Outline of the talk

1. Definition and simple properties

2. Dynamic programming on tree decompositions
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3. Brambles and duality

4. Computing treewidth
1 Definition and simple properties

2 Dynamic programming on tree decompositions
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3 Brambles and duality

4 Computing treewidth
The multiples origins of treewidth

- **1972**: Bertelè and Brioschi (dimension).
- **1976**: Halin (*S*-functions of graphs).
- **1984**: Arnborg and Proskurowski (partial *k*-trees).
- **1984**: Robertson and Seymour (treewidth).
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a tree.
A measure of the similarity with a tree

**Treewidth** measures the (topological) *similarity* of a graph with a *tree*.

Natural candidates:

- Number of cycles.
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) **similarity** of a graph with a **tree**.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) *similarity* of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a **tree**.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) **similarity** of a graph with a **tree**.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
A measure of the similarity with a tree

**Treewidth** measures the (topological) similarity of a graph with a tree.

Natural candidates:
- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then \textit{iteratively} adding a vertex connected to a $k$-clique.
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

Example of a 2-tree:

[Figure by Julien Baste]
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.
Treewidth via $k$-trees

For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

Example of a 2-tree:

[Figure by Julien Baste]
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

Example of a 2-tree:

[Figure by Julien Baste]
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

Example of a 2-tree:

[Figure by Julien Baste]
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then \textit{iteratively} adding a vertex connected to a $k$-clique.

Example of a 2-tree:
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a $k$-clique.
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

A partial $k$-tree is a subgraph of a $k$-tree.
For \( k \geq 1 \), a \( k \)-tree is a graph that can be built starting from a \((k + 1)\)-clique and then iteratively adding a vertex connected to a \( k \)-clique.

A partial \( k \)-tree is a subgraph of a \( k \)-tree.

**Treewidth** of a graph \( G \), denoted \( tw(G) \): smallest integer \( k \) such that \( G \) is a partial \( k \)-tree.
Treewidth via $k$-trees

For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

A partial $k$-tree is a subgraph of a $k$-tree.

**Treewidth** of a graph $G$, denoted $\text{tw}(G)$: smallest integer $k$ such that $G$ is a partial $k$-tree.

Invariant that measures the topological resemblance of a graph to a forest.
For $k \geq 1$, a $k$-tree is a graph that can be built starting from a $(k + 1)$-clique and then iteratively adding a vertex connected to a $k$-clique.

A partial $k$-tree is a subgraph of a $k$-tree.

**Treewidth** of a graph $G$, denoted $\text{tw}(G)$: smallest integer $k$ such that $G$ is a partial $k$-tree.

Invariant that measures the topological resemblance of a graph to a forest.

Construction suggests the notion of tree decomposition: small separators.
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where
  
  $T$ is a **tree**, and
  
  $X_t \subseteq V(G)$ $\forall t \in V(T)$ (**bags**),

  satisfying the following:

  $\bigcup_{t \in V(T)} X_t = V(G)$,

  $\forall \{u, v\} \in E(G)$, there exists $t \in V(T)$ with $\{u, v\} \subseteq X_t$.

  $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$. 

  **Width of a tree decomposition**: $\max_{t \in V(T)} |X_t| - 1$.

  **Treewidth of a graph** $G$, $\text{tw}(G)$:

  minimum width of a tree decomposition of $G$.
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where
  - $T$ is a tree, and
  - $X_t \subseteq V(G)$ $\forall t \in V(T)$ (bags),

  satisfying the following:

  $\bigcup_{t \in V(T)} X_t = V(G)$,
  $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$,
  $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$. 

- **Width of a tree decomposition**: $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth of a graph $G$**, $\text{tw}(G)$:

  minimum width of a tree decomposition of $G$. 

An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where
  
  $T$ is a tree, and
  
  $X_t \subseteq V(G) \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where $T$ is a tree, and $X_t \subseteq V(G)$ $\forall t \in V(T)$ (bags), satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$. 

  **Width of a tree decomposition**:
  
  $\max_{t \in V(T)} |X_t| - 1$.

  **Treewidth of a graph** $G$, $\text{tw}(G)$:
  
  minimum width of a tree decomposition of $G$. 

