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)
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

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

\[ \forall n > N \quad f(n) \leq C \cdot g(n). \]
\textit{O-notation}

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]
\[ \forall n > N \quad f(n) \leq C \cdot g(n). \]

\textbf{Examples:}

\[ 2016 = O(1) = O(3): \text{ Constant} \]
**O-notation**

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]
\[ \forall n > N \quad f(n) \leq C \cdot g(n). \]

▶ **Examples:**

\[ 2016 = O(1) = O(3): \text{Constant} \]

\[ 4 \log_4 n + 5 \log_3 n = O(\log n): \text{Logarithmic} \]
\textit{O-notation}

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

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

\begin{itemize}
  \item \textbf{Examples:}
  \item \[2016 = O(1) = O(3): \text{Constant}\]
  \item \[4 \log_4 n + 5 \log_3 n = O(\log n): \text{Logarithmic}\]
  \item \[23 \log^5 n = O(\log^5 n) = \log^{O(1)} n: \text{polylogarithmic}\]
\end{itemize}
**$O$-notation**

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]
\[
\forall n > N \ f(n) \leq C \cdot g(n).
\]

▶ **Examples:**

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

\[ 4 \log_4 n + 5 \log_3 n = O(\log n): \text{Logarithmic} \]

\[ 23 \log^5 n = O(\log^5 n) = \log^{O(1)} n: \text{polylogarithmic} \]

\[ 2016\sqrt{n} + 3016 = O(\sqrt{n}): \text{sublinear} \]
**O-notation**

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

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

▶ **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

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

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

► 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**

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

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

▶ **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
\textbf{O-notation}

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]
\[ \forall n > N \ f(n) \leq C \cdot g(n). \]

▶ **Examples:**

\[ 2016 = O(1) = O(3): \text{ Constant} \]
\[ 4 \log_4 n + 5 \log_3 n = O(\log n): \text{ Logarithmic} \]
\[ 23 \log^5 n = O(\log^5 n) = \log^{O(1)} n: \text{ polylogarithmic} \]
\[ 2016 \sqrt{n} + 3016 = O(\sqrt{n}): \text{ sublinear} \]
\[ 2016n + 3016 = O(n): \text{ linear} \]
\[ 23n^2 + 16n + 4 = O(n^2): \text{ quadratic} \]
\[ 89n^2 + 6n^2 - 4n + 45676 = O(n^3): \text{ cubic} \]
\[ 5n^c = O(n^c) = n^{O(1)}: \text{ polynomial} \]
**O-notation**

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

\[ \forall n > N \ f(n) \leq C \cdot g(n). \]

▶ **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

\[ f(n) = O(g(n)) \iff \text{there exist constants } N \text{ and } C \text{ such that} \]

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

▶ Examples:

\[ 2016 = O(1) = O(3): \text{Constant} \]

\[ 4 \log_4 n + 5 \log_3 n = O(\log n): \text{Logarithmic} \]

\[ 23 \log^5 n = O(\log^5 n) = \log^{O(1)} n: \text{polylogarithmic} \]

\[ 2016\sqrt{n} + 3016 = O(\sqrt{n}): \text{sublinear} \]

\[ 2016n + 3016 = O(n): \text{linear} \]

\[ 23n^2 + 16n + 4 = O(n^2): \text{quadratic} \]

\[ 89n^2 + 6n^2 - 4n + 45676 = O(n^3): \text{cubic} \]

\[ 5n^c = O(n^c) = n^{O(1)}: \text{polynomial} \]

\[ n^{3.4} \cdot 3\sqrt{n} + n^{2016} = 2^{O(\sqrt{n})}: \text{subexponential} \]

\[ 30^{7n+3} = 2^{O(n)}: \text{singly exponential} \]
Problems and algorithms

**Definition** of an problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where $\Sigma$ 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 $\Sigma$ 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 \( \Sigma \) 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 $\Sigma$ 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!
Definition of a problem:

A set of YES-instances $\Pi \subseteq \Sigma^*$ where $\Sigma$ 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 \( G \), \( G_{\text{all}} \): the set of all graphs

A problem on graphs:

**Vertex Cover**

**Instance:** A graph \( G \) and an integer \( k \geq 0 \).

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

Here:

- **Input:** \( x = \langle G, k \rangle \), i.e., \( x \) encodes the graph \( G \) and the integer \( k \).
- **Problem:** \( \Pi_{\text{vc}} = \{ \langle G, k \rangle | G \text{ has a vertex cover of size } \leq k \} \)

Algorithm for **Vertex Cover**: a procedure that receives as input \( x = \langle G, k \rangle \) and outputs whether \( x \in \Pi_{\text{vc}} \).
Graphs

We mostly work on algorithms on graphs:

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

A problem on graphs:

**Vertex Cover**

*Instance:* A graph \( G \) and an integer \( k \geq 0 \).

*Question:* \( \exists S \in V(G) : |S| \leq k \land \forall e \in E(G) \ |e \cap S| \geq 1 \)?
Graphs

We mostly work on algorithms on graphs:

- $V(G)$: vertices of $G$
- $E(G)$: edges of $G$
- $G_{all}$: the set of all graphs

A problem on graphs:

**Vertex Cover**

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

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

We mostly work on algorithms on graphs:

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

A problem on graphs:

**Vertex Cover**

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

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

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 \( G \), \( G_{\text{all}} \): the set of all graphs

A problem on graphs:

**Vertex Cover**

*Instance:* A graph \( G \) and an integer \( k \geq 0 \).

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

Here:

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

**Problem:** \( \Pi_{\text{cv}} = \{ \langle G, k \rangle \mid G \text{ has a vertex cover of size } \leq k \} \)
Graphs

We mostly work on algorithms on graphs:

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

A problem on graphs:

**Vertex Cover**

*Instance*: A graph \( G \) and an integer \( k \geq 0 \).

*Question*: \( \exists S \in V(G) : |S| \leq k \land \forall e \in E(G) \, |e \cap S| \geq 1 \)?

Here:

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

**Problem**: \( \Pi_{vc} = \{ \langle G, k \rangle \mid G \text{ has a vertex cover of size } \leq k \} \)

**Algorithm for Vertex Cover**: a procedure than receives as input \( x = \langle G, k \rangle \) and outputs whether \( x \in \Pi_{vc} \)
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.

Most graphs in this lecture 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 a function of \( n = |V(G)| \).
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 for graphs: Adjacency list, Adjacency matrix

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

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

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 for graphs: Adjacency list, Adjacency matrix

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

Vertex Cover

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

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

$S$ 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: $P \subseteq NP$

$-question: is it correct that $P \neq NP$?

A problem $\Pi$ is NP-hard if it is "as hard as" all problems in NP

A problem $\Pi$ 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: $\exists S \in V(G): |S| \leq k \land \forall e \in E(G) |e \cap S| \geq 1$?

$S$ 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: $P \subseteq NP$

$-question: is it correct that $P \neq NP$?

A problem $\Pi$ is NP-hard if it is “as hard as” all problems in NP.

A problem $\Pi$ 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:** $\exists S \in V(G) : |S| \leq k \land \forall e \in E(G) \ |e \cap S| \geq 1$ ?
Complexity classes

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

\textbf{NP}: contains all problems that can be “certified” in \(n^{O(1)}\) steps

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\textbf{Vertex Cover} \\
\hline
\textit{Instance}: A graph \(G\) and an integer \(k \geq 0\). \\
\textit{Question}: \(\exists S \in V(G) : |S| \leq k \end{array} \quad \forall e \in E(G) \quad |e \cap S| \geq 1\end{array} \quad ?\end{array}
\end{tabular}
\end{table}

\(S\) 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

<table>
<thead>
<tr>
<th><strong>Vertex Cover</strong></th>
</tr>
</thead>
</table>
| **Instance:** A graph \( G \) and an integer \( k \geq 0 \).
| **Question:** \( \exists S \in V(G) : |S| \leq k \land \forall e \in E(G) \ |e \cap S| \geq 1 \) ? |

\( S \) 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**: \( \exists S \in V(G) : |S| \leq k \land \forall e \in E(G) \ |e \cap S| \geq 1 \)?

\( S \) 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

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

\( \mathbf{NP} \): 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:* \( \exists S \in V(G) : |S| \leq k \land \forall e \in E(G) |e \cap S| \geq 1 \) ?

\( S \) 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: \( \mathbf{P} \subseteq \mathbf{NP} \)

10\(^6\)$-question: is it correct that \( \mathbf{P} \neq \mathbf{NP} \)?
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:* $\exists S \in V(G) : \vert S \vert \leq k \land \forall e \in E(G) \vert e \cap S \vert \geq 1$?

$S$ 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: $P \subseteq NP$

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

A problem $\Pi$ 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:* \( \exists S \in V(G) : |S| \leq k \land \forall e \in E(G) |e \cap S| \geq 1 \) ?

\( S \) 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: \( P \subseteq NP \)

10^6$-question: is it correct that \( P \neq NP \)?

A problem \( \Pi \) is **NP-hard** if it is “as hard as” all problems in \( NP \)

A problem \( \Pi \) 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

<table>
<thead>
<tr>
<th><strong>Vertex Cover</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> A graph $G$ and an integer $k \geq 0$.</td>
</tr>
<tr>
<td><strong>Question:</strong> $\exists S \in V(G) :</td>
</tr>
</tbody>
</table>

$S$ 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: $P \subseteq NP$  

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

A problem $\Pi$ is **NP-hard** if it is “as hard as” all problems in $NP$.

A problem $\Pi$ 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

<table>
<thead>
<tr>
<th>VERTEX COVER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance</strong>: A graph $G$ and an integer $k \geq 0$.</td>
</tr>
<tr>
<td><strong>Question</strong>: $\exists S \in V(G) :</td>
</tr>
</tbody>
</table>

$S$ 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: $P \subseteq NP$  

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

A problem $\Pi$ is **NP-hard** if it is “as hard as” all problems in $NP$.

A problem $\Pi$ 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:* $\exists \sigma : V(G) \rightarrow \{1, \ldots, k\}: \forall \{v, u\} \in E(G) \ \sigma(v) \neq \sigma(u)$?
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:* $\exists S \in V(G) : |S| \geq k \land \forall e \in E(G) \ |e \cap S| \leq 1$?
Three NP-complete problems

**Independent Set**

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

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

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:* $\exists S \in V(G) : |S| \leq k \land \forall e \in E(G) \ |e \cap S| \geq 1$?
Three NP-complete problems

**Vertex Cover**

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

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

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

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:**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vertex Coloring</strong></td>
<td>$O(n^2 \cdot k^n)$</td>
</tr>
<tr>
<td><strong>Independent Set</strong></td>
<td>$O(n^{k+1})$</td>
</tr>
<tr>
<td><strong>Vertex Cover</strong></td>
<td>$O(2^k \cdot n)$</td>
</tr>
</tbody>
</table>

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

**Summary:**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Coloring</td>
<td>$O(n^2 \cdot k^n)$</td>
<td></td>
</tr>
<tr>
<td>Independent Set</td>
<td>$O(n^{k+1})$</td>
<td></td>
</tr>
<tr>
<td>Vertex Cover</td>
<td>$O(2^k \cdot n)$</td>
<td>good</td>
</tr>
</tbody>
</table>

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

Summary:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Coloring</td>
<td>$O(n^2 \cdot k^n)$</td>
<td></td>
</tr>
<tr>
<td>Independent Set</td>
<td>$O(n^{k+1})$</td>
<td>bad</td>
</tr>
<tr>
<td>Vertex Cover</td>
<td>$O(2^k \cdot n)$</td>
<td>good</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>$O(n^2 \cdot k^n)$</th>
<th>ugly</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vertex Coloring</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Independent Set</strong></td>
<td>$O(n^{k+1})$</td>
<td>bad</td>
</tr>
<tr>
<td><strong>Vertex Cover</strong></td>
<td>$O(2^k \cdot n)$</td>
<td>good</td>
</tr>
</tbody>
</table>

Different **Interleavings** between the parameter $k$ and the main part $n$ of the input.
Comparison between $O(2^k \cdot n)$ and $O(n^{k+1})$

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

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 algorithms.
**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.
**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.
Given an alphabet $\Sigma$,
Parameterized problems

Given an alphabet $\Sigma$,

(1) A parameterization of $\Sigma^*$ is a recursive function $\kappa : \Sigma^* \rightarrow \mathbb{N}$
Given an alphabet $\Sigma$,

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

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

\[ \kappa(G, k) = \text{genus}(G) \]

\[ \kappa(G, k) = \Delta(G) + k \]
Some parameterized problems

$p$-Dominating Set

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

*Parameter:* $k$

*Question:*

$$\exists S \in V(G) : |S| \leq k \land \forall v \in V(G) - S \exists u \in S \{v, u\} \in E(G)?$$
Some parameterized problems

\begin{center}
\textbf{p-Path}
\end{center}

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

\textit{Parameter:} $k$

\textit{Question:} Does $G$ contain a path of length $k$?
Some parameterized problems

\textit{p-Clique}

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

\textbf{Parameter}: $k$,

\textbf{Question}: $\exists S \in V(G) : \vert S \vert \leq k \land \forall v, u \in S \{v, u\} \in E(G)$?
More parameterized problems

\begin{center}
\textbf{p-Steiner Tree}
\end{center}

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

\textit{Parameter:} \( k \)

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

Here \( \kappa(G, S, k) = k \)
More parameterized problems

\psteiner_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 \( \kappa(G, S, k) = k \)

\pprime_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 $\Sigma$ and a parameterization $\kappa : \Sigma^* \rightarrow \mathbb{N}$,
The class FPT

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

(a) An algorithm $A$ is a FPT-algorithm with respect to $\kappa$ if there is a function computable $f : \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial function $p : \mathbb{N} \rightarrow \mathbb{N}$ 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 $\Sigma$ and a parameterization $\kappa : \Sigma^* \to \mathbb{N},$

(a) An algorithm $A$ is a FPT-algorithm with respect to $\kappa$ 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 $(L, \kappa)$ is fixed parameter tractable if there exists an FPT-algorithm with respect to $\kappa$ that decides $L$. 
The class FPT

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

(a) An algorithm $A$ is a FPT-algorithm with respect to $\kappa$ 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

$$\leq f(\kappa(x)) \cdot p(|x|)$$

steps.

(b) A parameterized problem $(L, \kappa)$ is fixed parameter tractable if there exists an FPT-algorithm with respect to $\kappa$ that decides $L$.

We then say that $(L, \kappa) \in \text{FPT}$ or, more precisely, $f$-FPT.
The class FPT

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

(a) An algorithm $A$ is a FPT-algorithm with respect to $\kappa$ 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 $(L, \kappa)$ is fixed parameter tractable if there exists an FPT-algorithm with respect to $\kappa$ that decides $L$.

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

▶ The function $f$ is called parameterized dependence of the running time of the FPT-algorithm
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]

$$\text{algvc}(G, k)$$

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 $\text{algvc}(G - v, k - 1) \lor \text{algvc}(G - u, k - 1)$

Therefore, $p$-Vertex Cover $\in 2^{O(k)}$-FPT.
An algorithm for **Vertex Cover:**

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

*[Bounded Search Tree Method]*

\[
\text{algvc}(G, k)
\]

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 \( \text{algvc}(G - v, k - 1) \cup \text{algvc}(G - u, k - 1) \)

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]

\[
\text{algvc}(G, k)
\]

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 $\text{algvc}(G - v, k - 1) \cup \text{algvc}(G - u, k - 1)$

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]

\[
\text{algvc}(G, k)
\]

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 \text{algvc}(G - v, k - 1) \lor \text{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

- para-NP
- W[SAT]
- W[1]
- W[2]
- W[P]
- FPT

- 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
Panorama of Parameterized complexity classes

- para-NP
- \(W[1]\)
- \(W[2]\)
- FPT
- XP

\(p\)-Vertex Cover: FPT
Panorama of Parameterized complexity classes

- $p$-Vertex Cover: FPT
- $p$-Path: FPT
Panorama of Parameterized complexity classes

\begin{itemize}
  \item \textit{p-Vertex Cover}: FPT
  \item \textit{p-Path}: FPT
  \item \textit{p'-Steiner Tree}: FPT
\end{itemize}
Panorama of Parameterized complexity classes

\[ \text{para-NP} \subseteq W[P] \subseteq W[\text{SAT}] \subseteq \ldots \subseteq W[2] \subseteq W[1] \subseteq \text{FPT} \subseteq \text{XP} \subseteq \text{para-NP} \]

\text{p-Vertex Cover: FPT}
\text{p-Path: FPT}
\text{p'-Steiner Tree: FPT}
\text{p-Clique: W[1]-complete}
Panorama of Parameterized complexity classes

- **para-NP**
- **XP**
- **W[1]**
- **W[2]**
- **W[3]**
- **FPT**

- **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
Panorama of Parameterized complexity classes

\[ \text{para-NP} \quad \text{XP} \]

\[ \text{W}[P] \quad \text{W}[2] \quad \text{W}[1] \quad \text{FPT} \]

- \text{p-Vertex Cover: FPT}
- \text{p-Path: FPT}
- \text{p'-Steiner Tree: FPT}
- \text{p-Clique: W[1]-complete}
- \text{p-Independent Set: W[1]-complete}
- \text{p-Dominating Set: W[2]-complete}
- \text{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
$p$-Steiner Tree: $W[2]$-complete
$p$-Coloring: para-NP-complete
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:

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 $T$.

