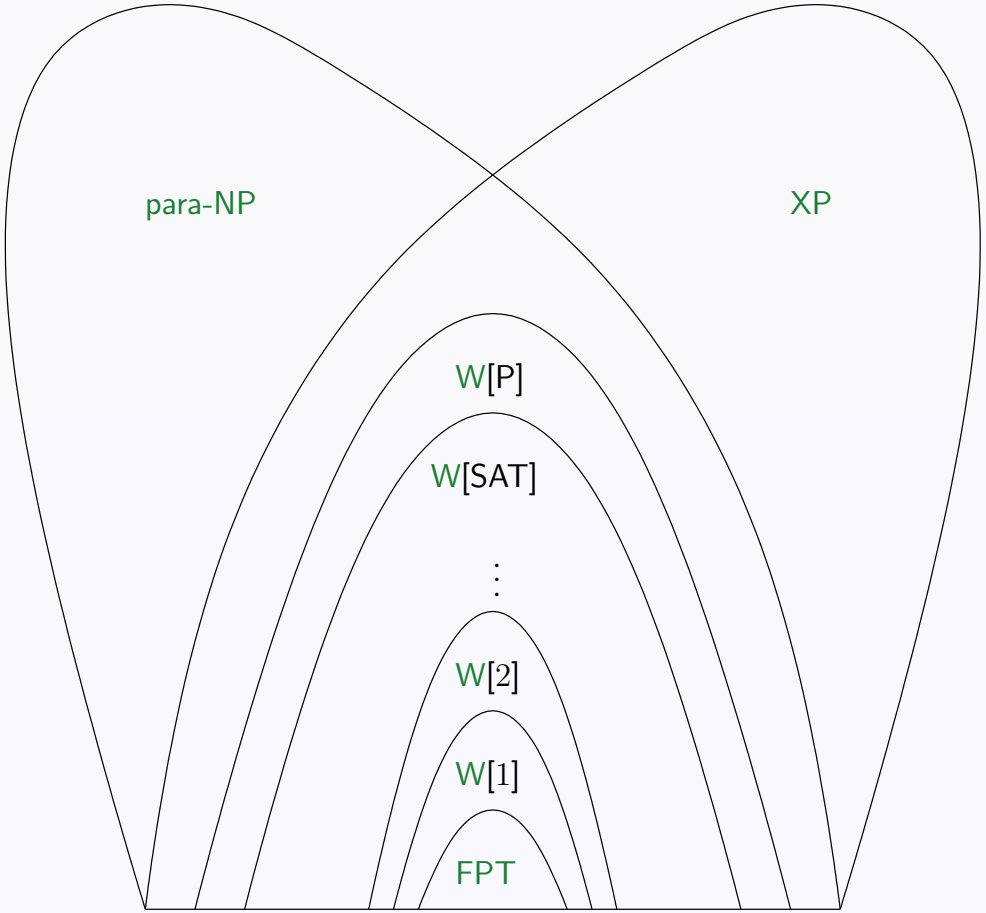
p'-Steiner Tree: FPT

p-Clique: W[1]-complete

p-Independent Set: W[1]-complete

p-Dominating Set: W[2]-complete

Panorama of Parameterized complexity classes



p-Vertex Cover: FPT

p-Path: FPT

p'-Steiner Tree: FPT

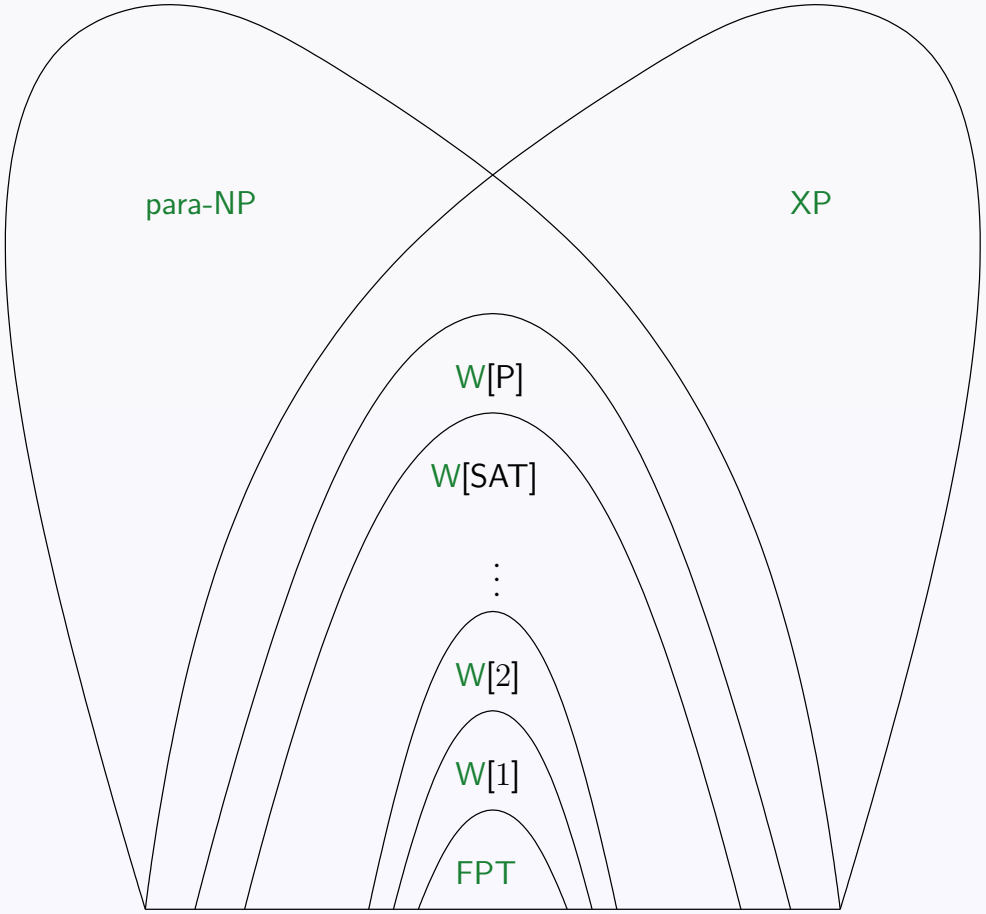
p-Clique: W[1]-complete

p-Independent Set: W[1]-complete

p-Dominating Set: W[2]-complete

p-Steiner Tree: W[2]-complete

Panorama of Parameterized complexity classes



p-Vertex Cover: FPT

p-Path: FPT

p'-Steiner Tree: FPT

p-Clique: W[1]-complete

p-Independent Set: W[1]-complete

p-Dominating Set: W[2]-complete

p-Steiner Tree: W[2]-complete

p-Coloring: para-NP-complete

Part 2, Monday 09/05/2016 - 16:00–17:30 (90')

Tree decompositions

Treewidth

Courcelle's Theorem

Dynamic programming

Tree decompositions

A *tree decomposition* (ou décomposition arborescente) of a graph G is a pair $D = (T, \mathcal{X})$ such that T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of G. such that:

Tree decompositions

A *tree decomposition* (ou décomposition arborescente) of a graph G is a pair $D = (T, \mathcal{X})$ such that T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of G. such that:

1. Any vertex $v \in V(G)$ and the end points of any edge $e \in E(G)$ belong in some node X_t of D

Tree decompositions

A *tree decomposition* (ou décomposition arborescente) of a graph G is a pair $D = (T, \mathcal{X})$ such that T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of G. such that:

1. Any vertex $v \in V(G)$ and the end points of any edge $e \in E(G)$ belong in some node X_t of D
2. For any $v \in V(G)$, the set $\{t \in V(T) \mid v \in X_t\}$ is a subtree of T.

Tree decompositions

A *tree decomposition* (ou décomposition arborescente) of a graph G is a pair $D = (T, \mathcal{X})$ such that T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of G. such that:

1. Any vertex $v \in V(G)$ and the end points of any edge $e \in E(G)$ belong in some node X_t of D
2. For any $v \in V(G)$, the set $\{t \in V(T) \mid v \in X_t\}$ is a subtree of T.

- $X_t \in \mathcal{X}$ corresponds to a vertex $t \in V(T)$ – X_t is a *node*/*bag* of D

Tree decompositions

A *tree decomposition* (ou décomposition arborescente) of a graph G is a pair $D = (T, \mathcal{X})$ such that T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of G. such that:

1. Any vertex $v \in V(G)$ and the end points of any edge $e \in E(G)$ belong in some node X_t of D
2. For any $v \in V(G)$, the set $\{t \in V(T) \mid v \in X_t\}$ is a subtree of T.

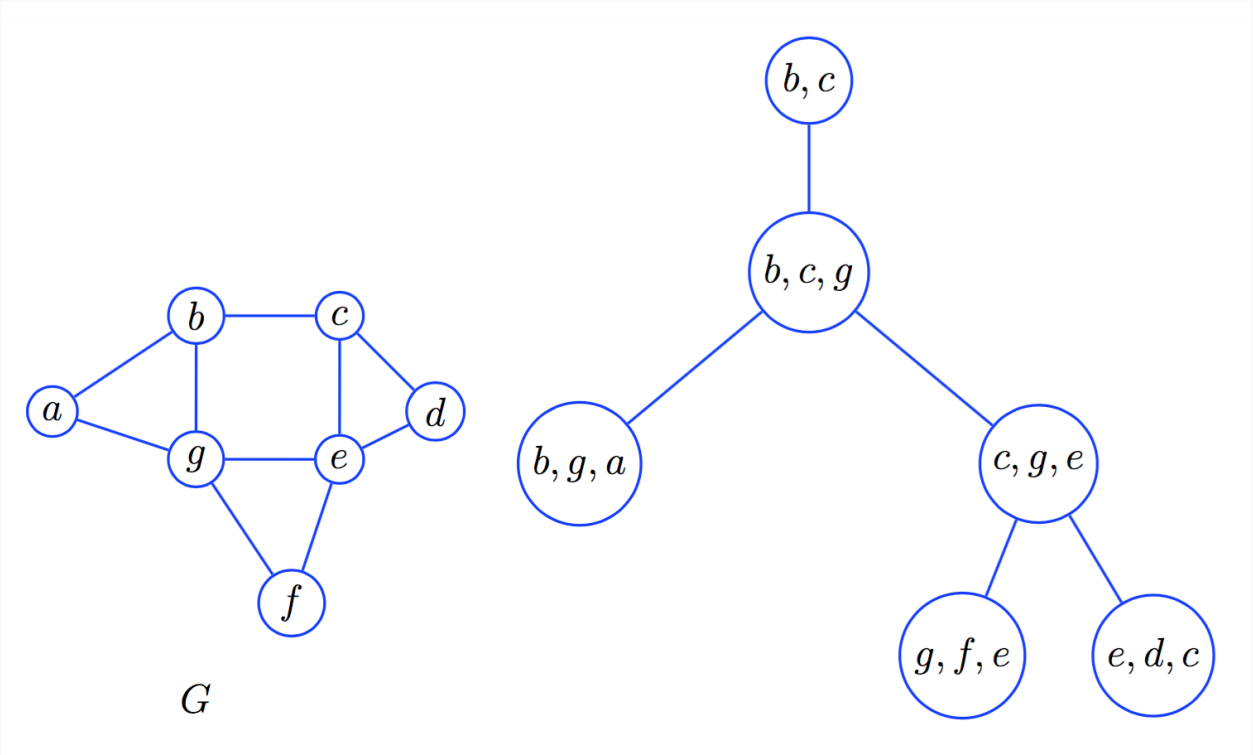
- $X_t \in \mathcal{X}$ corresponds to a vertex $t \in V(T)$ – X_t is a *node/bag* of D
- The *width* of a tree decomposition $(T, \mathcal{X})$ is $\max_{t \in V(T)} |X_t| - 1$

Tree decompositions

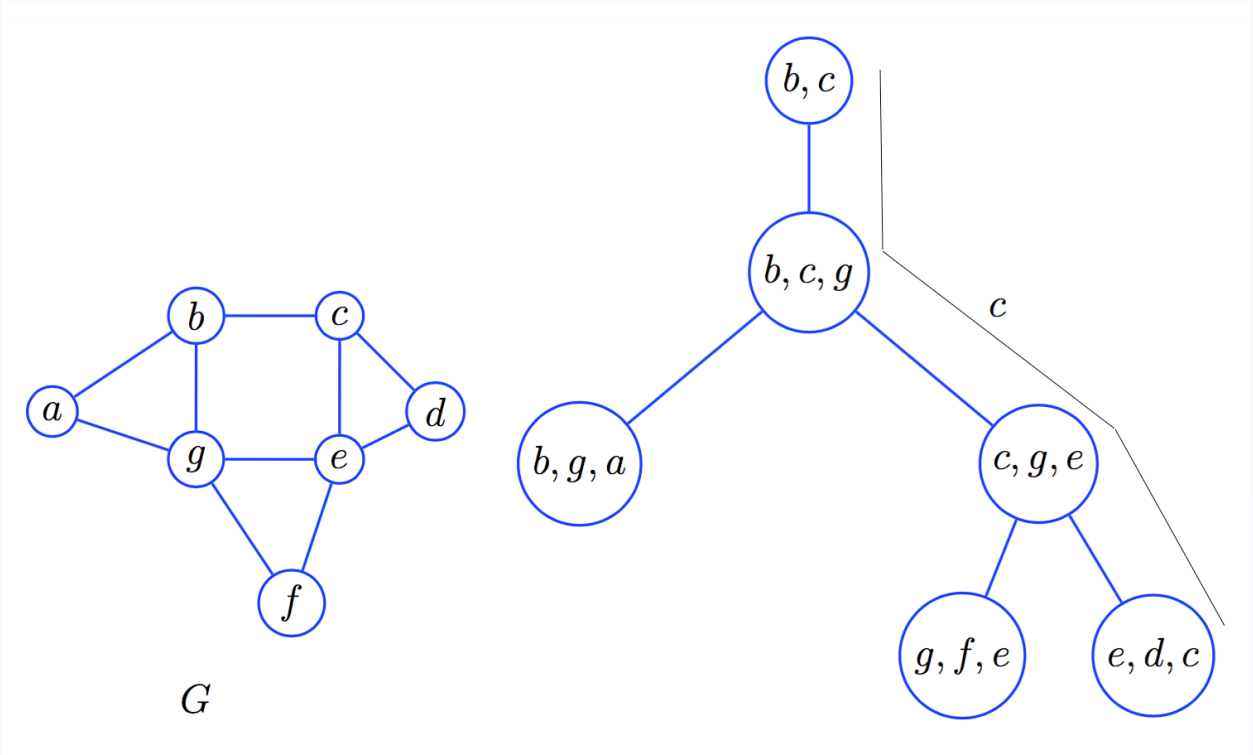
A *tree decomposition* (ou décomposition arborescente) of a graph G is a pair $D = (T, \mathcal{X})$ such that T is a tree and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of G. such that:

1. Any vertex $v \in V(G)$ and the end points of any edge $e \in E(G)$ belong in some node X_t of D
2. For any $v \in V(G)$, the set $\{t \in V(T) \mid v \in X_t\}$ is a subtree of T.

- $X_t \in \mathcal{X}$ corresponds to a vertex $t \in V(T)$ – X_t is a *node/bag* of D
- The *width* of a tree decomposition $(T, \mathcal{X})$ is $\max_{t \in V(T)} |X_t| - 1$
- The *tree-width* (ou largeur arborescente ou largeur d'arbre) of a graph G (**tw**(G)) is the minimum width over all tree decompositions of G



Each vertex of G has a continuous "trace" in the tree of the tree decomposition



Each vertex of G has a continuous "trace" in the tree of the tree decomposition

Another definition for Treewidth

- A vertex in G is k-*simplicial* if its neighborhood induces a k-clique.

Another definition for Treewidth

- A vertex in G is k-*simplicial* if its neighborhood induces a k-clique.
- A graph G is a k-tree if one of the following holds

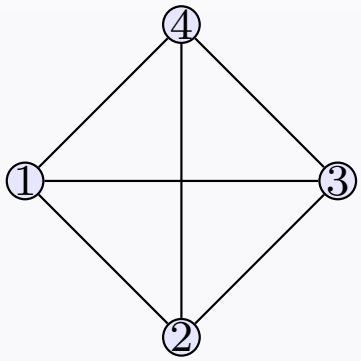
Another definition for Treewidth

- A vertex in G is k-*simplicial* if its neighborhood induces a k-clique.
- A graph G is a k-tree if one of the following holds
  - $G = K_{k+1}$ or
  - the removal of G of a k-simplicial vertex creates a k-tree.

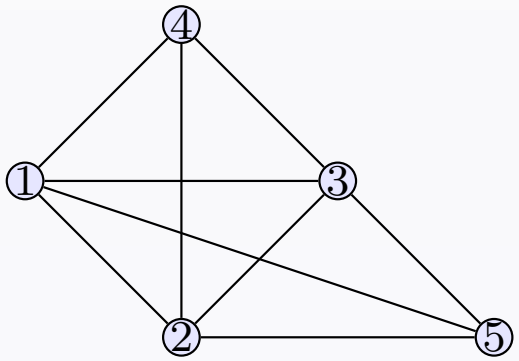
Another definition for Treewidth

- A vertex in G is k-*simplicial* if its neighborhood induces a k-clique.
- A graph G is a k-tree if one of the following holds
  - $G = K_{k+1}$ or
  - the removal of G of a k-simplicial vertex creates a k-tree.
- The treewidth of a graph G is defined as follows

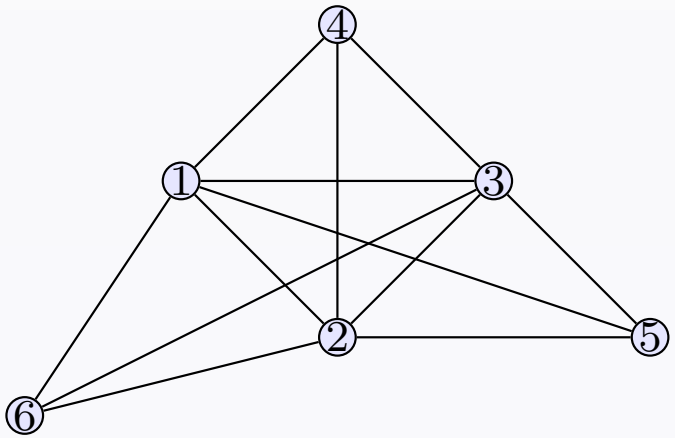
$$\mathbf{tw}(G) = \min\{k \mid G \text{ is a subgraph of some } k\text{-tree}\}$$