An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where
  
  $T$ is a tree, and
  
  $X_t \subseteq V(G) \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$. 

  **Width of a tree decomposition**:

  $\max_{t \in V(T)} |X_t| - 1$.

  **Treewidth of a graph $G$**, $\text{tw}(G)$:

  minimum width of a tree decomposition of $G$. 

An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where
  - $T$ is a tree, and
  - $X_t \subseteq V(G)$ $\forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- **Width** of a tree decomposition:

  $\max_{t \in V(T)} |X_t| - 1$. 
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where
  $T$ is a tree, and
  $X_t \subseteq V(G) \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- **Width** of a tree decomposition:

  $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $tw(G)$: minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where $T$ is a **tree**, and
  $X_t \subseteq V(G) \forall t \in V(T)$ (**bags**),
  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a **connected** subtree of $T$.

- **Width** of a tree decomposition:

  $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $tw(G)$:

  minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where
  
  $T$ is a tree, and
  
  $X_t \subseteq V(G) \; \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G),$
  
  - $\forall \{u, v\} \in E(G), \exists t \in V(T)$ with $\{u, v\} \subseteq X_t.$
  
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- Width of a tree decomposition:
  
  $\max_{t \in V(T)} |X_t| - 1.$

- Treewidth of a graph $G$, $tw(G)$:
  
  minimum width of a tree decomposition of $G.$
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where $T$ is a tree, and
  
  $X_t \subseteq V(G) \ \forall t \in V(T)$ (bags),
  
  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  
  - $\forall \{u,v\} \in E(G), \exists t \in V(T)$ with $\{u,v\} \subseteq X_t$.
  
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

  - **Width** of a tree decomposition: $\max_{t \in V(T)} |X_t| - 1$.

  - **Treewidth** of a graph $G$, $tw(G)$: minimum width of a tree decomposition of $G$. 

An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:

  pair $(T, \{X_t \mid t \in V(T)\})$, where $T$ is a tree, and $X_t \subseteq V(G)$ for all $t \in V(T)$ (bags), satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G), \exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- **Width** of a tree decomposition: $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $tw(G)$: minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where
  - $T$ is a tree, and
  - $X_t \subseteq V(G)$ $\forall t \in V(T)$ (bags),

  satisfying the following:
  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- **Width** of a tree decomposition:
  - $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $\text{tw}(G)$: minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where $T$ is a tree, and $X_t \subseteq V(G)$ $\forall t \in V(T)$ (bags), satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- **Width** of a tree decomposition: $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $tw(G)$: minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth:

- **Tree decomposition** of a graph $G$:
  
  Pair $(T, \{X_t \mid t \in V(T)\})$, where $T$ is a tree, and $X_t \subseteq V(G)$ \(\forall t \in V(T)\) (bags), satisfying the following:
  
  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

- **Width** of a tree decomposition:
  
  $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $tw(G)$: minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where
  
  $T$ is a **tree**, and
  
  $X_t \subseteq V(G) \ \forall t \in V(T)$ (**bags**),
  
  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  
  - $\forall \{u, v\} \in E(G), \exists t \in V(T)$
    
    with $\{u, v\} \subseteq X_t$.
  
  - $\forall v \in V(G)$, bags containing $v$
    
    define a **connected** subtree of $T$.

- **Width** of a tree decomposition:
  
  $\max_{t \in V(T)} |X_t| - 1$.

- **Treewidth** of a graph $G$, $tw(G)$:
  
  minimum width of a tree decomposition of $G$. 
An equivalent (and more common) definition of treewidth

- **Tree decomposition** of a graph $G$:
  
  pair $(T, \{X_t \mid t \in V(T)\})$, where
  
  - $T$ is a tree, and
  
  - $X_t \subseteq V(G)$ $\forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.
  
  - $\forall v \in V(G)$, bags containing $v$ define a connected subtree of $T$.

  - **Width** of a tree decomposition: $\max_{t \in V(T)} |X_t|-1$.

  - **Treewidth** of a graph $G$, $\text{tw}(G)$: minimum width of a tree decomposition of $G$. 