- $X_t$ 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 $G$. 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 $D = (T, X)$ such that $T$ is a tree and $X = \{X_t \mid t \in V(T)\}$ 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 $T$.
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 $\max_{t \in V(T)} |X_t| - 1$
A \textit{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 \textit{node/bag} of $D$

- The \textit{width} of a tree decomposition $(T, \mathcal{X})$ is $\max_{t \in V(T)} |X_t| - 1$

- The \textit{tree-width} (ou largeur arborescente ou largeur d’arbre) of a graph $G$ ($\text{tw}(G)$) is the \textit{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 \textit{continuous} “trace” in the tree of the tree decomposition
Another definition for Treewidth

- A vertex in $G$ is \textit{k-simplicial} if its neighborhood induces a \textit{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

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

- 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.
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.
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

\begin{flushleft}
\textbf{p-Treewidth} \\
\textbf{Instance:} A graph $G$ and an integer $k \geq 0$. \\
\textbf{Parameter:} $k$ \\
\textbf{Question:} $\text{tw}(G) \leq k$?
\end{flushleft}
Parameterizing treewidth

\[ p\text{-Treewidth} \]

*Instance:* A graph \( G \) and an integer \( k \geq 0 \).

*Parameter:* \( k \)

*Question:* \( \text{tw}(G) \leq k? \)

\( p\text{-Treewidth} \) is in \( \text{FPT} \) 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
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:** $x = y$, $x \in X$, $\{x, y\} \in E$ ($E(x, y)$)
- **Formulae:** $\neg x$, $x \lor y$, $x \land y$, $x \rightarrow y$, $x \leftrightarrow y$, $\exists x \phi$, $\forall x \phi$, $\exists X \phi$, $\forall X \phi$, $\exists X \phi$, $\forall X \phi$, $\exists X \phi$, $\forall X \phi$,
Examples of properties expressible in MSO

3-Colorability:
\[\exists R \exists G \exists B [\forall x [(x \in R \lor x \in G \lor x \in B) \land
\neg(x \in R \land x \in G) \land \neg(x \in B \land x \in G) \land \neg(x \in R \land x \in B)]\]
\land \neg[\exists x \exists y \{(x, y) \in E \land
((x \in R \land y \in R) \lor (x \in G \land y \in G) \lor (x \in B \land y \in B))]\]}
Examples of properties expressible in MSO

Having an clique of size $\geq k$:

$$\exists x_1 \exists x_2 \cdots \exists x_k \wedge_{1 \leq i < j \leq k} \{x_i, x_j\} \in E$$
Examples of properties expressible in MSO

Having an independent set of size $k$:

$$
\exists x_1 \exists x_2 \cdots \exists x_k \bigwedge_{1 \leq i < j \leq k} (\neg \{x_i, x_j\} \in E) \land \neg (x_i \neq x_j)
$$
Examples of properties expressible in MSO

Having a vertex cover of size \( k \):

\[
\exists x_1 \exists x_2 \cdots \exists x_k \ (\forall x \ \forall y \ \{x, y\} \in E \rightarrow (\bigvee_{1 \leq i \leq k} (x = x_i \lor y = x_i)))
\]
Examples of properties expressible in MSO

Having a dominating set of size $k$:

$$\exists x_1 \exists x_2 \cdots \exists x_k \forall y \bigvee_{1 \leq i \leq k} (\{x_i, y\} \in E \lor y = x_i)$$
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 \( \varphi \) can be solved in \( f(tw(G), |\varphi|) \cdot n \) steps.

In other words: If \( \Pi \subseteq G \) all is a MSO-expressible set, then \( (\Pi, tw) \in FPT \) or every MSO-expressible problem of graphs is fixed parameter tractable when parameterized by the treewidth of its input graph.
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 $\phi$ can be solved in $f(\text{tw}(G), |\phi|) \cdot n$ 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 $\phi$ can be solved in $f(\text{tw}(G), |\phi|) \cdot n$ 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 $\phi$ can be solved in $f(\text{tw}(G), |\phi|) \cdot n$ steps.

In other words:

If $\Pi \subseteq G_{all}$ is a **MSO**-expressible set, then $(\Pi, \text{tw}) \in \text{FPT}$
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 \( \phi \) can be solved in \( f(\text{tw}(G), |\phi|) \cdot n \) steps.*

In other words:

If \( \Pi \subseteq G_{\text{all}} \) is a MSO-expressible set, then \((\Pi, \text{tw}) \in \text{FPT}\)

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 $G$. 

Inputs of small treewidth can be seen as tree-string: inputs of a tree-automaton generated by the MSO formula expressing $G$. 
Theorem: Every problem on graphs that can be expressed by a MSO formula $\phi$ can be solved in $f(tw(G), |\phi|) \cdot n$ steps.
Theorem: Every problem on graphs that can be expressed by a MSO formula $\phi$ can be solved in $f(\text{tw}(G), |\phi|) \cdot n$ 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 $\phi$ can be solved in $f(\text{tw}(G), |\phi|) \cdot n$ 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 as 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
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 $D = (T, X)$ 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 \((T, \mathcal{A})\) 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.

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.
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:* $k = \text{tw}(G)$

*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 $\chi : S \rightarrow I$ 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 $\phi : X_t \rightarrow \{1, 2, 3\}$, we define

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

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

$G = G_r$ is 3-colourable iff $B_r(\emptyset) = 1$
2nd step: tables for leaf nodes:

Let $X_l$ be an leaf node

we have

$$B_l(\emptyset) = 1$$
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 $\phi : X_t \rightarrow \{1, 2, 3\}$, we have

$$B_t(\phi) = B_{t'}(\phi - (v, \phi(v))) \bigwedge_{u \in N_{G_t}(v)} [\phi(v) \neq \phi(u)]$$
4th 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 a join node

let $t_1, t_2$ be the children of $t$

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

$$B_t(\phi) = B_{t_1}(\phi) \land B_{t_2}(\phi)$$
Conclusion:

Given a tree decomposition of $G$, the following $tw$-$3$-$\text{ VERTEX-COLORING}$ 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:* $k = tw(G)$

*Question:* does $G$ contain a spanning cycle?
A pairing of vertices in $X_t$ is a graph $H$ (with loops) such that $V(G) = X_t$ and $\forall x \in X_t \deg_H(x) \leq 2$.

The restriction of a cycle to $G_t$ is a collection $P$ of internally disjoint paths in $G_t$ with ends in $X_t$. Each $P$ corresponds to some pairing $H_P$ of $X_t$.

For any set $S$, let $\text{pairs}(S)$ be the set of all pairings of $S$. 

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

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 vertices in $X_t$ is a graph $H$ (with loops) s.t. $V(G) = X_i$ and $\forall x \in X_i \deg_H(x) \leq 2$

- 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 vertices in $X_t$ is a graph $H$ (with loops) s.t. $V(G) = X_i$ and $\forall x \in X_i \; \deg_H(x) \leq 2$

- 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 $\text{pairs}(S)$ 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:

\[
\forall H \in \text{pairs}(X_i), \quad B_t(H) = [H \text{ is the pairing of some } t\text{-path collection } \mathcal{P}]
\]

\(G = G_r\) has a Hamiltonian cycle iff \(B_r(H_w) = 1\)
2nd step: tables for leaf nodes:

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

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

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

\[ \forall H \in \text{pairs}(t) \quad B_t(H) = [|E(H)| = 0] \]
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 $\forall H \in \text{pairs}(t)$ we have

$$B_t(H) = [B_{t'}(H - v)] \land [N_H(v) \subseteq N_{G_t}(v)]$$
4th step: tables for forget nodes:

Let $X_t$ be a forget node

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

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

$$B_t(H) = \bigvee_{H' \in \text{pairs}(t')} B_{t'}(H')$$

$H$ is a contraction of $H'$
5th step: tables for join nodes:

Let \( X_t \) be a join node

let \( t_1, t_2 \) be the children of \( t \)

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

\[
B_t(H) = \bigvee_{H_1 \in \text{pairs}(t_1)} B_{t_1}(H_1) \land B_{t_2}(H_2)
\]  

\[
H = H_1 \cup H_2
\]
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
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 \((T, \tau)\)
Branch decompositions

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

A *branch decomposition* is a pair \((T, \tau)\)

where

1. \(T\) is a ternary tree and
2. \(\tau\) 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 \((T, \tau)\)

where

1. \(T\) is a ternary tree and
2. \(\tau\) 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 \(\text{mid}(e) = \partial E_e\).
A graph $G$ and a branch decomposition of it.

The width of a branch decomposition $(T, \tau)$ is $\max \{|\tau(e)| : e \in E(T)\}$.

The branchwidth $bw(G)$ of a graph $G$ is then the minimum width a branch decomposition of $G$ may have.
A graph $G$ and a branch decomposition of it.

The width of a branch decomposition $(T, \tau)$ is $\max\{|\text{mid}(e)| \mid e \in E(T)\}$.
A graph $G$ and a branch decomposition of it.

The \textit{width} of a branch decomposition $(T, \tau)$ is \( \max\{\mid\text{mid}(e)\mid \mid e \in E(T)\} \)

The \textit{branchwidth}, \(\text{bw}(G)\), of a graph $G$ is then \textit{minimum} width a branch decomposition of $G$ may have.
Theorem:

Robertson and Seymour, GM-10

If $G$ is not acyclic, then $bw(G) \leq tw(G) + 1 \leq \frac{3}{2} bw(G)$

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

$tw(K_6) = bw(K_6) = 6$

$bw(K_6) = 4 < tw(K_6) = 5$
Theorem: [Robertson and Seymour, GM-10] If $G$ is not acyclic, then

$$bw(G) \leq tw(G) + 1 \leq \frac{3}{2} bw(G)$$
Theorem: [Robertson and Seymour, GM-10] If $G$ is not acyclic, then
\[ \text{bw}(G) \leq \text{tw}(G) + 1 \leq \frac{3}{2} \text{bw}(G) \]

If $T$ is a tree, then $0 \leq \text{bw}(G) \leq 2$.
\[ \text{tw}(\begin{array}{c} \text{\Tiny grid} \end{array}) = \text{bw}(\begin{array}{c} \text{\Tiny grid} \end{array}) = 6 \]
\[ \text{bw}(K_6) = 4 < \text{tw}(K_6) = 5 \]
Dynamic programming for graphs of small branchwidth
Given a branch decomposition \((T, \tau)\), (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 below \(e\).
Given a branch decomposition \((T, \tau)\), (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 below \(e\).
2. Define, for each $e \in E(T)$, a table encoding the information of a partial solution for $G_e$ as restricted to $\text{mid}(e)$. 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

\[ \text{mid}(e) \]

\[ G_{e_1} \]

\[ G_{e_2} \]
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 \( \text{vc}(G, X, X') \leq k \) 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 $\text{vc}(G, X, X') \leq k$ 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 $\text{vc}(G, X, X') \leq k$ if $G$ contains a vertex cover $S$ where $|S| \leq k$ and $X \subseteq S \subseteq V(G) \setminus X'$. 

Let $R_e = \{(X, k) \mid X \subseteq \text{mid}(e) \land \text{vc}(G_e, X, \text{mid}(e) \setminus X) \leq k\}$
Let $G$ be a graph and $X, X' \subseteq V(G)$ where $X \cap X' = \emptyset$.

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

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

observe that $\text{vc}(G) \leq k$ iff $(\emptyset, k) \in \mathcal{R}_e$. 

Compute $R_e$ by using the following dynamic programming formula:
Compute $R_e$ by using the following dynamic programming formula:

$$R_e = \begin{cases} 
\{(X, k) \mid X \subseteq e \land X \neq \emptyset \land k \geq |X|\} & \text{if } e \in L(T) \\
\{(X, k) \mid X \subseteq \text{mid}(e) \land \exists (X_1, k_1) \in R_e_1, \exists (X_2, k_2) \in R_e_2 : (X_1 \cup X_2) \cap \text{mid}(e) = X \land k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \not\in L(T) 
\end{cases}$$
Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$
\mathcal{R}_e = \begin{cases} 
\{(X, k) \mid X \subseteq e \land X \neq \emptyset \land k \geq |X|\} & \text{if } e \in L(T) \\
\{(X, k) \mid X \subseteq \text{mid}(e) \land \exists (X_1, k_1) \in \mathcal{R}_{e_1}, \exists (X_2, k_2) \in \mathcal{R}_{e_2} : (X_1 \cup X_2) \cap \text{mid}(e) = X \land k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \not\in L(T)
\end{cases}
$$
Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$
\mathcal{R}_e = \begin{cases} 
\{(X, k) \mid X \subseteq e \land X \neq \emptyset \land k \geq |X|\} & \text{if } e \in L(T) \\
\{(X, k) \mid X \subseteq \text{mid}(e) \land \exists (X_1, k_1) \in \mathcal{R}_{e_1}, \exists (X_2, k_2) \in \mathcal{R}_{e_2} : \text{mid}(e) \} & \text{if } e \not\in L(T)
\end{cases}
$$

$\forall e \in E(T), |\mathcal{R}_e| \leq 2^{|\text{mid}(e)|} \cdot \ell.$
Compute $\mathcal{R}_e$ by using the following dynamic programming formula:

$$
\mathcal{R}_e = \begin{cases} 
\{(X, k) \mid X \subseteq e \land X \neq \emptyset \land k \geq |X|\} & \text{if } e \in L(T) \\
\{(X, k) \mid X \subseteq \text{mid}(e) \land \exists (X_1, k_1) \in \mathcal{R}_{e_1}, \exists (X_2, k_2) \in \mathcal{R}_{e_2} : (X_1 \cup X_2) \cap \text{mid}(e) = X \land k_1 + k_2 - |X_1 \cap X_2| \leq k\} & \text{if } e \not\in L(T) 
\end{cases}
$$

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

Suppose that $G$ is a planar graph embedded on the sphere $S_0$. 
Suppose that $G$ is a planar graph embedded on the sphere $S_0$.

A **sphere-cut decomposition** of $G$ is a branch decomposition $(T, \tau)$ where for any $e \in E(T)$, the vertices in $\text{mid}(e)$ are the vertices in a Jordan curve of $S_0$ – called a **noose** – that meets no edges of $G$. 
Sphere-Cut Decompositions

Suppose that $G$ is a planar graph embedded on the sphere $S_0$

A sphere-cut decomposition of $G$ is a branch decomposition $(T, \tau)$ where for any $e \in E(T)$, the vertices in $\text{mid}(e)$ are the vertices in a Jordan curve of $S_0$ – called noose – that meets no edges of $G$. 
Suppose that $G$ is a planar graph embedded on the sphere $S_0$.

A sphere-cut decomposition of $G$ is a branch decomposition $(T, \tau)$ where for any $e \in E(T)$, the vertices in $\text{mid}(e)$ are the vertices in a Jordan curve of $S_0$ — called a noose — that meets no edges of $G$. 
Sphere-Cut Decompositions

Suppose that $G$ is a planar graph embedded on the sphere $S_0$.