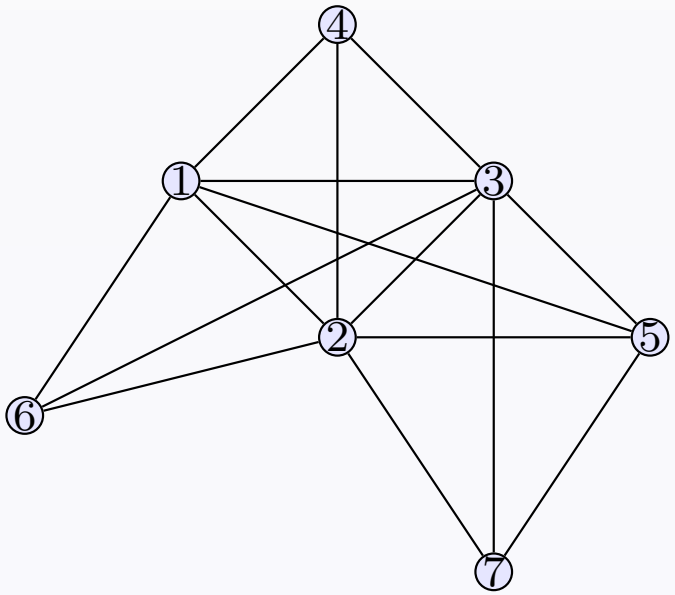
A 3-tree



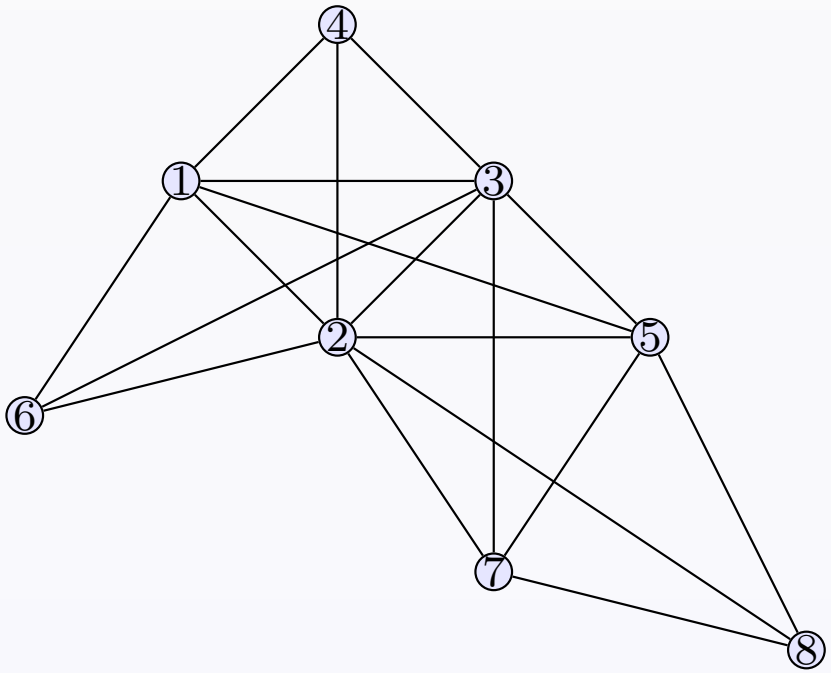
A 3-tree



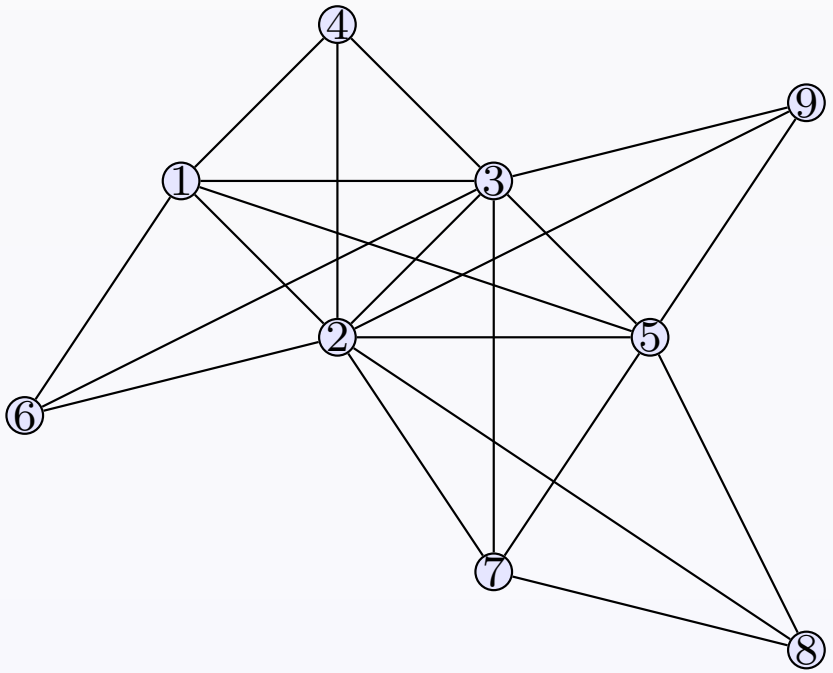
A 3-tree



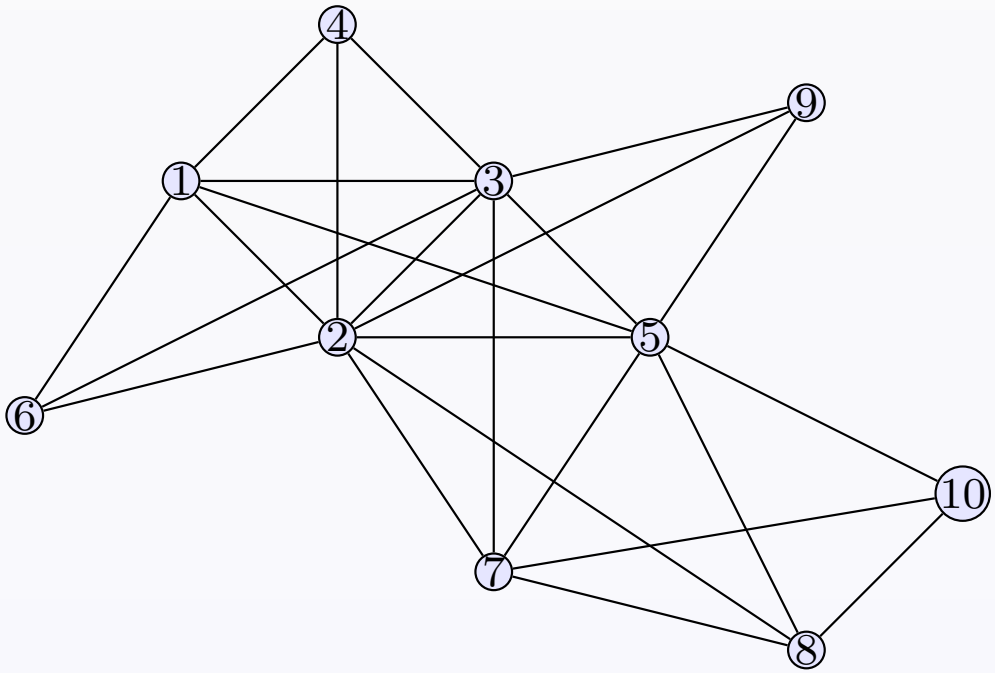
A 3-tree



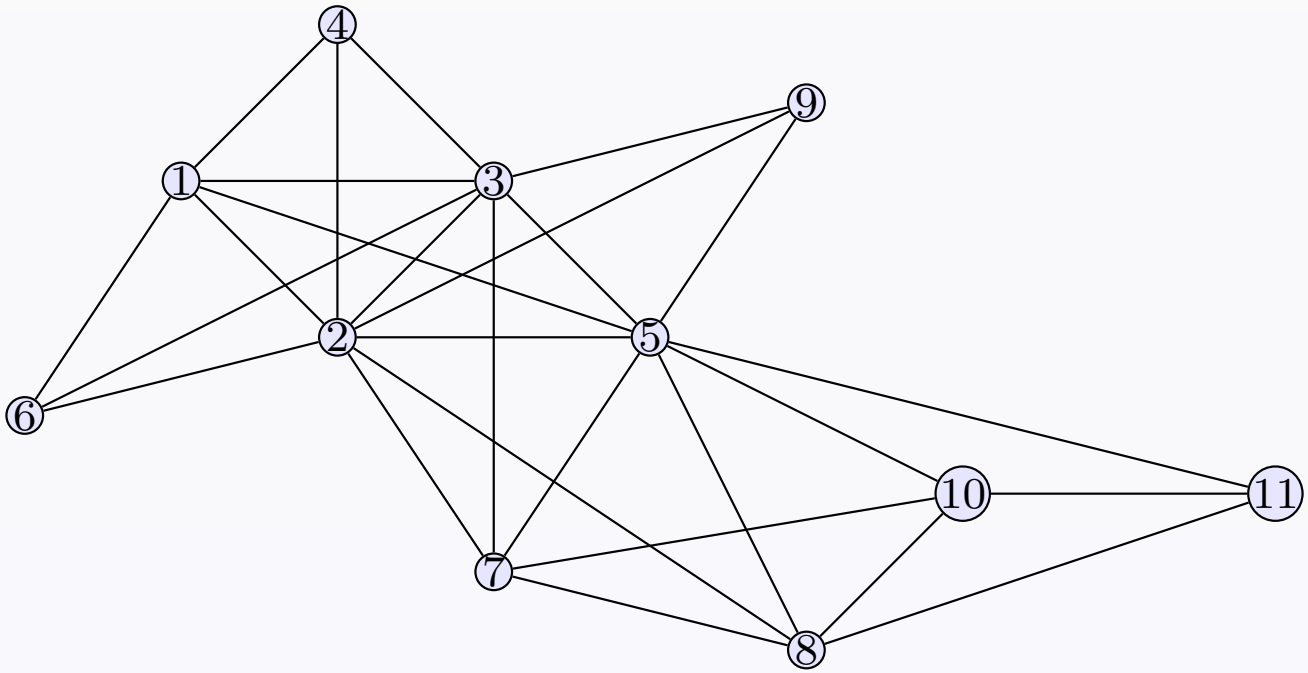
A 3-tree



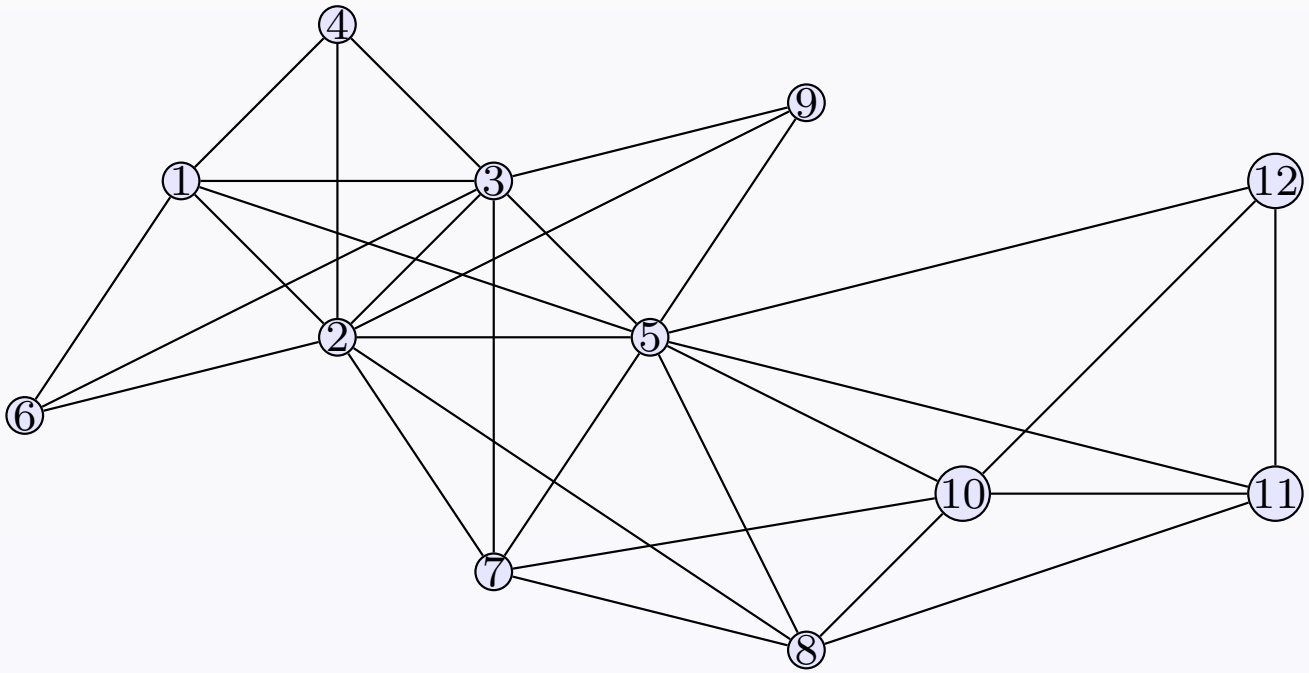
A 3-tree



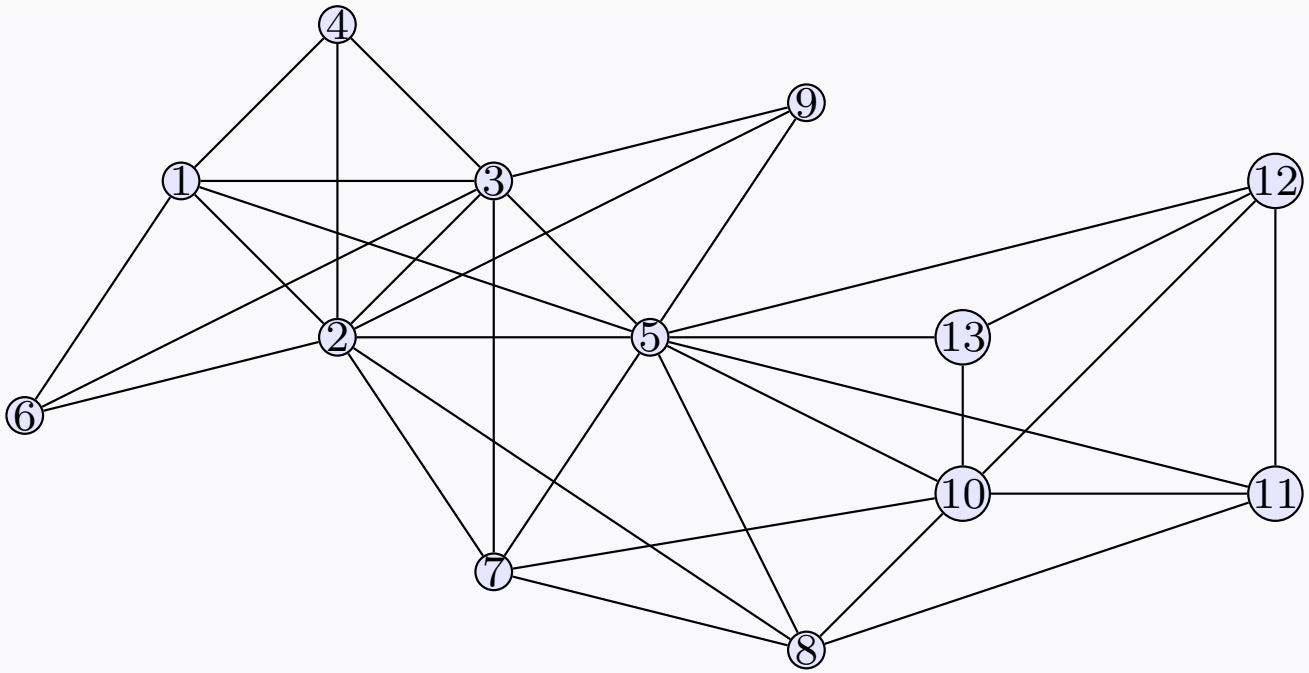
A 3-tree



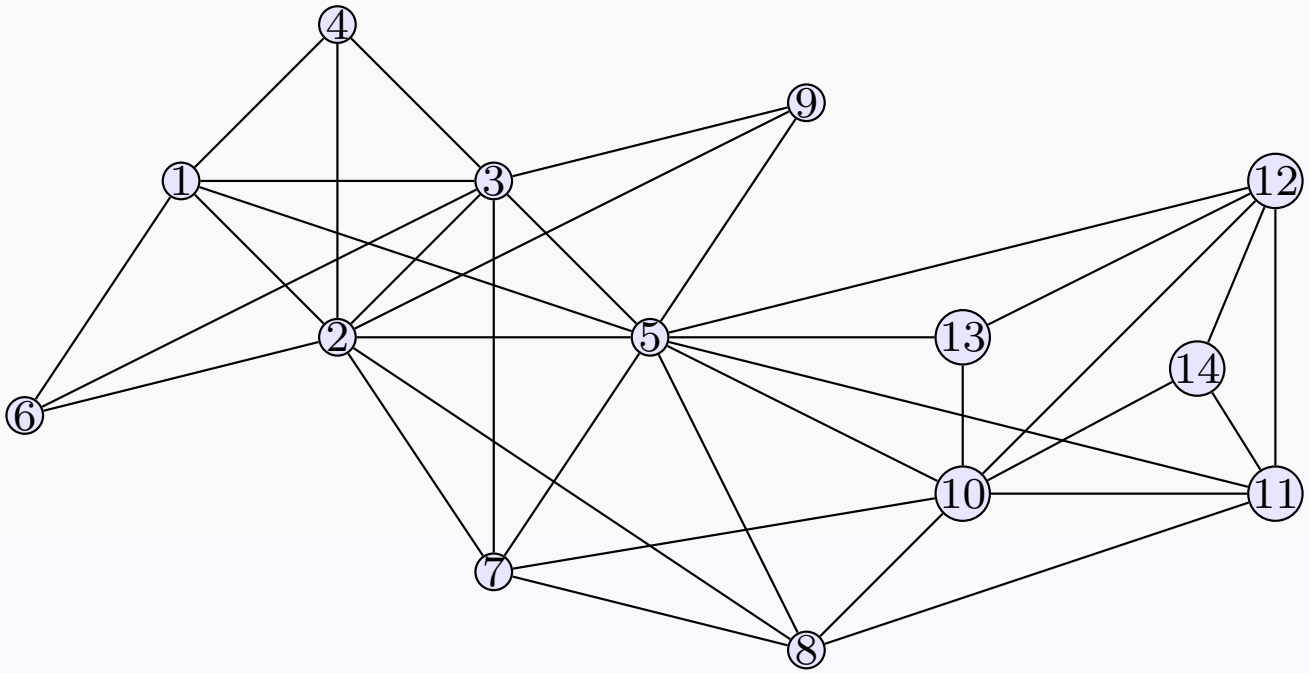
A 3-tree



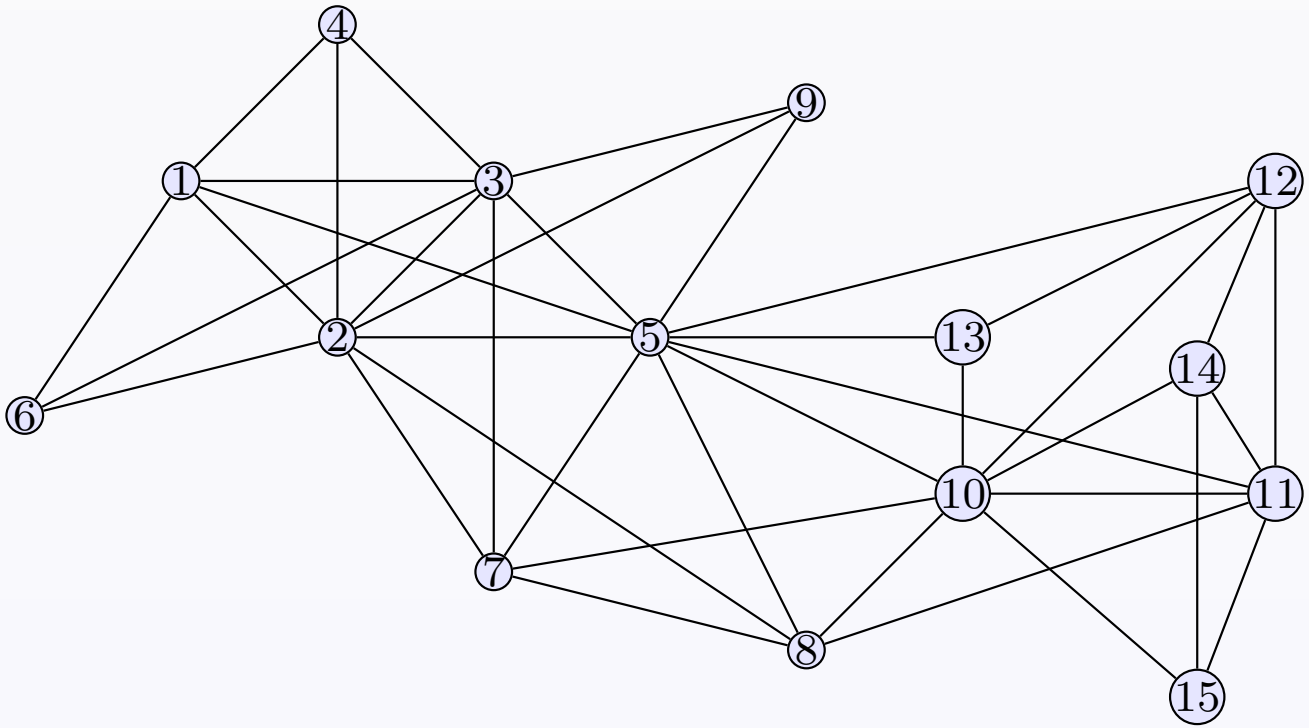
A 3-tree



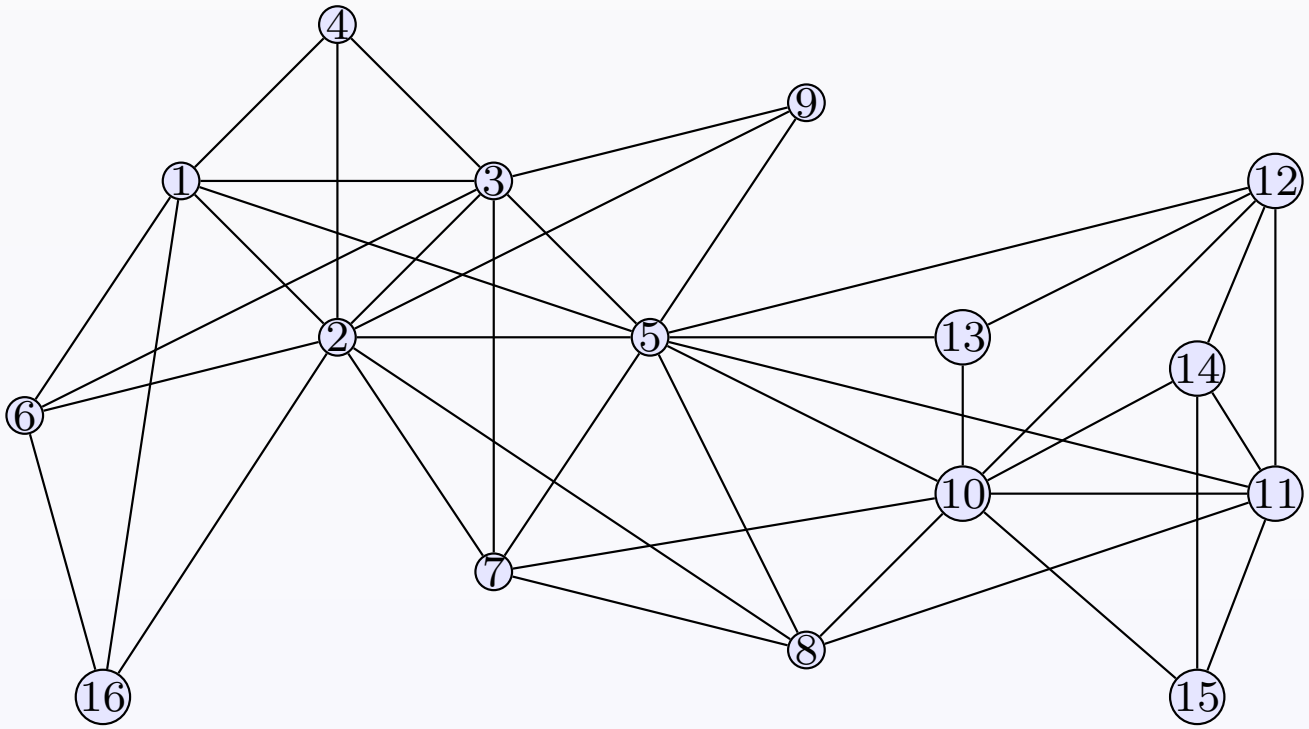
A 3-tree



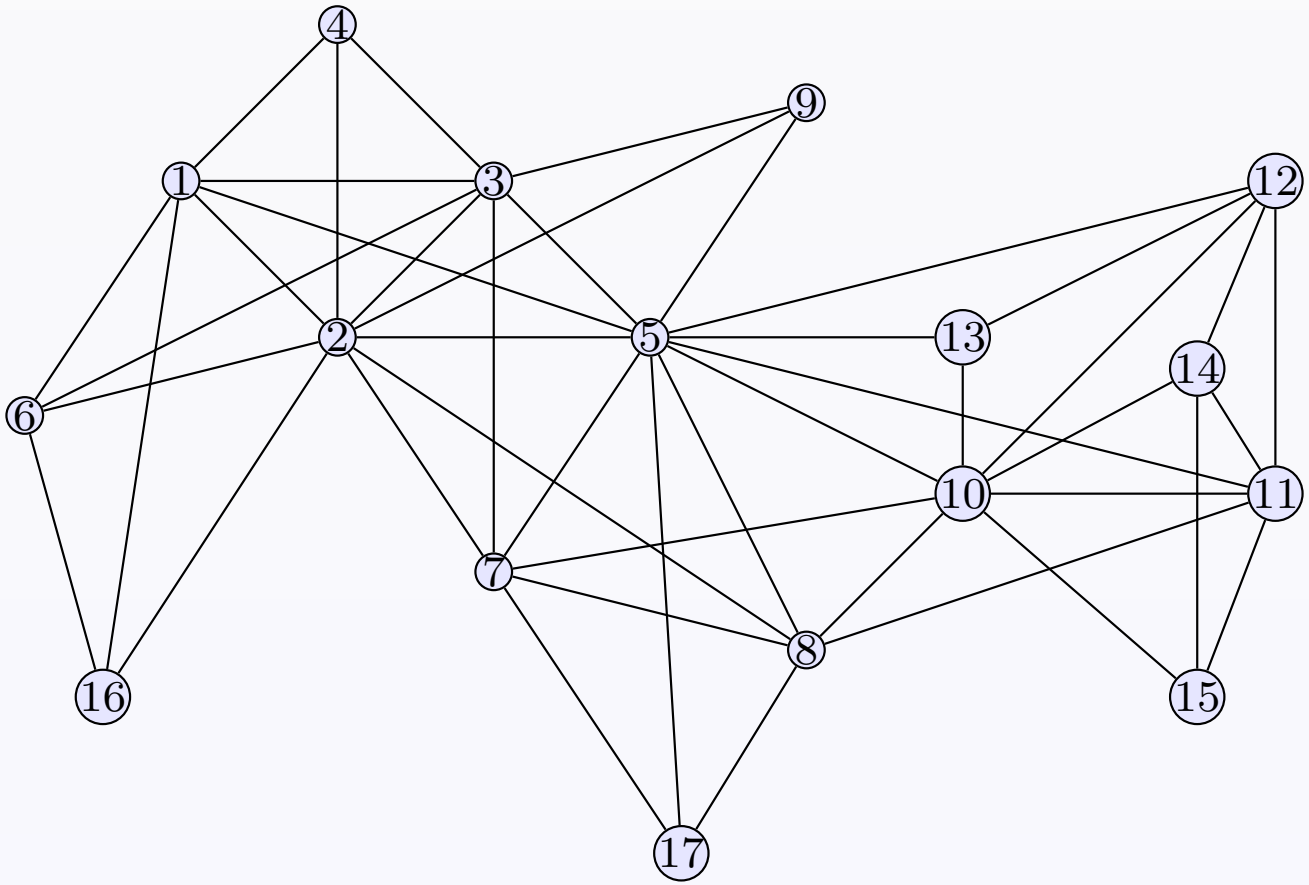
A 3-tree



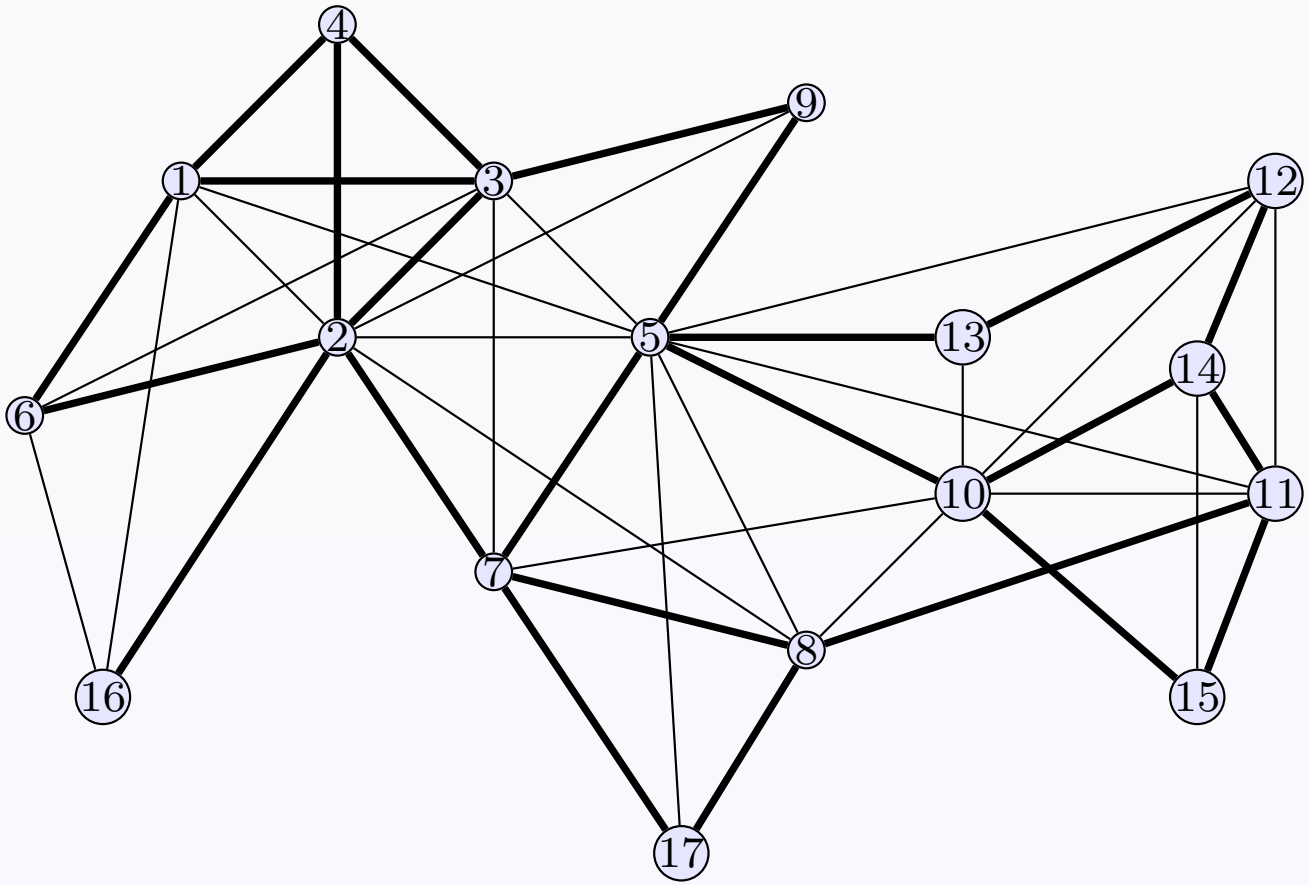
A 3-tree



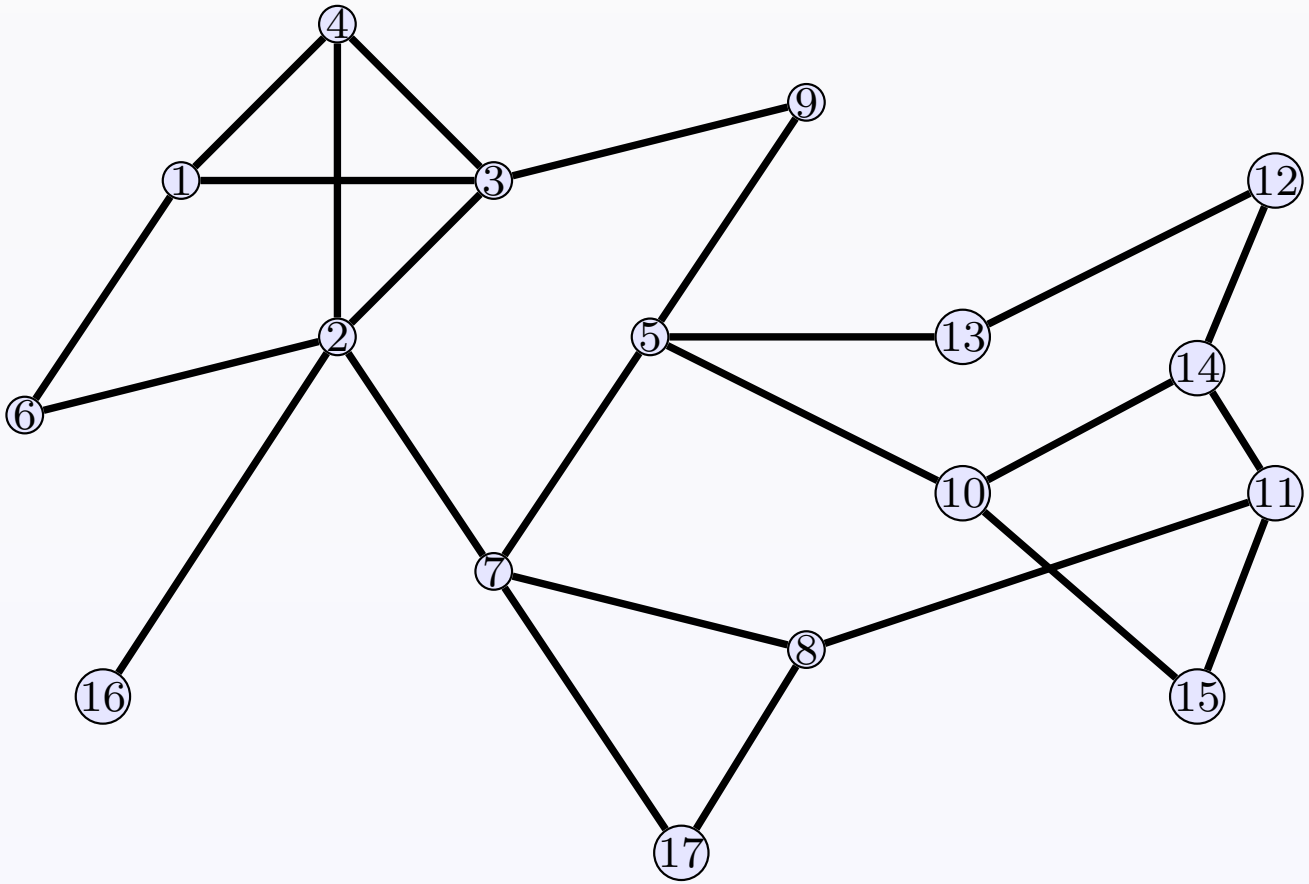
A 3-tree



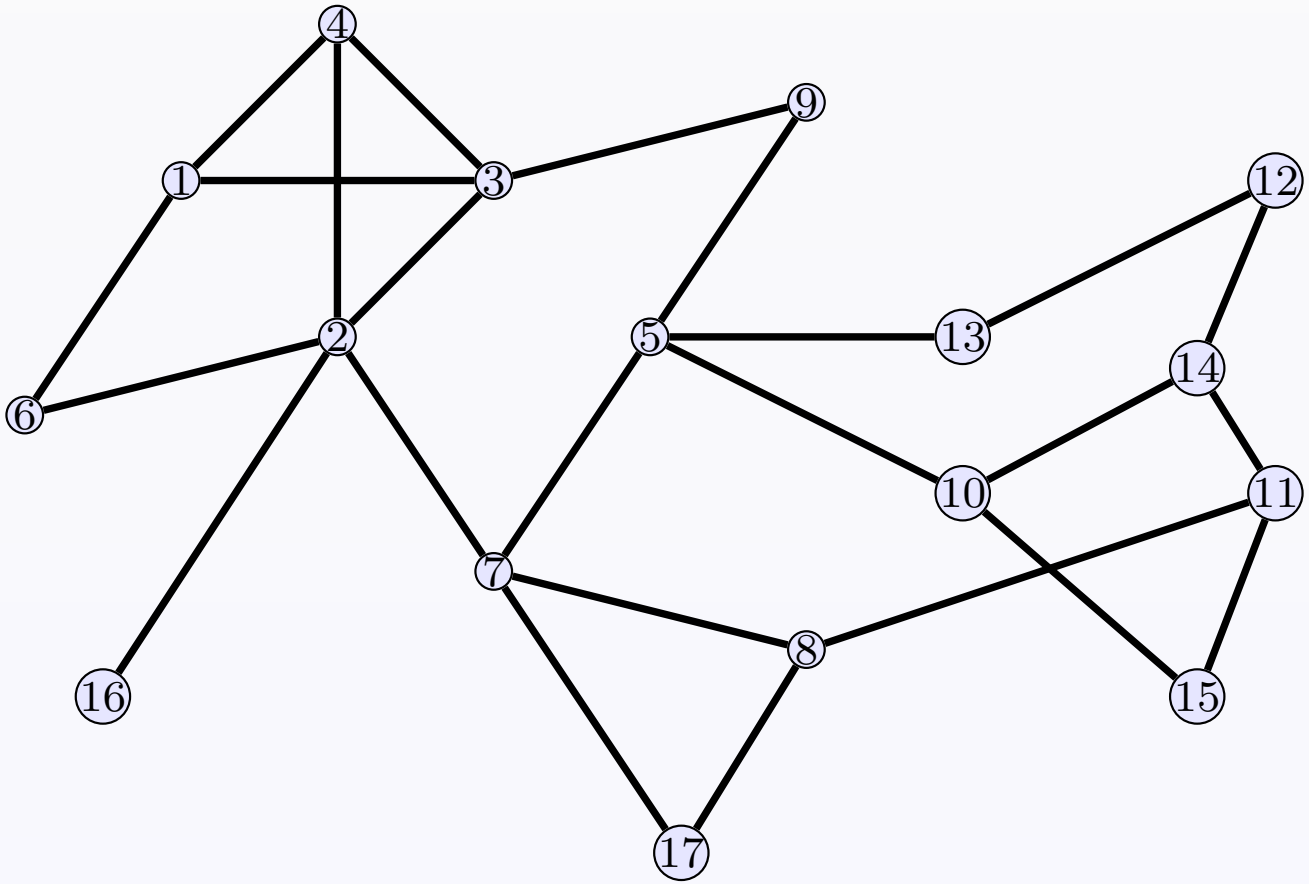
A 3-tree



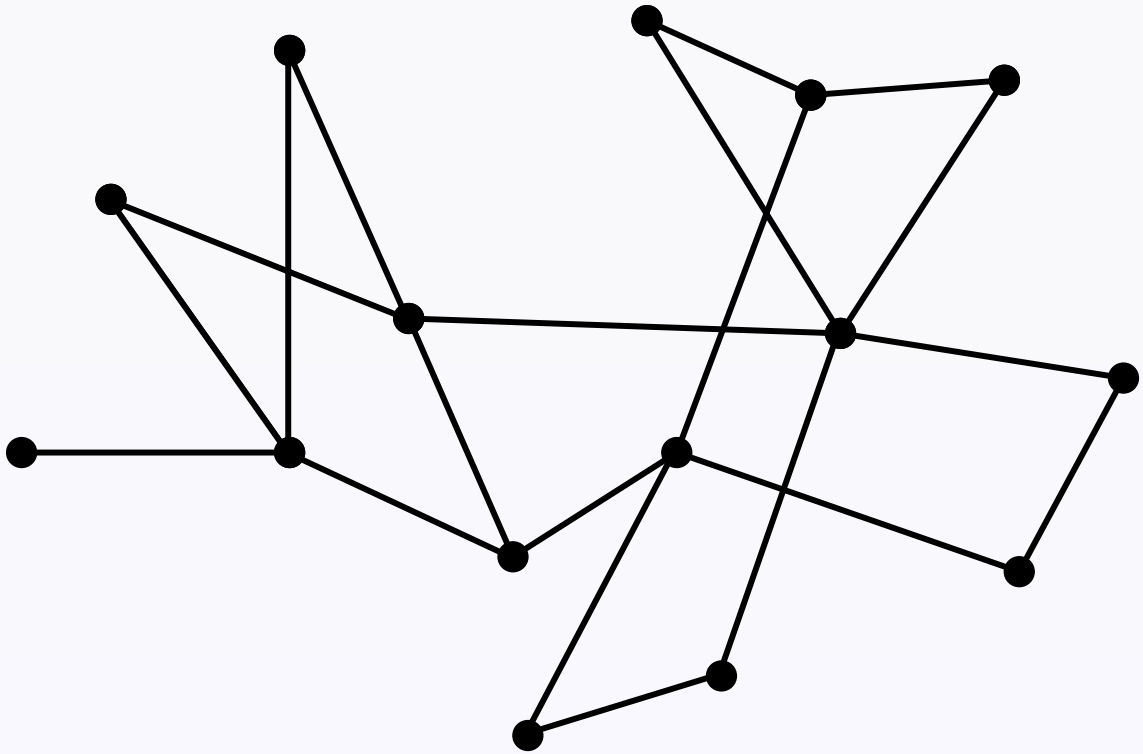
A 3-tree

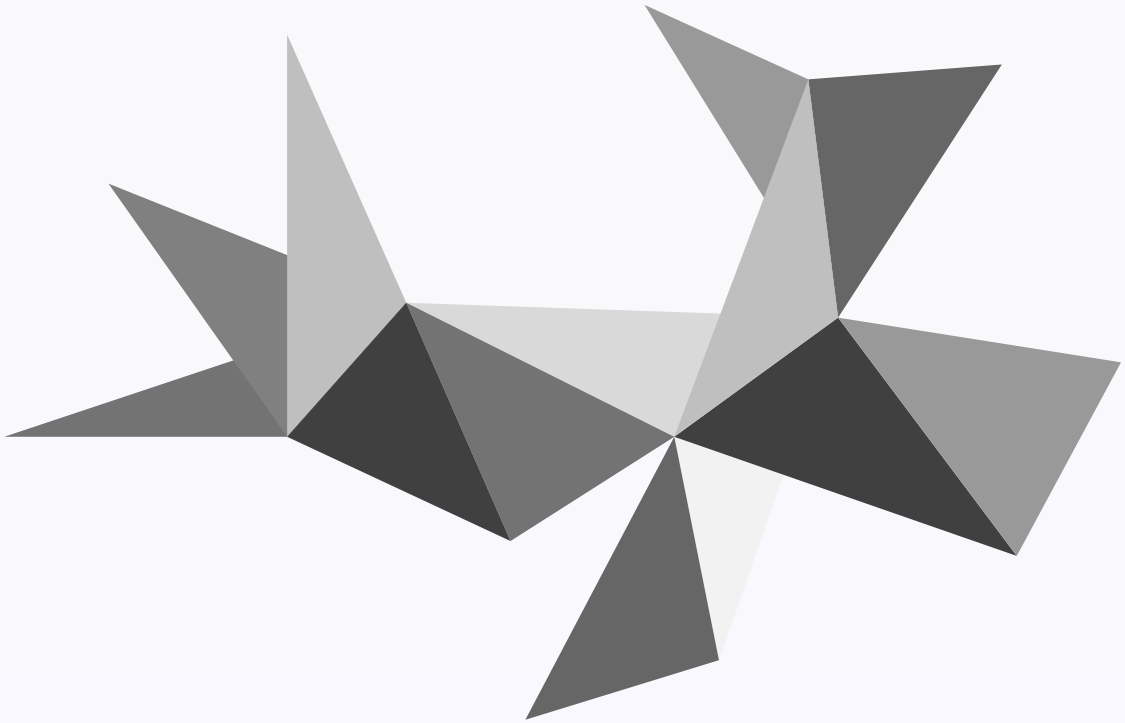


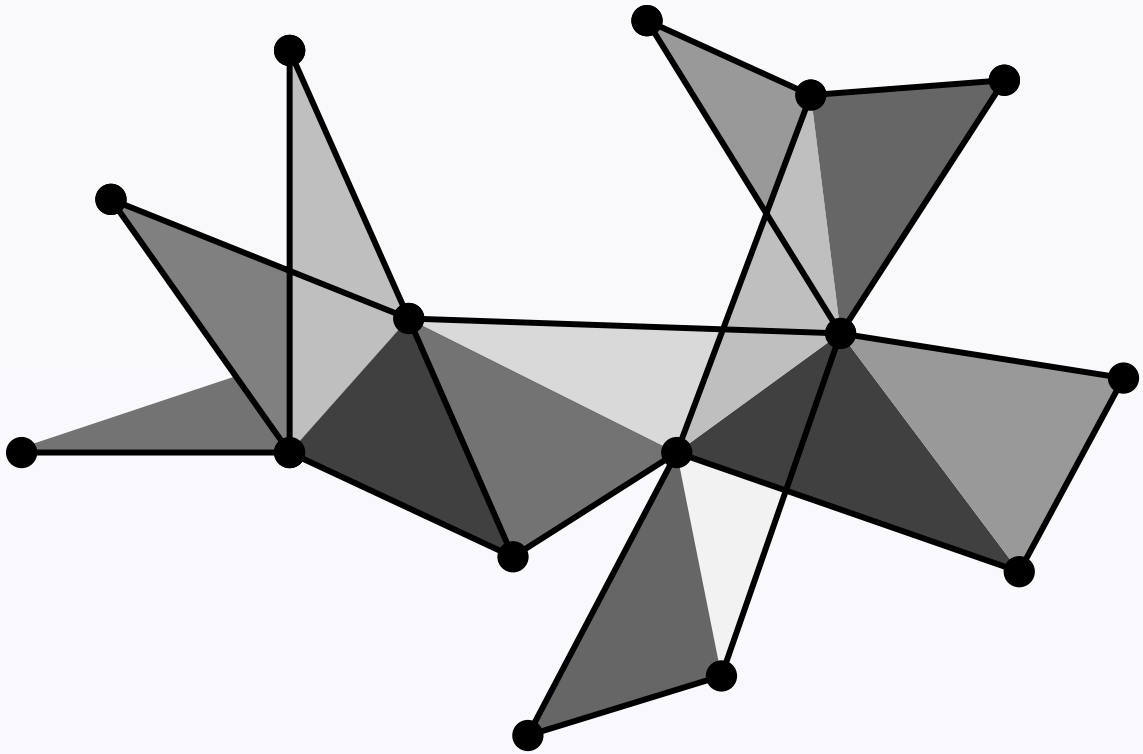
A subgraph of a 3-tree

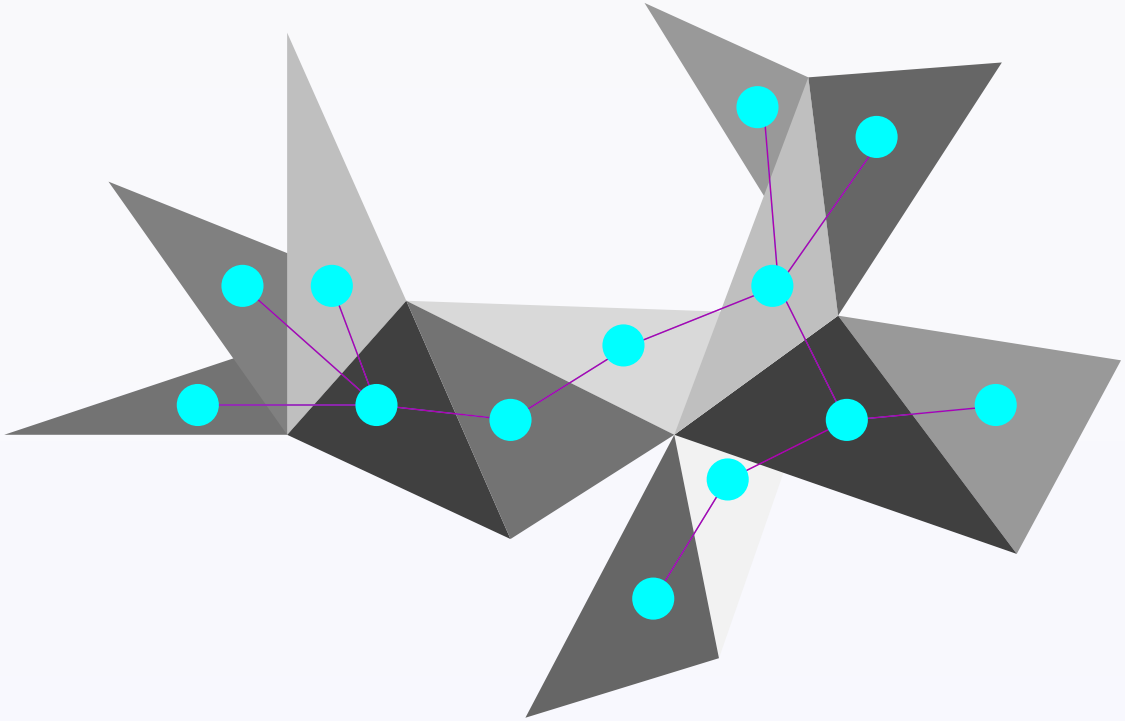


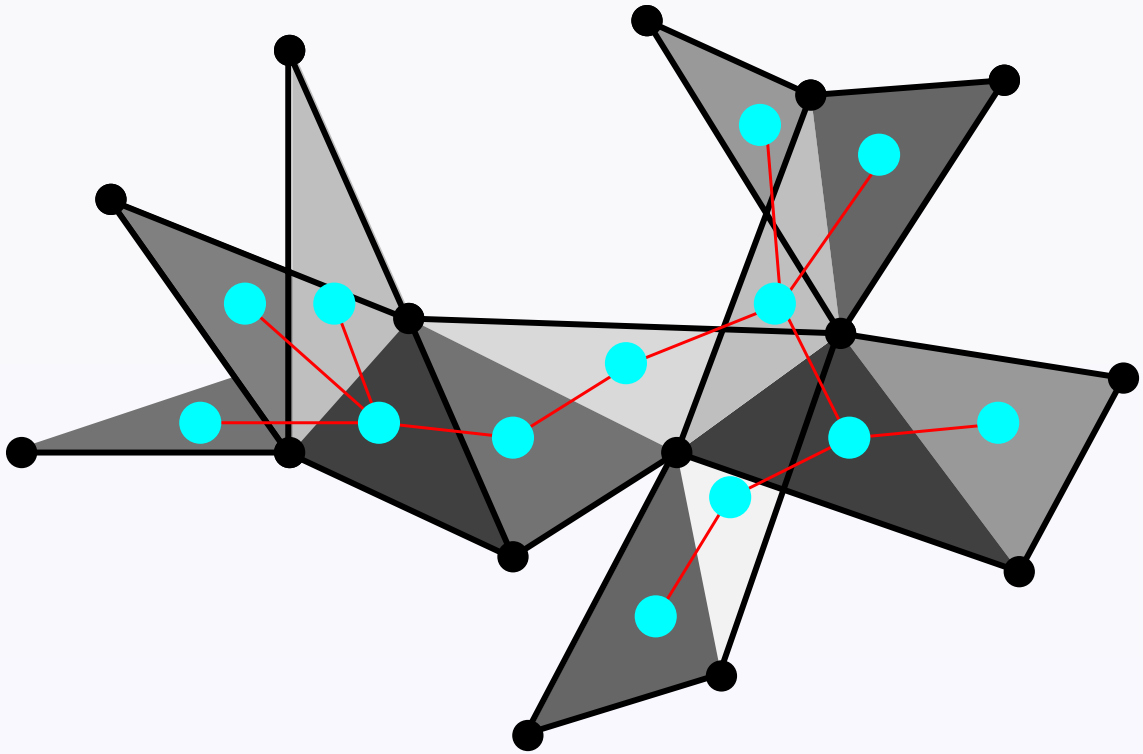
A subgraph of a 3-tree: a graph with treewidth at most 3











Facts about treewidth

- Defined for the first time by Bertelé & Brioschi on 1972 under the name *dimension*

Facts about treewidth

- Defined for the first time by Bertelé & Brioschi on 1972 under the name *dimension*
- Named *treewidth* by Roberson and Seymour in GM-II on 1986.

Facts about treewidth

- Defined for the first time by Bertelé & Brioschi on 1972 under the name *dimension*
- Named *treewidth* by Roberson and Seymour in GM-II on 1986.
- There are more alternative definitions of treewidth (at least six!)

Facts about treewidth

- Defined for the first time by Bertelé & Brioschi on 1972 under the name *dimension*
- Named *treewidth* by Roberson and Seymour in GM-II on 1986.
- There are more alternative definitions of treewidth (at least six!)
- Treewidth can be seen as a measure of the topological similarity of a graph to a tree

Facts about treewidth

- Defined for the first time by Bertelé & Brioschi on 1972 under the name *dimension*
- Named *treewidth* by Roberson and Seymour in GM-II on 1986.
- There are more alternative definitions of treewidth (at least six!)
- Treewidth can be seen as a measure of the topological similarity of a graph to a tree
- Treewidth is important in algorithm design (not only there)

Facts about treewidth

- Defined for the first time by Bertelé & Brioschi on 1972 under the name *dimension*
- Named *treewidth* by Roberson and Seymour in GM-II on 1986.
- There are more alternative definitions of treewidth (at least six!)
- Treewidth can be seen as a measure of the topological similarity of a graph to a tree
- Treewidth is important in algorithm design (not only there)
- Many NP-hard problems on graphs become polynomially solvable when their instances are restricted to graphs with constant treewidth.

Parameterizing treewidth

p-Treewidth

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: $\mathbf{tw}(G) \leq k$?

Parameterizing treewidth

p-Treewidth

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: $\mathbf{tw}(G) \leq k$?

p-Treewidth is in FPT by an $2^{O(k^3)} \cdot O(n)$ algorithm of Bodlaender

[SIAM J. Comp., 1996]

Monadic Second Order Logic

- A property in graphs may be expressed in MSO Logic

Monadic Second Order Logic

- A property in graphs may be expressed in MSO Logic

Universe: the vertex set V of the graph $G = (V, E)$

An MSO formula can be build using:

Variables: vertices $x, y, z, \ldots$ and sets of vertices $X, Y, Z, \ldots$

Atomic Formulae: $x = y$, $x \in X$, $\{x, y\} \in E$ $\quad$ $(E(x, y))$

Formulae: $\neg x$, $x \vee y$, $x \wedge y$, $x \rightarrow y$, $x \leftrightarrow y$, $\exists x \phi$, $\forall x \phi$, $\exists X \phi$, $\forall X \phi$,

Examples of properties expressible in MSO

3-Colorability:

$$\exists R\ \exists G\ \exists B\ [\forall x\ [(x \in R \vee x \in G \vee x \in B) \wedge$$

$$\neg(x \in R \wedge x \in G) \wedge \neg(x \in B \wedge x \in G) \wedge \neg(x \in R \wedge x \in B)]]$$

$$\wedge \neg[\exists x\ \exists y\ (\{x, y\} \in E \wedge$$

$$((x \in R \wedge y \in R) \vee (x \in G \wedge y \in G) \vee (x \in B \wedge y \in B)))]$$

Examples of properties expressible in MSO

Having an clique of size $\geq k$:

$\exists x_1 \ \exists x_2 \cdots \exists x_k \bigwedge_{1 \leq i < j \leq k} \{x_i, x_j\} \in E$

Examples of properties expressible in MSO

Having an independent set of size k:

$\exists x_1 \ \exists x_2 \cdots \exists x_k \bigwedge_{1 \leq i < j \leq k} (\neg \{x_i, x_j\} \in E) \wedge \neg (x_i \neq x_j)$

Examples of properties expressible in MSO

Having a vertex cover of size k:

$$\exists x_1\ \exists x_2 \cdots \exists x_k\ (\forall x\ \forall y\ \{x, y\} \in E \rightarrow (\bigvee\nolimits_{1 \leq i \leq k} (x = x_i \vee y = x_i)))$$

Examples of properties expressible in MSO

Having a dominating set of size k:

$\exists x_1 \ \exists x_2 \cdots \exists x_k \ \forall y \ \bigvee_{1 \leq i \leq k} (\{x_i, y\} \in E \vee y = x_i)$

Courcelle's theorem

MSO: Monadic Second Order Logic

Courcelle's theorem

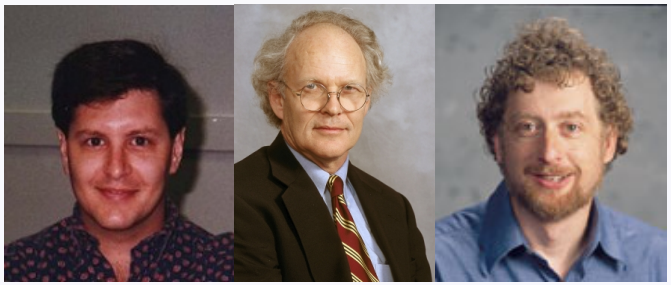
MSO: Monadic Second Order Logic

Theorem: [Courcelle], [Seese], & [Borie, Parker & Tovey] *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*

Courcelle's theorem

MSO: Monadic Second Order Logic

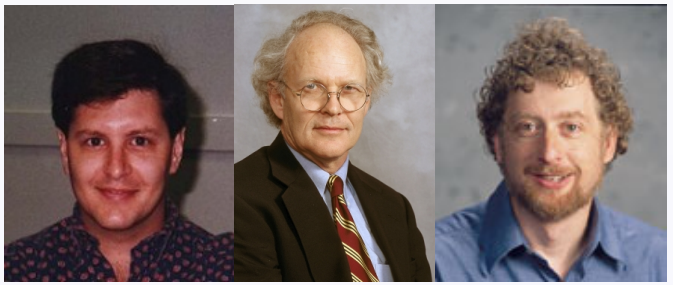
Theorem: [Courcelle], [Seese], & [Borie, Parker & Tovey] *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*



Courcelle's theorem

MSO: Monadic Second Order Logic

Theorem: [Courcelle], [Seese], & [Borie, Parker & Tovey] *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*



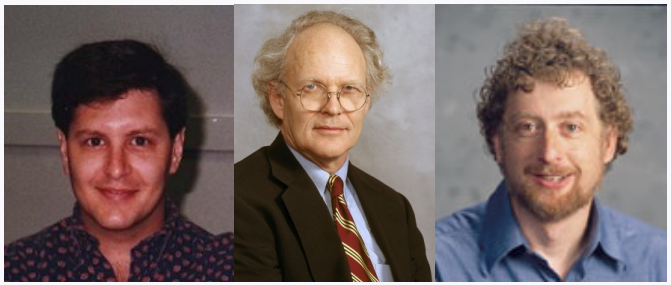
In other words:

If $\Pi \subseteq \mathcal{G}_{\text{all}}$ is a MSO-expressible set, then $(\Pi, \mathbf{tw}) \in$ FPT

Courcelle's theorem

MSO: Monadic Second Order Logic

Theorem: [Courcelle], [Seese], & [Borie, Parker & Tovey] *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*



In other words:

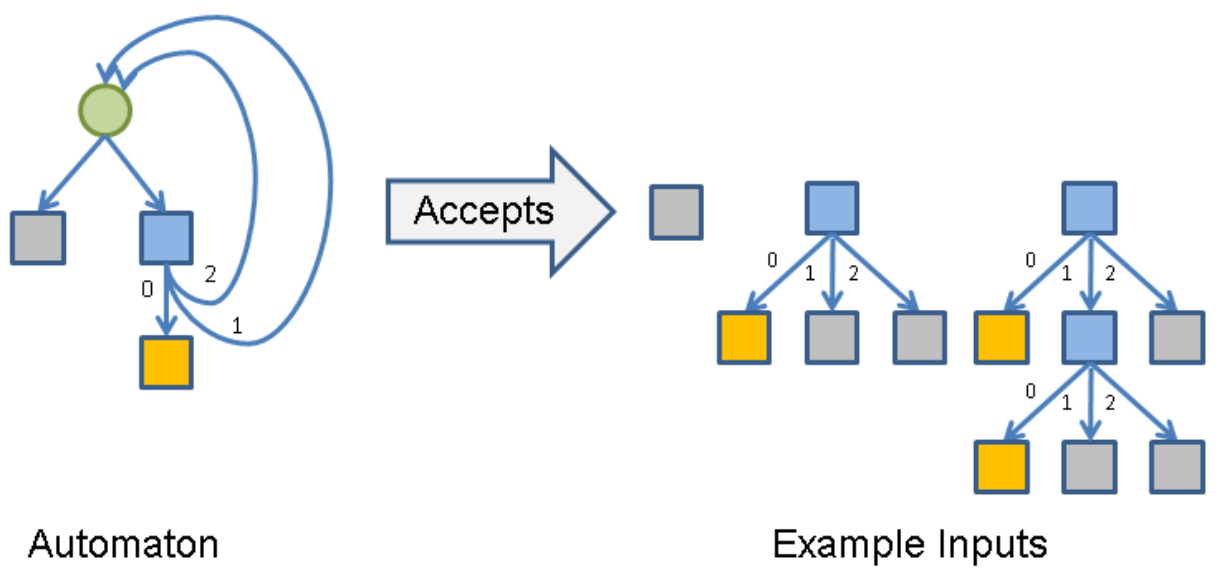
If $\Pi \subseteq \mathcal{G}_{\text{all}}$ is a MSO-expressible set, then $(\Pi, \mathbf{tw}) \in$ FPT

or

Every MSO-expressible problem of graphs is fixed parameter tractable when parameterized by the treewidth of its input graph

Inputs of small treewidth can be seen as tree-string: inputs of a tree-automaton generated by the MSO formula expressing $\mathcal{G}$.

Inputs of small treewidth can be seen as tree-string: inputs of a tree-automaton generated by the MSO formula expressing $\mathcal{G}$.



Theorem: *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*

Theorem: *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*

- Courcelle proved a stronger version where quantification on sets of edges is also allowed.

Advantage of Courcelle's Theorem: It constructs the algorithm

Theorem: *Every problem on graphs that can be expressed by a MSO formula ϕ can be solved in $f(\mathbf{tw}(G), |\phi|) \cdot n$ steps.*

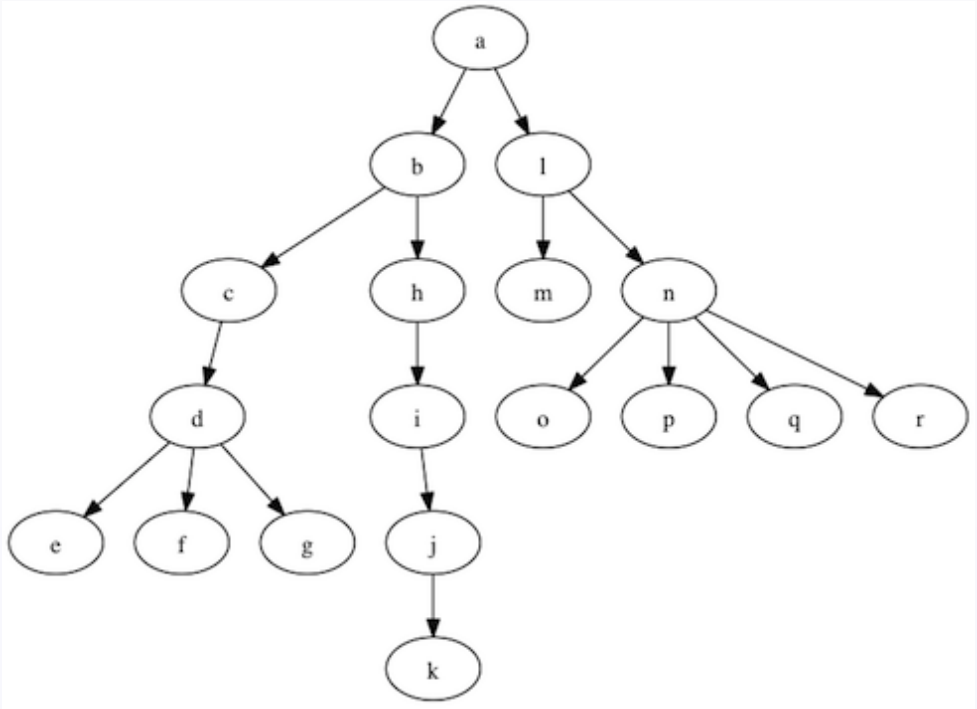
- Courcelle proved a stronger version where quantification on sets of edges is also allowed.

Advantage of Courcelle's Theorem: It constructs the algorithm

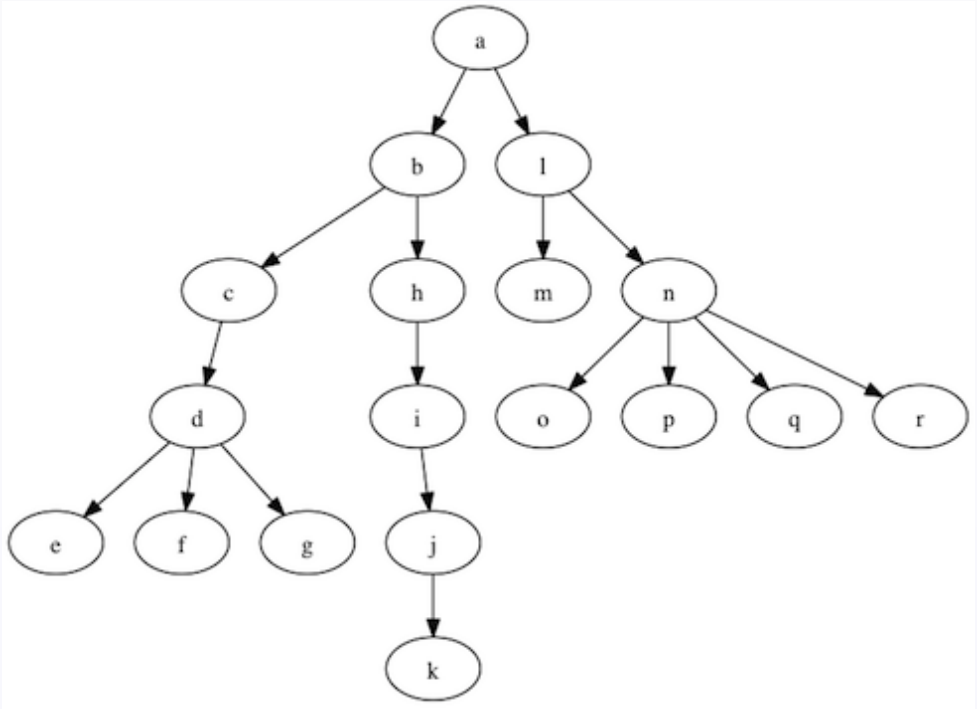
Drawback of Courcelle's Theorem: the contribution of the formula and the treewidth in the running time is immense.

In topological terms: treewidth helps us treat the input graph a mono-dimensional entity!

In topological terms: treewidth helps us treat the input graph a mono-dimensional entity!

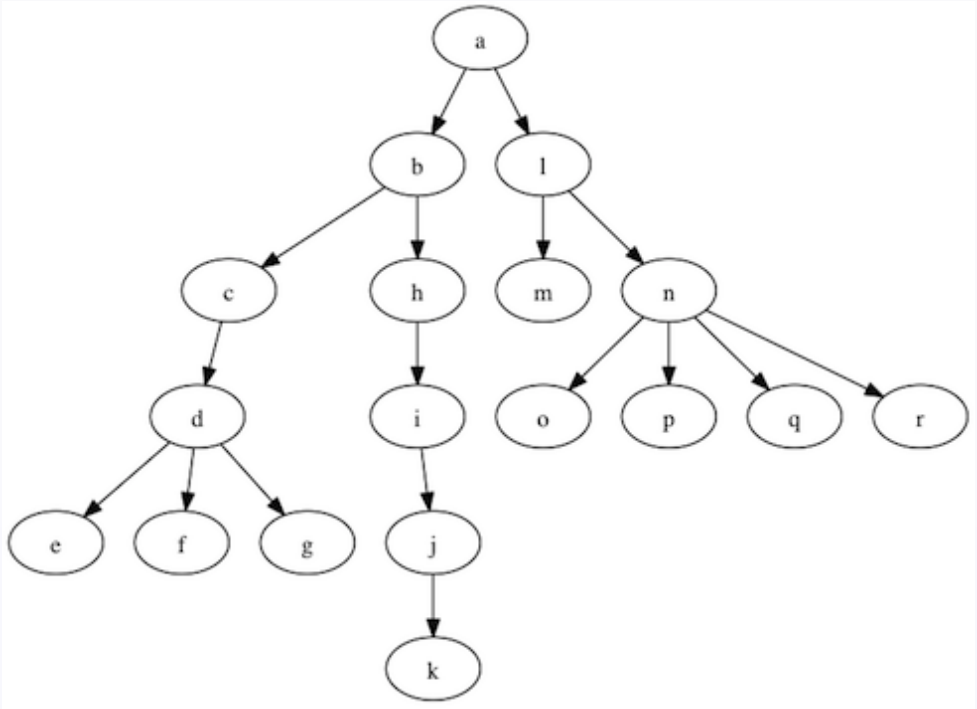


In topological terms: treewidth helps us treat the input graph a mono-dimensional entity!



Treewidth is a measure of the possibility of recursively cutting the graph in smaller pieces and process them separately:

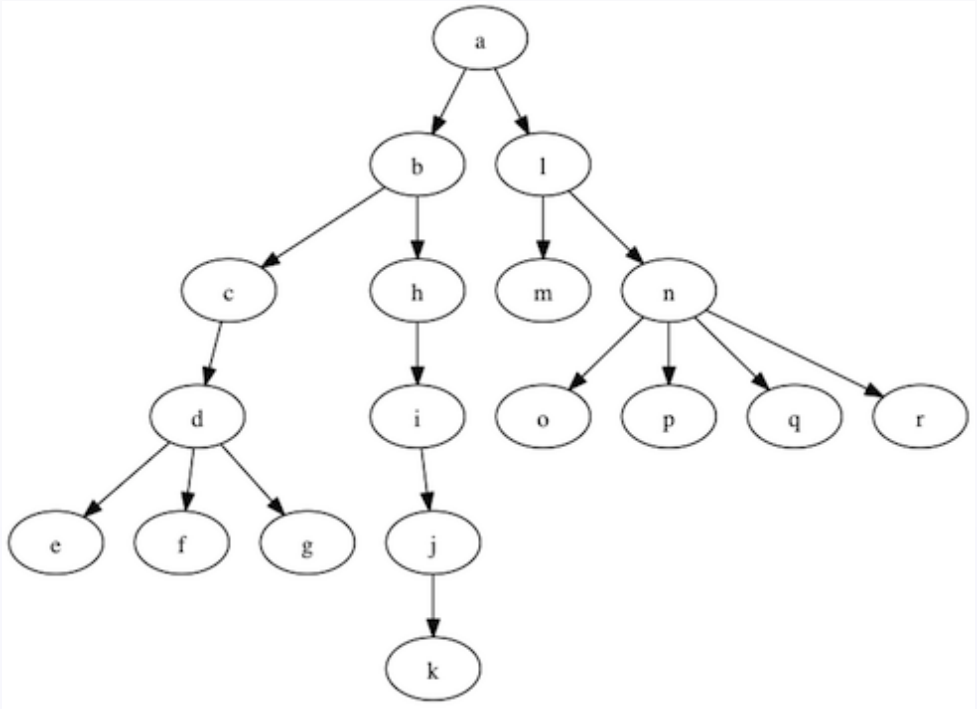
In topological terms: treewidth helps us treat the input graph a mono-dimensional entity!



Treewidth is a measure of the possibility of recursively cutting the graph in smaller pieces and process them separately:

In an algorithmic terms: Divide and Conquer!

In topological terms: treewidth helps us treat the input graph a mono-dimensional entity!



Treewidth is a measure of the possibility of recursively cutting the graph in smaller pieces and process them separately:

In an algorithmic terms: Divide and Conquer!

Which in our case is: Dynamic Programming

Nice tree decompositions

A tree decomposition $D = (T, \mathcal{X})$ is *nice* if T is rooted to some leaf r and

Nice tree decompositions

A tree decomposition $D = (T, \mathcal{X})$ is *nice* if T is rooted to some leaf r and

- for any leaf l of T where $l \neq r$, $X_l = \emptyset$

  (we call X_l *leaf node* of D except from X_r that we call root node)

Nice tree decompositions

A tree decomposition $D = (T, \mathcal{X})$ is *nice* if T is rooted to some leaf r and

- for any leaf l of T where $l \neq r$, $X_l = \emptyset$

  (we call X_l *leaf node* of D except from X_r that we call root node)

- any non-leaf $t \in V(T)$ (including the root) has one or two children.

Nice tree decompositions

A tree decomposition $D = (T, \mathcal{X})$ is *nice* if T is rooted to some leaf r and

- for any leaf l of T where $l \neq r$, $X_l = \emptyset$

  (we call X_l *leaf node* of D except from X_r that we call root node)
- any non-leaf $t \in V(T)$ (including the root) has one or two children.
- if t has two children t_1 and t_2 then, $X_t = X_{t_1} = X_{t_2}$

  (we call X_t *join* node)

Nice tree decompositions

A tree decomposition $D = (T, \mathcal{X})$ is *nice* if T is rooted to some leaf r and

- for any leaf l of T where $l \neq r$, $X_l = \emptyset$

  (we call X_l *leaf node* of D except from X_r that we call root node)
- any non-leaf $t \in V(T)$ (including the root) has one or two children.
- if t has two children t_1 and t_2 then, $X_t = X_{t_1} = X_{t_2}$

  (we call X_t *join* node)
- if t has one child t' then
  - either $X_t = X_{t'} \cup \{v\}$

    (we call X_t *insert* node and v is the *insert vertex*)
  - or $X_{t'} = X_t \cup \{v\}$

    (we call X_t *forget* node and v is the *forget vertex*)

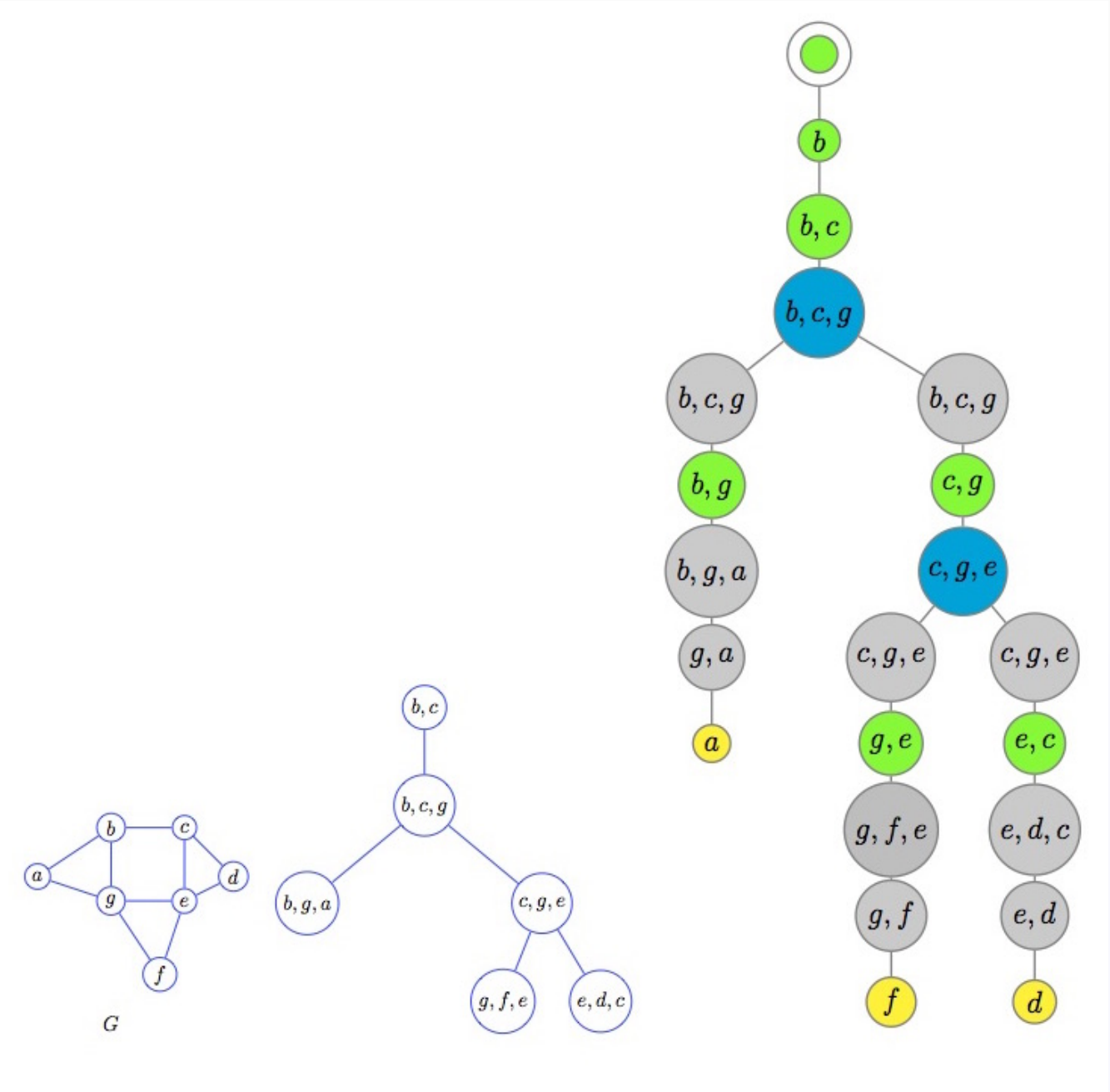
If $(T, \mathcal{X})$ is a nice tree decomposition rooted on r, then

for any $t \in V(T)$, $G_t = G[\bigcup_{t' \text{ is } t \text{ or a descendant of } t \text{ in } T} X_{t'}]$

If $(T, \mathcal{X})$ is a nice tree decomposition rooted on r, then

for any $t \in V(T)$, $G_t = G[\bigcup_{t' \text{ is } t \text{ or a descendant of } t \text{ in } T} X_{t'}]$

Lemma: *There exists an $O(n)$-step algorithm that transforms any tree decomposition with n nodes to a nice tree decomposition of $\leq 4n$ nodes of the same width.*



A graph G, a tree decomposition, and a nice tree decomposition

How to do dynamic programming for graphs of small treewidth

1. Define, for each $t \in V(T)$, a table that encodes the information of a partial solution for G_t. The values of this table for the root node should provide a global answer.

How to do dynamic programming for graphs of small treewidth

1. Define, for each $t \in V(T)$, a table that encodes the information of a partial solution for G_t. The values of this table for the root node should provide a global answer.
2. Define the values of this table for the leaf nodes.

How to do dynamic programming for graphs of small treewidth

1. Define, for each $t \in V(T)$, a table that encodes the information of a partial solution for G_t. The values of this table for the root node should provide a global answer.
2. Define the values of this table for the leaf nodes.
3. Provide the way to compute the table of an insert node, given the table of its child.

How to do dynamic programming for graphs of small treewidth

1. Define, for each $t \in V(T)$, a table that encodes the information of a partial solution for G_t. The values of this table for the root node should provide a global answer.
2. Define the values of this table for the leaf nodes.
3. Provide the way to compute the table of an insert node, given the table of its child.
4. Provide the way to compute the table of a forget node, given the table its child.

How to do dynamic programming for graphs of small treewidth

1. Define, for each $t \in V(T)$, a table that encodes the information of a partial solution for G_t. The values of this table for the root node should provide a global answer.
2. Define the values of this table for the leaf nodes.
3. Provide the way to compute the table of an insert node, given the table of its child.
4. Provide the way to compute the table of a forget node, given the table its child.
5. Provide a way to compute the table of a join node, given the tables of its children.

Parameterizing 3-Coloring by treewidth

tw-3-Vertex Coloring

Instance: A graph G.

Parameter: $k = \mathbf{tw}(G)$

Question: $\exists \chi : V(G) \to \{1, 2, 3\} : \forall \{v, u\} \in E(G)\ \chi(v) \neq \chi(u)$?

For any $\chi : S \to I$ and $R \subseteq S$, we define $\chi[R] = \{(v, \chi(v)) \in \chi \mid v \in R\}$

For any $\chi : S \to I$ and $R \subseteq S$, we define $\chi[R] = \{(v, \chi(v)) \in \chi \mid v \in R\}$

1st step: Definition of the tables:

For any $t \in V(T)$ and any 3-coloring $\phi : X_t \to \{1, 2, 3\}$, we define

$$B_t(\phi) = [\exists \chi : V(G_t) \to \{1, 2, 3\} \text{ such that } \chi[X_t] = \phi]$$

(the table of t contains an array of $3^{|X_t|}$ bits)

For any $\chi : S \to I$ and $R \subseteq S$, we define $\chi[R] = \{(v, \chi(v)) \in \chi \mid v \in R\}$

1st step: Definition of the tables:

For any $t \in V(T)$ and any 3-coloring $\phi : X_t \to \{1, 2, 3\}$, we define

$$B_t(\phi) = [\exists \chi : V(G_t) \to \{1, 2, 3\} \text{ such that } \chi[X_t] = \phi]$$

(the table of t contains an array of $3^{|X_t|}$ bits)

$G = G_r$ is 3-colourable iff $B_r(\varnothing) = 1$

2nd step: tables for leaf nodes:

Let X_l be an leaf node

we have

$$B_l(\varnothing) = 1$$

3rd step: tables for insert nodes:

Let X_t be an insert node

let t' be the child of t and v be the insert vertex.

For any $\phi : X_t \to \{1, 2, 3\}$, we have

$$B_t(\phi) = B_{t'}(\phi - (v, \phi(v))) \bigwedge_{u \in N_{G_t}(v)} [\phi(v) \neq \phi(u)]$$

4nd step: tables for forget nodes:

Let X_t be a forget node

let t' be the child of t and v be the forget vertex.