---

**Diagram:**

- Nodes represent vertices $u, v, w, s, t, z$.
- Edges connect these nodes.
- The tree $T$ is depicted with leaves labeled $X_t, X_s, X_v, X_u, X_w, X_z$.
Treewidth measures the tree-likeness of a graph
Treewidth measures the tree-likeness of a graph.
Treewidth measures the tree-likeness of a graph
Treewidth measures the tree-likeness of a graph
Treewidth measures the tree-likeness of a graph.
Treewidth measures the tree-likeness of a graph.
Treewidth measures the tree-likeness of a graph.
Every bag of a tree decomposition is a separator

Let \((T, \mathcal{X} = \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).
Every bag of a tree decomposition is a separator

Let \((T, \mathcal{X} = \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
Every bag of a tree decomposition is a separator

Let \((T, \mathcal{X} = \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
- For every edge \(\{t_1, t_2\} \in E(T)\), \(X_{t_1} \cap X_{t_2}\) is a separator in \(G\).
Every bag of a tree decomposition is a separator

Let \((T, \mathcal{X} = \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
- For every edge \(\{t_1, t_2\} \in E(T)\), \(X_{t_1} \cap X_{t_2}\) is a separator in \(G\).
Every bag of a tree decomposition is a separator

Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph $G$.

- For every $t \in V(T)$, $X_t$ is a separator in $G$.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in $G$. 

![Diagram of tree decomposition with separators and vertices]
Every bag of a tree decomposition is a separator

Let \((T, \mathcal{X} = \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
- For every edge \(\{t_1, t_2\} \in E(T)\), \(X_{t_1} \cap X_{t_2}\) is a separator in \(G\).
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique.
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$. Proof by induction on $t$. True for $t \leq 2$. Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Every clique is entirely contained in some bag

Let \( G \) be a graph, \( (T, \mathcal{X}) \) be a tree decomposition of \( G \), and let \( K \subseteq V(G) \) be a clique. Then there exists a bag \( X_t \in \mathcal{X} \) such that \( K \subseteq X_t \).

Let \( K = \{v_1, \ldots, v_t\} \). Proof by induction on \( t \).
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$. 

Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Every clique is entirely contained in some bag

Let \( G \) be a graph, \((T, \mathcal{X})\) be a tree decomposition of \( G \), and let \( K \subseteq V(G) \) be a clique. Then there exists a bag \( X_t \in \mathcal{X} \) such that \( K \subseteq X_t \).

Let \( K = \{v_1, \ldots, v_t\} \). Proof by induction on \( t \). True for \( t \leq 2 \).

Consider the subtrees in \((T, \mathcal{X})\) corresponding to vertices \( \{v_1, \ldots, v_{t-1}\} \):
Every clique is entirely contained in some bag

Let \( G \) be a graph, \((T, \mathcal{X})\) be a tree decomposition of \( G \), and let \( K \subseteq V(G) \) be a clique. Then there exists a bag \( X_t \in \mathcal{X} \) such that \( K \subseteq X_t \).

Let \( K = \{v_1, \ldots, v_t\} \). Proof by induction on \( t \). True for \( t \leq 2 \).

Consider the subtrees in \((T, \mathcal{X})\) corresponding to vertices \( \{v_1, \ldots, v_{t-1}\} \):
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, X)$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in X$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$.

Consider the subtrees in $(T, X)$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Every clique is entirely contained in some bag

Let $G$ be a graph, $(T, \mathcal{X})$ be a tree decomposition of $G$, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on $t$. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:
Examples

- If $F$ is a forest, then $tw(F) = 1$. 
Examples

- If $F$ is a forest, then $\text{tw}(F) = 1$.
- If $C$ is a cycle, then $\text{tw}(C) = 2$. 
Examples

- If $F$ is a forest, then $\text{tw}(F) = 1$.
- If $C$ is a cycle, then $\text{tw}(C) = 2$.
- If $K_n$ is the clique on $n$ vertices, then $\text{tw}(K_n) = n - 1$. 

If $K_a, b$ is the complete bipartite graph with parts of sizes $a$ and $b$, then $\text{tw}(K_{a, b}) = \min\{a, b\} + 1$. 

If $G$ is an outerplanar graph, or a series-parallel graph, then $\text{tw}(G) = 2$. 