A sphere-cut decomposition of $G$ is a branch decomposition $(T, \tau)$ where for any $e \in E(T)$, the vertices in $\text{mid}(e)$ are the vertices in a Jordan curve of $S_0$ — called noose — that meets no edges of $G$. 

![Diagram of a planar graph and a sphere-cut decomposition]

1. $e_2$
2. $e_3$
3. $e_4$
4. $e_5$
5. $e_6$
6. $e_7$
7. $e_8$
Sphere-Cut Decompositions

Suppose that $G$ is a planar graph embedded on the sphere $S_0$.

A *sphere-cut decomposition* of $G$ is a branch decomposition $(T, \tau)$ where for any $e \in E(T)$, the vertices in $\text{mid}(e)$ are the vertices in a Jordan curve of $S_0$ — called a *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 can 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 can be constructed in $O(n^3)$ steps.

For doing dynamic programming on a sphere cut decomposition $(T, \tau)$ again we define, for any $e \in E(T)$ the set $\text{pairs}(\text{mid}(e))$ be the set of all pairings of $\text{mid}(e)$
**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 can be constructed in $O(n^3)$ steps.

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

The “usual” bound for $\text{mid}(e)$ is $2^{O(k \cdot \log k)}$ (recall that $|\text{mid}(e)| = \Omega(\frac{k}{2}!)$)
**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 can be constructed in $O(n^3)$ steps.

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

The “usual” bound for $\text{mid}(e)$ is $2^{O(k \cdot \log k)}$

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

However, we now have that

1: the vertices of $\text{mid}(e)$ 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 can be constructed in $O(n^3)$ steps.

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

The “usual” bound for $\text{mid}(e)$ is $2^{O(k \cdot \log k)}$

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

However, we now have that

1: the vertices of $\text{mid}(e)$ 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.
In case of Hamiltonian 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 Hamiltonian 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 Hamiltonian 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 **Hamiltonian Cycle**, each non-crossing pair on $O_P$ is the union of two non-crossing pairs on $O_L$ and $O_R$. 
It follows that \( \text{pairs}(\text{mid}(e)) = O(C(|\text{mid}(e)|)) = O(C(k)) \)

Where \( C(k) \) is the \( k \)-th Catalan Number.
Catalan Structures

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

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 $\text{pairs}(\text{mid}(e)) = O(C(|\text{mid}(e)||)) = O(C(k))$

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 pairs(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 \( \textbf{pairs}(\textbf{mid}(e)) \)
can be defined for controlling the size of the tables in dynamic programming.

\textbf{In general}: These are pairs where the tables encode pairings.
The same holds for several other problems where an analogue of \( \text{pairs}(\text{mid}(e)) \) 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 \( 2^{O(\text{tw}(G))} \cdot n^{O(1)} \) step algorithms for the planar versions of \textbf{Cycle Cover}, \textbf{Path Cover}, \textbf{Longest Path}, \textbf{Longest Cycle}, \textbf{Hamiltonian Cycle}, and \textbf{Graph Metric TSP} and others.

[Dorn, Penninkx, Bodlaender, and Fomin. ICALP 2005]
The same holds for several other problems where an analogue of $\text{pairs} (\text{mid} (e))$ 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 $2^{O(\text{tw}(G))} \cdot n^{O(1)}$ 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

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

[Sau, Ru´e, Thilikos, TALG 2014]

Extensions/alternatives:
▶ For H-minor free graphs: [Sau, Ru´e, Thilikos, COCOON 2012]
▶ Surface split decompositions: [Bonsma, STACS 2012]
▶ Brick Decompositions: [Cohen-Addad & de Mesmay, ESA 2015]
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

Extensions/alternatives:
▶ For H-minor free graphs: [Sau, Rue, Thilikos, COCOON 2012]
▶ Surface split decompositions: [Bonsma, STACS 2012]
▶ Brick Decompositions: [Cohen-Addad & de Mesmay, ESA 2015]
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 \textit{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]
Bidimensionality

Subexponential parameterized algorithms
The minor relation

\[ H \text{ is a minor of } G \ (H \leq G): \]

\[ H \text{ occurs from a subgraph of } G \text{ by applying edge contractions} \]
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 \( G \) is *contraction-closed* if:

every contraction of a graph in \( G \) is also a graph in \( G \)
Theorem: [Robertson & Seymour – main combinatorial result of GM] Every infinite set of graphs contains two graphs comparable under the minor relation.
\textbf{Theorem:} [Robertson & Seymour – main combinatorial result of GM] \textit{Every infinite set of graphs contains two graphs comparable under the minor relation.}

\begin{itemize}
  \item Equivalently: Graphs are \textit{Well Quasi Ordered} w.r.t. the \textit{minor} relation
\end{itemize}
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.

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

**Consequence of R&S theorem:**

$|\text{obs}(G)| < \aleph_0$ 

**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 $G$, the problem asking whether $G \in G$ belongs in PTIME, i.e., can be solved in $O(n^3)$ steps!
Let $G$ be a minor-closed graph class.

- $\text{obs}(G)$ is the set of minor-minimal elements not in $G$.
- If $G$ is minor-closed, then $G \in G \iff \forall H \in \text{obs}(G) \ H \not\in G$
Let \( G \) be a minor-closed graph class.

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

Consequence of R&S theorem: \( |\text{obs}(G)| < \aleph_0 \)
Let $G$ be a minor-closed graph class.

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

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

---

**Theorem:** [Robertson & Seymour – main algorithmic consequence of GM] For every $H$, checking whether $H \preceq G$ can be done in $O(n^3)$ steps.
Let $G$ be a minor-closed graph class.

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

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

---

**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 $G$, the problem asking whether $G \in G$ belongs in PTIME, i.e., can be solved in $O(n^3)$ steps!
Graph optimization parameters

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

We consider *minimization/maximization parameters* $p$ defined as follows

$$p(G) = \min \{ k | \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}$$

▶ Vertex Cover, $vc(G)$: $\min$, $\phi(G, S) = \forall e \in E(G), e \cap S \neq \emptyset$

▶ Dominating Set, $dc(G)$: $\min$, $\phi(G, S) = V(G) = N(G[S])$

▶ Longest Path, $pl(G)$: $\max$, $\phi(G, S) = G[S]$ is a path

▶ Scattered Set, $sc(G)$: $\max$, $\phi(G, S) = \forall x \in V(G), |N[x] \cap S| \leq 1$
Graph optimization parameters

Graph parameter: a function $p : G_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $p$ defined as follows

$$p(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true}\}$$

$$p(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \land \phi(G, S) = \text{true}\}$$

In any case, we call a set $S$ where $|S| = p(G)$ a solution certificate for $p(G)$. We call such parameters graph optimization parameters.

Three examples:

▶ **Vertex Cover**, $vc(G)$:

$$\min, \phi(G, S) = \forall e \in E(G) \ e \cap S \neq \emptyset$$

▶ **Dominating Set**, $dc(G)$:

$$\min, \phi(G, S) = V(G) = N_G(S)$$

▶ **Longest Path**, $pl(G)$:

$$\max, \phi(G, S) = G[S] \text{ is a path}$$

▶ **Scattered Set**, $sc(G)$:

$$\max, \phi(G, S) = \forall x \in V(G) \ |N_{G}(x) \cap S| \leq 1$$
Graph optimization parameters

Graph parameter: a function $p : G_{\text{all}} \to \mathbb{N}$

We consider *minimization/maximization parameters* $p$ defined as follows

$$p(G) = \min \{ k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}$$

$$p(G) = \max \{ k \mid \exists S \subseteq V(G) : |S| \geq k \land \phi(G, S) = \text{true} \}$$

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

We call such parameters *graph optimization* parameters.
Graph optimization parameters

Graph parameter: a function $p : G_{all} \rightarrow \mathbb{N}$

We consider *minimization/maximization parameters* $p$ defined as follows

$$p(G) = \min \{ k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}$$

$$p'(G) = \max \{ k \mid \exists S \subseteq V(G) : |S| \geq k \land \phi(G, S) = \text{true} \}$$

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

We call such parameters *graph optimization parameters*.

*Three examples:*

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

- **Dominating Set**, $\text{dc}(G)$: $\text{min}$, $\phi(G, S) = V(G) = \mathbb{N}_G(S)$

- **Longest Path**, $\text{pl}(G)$: $\text{max}$, $\phi(G, S) = G[S]$ is a path
Graph optimization parameters

Graph parameter: a function $p : G_{all} \rightarrow \mathbb{N}$

We consider *minimization/maximization parameters* $p$ defined as follows

$$p(G) = \min \{ k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}$$

$$p(G) = \max \{ k \mid \exists S \subseteq V(G) : |S| \geq k \land \phi(G, S) = \text{true} \}$$

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

We call such parameters *graph optimization* parameters.

**Three examples:**

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

- **Dominating Set**, $\text{dc}(G)$: $\min\ , \phi(G, S) = V(G) = N_G(S)$
Graph optimization parameters

Graph parameter: a function \( p : G_{all} \rightarrow \mathbb{N} \)

We consider \textit{minimization/maximization parameters} \( p \) defined as follows

\[
p(G) = \min \{ k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}
\]

\[
p(G) = \max \{ k \mid \exists S \subseteq V(G) : |S| \geq k \land \phi(G, S) = \text{true} \}
\]

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

We call such parameters \textit{graph optimization} parameters.

\textbf{Three examples:}

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

- \textbf{Dominating Set}, \( \text{dc}(G) \): min, \( \phi(G, S) = V(G) = N_G(S) \)

- \textbf{Longest Path}, \( \text{pl}(G) \): max, \( \phi(G, S) = G[S] \text{ is a path} \)
Graph optimization parameters

Graph parameter: a function \( p : G_{\text{all}} \rightarrow \mathbb{N} \)

We consider *minimization/maximization parameters* \( p \) defined as follows

\[
p(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}
\]

\[
p(G) = \max\{k \mid \exists S \subseteq V(G) : |S| \geq k \land \phi(G, S) = \text{true} \}
\]

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

We call such parameters *graph optimization* parameters.

Three examples:

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

- **Dominating Set**, \( \text{dc}(G) \): \( \min \), \( \phi(G, S) = V(G) = N_G(S) \)

- **Longest Path**, \( \text{pl}(G) \): \( \max \), \( \phi(G, S) = G[S] \text{ is a path} \)

- **Scattered Set**, \( \text{sc}(G) \): \( \max \), \( \phi(G, S) = \forall x \in V(G) \ |N[x] \cap S| \leq 1 \)
We say that \( p \) is \textit{minor closed} if \( H \preceq_m G \Rightarrow p(H) \leq p(G) \).
We say that $p$ is minor closed if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:
We say that $p$ is *minor closed* if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:

  - $vc$: minimum vertex cover of $G$

  - $fvs$: minimum feedback vertex set of $G$

  - $fc$: minimum face cover of a planar $G$

  - $lp$: maximum $k$ for which $G$ contain a $k$-path

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

  - $tw$: the tree-width of $G$

  - $bw$: the branch-width of $G$

  - $eg$: the Euler genus of $G$

  - $r$-twm: ($= \min \{ |S| \mid tw(G \setminus S) \leq r \}$)
We say that $p$ is *minor closed* if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:
  
  - $vc$: minimum vertex cover of $G$
  - $fvs$: minimum feedback vertex set of $G$
We say that $p$ is \textit{minor closed} if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:
  - $\text{vc}$: minimum vertex cover of $G$
  - $\text{fvs}$: minimum feedback vertex set of $G$
  - $\text{fc}$: minimum face cover of a planar $G$
We say that \( p \) is minor closed if \( H \leq_m G \Rightarrow p(H) \leq p(G) \).

- Examples of minor-closed graph parameters:

  - \( \text{vc} \): minimum vertex cover of \( G \)
  - \( \text{fvs} \): minimum feedback vertex set of \( G \)
  - \( \text{fc} \): minimum face cover of a planar \( G \)
  - \( \text{lp} \): maximum \( k \) for which \( G \) contain a \( k \)-path
We say that $p$ is \textit{minor closed} if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:

  - $vc$: minimum vertex cover of $G$
  - $fvs$: minimum feedback vertex set of $G$
  - $fc$: minimum face cover of a planar $G$
  - $lp$: maximum $k$ for which $G$ contain a $k$-path
  - $vp$: vertex planarizer number of $G$ ($= \min\{|S| \mid G \setminus S \text{ is planar}\}$)
  - $tw$: the tree-width of $G$
  - $bw$: the branch-width of $G$
  - $eg$: the Euler genus of $G$
  - $r\cdot twm$: ($= \min\{|S| \mid tw(G \setminus S) \leq r\}$)
We say that $p$ is *minor closed* if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:

  - $vc$: minimum vertex cover of $G$
  - $fvs$: minimum feedback vertex set of $G$
  - $fc$: minimum face cover of a planar $G$
  - $lp$: maximum $k$ for which $G$ contain a $k$-path
  - $vp$: vertex planarizer number of $G$  \( (= \min\{|S| \mid G \setminus S \text{ is planar}\}) \)
  - $tw$: the tree-width of $G$
We say that \( p \) is minor closed if \( H \leq_m G \Rightarrow p(H) \leq p(G) \).

Examples of minor-closed graph parameters:

- \( \text{vc} \): minimum vertex cover of \( G \)
- \( \text{fvs} \): minimum feedback vertex set of \( G \)
- \( \text{fc} \): minimum face cover of a planar \( G \)
- \( \text{lp} \): maximum \( k \) for which \( G \) contain a \( k \)-path
- \( \text{vp} \): vertex planarizer number of \( G \) \( (= \min\{|S| \mid G \setminus S \text{ is planar}\}) \)
- \( \text{tw} \): the tree-width of \( G \)
- \( \text{bw} \): the branch-width of \( G \)
We say that $p$ is \textit{minor closed} if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:
  - $vc$: minimum vertex cover of $G$
  - $fvs$: minimum feedback vertex set of $G$
  - $fc$: minimum face cover of a planar $G$
  - $lp$: maximum $k$ for which $G$ contain a $k$-path
  - $vp$: vertex planarizer number of $G$ \ ($= \min\{|S| \mid G \setminus S \text{ is planar}\}$
  - $tw$: the tree-width of $G$
  - $bw$: the branch-width of $G$
  - $eg$: the Euler genus of $G$
We say that $p$ is *minor closed* if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:
  - $vc$: minimum vertex cover of $G$
  - $fvs$: minimum feedback vertex set of $G$
  - $fc$: minimum face cover of a planar $G$
  - $lp$: maximum $k$ for which $G$ contain a $k$-path
  - $vp$: vertex planarizer number of $G$ ($= \min\{|S| \mid G \setminus S \text{ is planar}\}$)
  - $tw$: the tree-width of $G$
  - $bw$: the branch-width of $G$
  - $eg$: the Euler genus of $G$
  - $r-tw_m$: ($= \min\{|S| \mid tw(G \setminus S) \leq r\}$)
We say that $p$ is minor closed if $H \leq_m G \Rightarrow p(H) \leq p(G)$.

- Examples of minor-closed graph parameters:
  
  - $vc$: minimum vertex cover of $G$
  - $fvs$: minimum feedback vertex set of $G$
  - $fc$: minimum face cover of a planar $G$
  - $lp$: maximum $k$ for which $G$ contain a $k$-path
  - $vp$: vertex planarizer number of $G$ \( (= \min\{|S| \mid G \setminus S \text{ is planar}\}) \)
  - $tw$: the tree-width of $G$
  - $bw$: the branch-width of $G$
  - $eg$: the Euler genus of $G$
  - $r-twm$: \( (= \min\{|S| \mid tw(G \setminus S) \leq r\}) \)
We say that $p$ is *contraction closed* if $H \leq_c G \Rightarrow p(H) \leq p(G)$. 

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

- $ds$: minimum dominating set of $G$
- $cd$: minimum cycle domination set of $G$
- $vc$: minimum connected vertex cover of $G$
- $ctw$: minimum connected treewidth of $G$—i.e., all bags induce connected subgraphs
- $icp$: maximum induced cycle packing
- $sc$: 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.
We say that \( p \) is *contraction closed* if \( H \leq_c G \Rightarrow p(H) \leq p(G) \).