For any $\phi : X_t \to \{1, 2, 3\}$, we have

$$B_t(\phi) = \bigvee_{i \in \{1,2,3\}} B_{t'}(\phi \cup \{v, i\})$$

5th step: tables for join nodes:

Let X_t be an join node

let t_1, t_2 be the children of t

For any $\phi : X_t \to \{1, 2, 3\}$, we have

$$B_t(\phi) = B_{t_1}(\phi) \wedge B_{t_2}(\phi)$$

Conclusion:

Given a tree decomposition of G,

the following tw-3-Vertex-Coloring problem is in $2^{O(k)}$-FTP:

(we gave an $O(3^k \cdot k \cdot n)$ dynamic programming algorithm)

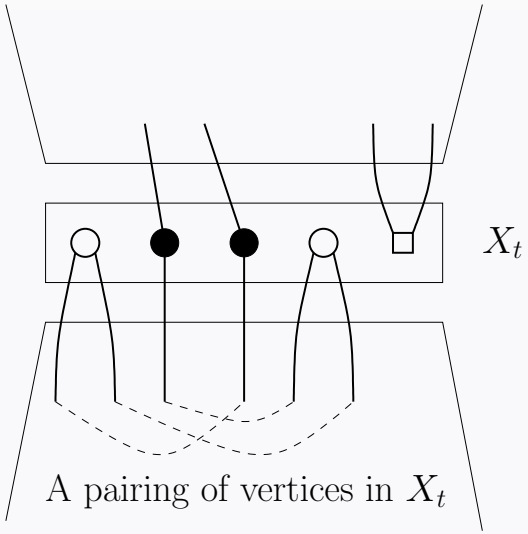
Parameterizing Hamiltonian Cycle by treewidth:

tw-Hamiltonian Cycle

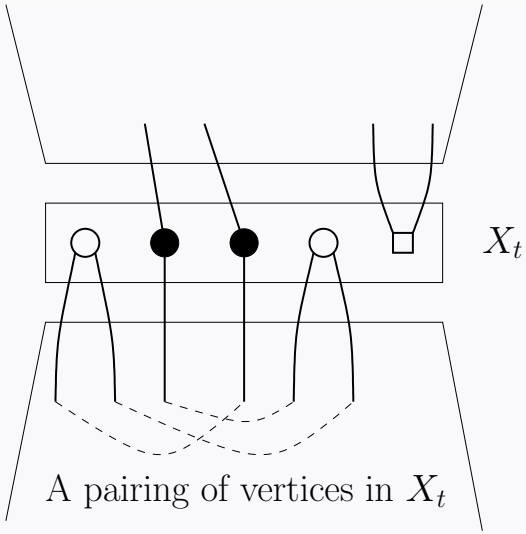
Instance: A graph G.

Parameter: $k = \mathbf{tw}(G)$

Question: does G contain a spanning cycle?

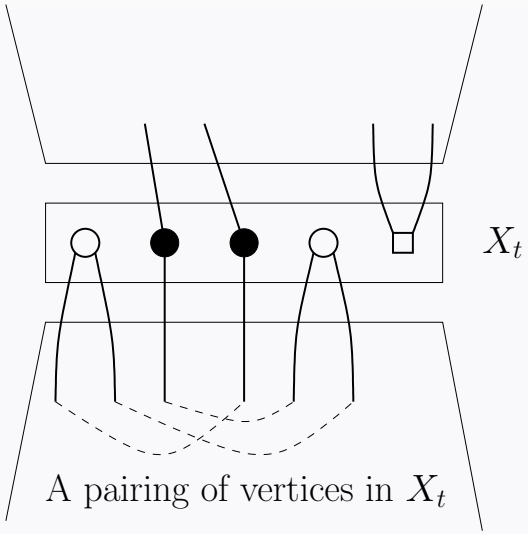


A pairing of vertices in X_t



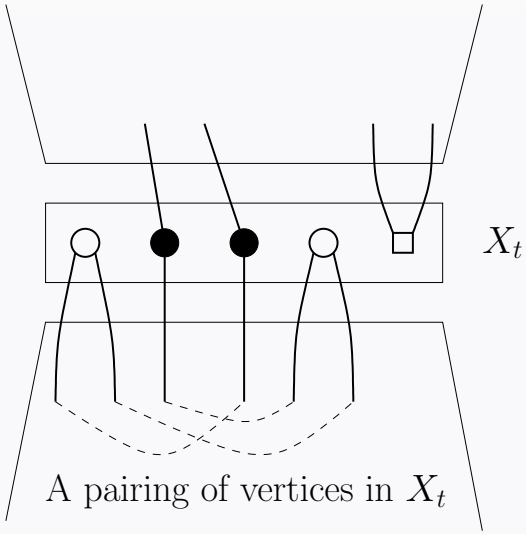
A pairing of vertices in X_t

A *pairing* of X_t is a graph H (with loops) s.t. $V(G) = X_i$ and $\forall x \in X_i$ $\mathbf{deg}_H(x) \leq 2$



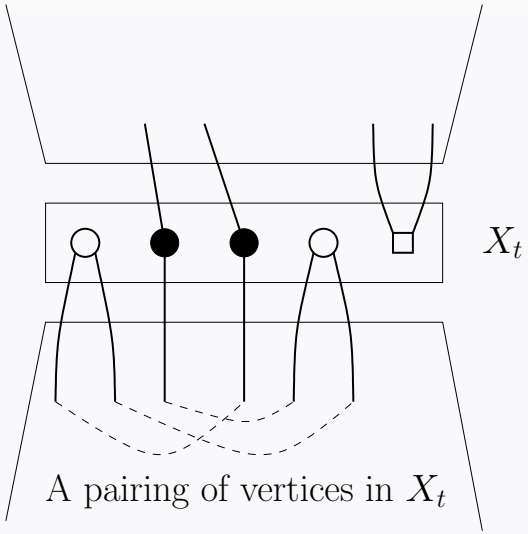
A *pairing* of X_t is a graph H (with loops) s.t. $V(G) = X_i$ and $\forall x \in X_i$ $\mathbf{deg}_H(x) \leq 2$

- The restriction of a cycle to G_t is a collection $\mathcal{P}$ of internally disjoint paths in G_t with ends in X_i.



A *pairing* of X_t is a graph H (with loops) s.t. $V(G) = X_i$ and $\forall x \in X_i$ $\mathbf{deg}_H(x) \leq 2$

- The restriction of a cycle to G_t is a collection $\mathcal{P}$ of internally disjoint paths in G_t with ends in X_i.
- Each $\mathcal{P}$ corresponds to some pairing $H_{\mathcal{P}}$ of X_t



A *pairing* of X_t is a graph H (with loops) s.t. $V(G) = X_i$ and $\forall x \in X_i$ $\mathbf{deg}_H(x) \leq 2$

- The restriction of a cycle to G_t is a collection $\mathcal{P}$ of internally disjoint paths in G_t with ends in X_i.
- Each $\mathcal{P}$ corresponds to some pairing $H_{\mathcal{P}}$ of X_t
- For any set S, let $\texttt{pairs}(S)$ be the set of all pairings of S

Let $(T, \mathcal{X})$ be a tree decomposition of G where $X_r = \{w\}$

let H_w be just the vertex w looped.

Let $(T, \mathcal{X})$ be a tree decomposition of G where $X_r = \{w\}$

let H_w be just the vertex w looped.

1st Step: For each $t \in V(T)$ we define:

$\forall H \in \texttt{pairs}(X_i)$,

$$B_t(H) = [H \text{ is the pairing of some } t\text{-path collection } \mathcal{P}]$$

$G = G_r$ has a Hamiltonian cycle iff $B_r(H_w) = 1$

2nd step: tables for leaf nodes:

Let X_l be an leaf node (assume that $X_l = \{y\}$)

Notice that $\texttt{pairs}(t) = \{H_0, H_1\}$

where $H_0(H_1)$ is the vertex y looped (unlooped)

$$\forall H \in \texttt{pairs}(t)\ B_l(H) = [|E(H)| = 0]$$

3rd step: tables for insert nodes:

Let X_t be an insert node

let t' be the child of t and v be the insert vertex.

For any $\forall H \in \texttt{pairs}(t)$ we have

$$B_t(H) = [B_{t'}(H - v)] \wedge \ [N_H(v) \subseteq N_{G_t}(v)]$$

4st step: tables for forget nodes:

Let X_t be an forget node

let t' be the child of t and v be the insert vertex.

For any $\forall H \in \texttt{pairs}(t)$ we have

$$B_t(H) = \bigvee_{\substack{H' \in \texttt{pairs}(t') \\ H \text{ is a contraction of } H'}} B_{t'}(H')$$

5th step: tables for join nodes:

Let X_t be an join node

let t_1, t_2 be the children of t

For any $\forall H \in \texttt{pairs}(t)$ we have

$$B_t(H) = \bigvee_{\substack{H_1 \in \texttt{pairs}(t_1) \\ H_2 \in \texttt{pairs}(t_2) \\ H = H_1 \cup H_2}} B_{t_1}(H_1) \wedge B_{t_2}(H_2)$$

There are $2^{O(k \log k)}$ pairings for each bug X_t of $k+1$ vertices.

Conclusion:

There are $2^{O(k \log k)}$ pairings for each bug X_t of $k+1$ vertices.

Conclusion:

tw-Hamiltonian Cycle admits a $2^{O(k \log k)} \cdot n$-step algorithm

Therefore, it belongs in $2^{O(k \log k)}$-FPT

There are $2^{O(k \log k)}$ pairings for each bug X_t of $k+1$ vertices.

Conclusion:

tw-Hamiltonian Cycle admits a $2^{O(k \log k)} \cdot n$-step algorithm

Therefore, it belongs in $2^{O(k \log k)}$-FPT

Our next step is to show that tw-Planar Hamiltonian Cycle $\in 2^{O(k)}$-FPT

Part 3, Tuesday 10/05/2016 - 16:00–17:00 (60')

Branch decompositions

Sphere cut decompositions

Dynamic programming on planar graphs

Branch decompositions

Branchwidth is a (topological) tree-likeness measure, alternative to treewidth, appeared in GM-X (1991).

Branch decompositions

Branchwidth is a (topological) tree-likeness measure, alternative to treewidth, appeared in GM-X (1991).

A *branch decomposition* is a pair (T, τ)

Branch decompositions

Branchwidth is a (topological) tree-likeness measure, alternative to treewidth, appeared in GM-X (1991).

A *branch decomposition* is a pair (T, τ)

where

1. T is a ternary tree and
2. τ is a bijection mapping the edges of G to the leaves of T.

Branch decompositions

Branchwidth is a (topological) tree-likeness measure, alternative to treewidth, appeared in GM-X (1991).

A *branch decomposition* is a pair (T, τ)

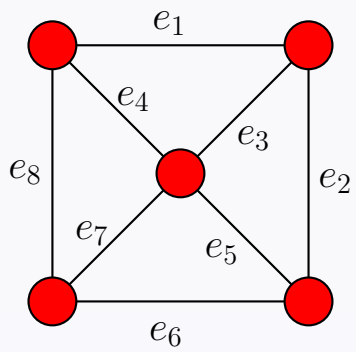
where

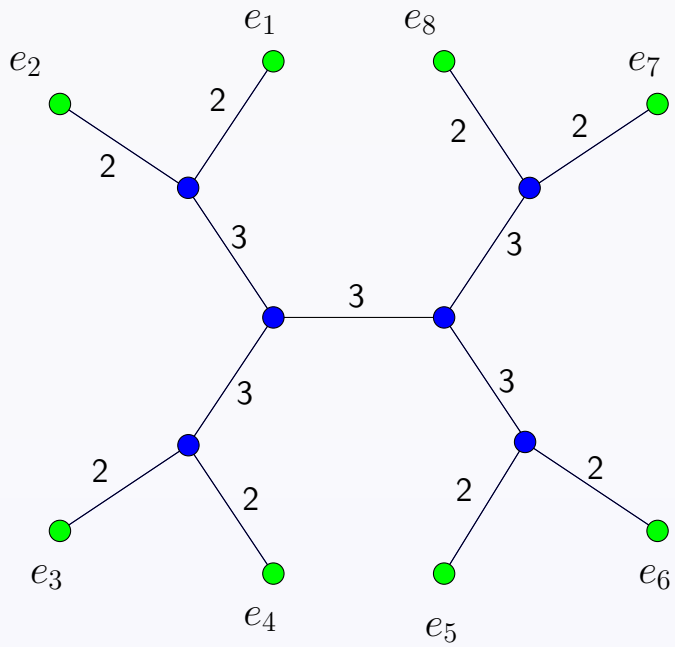
1. T is a ternary tree and
2. τ is a bijection mapping the edges of G to the leaves of T.

if T_1 is one of the connected components of $T - e$ then we set

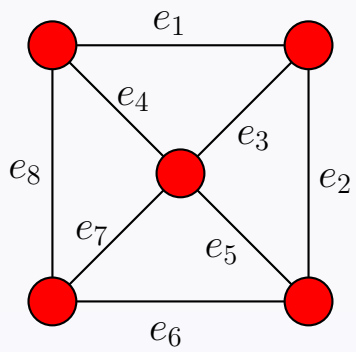
$E_e = \tau^{-1}(\text{leaves of } T_1)$ and $\mathbf{mid}(e) = \partial E_e$.

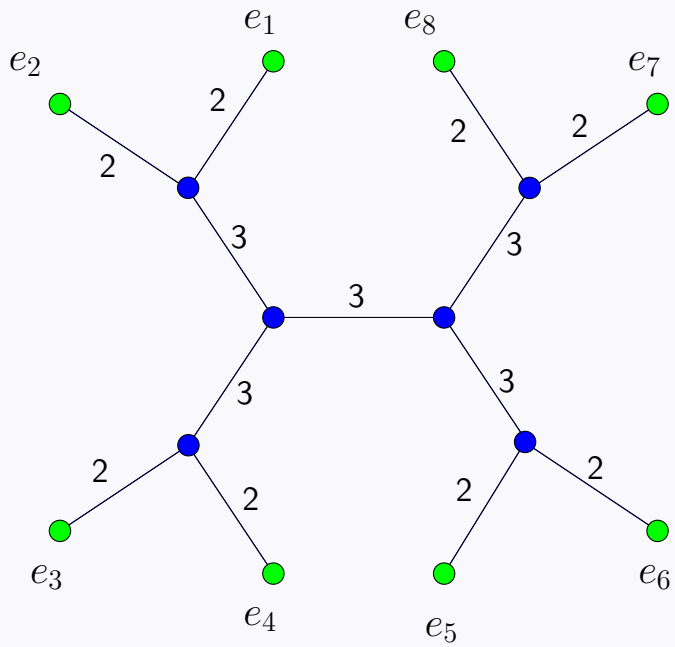
A graph G and a branch decomposition of it.





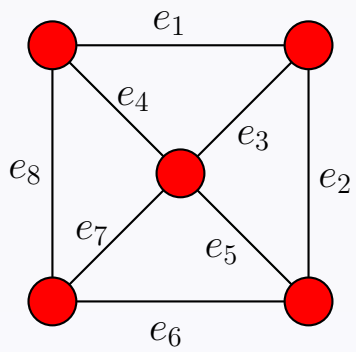
A graph G and a branch decomposition of it.

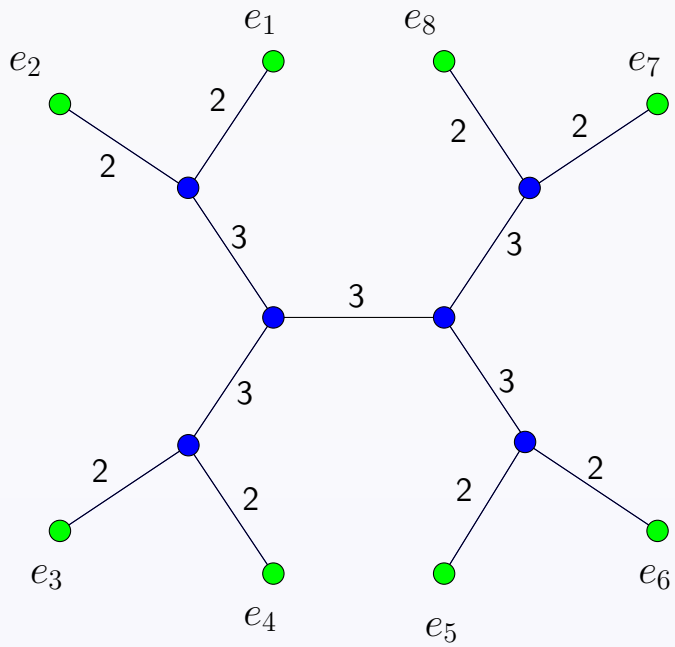




The *width* of a branch decomposition (T, τ) is $\max\{|\mathbf{mid}(e)| \mid e \in E(T)\}$

A graph G and a branch decomposition of it.





The *width* of a branch decomposition (T, τ) is $\max\{|\mathbf{mid}(e)| \mid e \in E(T)\}$

The *branchwidth*, $\mathbf{bw}(G)$, of a graph G is then minimum width a branch decomposition of G may have.

Combinatorics of branchwidth

Combinatorics of branchwidth

Theorem: [Robertson and Seymour, GM-10] *If G is not acyclic, then*

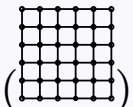
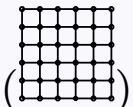
$\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2}\mathbf{bw}(\mathbf{G})$

Combinatorics of branchwidth

Theorem: [Robertson and Seymour, GM-10] *If G is not acyclic, then*

$\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2}\mathbf{bw}(\mathbf{G})$

If T is a tree, then $0 \leq \mathbf{bw}(G) \leq 2$.

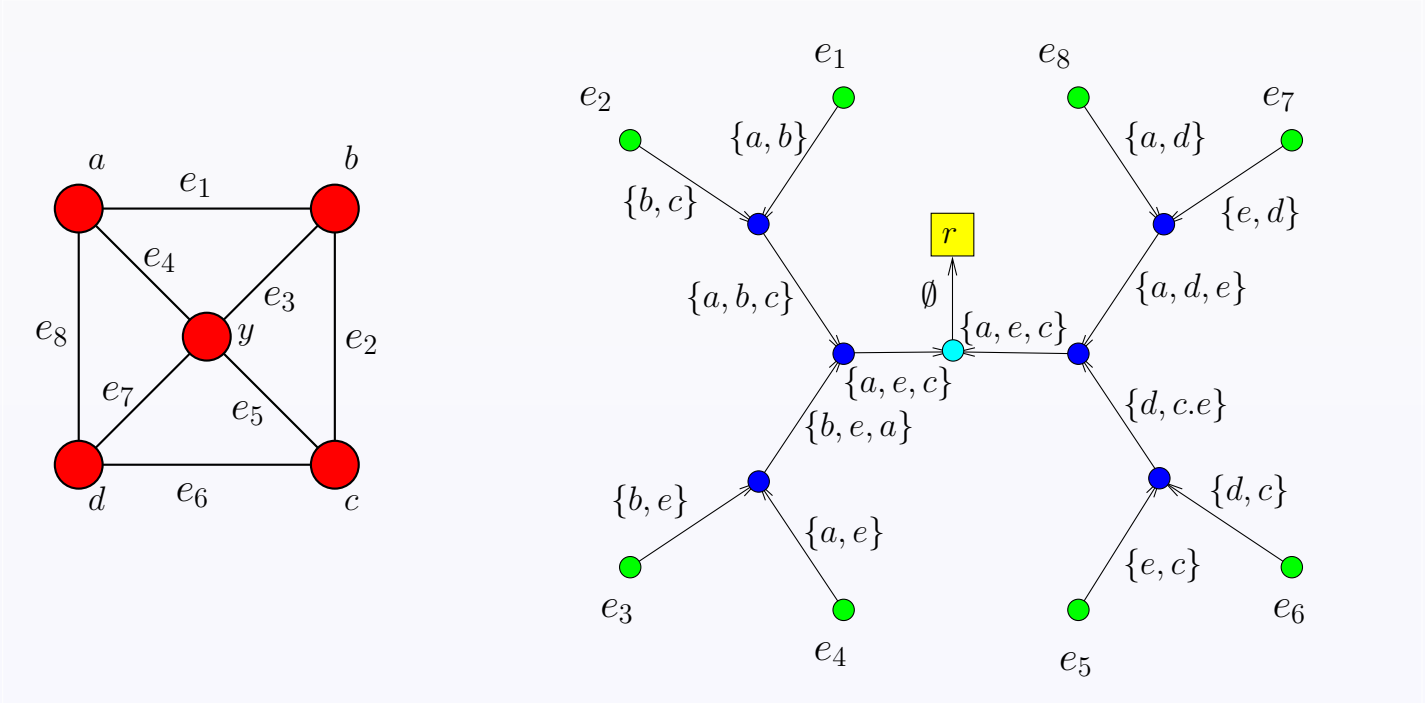
$\mathbf{tw}($  $) = \mathbf{bw}($  $) = 6$

$\mathbf{bw}(K_6) = 4 < \mathbf{tw}(K_6) = 5$

Dynamic programming for graphs of small branchwidth

Given a branch decomposition (T, τ), (of small width)

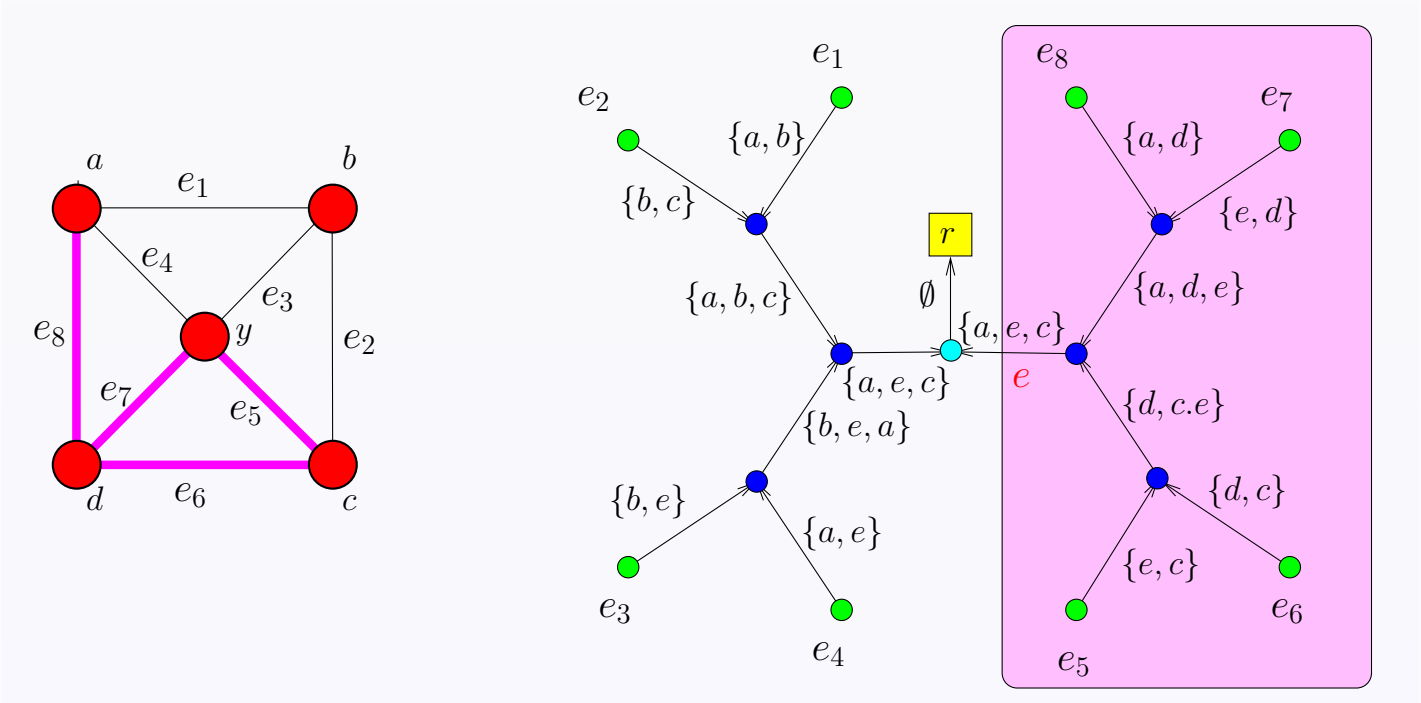
1. Root T to some vertex r without preimage



For each $e \in E(T)$, we denote as G_e the graph induced by the edges mapped bellow e.

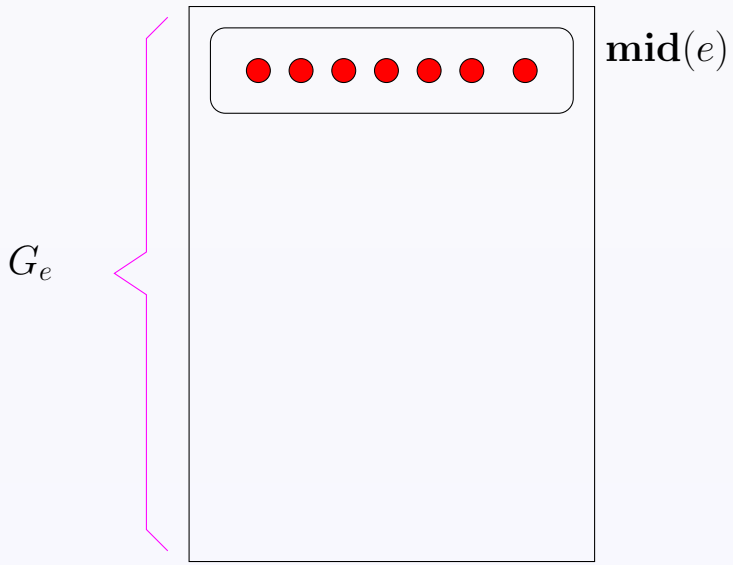
Given a branch decomposition (T, τ), (of small width)

1. Root T to some vertex r without preimage

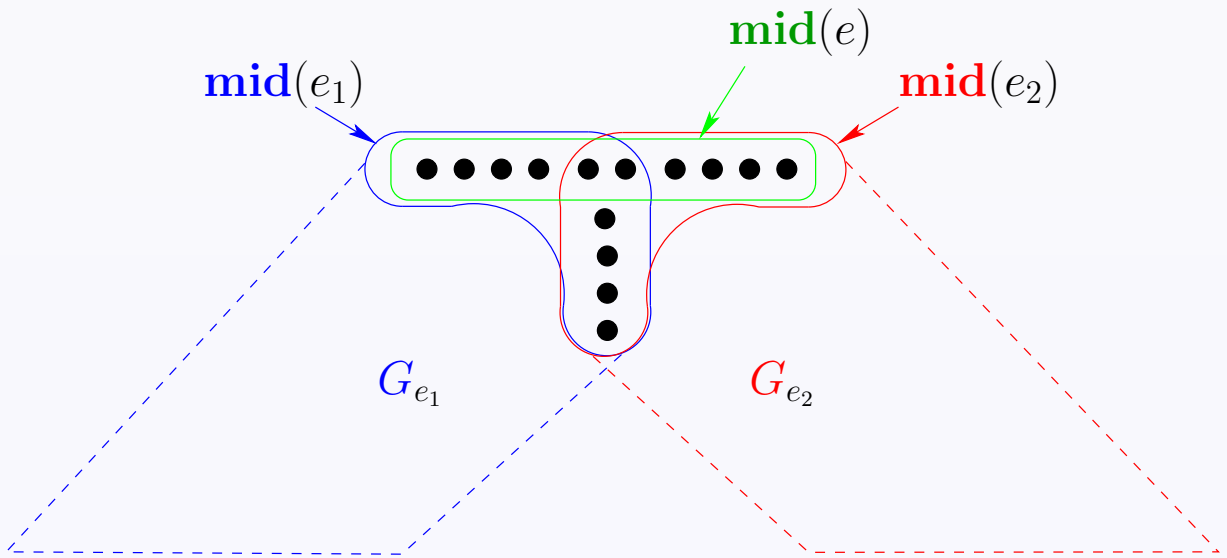


For each $e \in E(T)$, we denote as G_e the graph induced by the edges mapped bellow e.

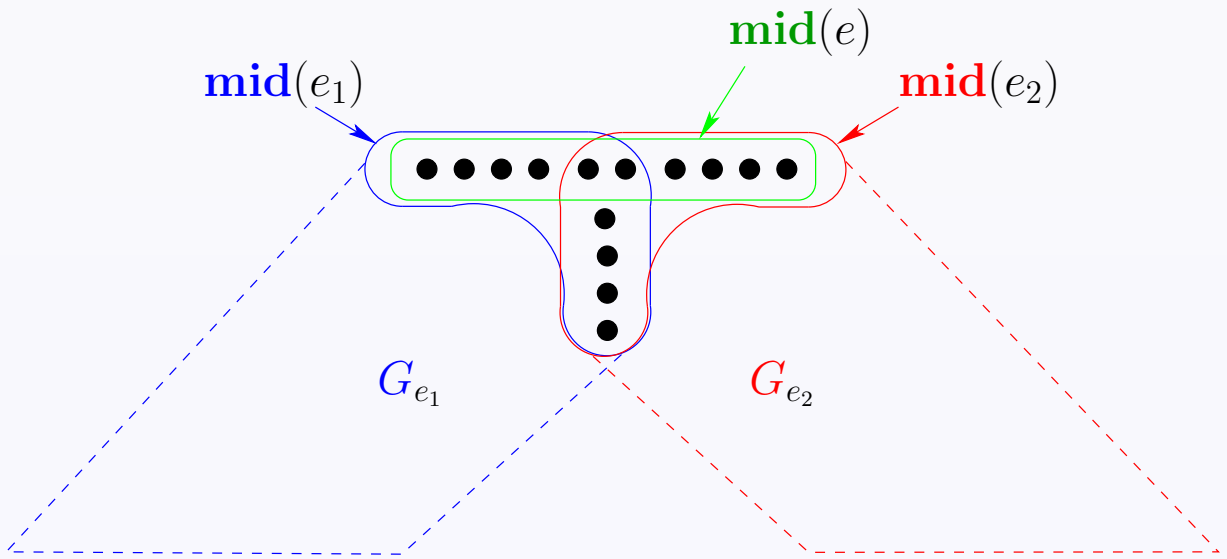
2. Define, for each $e \in E(T)$, a table encoding the information of a partial solution for G_e as restricted to $\mathbf{mid}(e)$. The values of this table for the root node should provide a global answer.



3. Define the values of this table for the leaf nodes



3. Define the values of this table for the leaf nodes
4. Provide the way to compute the table of an edge using the tables of its children edge.



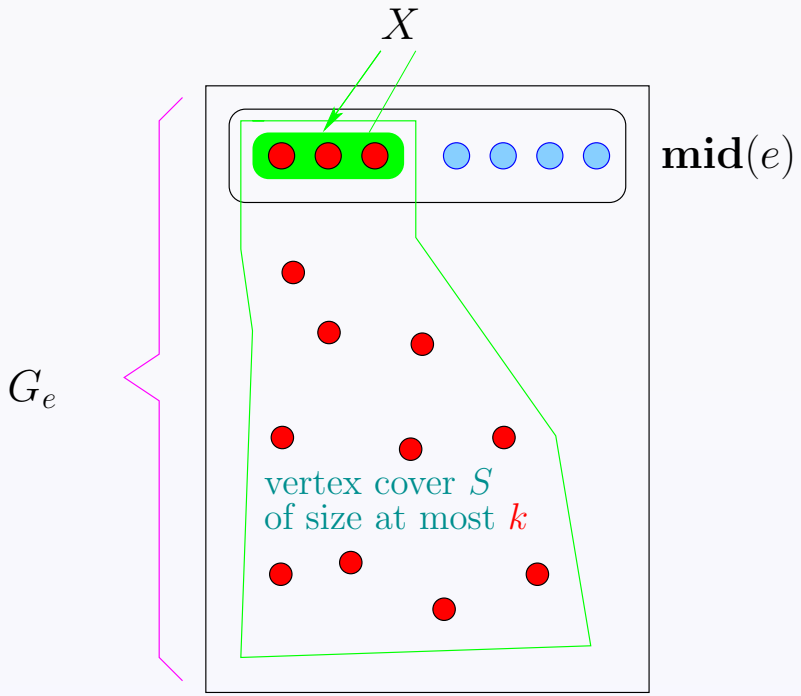
An example: Vertex Cover

Let G be a graph and $X, X' \subseteq V(G)$ where $X \cap X' = \emptyset$.

We say that $\mathbf{vc}(G, X, X') \leq k$ if G contains a vertex cover S where $|S| \leq k$ and $X \subseteq S \subseteq V(G) \setminus X'$.

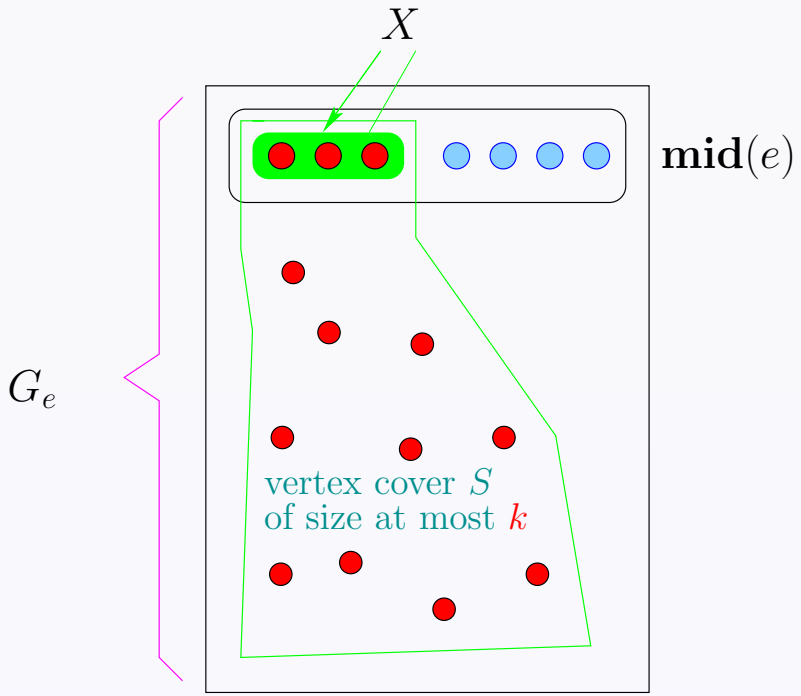
Let G be a graph and $X, X' \subseteq V(G)$ where $X \cap X' = \emptyset$.

We say that $\mathbf{vc}(G, X, X') \leq k$ if G contains a vertex cover S where $|S| \leq k$ and $X \subseteq S \subseteq V(G) \setminus X'$.



Let G be a graph and $X, X' \subseteq V(G)$ where $X \cap X' = \emptyset$.

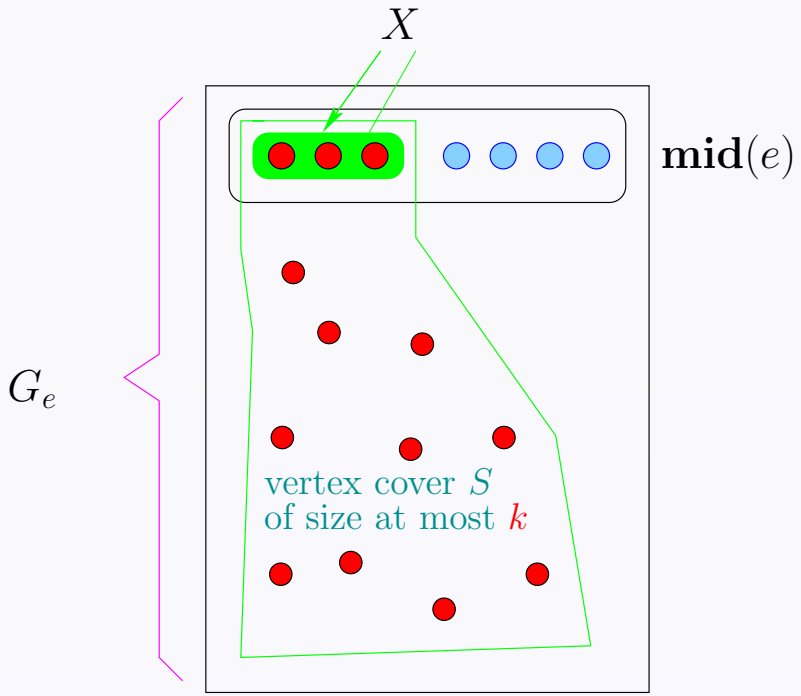
We say that $\mathbf{vc}(G, X, X') \leq k$ if G contains a vertex cover S where $|S| \leq k$ and $X \subseteq S \subseteq V(G) \setminus X'$.



Let $\mathcal{R}_e = \{(X, k) \mid X \subseteq \mathbf{mid}(e) \wedge \mathbf{vc}(G_e, X, \mathbf{mid}(e) \setminus X) \leq k\}$

Let G be a graph and $X, X' \subseteq V(G)$ where $X \cap X' = \emptyset$.

We say that $\mathbf{vc}(G, X, X') \leq k$ if G contains a vertex cover S where $|S| \leq k$ and $X \subseteq S \subseteq V(G) \setminus X'$.



Let $\mathcal{R}_e = \{(X, k) \mid X \subseteq \mathbf{mid}(e) \wedge \mathbf{vc}(G_e, X, \mathbf{mid}(e) \setminus X) \leq k\}$

observe that $\mathbf{vc}(G) \leq k$ iff $(\emptyset, k) \in \mathcal{R}_{e_r}$.

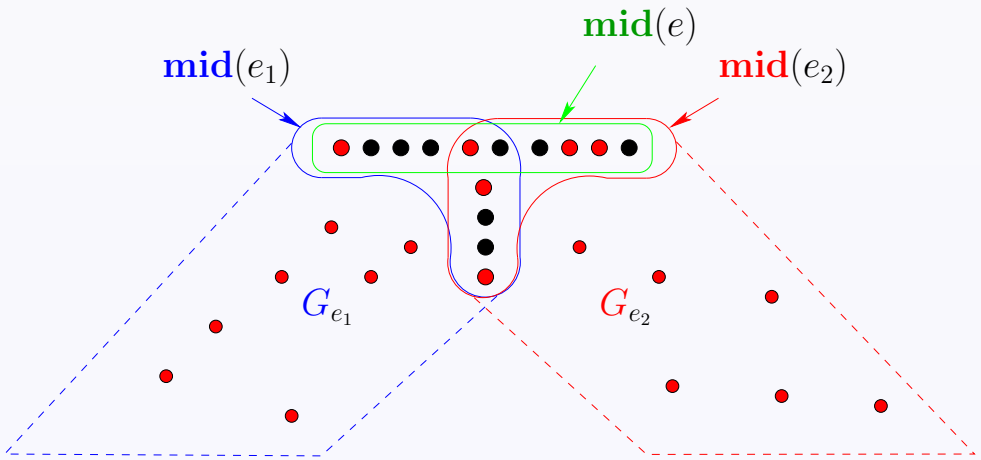
Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$\mathcal{R}_e = \begin{cases} \{(X,k) \mid X \subseteq e \wedge X \neq \emptyset \wedge k \geq |X|\} & \text{if } e \in L(T) \\ \{(X,k) \mid X \subseteq \mathbf{mid}(e) \wedge \exists (X_1,k_1) \in \mathcal{R}_{e_1}, \exists (X_2,k_2) \in \mathcal{R}_{e_2} : \\ \qquad (X_1 \cup X_2) \cap \mathbf{mid}(e) = X \wedge k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \notin L(T) \end{cases}$$

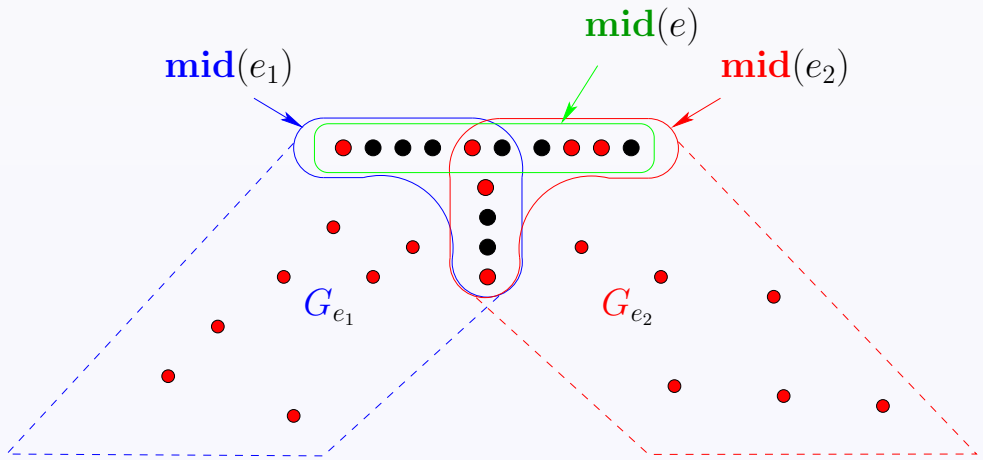
Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$\mathcal{R}_e = \begin{cases} \{(X,k) \mid X \subseteq e \wedge X \neq \emptyset \wedge k \geq |X|\} & \text{if } e \in L(T) \\ \{(X,k) \mid X \subseteq \mathbf{mid}(e) \wedge \exists (X_1,k_1) \in \mathcal{R}_{e_1}, \exists (X_2,k_2) \in \mathcal{R}_{e_2} : \\ \qquad (X_1 \cup X_2) \cap \mathbf{mid}(e) = X \wedge k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \notin L(T) \end{cases}$$



Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

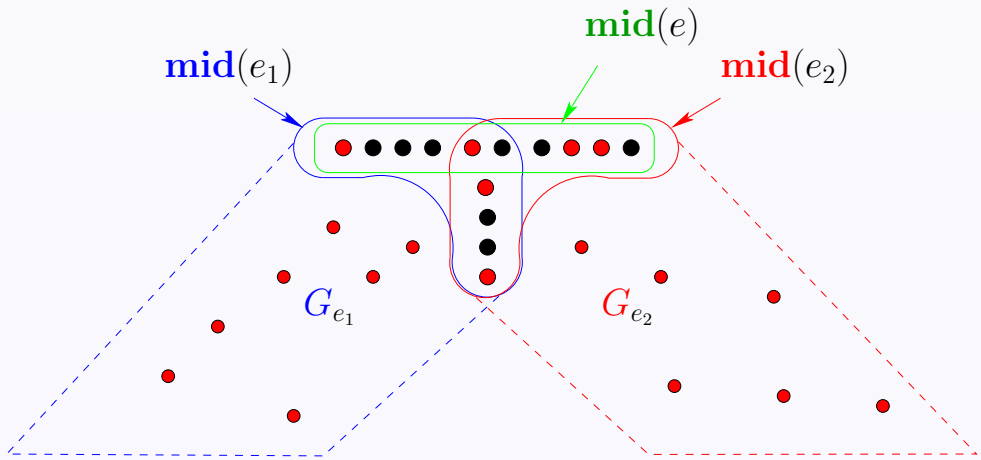
$$\mathcal{R}_e = \begin{cases} \{(X,k) \mid X \subseteq e \wedge X \neq \emptyset \wedge k \geq |X|\} & \text{if } e \in L(T) \\ \{(X,k) \mid X \subseteq \mathbf{mid}(e) \wedge \exists (X_1,k_1) \in \mathcal{R}_{e_1}, \exists (X_2,k_2) \in \mathcal{R}_{e_2} : \\ \qquad (X_1 \cup X_2) \cap \mathbf{mid}(e) = X \wedge k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \notin L(T) \end{cases}$$



- $\forall e \in E(T)$, $|\mathcal{R}_e| \leq 2^{|\mathbf{mid}(e)|} \cdot \ell$.

Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$\mathcal{R}_e = \begin{cases} \{(X,k) \mid X \subseteq e \wedge X \neq \emptyset \wedge k \geq |X|\} & \text{if } e \in L(T) \\ \{(X,k) \mid X \subseteq \mathbf{mid}(e) \wedge \exists (X_1,k_1) \in \mathcal{R}_{e_1}, \exists (X_2,k_2) \in \mathcal{R}_{e_2} : \\ \qquad (X_1 \cup X_2) \cap \mathbf{mid}(e) = X \wedge k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \notin L(T) \end{cases}$$



- $\forall e \in E(T)$, $|\mathcal{R}_e| \leq 2^{|\mathbf{mid}(e)|} \cdot \ell$.
- we can check whether $\mathbf{vc}(G) \leq \ell$ in $O(4^{\mathbf{bw}(G)} \cdot \ell^2 \cdot |V(T)|)$ steps.

Sphere-Cut Decompositions

Suppose that G is a planar graph embedded on the sphere $\mathcal{S}_0$

Sphere-Cut Decompositions

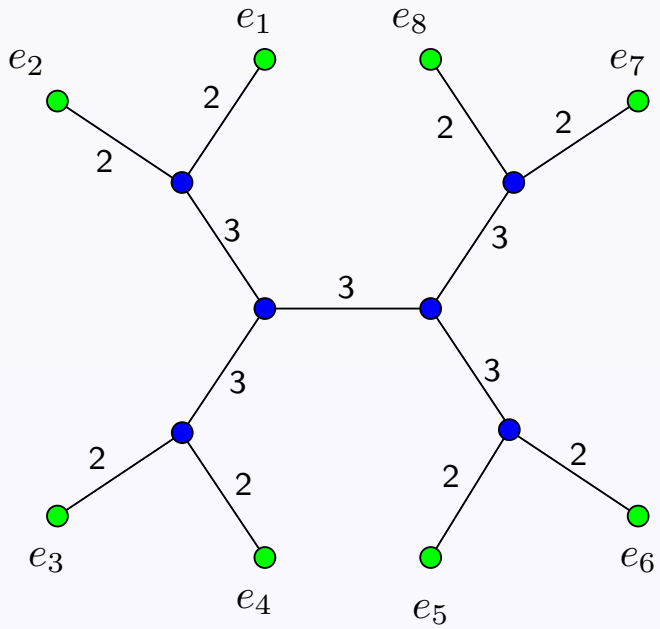
Suppose that G is a planar graph embedded on the sphere $\mathcal{S}_0$

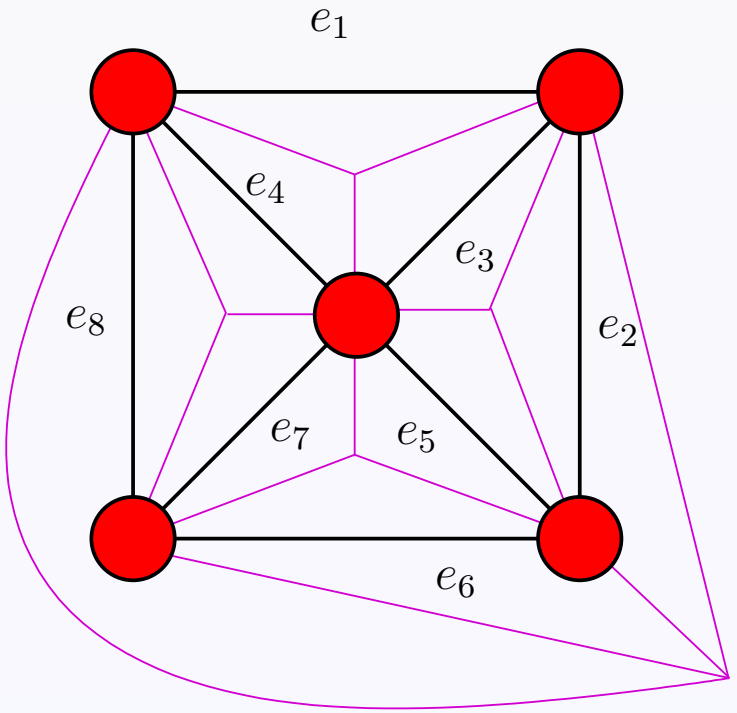
A *sphere-cut decomposition* of G is a branch decomposition (T, τ) where for any $e \in E(T)$, the vertices in $\mathbf{mid}(e)$ are the vertices in a Jordan courve of $\mathcal{S}_0$ – called *noose* – that meets no edges of G.

Sphere-Cut Decompositions

Suppose that G is a planar graph embedded on the sphere $\mathcal{S}_0$

A *sphere-cut decomposition* of G is a branch decomposition (T, τ) where for any $e \in E(T)$, the vertices in $\mathbf{mid}(e)$ are the vertices in a Jordan courve of $\mathcal{S}_0$ – called *noose* – that meets no edges of G.

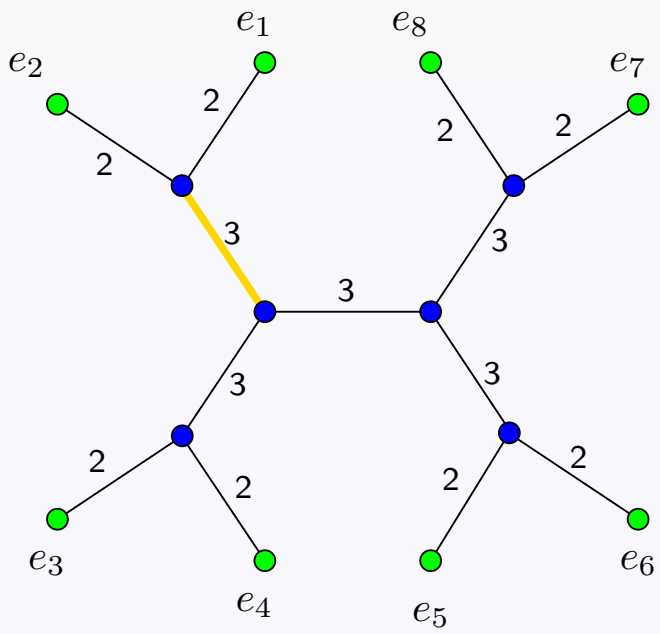


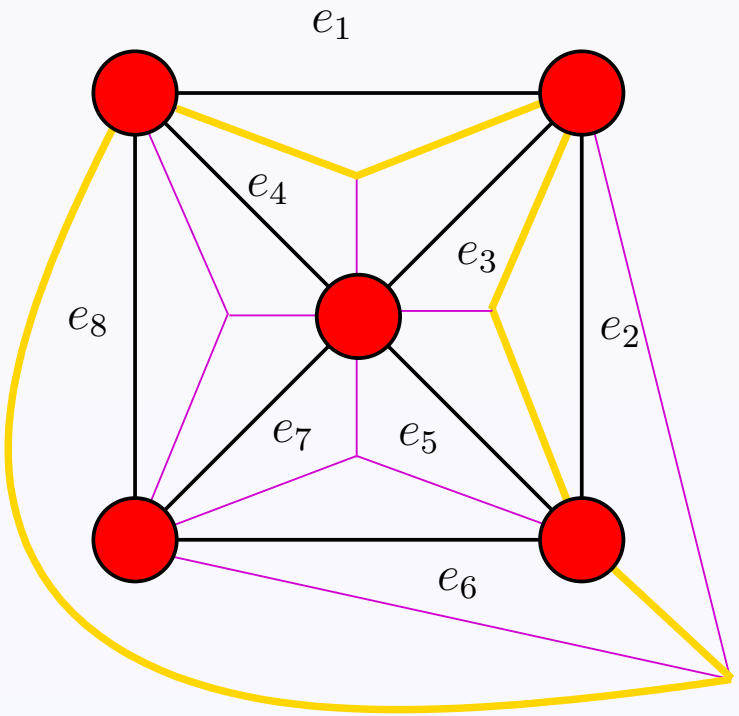


Sphere-Cut Decompositions

Suppose that G is a planar graph embedded on the sphere $\mathcal{S}_0$

A *sphere-cut decomposition* of G is a branch decomposition (T, τ) where for any $e \in E(T)$, the vertices in $\mathbf{mid}(e)$ are the vertices in a Jordan courve of $\mathcal{S}_0$ – called *noose* – that meets no edges of G.