If $G$ is a planar graph on $n$ vertices, then $\text{tw}(G) = O(\sqrt{n})$. 
Examples

- If $F$ is a forest, then $\text{tw}(F) = 1$.
- If $C$ is a cycle, then $\text{tw}(C) = 2$.
- If $K_n$ is the clique on $n$ vertices, then $\text{tw}(K_n) = n - 1$.
- If $K_{a,b}$ is the complete bipartite graph with parts of sizes $a$ and $b$, then $\text{tw}(K_{a,b}) = \min\{a, b\} + 1$. 
Examples

- If $F$ is a forest, then $\text{tw}(F) = 1$.
- If $C$ is a cycle, then $\text{tw}(C) = 2$.
- If $K_n$ is the clique on $n$ vertices, then $\text{tw}(K_n) = n - 1$.
- If $K_{a,b}$ is the complete bipartite graph with parts of sizes $a$ and $b$, then $\text{tw}(K_{a,b}) = \min\{a, b\} + 1$.
- If $G$ is an outerplanar graph, or a series-parallel graph, then $\text{tw}(G) = 2$. 
Examples

- If $F$ is a forest, then $\text{tw}(F) = 1$.
- If $C$ is a cycle, then $\text{tw}(C) = 2$.
- If $K_n$ is the clique on $n$ vertices, then $\text{tw}(K_n) = n - 1$.
- If $K_{a,b}$ is the complete bipartite graph with parts of sizes $a$ and $b$, then $\text{tw}(K_{a,b}) = \min\{a, b\} + 1$.
- If $G$ is an outerplanar graph, or a series-parallel graph, then $\text{tw}(G) = 2$.
- If $G$ is a planar graph on $n$ vertices, then $\text{tw}(G) = \mathcal{O}(\sqrt{n})$. 
Why treewidth?

Treewidth is important for (at least) 3 different reasons:
Why treewidth?

Treewidth is **important** for (at least) 3 different reasons:

1. Treewidth is a fundamental **combinatorial tool** in graph theory: key role in the *Graph Minors* project of Robertson and Seymour.
Why treewidth?

Treewidth is important for (at least) 3 different reasons:

1. Treewidth is a fundamental **combinatorial tool** in graph theory: key role in the Graph Minors project of Robertson and Seymour.

2. Treewidth behaves very well **algorithmically**, and algorithms parameterized by treewidth appear very often in FPT algorithms.
Why treewidth?

Treewidth is important for (at least) 3 different reasons:

1. Treewidth is a fundamental combinatorial tool in graph theory: key role in the Graph Minors project of Robertson and Seymour.

2. Treewidth behaves very well algorithmically, and algorithms parameterized by treewidth appear very often in FPT algorithms.

3. In many practical scenarios, it turns out that the treewidth of the associated graph is small (programming languages, road networks, ...).
1 Definition and simple properties

2 Dynamic programming on tree decompositions
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3 Brambles and duality

4 Computing treewidth
1. Definition and simple properties

2. **Dynamic programming on tree decompositions**
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3. Brambles and duality

4. Computing treewidth
Weighted Independent Set on trees

[slides borrowed from Christophe Paul]
Weighted Independent Set on trees

[slides borrowed from Christophe Paul]
Observations:

1. Every vertex of a tree is a separator.
2. The union of independent sets of distinct connected components is an independent set.
Let $x$ be the root of $T$, $x_1 \ldots x_\ell$ its children, $T_1, \ldots T_\ell$ subtrees of $T - x$:

- $\text{wIS}(T, x)$: maximum weighted independent set containing $x$.
- $\text{wIS}(T, \overline{x})$: maximum weighted independent set not containing $x$. 
Let $x$ be the root of $T$, $x_1 \ldots x_\ell$ its children, $T_1, \ldots T_\ell$ subtrees of $T - x$:

- $wIS(T, x)$: maximum weighted independent set containing $x$.
- $wIS(T, \overline{x})$: maximum weighted independent set not containing $x$.

\[
\begin{align*}
\text{wIS}(T, x) &= \omega(x) + \sum_{i \in [\ell]} \text{wIS}(T_i, \overline{x_i})
\end{align*}
\]
Let $x$ be the root of $T$, $x_1 \ldots x_\ell$ its children, $T_1, \ldots T_\ell$ subtrees of $T - x$:

- $\text{wIS}(T, x)$: maximum weighted independent set containing $x$.
- $\text{wIS}(T, \overline{x})$: maximum weighted independent set not containing $x$.

\[
\begin{align*}
\text{wIS}(T, x) &= \omega(x) + \sum_{i \in [\ell]} \text{wIS}(T_i, \overline{x_i}) \\
\text{wIS}(T, \overline{x}) &= \sum_{i \in [\ell]} \max\{\text{wIS}(T_i, x_i), \text{wIS}(T_i, \overline{x_i})\}
\end{align*}
\]
Dynamic programming on tree decompositions

- Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.
Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition. Starting from the leaves of the tree decomposition, a set of appropriately defined partial solutions is computed recursively until the root, where a global solution is obtained.
Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.

Starting from the leaves of the tree decomposition, a set of appropriately defined partial solutions is computed recursively until the root, where a global solution is obtained.

The way that these partial solutions are defined depends on each particular problem:
Back to tree decompositions

Let \((T, \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
- For every edge \(\{t_1, t_2\} \in E(T)\), \(X_{t_1} \cap X_{t_2}\) is a separator in \(G\).
Back to tree decompositions

Let \((T, \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
- For every edge \(\{t_1, t_2\} \in E(T)\), \(X_{t_1} \cap X_{t_2}\) is a separator in \(G\).

Notation: If we root \((T, \{X_t \mid t \in V(T)\})\), then:
Back to tree decompositions

Let \((T, \{X_t \mid t \in V(T)\})\) be a tree decomposition of a graph \(G\).

- For every \(t \in V(T)\), \(X_t\) is a separator in \(G\).
- For every edge \(\{t_1, t_2\} \in E(T)\), \(X_{t_1} \cap X_{t_2}\) is a separator in \(G\).

**Notation:** If we root \((T, \{X_t \mid t \in V(T)\})\), then:

- \(V_t\): all vertices of \(G\) appearing in bags that are descendants of \(t\).
- \(G_t = G[V_t]\).
\[ \forall S \subseteq X_t, \ IS(S, t) \ = \ \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S \]
∀ S \subseteq X_t, IS(S, t) = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S
\( \forall S \subseteq X_t, \ IS(S, t) = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S \)
∀S ⊆ X_t, IS(S, t) = maximum independent set I of G_t s.t. I ∩ X_t = S

**Lemma** If S ⊆ X_t and S_j = S ∩ X_{t_j}, then |IS(S, t) ∩ V_{t_j}| = |IS(S_j, t_j)|.
\[ \forall S \subseteq X_t, \ IS(S, t) = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S \]

**Lemma** If \( S \subseteq X_t \) and \( S_j = S \cap X_{t_j} \), then \( |IS(S, t) \cap V_{t_j}| = |IS(S_j, t_j)| \).

For contradiction: suppose \( IS(S, t) \cap V_{t_j} \) is not maximum in \( G_{t_j} \).
∀S ⊆ X_t, IS(S, t) = maximum independent set I of G_t s.t. I ∩ X_t = S

**Lemma** If S ⊆ X_t and S_j = S ∩ X_{t_j}, then |IS(S, t) ∩ V_{t_j}| = |IS(S_j, t_j)|.

For contradiction: suppose IS(S, t) ∩ V_{t_j} is not maximum in G_{t_j}.

⇒ ∃y ∈ (S \ S_j) ⊆ X_t and ∃x ∈ IS(S_j, t_j) \ X_{t_j} such that \{x, y\} ∈ E(G).
**Independent Set on tree decompositions**

\[ \forall S \subseteq X_t, \ IS(S, t) = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S \]

**Lemma** If \( S \subseteq X_t \) and \( S_j = S \cap X_{t_j} \), then \( |IS(S, t) \cap V_{t_j}| = |IS(S_j, t_j)| \).

For contradiction: suppose \( IS(S, t) \cap V_{t_j} \) is not maximum in \( G_{t_j} \).

\[ \Rightarrow \exists y \in (S \setminus S_j) \subseteq X_t \text{ and } \exists x \in IS(S_j, t_j) \setminus X_{t_j} \text{ such that } \{x, y\} \in E(G). \]