- Examples of contraction-closed (but not minor-closed) graph parameters:
We say that $p$ is *contraction closed* if $H \leq_c G \Rightarrow p(H) \leq p(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:
  - $ds$: minimum dominating set of $G$
  - $cd$: minimum cycle domination set of $G$
  - $vc$: minimum connected vertex cover of $G$
  - $ctw$: minimum connected treewidth of $G$, i.e., all bags induce connected subgraphs
  - $icp$: maximum induced cycle packing
  - $sc$: 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.
We say that $p$ is \textit{contraction closed} if $H \leq_c G \Rightarrow p(H) \leq p(G)$.

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

  - $\text{ds}$: minimum dominating set of $G$
  - $\text{cd}$: minimum cycle domination set of $G$
We say that $p$ is *contraction closed* if $H \leq_c G \Rightarrow p(H) \leq p(G)$.

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

- $\text{ds}$: minimum dominating set of $G$
- $\text{cd}$: minimum cycle domination set of $G$
- $\text{vc}$: minimum connected vertex cover of $G$

$S$ is a scattered: no two distinct vertices of $S$ have a common neighbor.
We say that $p$ is *contraction closed* if $H \leq_c G \Rightarrow p(H) \leq p(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:
  
  - $ds$: minimum dominating set of $G$
  - $cd$: minimum cycle domination set of $G$
  - $vc$: minimum connected vertex cover of $G$
  - $ctw$: minimum connected treewidth of $G$ i.e., all bags induce *connected* subgraphs
  
  **$icp$**: maximum induced cycle packing
  
  **$sc$**: 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.)
We say that $p$ is *contraction closed* if $H \leq_c G \Rightarrow p(H) \leq p(G)$.

- Examples of contraction-closed (but not minor-closed) graph parameters:
  - $ds$: minimum dominating set of $G$
  - $cd$: minimum cycle domination set of $G$
  - $vc$: minimum connected vertex cover of $G$
  - $ctw$: minimum connected treewidth of $G$ i.e., all bags induce connected subgraphs
  - $icp$: maximum induced cycle packing
We say that $p$ is *contraction closed* if $H \leq_c G \Rightarrow p(H) \leq p(G)$.

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

- **ds**: minimum dominating set of $G$
- **cd**: minimum cycle domination set of $G$
- **vc**: minimum connected vertex cover of $G$
- **ctw**: minimum connected treewidth of $G$ i.e., all bags induce connected subgraphs
- **icp**: maximum induced cycle packing
- **sc**: maximum $k$ for which $G$ contains a scattered set of $k$ vertices.
We say that $p$ is \textit{contraction closed} if $H \leq_G G \Rightarrow p(H) \leq p(G)$.

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

- \textbf{ds}: minimum dominating set of $G$
- \textbf{cd}: minimum cycle domination set of $G$
- \textbf{vc}: minimum connected vertex cover of $G$
- \textbf{ctw}: minimum connected treewidth of $G$ i.e., all bags induce \textit{connected} subgraphs
- \textbf{icp}: maximum induced cycle packing
- \textbf{sc}: maximum $k$ for which $G$ contains a \textit{scattered} set of $k$ vertices.

$[S$ is a \textit{scattered}: no two distinct vertices of $S$ have a common neighbor.$]$
Let $\mathbf{p}$ be a minor- (contraction-) closed graph parameter.
Let $p$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem $p$-Checking Value of $p = (G_{all}, p)$.
Let \( p \) be a \textit{minor- (contraction-)} closed graph parameter.

We define the parameterized problem \( \text{\textit{p-Checking Value of}} \ p = (G_{\text{all}}, p) \).

In other words, \( \text{\textit{p-Checking Value of}} \ p \) corresponds to the following Meta-problem:

\textbf{Instance:} A graph \( G \) and an integer \( k \geq 0 \).

\textbf{Parameter:} \( k \)

\textbf{Question:} is it correct that \( p(G) \leq k \)?

Here “\( \leq \)” is \( \geq \) if \( p \) is a minimization/maximization parameter.

For simplicity: \( \Pi_p = \text{\textit{p-Checking Value of}} \ p \)

We call such a problem \( \Pi_p \)-subset optimization problem.

For every \( k \in \mathbb{N} \), we define the \( k \)-th layer of \( (G_{\text{all}}, p) \) as the class \( G_p^k = \{ G \mid p(G) \leq k \} \).

These are \textbf{YES}-instances for minimization problems and \textbf{NO}-instances for maximization problems.
Let $p$ be a minor- (contraction-) closed graph parameter. We define the parameterized problem $p$-Checking Value of $p = (G_{\text{all}}, p)$.

In other words, $p$-Checking Value of $p$ corresponds to the following Meta-problem:

<table>
<thead>
<tr>
<th><strong>p-Checking Value of $p$</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> A graph $G$ and an integer $k \geq 0$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $k$</td>
</tr>
<tr>
<td><strong>Question:</strong> is it correct that $p(G) \leq k$?</td>
</tr>
</tbody>
</table>

Here “$\leq$” = “$\leq$” / “$\geq$” = “$\geq$” if $p$ is a minimization/maximization parameter.
Let $p$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem $p$-Checking Value of $p$ = $(G_{\text{all}}, p)$.

In other words, $p$-Checking Value of $p$ corresponds to the following Meta-problem:

**Meta-problem:**

$p$-Checking Value of $p$

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

Parameter: $k$

Question: is it correct that $p(G) \leq k$?

Here $\leq$ means $\leq$ if $p$ is a minimization parameter and $\geq$ if $p$ is a maximization parameter.

For simplicity: $\Pi_p = p$-Checking Value of $p$.
Let $p$ be a minor- (contraction-) closed graph parameter.

We define the parameterized problem $p$-Checking Value of $p = (G_{all}, p)$

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

<table>
<thead>
<tr>
<th>$p$-Checking Value of $p$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> A graph $G$ and an integer $k \geq 0$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $k$</td>
</tr>
<tr>
<td><strong>Question:</strong> is it correct that $p(G) \leq k$?</td>
</tr>
</tbody>
</table>

Here "$\leq" = "\leq" / "$\geq" = "\geq"" if $p$ is a minimization/maximization parameter

For simplicity: $\Pi_p = p$-Checking Value of $p$

We call such a problem $\Pi_p$ *subset optimization problem*
Let \( p \) be a minor- (contraction-) closed graph parameter.

We define the parameterized problem \( p\text{-Checking Value of } p = (G_{\text{all}}, p) \).

In other words, \( p\text{-Checking Value of } p \) corresponds to the following Meta-problem:

- **Instance**: A graph \( G \) and an integer \( k \geq 0 \).
- **Parameter**: \( k \)
- **Question**: is it correct that \( p(G) \leq k \)?

Here \( \lessgtr = \leq / \geq \) if \( p \) is a minimization/maximization parameter.

For simplicity: \( \Pi_p = p\text{-Checking Value of } p \)

We call such a problem \( \Pi_p \) a **subset optimization problem**.

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

\[
G^p_k = \{ G \mid p(G) \leq k \}
\]

These are **YES**-instances for minimization problems and **NO**-instances for maximization problems.
Observe: for every $k$, $G^p_k$ is minor- (contraction-) closed.
Observe: for every $k$, $G_k^{P}$ is minor- (contraction-) closed.

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

▶ \( \text{obs}(\mathcal{G}^\mathcal{P}_k) \) is the set of minor-minimal elements not in \( \mathcal{G} \).

▶ If \( \mathcal{P} \) is minor-closed, then so is \( \mathcal{G}^\mathcal{P}_k \) and thus \( G \in \mathcal{G}^\mathcal{P}_k \iff \forall H \in \text{obs}(\mathcal{G}^\mathcal{P}_k) \, H \not\subseteq G \)

**Consequence of R&S theorem:** \( \forall k \in \mathbb{N} \, |\text{obs}(\mathcal{G}^\mathcal{P}_k)| < \aleph_0 \)
- Observe: for every $k$, $G^p_k$ is minor- (contraction-) closed.
- $\text{obs}(G^p_k)$ is the set of minor-minimal elements not in $G$.
- If $p$ is minor-closed, then so is $G^p_k$ and thus $G \in G^p_k \iff \forall H \in \text{obs}(G^p_k) \ H \not\sim G$

Consequence of R&S theorem: $\forall k \in \mathbb{N} \ |\text{obs}(G^p_k)| < \aleph_0$

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 $k$, $G^P_k$ is minor- (contraction-) closed.

- $\text{obs}(G^P_k)$ is the set of minor-minimal elements not in $G$.

- If $p$ is minor-closed, then so is $G^P_k$ and thus $G \in G^P_k \iff \forall H \in \text{obs}(G^P_k) \; H \not\leq G$

Consequence of R&S theorem: $\forall k \in \mathbb{N} \mid |\text{obs}(G^P_k)| < \aleph_0$

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 $p$ is minor-closed, then

$p$-Checking Value of $p \in \text{FPT}$. 
Observe: for every $k$, $G_k^p$ is minor- (contraction-) closed.

$\text{obs}(G_k^p)$ is the set of minor-minimal elements not in $G$.

If $p$ is minor-closed, then so is $G_k^p$ and thus $G \in G_k^p \iff \forall H \in \text{obs}(G_k^p) \ H \not\leq G$

Consequence of R&S theorem: $\forall k \in \mathbb{N} \ |\text{obs}(G_k^p)| < \aleph_0$

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 $p$ is minor-closed, then

$p\text{-Checking Value of } p \in \text{FPT}.$

In other words:

There exists an algorithm that solves $p\text{-Checking Value of } p$ in $f(k) \cdot n^3$ steps.
Corollary: If $p$ is a minor-closed graph parameter, then

$$\Pi_p^* = \text{p-CHECKING VALUE OF } p$$

can be solved in $f(k) \cdot n^3$ steps
Corollary: If $p$ is a minor-closed graph parameter, then

$$\Pi_p = \text{p-Checking Value of } p$$

can be solved in $f(k) \cdot n^3$ steps

▶ We have a (non-constructive) proof that an algorithm exists!
Corollary: If $p$ is a minor-closed graph parameter, then

\[ \Pi_p = \text{p-CHECKING VALUE OF } p \] can be solved in $f(k) \cdot n^3$ steps

▶ We have a (non-constructive) proof that an algorithm exists!
Corollary: If $p$ is a minor-closed graph parameter, then

\[ \Pi_p = p\text{-Checking Value of } p \text{ can be solved in } f(k) \cdot n^3 \text{ steps} \]

► We have a (non-constructive) proof that an algorithm exists!
► Is an encouraging theory (if you know that something exists...)

"Half of science is asking the right questions." 
Roger Bacon
Corollary: If $p$ is a minor-closed graph parameter, then

$$\Pi_p = \text{p-CHECKING VALUE OF } p$$

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 $p$ is a minor-closed graph parameter, then

\[ \Pi_p = \text{p-Checking Value of } p \] 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 $p$ is a minor-closed graph parameter, then

$$\Pi_p = \textit{p-CHECKING VALUE OF } p$$

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 $p$ is a minor-closed graph parameter, then

$$\Pi_p = \text{p-CHECKING VALUE OF } p$$

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 $p$ is a minor-closed graph parameter, then

$$
\Pi_p = \text{p-CHECKING VALUE OF } p \\
\text{can be solved in } f(k) \cdot n^3 \text{ steps}
$$

- 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 $p$ is a minor-closed graph parameter, then

\[ \Pi_p = \text{p-CHECKING VALUE OF } p \text{ can be solved in } f(k) \cdot n^3 \text{ steps} \]

- 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

"... NP-completeness in this context suggests that a problem area or approach is mathematically nasty.", Christos Papadimitriou – ICALP 1997

"P versus NP – a gift to mathematics from computer science." Steve Smale, 1998
Comments on P versus NP: from CS and Maths

“... 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 in this context suggests that a problem area or approach is mathematically nasty.”, Christos Papadimitriou – ICALP 1997

“P versus NP – a gift to mathematics from computer science.” Steve Smale, 1998
A metaphor on graph minors

"Graph Minors suggests that some problems are algorithmically not so nasty"

"Graph Minors – a gift to computer science from mathematics"
A metaphor on graph minors

“Graph Minors suggests that some problems are algorithmically not so nasty”
A metaphor 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] $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
   i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $p(G) \leq k$ in $2^{O(tw(G))} \cdot n$ steps
   Typically this is done by Dynamic Programming.

Proof:
This algorithm first checks whether $tw(G) \leq f(k)$.
If the answer is negative, then outputs a negative/positive answer (by 1).
If the answer is positive, then runs DP algorithm (by 2).
General requirements

We need the following two facts (for suitable functions $f$ and $G$):

1. [Combinatorial] $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $p(G) \leq k$ in $2^{g(tw(G))} \cdot n^{O(1)}$ steps
Typically this is done by Dynamic Programming.
We need the following two facts (for suitable functions $f$ and $G$):

1. **[Combinatorial]** $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
   i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. **[Algorithmic]** One can check whether $p(G) \leq k$ in $2^{g(tw(G)) \cdot n^{O(1)}}$ steps
   Typically this is done by Dynamic Programming.

$1 + 2 \rightarrow \Pi_p$ 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] $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
   i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $p(G) \leq k$ in $2^{g(tw(G))} \cdot n^{O(1)}$ steps
   Typically this is done by Dynamic Programming.

$1+2 \rightarrow \Pi_p$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

Proof:

This algorithm first checks whether $tw(G) \leq f(k)$.
If the answer is negative, then outputs a negative/positive answer (by 1).
If the answer is positive, then runs DP algorithm (by 2).
$1+2 \rightarrow \Pi_p$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

- $g(\cdot)$ is linear: $\Pi_p$ is \textit{singly exponentially solvable w.r.t. treewidth}
$1 + 2 \rightarrow \Pi_p$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm.

▶ $g(\cdot)$ is linear: $\Pi_p$ is *singly exponentially solvable w.r.t. treewidth*
$1 + 2 \rightarrow \Pi_\mathbf{p}$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

$\blacktriangleright$ $g(\cdot)$ is linear: $\Pi_\mathbf{p}$ 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

$2^{O(\sqrt{k})} \cdot n^{O(1)}$ for their planar restrictions
$1 + 2 \rightarrow \Pi_p$ has a $2^{g(f(k))} \cdot n^{O(1)}$ step algorithm

$\blacktriangleright$ $g(\cdot)$ is linear: $\Pi_p$ 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 $2^{O(\sqrt{k})} \cdot n^{O(1)}$ for their planar restrictions $\rightarrow$ when can we match this?

$\blacktriangleright$ 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 $\Pi_p$ have $\text{tw} = O(k)$ (or $o(k)$ in planar graphs)
We care about combinatorial condition:

1. [Combinatorial] YES/NO-instances of $\Pi_p$ have $\text{tw} = O(k)$ (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 $\Pi_p$ have $\text{tw} = O(k)$ (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 $\Pi_p$ have $\text{tw} = O(k)$ (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 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 $\Pi_p$ have $\text{tw} = O(k)$ (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 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 $\forall \alpha \ tw(G) > \delta(\alpha) \Rightarrow G \geq_{m} \bigboxplus_{\alpha}$. 
Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \rightarrow \mathbb{N}$ such that $\forall \alpha \ tw(G) > \delta(\alpha) \Rightarrow G \geq m \bigoplus \alpha$.

Upper bound for $\delta$: remained exponential for a long time...
Theorem: [Robertson & Seymour – GM-V]

There is a $\delta : \mathbb{N} \to \mathbb{N}$ such that $\forall \alpha \ t_w(G) > \delta(\alpha) \Rightarrow G \geq_m \Box_\alpha$.

**Upper bound** for $\delta$: remained **exponential** for a long time...

**Until:**
Theorem: [Robertson & Seymour – GM-V]

There is a \( \delta : \mathbb{N} \to \mathbb{N} \) such that \( \forall \alpha \ tw(G) > \delta(\alpha) \Rightarrow G \geq_m \biguplus_{\alpha} \).

**Upper bound** for \( \delta \): 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 $\forall \alpha \ tw(G) > \delta(\alpha) \Rightarrow G \geq_m \bigoplus_\alpha$.