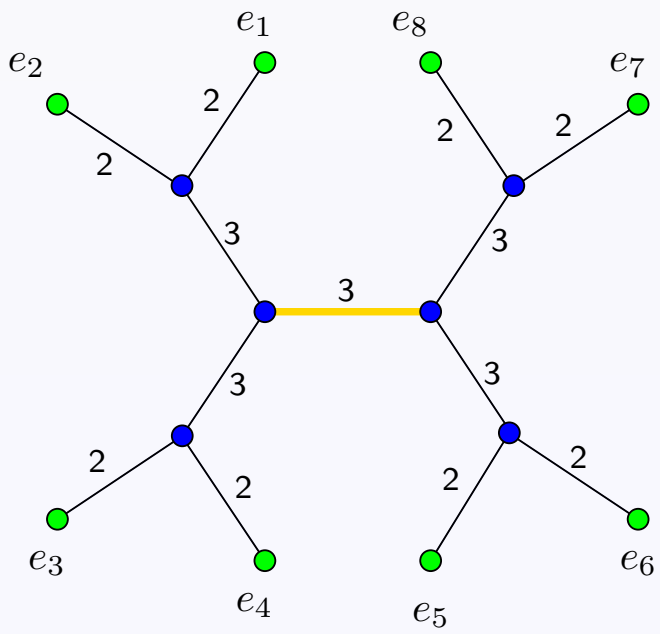


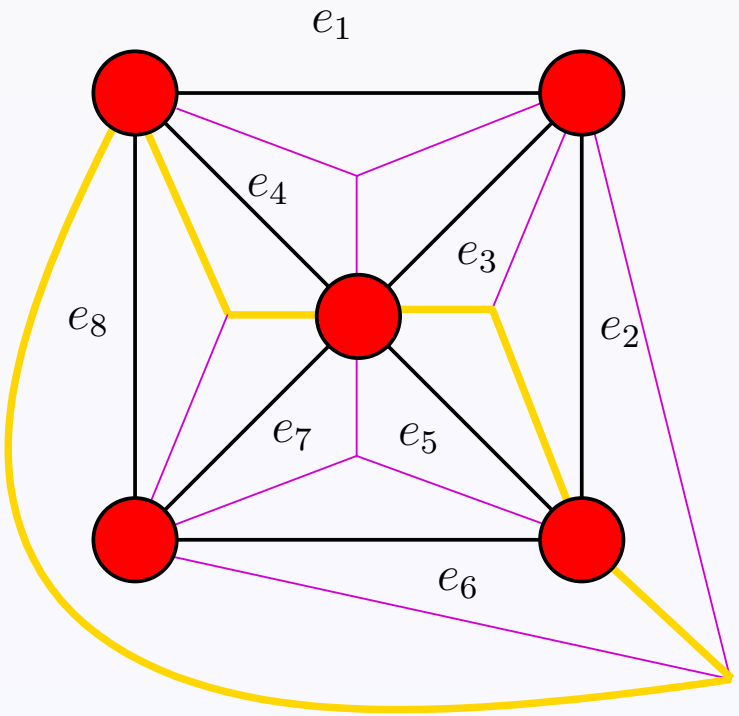


Sphere-Cut Decompositions

Suppose that G is a planar graph embedded on the sphere $\mathcal{S}_0$

A *sphere-cut decomposition* of G is a branch decomposition (T, τ) where for any $e \in E(T)$, the vertices in $\mathbf{mid}(e)$ are the vertices in a Jordan courve of $\mathcal{S}_0$ – called *noose* – that meets no edges of G.

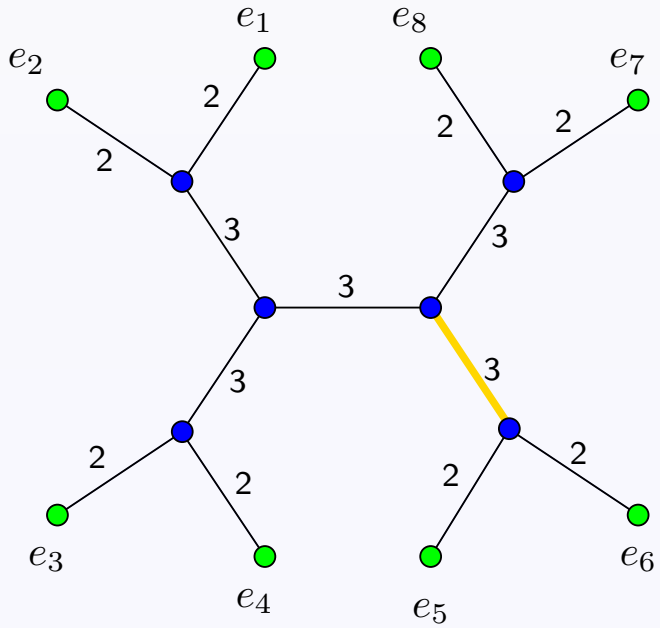


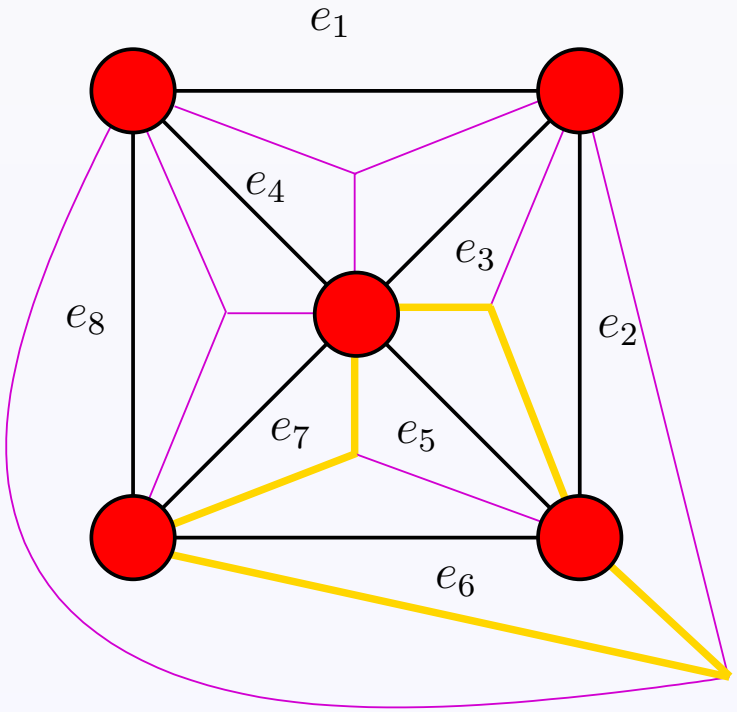


Sphere-Cut Decompositions

Suppose that G is a planar graph embedded on the sphere $\mathcal{S}_0$

A *sphere-cut decomposition* of G is a branch decomposition (T, τ) where for any $e \in E(T)$, the vertices in $\mathbf{mid}(e)$ are the vertices in a Jordan courve of $\mathcal{S}_0$ – called *noose* – that meets no edges of G.





Theorem: [Roberston & Seymour GM-X] *If G is planar and has a branch decomposition with width $\leq k$ then G has a sphere-cut decomposition of G with width $\leq k$ that cane be constructed in $O(n^3)$ steps.*

Theorem: [Roberston & Seymour GM-X] *If G is planar and has a branch decomposition with width $\leq k$ then G has a sphere-cut decomposition of G with width $\leq k$ that cane be constructed in $O(n^3)$ steps.*

For doing dynamic programming on a sphere cut decomposition (T, τ) again we define, for any $e \in E(T)$ the set $\texttt{pairs}(\mathbf{mid}(e))$ be the set of all pairings of $\mathbf{mid}(e)$

Theorem: [Roberston & Seymour GM-X] *If G is planar and has a branch decomposition with width $\leq k$ then G has a sphere-cut decomposition of G with width $\leq k$ that cane be constructed in $O(n^3)$ steps.*

For doing dynamic programming on a sphere cut decomposition (T, τ) again we define, for any $e \in E(T)$ the set $\texttt{pairs}(\mathbf{mid}(e))$ be the set of all pairings of $\mathbf{mid}(e)$

The "usual" bound for $\mathbf{mid}(e)$ is $2^{O(k \cdot \log k)}$

(recall that $|\mathbf{mid}(e)| = \Omega(\frac{k}{2}!)$)

Theorem: [Roberston & Seymour GM-X] *If G is planar and has a branch decomposition with width $\leq k$ then G has a sphere-cut decomposition of G with width $\leq k$ that cane be constructed in $O(n^3)$ steps.*

For doing dynamic programming on a sphere cut decomposition (T, τ) again we define, for any $e \in E(T)$ the set $\texttt{pairs}(\mathbf{mid}(e))$ be the set of all pairings of $\mathbf{mid}(e)$

The "usual" bound for $\mathbf{mid}(e)$ is $2^{O(k \cdot \log k)}$

(recall that $|\mathbf{mid}(e)| = \Omega(\frac{k}{2}!)$)

However, we now have that

1: the vertices of $\mathbf{mid}(e)$ lay on the boundary of a disk and

Theorem: [Roberston & Seymour GM-X] *If G is planar and has a branch decomposition with width $\leq k$ then G has a sphere-cut decomposition of G with width $\leq k$ that cane be constructed in $O(n^3)$ steps.*

For doing dynamic programming on a sphere cut decomposition (T, τ) again we define, for any $e \in E(T)$ the set $\texttt{pairs}(\mathbf{mid}(e))$ be the set of all pairings of $\mathbf{mid}(e)$

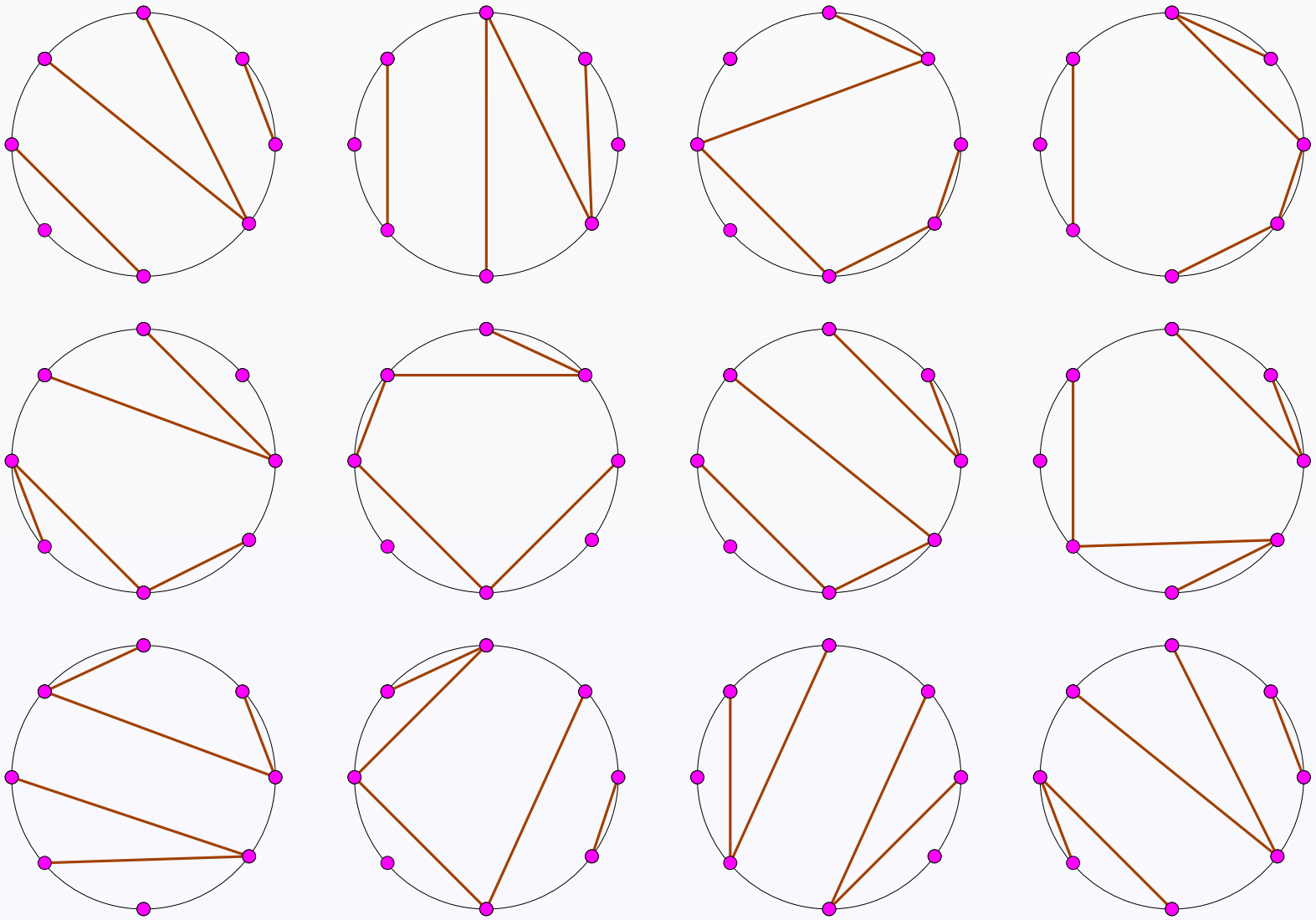
The "usual" bound for $\mathbf{mid}(e)$ is $2^{O(k \cdot \log k)}$

(recall that $|\mathbf{mid}(e)| = \Omega(\frac{k}{2}!)$)

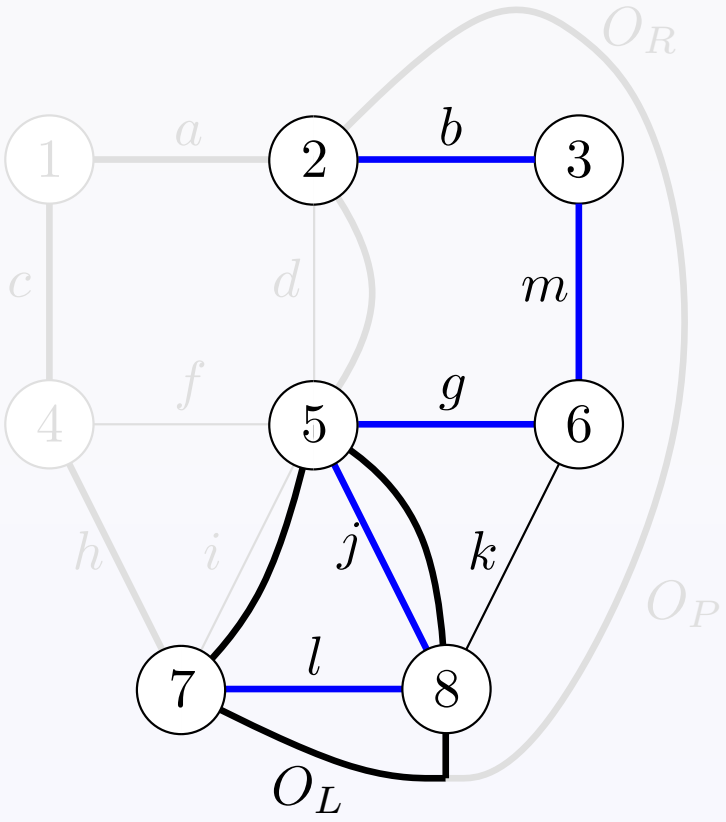
However, we now have that

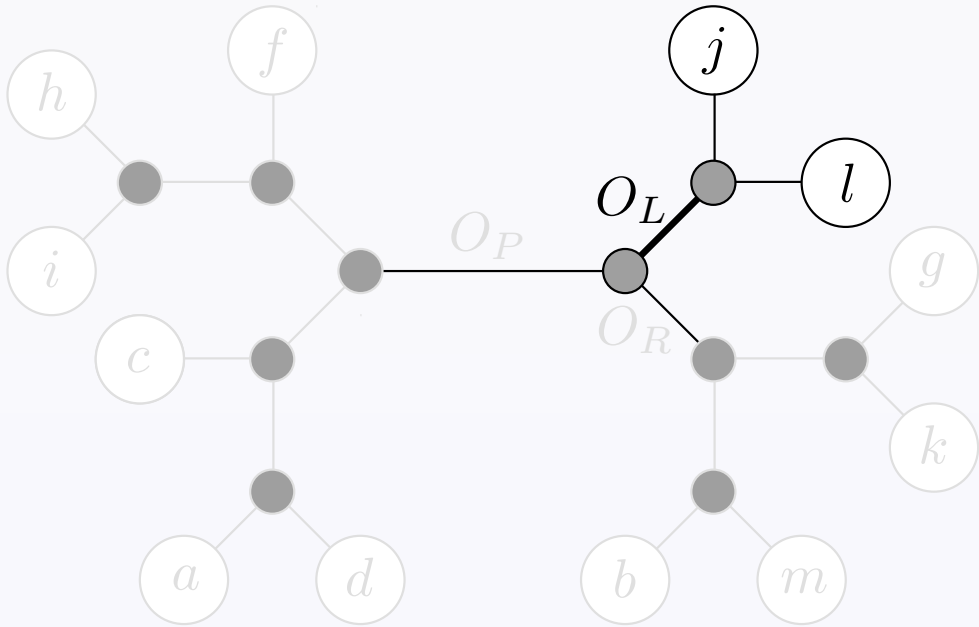
1: the vertices of $\mathbf{mid}(e)$ lay on the boundary of a disk and

2: the pairings cannot be crossing because of planarity.

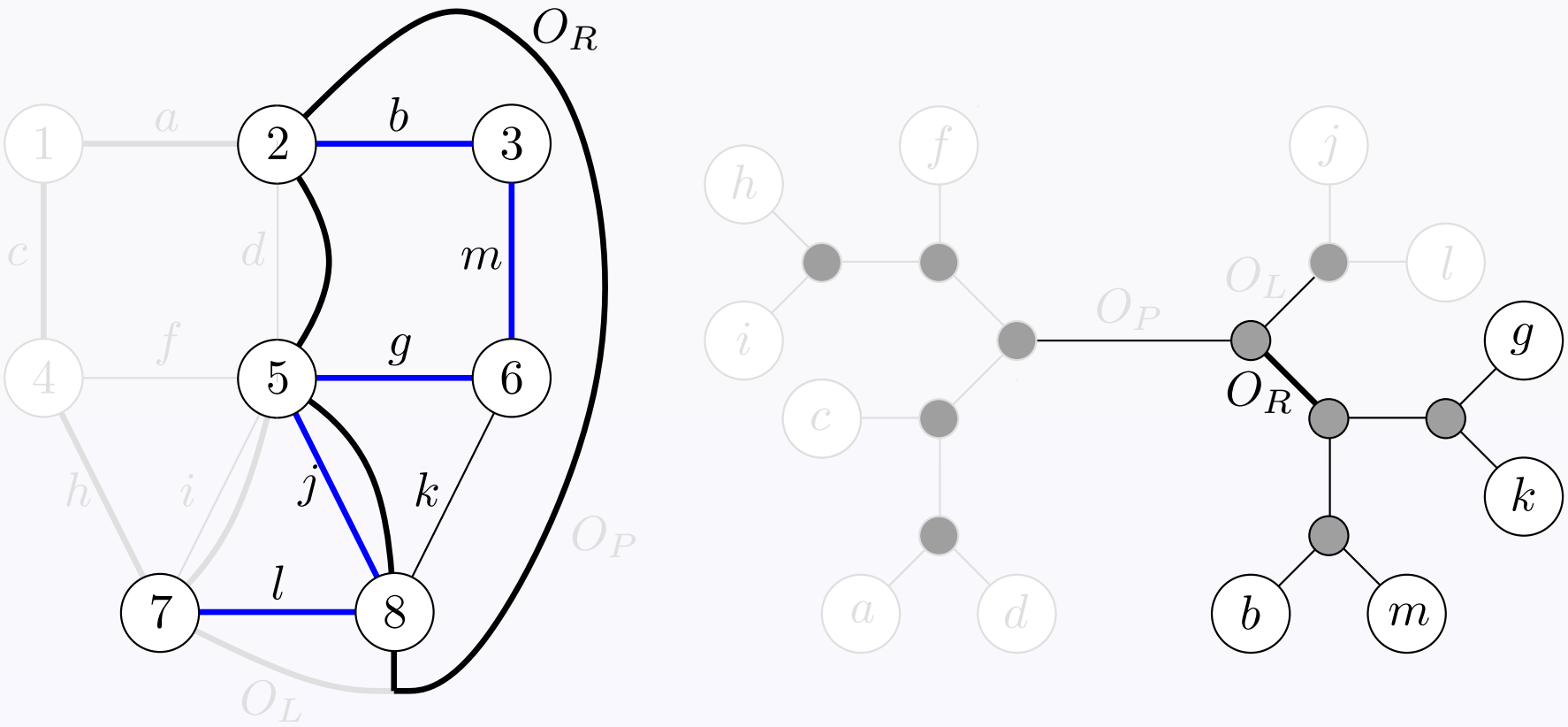


Non crossing pairings

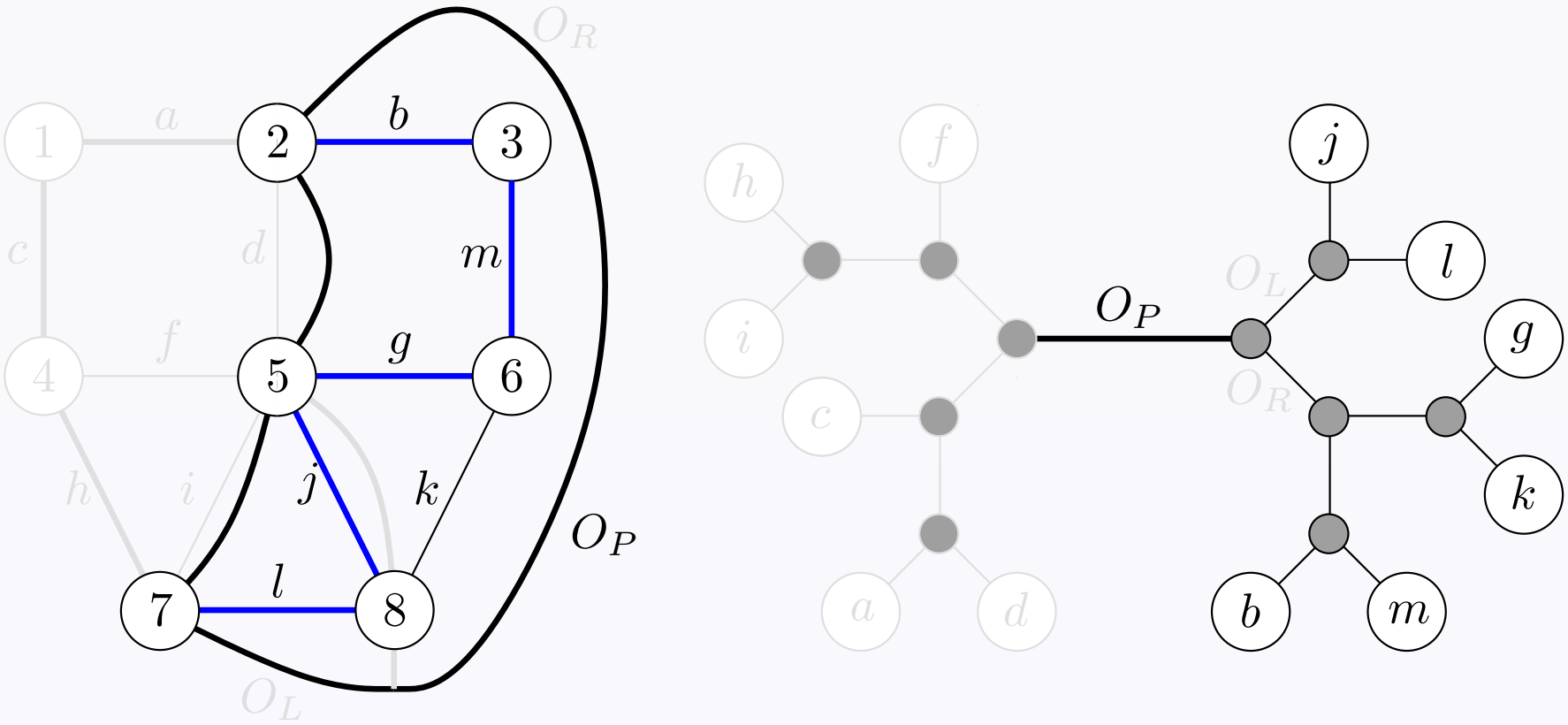




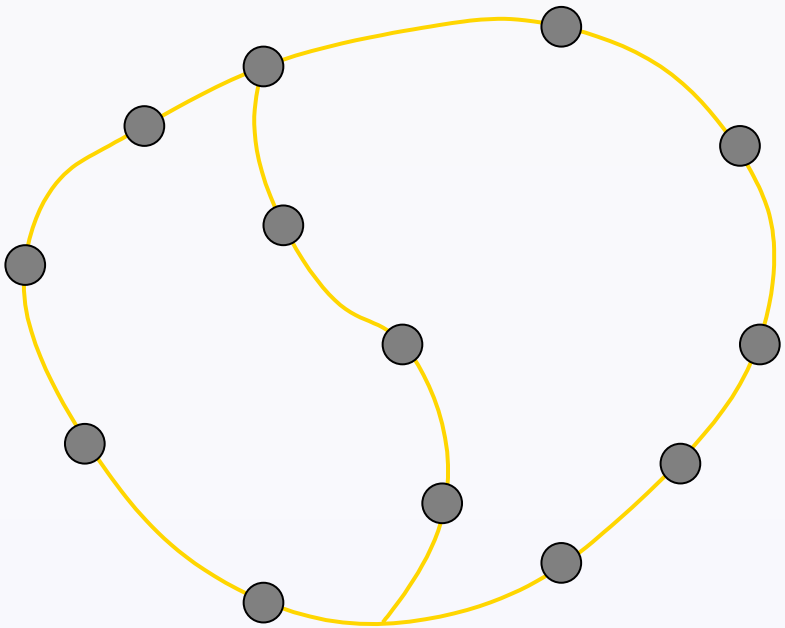
The two nooses O_L and O_R of the two children for a nose O_P for the parent.



The two nooses O_L and O_R of the two children for a nose O_P for the parent.

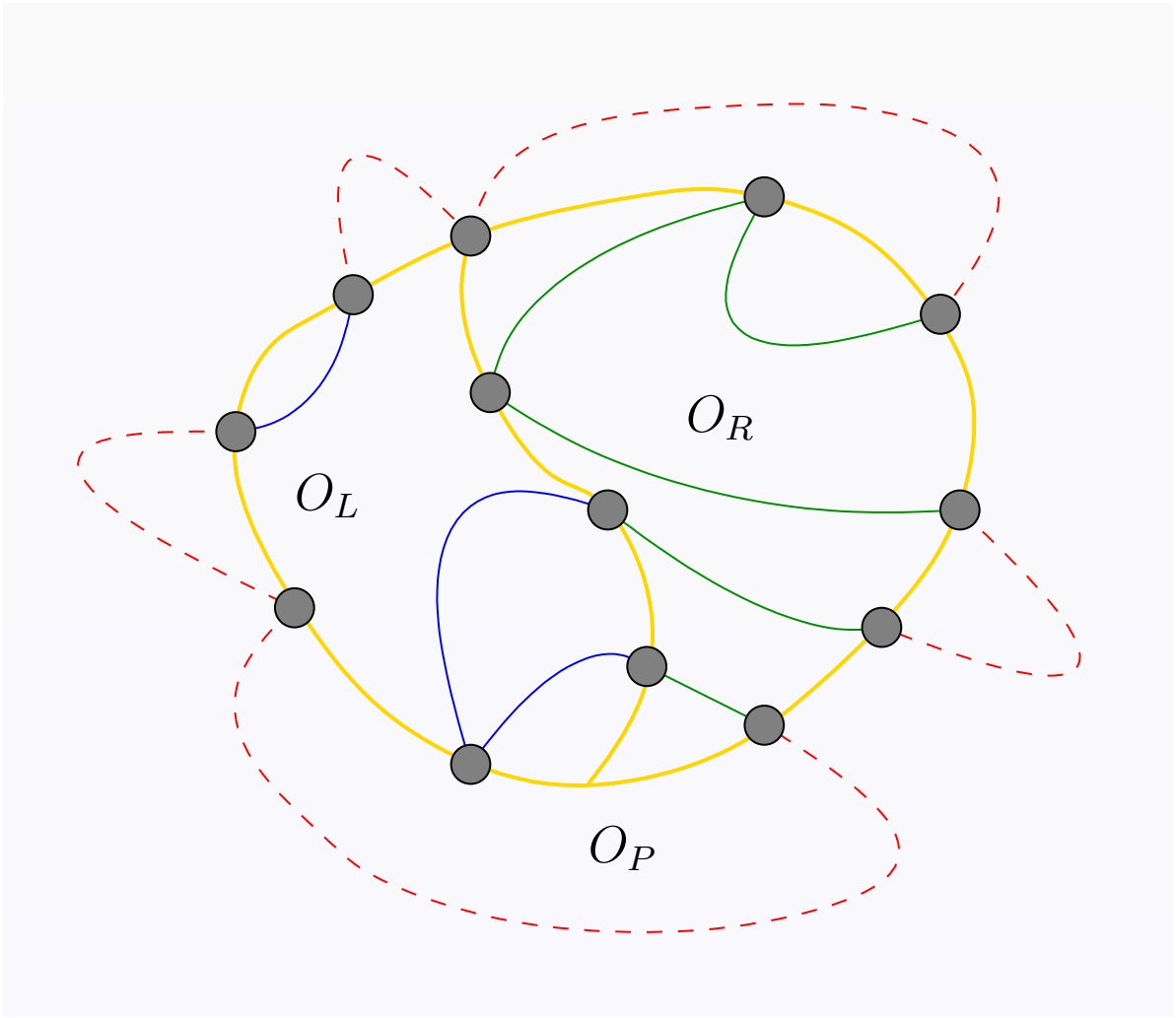


The two nooses O_L and O_R of the two children for a nose O_P for the parent.

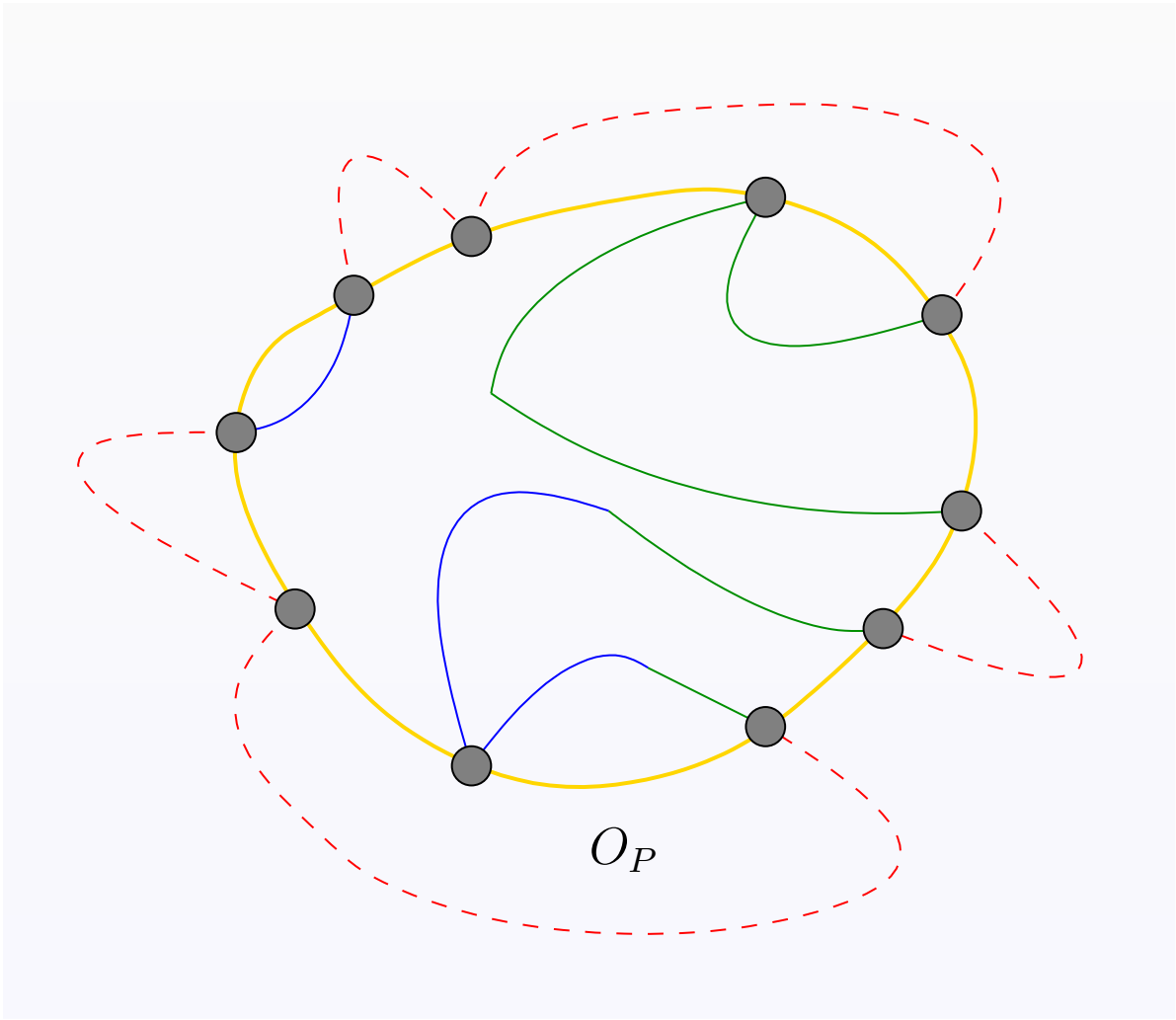


O_P

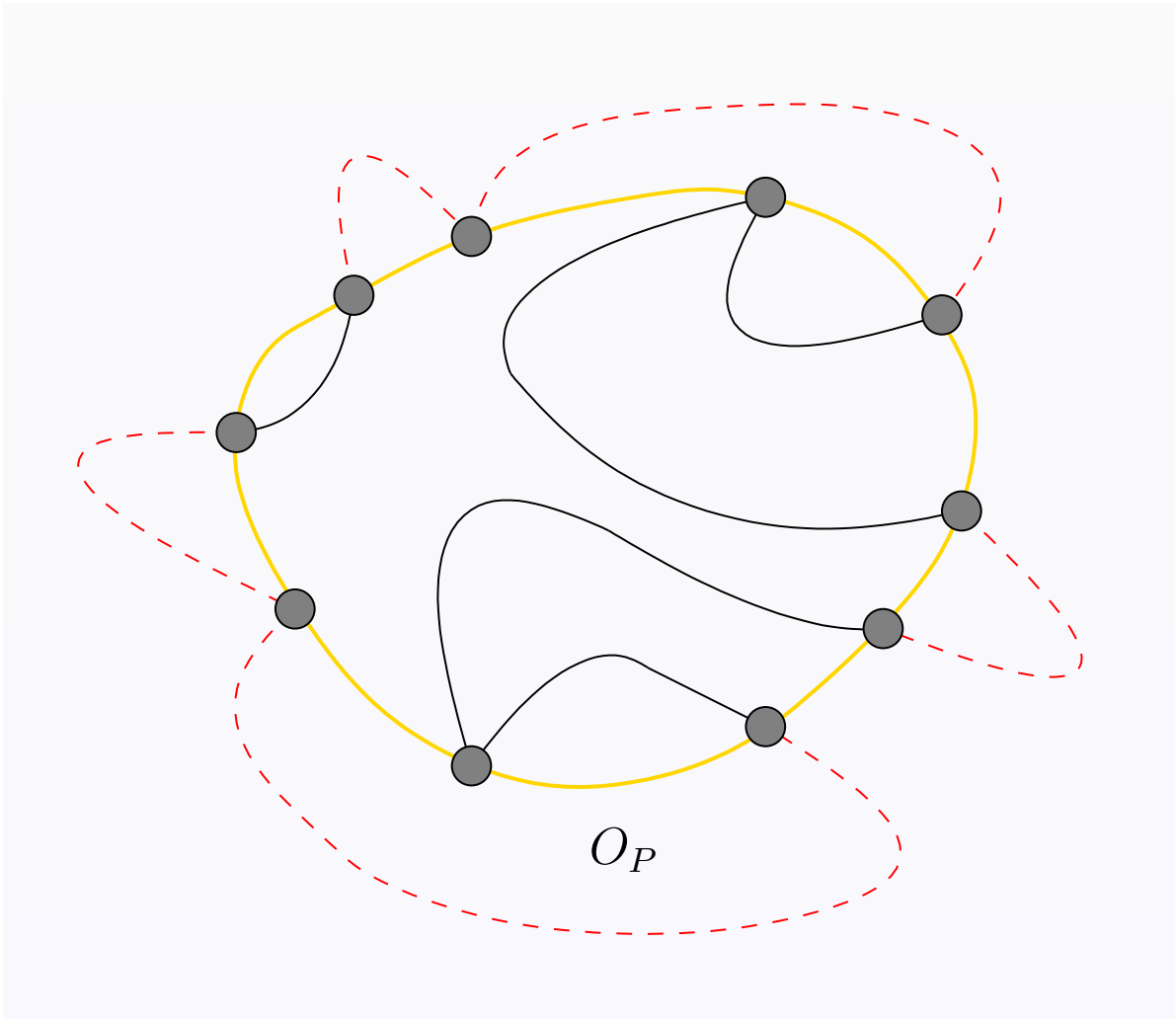
In case of Hamiltonial Cycle, each non-crossing pair on O_P is the union of two non-crossing pairs on O_L and O_R.



In case of Hamiltonial Cycle, each non-crossing pair on O_P is the union of two non-crossing pairs on O_L and O_R.



In case of Hamiltonial Cycle, each non-crossing pair on O_P is the union of two non-crossing pairs on O_L and O_R.



In case of Hamiltonial Cycle, each non-crossing pair on O_P is the union of two non-crossing pairs on O_L and O_R.

Catalan Structures

It follows that $\texttt{pairs}(\mathbf{mid}(e)) = O(C(|\mathbf{mid}(e)|)) = O(C(k))$

Where $C(k)$ is the k-th *Catalan Number*.

Catalan Structures

It follows that $\texttt{pairs}(\mathbf{mid}(e)) = O(C(|\mathbf{mid}(e)|)) = O(C(k))$

Where $C(k)$ is the k-th *Catalan Number*.

It is known that $C(k) \sim \frac{4^k}{k^{3/2}\sqrt{\Pi}} = 2^{O(k)}$

Catalan Structures

It follows that $\texttt{pairs}(\mathbf{mid}(e)) = O(C(|\mathbf{mid}(e)|)) = O(C(k))$

Where $C(k)$ is the k-th *Catalan Number*.

It is known that $C(k) \sim \frac{4^k}{k^{3/2}\sqrt{\Pi}} = 2^{O(k)}$

Therefore: dynamic programming for Hamiltonian Cycle of a planar graph G on a sphere cut decompositions of G with width $\leq k$ takes $2^{O(k)} \cdot O(n)$ steps.

- The same holds for several other problems where an analogue of $\texttt{pairs}(\mathbf{mid}(e))$ can be defined for controlling the size of the tables in dynamic programming.

- The same holds for several other problems where an analogue of $\texttt{pairs}(\mathbf{mid}(e))$ can be defined for controlling the size of the tables in dynamic programming.

In general: These are pairs where the tables encode pairings.

- The same holds for several other problems where an analogue of $\texttt{pairs}(\mathbf{mid}(e))$ can be defined for controlling the size of the tables in dynamic programming.

In general: These are pairs where the tables encode pairings.

- Like that one can design $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$ step algorithms for the planar versions of Cycle Cover, Path Cover, Longest Path, Longest Cycle, Hamiltonian Cycle, and Graph Metric TSP and others.

[Dorn, Penninkx, Bodlaender, and Fomin. ICALP 2005]

- The same holds for several other problems where an analogue of $\texttt{pairs}(\mathbf{mid}(e))$ can be defined for controlling the size of the tables in dynamic programming.

In general: These are pairs where the tables encode pairings.

- Like that one can design $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$ step algorithms for the planar versions of Cycle Cover, Path Cover, Longest Path, Longest Cycle, Hamiltonian Cycle, and Graph Metric TSP and others.

[Dorn, Penninkx, Bodlaender, and Fomin. ICALP 2005]

The idea of using properties of the embedding (for pairings) has been extended for bounded genus graphs in [Dorn, Fomin, and Thilikos. SWAT 2006]

and for H-minor-free graphs in [Dorn, Fomin, and Thilikos. SODA 2008]

Common idea: Planarization

For more complicated problems planarization becomes very hard to handle as here tables encode packings instead of pairings.

For more complicated problems planarization becomes very hard to handle as here tables encode packings instead of pairings.

For this, single exponential dynamic programming has been done by

For more complicated problems planarization becomes very hard to handle as here tables encode packings instead of pairings.

For this, single exponential dynamic programming has been done by

1. Moving from sphere cut decompositions to *surface cut decompositions*

For more complicated problems planarization becomes very hard to handle as here tables encode packings instead of pairings.

For this, single exponential dynamic programming has been done by

1. Moving from sphere cut decompositions to *surface cut decompositions*
2. Counting non intersecting packings on surfaces with boundary.

For more complicated problems planarization becomes very hard to handle as here tables encode packings instead of pairings.

For this, single exponential dynamic programming has been done by

1. Moving from sphere cut decompositions to *surface cut decompositions*
2. Counting non intersecting packings on surfaces with boundary.

[Sau, Rué, Thilikos, TALG 2014]

For more complicated problems planarization becomes very hard to handle as here tables encode packings instead of pairings.

For this, single exponential dynamic programming has been done by

1. Moving from sphere cut decompositions to *surface cut decompositions*
2. Counting non intersecting packings on surfaces with boundary.

[Sau, Rué, Thilikos, TALG 2014]

Extensions/alternatives:

- For H-minor free graphs: [Sau, Rué, Thilikos, COCOON 2012]
- Surface split decompositions: [Bonsma, STACS 2012]
- Brick Decompositions: [Cohen-Addad & de Mesmay, ESA 2015]

Part 4, Thursday 10/05/2016 - 11:00–12:30 (90')

Bidimensionality

Subexponential parameterized algorithms

The minor relation

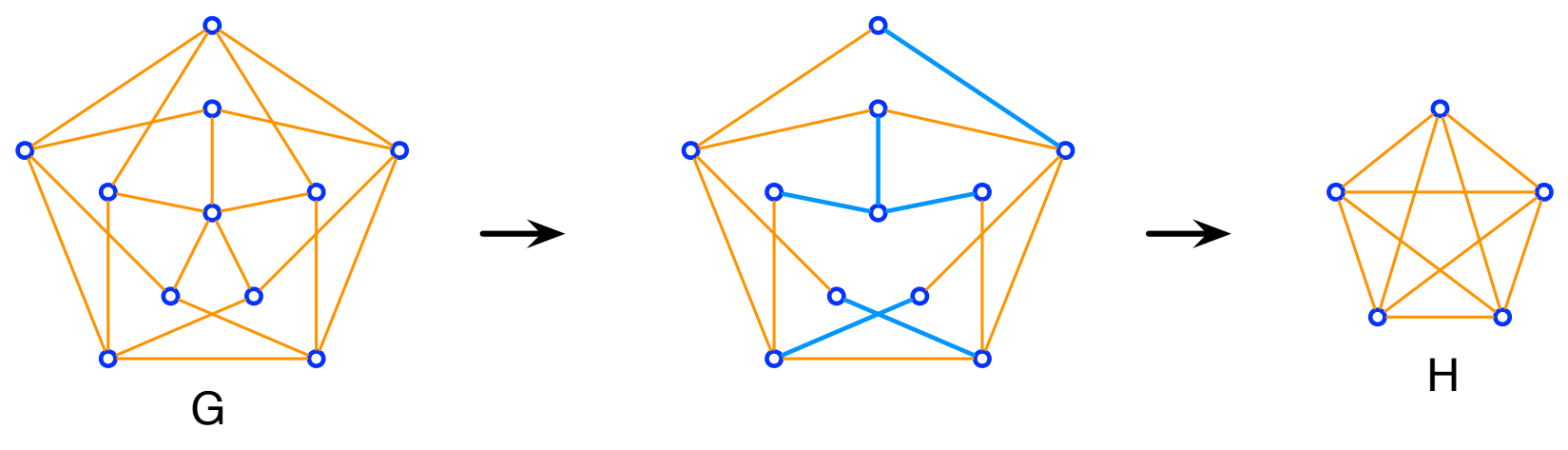
H is a minor of G ($H \leq G$):

H occurs from a subgraph of G by applying edge contractions

The minor relation

H is a minor of G ($H \leq G$):

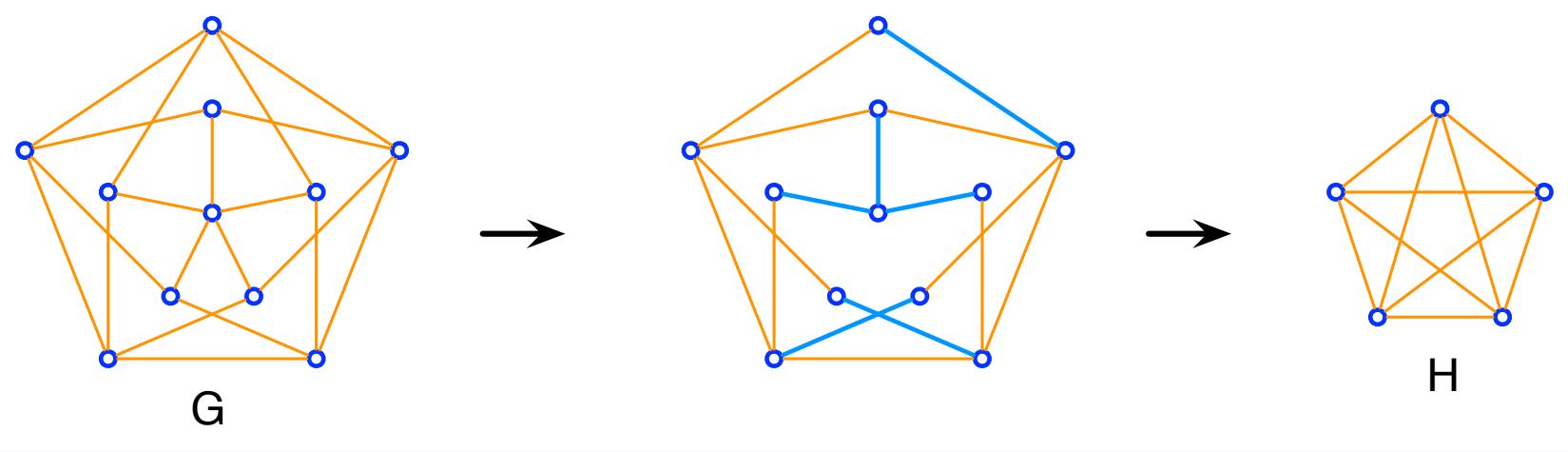
H occurs from a subgraph of G by applying edge contractions



The minor relation

H is a minor of G ($H \leq G$):

H occurs from a subgraph of G by applying edge contractions



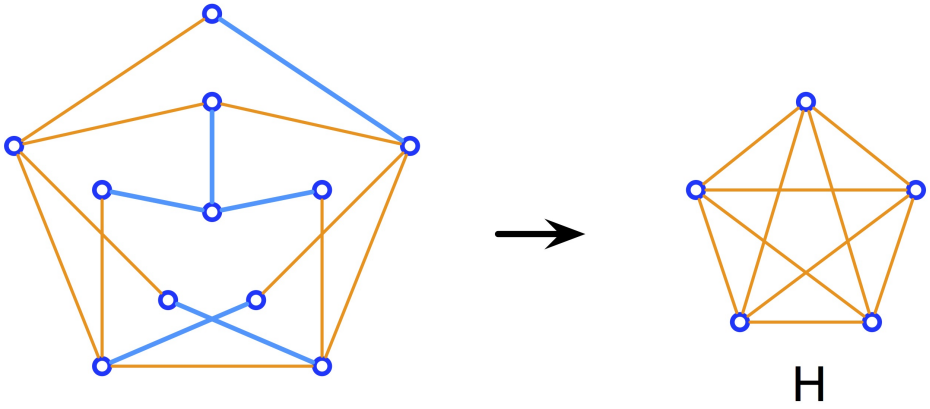
A graph class $\mathcal{G}$ is *minor-closed* if:

every minor of a graph in $\mathcal{G}$ is also a graph in $\mathcal{G}$

The contraction relation

H is a contraction of G ($H \leq_c G$):

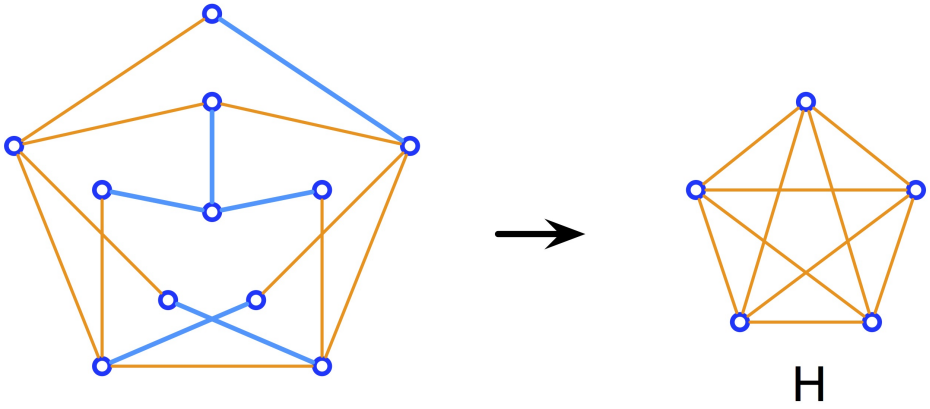
H occurs by applying edge contractions



The contraction relation

H is a contraction of G ($H \leq_c G$):

H occurs by applying edge contractions



A graph class $\mathcal{G}$ is *contraction-closed* if:

every contraction of a graph in $\mathcal{G}$ is also a graph in $\mathcal{G}$

Theorem: [Robertson & Seymour – main combinatorial result of GM] *Every infinite set of graphs contains two graphs comparable under the minor relation.*

Theorem: [Robertson & Seymour – main combinatorial result of GM] *Every infinite set of graphs contains two graphs comparable under the minor relation.*

Equivalently: Graphs are Well Quasi Ordered w.r.t. the minor relation

Theorem: [Robertson & Seymour – main combinatorial result of GM] *Every infinite set of graphs contains two graphs comparable under the minor relation.*

Equivalently: Graphs are Well Quasi Ordered w.r.t. the minor relation

Let G be a minor-closed graph class.