Contradiction! \( X_{t_j} \) is not a separator.
Independent Set on tree decompositions

Idea of the dynamic programming algorithm:

How to compute $|IS(S, t)|$ from $|IS(S^i_j, t_j)|$, $\forall j \in [\ell]$, $\forall S^i_j \subseteq X_{t_j}$:
Idea of the dynamic programming algorithm:

How to compute $|IS(S, t)|$ from $|IS(S_j, t_j)|$, $\forall j \in [\ell]$, $\forall S_j \subseteq X_{t_j}$:
Independent Set on tree decompositions

Idea of the dynamic programming algorithm:

How to compute $|IS(S, t)|$ from $|IS(S_j, t_j)|$, $\forall j \in [\ell]$, $\forall S_j \subseteq X_{t_j}$:

- verify that $S_j \cap X_t = S \cap X_{t_j} = S_j$ and $S_j \subseteq S_j$. 
**Independent Set on tree decompositions**

Idea of the dynamic programming algorithm:

\[
|\text{IS}(S, t)| = \begin{cases} 
|S| + \sum_{i \in [\ell]} \max\left\{ |\text{IS}(S_i^j, t_j)| - |S_j| : S_i^j \cap X_t = S_j \land S_j \subseteq S_i^j \right\} 
\end{cases}
\]

How to compute \( |\text{IS}(S, t)| \) from \( |\text{IS}(S_i^j, t_j)| \), \( \forall j \in [\ell] \), \( \forall S_i^j \subseteq X_{t_j} \):

- verify that \( S_i^j \cap X_t = S \cap X_{t_j} = S_j \) and \( S_j \subseteq S_i^j \).
- verify that \( S_i^j \) is an independent set.
**Independent Set** on tree decompositions

Idea of the dynamic programming algorithm:

![Diagram showing tree decomposition]

How to compute $|IS(S, t)|$ from $|IS(S^j, t_j)|$, $\forall j \in [\ell]$, $\forall S^j \subseteq X_{t_j}$:

- verify that $S^j \cap X_t = S \cap X_{t_j} = S_j$ and $S_j \subseteq S^j$.
- verify that $S^j$ is an independent set.

$$|IS(S, t)| = \left\{ \sum_{i \in [\ell]} \max \left\{ |S| + \left( |IS(S^j, t_j)| - |S_j| \right) : S^j \cap X_t = S_j \land S_j \subseteq S^j \text{ independent} \right\} \right\}$$
Independent Set on tree decompositions

\[ |IS(S, t)| = \left\{ \sum_{i \in [\ell]} \max \{ |S| + |IS(S_i, t_j)| - |S_j| : S_j \cap X_t = S_j \land S_j \subseteq S_i \text{ independent} \} \right\} \]

Analysis of the running time, with bags of size \( k \):
Independent Set on tree decompositions

\[ |IS(S, t)| = \begin{cases} \sum_{i \in [\ell]} \max \left\{ |S| + \left| IS(S_i^j, t_j) \right| - |S_j| : S_i^j \cap X_t = S_j \land S_j \subseteq S_i^j \text{ independent} \right\} \\ \end{cases} \]

Analysis of the running time, with bags of size \( k \):

- Computing \( IS(S, t) \): \( \mathcal{O}(2^k \cdot k^2 \cdot \ell) \).
Independent Set on tree decompositions

\[ |IS(S, t)| = \begin{cases} 
\sum_{i \in [\ell]} \max \left\{|S| + \sum_{j \in S_j} |IS(S_j^i, t_j)| - |S_j| : \right. \\
S_j^i \cap X_t = S_j \land S_j \subseteq S_j^i \text{ independent} \end{cases} \]

Analysis of the running time, with bags of size \( k \):

- Computing \( IS(S, t) \): \( \mathcal{O}(2^k \cdot k^2 \cdot \ell) \).
- Computing \( IS(S, t) \) for every \( S \subseteq X_t \): \( \mathcal{O}(2^k \cdot 2^k \cdot k^2 \cdot \ell) \).