**Upper bound** for $\delta$: remained **exponential** for a long time...

**Until:** Julia Chuzhoy proved: $\delta(k) = O(k^{20})$

► The best known lower bound is $\delta(k) = \Omega(k^2 \cdot \log k)$
Definition: \( p^{-1}(k) = \min\{\alpha \mid p(\oplus \alpha) > k\} \)
Definition: \( p^{-1}(k) = \min\{\alpha \mid p(\oplus \alpha) > k\} \)

Observe the following:
Definition: $p^{-1}(k) = \min\{\alpha \mid p(\square \alpha) > k\}$

Observe the following:

Lemma: If $p$ is minor-closed, then

1. [Combinatorial] $p(G) \leq k \Rightarrow tw(G) \leq \delta(p^{-1}(k))$
**Definition:** \( p^{-1}(k) = \min\{\alpha \mid p(\boxplus \alpha) > k\} \)

Observe the following:

**Lemma:** If \( p \) is minor-closed, then

1. **[Combinatorial]** \( p(G) \leq k \Rightarrow \text{tw}(G) \leq \delta(p^{-1}(k)) \)

**Proof:** Let \( \alpha = p^{-1}(k) \). Then \( k < p(\boxplus \alpha) \) (1).

Assume \( \delta(\alpha) < \text{tw}(G) \) (2).
Definition: \( p^{-1}(k) = \min\{\alpha \mid p(\boxplus \alpha) > k\} \)

Observe the following:

Lemma: If \( p \) is minor-closed, then

1. [Combinatorial] \( p(G) \leq k \Rightarrow \text{tw}(G) \leq \delta(p^{-1}(k)) \)

Proof: Let \( \alpha = p^{-1}(k) \). Then \( k < p(\boxplus \alpha) \) (1).

Assume \( \delta(\alpha) < \text{tw}(G) \) (2).

[By grid exclusion]: (2) \( \Rightarrow \boxplus \alpha \leq_m G \). (1)
Definition: \( p^{-1}(k) = \min\{ \alpha \mid p(\boxplus \alpha) > k \} \)

Observe the following:

Lemma: *If \( p \) is minor-closed, then*

1. \([\text{Combinatorial}]\) \( p(G) \leq k \Rightarrow tw(G) \leq \delta(p^{-1}(k)) \)

Proof: Let \( \alpha = p^{-1}(k) \). Then \( k < p(\boxplus \alpha) \) (1).

Assume \( \delta(\alpha) < tw(G) \) (2).

[By grid exclusion]: (2) \( \Rightarrow \boxplus \alpha \leq m G \). (1)

[By minor-closedness]: if \( \boxplus \alpha \leq m G \), then \( p(\boxplus \alpha) \leq p(G) \). (3)
Definition: \( p^{-1}(k) = \min \{ \alpha \mid p(\boxplus \alpha) > k \} \)

Observe the following:

**Lemma:** If \( p \) is minor-closed, then

1. **[Combinatorial]** \( p(G) \leq k \Rightarrow tw(G) \leq \delta(p^{-1}(k)) \)

**Proof:** Let \( \alpha = p^{-1}(k) \). Then \( k < p(\boxplus \alpha) \) \( (1) \).

Assume \( \delta(\alpha) < tw(G) \) \( (2) \).

[By grid exclusion]: \( (2) \Rightarrow \boxplus \alpha \leq_m G. \) \( (1) \)

[By minor-closedness]: if \( \boxplus \alpha \leq_m G \), then \( p(\boxplus \alpha) \leq p(G) \). \( (3) \)

\( (1) \) and \( (3) \) \( \Rightarrow k < p(G') \).
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 $\Pi$ minor-bidimensional if $p$ is minor-closed and $p(k) = O(\sqrt{k})$.

Not all minor-closed problems are bidimensional! such as Treewidth, Pathwidth, Branchwidth, Tree-depth, and Genus.
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 $\Pi$ _minor-bidimensional_ if $p^{-1}(k) = O(\sqrt{k})$.

Not all minor-closed problems are bidimensional! such as Treewidth, Pathwidth, Branchwidth, Tree-depth, and Genus.

$$\text{vc}^{-1}(k) = O(\sqrt{k})$$
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 \( \Pi \) minor-bidimensional if \( p \) is minor-closed and \( p^{-1}(k) = O(\sqrt{k}) \).

Not all minor-closed problems are bidimensional, such as Treewidth, Pathwidth, Branchwidth, Tree-depth, and Genus.

\[
\begin{align*}
\text{vc}^{-1}(k) &= O(\sqrt{k}) \\
\text{lp}^{-1}(k) &= O(\sqrt{k})
\end{align*}
\]
The same holds for the following \textit{minor-closed} parameters:

\begin{align*}
\text{vc}^{-1}(k) &= O(\sqrt{k}) \\
\text{lp}^{-1}(k) &= O(\sqrt{k})
\end{align*}
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 $\Pi$ minor-bidimensional if $p(k) = O(\sqrt{k})$.

Not all minor-closed problems are bidimensional! such as Treewidth, Pathwidth, Branchwidth, Tree-depth, and Genus.

\[
\text{vc}^{-1}(k) = O(\sqrt{k}) \\
\text{lp}^{-1}(k) = O(\sqrt{k})
\]
The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,

\[
\begin{align*}
vc^{-1}(k) &= O(\sqrt{k}) \\
lp^{-1}(k) &= O(\sqrt{k})
\end{align*}
\]
The same holds for the following **minor-closed** parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,

\[
\text{vc}^{-1}(k) = O(\sqrt{k})
\]
\[
\text{lp}^{-1}(k) = O(\sqrt{k})
\]
The same holds for the following **minor-closed** parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,
- Face Cover

\[
\begin{align*}
\text{vc}^{-1}(k) &= O(\sqrt{k}) \\
\text{lp}^{-1}(k) &= O(\sqrt{k})
\end{align*}
\]
The same holds for the following minor-closed parameters:

- Feedback Vertex Set,
- Longest Cycle,
- Cycle Packing,
- Face Cover
- Max Series-Parallel Subgraph

\[ \text{vc}^{-1}(k) = O(\sqrt{k}) \]
\[ \text{lp}^{-1}(k) = O(\sqrt{k}) \]
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 $\Pi_p$ minor-bidimensional if $p$ is minor-closed and $p^{-1}(k) = \sqrt{k}$.
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 \( \Pi_p \) minor-bidimensional if \( p \) is minor-closed and \( p^{-1}(k) = \sqrt{k} \)

\[ \text{vc}^{-1}(k) = O(\sqrt{k}) \]
\[ \text{lp}^{-1}(k) = O(\sqrt{k}) \]

- Not all minor-closed problems are bidimensional!

such as Treewidth, Pathwidth, Branchwidth, Tree-depth, and Genus
For minor-bidimensional parameters:

\[
\text{Combinatorial } p(G) \leq k \Rightarrow tw(G) \leq \delta(\sqrt{k})
\]

Recall that \(\delta(k) = \Omega(k^2 \cdot \log k)\).

Best of all scenarios:

\[
O(k \log k) \cdot n^{O(1)} \text{-step algorithms}
\]

"Best of all scenarios" means that

\(\delta(k) = O(k^2 \cdot \log k)\) (which is conjectured but not sure!) and

\(\Pi_p\) is singly exponentially solvable w.r.t. treewidth (which is the case for many problems).
For minor-bidimensional parameters:

1. [Combinatorial] $p(G) \leq k \Rightarrow tw(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all senarios:

$O(k \log k) \cdot n$ step algorithms

"Best of all senarios" means that

- $\delta(k) = O(k^2 \cdot \log k)$ (which is conjectured but not sure!)
- $\Pi_p$ is singly exponentially solvable w.r.t. treewidth (which is the case for many problems)
For minor-bidimensional parameters:

1. [Combinatorial] \( p(G) \leq k \Rightarrow tw(G) \leq \delta(\sqrt{k}) \)

Recall that \( \delta(k) = \Omega(k^2 \cdot \log k) \).
For minor-bidimensional parameters:

1. [Combinatorial] \( p(G) \leq k \Rightarrow \text{tw}(G) \leq \delta(\sqrt{k}) \)

Recall that \( \delta(k) = \Omega(k^2 \cdot \log k) \)

Best of all scenarios: \( 2^{O(k \log k)} \cdot n^{O(1)} \) step algorithms
For minor-bidimensional parameters:

1. [Combinatorial] $p(G) \leq k \Rightarrow \text{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all scenarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms

“Best of all scenarios” means that
For minor-bidimensional parameters:

1. [Combinatorial] $\text{p}(G) \leq k \Rightarrow \text{tw}(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all scenarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms

“Best of all scenarios” means that

$\delta(k) = O(k^2 \cdot \log k)$ (which is conjectured but not sure!) and
For minor-bidimensional parameters:

1. [Combinatorial] $p(G) \leq k \Rightarrow tw(G) \leq \delta(\sqrt{k})$

Recall that $\delta(k) = \Omega(k^2 \cdot \log k)$

Best of all scenarios: $2^{O(k \log k)} \cdot n^{O(1)}$ step algorithms

“Best of all scenarios” means that

$\Rightarrow \delta(k) = O(k^2 \cdot \log k)$ (which is conjectured but not sure!) and

$\Pi_p$ 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.

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 $\delta$ is better bounded).

4. For even better (e.g. subexponential) parameterized dependency we must restrict our attention 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.

4. For even better (e.g. subexponential) parameterized dependency we must restrict our attention 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.
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 $\delta$ 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 $\delta$ is better bounded)

4. For even better (e.g. subexponential) parameterized dependency we must restrict our attention to special graph classes.
Subexponential parameterized algorithms

Definition:
A graph class $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k \geq G \Rightarrow \text{tw}(G) = O(k^c)$.

Then:
YES/NO-instances of a bidimensional problem $\Pi$ have $\text{tw}(G) = (p-1)(k)^c = O((\sqrt{k})^c) = o(k)$ if $\Pi$ is singly exponentially solvable w.r.t. treewidth, then $\Pi$ can be solved in $2^{o(k)} \cdot n^O(1)$ steps.

Planar graphs have the SQGM property for $c = 1$.

As $\text{tw}(G) = O(\text{bw}(G))$, the above follows from the following:

Theorem: [Robertson, Seymour, & Thomas 1994]
If $G$ is planar and $\text{bw}(G) \geq 4k$, then $k \leq m_G$.

We sketch the Idea of the proof of the above theorem:
**Subexponential parameterized algorithms**

**Definition:** A graph class $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c)$
Subexponential parameterized algorithms

Definition: A graph class $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k \Box_k \not\subseteq G \Rightarrow tw(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\Pi_p$ have $tw = (p^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$
**Subexponential parameterized algorithms**

**Definition:** A graph class $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k \begin{bmatrix} k & k \end{bmatrix} \not\subseteq G \Rightarrow tw(G) = O(k^c)$

**Then:** YES/NO-instances of a bidimensional problem $\Pi_p$ have $tw = (p^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

- if $\Pi$ is singly exponentially solvable w.r.t. treewidth, then $\Pi$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.
Subexponential parameterized algorithms

**Definition:** A graph class $G$ has the subquadratic grid minor property (**SQGM**) if there exist $1 \leq c < 2$ such that $\forall k \left[ \square_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \right]$

Then: YES/NO-instances of a bidimensional problem $\Pi_p$ have $\text{tw} = (p^{-1}(k))^c = O((\sqrt{k}^c) = o(k)$

► if $\Pi$ is singly exponentially solvable w.r.t. treewidth, then $\Pi$ 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 $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k \; \square_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c)$

Then: YES/NO-instances of a bidimensional problem $\Pi_p$ have $\text{tw} = (p^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

► if $\Pi$ is singly exponentially solvable w.r.t. treewidth, then $\Pi$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Planar graphs have the SQGM property for $c = 1$

As $\text{tw}(G) = O(\text{bw}(G))$, the above follows from the following:
Subexponential parameterized algorithms

**Definition:** A graph class $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k$ $k \times k \not\leq G \Rightarrow \text{tw}(G) = O(k^c)$

Then: **YES/NO**-instances of a bidimensional problem $\Pi_p$ have

$tw = (p^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

If $\Pi$ is singly exponentially solvable w.r.t. treewidth, then $\Pi$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Planar graphs have the SQGM property for $c = 1$

As $tw(G) = O(bw(G))$, the above follows from the following:

**Theorem:** [Robertson, Seymour, & Thomas 1994] If $G$ is planar and $bw(G) \geq 4k$, then $k \times k \leq_m G$. 
Subexponential parameterized algorithms

**Definition:** A graph class $G$ has the subquadratic grid minor property (SQGM) if there exist $1 \leq c < 2$ such that $\forall k \begin{array}{c} k \times k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \end{array}$

Then: YES/NO-instances of a bidimensional problem $\Pi_p$ have $\text{tw} = (p^{-1}(k))^c = O((\sqrt{k})^c) = o(k)$

If $\Pi$ is singly exponentially solvable w.r.t. treewidth, then $\Pi$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps.

Planar graphs have the SQGM property for $c = 1$

As $\text{tw}(G) = O(\text{bw}(G))$, the above follows from the following:

**Theorem:** [Robertson, Seymour, & Thomas 1994] If $G$ is planar and $\text{bw}(G) \geq 4k$, then $\begin{array}{c} k \times k \leq_m G \end{array}$

We sketch the idea of the proof of the above theorem:
Suppose that we constructed a partial branch decomposition of the part of the graphs that is inside a disk.
“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 $k$ as a minor.
Bidimensionality race:

**SQGM**: \( \forall k \bigoplus_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \) for some \( c < 2 \).

When **SQGM** property holds?
Bidimensionality race:

\[ \forall k \exists k \leq G \Rightarrow \text{tw}(G) = O(k^c) \text{ for some } c < 2. \]

When \textbf{SQGM} property holds?

Planar: [Robertson and Seymour, JCSTB 1986]
Bidimensionality race:

**SQGM**:

\[ \forall k \quad \square_k \not\preceq G \Rightarrow tw(G) = O(k^c) \text{ for some } c < 2. \]

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 race:

**SQGM**: $\forall k \square_k \not\subseteq G \Rightarrow tw(G) = O(k^c)$ 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]

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 race:

**SQGM**: \( \forall k \bigoplus_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \) 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**: \( \forall k \bigoplus_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \) 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**: \( \forall k \; \mathbb{P}_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \) 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:

\[ \forall k \, \square_k \not\subseteq G \Rightarrow \text{tw}(G) = O(k^c) \text{ for some } c < 2. \]

When \textbf{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]

\[ \rightarrow \text{ In all above cases we have topologically refined graph classes and } c = 1. \]
Bidimensionality race:

\[
\text{SQGM: } \forall k \bigoplus_k \not\preceq G \Rightarrow \text{tw}(G) = O(k^c) \text{ for some } c < 2.
\]

When \text{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 $\Pi_{ds}$]
Bidimensionality for contraction-closed problems

[such as $\Pi_{ds}$]

$k$ is replaced by the uniformly triangulated grid $\Gamma_k$:

Let $\tilde{p}^{-1}(k) = \min\{\alpha \mid p(\Gamma_{\alpha}) > k\}$
Bidimensionality for contraction-closed problems

[such as \( \Pi_{ds} \)]

\( \boxplus_k \) is replaced by the uniformly triangulated grid \( \Gamma_k \):

Let \( \tilde{p}^{-1}(k) = \min\{\alpha \mid p(\Gamma_{\alpha}) > k\} \)

\[ \]

\textbf{Definition:}

We call a problem \( \Pi_p \) contraction-bidimensional if \( p \) is contraction-closed and \( \tilde{p}^{-1}(k) = \sqrt{k} \)
Bidimensionality for contraction-closed problems

[such as \( \Pi_{ds} \)]

\( \boxplus_k \) is replaced by the uniformly triangulated grid \( \Gamma_k \):

\[
\tilde{p}^{-1}(k) = \min\{\alpha \mid p(\Gamma_{\alpha}) > k\}
\]

**Definition:**

We call a problem \( \Pi_p \) contraction-bidimensional if \( p \) is contraction-closed and \( \tilde{p}^{-1}(k) = \sqrt{k} \)

**Definition:** A class \( \mathcal{G} \) has the *subquadratic grid contraction property* (**SQGC**) if there exist \( 1 \leq c < 2 \) such that \( \forall k \quad \Gamma_k \not\subset_c \mathcal{G} \Rightarrow tw(\mathcal{G}) = O(k^c) \)
Subquadratic grid contraction property (SQGC) holds for planar graphs because of:

**Theorem:** [Robertson, Seymour, & Thomas 1994] If $G$ is planar and $bw(G) \geq 4k$, 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 $\Gamma_k$. Therefore $\Gamma_k \not\leq c G \Rightarrow tw(G) = O(kc)$, thus SQGC holds for $c = 1$. 