- $\mathbf{obs}(\mathcal{G})$ is the set of minor-minimal elements not in $\mathcal{G}$.

Let G be a minor-closed graph class.

- $\mathbf{obs}(\mathcal{G})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathcal{G}$ is minor-closed, then $G \in \mathcal{G} \iff \forall H \in \mathbf{obs}(\mathcal{G})\ H \not\leq G$

Let G be a minor-closed graph class.

- $\mathbf{obs}(\mathcal{G})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathcal{G}$ is minor-closed, then $G \in \mathcal{G} \iff \forall H \in \mathbf{obs}(\mathcal{G})\ H \not\leq G$

Consequence of R&S theorem: $|\mathbf{obs}(\mathcal{G})| < \aleph_0$

Let G be a minor-closed graph class.

- $\mathbf{obs}(\mathcal{G})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathcal{G}$ is minor-closed, then $G \in \mathcal{G} \iff \forall H \in \mathbf{obs}(\mathcal{G})\ H \not\leq G$

Consequence of R&S theorem: $|\mathbf{obs}(\mathcal{G})| < \aleph_0$

Theorem: [Robertson & Seymour – main algorithmic consequence of GM] *For every H, checking whether $H \leq G$ can be done in $O(n^3)$ steps.*

Let $\mathcal{G}$ be a minor-closed graph class.

- $\mathbf{obs}(\mathcal{G})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathcal{G}$ is minor-closed, then $G \in \mathcal{G} \iff \forall H \in \mathbf{obs}(\mathcal{G})\ H \not\leq G$

Consequence of R&S theorem: $|\mathbf{obs}(\mathcal{G})| < \aleph_0$

Theorem: [Robertson & Seymour – main algorithmic consequence of GM] *For every H, checking whether $H \leq G$ can be done in $O(n^3)$ steps.*

- Meta-Algorithmic Consequence: For every minor-closed graph class $\mathcal{G}$, the problem asking whether $G \in \mathcal{G}$ belongs in PTIME,

i.e., can be solved in $O(n^3)$ steps!

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

$$\mathbf{p}(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

$$\mathbf{p}(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

In any case, we call a set S where $|S| = \mathbf{p}(G)$ *solution certificate* for $\mathbf{p}(G)$

We call such parameters *graph optimization* parameters.

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

$$\mathbf{p}(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

In any case, we call a set S where $|S| = \mathbf{p}(G)$ *solution certificate* for $\mathbf{p}(G)$

We call such parameters *graph optimization* parameters.

Three examples:

- Vertex Cover, $\mathbf{vc}(G)$: min, $\phi(G, S)=$ $\boxed{\forall e \in E(G)\ e \cap S \neq \emptyset}$

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

$$\mathbf{p}(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

In any case, we call a set S where $|S| = \mathbf{p}(G)$ *solution certificate* for $\mathbf{p}(G)$

We call such parameters *graph optimization* parameters.

Three examples:

- Vertex Cover, $\mathbf{vc}(G)$: min, $\phi(G, S)=$ $\boxed{\forall e \in E(G)\ e \cap S \neq \emptyset}$
- Dominating Set, $\mathbf{dc}(G)$: min, $\phi(G, S)=$ $\boxed{V(G) = N_G(S)}$

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

$$\mathbf{p}(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

In any case, we call a set S where $|S| = \mathbf{p}(G)$ *solution certificate* for $\mathbf{p}(G)$

We call such parameters *graph optimization* parameters.

Three examples:

- Vertex Cover, $\mathbf{vc}(G)$: min, $\phi(G, S)=$ $\boxed{\forall e \in E(G)\ e \cap S \neq \emptyset}$
- Dominating Set, $\mathbf{dc}(G)$: min, $\phi(G, S)=$ $\boxed{V(G) = N_G(S)}$
- Longest Path, $\mathbf{pl}(G)$: max, $\phi(G, S)=$ $\boxed{G[S] \text{ is a path}\}$

Graph optimization parameters

Graph parameter: a function $\mathbf{p} : \mathcal{G}_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $\mathbf{p}$ defined as follows

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \wedge \boxed{\phi(G,S)=\texttt{true}}\}$$

$$\mathbf{p}(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \wedge \boxed{\phi(G,S)=\texttt{true}}\}$$

In any case, we call a set S where $|S| = \mathbf{p}(G)$ *solution certificate* for $\mathbf{p}(G)$

We call such parameters *graph optimization* parameters.

Three examples:

- Vertex Cover, $\mathbf{vc}(G)$: min, $\phi(G,S)=$ $\boxed{\forall e \in E(G)\ e \cap S \neq \emptyset}$
- Dominating Set, $\mathbf{dc}(G)$: min, $\phi(G,S)=$ $\boxed{V(G) = N_G(S)}$
- Longest Path, $\mathbf{pl}(G)$: max, $\phi(G,S)=$ $\boxed{G[S] \text{ is a path}\}$
- Scattered Set, $\mathbf{sc}(G)$: max, $\phi(G,S)=$ $\boxed{\forall x \in V(G)\ |N[x] \cap S| \leq 1}$

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

$\mathbf{vp}$: vertex planarizer number of G $\quad (= \min\{|S| \mid G \setminus S \text{ is planar}\})$

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

$\mathbf{vp}$: vertex planarizer number of G $\quad (= \min\{|S| \mid G \setminus S \text{ is planar}\})$

$\mathbf{tw}$: the tree-width of G

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

$\mathbf{vp}$: vertex planarizer number of G $\quad (= \min\{|S| \mid G \setminus S \text{ is planar}\})$

$\mathbf{tw}$: the tree-width of G

$\mathbf{bw}$: the branch-width of G

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

$\mathbf{vp}$: vertex planarizer number of G $\quad (= \min\{|S| \mid G \setminus S \text{ is planar}\})$

$\mathbf{tw}$: the tree-width of G

$\mathbf{bw}$: the branch-width of G

$\mathbf{eg}$: the Euler genus of G

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

$\mathbf{vp}$: vertex planarizer number of G $\quad (= \min\{|S| \mid G \setminus S \text{ is planar}\})$

$\mathbf{tw}$: the tree-width of G

$\mathbf{bw}$: the branch-width of G

$\mathbf{eg}$: the Euler genus of G

r-$\mathbf{twm}$: $\quad (= \min\{|S| \mid \mathbf{tw}(G \setminus S\}) \leq r)$

We say that $\mathbf{p}$ is *minor closed* if $H \leq_m G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of minor-closed graph parameters:

$\mathbf{vc}$: minimum vertex cover of G

$\mathbf{fvs}$: minimum feedback vertex set of G

$\mathbf{fc}$: minimum face cover of a planar G

$\mathbf{lp}$: maximum k for which G contain a k-path

$\mathbf{vp}$: vertex planarizer number of G $\quad (= \min\{|S| \mid G \setminus S \text{ is planar}\})$

$\mathbf{tw}$: the tree-width of G

$\mathbf{bw}$: the branch-width of G

$\mathbf{eg}$: the Euler genus of G

r-$\mathbf{twm}$: $\quad (= \min\{|S| \mid \mathbf{tw}(G \setminus S\}) \leq r)$

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

$\mathbf{cd}$: minimum cycle domination set of G

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

$\mathbf{cd}$: minimum cycle domination set of G

$\mathbf{vc}$: minimum connected vertex cover of G

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

$\mathbf{cd}$: minimum cycle domination set of G

$\mathbf{vc}$: minimum connected vertex cover of G

$\mathbf{ctw}$: minimum connected treewidth of G i.e., all bags induce connected subgraphs

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

$\mathbf{cd}$: minimum cycle domination set of G

$\mathbf{vc}$: minimum connected vertex cover of G

$\mathbf{ctw}$: minimum connected treewidth of G i.e., all bags induce connected subgraphs

$\mathbf{icp}$: maximum induced cycle packing

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

$\mathbf{cd}$: minimum cycle domination set of G

$\mathbf{vc}$: minimum connected vertex cover of G

$\mathbf{ctw}$: minimum connected treewidth of G i.e., all bags induce connected subgraphs

$\mathbf{icp}$: maximum induced cycle packing

$\mathbf{sc}$: maximum k for which G contains a scattered set of k vertices.

We say that $\mathbf{p}$ is *contraction closed* if $H \leq_c G \Rightarrow \mathbf{p}(H) \leq \mathbf{p}(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:

$\mathbf{ds}$: minimum dominating set of G

$\mathbf{cd}$: minimum cycle domination set of G

$\mathbf{vc}$: minimum connected vertex cover of G

$\mathbf{ctw}$: minimum connected treewidth of G i.e., all bags induce connected subgraphs

$\mathbf{icp}$: maximum induced cycle packing

$\mathbf{sc}$: maximum k for which G contains a scattered set of k vertices.

[S is a scattered: no two distinct vertices of S have a common neighbor.]

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem $\boxed{p\text{-Checking Value of } \mathbf{p}} = (\mathcal{G}_{\text{all}}, \mathbf{p})$

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem $\boxed{p\text{-Checking Value of } \mathbf{p}} = (\mathcal{G}_{\text{all}}, \mathbf{p})$

In other words $\boxed{p\text{-Checking Value of } \mathbf{p}}$ corresponds to the following Meta-problem:

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem p-Checking Value of $\mathbf{p}$ $= (\mathcal{G}_{\text{all}}, \mathbf{p})$

In other words p-Checking Value of $\mathbf{p}$ corresponds to the following Meta-problem:

p-Checking Value of $\mathbf{p}$

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: is it correct that $\mathbf{p}(G) \leqslant k$?

Here "$\leqslant$" = "$\leq$" / "$\leqslant$" = "$\geq$" if $\mathbf{p}$ is a minimization/maximization parameter

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem p-Checking Value of $\mathbf{p} = (\mathcal{G}_{\text{all}}, \mathbf{p})$

In other words p-Checking Value of $\mathbf{p}$ corresponds to the following Meta-problem:

p-Checking Value of $\mathbf{p}$

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: is it correct that $\mathbf{p}(G) \lessgtr k$?

Here "$\lessgtr$" = "$\leq$" / "$\lessgtr$" = "$\geq$" if $\mathbf{p}$ is a minimization/maximization parameter

For simplicity: $\Pi_{\mathbf{p}}$ = p-Checking Value of $\mathbf{p}$

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem p-Checking Value of $\mathbf{p}$ $= (\mathcal{G}_{\text{all}}, \mathbf{p})$

In other words p-Checking Value of $\mathbf{p}$ corresponds to the following Meta-problem:

p-Checking Value of $\mathbf{p}$

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: is it correct that $\mathbf{p}(G) \lesseqgtr k$?

Here "$\lesseqgtr$" = "$\leq$" / "$\lesseqgtr$" = "$\geq$" if $\mathbf{p}$ is a minimization/maximization parameter

For simplicity: $\Pi_{\mathbf{p}}$ = p-Checking Value of $\mathbf{p}$

We call such a problem $\Pi_{\mathbf{p}}$ *subset optimization problem*

Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem p-Checking Value of $\mathbf{p}$ $= (\mathcal{G}_{\text{all}}, \mathbf{p})$

In other words p-Checking Value of $\mathbf{p}$ corresponds to the following Meta-problem:

p-Checking Value of $\mathbf{p}$

Instance: A graph G and an integer $k \geq 0$.

Parameter: k

Question: is it correct that $\mathbf{p}(G) \leqslant k$?

Here "$\leqslant$" = "$\leq$" / "$\leqslant$" = "$\geq$" if $\mathbf{p}$ is a minimization/maximization parameter

For simplicity: $\Pi_{\mathbf{p}}$ = p-Checking Value of $\mathbf{p}$

We call such a problem $\Pi_{\mathbf{p}}$ *subset optimization problem*

For every $k \in \mathbb{N}$, we define the k-th layer of $(\mathcal{G}_{\text{all}}, \mathbf{p})$ as the class

$\mathcal{G}_k^{\mathbf{p}} = \{G \mid \mathbf{p}(G) \leq k\}$

These are YES-instances for minimization problems and

NO-instances for maximization problems

- Observe: for every k, $\mathcal{G}_k^{\mathbf{P}}$ is minor- (contraction-) closed.

- Observe: for every k, $\mathcal{G}_k^{\mathbf{P}}$ is minor- (contraction-) closed.
- $\mathbf{obs}(\mathcal{G}_k^{\mathbf{P}})$ is the set of minor-minimal elements not in $\mathcal{G}$.

- Observe: for every k, $\mathcal{G}_k^{\mathbf{p}}$ is minor- (contraction-) closed.
- $\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathbf{p}$ is minor-closed, then so is $\mathcal{G}_k^{\mathbf{p}}$ and thus $G \in \mathcal{G}_k^{\mathbf{p}} \iff \forall H \in \mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})\ H \not\leq G$

- Observe: for every k, $\mathcal{G}_k^{\mathbf{p}}$ is minor- (contraction-) closed.
- $\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathbf{p}$ is minor-closed, then so is $\mathcal{G}_k^{\mathbf{p}}$ and thus $G \in \mathcal{G}_k^{\mathbf{p}} \iff \forall H \in \mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})\ H \not\leq G$

Consequence of R&S theorem: $\forall_{k\in\mathbb{N}}\ |\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})| < \aleph_0$

- Observe: for every k, $\mathcal{G}_k^{\mathbf{p}}$ is minor- (contraction-) closed.
- $\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathbf{p}$ is minor-closed, then so is $\mathcal{G}_k^{\mathbf{p}}$ and thus $G \in \mathcal{G}_k^{\mathbf{p}} \iff \forall H \in \mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})\ H \not\leq G$

Consequence of R&S theorem: $\forall_{k\in\mathbb{N}}\ |\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})| < \aleph_0$

Recall:

Theorem: [Robertson & Seymour – main algorithmic consequence of GM]

For every H, checking whether $H \leq G$ can be done in $f(|V(H)|) \cdot n^3$ steps.

- Observe: for every k, $\mathcal{G}_k^{\mathbf{p}}$ is minor- (contraction-) closed.
- $\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathbf{p}$ is minor-closed, then so is $\mathcal{G}_k^{\mathbf{p}}$ and thus $G \in \mathcal{G}_k^{\mathbf{p}} \iff \forall H \in \mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})\ H \not\leq G$

Consequence of R&S theorem: $\forall_{k\in\mathbb{N}}\ |\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})| < \aleph_0$

Recall:

Theorem: [Robertson & Seymour – main algorithmic consequence of GM]

For every H, checking whether $H \leq G$ can be done in $f(|V(H)|) \cdot n^3$ steps.

- Meta-Algorithmic Consequence: If $\mathbf{p}$ is minor-closed, then

p-Checking Value of $\mathbf{p}$ $\in$ FPT.

- Observe: for every k, $\mathcal{G}_k^{\mathbf{p}}$ is minor- (contraction-) closed.
- $\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})$ is the set of minor-minimal elements not in $\mathcal{G}$.
- If $\mathbf{p}$ is minor-closed, then so is $\mathcal{G}_k^{\mathbf{p}}$ and thus $G \in \mathcal{G}_k^{\mathbf{p}} \iff \forall H \in \mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})\ H \not\leq G$

Consequence of R&S theorem: $\forall_{k\in\mathbb{N}}\ |\mathbf{obs}(\mathcal{G}_k^{\mathbf{p}})| < \aleph_0$

Recall:

Theorem: [Robertson & Seymour – main algorithmic consequence of GM]

For every H, checking whether $H \leq G$ can be done in $f(|V(H)|) \cdot n^3$ steps.

- Meta-Algorithmic Consequence: If $\mathbf{p}$ is minor-closed, then

p-Checking Value of $\mathbf{p}$ $\in$ FPT.

In other words:

There exists an algorithm that solves p-Checking Value of $\mathbf{p}$ in $f(k) \cdot n^3$ steps.

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}}$ = p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} =$ p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} =$ p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} = p$-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\boxed{\Pi_{\mathbf{p}}} = \boxed{p\text{-Checking Value of } \mathbf{p}}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)
- This does not mean that we have constructed such an algorithm
- But...

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} = p$-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)
- This does not mean that we have constructed such an algorithm
- But... We are encouraged to do so!

"**Half of science is asking the right questions.**" **Roger Bacon**

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} =$ p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)
- This does not mean that we have constructed such an algorithm
- But... We are encouraged to do so!

"**Half of science is asking the right questions.**" **Roger Bacon**

Questions:

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} =$ p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)
- This does not mean that we have constructed such an algorithm
- But... We are encouraged to do so!

"**Half of science is asking the right questions.**" **Roger Bacon**

Questions:

- What is the best (constructive) f we can have and when?

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} =$ p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)
- This does not mean that we have constructed such an algorithm
- But... We are encouraged to do so!

"**Half of science is asking the right questions.**" **Roger Bacon**

Questions:

- What is the best (constructive) f we can have and when?
- Can the (many) ideas from Graph Minors be used for this?

Corollary: If $\mathbf{p}$ is a minor-closed graph parameter, then

$\Pi_{\mathbf{p}} =$ p-Checking Value of $\mathbf{p}$ can be solved in $f(k) \cdot n^3$ steps

- We have a (non-constructive) proof that an algorithm **exists!**
- Is an **encouraging** theory (**if** you know that something **exists**...)
- This does not mean that we have constructed such an algorithm
- But... We are encouraged to do so!

"**Half of science is asking the right questions.**" **Roger Bacon**

Questions:

- What is the best (constructive) f we can have and when?
- Can the (many) ideas from Graph Minors be used for this?
- Can we derive results for problems closed under other relations?

Comments on P versus NP: from CS and Maths

Comments on P versus NP: from CS and Maths

NP-completeness: A Retrospective

Christos H. Papadimitriou*

University of California, Berkeley, USA

Abstract. For a quarter of a century now, NP-completeness has been computer science's favorite paradigm, fad, punching bag, buzzword, alibi, and intellectual export. This paper is a fragmentary commentary on its origins, its nature, its impact, and on the attributes that have made it so pervasive and contagious.

"... NP-completeness in this context suggests that a problem area or approach is mathematically nasty.", Christos Papadimitriou – ICALP 1997

Comments on P versus NP: from CS and Maths

NP-completeness: A Retrospective

Christos H. Papadimitriou*

University of California, Berkeley, USA

Abstract. For a quarter of a century now, NP-completeness has been computer science's favorite paradigm, fad, punching bag, buzzword, alibi, and intellectual export. This paper is a fragmentary commentary on its origins, its nature, its impact, and on the attributes that have made it so pervasive and contagious.

"... NP-completeness in this context suggests that a problem area or approach is mathematically nasty.", Christos Papadimitriou – ICALP 1997

Mathematical Problems for the Next Century[1]

Steve Smale

Department of Mathematics
City University of Hong Kong
Kowloon, Hong Kong

August 7, 1998

"P versus NP – a gift to mathematics from computer science." Steve Smale, 1998

A metaphore on graph minors

A metaphore on graph minors

"Graph Minors suggests that some problems are algorithmically not so nasty"

A metaphore on graph minors

"Graph Minors suggests that some problems are algorithmically not so nasty"

"Graph Minors – a gift to computer science from mathematics"

General requirements

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

General requirements

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $2^{g(\mathbf{tw}(G))} \cdot n^{O(1)}$ steps

Typically this is done by Dynamic Programming.

General requirements

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $2^{g(\mathbf{tw}(G))} \cdot n^{O(1)}$ steps

Typically this is done by Dynamic Programming.

1+2 $\rightarrow$ $\boxed{\Pi_{\mathbf{p}}}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

General requirements

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $2^{g(\mathbf{tw}(G))} \cdot n^{O(1)}$ steps

Typically this is done by Dynamic Programming.

1+2 $\rightarrow$ $\boxed{\Pi_{\mathbf{p}}}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

---

Proof:

This algorithm first checks whether $\mathbf{tw}(G) \leq f(k)$.

If the answer is negative, then outputs a negative/positive answer (by **1**).

If the answer is positive, then runs DP algorithm (by **2**).

1+**2** $\rightarrow$ $\boxed{\Pi_{\mathbf{p}}}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

- $g(\cdot)$ is linear: $\boxed{\Pi_{\mathbf{p}}}$ is *singly exponentially solvable w.r.t. treewidth*

1+**2** $\rightarrow$ $\boxed{\Pi_{\mathbf{p}}}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

- $g(\cdot)$ is linear: $\boxed{\Pi_{\mathbf{p}}}$ is *singly exponentially solvable w.r.t. treewidth*

1+2 $\rightarrow$ $\boxed{\Pi_{\mathbf{p}}}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

- $g(\cdot)$ is linear: $\boxed{\Pi_{\mathbf{p}}}$ is *singly exponentially solvable w.r.t. treewidth*

[Liming Cai and David Juedes, 2003]:

For several problems, assuming ETH, the best running time we can expect is $2^{O(k)} \cdot n^{O(1)}$, in general

$\boxed{2^{O(\sqrt{k})} \cdot n^{O(1)}}$ for their planar restrictions

1+2 $\rightarrow$ $\boxed{\Pi_{\mathbf{p}}}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

- $g(\cdot)$ is linear: $\boxed{\Pi_{\mathbf{p}}}$ is *singly exponentially solvable w.r.t. treewidth*

[Liming Cai and David Juedes, 2003]:

For several problems, assuming ETH, the best running time we can expect is $2^{O(k)} \cdot n^{O(1)}$, in general

$\boxed{2^{O(\sqrt{k})} \cdot n^{O(1)}}$ for their planar restrictions $\longrightarrow$ **when** can we match this?

- Here we care about such questions!

Exponential Time Hypothesis (ETH):

There is no $2^{o(n)}$-step algorithm that solves 3-SAT

We care about combinatorial condition:

1. [Combinatorial] YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have **tw** $= O(k)$ (or $o(k)$ in planar graphs)

We care about combinatorial condition:

1. [Combinatorial] YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have **tw** $= O(k)$ (or $o(k)$ in planar graphs)

Idea: For minor-closed problems, the existence of a $(k \times k)$-grid as a minor of the input graph may serve as a YES/NO certificate

We care about combinatorial condition:

1. [Combinatorial] YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have **tw** $= O(k)$ (or $o(k)$ in planar graphs)

Idea: For minor-closed problems, the existence of a $(k \times k)$-grid as a minor of the input graph may serve as a YES/NO certificate

Fact: "Big" treewidth implies the the existence of a $(k \times k)$-grid as a minor

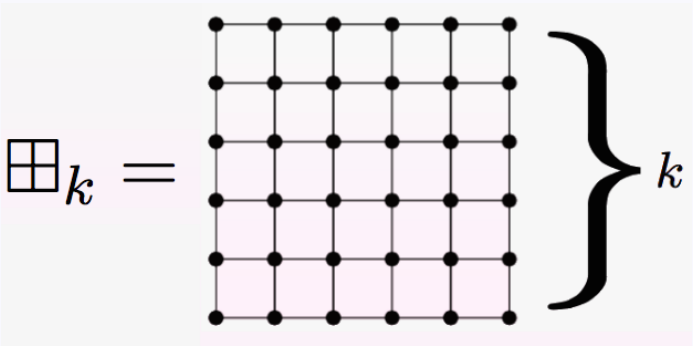
We care about combinatorial condition:

1. [Combinatorial] YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have **tw** $= O(k)$ (or $o(k)$ in planar graphs)

Idea: For minor-closed problems, the existence of a $(k \times k)$-grid as a minor of the input graph may serve as a YES/NO certificate

Fact: "Big" treewidth implies the the existence of a $(k \times k)$-grid as a minor

The $(k \times k)$-grid



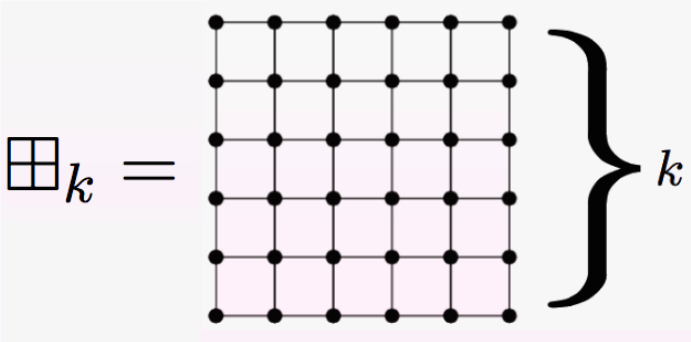
We care about combinatorial condition:

1. [Combinatorial] YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have **tw** $= O(k)$ (or $o(k)$ in planar graphs)

Idea: For minor-closed problems, the existence of a $(k \times k)$-grid as a minor of the input graph may serve as a YES/NO certificate

Fact: "Big" treewidth implies the the existence of a $(k \times k)$-grid as a minor

The $(k \times k)$-grid



This fact was first proved in GM-V by Robertson and Seymour.

Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \to \mathbb{N}$ *such that* $\forall \alpha \quad \mathbf{tw}(G) > \delta(\alpha) \Rightarrow G \geq_m \boxplus_\alpha$.

Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \to \mathbb{N}$ *such that* $\forall \alpha \quad \mathbf{tw}(G) > \delta(\alpha) \Rightarrow G \geq_m \boxplus_\alpha$.

Upper bound for δ: remained **exponential** for a long time...

Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \to \mathbb{N}$ *such that* $\forall \alpha \quad \mathbf{tw}(G) > \delta(\alpha) \Rightarrow G \geq_m \boxplus_\alpha$.

Upper bound for δ: remained **exponential** for a long time...

Untill:

Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \to \mathbb{N}$ *such that* $\forall \alpha \quad \mathbf{tw}(G) > \delta(\alpha) \Rightarrow G \geq_m \boxplus_\alpha$.

Upper bound for δ: remained **exponential** for a long time...

Untill: Julia Chuzhoy proved: $\delta(k) = O(k^{20})$

Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \to \mathbb{N}$ *such that* $\forall \alpha \quad \mathbf{tw}(G) > \delta(\alpha) \Rightarrow G \geq_m \boxplus_\alpha$.

Upper bound for δ: remained **exponential** for a **long** time...

Untill: Julia Chuzhoy proved: $\delta(k) = O(k^{20})$

- The best known lower bound is $\delta(k) = \Omega(k^2 \cdot \log k)$

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Observe the following:

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Observe the following:

Lemma: *If* $\mathbf{p}$ *is minor-closed, then*

1. [*Combinatorial*] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\mathbf{p}^{-1}(k))$

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Observe the following:

Lemma: *If* $\mathbf{p}$ *is minor-closed, then*

1. [*Combinatorial*] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\mathbf{p}^{-1}(k))$

Proof: Let $\alpha = \mathbf{p}^{-1}(k)$. Then $k < \mathbf{p}(\boxplus_\alpha)$ **(1)**.

Assume $\delta(\alpha) < \mathbf{tw}(G)$ **(2).**

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Observe the following:

Lemma: *If* $\mathbf{p}$ *is minor-closed, then*

1. [*Combinatorial*] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\mathbf{p}^{-1}(k))$

Proof: Let $\alpha = \mathbf{p}^{-1}(k)$. Then $k < \mathbf{p}(\boxplus_\alpha)$ **(1)**.

Assume $\delta(\alpha) < \mathbf{tw}(G)$ **(2).**

[By grid exclusion]: **(2)** $\Rightarrow \boxplus_\alpha \leq_m G$. **(1)**

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Observe the following:

Lemma: *If* $\mathbf{p}$ *is minor-closed, then*

1. [*Combinatorial*] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\mathbf{p}^{-1}(k))$

Proof: Let $\alpha = \mathbf{p}^{-1}(k)$. Then $k < \mathbf{p}(\boxplus_\alpha)$ **(1)**.

Assume $\delta(\alpha) < \mathbf{tw}(G)$ **(2).**

[By grid exclusion]: **(2)** $\Rightarrow \boxplus_\alpha \leq_m G$. **(1)**

[By minor-closedness]: if $\boxplus_\alpha \leq_m G$, then $\mathbf{p}(\boxplus_\alpha) \leq \mathbf{p}(G)$. **(3)**

Definition: $\mathbf{p}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\boxplus_\alpha) > k)\}$

Observe the following:

Lemma: *If* $\mathbf{p}$ *is minor-closed, then*

1. [*Combinatorial*] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\mathbf{p}^{-1}(k))$

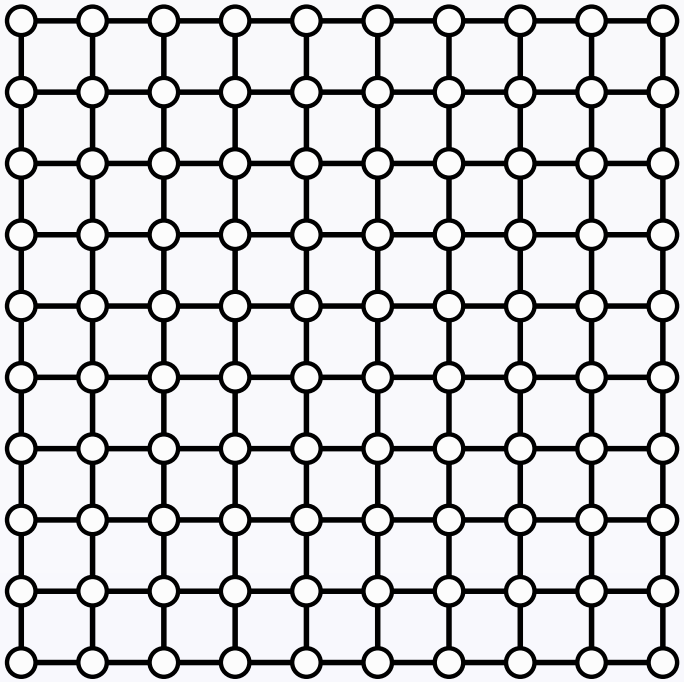
Proof: Let $\alpha = \mathbf{p}^{-1}(k)$. Then $k < \mathbf{p}(\boxplus_\alpha)$ **(1)**.

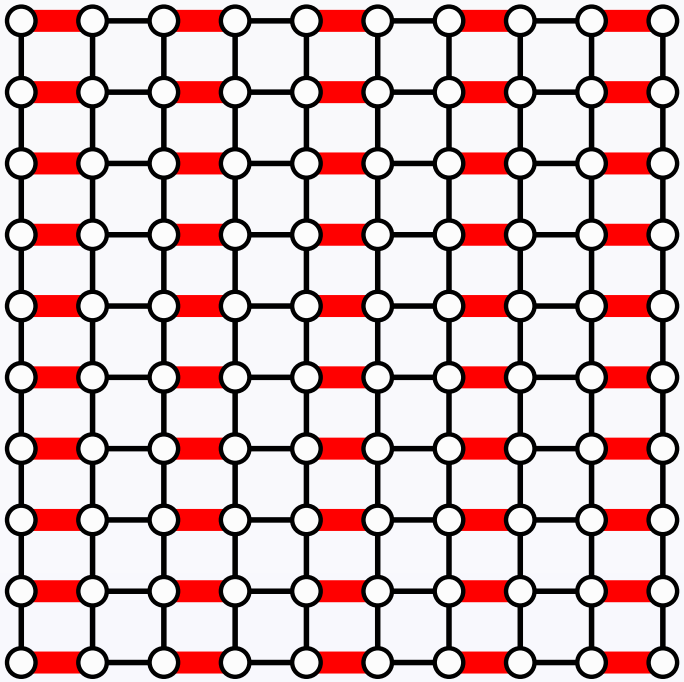
Assume $\delta(\alpha) < \mathbf{tw}(G)$ **(2)**.

[By grid exclusion]: **(2)** $\Rightarrow \boxplus_\alpha \leq_m G$. **(1)**

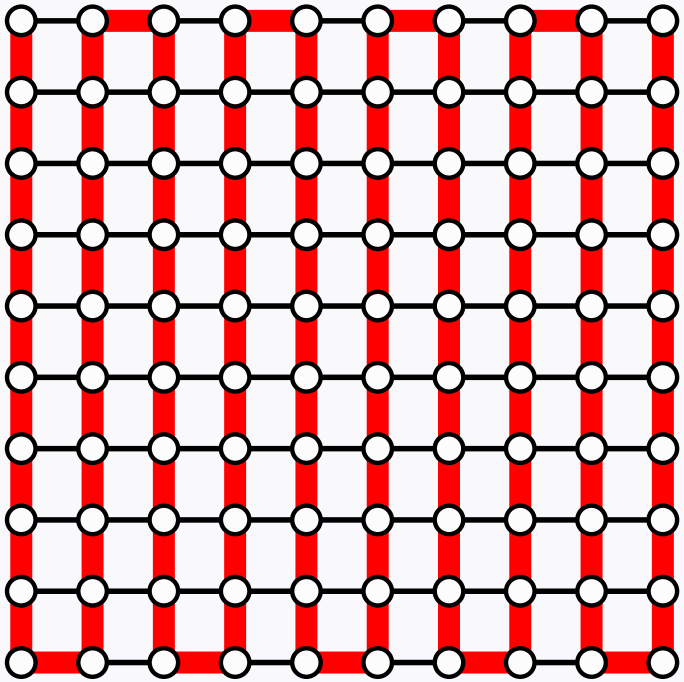
[By minor-closedness]: if $\boxplus_\alpha \leq_m G$, then $\mathbf{p}(\boxplus_\alpha) \leq \mathbf{p}(G)$. **(3)**

(1) and **(3)** $\Rightarrow k < \mathbf{p}(G)$.





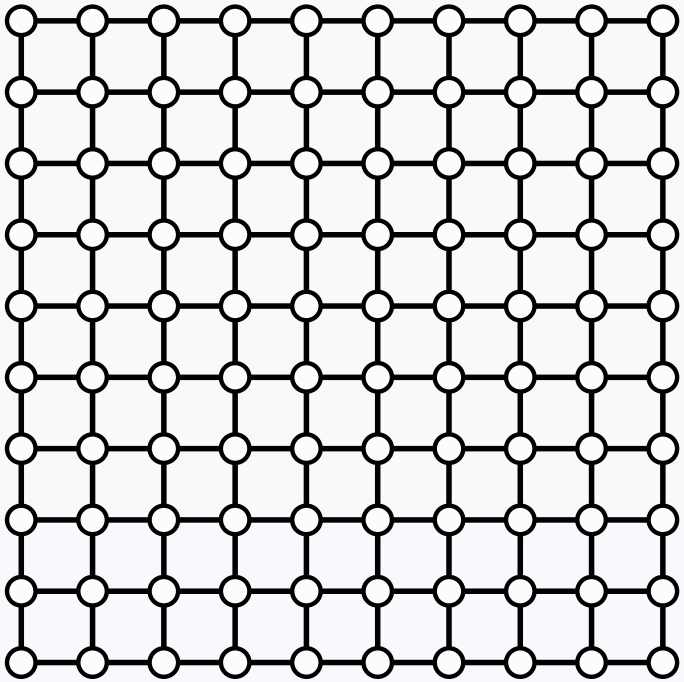
$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$



$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

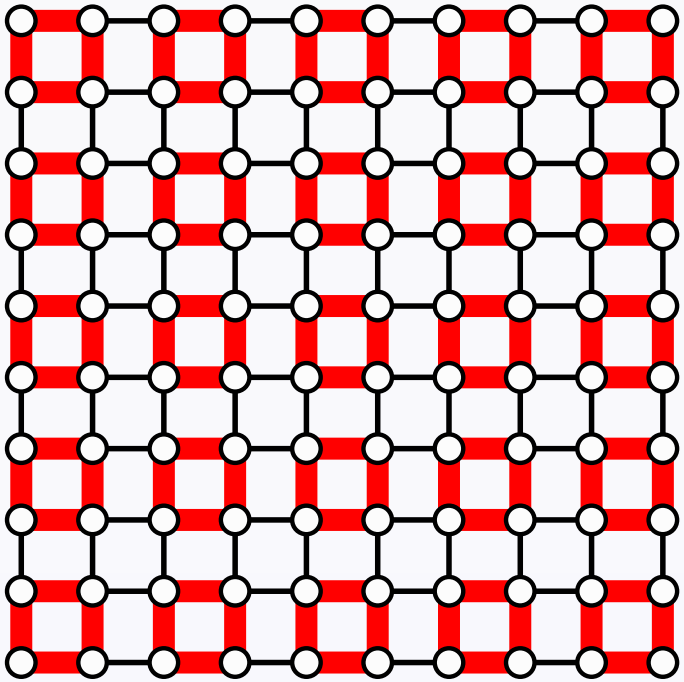
$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:



$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

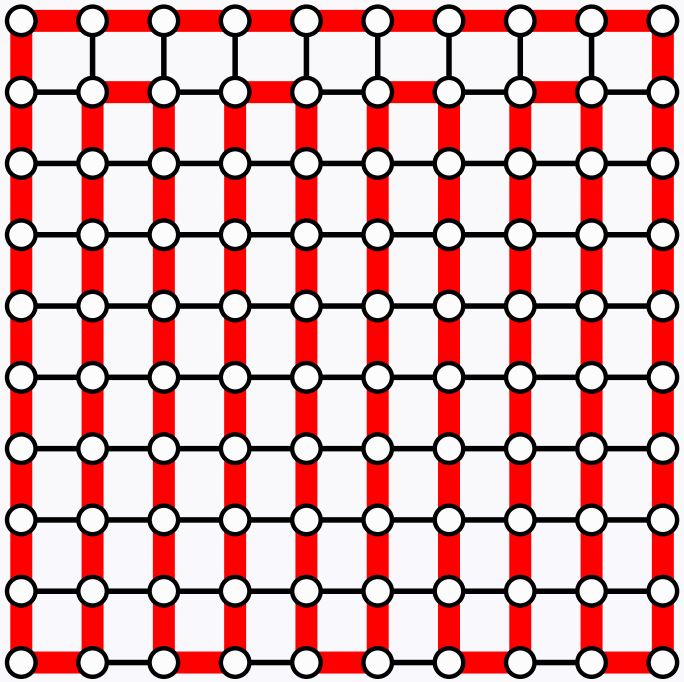


$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,

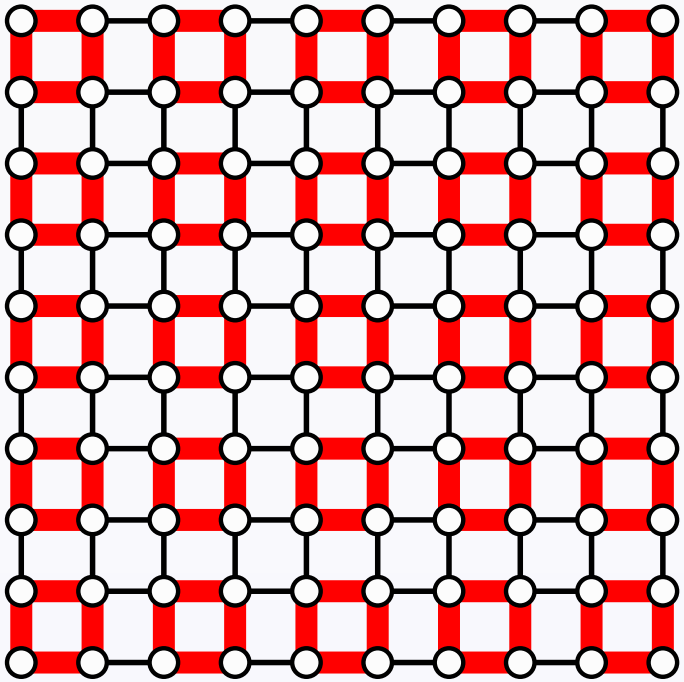


$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,

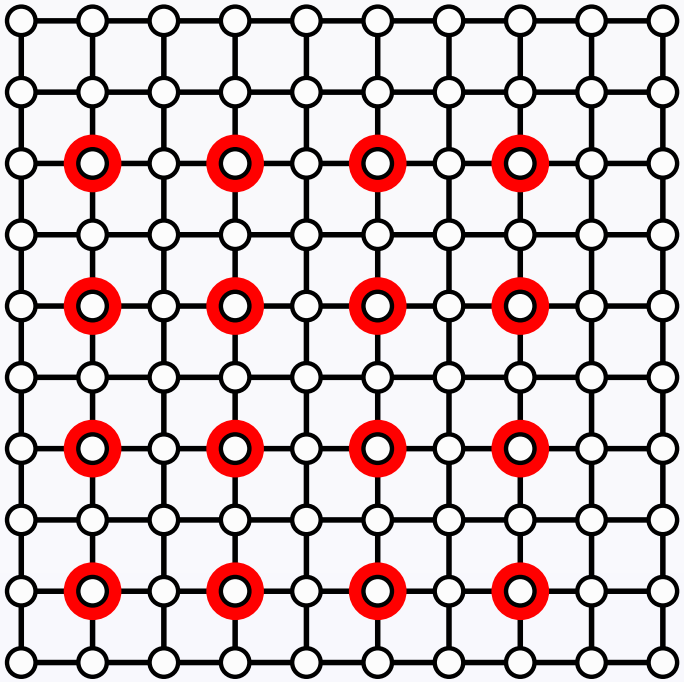


$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,

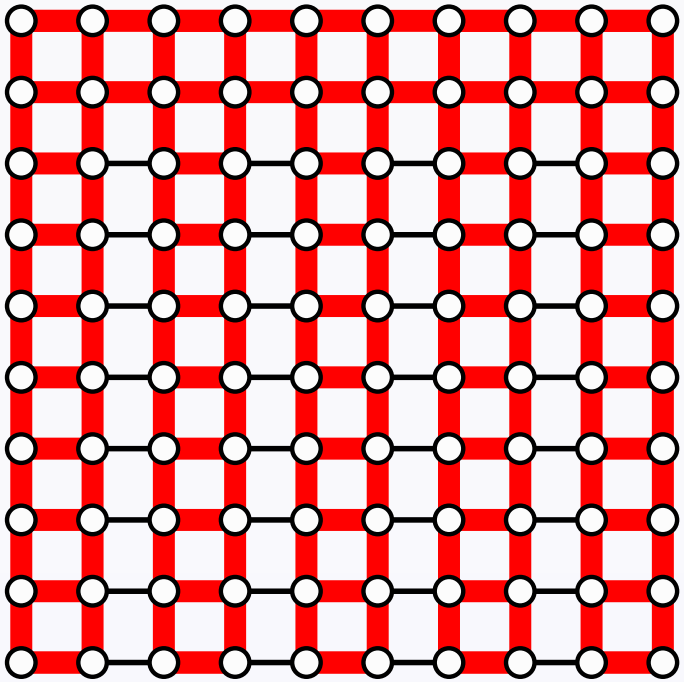


$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,
- Face Cover

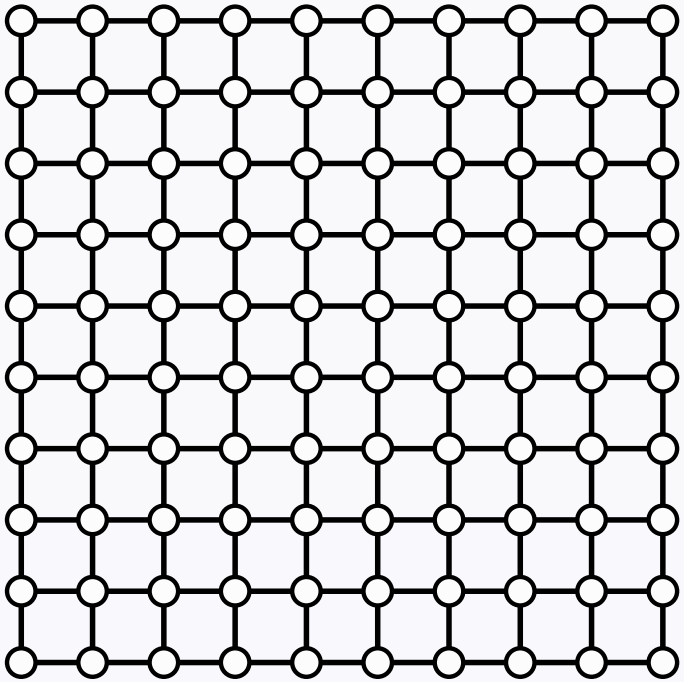


$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,
- Face Cover
- Max Series-Parallel Subgraph



$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

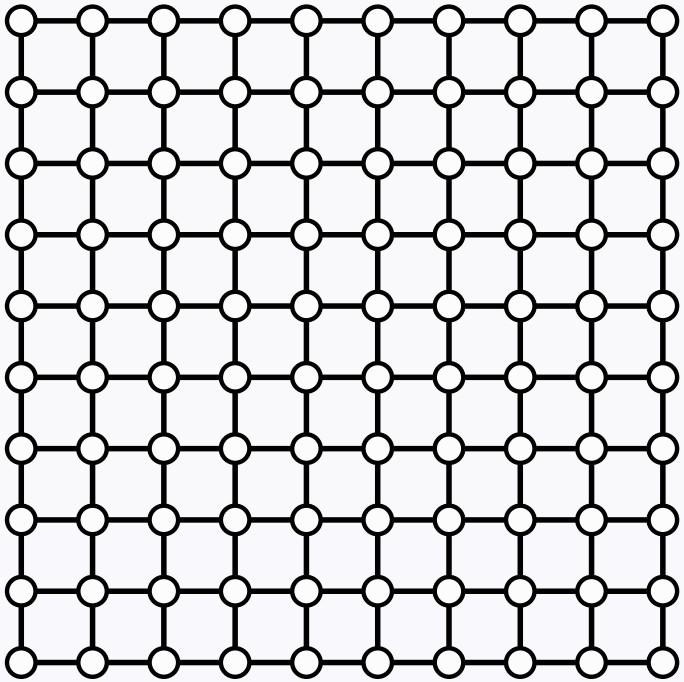
$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,
- Face Cover
- Max Series-Parallel Subgraph

Definition:

We call a problem $\Pi_{\mathbf{p}}$ minor-bidimensional if $\mathbf{p}$ is minor-closed and $\mathbf{p}^{-1}(k) = \sqrt{k}$



$\mathbf{vc}^{-1}(k) = O(\sqrt{k})$

$\mathbf{lp}^{-1}(k) = O(\sqrt{k})$

The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,
- Face Cover
- Max Series-Parallel Subgraph

Definition:

We call a problem $\Pi_{\mathbf{p}}$ minor-bidimensional if $\mathbf{p}$ is minor-closed and $\mathbf{p}^{-1}(k) = \sqrt{k}$

- Not all minor-closed problems are bidimensional!

such as Treewidth, Pathwidth, Branchwidth, Tree-depth, and Genus

For minor-bidimensional parameters:

For minor-bidimensional parameters:

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\sqrt{k})$

For minor-bidimensional parameters:

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

For minor-bidimensional parameters:

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all senarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms

For minor-bidimensional parameters:

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all senarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms

"Best of all senarios" means that

For minor-bidimensional parameters:

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all senarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms

"Best of all senarios" means that

- $\delta(k) = O(k^2 \cdot \log k)$ (which is conjectured but not sure!) and

For minor-bidimensional parameters:

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all senarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms 

"Best of all senarios" means that

- $\delta(k) = O(k^2 \cdot \log k)$ (which is conjectured but not sure!) and
- $\boxed{\Pi_{\mathbf{p}}}$ is singly exponentially solvable w.r.t. treewidth

(which is the case for many problems)

- **Conclusion:**

1. To design (optimal) $2^{O(k)} \cdot n^{O(1)}$ step algorithms for general graphs one needs a problem-specific analysis.

- **Conclusion:**

1. To design (optimal) $2^{O(k)} \cdot n^{O(1)}$ step algorithms for general graphs one needs a problem-specific analysis.

2. proving that $\delta(k) = O(k^2 \cdot \log k)$ will have interesting algorithmic consequences.

- **Conclusion:**

1. To design (optimal) $2^{O(k)} \cdot n^{O(1)}$ step algorithms for general graphs one needs a problem-specific analysis.

2. proving that $\delta(k) = O(k^2 \cdot \log k)$ will have interesting algorithmic consequences.

3. If we want $2^{O(k)} \cdot n^{O(1)}$ step algorithms we must restrict G to special graph classes.

- **Conclusion:**

1. To design (optimal) $2^{O(k)} \cdot n^{O(1)}$ step algorithms for general graphs one needs a problem-specific analysis.

2. proving that $\delta(k) = O(k^2 \cdot \log k)$ will have interesting algorithmic consequences.

3. If we want $2^{O(k)} \cdot n^{O(1)}$ step algorithms we must restrict G to special graph classes.

In particular: topological graph classes (where δ is better bounded)

▶ **Conclusion:**

1. To design (optimal) $2^{O(k)} \cdot n^{O(1)}$ step algorithms for general graphs one needs a problem-specific analysis.

2. proving that $\delta(k) = O(k^2 \cdot \log k)$ will have interesting algorithmic consequences.

3. If we want $2^{O(k)} \cdot n^{O(1)}$ step algorithms we must restrict G to special graph classes.

In particular: topological graph classes (where δ is better bounded)

4. For even better (e.g. **subexponential**) parameterized dependency we must restrict our attention to special graph classes.

Subexponential parameterized algorithms

Subexponential parameterized algorithms

Definition: *A graph class* $\mathcal{G}$ *has the* subquadratic grid minor property *(***SQGM***) if there exist* $1 \leq c < 2$ *such that* $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Subexponential parameterized algorithms

Definition: *A graph class* $\mathcal{G}$ *has the* subquadratic grid minor property *(***SQGM***) if there exist* $1 \leq c < 2$ *such that* $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\boxed{\Pi_{\mathbf{p}}}$ have

$\mathbf{tw} = (\mathbf{p}^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

Subexponential parameterized algorithms

Definition: *A graph class* $\mathcal{G}$ *has the* subquadratic grid minor property *(***SQGM***) if there exist* $1 \leq c < 2$ *such that* $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\boxed{\Pi_{\mathbf{p}}}$ have

$\mathbf{tw} = (\mathbf{p}^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

- if Π is singly exponentially solvable w.r.t. treewidth, then

Π can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Subexponential parameterized algorithms

Definition: *A graph class $\mathcal{G}$ has the* subquadratic grid minor property *(**SQGM**) if there exist $1 \leq c < 2$ such that* $\forall k \; \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\boxed{\Pi_{\mathbf{p}}}$ have

$\mathbf{tw} = (\mathbf{p}^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

- if Π is singly exponentially solvable w.r.t. treewidth, then

Π can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Planar graphs have the **SQGM** property for $c = 1$

Subexponential parameterized algorithms

Definition: *A graph class $\mathcal{G}$ has the* subquadratic grid minor property *(**SQGM**) if there exist $1 \leq c < 2$ such that* $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\boxed{\Pi_{\mathbf{p}}}$ have

$\mathbf{tw} = (\mathbf{p}^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

- if Π is singly exponentially solvable w.r.t. treewidth, then

Π can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Planar graphs have the **SQGM** property for $c = 1$

As $\mathbf{tw}(G) = O(\mathbf{bw}(G))$, the above follows from the following:

Subexponential parameterized algorithms

Definition: *A graph class $\mathcal{G}$ has the* subquadratic grid minor property *(**SQGM**) if there exist $1 \leq c < 2$ such that* $\forall k \; \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\boxed{\Pi_{\mathbf{p}}}$ have

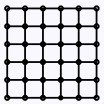
$\mathbf{tw} = (\mathbf{p}^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

- if Π is singly exponentially solvable w.r.t. treewidth, then

Π can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Planar graphs have the **SQGM** property for $c = 1$

As $\mathbf{tw}(G) = O(\mathbf{bw}(G))$, the above follows from the following:

Theorem: [Robertson, Seymour, & Thomas 1994] *If G is planar and $\mathbf{bw}(G) \geq 4k$, then*  $_k \leq_m G$.