We have to add the time in order to compute a "good" tree decomposition of the input graph (we discuss this later).
Independent Set on tree decompositions

\[
|IS(S, t)| = \left\{ \sum_{i \in [\ell]} \max \{ |S| + |IS(S^i_j, t_j)| - |S_j| : S^i_j \cap X_t = S_j \land S_j \subseteq S^i_j \text{ independent} \} \right\}
\]

Analysis of the running time, with bags of size \(k\):

- Computing \(IS(S, t)\): \(O(2^k \cdot k^2 \cdot \ell)\).
- Computing \(IS(S, t)\) for every \(S \subseteq X_t\): \(O(2^k \cdot 2^k \cdot k^2 \cdot \ell)\).
- Computing an optimal solution: \(O(4^k \cdot k^2 \cdot n)\).
Independent Set on tree decompositions

\[ |IS(S, t)| = \begin{cases} 
\sum_{i \in [\ell]} \max \{ |S| + \sum_{i \in [\ell]} \max \{ |IS(S_i^j, t_j)| - |S_j| : S_i^j \cap X_t = S_j \land S_j \subseteq S_i^j \text{ independent} \} \}
\end{cases} \]

Analysis of the running time, with bags of size \( k \):

- Computing \( IS(S, t) \): \( \mathcal{O}(2^k \cdot k^2 \cdot \ell) \).
- Computing \( IS(S, t) \) for every \( S \subseteq X_t \): \( \mathcal{O}(2^k \cdot 2^k \cdot k^2 \cdot \ell) \).
- Computing an optimal solution: \( \mathcal{O}(4^k \cdot k^2 \cdot n) \).

\( \star \) We have to add the time in order to compute a “good” tree decomposition of the input graph (we discuss this later).
Helpful tool: nice tree decompositions
Helpful tool: nice tree decompositions

A rooted tree decomposition \((T, \{X_t : t \in T\})\) of a graph \(G\) is nice if every node \(t \in V(T) \setminus \text{root}\) is of one of the following four types:
A rooted tree decomposition \((T, \{X_t : t \in T\})\) of a graph \(G\) is nice if every node \(t \in V(T) \setminus \text{root}\) is of one of the following four types:

- **Leaf:** no children and \(|X_t| = 1\).
Helpful tool: nice tree decompositions

A rooted tree decomposition \((T, \{X_t : t \in T\})\) of a graph \(G\) is nice if every node \(t \in V(T) \setminus \text{root}\) is of one of the following four types:

- **Leaf**: no children and \(|X_t| = 1\).
- **Introduce**: a unique child \(t'\) and \(X_t = X_{t'} \cup \{v\}\) with \(v \notin X_{t'}\).
A rooted tree decomposition \((T, \{X_t : t \in T\})\) of a graph \(G\) is nice if every node \(t \in V(T) \setminus \text{root}\) is of one of the following four types:

- **Leaf**: no children and \(|X_t| = 1\).
- **Introduce**: a unique child \(t'\) and \(X_t = X_{t'} \cup \{v\}\) with \(v \notin X_{t'}\).
- **Forget**: a unique child \(t'\) and \(X_t = X_{t'} \setminus \{v\}\) with \(v \in X_{t'}\).
A rooted tree decomposition \((T, \{X_t : t \in T\})\) of a graph \(G\) is nice if every node \(t \in V(T) \setminus \text{root}\) is of one of the following four types:

- **Leaf**: no children and \(|X_t| = 1\).
- **Introduce**: a unique child \(t'\) and \(X_t = X_{t'} \cup \{v\}\) with \(v \notin X_{t'}\).
- **Forget**: a unique child \(t'\) and \(X_t = X_{t'} \setminus \{v\}\) with \(v \in X_{t'}\).
- **Join**: two children \(t_1\) and \(t_2\) with \(X_t = X_{t_1} = X_{t_2}\).
A rooted tree decomposition \((T, \{X_t : t \in T\})\) of a graph \(G\) is nice if every node \(t \in V(T) \setminus \text{root}\) is of one of the following four types:

- **Leaf**: no children and \(|X_t| = 1\).
- **Introduce**: a unique child \(t'\) and \(X_t = X_{t'} \cup \{v\}\) with \(v \notin X_{t'}\).
- **Forget**: a unique child \(t'\) and \(X_t = X_{t'} \setminus \{v\}\) with \(v \in X_{t'}\).
- **Join**: two children \(t_1\) and \(t_2\) with \(X_t = X_{t_1} = X_{t_2}\).