Subquadratic grid contraction property (SQGC) holds for planar graphs because of:

**Theorem:** [Robertson, Seymour, & Thomas 1994] If $G$ is planar and $bw(G) \geq 4k$, 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 $\Gamma_k$. 
Subquadratic grid contraction property (SQGC) holds for planar graphs because of:

Theorem: [Robertson, Seymour, & Thomas 1994] If $G$ is planar and $bw(G) \geq 4k$, then $\Gamma_k \leq_m G$.

Proof: if we do not apply edge removals while obtaining $\Gamma_k$ from $G$ we end up to a partially triangulated grid that can be further be contracted to the uniformly triangulated grid $\Gamma_k$. 

![Diagram showing the process of grid contraction](image)
Subquadratic grid contraction property (SQGC) holds for planar graphs because of:

**Theorem:** [Robertson, Seymour, & Thomas 1994] If $G$ is planar and $bw(G) \geq 4k$, then $\Gamma_k \not\leq_m G$.

**Proof:** if we do not apply edge removals while obtaining $\Gamma_k$ from $G$ we end up to a partially triangulated grid that can be further be contracted to the uniformly triangulated grid $\Gamma_k$.

Therefore $\Gamma_k \not\leq_c G \Rightarrow tw(G) = O(k^c)$, thus SQGC holds for $c = 1$. 

![Image of grid contraction process](image.png)
Bidimensionality race:

\[ SQGC: \forall k \Gamma_k \preceq_c G \Rightarrow tw(G) = O(k^c) \text{ for some } c < 2. \]

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**: \( \forall k \Gamma_k \nsubseteq_c G \Rightarrow \text{tw}(G) = O(k^c) \) 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:** \( \forall k \; \Gamma_k \not\subseteq_c G \Rightarrow \text{tw}(G) = O(k^c) \) 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

\[ \exists v \in V(H) : H - v \text{ is planar} \]
Bidimensionality and subexponential algorithms.

**Theorem:** Let \( \Pi_p \) 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 \( \text{SQGM} / \text{SQGC} \)-graph class

Then \( \Pi \) can be solved in \( 2^{o(k)} \cdot n^{O(1)} \) steps.
Theorem: Let $\Pi_p$ 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 $\text{SQGM}/\text{SQGC}$-graph class

Then $\Pi_p$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps
Bidimensionality and subexponential algorithms.

**Theorem**: Let $\Pi_p$ 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 $\Pi_p$ 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 $\Pi_p$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps
Bidimensionality and subexponential algorithms.

Theorem: Let $\Pi_p$ 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 $\Pi$ can be solved in $2^{o(k)} \cdot n^{O(1)}$ steps
Some bidimensional problems

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]
[Rué, Sau, Thilikos, TALG 2014]
The previous theorem can become an algorithmic meta-theorem as

\[ \text{ii. is singly exponentially solvable w.r.t. treewidth} \]

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] ★
[Rué, Sau, Thilikos, TALG 2014] ★
[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk, FOCS 2011] ★★
[Bodlaender, Cygan, Kratsch, Nederlof, ICALP 2013] ★★
[Fomin, Lokshtanov, Saurabh, SODA 2014] ★★
Bidimensionality and Kernelization
Irrelevant vertex technique
Bidimensionality and Kernelization
**Kernelization**

Let $(\Pi, \kappa)$ be a parameterized problem.
Kernelization

Let \((\Pi, \kappa)\) be a parameterized problem. Recall: that \(\Pi \subseteq \Sigma^*\) and \(\kappa : \Sigma^* \rightarrow \mathbb{N}\).
Kernelization

Let \((\Pi, \kappa)\) be a parameterized problem. Recall: that \(\Pi \subseteq \Sigma^*\) and \(\kappa : \Sigma^* \rightarrow \mathbb{N}\).

A polynomial algorithm \(A\) is a kernelization algorithm for \((\Pi, \kappa)\) if there exist some computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\) such that, for every \(x \in \Sigma^*\), the output \(x' = A(x)\) satisfies the following:

1. \(x \in \Pi \iff x' \in \Pi\) (and \(x'\) and \(x\) are equivalent).
2. \(|x'| \leq g(k)\) (new instance has size bounded by a function of the parameter).
Kernelization

Let \((\Pi, \kappa)\) be a parameterized problem. Recall that \(\Pi \subseteq \Sigma^*\) and \(\kappa : \Sigma^* \rightarrow \mathbb{N}\).

A polynomial algorithm \(A\) is a \textit{kernelization algorithm} for \((\Pi, \kappa)\) if there exist some computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\) such that, for every \(x \in \Sigma^*\), the output \(x' = A(x)\) satisfies the following:

1. \(x \in \Pi \iff x' \in \Pi\) (\(x\) and \(x'\) are equivalent)
Kernelization

Let \((\Pi, \kappa)\) be a parameterized problem. Recall: that \(\Pi \subseteq \Sigma^*\) and \(\kappa: \Sigma^* \rightarrow \mathbb{N}\).

A polynomial algorithm \(A\) is a kernelization algorithm for \((\Pi, \kappa)\) if there exist some computable function \(g: \mathbb{N} \rightarrow \mathbb{N}\) such that, for every \(x \in \Sigma^*\), the output \(x' = A(x)\) satisfies the following:

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 \((\Pi, \kappa)\) be a parameterized problem. Recall: that \(\Pi \subseteq \Sigma^*\) and \(\kappa : \Sigma^* \rightarrow \mathbb{N}\).

A polynomial algorithm \(A\) is a \textit{kernelization algorithm} for \((\Pi, \kappa)\) if there exist some computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\) such that, for every \(x \in \Sigma^*\), the output \(x' = A(x)\) satisfies the following:

1. \(x \in \Pi \iff x' \in \Pi\) (\(x\) and \(x'\) are equivalent)
2. \(|x'| \leq g(k)\) (new instance has size bounded by a function of the parameter).

![Kernelization Diagram](image)
Kernelization

Let \((\Pi, \kappa)\) be a parameterized problem. Recall: that \(\Pi \subseteq \Sigma^*\) and \(\kappa : \Sigma^* \rightarrow \mathbb{N}\).

A polynomial algorithm \(A\) is a *kernelization algorithm* for \((\Pi, \kappa)\) if there exist some computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\) such that, for every \(x \in \Sigma^*\), the output \(x' = A(x)\) satisfies the following:

1. \(x \in \Pi \iff x' \in \Pi\) (\(x\) 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 $(\Pi, \kappa)$ be a parameterized problem. Recall: that $\Pi \subseteq \Sigma^*$ and $\kappa : \Sigma^* \rightarrow \mathbb{N}$.

- A polynomial algorithm $A$ is a *kernelization algorithm* for $(\Pi, \kappa)$ if there exist some computable function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that, for every $x \in \Sigma^*$, the output $x' = A(x)$ satisfies the following:
  1. $x \in \Pi \iff x' \in \Pi$ ($x$ 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 $k = \kappa(x)$.
Let \((\Pi, \kappa)\) be a parameterized problem. Recall: that \(\Pi \subseteq \Sigma^*\) and \(\kappa : \Sigma^* \rightarrow \mathbb{N}\).

A polynomial algorithm \(A\) is a kernelization algorithm for \((\Pi, \kappa)\) if there exist some computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\) such that, for every \(x \in \Sigma^*\), the output \(x' = A(x)\) satisfies the following:

1. \(x \in \Pi \iff x' \in \Pi\) (\(x\) 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.
\(p\text{-Vertex Cover}\) has kernelization algorithm that produces a kernel of \(\leq 2k\) vertices.
\textit{p-Vertex Cover} has kernelization algorithm that produces a \textit{kernel} of $\leq 2k$ vertices.

Alternatively, we say that \textit{p-Vertex Cover} has a \textit{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.

\( p\text{-Dominating Set} \) is \( W[2]\)-complete, this it is not expected to have a kernel.
A parameterized problem has a kernel iff it is in FPT.

\( p\text{-Dominating Set} \) is \( W[2]\)-complete, this it is not expected to have a kernel.

Not all problems in FPT are expected to have polynomial kernels (\( p\text{-Path} \)).

[ Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]
- A parameterized problem has a kernel iff it is in FPT
- \textbf{p-Dominating Set} is \textit{W}[2]-complete, this it is not expected to have a kernel.

---

- Not all problems in \textbf{FPT} are expected to have polynomial kernels (\textit{p-Path})
  [Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]
- \textbf{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 \textit{FPT}.

\textit{p-Dominating Set} is \textit{W}[2]-complete, this it is not expected to have a kernel.

Not all problems in \textit{FPT} are expected to have polynomial kernels \(p\text{-Path}\).

[\textit{Bodlaender, Downey, Fellows, Hermelin, JCSS 2009}]

\textit{p-Feedback Vertex Set} has a kernel of \(O(k^2)\) edges.

[\textit{Thomassé, TALG 2010}]

Not all FPT-problems are expected to have linear kernel \(p\text{-Feedback Vertex Set}\).

[\textit{Dell, van Melkebeek, STOC 2010}]
A parameterized problem has a kernel iff it is in FPT.

\textbf{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 (\textbf{p-Path})

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

\textbf{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 (\textbf{p-Feedback Vertex Set})

[Dell, van Melkebeek, STOC 2010]

\textbf{p-Planar Dominating Set} kernel of $67k$ vertices.

[Chen, Fernau, Kanj, Xia, SICOMB 2007]
A parameterized problem has a kernel \textit{iff} it is in FPT

\textbf{p-Dominating Set} is \textit{W}[2]-complete, this it is not expected to have a kernel.

Not all problems in FPT are expected to have polynomial kernels (\textit{p-Path})

[Bodlaender, Downey, Fellows, Hermelin, JCSS 2009]

\textbf{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 (\textit{p-Feedback Vertex Set})

[Dell, van Melkebeek, STOC 2010]

\textbf{p-Planar Dominating Set} kernel of $67k$ vertices.

[Chen, Fernau, Kanj, Xia, SICOMB 2007]

\textbf{p-Planar Feedback Vertex Set} has a kernel of $13k$ vertices.

[Bonamy, Kowalik, IPEC 2014]
A parameterized problem has a kernel \textit{iff} it is in \textsc{FPT}.

\textsc{p-Dominating Set} is \textsc{W}[2]-complete, this it is not expected to have a kernel.

Not all problems in \textsc{FPT} are expected to have polynomial kernels (\textsc{p-Path}).

[\textit{Bodlaender, Downey, Fellows, Hermelin, JCSS 2009}]

\textsc{p-Feedback Vertex Set} has a kernel of $O(k^2)$ edges.

[\textit{Thomassé, TALG 2010}]

Not all \textsc{FPT}-problems are expected to have linear kernel (\textsc{p-Feedback Vertex Set}).

[\textit{Dell, van Melkebeek, STOC 2010}]

\textsc{p-Planar Dominating Set} kernel of $67k$ vertices.

[\textit{Chen, Fernau, Kanj, Xia, SICOMB 2007}]

\textsc{p-Planar Feedback Vertex Set} has a kernel of $13k$ vertices.

[\textit{Bonamy, Kowalik, IPEC 2014}]

\textsc{p-Path}, \textsc{p-Dominating Set}, and \textsc{p-Feedback Vertex Set} are bidimensional.
Protrusions

Protrusions in graph theory are defined as a set $X \subseteq V(G)$ such that $\tw(G[X]) \leq r \rightarrow \text{treewidth is bounded}$ and $|\partial G(G)| \leq r \rightarrow \text{its boundary is of bounded size}$. 

Felix Reidl
Protrusions

*r-protrusion*: a set $X \subseteq V(G)$ where

- $\text{treewidth is bounded}$
- $|\partial G(X)| \leq r \Rightarrow$ its boundary is of bounded size
**Protrusions**

An *$r$-protrusion* is a set $X \subseteq V(G)$ where

- $\text{tw}(G[X]) \leq r \rightarrow$ treewidth is bounded
Protrusions

\( r\text{-protrusion} \): a set \( X \subseteq V(G) \) where

- \( \text{tw}(G[X]) \leq r \rightarrow \text{treewidth is bounded} \)
- \( |\partial_G(G)| \leq r \rightarrow \text{its boundary is of bounded size} \)
Protrusion decompositions

An \((\alpha, \beta)\)-protrusion decomposition of \(G\) is a partition \(P = \{R_0, R_1, \ldots, R_\rho\}\) of \(V(G)\) such that

\[
\max\{\rho, |R_0|\} \leq \alpha,
\]

each \(N_{G[R_i]}, i \in \{1, \ldots, \rho\}\), is a \(\beta\)-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 decompositions

An $\alpha,\beta$-protrusion decomposition of $G$ is a partition $P = \{R_0, R_1, \ldots, R_\rho\}$ of $V(G)$ such that

\[
\max\{\rho, |R_0|\} \leq \alpha,
\]

each $N_{G}[R_i], i \in \{1, \ldots, \rho\},$ is a $\beta$-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 decompositions

An \((\alpha, \beta)\)-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 \(\beta\)-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 decompositions

An \((\alpha, \beta)\)-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 \(\beta\)-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!
An \((\alpha, \beta)\)-protrusion decomposition of \(G\) is a partition \(P = \{R_0, R_1, \ldots, R_\rho\}\) of \(V(G)\) such that

\[
\max\{\rho, |R_0|\} \leq \alpha,
\]

Remark: actually, this last condition is not necessary! But makes things more visualizable!
An \((\alpha, \beta)\)-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 \(\beta\)-protrusion of \(G\), and

Remark: actually, this last condition is not necessary! But makes things more visualizable!
Protrusion decompositions

An \((\alpha, \beta)\)-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 \(\beta\)-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!
$f$-protrusion replacement family for $\Pi_p$: 

$a$ collection $A = \{A_i | i \geq 0\}$ of algorithms, such that algorithm $A_i$ receives an instance $(G,k)$ of $\Pi_p$ and an $i$-protrusion $X$ of $G$ with at least $f(i)$ vertices and outputs an equivalent instance $(G',k')$ of $\Pi_p$ where $|V(G')| < |V(G)|$ and $k' \leq k$. 

Protrusion replacement
**Protrusion replacement**

A *f-protrusion replacement family* for $\Pi_p$: a collection $\mathcal{A} = \{A_i \mid i \geq 0\}$ of algorithms, such that algorithm $A_i$ receives an instance $(G, k)$ of $\Pi_p$ and an $i$-protrusion $X$ of $G$ with at least $f(i)$ vertices and outputs an equivalent instance $(G', k')$ of $\Pi_p$ where $|V(G')| < |V(G)|$ and $k' \leq k$. 
Protrusion replacement

\textit{f-protrusion replacement family} for $\Pi_p$: a collection $\mathcal{A} = \{A_i \mid i \geq 0\}$ of algorithms, such that algorithm $A_i$ receives an instance $(G, k)$ of $\Pi_p$ and an $i$-protrusion $X$ of $G$ with at least $f(i)$ vertices and outputs an equivalent instance $(G', k')$ of $\Pi_p$ where $|V(G')| < |V(G)|$ and $k' \leq k$. 
Conditions for the existence of linear kernels

1. Combinatorial
   If \( p(G) \leq k \), then \( G \) has an \((O(k),O(1))\)-protrusion decomposition.

2. Algorithmic
   \( \Pi_p \) has a protrusion replacement family.

To achieve Conditions 1 and 2, we need some more definitions!
Conditions for the existence of linear kernels

1. [Combinatorial] If \( p(G) \leq k \), then \( G \) has an \( (O(k), O(1)) \)-protrusion decomposition.
Conditions for the existence of linear kernels

1. [Combinatorial] If $p(G) \leq k$, then $G$ has an $(O(k), O(1))$-protrusion decomposition.