Subexponential parameterized algorithms

Definition: *A graph class* $\mathcal{G}$ *has the* subquadratic grid minor property *(***SQGM***) if there exist* $1 \leq c < 2$ *such that* $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\boxed{\Pi_{\mathbf{p}}}$ have

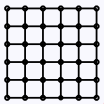
$\mathbf{tw} = (\mathbf{p}^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

- if Π is singly exponentially solvable w.r.t. treewidth, then

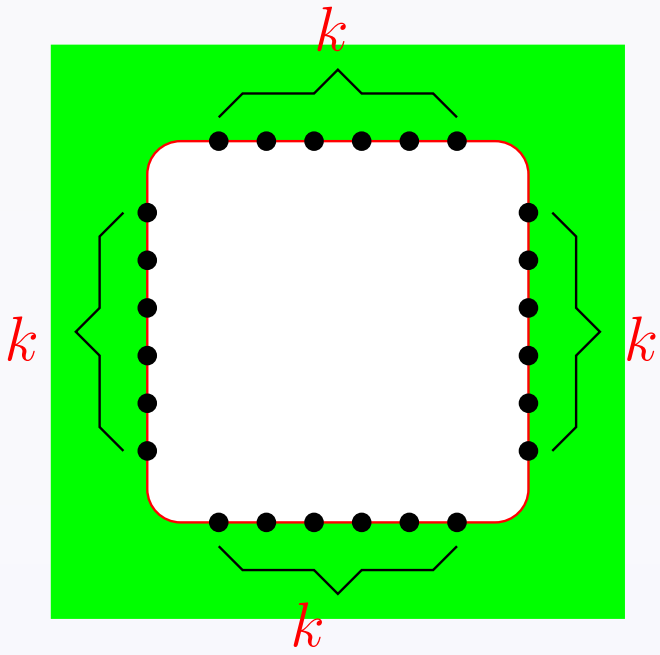
Π can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

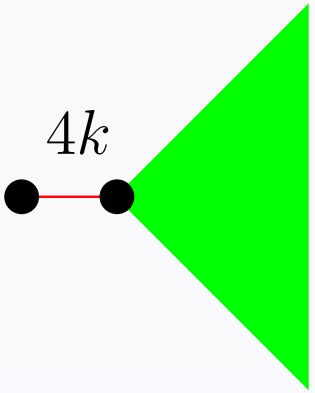
Planar graphs have the **SQGM** property for $c = 1$

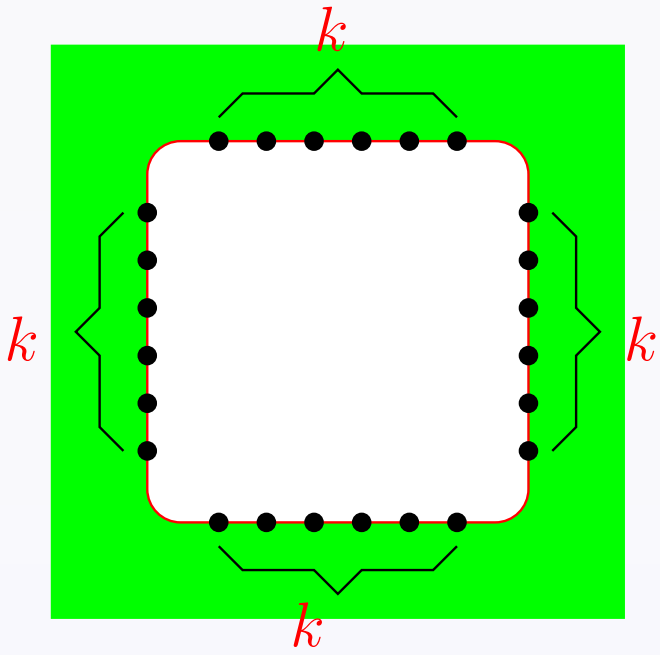
As $\mathbf{tw}(G) = O(\mathbf{bw}(G))$, the above follows from the following:

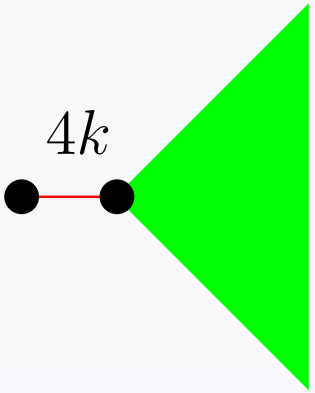
Theorem: [Robertson, Seymour, & Thomas 1994] *If* G *is planar and* $\mathbf{bw}(G) \geq 4k$, *then*  $_k \leq_m G$.

- We sketch the Idea of the proof of the above theorem:

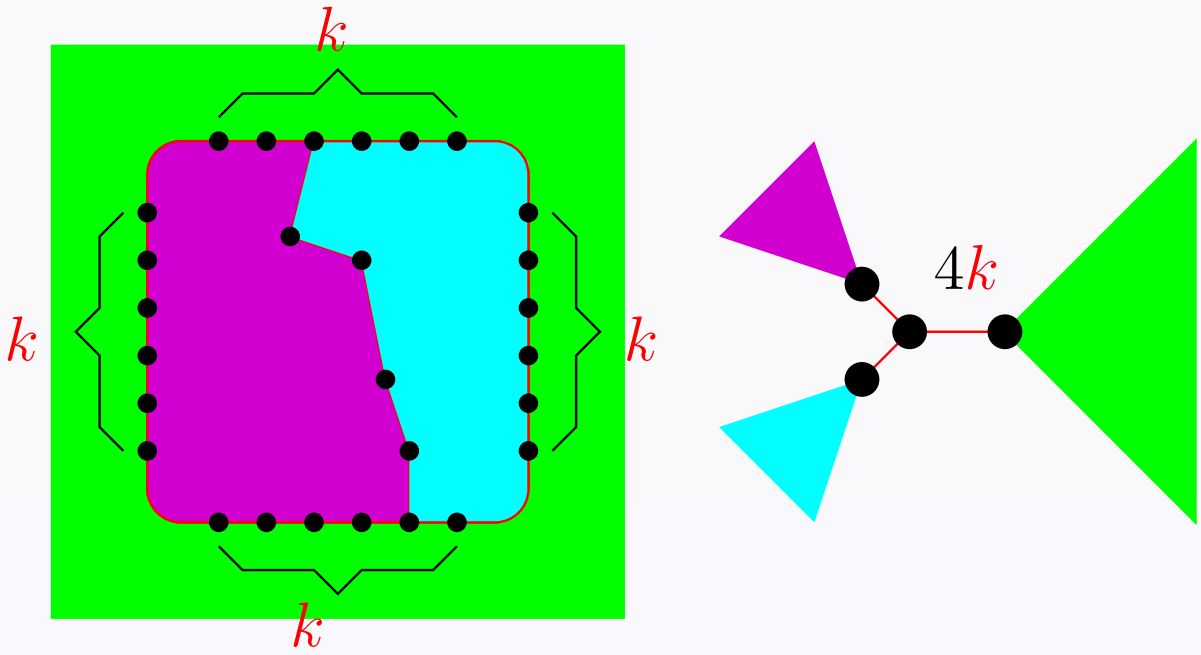

k
k
k
k


4k

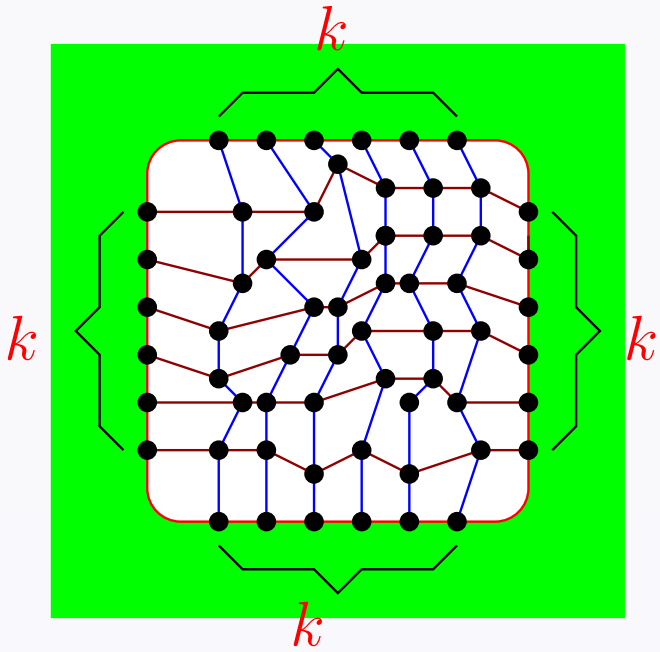


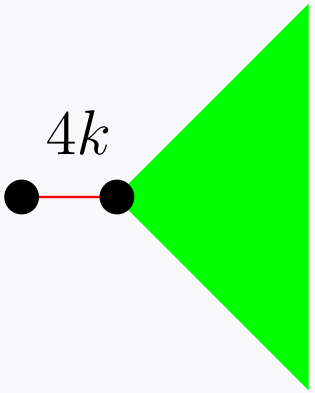


"Suppose" that we constructed a partial branch decomposition of the part of the graphs that is inside a disk.

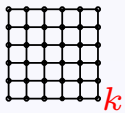


If there is a path from north-south or east-west, partition the disk: one more step further with the construction of a branch decomposition of width $\leq 4k$.





Such a path must exist,

otherwise, from Menger's theorem, the graph contains  as a minor.

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\preceq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\preceq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\preceq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Apex-minor free graphs: [Demaine, Fomin, Hajiaghayi, Thilikos, SIDMA 2004]

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Apex-minor free graphs: [Demaine, Fomin, Hajiaghayi, Thilikos, SIDMA 2004]

H-minor free graphs: [Demaine, Hajiaghayi, Combinatorica 2008]

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Apex-minor free graphs: [Demaine, Fomin, Hajiaghayi, Thilikos, SIDMA 2004]

H-minor free graphs: [Demaine, Hajiaghayi, Combinatorica 2008]

Bounded degree unit disk graphs [Fomin, Lokshtanov, Saurabh, SODA 2012]

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Apex-minor free graphs: [Demaine, Fomin, Hajiaghayi, Thilikos, SIDMA 2004]

H-minor free graphs: [Demaine, Hajiaghayi, Combinatorica 2008]

Bounded degree unit disk graphs [Fomin, Lokshtanov, Saurabh, SODA 2012]

Families of 2D-geometric graphs [Grigoriev, Koutsonas, Thilikos, SOFSEM 2014]

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Apex-minor free graphs: [Demaine, Fomin, Hajiaghayi, Thilikos, SIDMA 2004]

H-minor free graphs: [Demaine, Hajiaghayi, Combinatorica 2008]

Bounded degree unit disk graphs [Fomin, Lokshtanov, Saurabh, SODA 2012]

Families of 2D-geometric graphs [Grigoriev, Koutsonas, Thilikos, SOFSEM 2014]

- In all above cases we have topologically refined graph classes and $c = 1$.

Bidimensionality race:

SQGM: $\forall k \boxplus_k \not\leq G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGM** property holds?

Planar: [Robertson and Seymour, JCSTB 1986]

Bounded Genus: [Demaine, Fomin, Hajiaghayi, Thilikos, JACM 2005]

Apex-minor free graphs: [Demaine, Fomin, Hajiaghayi, Thilikos, SIDMA 2004]

H-minor free graphs: [Demaine, Hajiaghayi, Combinatorica 2008]

Bounded degree unit disk graphs [Fomin, Lokshtanov, Saurabh, SODA 2012]

Families of 2D-geometric graphs [Grigoriev, Koutsonas, Thilikos, SOFSEM 2014]

- In all above cases we have topologically refined graph classes and $c = 1$.

- are there more general graph classes where $1 < c < 2$?

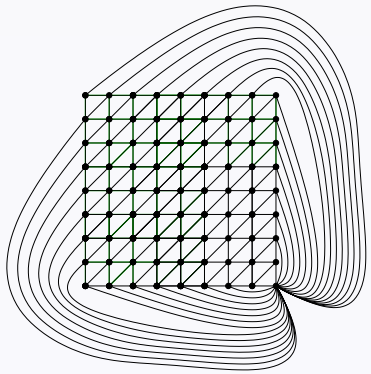
Bidimensionality for contraction-closed problems

[such as $\Pi_{\mathbf{ds}}$]

Bidimensionality for contraction-closed problems

[such as $\Pi_{\mathbf{ds}}$]

$\boxplus_k$ is replaced by the uniformly triangulated grid Γ_k:

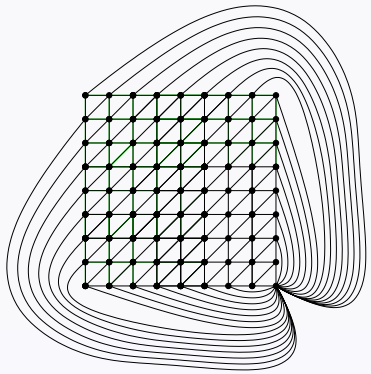


Let $\tilde{\mathbf{p}}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\Gamma_\alpha) > k)\}$

Bidimensionality for contraction-closed problems

[such as $\Pi_{\mathbf{ds}}$]

$\boxplus_k$ is replaced by the uniformly triangulated grid Γ_k:



Let $\tilde{\mathbf{p}}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\Gamma_\alpha) > k)\}$

Definition:

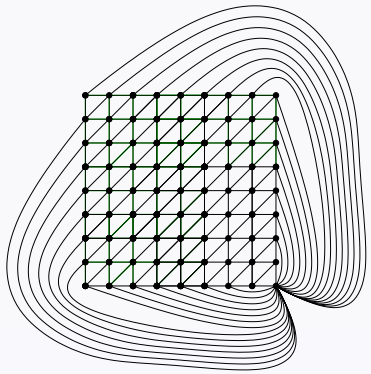
We call a problem $\Pi_{\mathbf{p}}$ contraction-bidimensional

if $\mathbf{p}$ is contraction-closed and $\tilde{\mathbf{p}}^{-1}(k) = \sqrt{k}$

Bidimensionality for contraction-closed problems

[such as $\Pi_{\mathbf{ds}}$]

$\boxplus_k$ is replaced by the uniformly triangulated grid Γ_k:



Let $\tilde{\mathbf{p}}^{-1}(k) = \min\{\alpha \mid \mathbf{p}(\Gamma_\alpha) > k)\}$

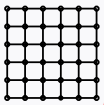
Definition:

We call a problem $\Pi_{\mathbf{p}}$ contraction-bidimensional

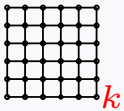
if $\mathbf{p}$ is contraction-closed and $\tilde{\mathbf{p}}^{-1}(k) = \sqrt{k}$

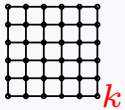
Definition: A class $\mathcal{G}$ has the *subquadratic grid contraction property* (**SQGC**) if there exist $1 \leq c < 2$ such that $\forall k\ \Gamma_k \not\preceq_c G \Rightarrow \mathbf{tw}(G) = O(k^c)$

Subquadratic grid contraction property (**SQGC**) holds for planar graphs because of:

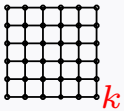
Theorem: [Robertson, Seymour, & Thomas 1994] *If* G *is planar and* $\mathbf{bw}(G) \geq 4k$, *then*  $_k \leq_m G$.

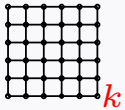
Subquadratic grid contraction property (**SQGC**) holds for planar graphs because of:

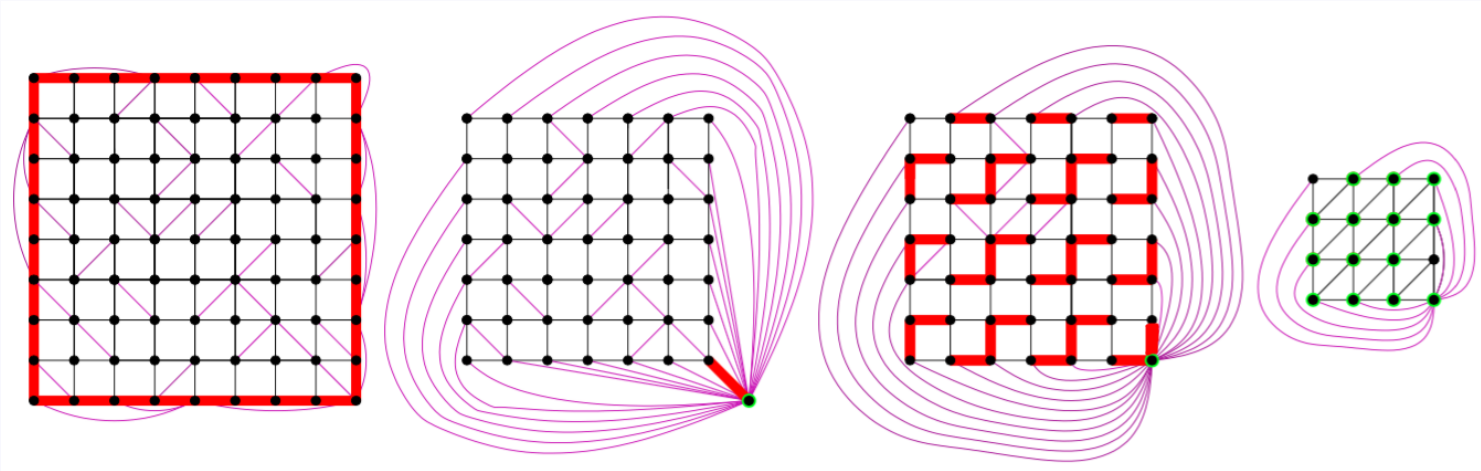
Theorem: [Robertson, Seymour, & Thomas 1994] *If* G *is planar and* $\mathbf{bw}(G) \geq 4k$, *then*  $_k \leq_m G$.

Proof: if we do not apply edge removals while obtaining  $_k$ from G we end up to a partially triangulated grid that can be further be contracted to the uniformly triangulated grid Γ_k.

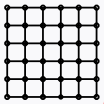
Subquadratic grid contraction property (**SQGC**) holds for planar graphs because of:

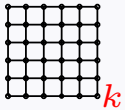
Theorem: [Robertson, Seymour, & Thomas 1994] *If* G *is planar and* $\mathbf{bw}(G) \geq 4k$, *then*  $_k \leq_m G$.

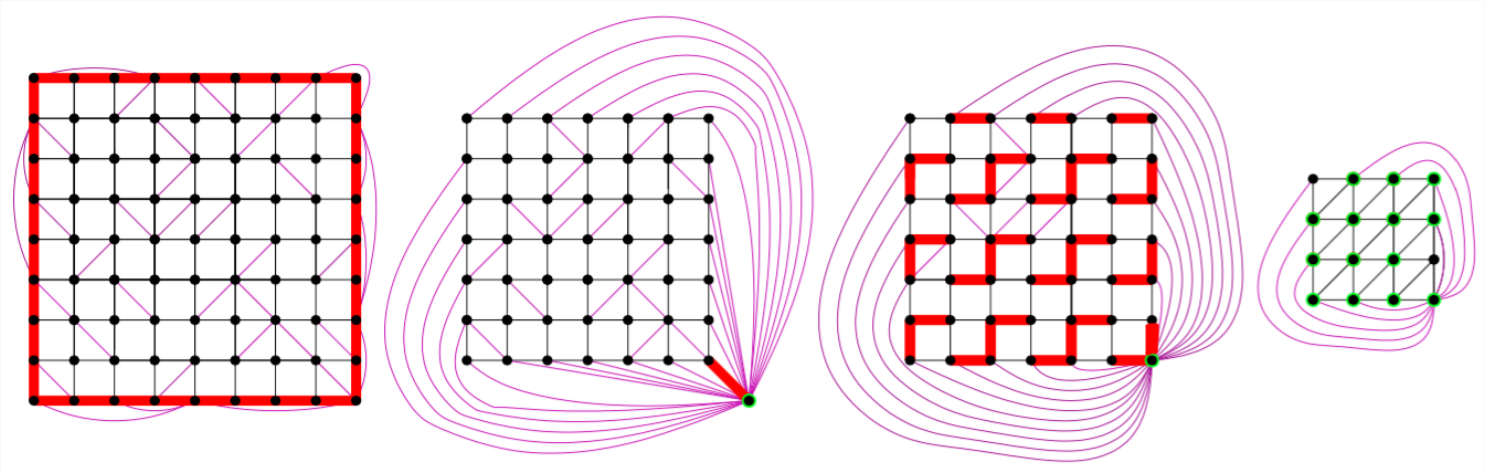
Proof: if we do not apply edge removals while obtaining  $_k$ from G we end up to a partially triangulated grid that can be further be contracted to the uniformly triangulated grid Γ_k.



Subquadratic grid contraction property (**SQGC**) holds for planar graphs because of:

Theorem: [Robertson, Seymour, & Thomas 1994] *If* G *is planar and* $\mathbf{bw}(G) \geq 4k$, *then*  $_k \leq_m G$.

Proof: if we do not apply edge removals while obtaining  $_k$ from G we end up to a partially triangulated grid that can be further be contracted to the uniformly triangulated grid Γ_k.



Therefore $\Gamma_k \not\leq_c G \Rightarrow \mathbf{tw}(G) = O(k^c)$, thus **SQGC** holds for $c = 1$.

Bidimensionality race:

SQGC: $\forall k\ \Gamma_k \not\leq_c G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGC** property holds?

Planar: follows from [Robertson and Seymour, JCSTB 1986]

Bounded Genus graphs: [Demaine, Hajiaghayi, Thilikos, SIDMA 2006]

Apex-minor free graphs: [Fomin, Golovach, Thilikos, JCTSB 2011]

Families of 2D-geometric graphs [Baste, Thilikos, in preparation]

Bidimensionality race:

SQGC: $\forall k\ \Gamma_k \not\leq_c G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGC** property holds?

Planar: follows from [Robertson and Seymour, JCSTB 1986]

Bounded Genus graphs: [Demaine, Hajiaghayi, Thilikos, SIDMA 2006]

Apex-minor free graphs: [Fomin, Golovach, Thilikos, JCTSB 2011]

Families of 2D-geometric graphs [Baste, Thilikos, in preparation]

Bidimensionality race:

SQGC: $\forall k\ \Gamma_k \not\leq_c G \Rightarrow \mathbf{tw}(G) = O(k^c)$ for some $c < 2$.

When **SQGC** property holds?

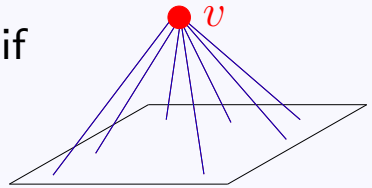
Planar: follows from [Robertson and Seymour, JCSTB 1986]

Bounded Genus graphs: [Demaine, Hajiaghayi, Thilikos, SIDMA 2006]

Apex-minor free graphs: [Fomin, Golovach, Thilikos, JCTSB 2011]

Families of 2D-geometric graphs [Baste, Thilikos, in preparation]

A graph H is an *apex graph* if



planar graph

$\exists v \in V(H)$: $H - v$ is planar

Bidimensionality and subexponential algorithms.

Theorem: *Let* $\boxed{\Pi_{\mathbf{p}}}$ *be a subset optimization parameterized problem that*

Bidimensionality and subexponential algorithms.

Theorem: *Let* $\boxed{\Pi_{\mathbf{p}}}$ *be a subset optimization parameterized problem that*

i. *is minor/contraction-bidimensional*

Bidimensionality and subexponential algorithms.

Theorem: *Let* $\boxed{\Pi_{\mathbf{p}}}$ *be a subset optimization parameterized problem that*

i. *is minor/contraction-bidimensional*

ii. *is singly exponentially solvable w.r.t. treewidth*

Bidimensionality and subexponential algorithms.

Theorem: *Let* $\boxed{\Pi_{\mathbf{p}}}$ *be a subset optimization parameterized problem that*

i. *is minor/contraction-bidimensional*

ii. *is singly exponentially solvable w.r.t. treewidth*

iii. *is restricted to some* **SQGM**/**SQGC**-*graph class*

Bidimensionality and subexponential algorithms.

Theorem: *Let* $\boxed{\Pi_{\mathbf{p}}}$ *be a subset optimization parameterized problem that*

i. *is minor/contraction-bidimensional*

ii. *is singly exponentially solvable w.r.t. treewidth*

iii. *is restricted to some* **SQGM**/**SQGC**-*graph class*

Then Π *can be solved in* $2^{o(k)} \cdot n^{O(1)}$ *steps*

Some bidimensional problems

(Connected) Vertex Cover, (Connected) Dominating Set, (Connected) Feedback Vertex Set, Induced Matching, Longest Cycle, (Connected) (Induced) Cycle Packing, (Connected) Cycle Domination, d-Scattered Set, Longest Path, (Induced) Path Packing, (Connected) r-Center, (Connectd) Diamond Hitting Set, Minimum Maximal Matching, Face Cover, Unweighted TSP Tour, Max Bounded Degree Connected Subgraph

- The previous theorem can become an algorithmic meta-theorem as

  ii. is singly exponentially solvable w.r.t. treewidth

is implied by expressibility in Existential Counting Modal Logic

because of [Michał Pilipczuk, MFCS 2011]

- The previous theorem can become an algorithmic meta-theorem as

  ii. is singly exponentially solvable w.r.t. treewidth

is implied by expressibility in Existential Counting Modal Logic

because of [Michał Pilipczuk, MFCS 2011]

- Some powerful techniques for ii.

[Dorn, Penninkx, Bodlaender, Fomin, Algorithmica 2010] $\star$

[Rué, Sau, Thilikos, TALG 2014] $\star$

- The previous theorem can become an algorithmic meta-theorem as

  ii. is singly exponentially solvable w.r.t. treewidth

is implied by expressibility in Existential Counting Modal Logic

because of [Michał Pilipczuk, MFCS 2011]

- Some powerful techniques for ii.

[Dorn, Penninkx, Bodlaender, Fomin, Algorithmica 2010] $\star$

[Rué, Sau, Thilikos, TALG 2014] $\star$

[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk, FOCS 2011] $\star\star$

[Bodlaender, Cygan, Kratsch, Nederlof, ICALP 2013] $\star\star$

[Fomin, Lokshtanov, Saurabh, SODA 2014] $\star\star$

Part 5, Friday 10/05/2016 - 09:00–10:30 (90')

Bidimensionality and Kernelization

Irrelevant vertex technique

Bidimensionality and Kernelization

Kernelization

Let (Π, κ) be a parameterized problem.

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

1. $x \in \Pi \Leftrightarrow x' \in \Pi$ (x and x' are equivalent)

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

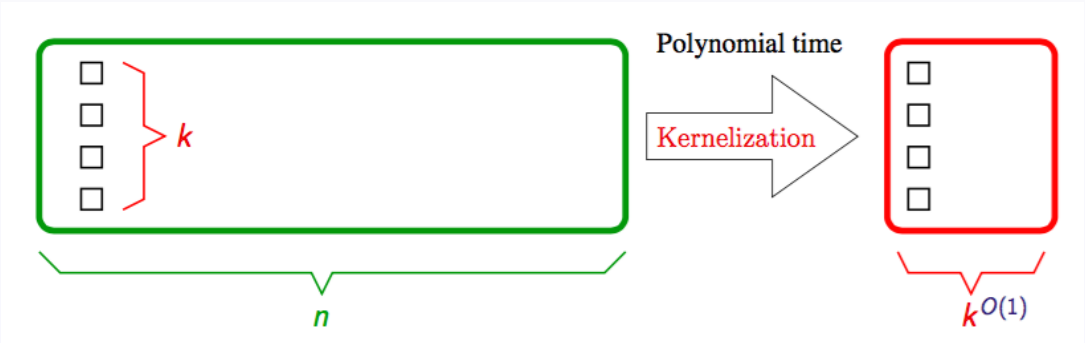
1. $x \in \Pi \Leftrightarrow x' \in \Pi$ (x and x' are equivalent)
2. $|x'| \leq g(k)$ (new instance has size bounded by a function of the parameter).

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

1. $x \in \Pi \Leftrightarrow x' \in \Pi$ (x and x' are equivalent)
2. $|x'| \leq g(k)$ (new instance has size bounded by a function of the parameter).

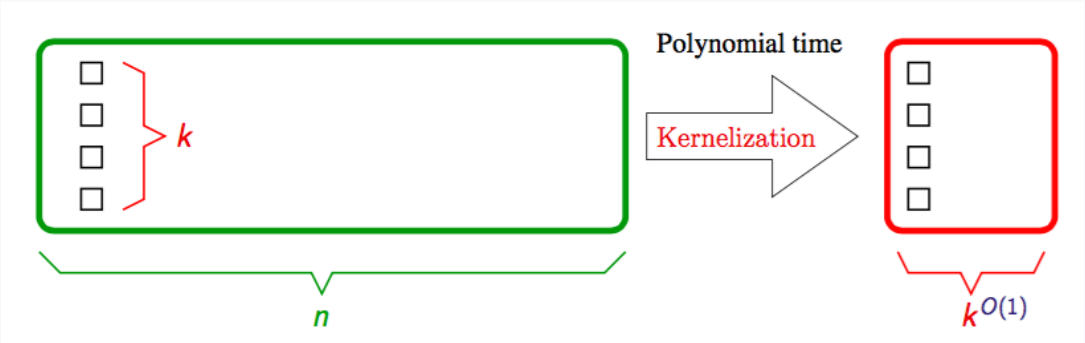


Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

1. $x \in \Pi \Leftrightarrow x' \in \Pi$ (x and x' are equivalent)
2. $|x'| \leq g(k)$ (new instance has size bounded by a function of the parameter).



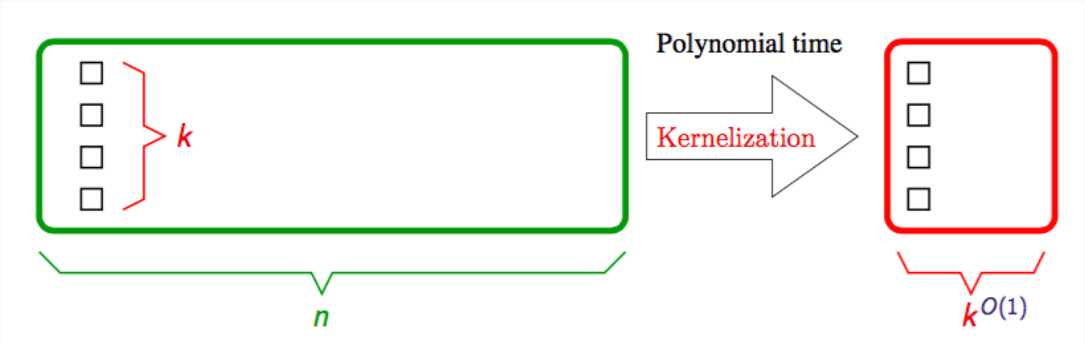
- If G is a polynomial (linear): polynomial (linear) kernel.

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

1. $x \in \Pi \Leftrightarrow x' \in \Pi$ (x and x' are equivalent)
2. $|x'| \leq g(k)$ (new instance has size bounded by a function of the parameter).



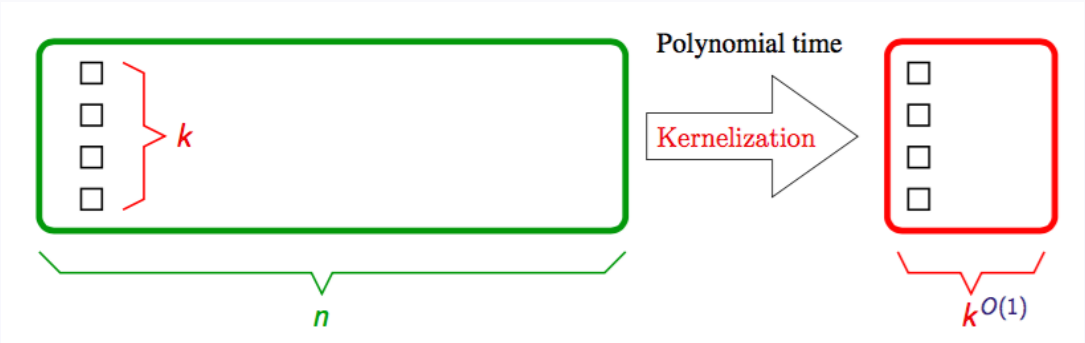
- If G is a polynomial (linear): polynomial (linear) kernel.
- a kernelization is a polynomial time many-one reduction of a problem to itself with the additional property that the image is bounded in terms of the parameter $k = \kappa(x)$.

Kernelization

Let (Π, κ) be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \to \mathbb{N}$.

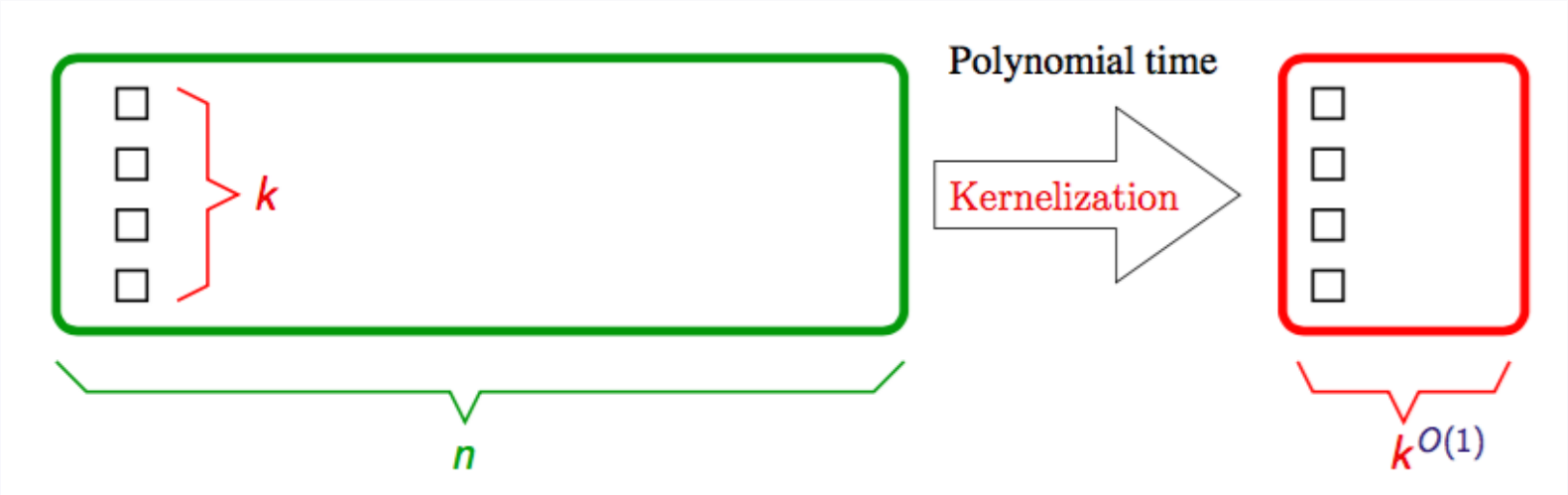
- A polynomial algorithm A is a *kernelization algorithm* for (Π, κ) if there exist some computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = \mathsf{A}(x)$ satisfies the folllowing:

1. $x \in \Pi \Leftrightarrow x' \in \Pi$ (x and x' are equivalent)
2. $|x'| \leq g(k)$ (new instance has size bounded by a function of the parameter).

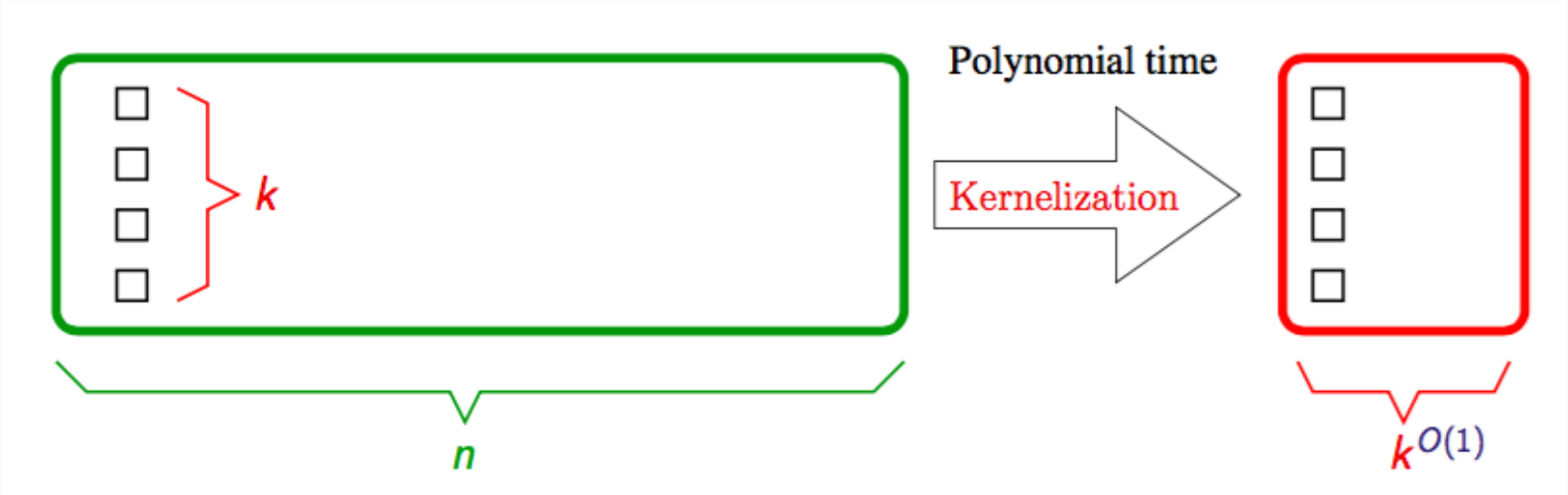


- If G is a polynomial (linear): polynomial (linear) kernel.
- a kernelization is a polynomial time many-one reduction of a problem to itself with the additional property that the image is bounded in terms of the parameter $k = \kappa(x)$.
- *Kernelization can be seen as a paradigm for* **preprocessing**

p-Vertex Cover has kernelization algorithm that produces a kernel of $\leq 2k$ vertices.



p-Vertex Cover has kernelization algorithm that produces a kernel of $\leq 2k$ vertices.



Alternatively, we say that p-Vertex Cover has a kernel of size $2k$.

- A parameterized problem has a kernel iff it is in FPT

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- Not all problems in FPT are expected to have polynomial kernels (p-Path)

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- Not all problems in FPT are expected to have polynomial kernels (p-Path)

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

- p-Feedback Vertex Set has a kernel of $O(k^2)$ edges.

[Thomassé, TALG 2010]

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- Not all problems in FPT are expected to have polynomial kernels (p-Path)

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

- p-Feedback Vertex Set has a kernel of $O(k^2)$ edges.

[Thomassé, TALG 2010]

- Not all FPT-problems are expected to have linear kernel (p-Feedback Vertex Set)

[Dell, van Melkebeek, STOC 2010]

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- Not all problems in FPT are expected to have polynomial kernels (p-Path)

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

- p-Feedback Vertex Set has a kernel of $O(k^2)$ edges.

[Thomassé, TALG 2010]

- Not all FPT-problems are expected to have linear kernel (p-Feedback Vertex Set)

[Dell, van Melkebeek, STOC 2010]

- p-Planar Dominating Set kernel of $67k$ vertices.

[Chen, Fernau, Kanj, Xia, SICOMB 2007]

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- Not all problems in FPT are expected to have polynomial kernels (p-Path)

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

- p-Feedback Vertex Set has a kernel of $O(k^2)$ edges.

[Thomassé, TALG 2010]

- Not all FPT-problems are expected to have linear kernel (p-Feedback Vertex Set)

[Dell, van Melkebeek, STOC 2010]

- p-Planar Dominating Set kernel of $67k$ vertices.

[Chen, Fernau, Kanj, Xia, SICOMB 2007]

- p-Planar Feedback Vertex Set has a kernel of $13k$ vertices.

[Bonamy, Kowalik, IPEC 2014]

- A parameterized problem has a kernel iff it is in FPT
- p-Dominating Set is W[2]-complete, this it is not expected to have a kernel.

- Not all problems in FPT are expected to have polynomial kernels (p-Path)

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

- p-Feedback Vertex Set has a kernel of $O(k^2)$ edges.

[Thomassé, TALG 2010]

- Not all FPT-problems are expected to have linear kernel (p-Feedback Vertex Set)

[Dell, van Melkebeek, STOC 2010]

- p-Planar Dominating Set kernel of $67k$ vertices.

[Chen, Fernau, Kanj, Xia, SICOMB 2007]

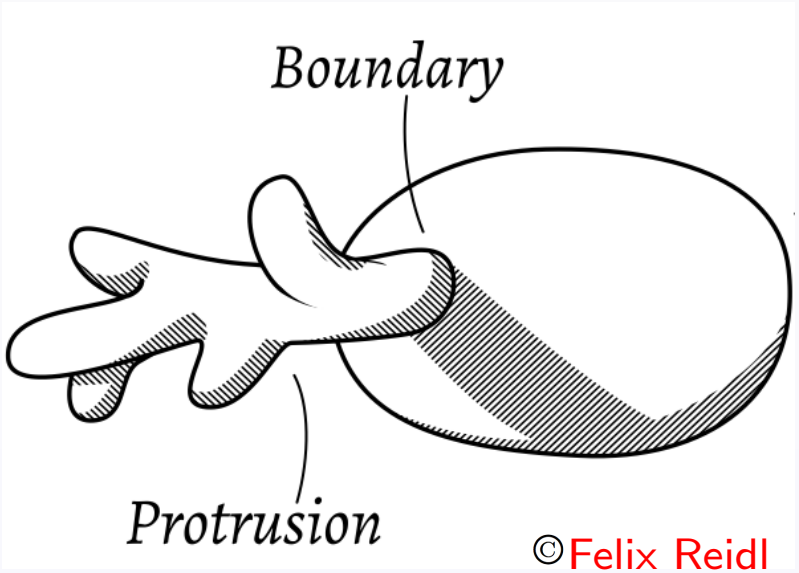
- p-Planar Feedback Vertex Set has a kernel of $13k$ vertices.

[Bonamy, Kowalik, IPEC 2014]

p-Path, p-Dominating Set, and p-Feedback Vertex Set are bidimensional.

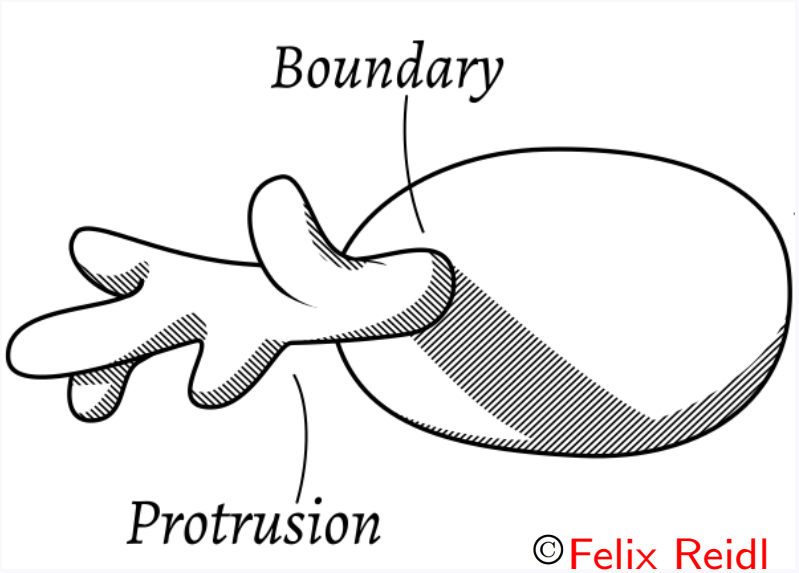
Protrusions

Protrusions



r-*protrusion*: a set $X \subseteq V(G)$ where

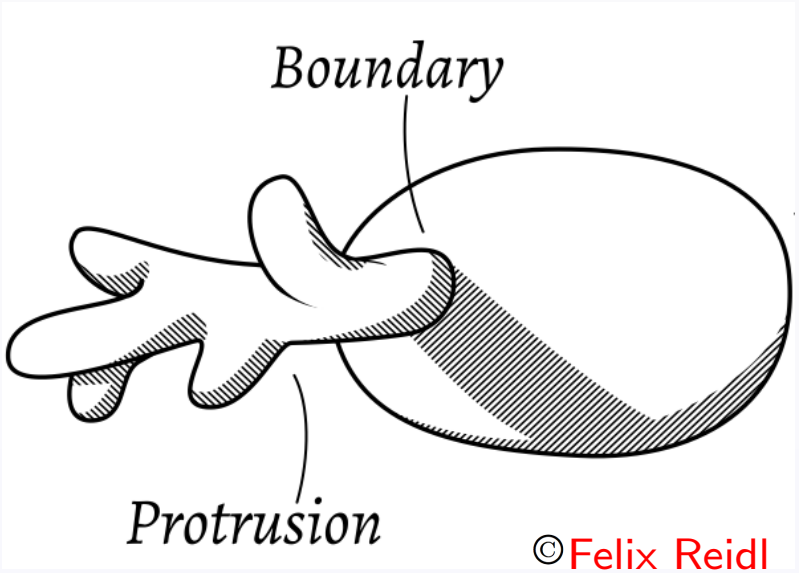
Protrusions



r-protrusion: a set $X \subseteq V(G)$ where

- $\mathbf{tw}(G[X]) \leq r \rightarrow$ treewidth is bounded

Protrusions



r-protrusion: a set $X \subseteq V(G)$ where

- $\mathbf{tw}(G[X]) \leq r \rightarrow$ treewidth is bounded
- $|\partial_G(G)| \leq r \rightarrow$ its boundary is of bounded size

Protrusion decompositions

Protrusion decompositions

Protrusion decompositions

An (α, β)-*protrusion decomposition* of G

is a partition $\mathcal{P} = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$

such that

Protrusion decompositions

An (α, β)-*protrusion decomposition* of G

is a partition $\mathcal{P} = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$

such that

Protrusion decompositions

An (α, β)-*protrusion decomposition* of G

is a partition $\mathcal{P} = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$

such that

- $\max\{\rho, |R_0|\} \leq \alpha$,

Protrusion decompositions

An (α, β)-*protrusion decomposition* of G

is a partition $\mathcal{P} = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$

such that

- $\max\{\rho, |R_0|\} \leq \alpha$,
- each

  $N_G[R_i]$, $i \in \{1, \ldots, \rho\}$, is a β-protrusion of G, and

Protrusion decompositions

An (α, β)-*protrusion decomposition* of G

is a partition $\mathcal{P} = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$

such that

- $\max\{\rho, |R_0|\} \leq \alpha$,
- each

  $N_G[R_i]$, $i \in \{1, \ldots, \rho\}$, is a β-protrusion of G, and
- for every $i \in \{1, \ldots, \rho\}$, $N_G(R_i) \subseteq R_0$.

  Remark: actually, this last condition is

  not necessary! But makes things more visualizable!

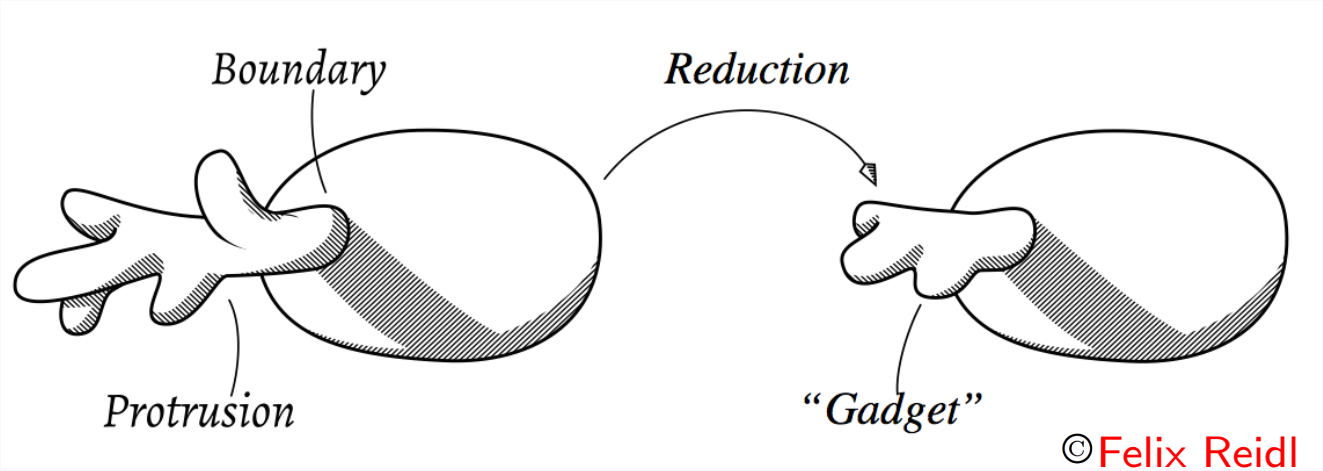
Protrusion replacement

f-protrusion replacement family for $\boxed{\Pi_{\mathbf{p}}}$:

Protrusion replacement

f-protrusion replacement family for $\boxed{\Pi_{\mathbf{p}}}$: a collection $\mathcal{A} = \{\mathsf{A}_i \mid i \geq 0\}$ of algorithms, such that algorithm A_i receives an instance (G, k) of $\boxed{\Pi_{\mathbf{p}}}$ and an i-protrusion X of G with at least $f(i)$ vertices and outputs an equivalent instance (G', k') of $\boxed{\Pi_{\mathbf{p}}}$ where $|V(G')| < |V(G)|$ and $k' \leq k$.

Protrusion replacement



f-protrusion replacement family for $\boxed{\Pi_{\mathbf{p}}}$: a collection $\mathcal{A} = \{\mathsf{A}_i \mid i \geq 0\}$ of algorithms, such that algorithm A_i receives an instance (G, k) of $\boxed{\Pi_{\mathbf{p}}}$ and an i-protrusion X of G with at least $f(i)$ vertices and outputs an equivalent instance (G', k') of $\boxed{\Pi_{\mathbf{p}}}$ where $|V(G')| < |V(G)|$ and $k' \leq k$.