2. [Algorithmic] $\Pi_p$ has a protrusion replacement family.
Conditions for the existence of linear kernels

1. [Combinatorial] If $p(G) \leq k$, then $G$ has an $(O(k), O(1))$-protrusion decomposition.

2. [Algorithmic] $\Pi_p$ has a protrusion replacement family.

$\rightarrow 1. + 2 \to$ a linear kernel for $\Pi_p$.

[Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh, Thilikos, FOCS 2008]
Conditions for the existence of linear kernels

1. [Combinatorial] If $p(G) \leq k$, then $G$ has an $(O(k), O(1))$-protrusion decomposition.

2. [Algorithmic] $\Pi_p$ has a protrusion replacement family.

$\implies$ 1. + 2 $\rightarrow$ a linear kernel for $\Pi_p$.

[Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh, Thilikos, FOCS 2008]

To achieve Conditions 1 and 2, we need some more definitions!
CMSO-expressibility

Let $p$ be a graph optimization parameter and let $\Pi_p$ be the corresponding graph optimization problem.

If $\phi$ is expressible in Monadic Second Order Logic, then we say that $\Pi_p$ is CMSO-expressible.
Let \( p \) be a graph optimization parameter and let \( \Pi_p \) be the corresponding graph optimization problem.

Recall that:

\[
p(G) = \min\{k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true}\}\]

If \( \phi \) is expressible in Monadic Second Order Logic, then we say that \( \Pi_p \) is CMSO-expressible.
Let $p$ be a graph optimization parameter and let $\Pi_p$ be the corresponding graph optimization problem.

Recall that:

$$p(G) = \min \{ k \mid \exists S \subseteq V(G) : |S| \leq k \land \phi(G, S) = \text{true} \}$$

- If $\phi$ is expressible in Monadic Second Order Logic, then we say that $\Pi_p$ is **CMSO-expressible**
Linear Separability

Let $p$ be an graph optimization parameter and $G$ be a graph

The subset optimization problem $\Pi_p$ is linearly separable if, for any graph $G$ and $L \subseteq V(G)$ such that $|C| = |\partial G(L)| \leq t$, it holds that $|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t$ where $S$ is a solution certificate for $p$.

More generally: $c \cdot t \rightarrow f(t)$ defines separable $\Pi_p$.

$p$-Path is not separable while $p$-Dominating Set, and $p$-Feedback Vertex Set are linearly separable.
Linear Separability

Let $p$ be an graph optimization parameter and $G$ be a graph

Let $L$, $C$, and $R$ be as defined in the diagram. The subset optimization problem $\Pi_p$ is linearly separable if

$$|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t$$

for any $G$ and $S \subseteq V(G)$ such that $|C| = |\partial G(L)| \leq t$.

More generally:

$$c \cdot t \rightarrow f(t)$$

defines separable $\Pi_p$. $p$-Path is not separable while $p$-Dominating Set, and $p$-Feedback Vertex Set are linearly separable.
Linear Separability

Let \( p \) be a graph optimization parameter and \( G \) be a graph

\[ L \quad C = \text{boundary of } L \quad R = \text{Rest of the graph} \]

The subset optimization problem \( \Pi_p \) is *linearly separable* if, for any graph \( G \) and \( L \subseteq V(G) \) such that \(|C| = |\partial_G(L)| \leq t\), it holds that

\[
|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t
\]
Linear Separability

Let $p$ be a graph optimization parameter and $G$ be a graph.

The subset optimization problem $\Pi_p$ is linearly separable if, for any graph $G$ and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t$$

The subset optimization problem $\Pi_p$ is linearly separable if, for any graph $G$ and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t$$

where $S$ is a solution certificate for $p$. 
Linear Separability

Let $p$ be a graph optimization parameter and $G$ be a graph.

The subset optimization problem $\Pi_p$ is linearly separable if, for any graph $G$ and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t$$

where $S$ is a solution certificate for $p$.

More generally: $c \cdot t \rightarrow f(t)$ defines separable $\Pi_p$. 
Linear Separability

Let $p$ be a graph optimization parameter and $G$ be a graph.

The subset optimization problem $\Pi_p$ is \textit{linearly separable} if, for any graph $G$ and $L \subseteq V(G)$ such that $|C| = |\partial_G(L)| \leq t$, it holds that

$$|S \cap L| - c \cdot t \leq p(G[L]) \leq |S \cap L| + c \cdot t$$

where $S$ is a solution certificate for $p$.

More generally: $c \cdot t \rightarrow f(t)$ defines \textit{separable} $\Pi_p$.$p$-\textsc{Path} is not separable while $p$-\textsc{Dominating Set} and $p$-\textsc{Feedback Vertex Set} are linearly separable.
Linear Separability

Let $\mathbf{p}$ be an graph optimization parameter and $G$ be a graph

The subset optimization problem $\Pi_p$ 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: $c \cdot t \rightarrow f(t)$ defines separable $\Pi_p$

$p$-Path is not separable while

$p$-Dominating Set, and $p$-Feedback Vertex Set are linearly separable.
Conditions for the existence of linear kernels

1. **Combinatorial**
   
   If $p(G) \leq k$, then $G$ has an $(O(k),O(1))$-protrusion decomposition.

2. **Algorithmic**
   
   $\Pi_p$ has a protrusion replacement family.

---

$\rightarrow$

SQGM/SQGC + Linear separability + bidimensionality →

1. [Fomin, Lokshtanov, Saurabh, Thilikos, SODA 2011]

   CMSO-expressibility + Linear separability →

2. [Fomin, Lokshtanov, Saurabh, Thilikos, 2015]
Conditions for the existence of linear kernels

1. **Combinatorial** If \( p(G) \leq k \), then \( G \) has an \( (O(k), O(1)) \)-protrusion decomposition.

2. **Algorithmic** \( \Pi_p \) has a protrusion replacement family.
Conditions for the existence of linear kernels

1. [Combinatorial] If $p(G) \leq k$, then $G$ has an $(O(k), O(1))$-protrusion decomposition.

2. [Algorithmic] $\Pi_p$ has a protrusion replacement family.

$\Rightarrow$ SQGM/SQGC + Linear separability + bidimensionality $\rightarrow$ 1.

[Fomin, Lokshtanov, Saurabh, Thilikos, SODA 2011]
Conditions for the existence of linear kernels

1. \([\textit{Combinatorial}]\) If \(p(G) \leq k\), then \(G\) has an \((O(k), O(1))\)-protrusion decomposition.

2. \([\textit{Algorithmic} P_p]\) has a protrusion replacement family.

\(-\) \(\text{SQGM/SQGC} + \text{Linear separability} + \text{bidimensionality} \rightarrow 1.\)

[Fomin, Lokshtanov, Saurabh, Thilikos, SODA 2011]

\(-\) \(\text{CMSO-expressibility} + \text{Linear separability} \rightarrow 2.\)

[Fomin, Lokshtanov, Saurabh, Thilikos, 2015]
We comment the proof of the first fact:

- **SQGM/SQGC** + Linear separability + bidimensionality $\xrightarrow{A+B} 1$. 

**Definition:** $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$, i.e., a certificate for $\eta$-twm($G$) $\leq k$.

**Lemma A:** Assume that:
1. $G$ is a graph class with the SQGM/SQGC property
2. $\Pi_p$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$p(G) \leq k \Rightarrow G$ has a treewidth $\eta$-modulator $S$ where $|S| \leq 2 \cdot k$.

**Remark:** the bidimensionality condition: $p - 1(k) = \sqrt{k}$ is necessary here!

**Lemma B:** Assume that:
1. $G$ is a graph class with the SQGM/SQGC property.
2. $G$ has a treewidth $\eta$-modulator $S$ for some positive integer $\eta$

Then there exists an integer $r$ such that $G$ has $(2 \cdot |S|, r)$-protrusion decomposition.
We comment the proof of the first fact:

\[ \text{SQGM/SQGC} + \text{Linear separability} + \text{bidimensionality} \xrightarrow{\text{A}+\text{B}} 1. \]

**Definition:** $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta\cdot\text{twm}(G) \leq k$
We comment the proof of the first fact:

\[ \text{SQGM/SQGC} \ + \ \text{Linear separability} \ + \ \text{bidimensionality} \xrightarrow{\text{A+B}} 1. \]

**Definition:** $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta\text{-twm}(G) \leq k$
We comment the proof of the first fact:

▶ SQGM/SQGC + Linear separability + bidimensionality $\xrightarrow{A+B} 1$.

**Definition:** $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta$-$\text{twm}(G) \leq k$

---

**Lemma A:** Assume that:

1. $G$ is a graph class with the SQGM/SQGC property
2. $\Pi_p$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$p(G) \leq k \Rightarrow G$ has a treewidth $\eta$-modulator $S$ where $|S| \leq 2 \cdot k$.

**Remark:** the bidimensionality condition: $p-1(k) = \sqrt{k}$ is necessary here!

**Lemma B:** Assume that:

1. $G$ is a graph class with the SQGM/SQGC property.
2. $G$ has a treewidth $\eta$-modulator $S$ for some positive integer $\eta$

Then there exists an integer $r$ such that $G$ has $(2 \cdot |S|, r)$-protrusion decomposition.
We comment the proof of the first fact:

- **SQGM/SQGC** + Linear separability + bidimensionality $\xrightarrow{\text{A+B}} 1$.

**Definition**: $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta \cdot \text{twm}(G) \leq k$

**Lemma A**: Assume that:

1. $G$ is a graph class with the **SQGM/SQGC** property
We comment the proof of the first fact:

\[ \text{SQGM/SQGC } + \text{ Linear separability } + \text{ bidimensionality } \xrightarrow{\text{A+B}} 1. \]

**Definition:** \( S \) is a treewidth \( \eta \)-modulator of \( G \) if \( \text{tw}(G \setminus S) \leq \eta \)

i.e., a certificate for \( \eta\text{-twm}(G) \leq k \)

---

**Lemma A:** Assume that:

1. \( G \) is a graph class with the SQGM/SQGC property
2. \( \Pi_p \) is minor/contraction-bidimensional and linear-separable
We comment the proof of the first fact:

- **SQGM/SQGC** + Linear separability + bidimensionality $\xrightarrow{A+B} 1$.

**Definition**: $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta \cdot \text{twm}(G) \leq k$

---

**Lemma A**: Assume that:

1. $\mathcal{G}$ is a graph class with the SQGM/SQGC property
2. $\Pi_p$ 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** $\rightarrow$ \textbf{1}.

**Definition:** $S$ is a tree-width $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta\text{-twm}(G) \leq k$

---

**Lemma A:** Assume that:

1. $G$ is a graph class with the SQGM/SQGC property
2. $\Pi_p$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$p(G) \leq k \Rightarrow G$ has a treewidth $\eta$-modulator $S$ where $|S| \leq 2 \cdot k$. 

Remark: The bidimensionality condition $p - 1(k) = \sqrt{k}$ is necessary here!
We comment the proof of the first fact:

\[ \text{SQGM/SQGC} + \text{Linear separability} + \text{bidimensionality} \xrightarrow{\text{A+B}} 1. \]

**Definition:** \( S \) is a treewidth \( \eta \)-modulator of \( G \) if \( \text{tw}(G \setminus S) \leq \eta \)
i.e., a certificate for \( \eta\text{-twm}(G) \leq k \)

---

**Lemma A:** Assume that:

1. \( G \) is a graph class with the \( \text{SQGM/SQGC} \) property
2. \( \Pi_p \) is minor/contraction-bidimensional and linear-separable

Then there exists an integer \( \eta \geq 0 \) such that the following holds:

\( p(G) \leq k \Rightarrow G \) has a treewidth \( \eta \)-modulator \( S \) where \( |S| \leq 2 \cdot k \).

---

**Remark:** the bidimensionality condition: \( p^{-1}(k) = \sqrt{k} \) is necessary here!
We comment the proof of the first fact:

- **SQGM/SQGC + Linear separability + bidimensionality** \( A+B \rightarrow 1. \)

**Definition**: \( S \) is a \( \eta \)-modulator of \( G \) if \( \text{tw}(G \setminus S) \leq \eta \)
i.e., a certificate for \( \eta\text{-twm}(G) \leq k \)

---

**Lemma A**: Assume that:

1. \( G \) is a graph class with the **SQGM/SQGC** property
2. \( \Pi_p \) is minor/contraction-bidimensional and linear-separable

Then there exists an integer \( \eta \geq 0 \) such that the following holds:

\[ p(G) \leq k \Rightarrow G \text{ has a treewidth } \eta\text{-modulator } S \text{ where } |S| \leq 2 \cdot k. \]

---

**Remark**: the bidimensionality condition: \( p^{-1}(k) = \sqrt{k} \) is necessary here!

---

**Lemma B**: Assume that:
We comment the proof of the first fact:

- \textbf{SQGM/SQGC} + Linear separability + bidimensionality $\xrightarrow{\text{A+B}} 1$.

**Definition:** $S$ is a a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$

i.e., a certificate for $\eta\text{-twm}(G) \leq k$

---

**Lemma A:** Assume that:

1. $G$ is a graph class with the \textbf{SQGM/SQGC} property
2. $\Pi_\mathcal{P}$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$p(G) \leq k \Rightarrow G$ has a treewidth $\eta$-modulator $S$ where $|S| \leq 2 \cdot k$.

---

**Remark:** the bidimensionality condition: $p^{-1}(k) = \sqrt{k}$ is necessary here!

---

**Lemma B:** Assume that:

1. $G$ is a graph class with the \textbf{SQGM/SQGC} property.
We comment the proof of the first fact:

▶ SQGM/SQGC + Linear separability + bidimensionality $\xrightarrow{A+B} 1$.

**Definition:** $S$ is a treewidth $\eta$-modulator of $G$ if $\text{tw}(G \setminus S) \leq \eta$
i.e., a certificate for $\eta\text{-twm}(G) \leq k$

---

**Lemma A:** Assume that:

1. $\mathcal{G}$ is a graph class with the SQGM/SQGC property
2. $\Pi_p$ is minor/contraction-bidimensional and linear-separable

Then there exists an integer $\eta \geq 0$ such that the following holds:

$p(G) \leq k \Rightarrow G$ has a treewidth $\eta$-modulator $S$ where $|S| \leq 2 \cdot k$.

---

**Remark:** the bidimensionality condition: $p^{-1}(k) = \sqrt{k}$ is necessary here!

---

**Lemma B:** Assume that:

1. $\mathcal{G}$ is a graph class with the SQGM/SQGC property.
2. $G$ has a treewidth $\eta$-modulator $S$ for some positive integer $\eta$.
We comment the proof of the first fact:

\[ \text{SQGM/SQGC} + \text{Linear separability} + \text{bidimensionality} \xrightarrow{A+B} 1. \]

**Definition:** \( S \) is a \( \eta \)-modulator of \( G \) if \( \text{tw}(G \setminus S) \leq \eta \)
i.e., a certificate for \( \eta \text{-twm}(G) \leq k \)

---

**Lemma A:** Assume that:

1. \( G \) is a graph class with the SQGM/SQGC property
2. \( \Pi_p \) is minor/contraction-bidimensional and linear-separable

Then there exists an integer \( \eta \geq 0 \) such that the following holds:

\[ p(G) \leq k \Rightarrow G \text{ has a treewidth } \eta \text{-modulator } S \text{ where } |S| \leq 2 \cdot k. \]

**Remark:** the bidimensionality condition: \( p^{-1}(k) = \sqrt{k} \) is necessary here!

---

**Lemma B:** Assume that:

1. \( G \) is a graph class with the SQGM/SQGC property.
2. \( G \) has a treewidth \( \eta \)-modulator \( S \) for some positive integer \( \eta \)

Then there exists an integer \( r \) such that \( G \) has \((2 \cdot |S|, r)\)-protrusion decomposition.
Bidimensionality and kernels.

**Theorem:** Let $\Pi_p$ be a subset optimization parameterized problem that

1. is CMSO-expressible
2. is minor- (resp. contraction-) bidimensional,
3. is linearly separable,
4. is restricted to some $\text{SQGM}/\text{SQGC}$-graph class

Then $\Pi_p$ admits a linear kernel.
Bidimensionality and kernels.

**Theorem:** Let $\Pi_p$ be a subset optimization parameterized problem that

i. is CMSO-expressible
**Theorem:** Let $\Pi_p$ be a subset optimization parameterized problem that

i. is CMSO-expressible

ii. is minor- (resp. contraction-) bidimensional,
Bidimensionality and kernels.

**Theorem:** Let $\Pi_p$ 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 $\Pi_p$ 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 $\text{SQGM}/\text{SQGC}$-graph class
Bidimensionality and kernels.

**Theorem:** Let $\Pi_p$ 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 $\Pi_p$ admits a linear kernel.
Bidimensionality and approximation.

- Just for the history we also mention the following:

\[ \text{Theorem: Let } \Pi_p \text{ be a subset optimization parameterized problem that}
\]
\[ \text{i. is } \approx \text{CMSO-expressible}
\]
\[ \text{ii. is minor- (resp. contraction-) bidimensional,}
\]
\[ \text{iii. is linearly separable,}
\]
\[ \text{iv. is restricted to some}
\]
\[ \text{SQGM/SQGC -graph class}
\]
\[ \text{Then } \Pi_p \text{ admits an EPTAS}
\]
\[ \text{EPTAS = (Efficient Polynomial-Time Approximation Scheme)}
\]
\[ \text{[Demaine, Hajiaghay, SODA 2005]}
\]
\[ \text{[Fomin, Lokshtanov, Raman, Saurabh, SODA 2011]}
\]
Bidimensionality and approximation.

- Just for the history we also mention the following:

---

**Theorem:** Let $\Pi_p$ be a subset optimization parameterized problem that

- is $\approx$ CMSO-expressible,
- is minor- (resp. contraction-) bidimensional,
- is linearly separable,
- is restricted to some $\text{SQGM}/\text{SQGC}$-graph class

Then $\Pi_p$ admits an EPTAS.

Demaine, Hajiaghay, SODA 2005

Fomin, Lokshtanov, Raman, Saurabh, SODA 2011
Bidimensionality and approximation.

Just for the history we also mention the following:

**Theorem:** Let $\Pi_p$ 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 $\text{SQGM}/\text{SQGC}$-graph class

Then $\Pi_p$ admits an EPTAS $\text{EPTAS} = (\text{Efficient Polynomial-Time Approximation Scheme})$ [Demaine, Hajiaghay, SODA 2005] [Fomin, Lokshtanov, Raman, Saurabh, SODA 2011]
Bidimensionality and approximation.

- Just for the history we also mention the following:

---

**Theorem**: Let $\Pi_p$ 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 $\Pi_p$ be a subset optimization parameterized problem that
  i. is \(\approx\) CMSO-expressible
  
  ii. is minor- (resp. contraction-) bidimensional,

  iii. is linearly separable,

Then $\Pi_p$ admits an EPTAS \(E\text{-}\text{PTAS} = (E\text{-}f\text{ficient\,}P\text{-olynomial\,-\,T\text{ime\,A\text{pproximation\,S\text{cheme}}})}$

[Demaine, Hajiaghay, SODA 2005]
[Fomin, Lokshtanov, Raman, Saurabh, SODA 2011]
Bidimensionality and approximation.

▶ Just for the history we also mention the following:

Theorem: Let $\Pi_p$ 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 $\Pi_p$ 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 $\Pi_p$ admits an EPTAS

---
Bidimensionality and approximation.

Just for the history we also mention the following:

**Theorem**: Let $\Pi_p$ 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 $\Pi_p$ admits an EPTAS

EPTAS = (Efficient Polynomial-Time Approximation Scheme)

[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 $G$):

1. \[p(G) \leq k \Rightarrow tw(G) \leq f(k)\]
   i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. \[p(G) \leq k\text{ can be checked in } g(tw(G)) \cdot n^{O(1)} \text{ steps}\]

Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $tw(G) > f(k)$ then we declare VICTORY! (enemy surrenders!)
- if $tw(G) \leq f(k)$ then the CAVALRY comes! (DP algorithms or just Courcelle's th.)

▶ What about when YES/NO-instances of $\Pi_p$ do not have bounded treewidth?

- In this case we have to FIGHT!!!! (until the CAVALRY comes or enemy surrenders)

▶ For many problems, instances of big enough treewidth may contain part whose removal does not change the answer to the original question.
General algorithmic strategy (so far):

We need the following two facts (for suitable functions $f$ and $G$):

1. **[Combinatorial]** $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

▶ Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

• If $tw(G) > f(k)$ then we declare VICTORY! (enemy surrenders!)

• if $tw(G) \leq f(k)$ then the CAVALRY comes! (DP algorithms or just Courcelle's th.)

▶ What about when YES/NO-instances of $\Pi_p$ do not have bounded treewidth?

• In this case we have to FIGHT!!!!! (until the CAVALRY comes or enemy surrenders)

For many problems, instances of big enough treewidth may contain parts whose removal does not change the answer to the original question.
General algorithmic strategy (so far):

We need the following two facts (for suitable functions $f$ and $G$):

1. **[Combinatorial]** $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
   
   i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. **[Algorithmic]** One can check whether $p(G) \leq k$ in $g(tw(G)) \cdot n^{O(1)}$ steps

---

What about when YES/NO-instances of $\Pi_p$ do not have bounded treewidth?

• In this case we have to FIGHT!!!!! (until the CAVALRY comes or enemy surrenders)

• For many problems, instances of big enough treewidth may contain parts whose removal does not change the answer to the original question.
General algorithmic strategy (so far):

We need the following two facts (for suitable functions $f$ and $G$):

1. [Combinatorial] $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $p(G) \leq k$ in $g(tw(G)) \cdot n^{O(1)}$ 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] $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
   i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $p(G) \leq k$ in $g(tw(G)) \cdot n^{O(1)}$ steps

- Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:
  - If $tw(G) > f(k)$ then we declare VICTORY!  (enemy surrenders!)
  - If $tw(G) \leq f(k)$ then the CAVALRY comes! (DP algorthims or just Courcelle's th.)

▶ What about when YES/NO-instances of $\Pi_p$ do not have bounded treewidth?
  - In this case we have to FIGHT!!!!! (until the CAVALRY comes or enemy surrenders)

  - For many problems, instances of big enough treewidth may contain part whose
    removal does not change the answer to the original question.
General algorithmic strategy (so far):

We need the following two facts (for suitable functions $f$ and $G$):

1. **[Combinatorial]** $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. **[Algorithmic]** One can check whether $p(G) \leq k$ in $g(tw(G)) \cdot n^{O(1)}$ steps

Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $tw(G) > f(k)$ then we declare VICTORY! (enemy surrenders!)
- If $tw(G) \leq f(k)$ then the CAVALRY comes! (DP algorithms or just Courcelle’s th.)

What about when YES/NO-instances of $\Pi_p$ do not have bounded treewidth?
- In this case we have to FIGHT!!!!! (until the CAVALRY comes or enemy surrenders)
- For many problems, instances of big enough treewidth may contain parts whose removal does not change the answer to the original question.
General algorithmic strategy (so far):

We need the following two facts (for suitable functions \( f \) and \( G \)):

1. [Combinatorial] \( p(G) \leq k \Rightarrow tw(G) \leq f(k) \)
i.e., YES/NO-instances of \( \Pi_p \) have treewidth \( \leq f(k) \)

2. [Algorithmic] One can check whether \( p(G) \leq k \) in \( g(tw(G)) \cdot n^{O(1)} \) steps

   ▶ Then we have an FPT-algorithm running in \( g(f(k)) \cdot n^{O(1)} \) steps because:
   
   • If \( tw(G) > f(k) \) then we declare VICTORY! (enemy surrenders!)
   • if \( tw(G) \leq f(k) \) then the CAVALRY comes! (DP algorithms or just Courcelle’s th.)

   ▶ What about when YES/NO-instances of \( \Pi_p \) do not have bounded treewidth?
General algorithmic strategy (so far):

We need the following two facts (for suitable functions $f$ and $G$):

1. [Combinatorial] $p(G) \leq k \Rightarrow tw(G) \leq f(k)$
i.e., YES/NO-instances of $\Pi_p$ have treewidth $\leq f(k)$

2. [Algorithmic] One can check whether $p(G) \leq k$ in $g(tw(G)) \cdot n^{O(1)}$ steps

Then we have an FPT-algorithm running in $g(f(k)) \cdot n^{O(1)}$ steps because:

- If $tw(G) > f(k)$ then we declare VICTORY! (enemy surrenders!)
- if $tw(G) \leq f(k)$ then the CAVALRY comes! (DP algorithms or just Courcelle’s th.)

What about when YES/NO-instances of $\Pi_p$ do not have bounded treewidth?

- In this case we have to FIGHT!!!!! (until the CAVALRY comes or enemy surrenders)
General algorithmic strategy (so far):

We need the following two facts (for suitable functions \( f \) and \( G \)):

1. **Combinatorial** \( p(G) \leq k \Rightarrow tw(G) \leq f(k) \)
   i.e., YES/NO-instances of \( \Pi_p \) have treewidth \( \leq f(k) \)

2. **Algorithmic** One can check whether \( p(G) \leq k \) in \( g(tw(G)) \cdot n^{O(1)} \) steps

   - Then we have an FPT-algorithm running in \( g(f(k)) \cdot n^{O(1)} \) steps because:
     - If \( tw(G) > f(k) \) then we declare VICTORY! (enemy surrenders!)
     - if \( tw(G) \leq f(k) \) then the CAVALRY comes! (DP algorithms or just Courcelle’s th.)

   - What about when YES/NO-instances of \( \Pi_p \) do not have bounded treewidth?
     - In this case we have to FIGHT!!!!! (until the CAVALRY comes or enemy surrenders)

   - 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:

Odd Cycle Packing

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

Parameter:
k,

Question:
Does $G$ contain $k$ 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 $G$ contain $k$ mutually vertex-disjoint odd cycles?
Consider the following problem:

\[ \text{p-Odd Cycle Packing} \]

**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:

\textbf{p-Odd Cycle Packing}

\textit{Instance:} A graph \(G\) and an integer \(k \geq 0\).

\textit{Parameter:} \(k\),

\textit{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:

\[ p\text{-Odd Cycle Packing} \]

*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:

\[ p\text{-Odd Cycle Packing} \]

**Instance:** A graph \( G \) and an integer \( k \geq 0 \).

**Parameter:** \( k \),

**Question:** Does \( G \) contain \( k \) 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 $G$ contain $k$ 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:

<table>
<thead>
<tr>
<th><strong>p-ODD CYCLE PACKING</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> A graph $G$ and an integer $k \geq 0$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $k$,</td>
</tr>
<tr>
<td><strong>Question:</strong> Does $G$ contains $k$ mutually vertex-disjoint odd cycles?</td>
</tr>
</tbody>
</table>

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:

\[ \text{p-Odd Cycle Packing} \]

**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!

![Grid Graph](image_url)
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!

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!

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 perimeties 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 $\iff$ 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 $\Leftarrow$ direction is trivial: if $G \setminus x$ has $k$ odd disjoint cycles, so does $G$.

For the $\Rightarrow$ 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 $\iff$ 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 $\Omega$ 4 times and $C$ crosses perimetry $P$ 5 times. We argue that none of these $k$ cycles can cross the inner cycle $\Omega$, thus $x$ is irrelevant!
Suppose, to the contrary, that some cycle $C$ crosses the inner cycle $\Omega$.
Consider an “extremal” chord $X$: one of the two paths of $\Omega$ does not contain any other endpoint of a chord.
Suppose, to the contrary, that some cycle $C$ crosses the inner cycle $\Omega$.
Consider an “extremal” chord! This defines a (same parity) segment $R$ of $\Omega$.
By minimality $R$ should be met by some
Suppose, to the contrary, that some cycle $C$ crosses the inner cycle $\Omega$.

Consider an “extremal” chord! This defines a (same parity) segment $R$ of $\Omega$.

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 $\Omega$. Thus $x$ is irrelevant.
To find the irrelevant vertex \( x \) can be done in polynomial time!
1. If $G$ has treewidth $O(k^{3/2})$ then CAVALRY comes: DP takes $2^{O(tw(G))} \cdot n$ steps.
General scheme

1. If $G$ has treewidth $O(k^{3/2})$ then CAVALRY comes: DP takes $2^{O(tw(G))} \cdot n$ steps.

2. Otherwise check whether we have VICTORY (all $k$ subwalls are non-bipartite)!

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 further!)

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]
General scheme

1. If $G$ has treewidth $O(k^{3/2})$ then CAVALRY comes: DP takes $2^{O(tw(G))} \cdot n$ 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]
General scheme

1. If $G$ has treewidth $O(k^{3/2})$ then CAVALRY comes: DP takes $2^{O(tw(G))} \cdot n$ 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
1. If $G$ has treewidth $O(k^{3/2})$ then **CAVALRY** comes: DP takes $2^{O(\text{tw}(G))} \cdot n$ 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 $2^{O(tw(G))} \cdot n$ 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 further!)

The general $p$-Odd Cycle Packing problem is in FPT

[Kawarabayashi, Reed, STOC 2010]
1. If $G$ has treewidth $O(k^{3/2})$ then \textbf{CAVALRY} comes: DP takes $2^{O(tw(G))} \cdot n$ steps.
2: Otherwise check whether we have \textbf{VICTORY} (all $k$ subwalls are non-bipartite)!
3: Otherwise \textbf{FIGHT}: find an irrelevant vertex $x$, set $G \leftarrow G \setminus x$ and go to 1.

The above proves that $p$-\textbf{Planar Odd Cycle Packing} $\in 2^{O(k^{3/2})}$-\textbf{FPT}

► All the above arguments extend for graphs of bounded genus! (and further!)
The general $p$-\textbf{Odd Cycle Packing} problem is in \textbf{FPT}
[Kawarabayashi, Reed, STOC 2010]

► The same ideas prove: $p$-\textbf{Planar Odd Induced Cycle Packing} $\in 2^{O(k^{3/2})}$-\textbf{FPT}
while general $p$-\textbf{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.

\[
\begin{array}{|l|}
\hline
\text{\textbf{\textit{p-Minor Containment}}} \\
\text{\textit{Instance:}} \text{ two graphs } G \text{ and } H. \\
\text{\textit{Parameter:}} \ k = |V(H)| \\
\text{\textit{Question:}} \ H \leq G? \\
\hline
\end{array}
\]

\[
\begin{array}{|l|}
\hline
\text{\textbf{\textit{p-Disjoint Paths}}} \\
\text{\textit{Instance:}} \ A \text{ graph } G \text{ and a sequence of pairs of terminals } (s_1, t_1), \ldots, (s_k, t_k). \\
\text{\textit{Parameter:}} \ k. \\
\text{\textit{Question:}} \text{ Are there } k \text{ pairwise vertex disjoint paths } P_1, \ldots, P_k \text{ in } G \text{ such that for every } i \in \{1, \ldots, k\}, P_i \text{ has endpoints } s_i \text{ and } t_i? \\
\hline
\end{array}
\]

Challenge: go further than planar graphs.
Irrelevant vertex technique

Introduced by [Robertson & Seymour GM-XIII] for proving that the following belong in FPT.

<table>
<thead>
<tr>
<th>p-Minor Containment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> two graphs $G$ and $H$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $k =</td>
</tr>
<tr>
<td><strong>Question:</strong> $H \leq G$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p-Disjoint Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> A graph $G$ and a sequence of pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $k$.</td>
</tr>
<tr>
<td><strong>Question:</strong> 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 $t_i$?</td>
</tr>
</tbody>
</table>

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.

\[ p-\text{Minor Containment} \]

\textbf{Instance:} two graphs \( G \) and \( H \).

\textbf{Parameter:} \( k = |V(H)| \)

\textbf{Question:} \( H \leq G \)?

\[ p-\text{Disjoint Paths} \]

\textbf{Instance:} A graph \( G \) and a sequence of pairs of terminals \( (s_1, t_1), \ldots, (s_k, t_k) \).

\textbf{Parameter:} \( k \).

\textbf{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!

La grand traverse...