Conditions for the existence of linear kernels

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.
2. [*Algorithmic*] $\boxed{\Pi_{\mathbf{p}}}$ has a protrusion replacement family.

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.
2. [*Algorithmic*] $\boxed{\Pi_{\mathbf{p}}}$ has a protrusion replacement family.

- **1.** $+$ **2** $\rightarrow$ a linear kernel for $\boxed{\Pi_{\mathbf{p}}}$.

[Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh, Thilikos, FOCS 2008]

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.
2. [*Algorithmic*] $\boxed{\Pi_{\mathbf{p}}}$ has a protrusion replacement family.

- **1.** + **2** $\rightarrow$ a linear kernel for $\boxed{\Pi_{\mathbf{p}}}$.

[Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh, Thilikos, FOCS 2008]

To achieve Conditions **1** and **2.** we need some more definitions!

CMSO-expressibility

Let $\mathbf{p}$ be a graph optimization parameter and

let $\boxed{\Pi_{\mathbf{p}}}$ be the corresponding graph optimization problem.

CMSO-expressibility

Let $\mathbf{p}$ be a graph optimization parameter and

let $\Pi_{\mathbf{p}}$ be the corresponding graph optimization problem.

Recall that:

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leqslant k \wedge \phi(G, S)=\texttt{true}\}\}$$

CMSO-expressibility

Let $\mathbf{p}$ be a graph optimization parameter and

let $\boxed{\Pi_{\mathbf{p}}}$ be the corresponding graph optimization problem.

Recall that:

$$\mathbf{p}(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leqslant k \wedge \boxed{\phi(G, S)=\texttt{true}}\}$$

- If ϕ is expressible in Monadic Second Order Logic, then we say that

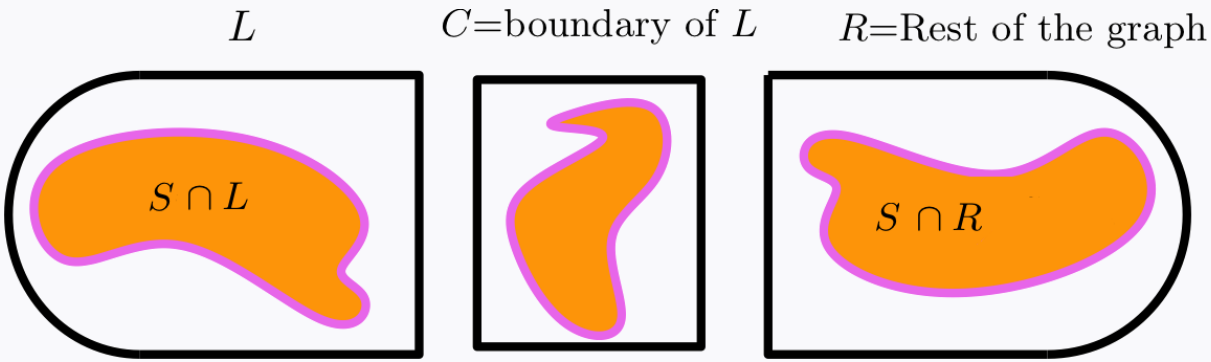
$\boxed{\Pi_{\mathbf{p}}}$ is *CMSO-expressibile*

Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and G be a graph

Linear Separability

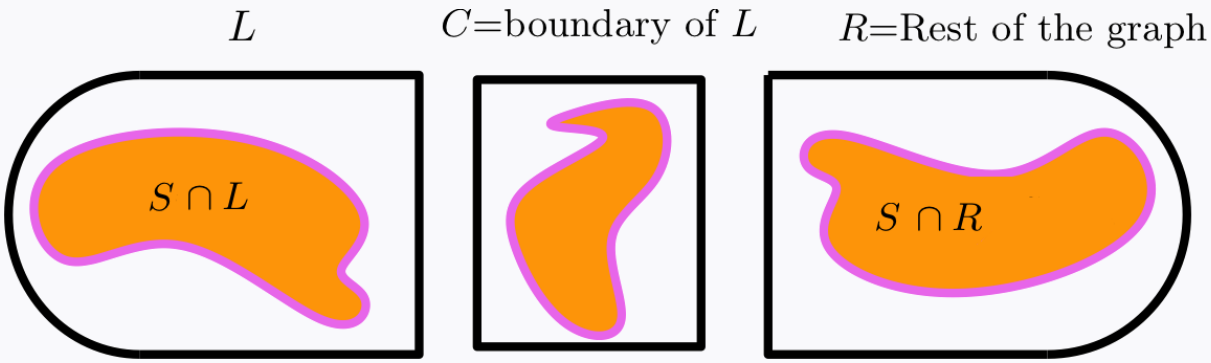
Let $\mathbf{p}$ be an graph optimization parameter and G be a graph



The subset optimization problem $\boxed{\Pi_{\mathbf{p}}}$ is *linearly separable* if

Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and G be a graph

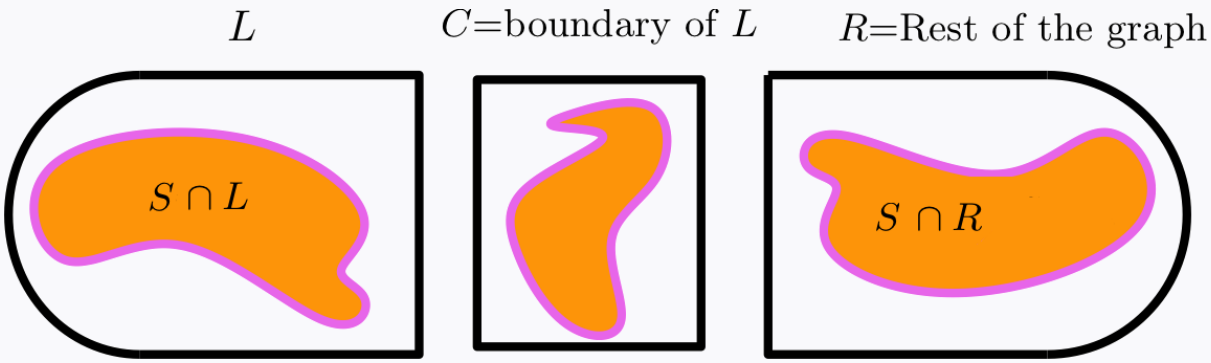


The subset optimization problem $\boxed{\Pi_{\mathbf{p}}}$ is *linearly separable* if, for any graph G and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq \mathbf{p}(G[L]) \leq |S \cap L| + c \cdot t$$

Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and G be a graph



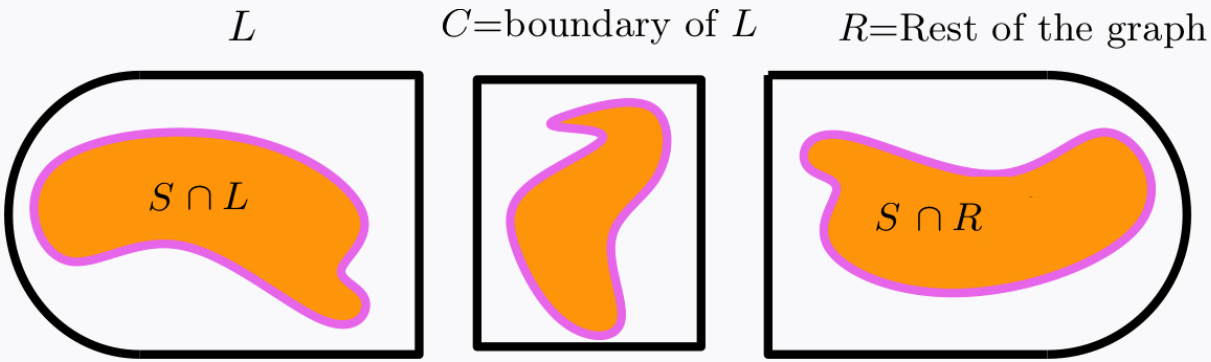
The subset optimization problem $\boxed{\Pi_{\mathbf{p}}}$ is *linearly separable* if, for any graph G and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq \mathbf{p}(G[L]) \leq |S \cap L| + c \cdot t$$

where S is a solution certificate for $\mathbf{p}$

Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and G be a graph



The subset optimization problem $\boxed{\Pi_{\mathbf{p}}}$ is *linearly separable* if, for any graph G and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

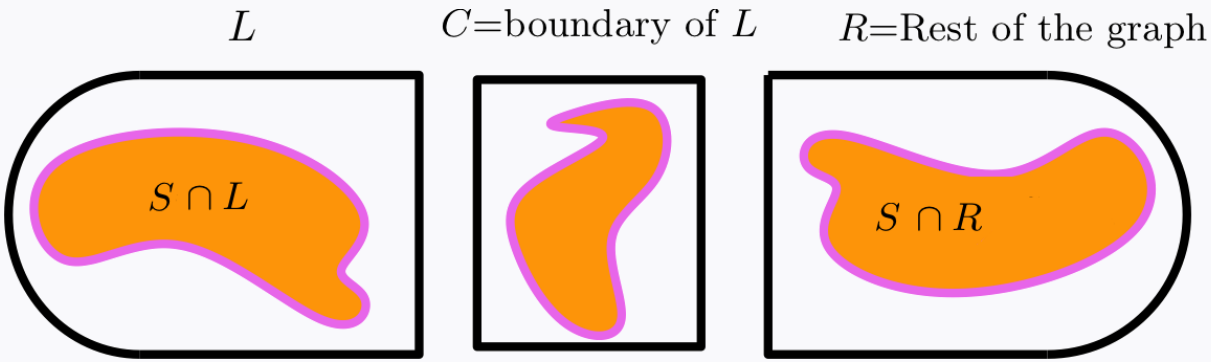
$$|S \cap L| - c \cdot t \leq \mathbf{p}(G[L]) \leq |S \cap L| + c \cdot t$$

where S is a solution certificate for $\mathbf{p}$

- More generally: $c \cdot t \to f(t)$ defines *separable* $\boxed{\Pi_{\mathbf{p}}}$

Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and G be a graph



The subset optimization problem $\boxed{\Pi_{\mathbf{p}}}$ is *linearly separable* if, for any graph G and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq \mathbf{p}(G[L]) \leq |S \cap L| + c \cdot t$$

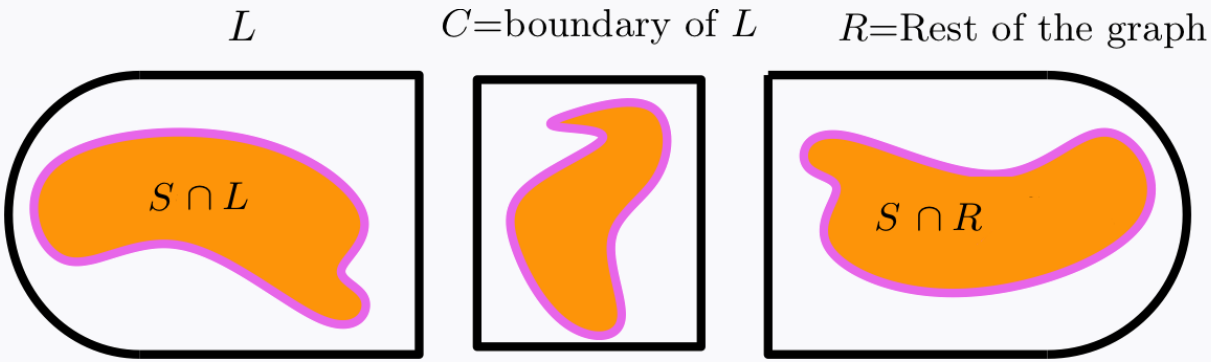
where S is a solution certificate for $\mathbf{p}$

- More generally: $c \cdot t \to f(t)$ defines *separable* $\boxed{\Pi_{\mathbf{p}}}$

p-PATH is not separable while

Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and G be a graph



The subset optimization problem $\boxed{\Pi_{\mathbf{p}}}$ is *linearly separable* if, for any graph G and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq \mathbf{p}(G[L]) \leq |S \cap L| + c \cdot t$$

where S is a solution certificate for $\mathbf{p}$

- More generally: $c \cdot t \to f(t)$ defines *separable* $\boxed{\Pi_{\mathbf{p}}}$

p-Path is not separable while

p-Dominating Set, and p-Feedback Vertex Set are linearly separable.

Conditions for the existence of linear kernels

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.
2. [*Algorithmic*] $\boxed{\Pi_{\mathbf{p}}}$ has a protrusion replacement family.

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.
2. [*Algorithmic*] $\boxed{\Pi_{\mathbf{p}}}$ has a protrusion replacement family.

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\rightarrow$ **1.**

[Fomin, Lokshtanov, Saurabh, Thilikos, SODA 2011]

Conditions for the existence of linear kernels

1. [*Combinatorial*] If $\mathbf{p}(G) \leq k$, then G has an $(O(k), O(1))$-protrusion decomposition.
2. [*Algorithmic*] $\boxed{\Pi_{\mathbf{p}}}$ has a protrusion replacement family.

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\rightarrow$ **1.**

[Fomin, Lokshtanov, Saurabh, Thilikos, SODA 2011]

- CMSO-expressibility + Linear separability $\rightarrow$ **2.**

[Fomin, Lokshtanov, Saurabh, Thilikos, 2015]

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\mathbf{A+B}}$ **1.**

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$\mathbf{p}(G) \leq k \Rightarrow G$ has a treewidth η-modulator S where $|S| \leq 2 \cdot k$.

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$\mathbf{p}(G) \leq k \Rightarrow G$ has a treewidth η-modulator S where $|S| \leq 2 \cdot k$.

Remark: the bidimensionality condition: $\mathbf{p}^{-1}(k) = \sqrt{k}$ is necessary here!

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$\mathbf{p}(G) \leq k \Rightarrow G$ has a treewidth η-modulator S where $|S| \leq 2 \cdot k$.

Remark: the bidimensionality condition: $\mathbf{p}^{-1}(k) = \sqrt{k}$ is necessary here!

Lemma B: Assume that:

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$\mathbf{p}(G) \leq k \Rightarrow G$ has a treewidth η-modulator S where $|S| \leq 2 \cdot k$.

Remark: the bidimensionality condition: $\mathbf{p}^{-1}(k) = \sqrt{k}$ is necessary here!

Lemma B: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property.

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$\mathbf{p}(G) \leq k \Rightarrow G$ has a treewidth η-modulator S where $|S| \leq 2 \cdot k$.

Remark: the bidimensionality condition: $\mathbf{p}^{-1}(k) = \sqrt{k}$ is necessary here!

Lemma B: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property.
2. G has a treewidth η-modulator S for some positive integer η

We comment the proof of the first fact:

- **SQGM**/**SQGC** + Linear separability + bidimensionality $\xrightarrow{\textbf{A+B}}$ **1.**

Definition: S is a a treewidth η-modulator of G if $\mathbf{tw}(G \setminus S) \leq \eta$

i.e., a certificate for η-$\mathbf{twm}(G) \leq k$

Lemma A: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property
2. $\boxed{\Pi_{\mathbf{p}}}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$\mathbf{p}(G) \leq k \Rightarrow G$ has a treewidth η-modulator S where $|S| \leq 2 \cdot k$.

Remark: the bidimensionality condition: $\mathbf{p}^{-1}(k) = \sqrt{k}$ is necessary here!

Lemma B: Assume that:

1. $\mathcal{G}$ is a graph class with the **SQGM**/**SQGC** property.
2. G has a treewidth η-modulator S for some positive integer η

Then there exists an integer r such that G has $(2 \cdot |S|, r)$-protrusion decomposition.

Bidimensionality and kernels.

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

Bidimensionality and kernels.

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is CMSO-expressible

Bidimensionality and kernels.

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

Bidimensionality and kernels.

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

Bidimensionality and kernels.

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

iv. is restricted to some **SQGM**/**SQGC**-graph class

Bidimensionality and kernels.

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

iv. is restricted to some **SQGM**/**SQGC**-graph class

Then $\boxed{\Pi_{\mathbf{p}}}$ admits a linear kernel.

Bidimensionality and approximation.

- Just for the history we also mention the following:

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is $\approx$ CMSO-expressible

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is $\approx$ CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is $\approx$ CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is $\approx$ CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

iv. is restricted to some **SQGM**/**SQGC**-graph class

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is $\approx$ CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

iv. is restricted to some **SQGM**/**SQGC**-graph class

Then $\boxed{\Pi_{\mathbf{p}}}$ admits an EPTAS

Bidimensionality and approximation.

- Just for the history we also mention the following:

Theorem: Let $\boxed{\Pi_{\mathbf{p}}}$ be a subset optimization parameterized problem that

i. is $\approx$ CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,

iii. is linearly separable,

iv. is restricted to some **SQGM**/**SQGC**-graph class

Then $\boxed{\Pi_{\mathbf{p}}}$ admits an EPTAS

EPTAS = (**E**fficient **P**olynomial-**T**ime **A**pproximation **S**cheme)

[Demaine, Hajiaghay, SODA 2005]
[Fomin, Lokshtanov, Raman, Saurabh, SODA 2011]

Irrelevant vertex technique

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:
- If $\mathbf{tw}(G) > f(k)$ then we we declare **VICTORY**! (enemy surrenters!)

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $\mathbf{tw}(G) > f(k)$ then we we declare **VICTORY**! (enemy surrenters!)
- if $\mathbf{tw}(G) \leq f(k)$ then the **CAVALRY** comes! (DP algorthims or just Courcelle's th.)

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $\mathbf{tw}(G) > f(k)$ then we we declare **VICTORY**! (enemy surrenters!)
- if $\mathbf{tw}(G) \leq f(k)$ then the **CAVALRY** comes! (DP algorthims or just Courcelle's th.)

- What about when YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ do not have bounded treewidth?

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $\mathbf{tw}(G) > f(k)$ then we we declare **VICTORY**! (enemy surrenters!)
- if $\mathbf{tw}(G) \leq f(k)$ then the **CAVALRY** comes! (DP algorthims or just Courcelle's th.)

- What about when YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ do not have bounded treewidth?

- In this case we have to **FIGHT**!!!!! (untl the **CAVALRY** comes or enemy surrenters)

General algorithmic strategy (so far):

We need the following two facts (for suitable functions f and G):

1. [Combinatorial] $\mathbf{p}(G) \leq k \Rightarrow \mathbf{tw}(G) \leq f(k)$

i.e., YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $\mathbf{p}(G) \leq k$ in $g(\mathbf{tw}(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $\mathbf{tw}(G) > f(k)$ then we we declare **VICTORY**! (enemy surrenters!)
- if $\mathbf{tw}(G) \leq f(k)$ then the **CAVALRY** comes! (DP algorthims or just Courcelle's th.)

- What about when YES/NO-instances of $\boxed{\Pi_{\mathbf{p}}}$ do not have bounded treewidth?

- In this case we have to **FIGHT**!!!!! (untl the **CAVALRY** comes or enemy surrenters)

- For many problems, instances of big enough treewidth may contain part whose removal does not change the answer to the original question.

Consider the following problem:

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

Suppose we have an instance G of big enough treewidth!

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

Suppose we have an instance G of big enough treewidth!

Then G contains a big grid as a minor

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

Suppose we have an instance G of big enough treewidth!

Then G contains a big grid as a minor

This means that G contains a subgraph that is a subdivision of a "big enough" wall!

Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

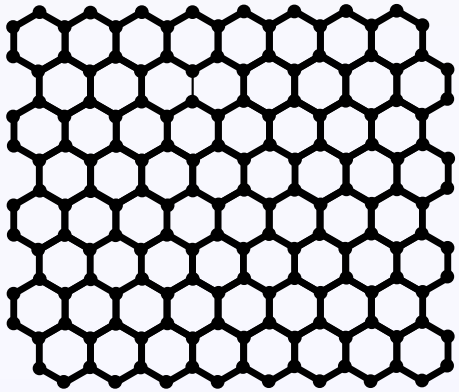
Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

Suppose we have an instance G of big enough treewidth!

Then G contains a big grid as a minor

This means that G contains a subgraph that is a subdivision of a "big enough" wall!



Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

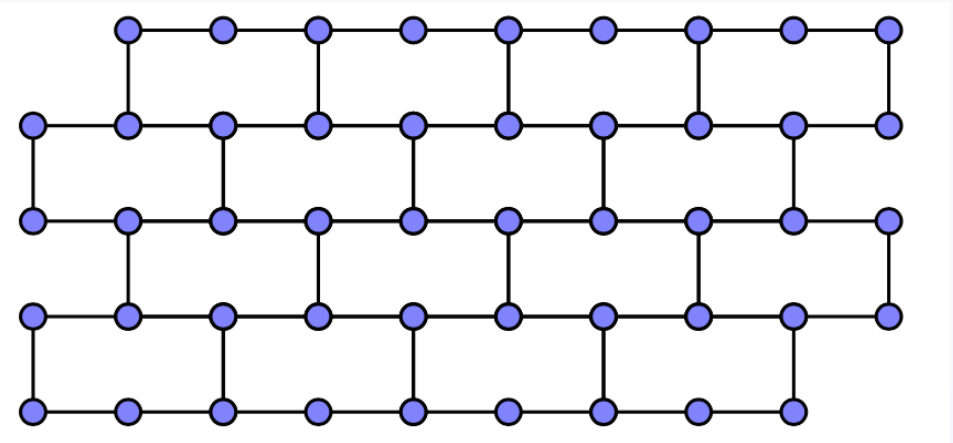
Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

Suppose we have an instance G of big enough treewidth!

Then G contains a big grid as a minor

This means that G contains a subgraph that is a subdivision of a "big enough" wall!



Consider the following problem:

p-Odd Cycle Packing

Instance: A graph G and an integer $k \geq 0$.

Parameter: k,

Question: Does G contains k mutually vertex-disjoint odd cycles?

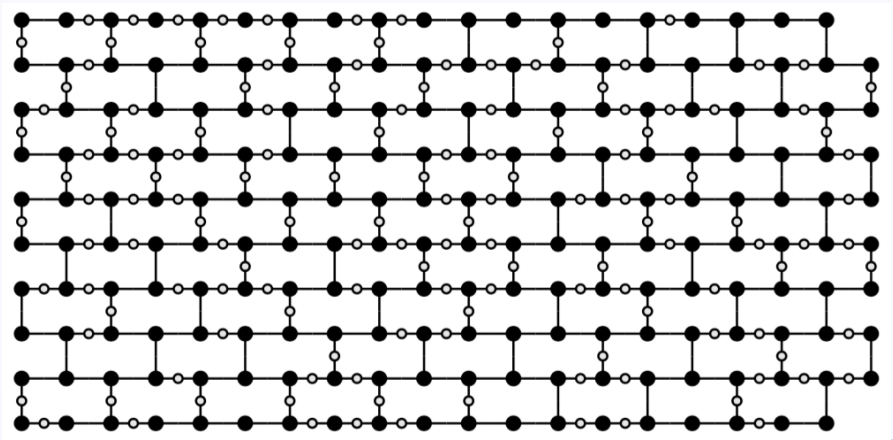
Without the "odd" demand, the problem is minor-bidimensional and we are not afraid!

How to deal with "oddness" demand (at least) for planar instances?

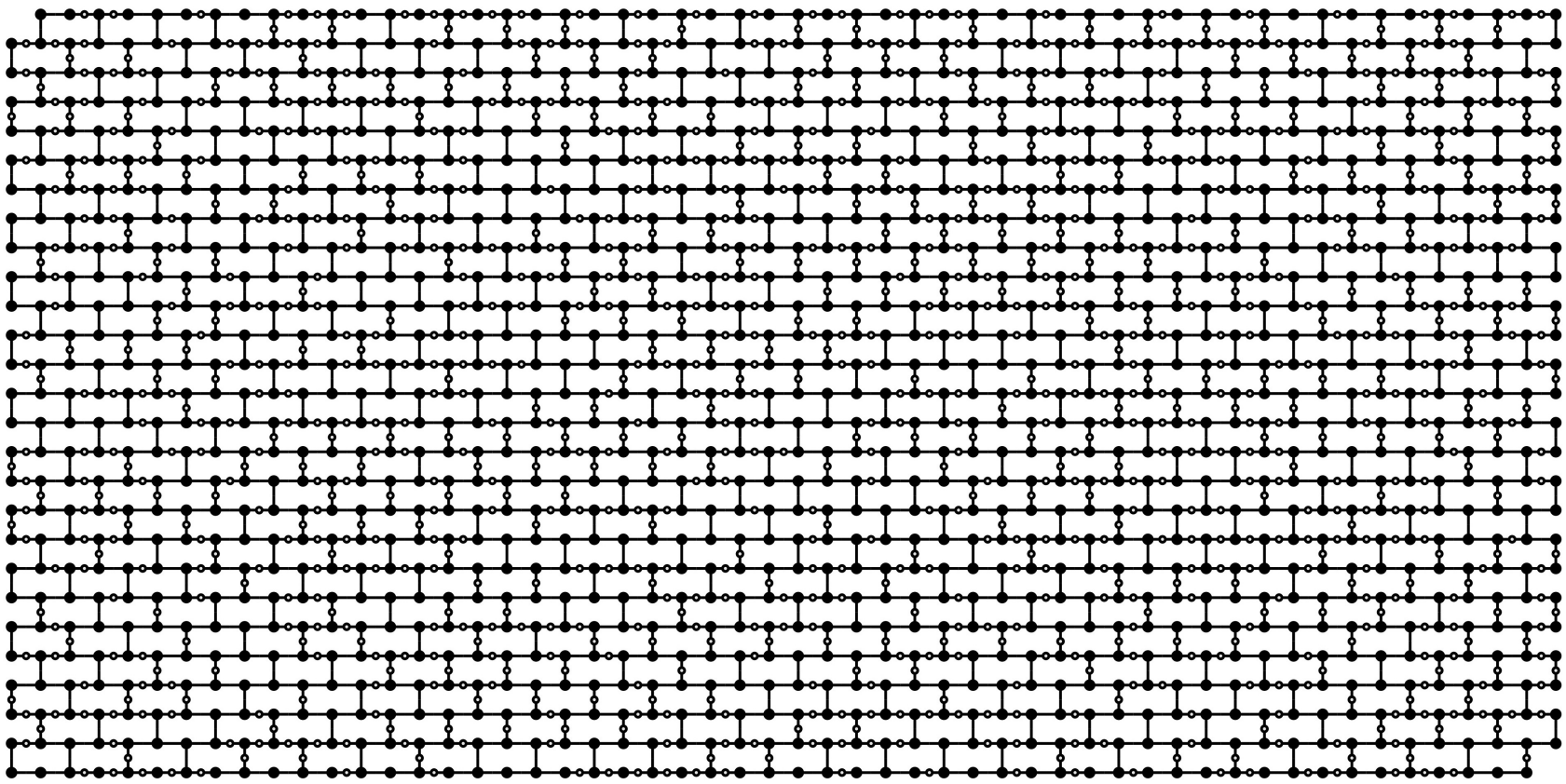
Suppose we have an instance G of big enough treewidth!

Then G contains a big grid as a minor

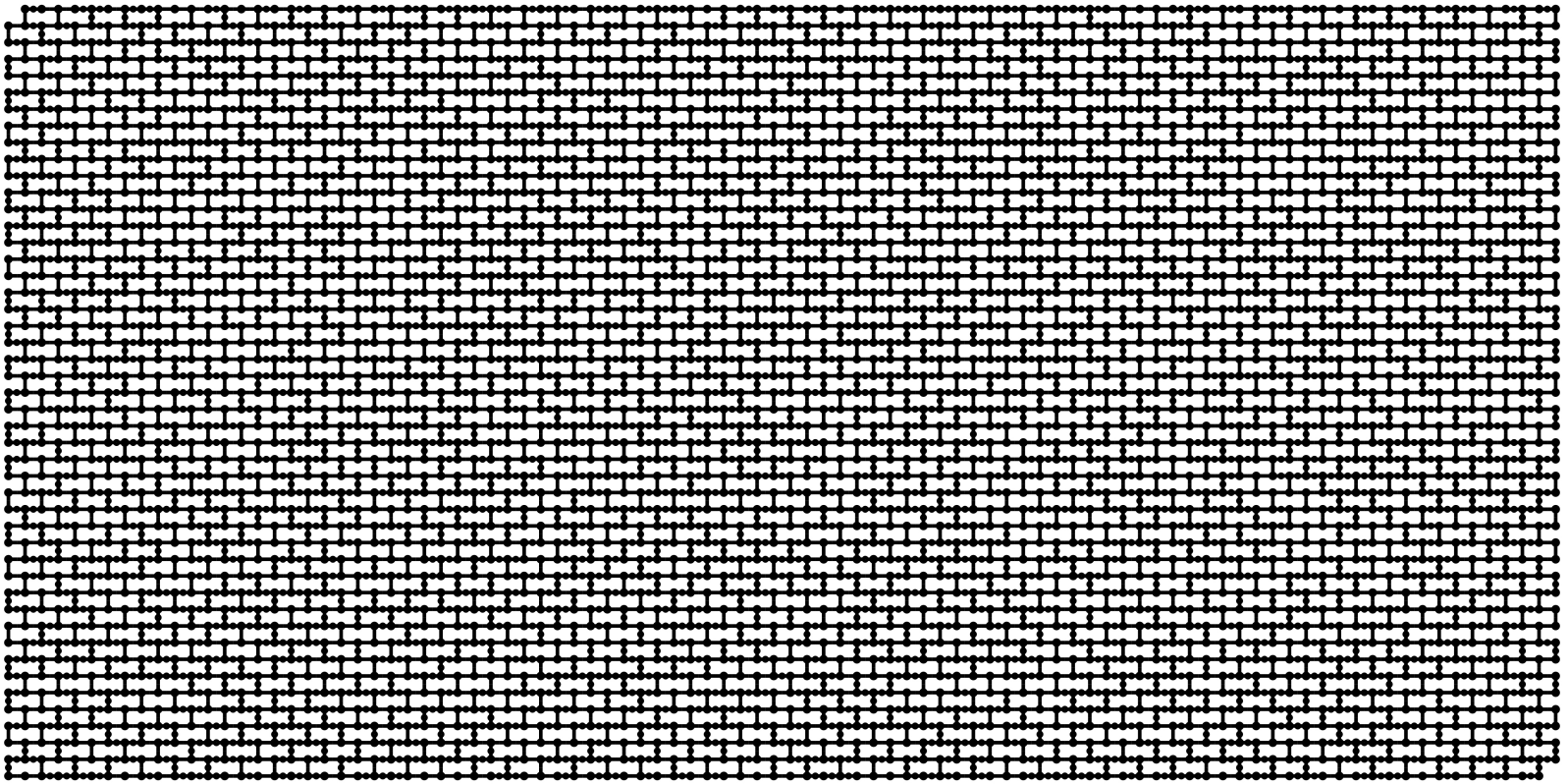
This means that G contains a subgraph that is a subdivision of a "big enough" wall!



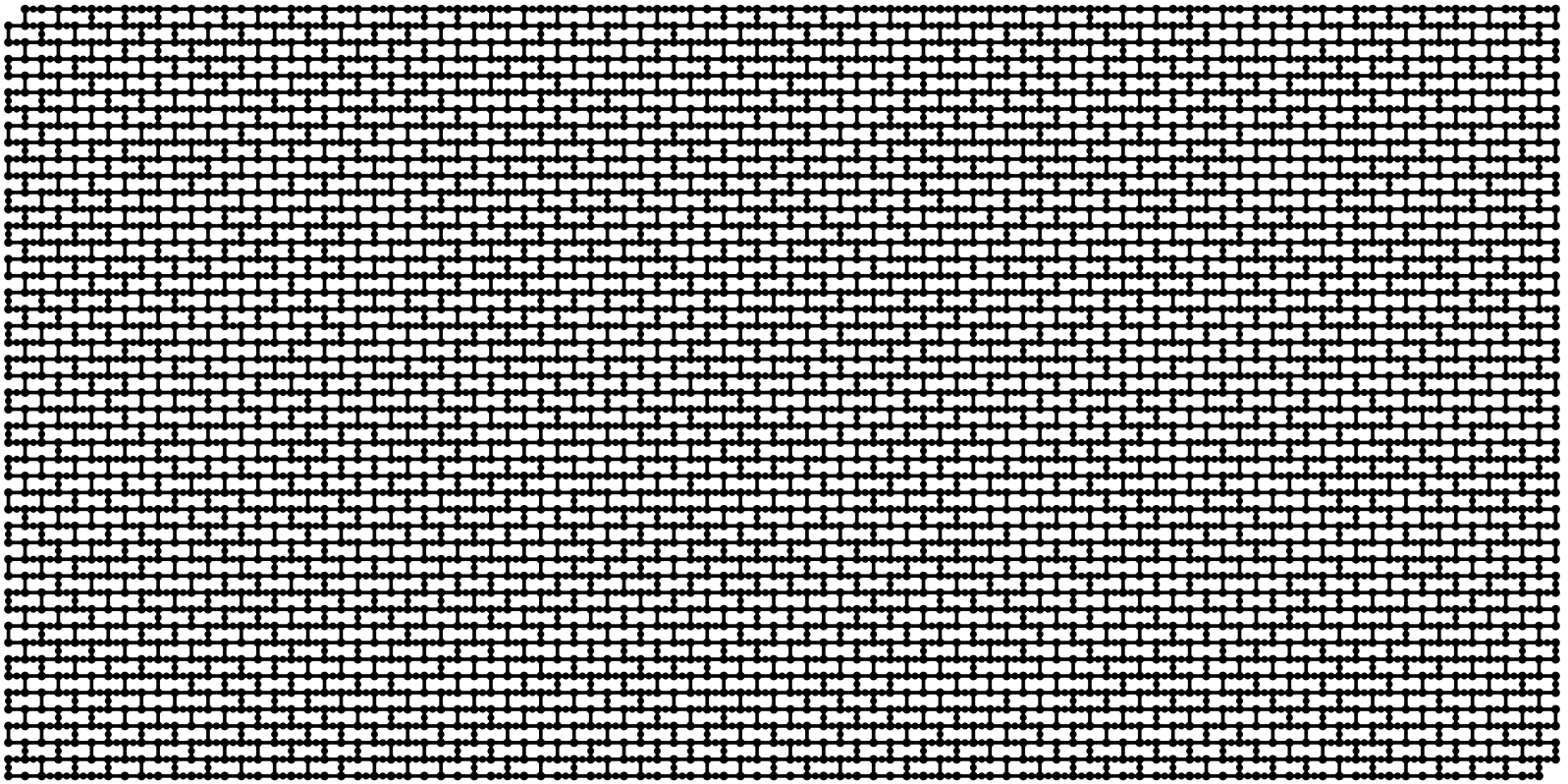
Actually we can assume we have the subdivision of a quite big wall!



Actually we can assume we have the subdivision of a quite big wall!

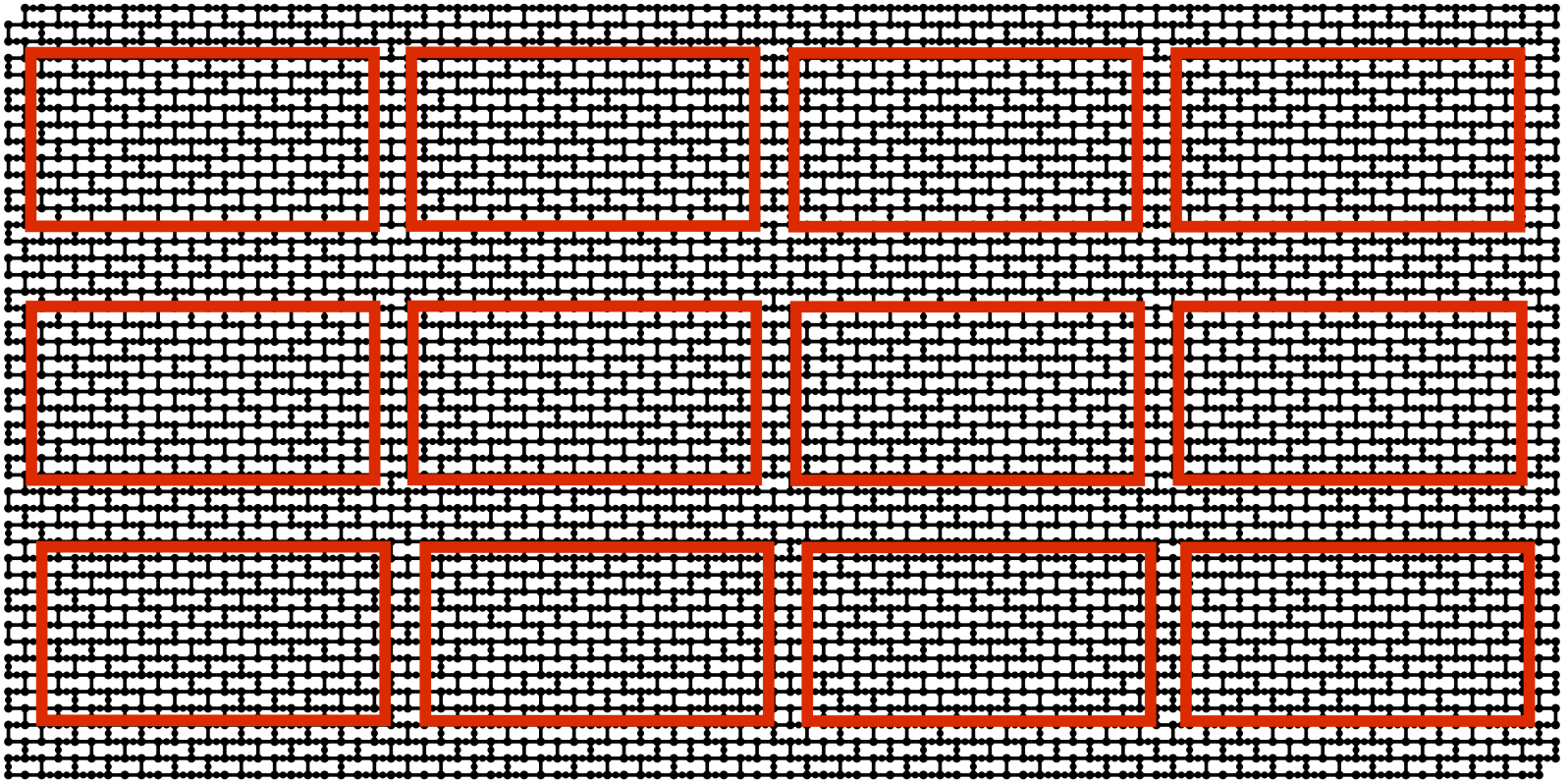


Actually we can assume we have the subdivision of a quite big wall!



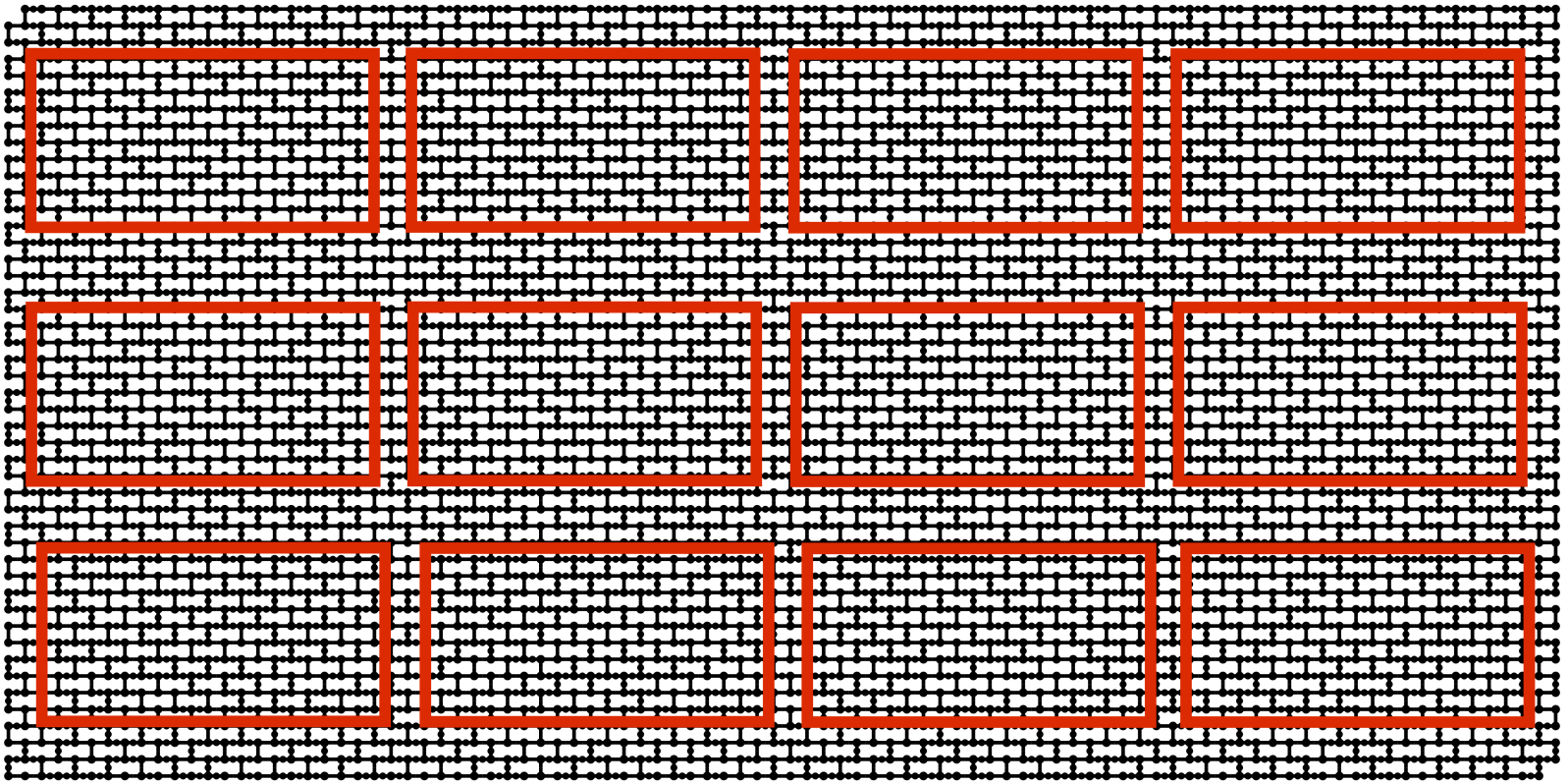
The height of this wall is $h = \lceil \sqrt{k} \rceil \cdot (2(k+1)+1)$ (in the s-wall above $h = 46$)

Actually we can assume we have the subdivision of a quite big wall!



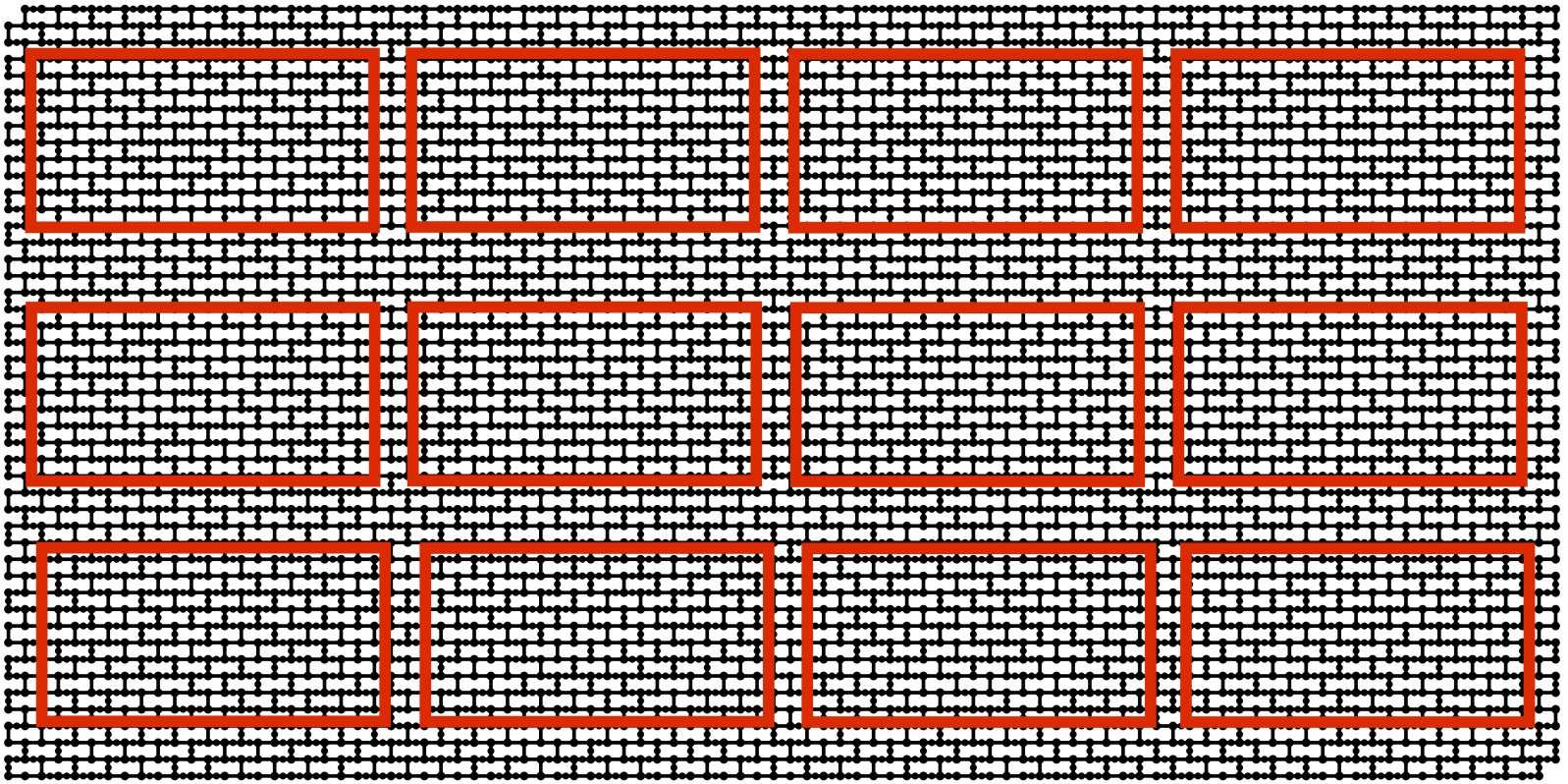
We locate k subwalls, each of heigh $2 \cdot (k+1)$.

Actually we can assume we have the subdivision of a quite big wall!



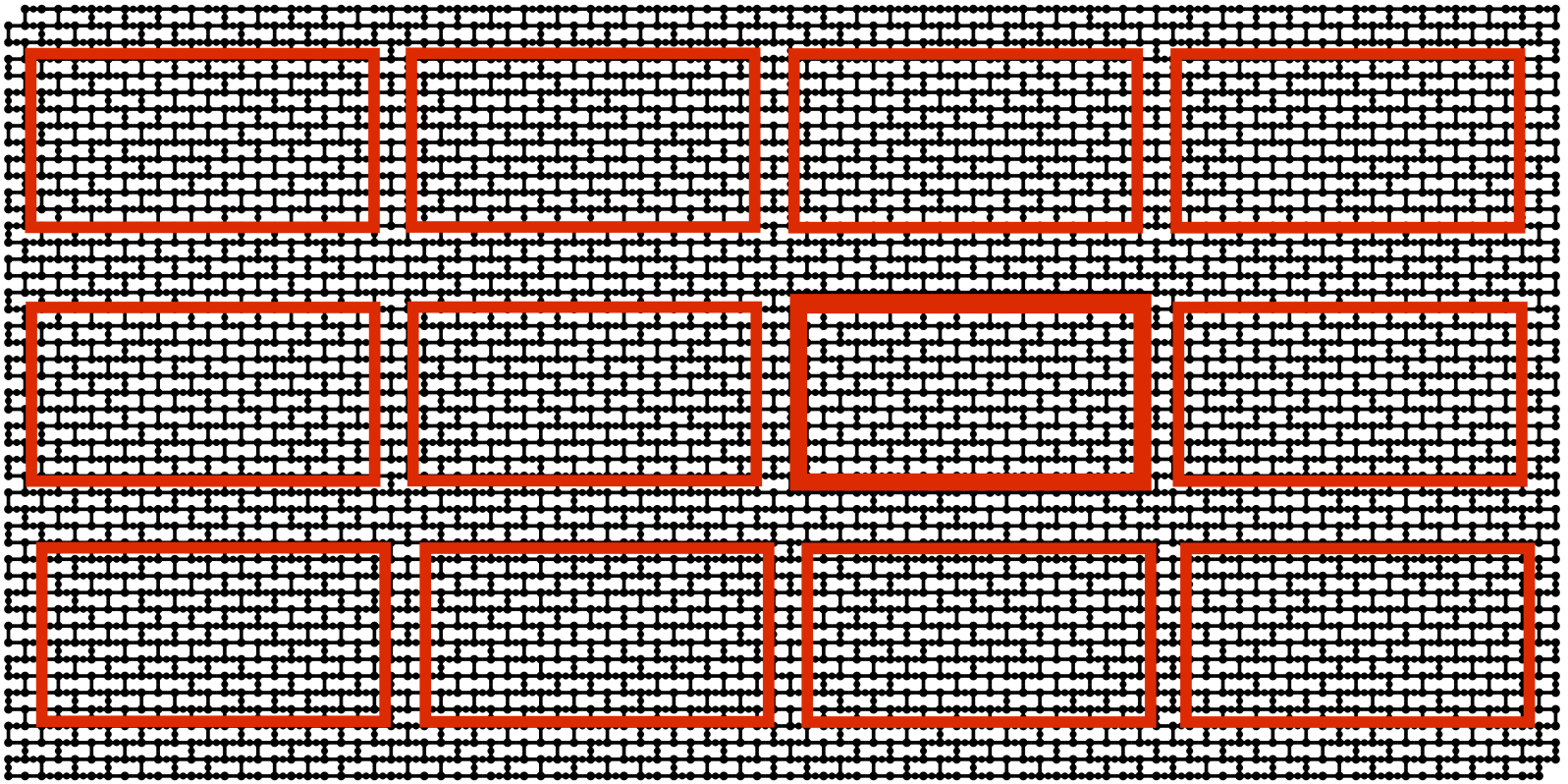
Let $G_1, \ldots, G_k$ be the graphs inside the perimetries of these subwalls

Actually we can assume we have the subdivision of a quite big wall!

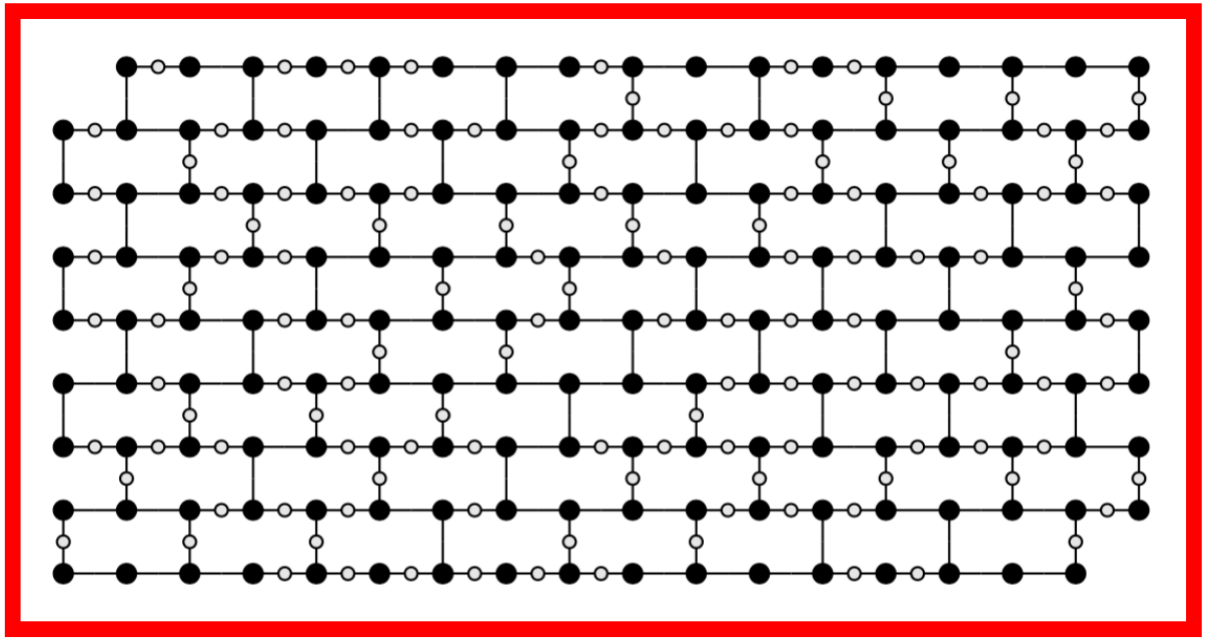


If all of them are non-bipartite then we answer YES and we are done!

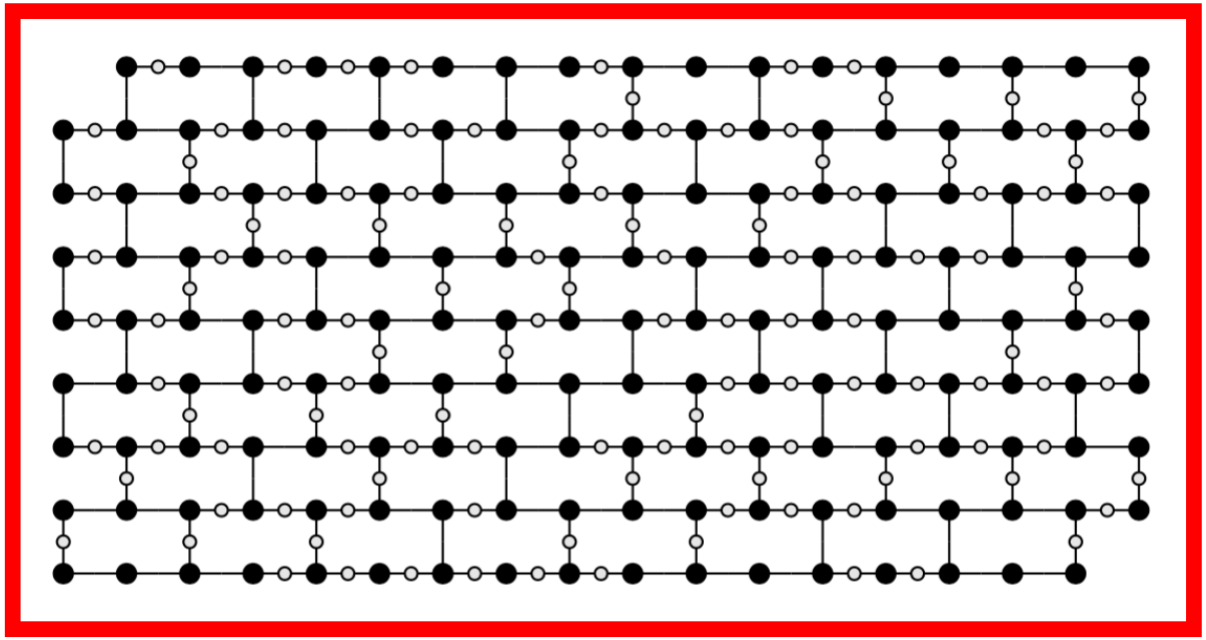
Actually we can assume we have the subdivision of a quite big wall!



If not, then consider the one that is bipartite.

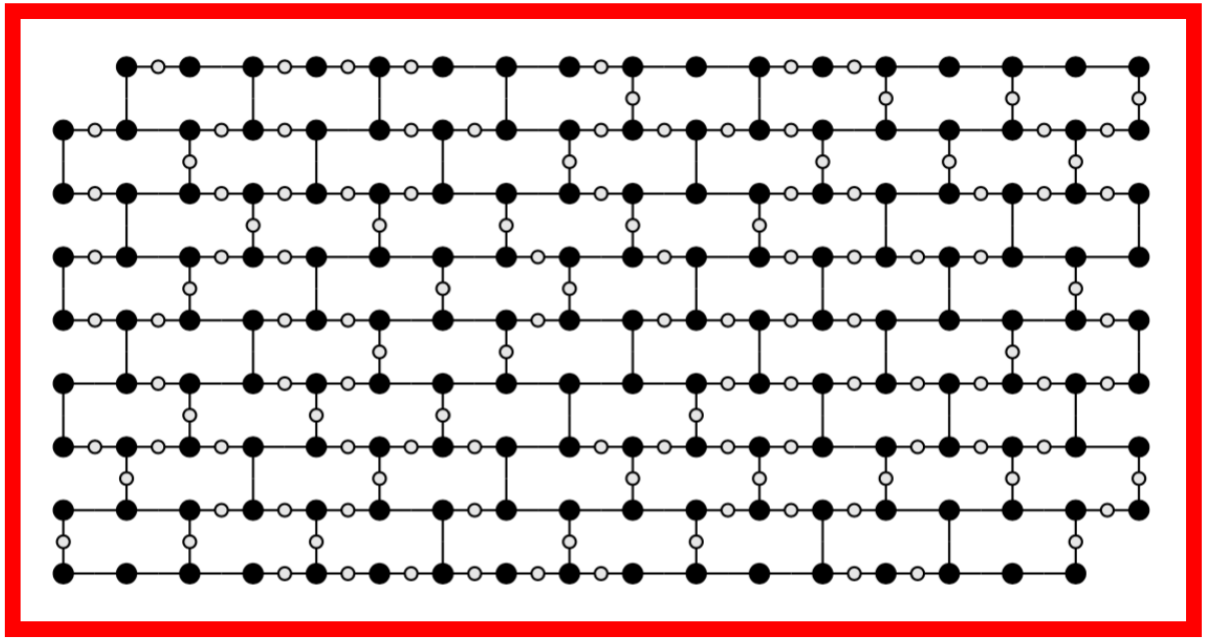


If not, then consider the one that is bipartite.



If not, then consider the one that is bipartite.

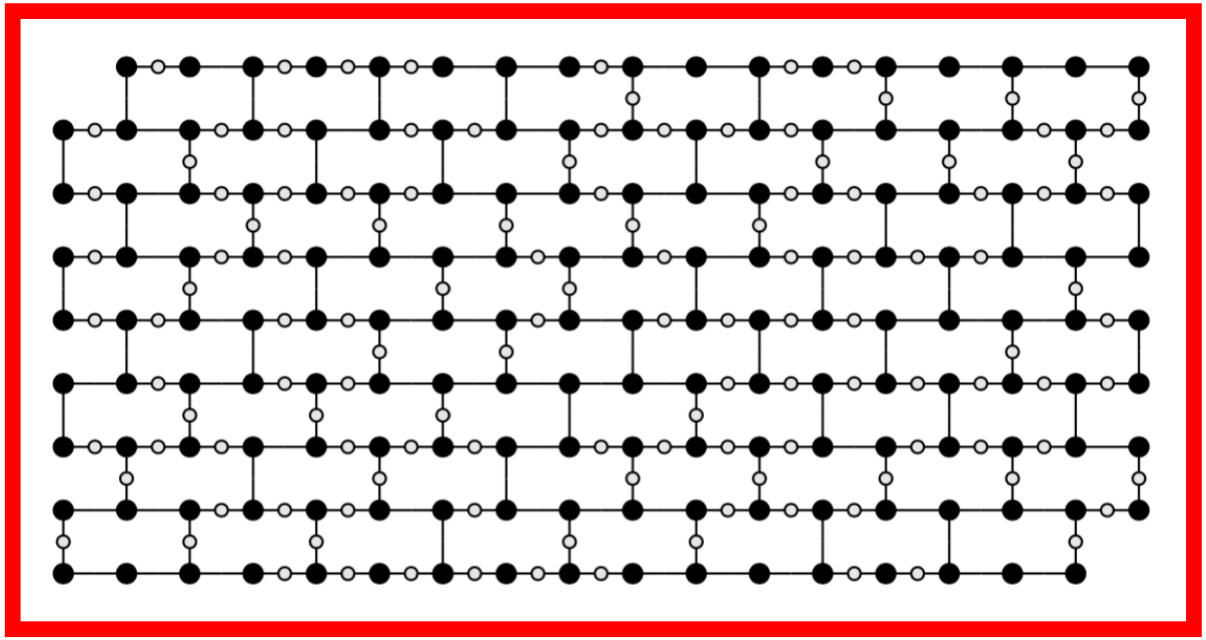
All Cycles entirely inside the perimetry of this subwall are even!



If not, then consider the one that is bipartite.

All Cycles entirely inside the perimetry of this subwall are even!

We claim that (given that the height is $2 \cdot (k+1)$) the middle vertex is irrelevant

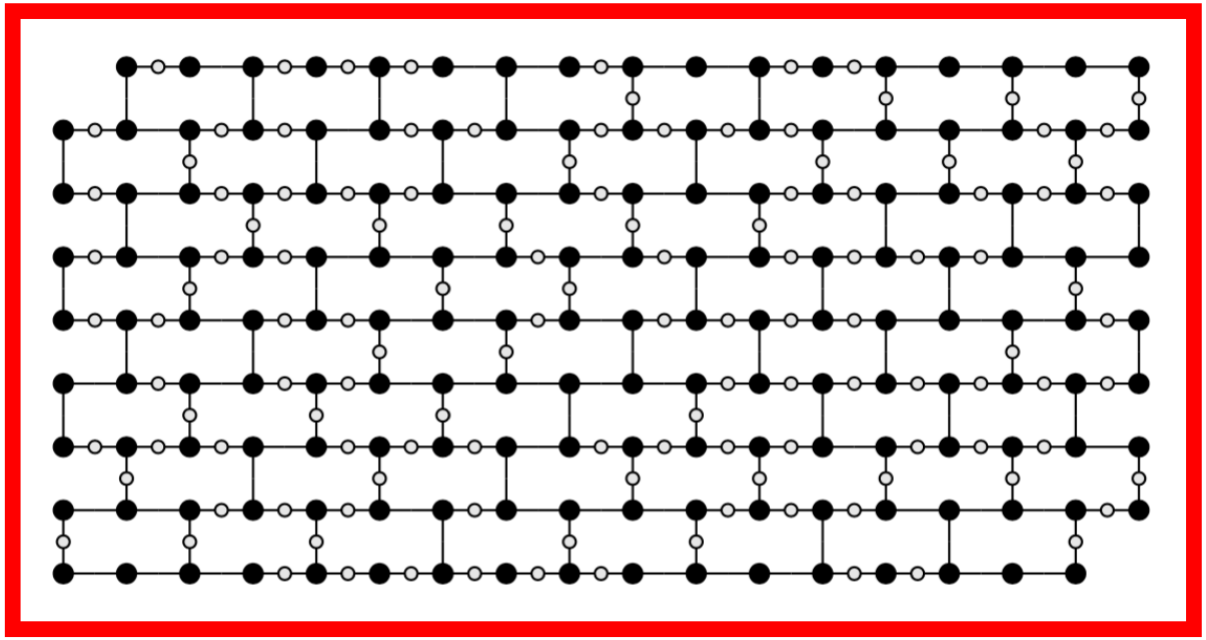


If not, then consider the one that is bipartite.

All Cycles entirely inside the perimetry of this subwall are even!

We claim that (given that the height is $2 \cdot (k+1)$) the middle vertex is irrelevant

We have to prove that (G, k) is a YES-instance $\iff$ $(G \setminus x, k)$ is a YES-instance



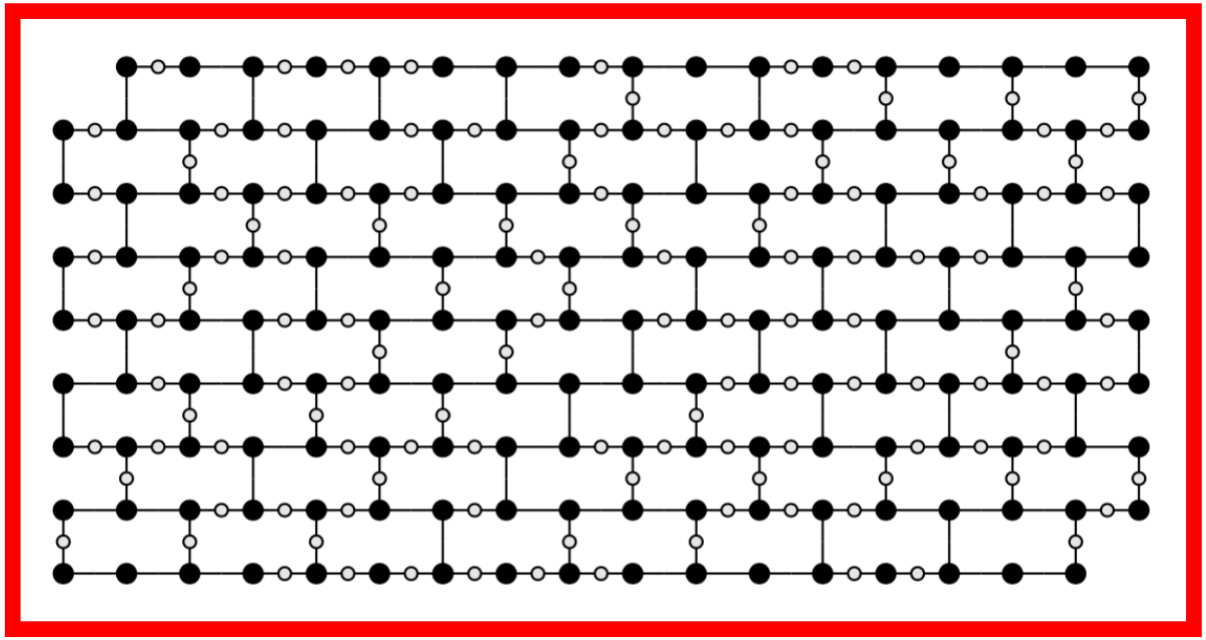
If not, then consider the one that is bipartite.

All Cycles entirely inside the perimetry of this subwall are even!

We claim that (given that the height is $2 \cdot (k+1)$) the middle vertex is irrelevant

We have to prove that (G, k) is a YES-instance $\iff$ $(G \setminus x, k)$ is a YES-instance

The $\Longleftarrow$ direction is trivial: if $G \setminus x$ has k odd disjoint cycles, so does G.



If not, then consider the one that is bipartite.

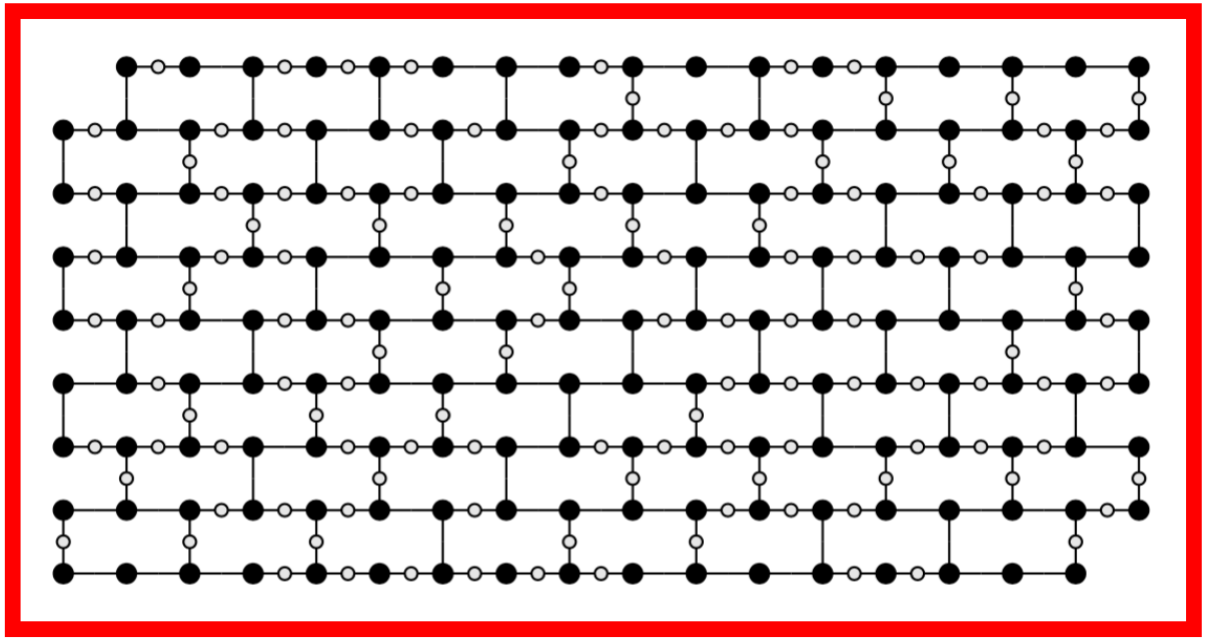
All Cycles entirely inside the perimetry of this subwall are even!

We claim that (given that the height is $2 \cdot (k+1)$) the middle vertex is irrelevant

We have to prove that (G, k) is a YES-instance $\iff$ $(G \setminus x, k)$ is a YES-instance

The $\Longleftarrow$ direction is trivial: if $G \setminus x$ has k odd disjoint cycles, so does G.

For the "$\Longrightarrow$" assume that $G \setminus x$ has $\geq k$ odd cycles.



If not, then consider the one that is bipartite.

All Cycles entirely inside the perimetry of this subwall are even!

We claim that (given that the height is $2 \cdot (k+1)$) the middle vertex is irrelevant

We have to prove that (G, k) is a YES-instance $\iff$ $(G \setminus x, k)$ is a YES-instance

The $\Longleftarrow$ direction is trivial: if $G \setminus x$ has k odd disjoint cycles, so does G.

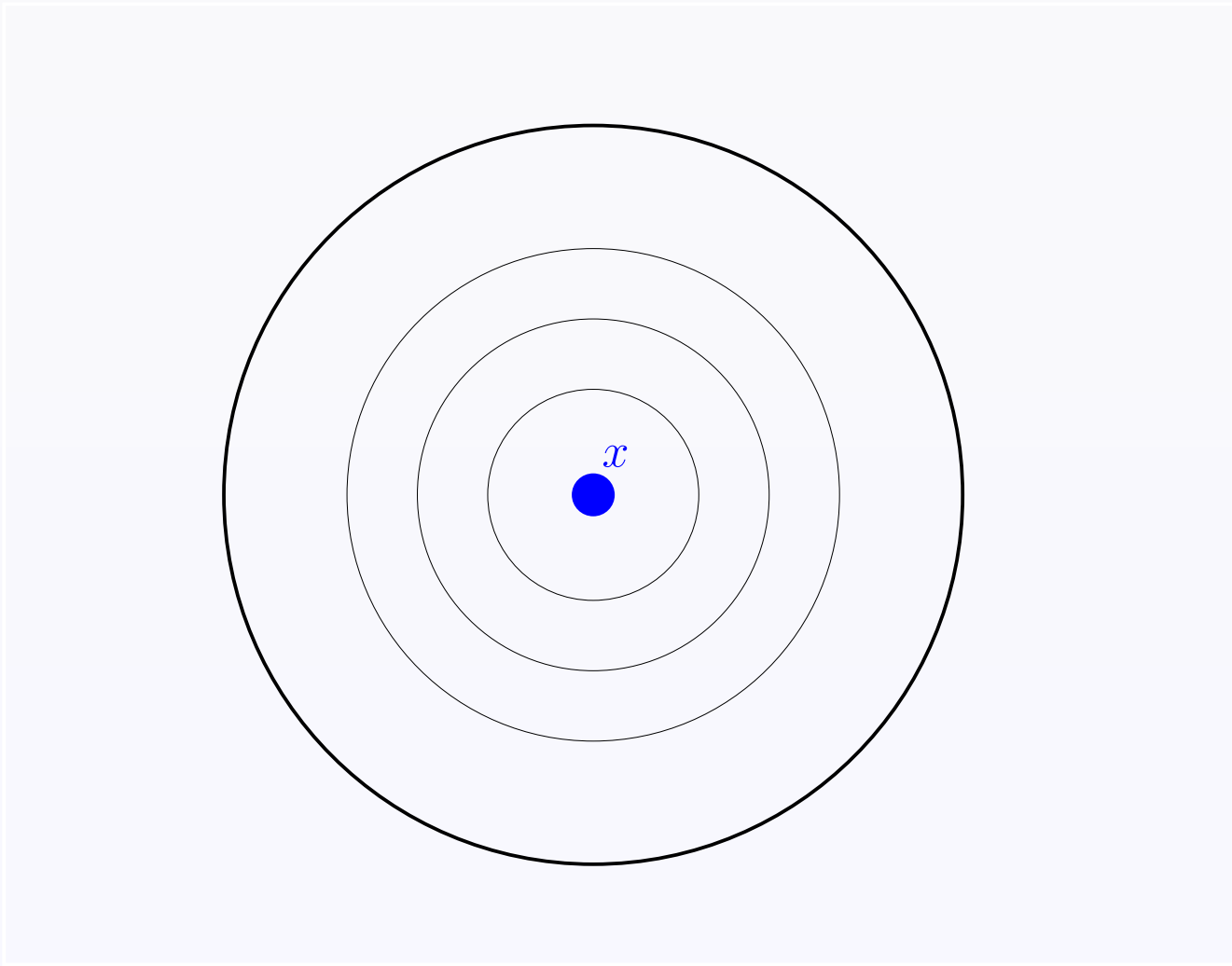
For the "$\Longrightarrow$" assume that $G \setminus x$ has $\geq k$ odd cycles.

We will prove that $G \setminus x$ has k odd disjoint cycles avoiding x.

We detect, using the layers of the wall, $k+1$, homocentric cycles around x

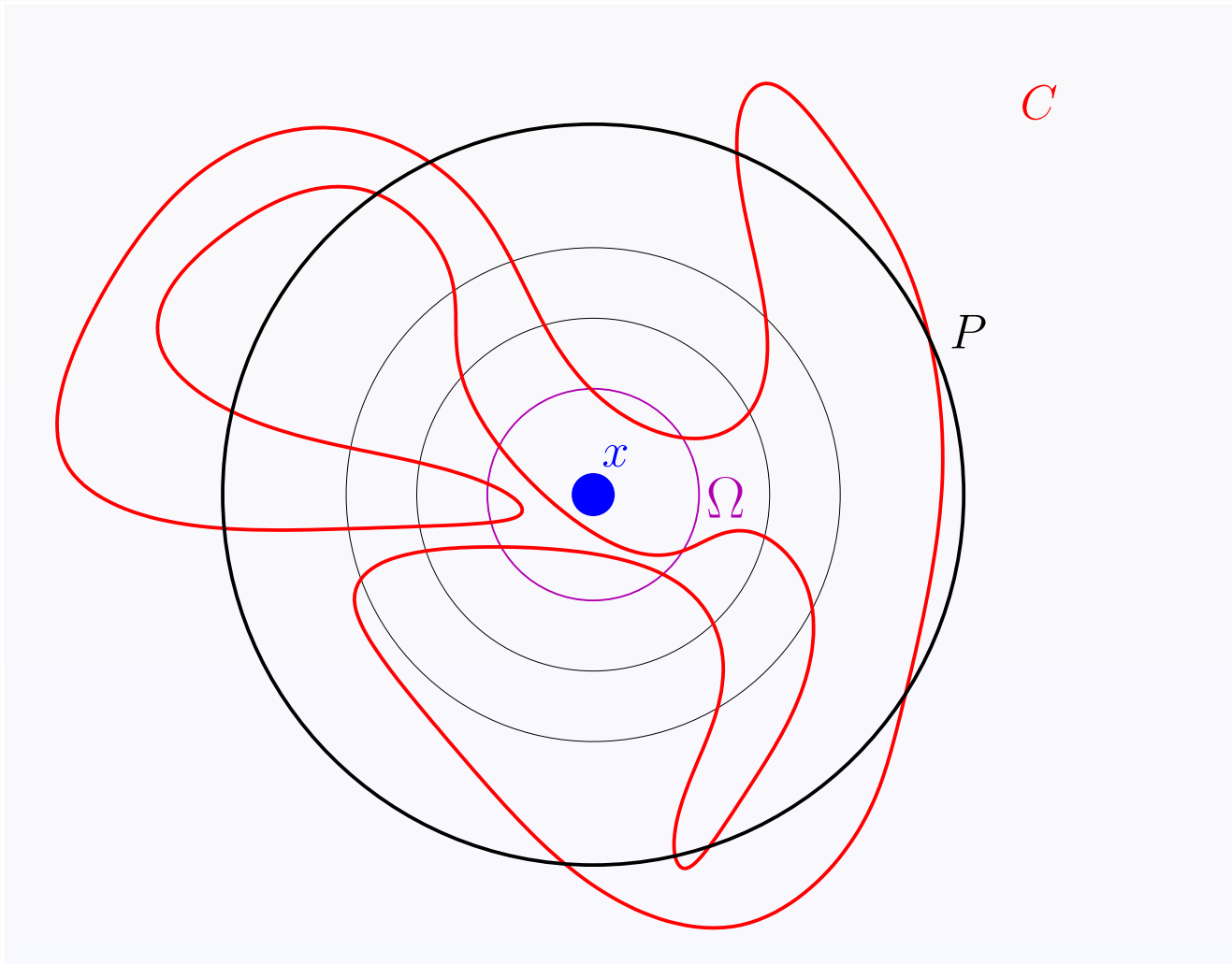
If G has $k+1$ disjoint odd cycles we are done (x meets only one of them)

Therefore G has at exactly k disjoint odd cycles.



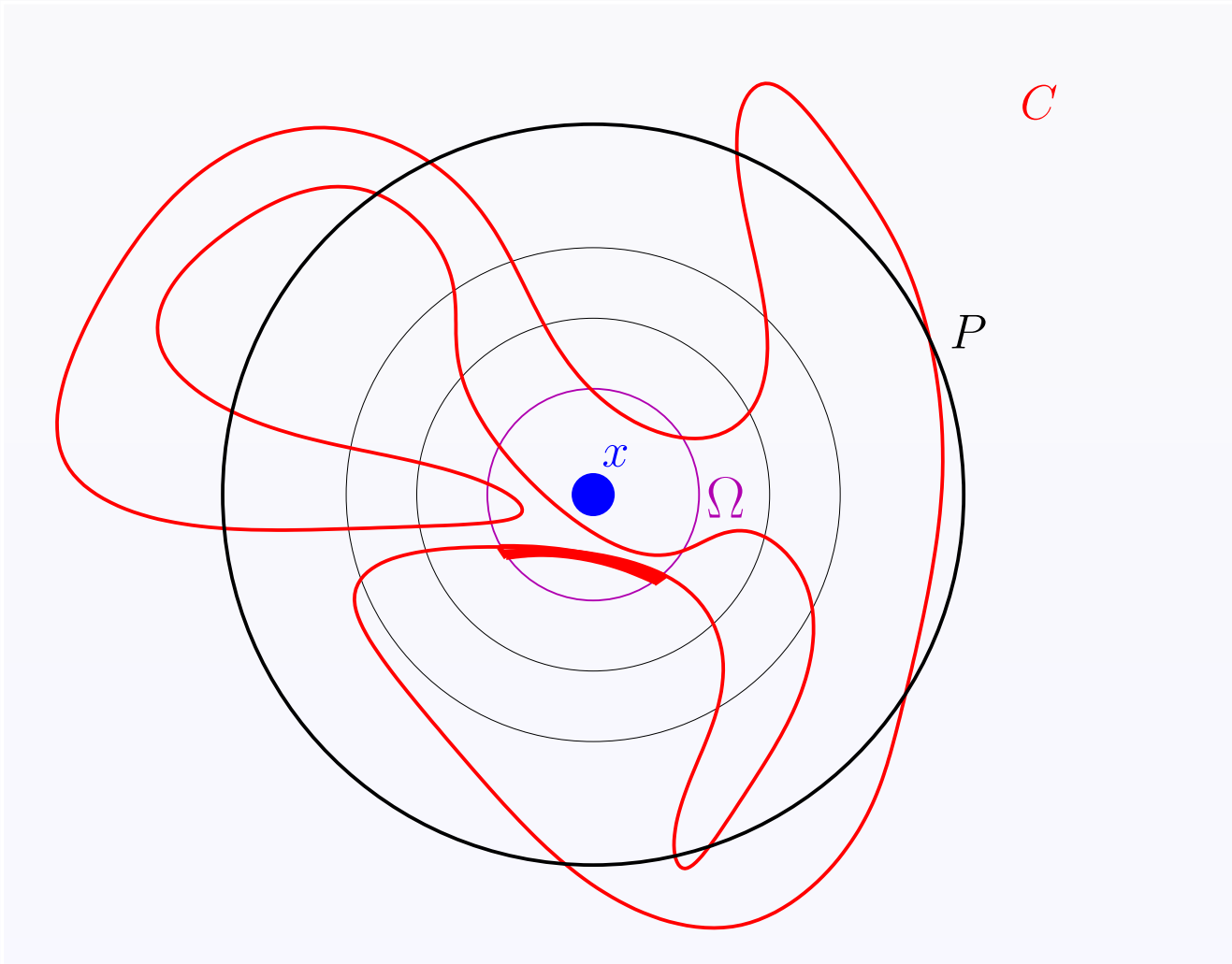
Assume # chords of the k disjoint cycles "cropped" by the homocentric cycles is minimized. For example C crosses Ω 4 times and C crosses perimetry P 5 times.

We argue that none of these k cycles can cross the inner cycle Ω, thus x is irrelevant!



Suppose, to the contrary, that some cycle C crosses the inner cycle Ω.

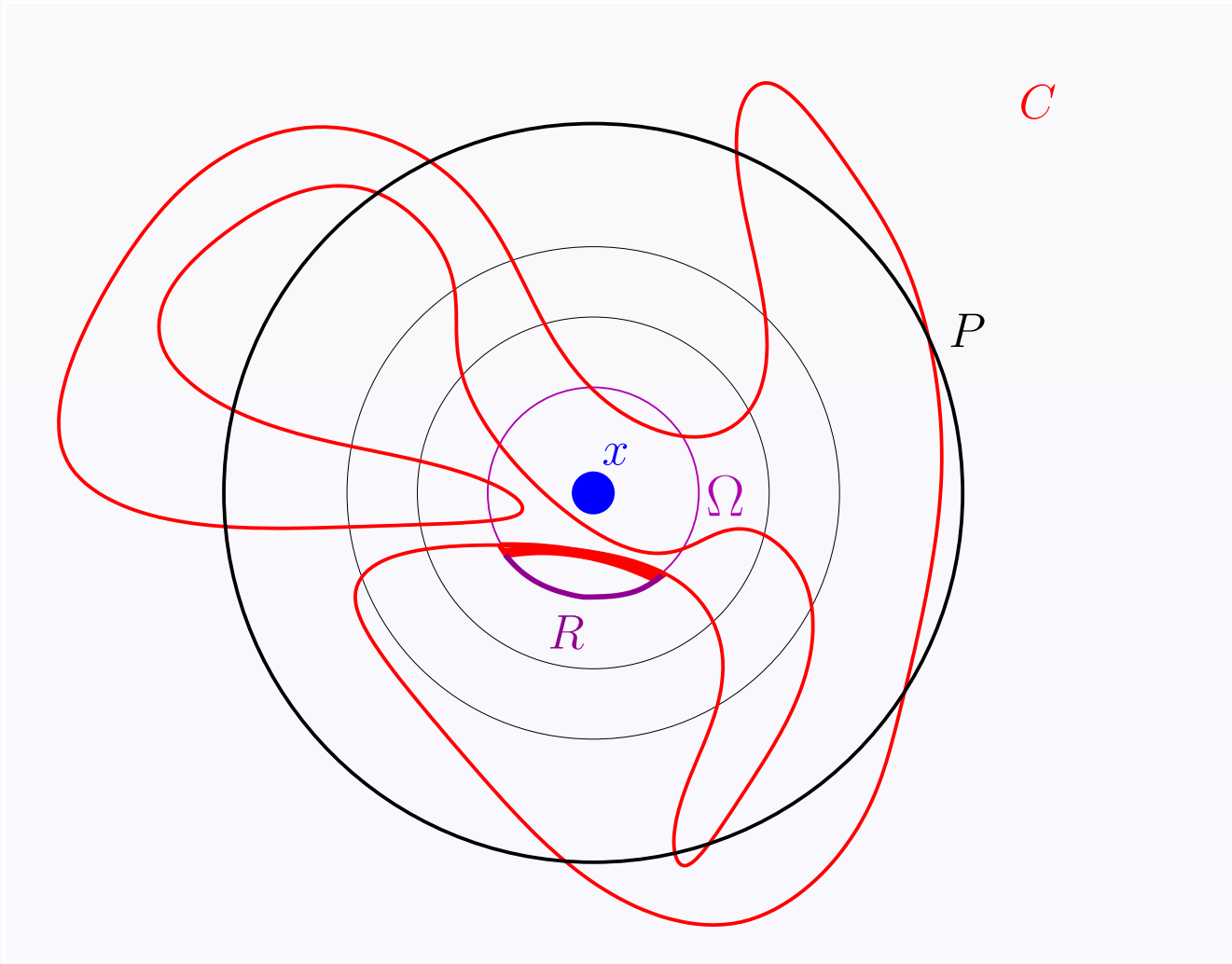
Consider an "extremal" chord X: one of the two paths of Ω does not contain any other endpoint of a chord.



Suppose, to the contrary, that some cycle C crosses the inner cycle Ω.

Consider an "extremal" chord! This defines a (same parity) segment R of Ω

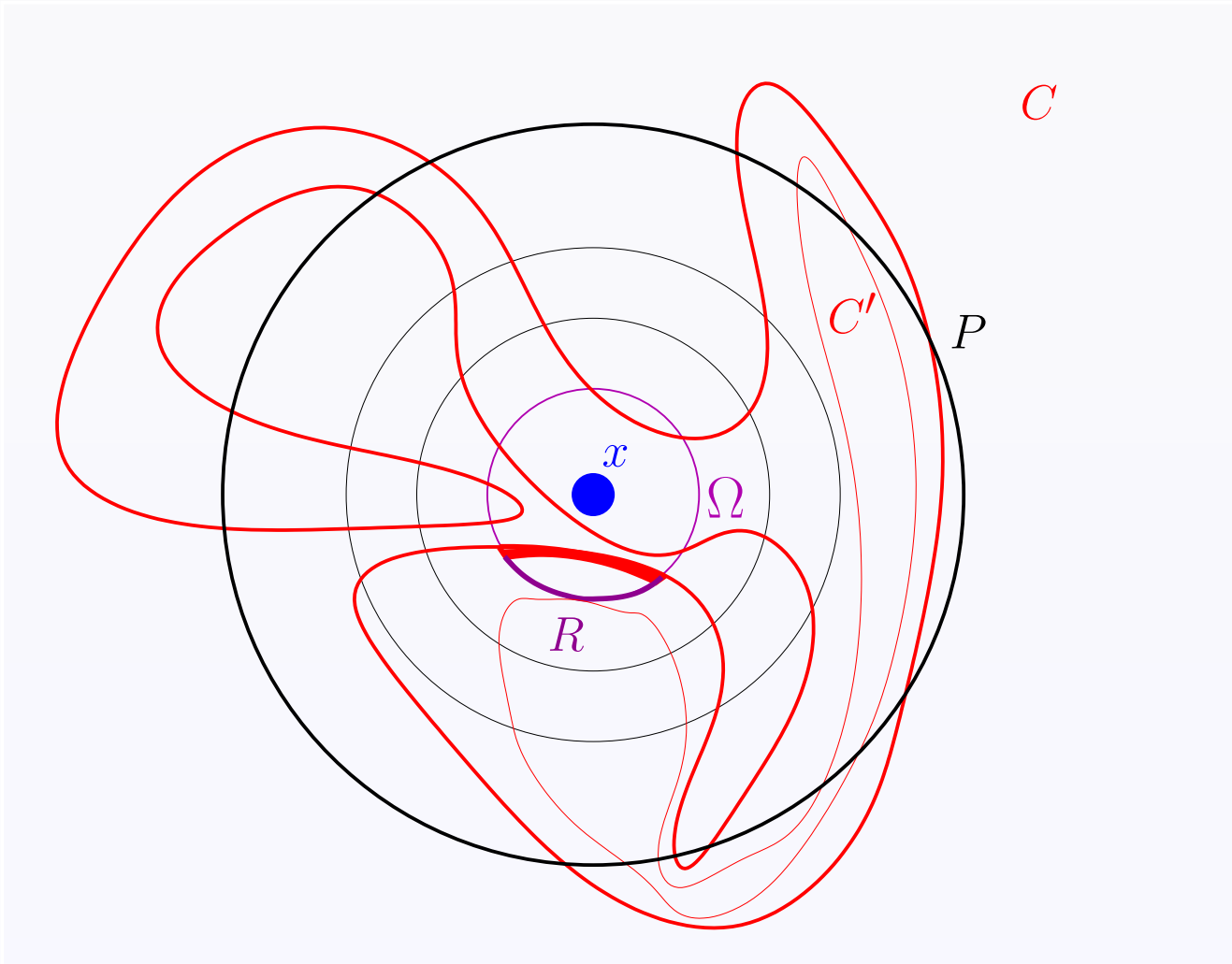
By minimality R should be met by some



Suppose, to the contrary, that some cycle C crosses the inner cycle Ω.

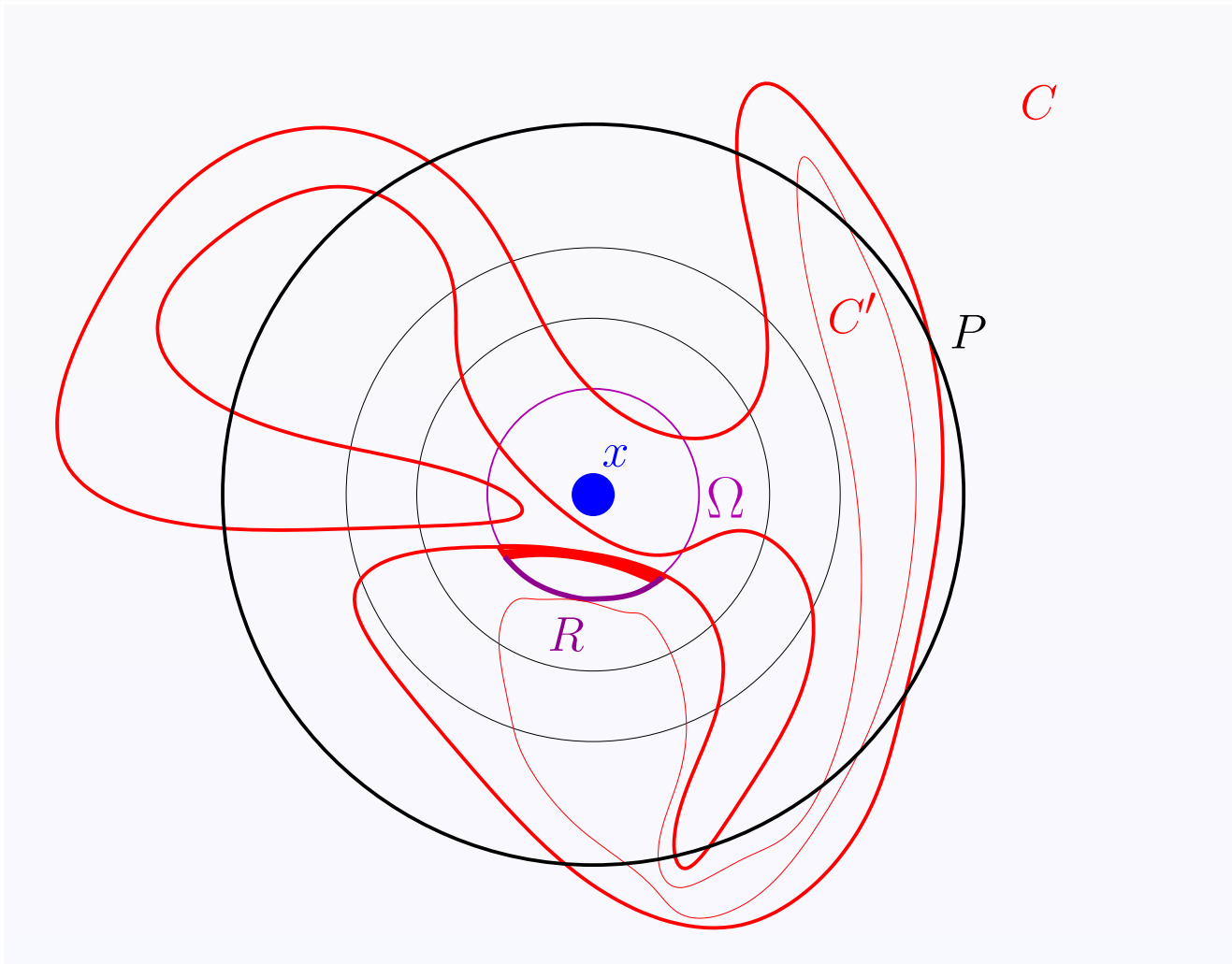
Consider an "extremal" chord! This defines a (same parity) segment R of Ω

By minimality R should be met by some other cycle $C' \neq C$.

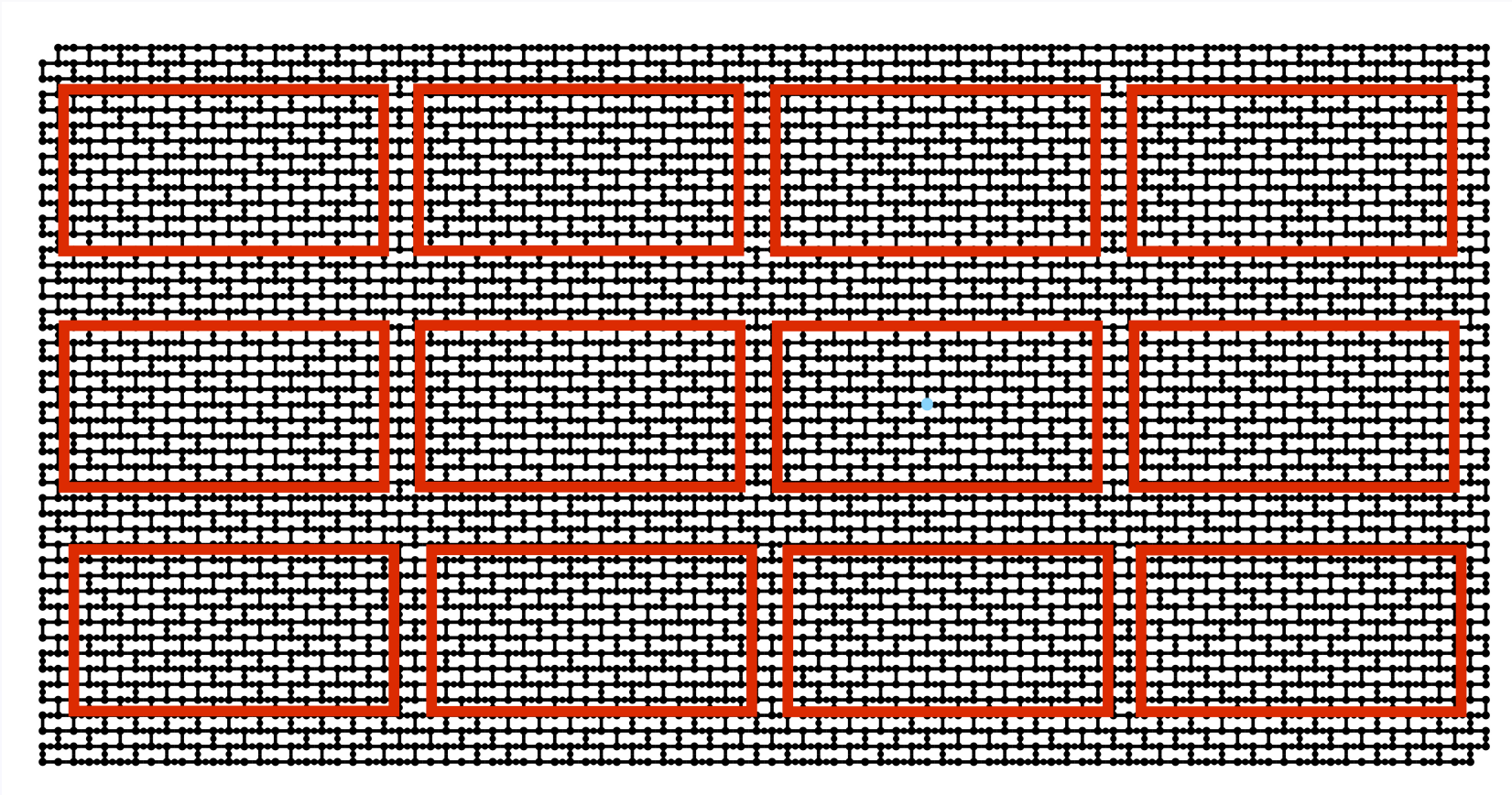


By repetitively applying this argument we find in G has as many disjoint odd cycles as its homocentric cycles that are $k+1$, a contradiction.

Therefore none of the k disjoint odd cycles crosses Ω. Thus x is irrelevant.



To find the irrelevant vertex x can be done in polynomial time!



General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

2: Otherwise check whether we have **VICTORY** (all k subwalls are non-bipartite)!

General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

2: Otherwise check whether we have **VICTORY** (all k subwalls are non-bipartite)!

3: Otherwise **FIGHT**: find an irrelevant vertex x, set $G \leftarrow G \setminus x$ and go to **1**.

General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

2: Otherwise check whether we have **VICTORY** (all k subwalls are non-bipartite)!

3: Otherwise **FIGHT**: find an irrelevant vertex x, set $G \leftarrow G \setminus x$ and go to **1**.

The above proves that p-Planar Odd Cycle Packing$\in 2^{O(k^{3/2})}$-FPT

General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

2: Otherwise check whether we have **VICTORY** (all k subwalls are non-bipartite)!

3: Otherwise **FIGHT**: find an irrelevant vertex x, set $G \leftarrow G \setminus x$ and go to **1**.

The above proves that p-Planar Odd Cycle Packing$\in 2^{O(k^{3/2})}$-FPT

- All the above arguments extend for graphs of bounded genus! (and futher!)

General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

2: Otherwise check whether we have **VICTORY** (all k subwalls are non-bipartite)!

3: Otherwise **FIGHT**: find an irrelevant vertex x, set $G \leftarrow G \setminus x$ and go to **1**.

The above proves that p-Planar Odd Cycle Packing$\in 2^{O(k^{3/2})}$-FPT

- All the above arguments extend for graphs of bounded genus! (and futher!)

The general p-Odd Cycle Packing problem is in FPT

[Kawarabayashi, Reed, STOC 2010]

General scheme

1. If G has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\mathbf{tw}(G))} \cdot n$ steps.

2: Otherwise check whether we have **VICTORY** (all k subwalls are non-bipartite)!

3: Otherwise **FIGHT**: find an irrelevant vertex x, set $G \leftarrow G \setminus x$ and go to **1**.

The above proves that p-Planar Odd Cycle Packing$\in 2^{O(k^{3/2})}$-FPT

- All the above arguments extend for graphs of bounded genus! (and futher!)

The general p-Odd Cycle Packing problem is in FPT

[Kawarabayashi, Reed, STOC 2010]

- The same ideas prove: p-Planar Odd Induced Cycle Packing$\in 2^{O(k^{3/2})}$-FPT

while general p-Odd Induced Cycle Packing problem is para-NP-hard.

[Golovach, Kamiński, Paulusma, Thilikos, TCS 2012]

Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

p-Minor Containment

Instance: two graphs G and H.

Parameter: $k = |V(H)|$

Question: $H \leq G$?

p-Disjoint Paths

Instance: A graph G and a sequence of pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$.

Parameter: k.

Question: Are there k pairwise vertex disjoint paths $P_1, \ldots, P_k$ in G such that for every $i \in \{1, \ldots, k\}$, P_i has endpoints s_i and t_i?

Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

p-Minor Containment

Instance: two graphs G and H.

Parameter: $k = |V(H)|$

Question: $H \leq G$?

p-Disjoint Paths

Instance: A graph G and a sequence of pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$.

Parameter: k.

Question: Are there k pairwise vertex disjoint paths $P_1, \ldots, P_k$ in G such that for every $i \in \{1, \ldots, k\}$, P_i has endpoints s_i and t_i?

Challenge: go further than planar graphs.

Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

p-Minor Containment

Instance: two graphs G and H.

Parameter: $k = |V(H)|$

Question: $H \leq G$?

p-Disjoint Paths

Instance: A graph G and a sequence of pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$.

Parameter: k.

Question: Are there k pairwise vertex disjoint paths $P_1, \ldots, P_k$ in G such that for every $i \in \{1, \ldots, k\}$, P_i has endpoints s_i and t_i?

Challenge: go further than planar graphs.

- Further than embedded graphs:

Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

p-Minor Containment

Instance: two graphs G and H.

Parameter: $k = |V(H)|$

Question: $H \leq G$?

p-Disjoint Paths

Instance: A graph G and a sequence of pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$.

Parameter: k.

Question: Are there k pairwise vertex disjoint paths $P_1, \ldots, P_k$ in G such that for every $i \in \{1, \ldots, k\}$, P_i has endpoints s_i and t_i?

Challenge: go further than planar graphs.

- Further than embedded graphs: a bigger story.

Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

p-Minor Containment

Instance: two graphs G and H.

Parameter: $k = |V(H)|$

Question: $H \leq G$?

p-Disjoint Paths

Instance: A graph G and a sequence of pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$.

Parameter: k.

Question: Are there k pairwise vertex disjoint paths $P_1, \ldots, P_k$ in G such that for every $i \in \{1, \ldots, k\}$, P_i has endpoints s_i and t_i?

Challenge: go further than planar graphs.

- Further than embedded graphs: a bigger story. Need another school to explain!

Merci beaucoup!