**Lemma**

A tree decomposition \((T, \{X_t : t \in T\})\) of width \(k\) and \(x\) nodes of an \(n\)-vertex graph \(G\) can be transformed in time \(O(k^2 \cdot n)\) into a nice tree decomposition of \(G\) of width \(k\) and \(k \cdot x\) nodes.
Simpler algorithm for **Independent Set**

How to compute $IS(S, t)$ for every $S \subseteq X_t$:
Simpler algorithm for \textbf{Independent Set}

How to compute $\text{IS}(S, t)$ for every $S \subseteq X_t$:

- If $t$ is a leaf: trivial.
Simpler algorithm for **Independent Set**

How to compute \( IS(S, t) \) for every \( S \subseteq X_t \):

- **If** \( t \) is a **leaf**: trivial.
- **\( t \) is an introduce node**: \( X_t = X_{t'} \cup \{ v \} \)
  
  \[
  |IS(S, t)| = \begin{cases} 
  |IS(S, t')| & \text{if } v \notin S \\
  |IS(S \setminus \{ v \}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\
  -\infty & \text{otherwise}
  \end{cases}
  \]
Simpler algorithm for **Independent Set**

How to compute $IS(S, t)$ for every $S \subseteq X_t$:

- **If** $t$ is a leaf: trivial.

- **$t$** is an introduce node: $X_t = X_{t'} \cup \{v\}$

  $$|IS(S, t)| = \begin{cases} 
  |IS(S, t')| & \text{if } v \notin S \\
  |IS(S \setminus \{v\}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\
  -\infty & \text{otherwise}
  \end{cases}$$

- **If** $t$ is a forget node: $X_t = X_{t'} \setminus \{v\}$

  $$|IS(S, t)| = \max\{|IS(S, t')|, |IS(S \cup \{v\}, t')|\}$$

**Complexity:** $O(2^k \cdot k^2 \cdot n)$
Simpler algorithm for **Independent Set**

How to compute $IS(S, t)$ for every $S \subseteq X_t$:

- If $t$ is a **leaf**: trivial.
- $t$ is an **introduce** node: $X_t = X_{t'} \cup \{v\}$
  \[
  |IS(S, t)| = \begin{cases} 
  |IS(S, t')| & \text{if } v \notin S \\
  |IS(S \setminus \{v\}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\
  -\infty & \text{otherwise}
  \end{cases}
  \]
- If $t$ is a **forget** node: $X_t = X_{t'} \setminus \{v\}$
  \[
  |IS(S, t)| = \max\{|IS(S, t')|, |IS(S \cup \{v\}, t')|\}
  \]
- If $t$ is a **join** node: $X_t = X_{t_1} = X_{t_2}$
  \[
  |IS(S, t)| = |IS(S, t_1)| + |IS(S, t_2)| - |S|
  \]

**Complexity:** $O(2^{2k} \cdot k^2 \cdot n)$
Simpler algorithm for **Independent Set**

How to compute \( IS(S, t) \) for every \( S \subseteq X_t \):

- If \( t \) is a **leaf**: trivial.
- \( t \) is an **introduce** node: \( X_t = X_{t'} \cup \{ v \} \)
  \[
  |IS(S, t)| = \begin{cases} 
  |IS(S, t')| & \text{if } v \notin S \\
  |IS(S \setminus \{ v \}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\
  -\infty & \text{otherwise}
  \end{cases}
  \]
- If \( t \) is a **forget** node: \( X_t = X_{t'} \setminus \{ v \} \)
  \[
  |IS(S, t)| = \max\{|IS(S, t')|, |IS(S \cup \{ v \}, t')|\}
  \]
- If \( t \) is a **join** node: \( X_t = X_{t_1} = X_{t_2} \)
  \[
  |IS(S, t)| = |IS(S, t_1)| + |IS(S, t_2)| - |S|
  \]

**Complexity**: \( \mathcal{O}(2^k \cdot k^2 \cdot n) \)
Let $C$ be a Hamiltonian cycle.

- Note that $C \cap G[V_t]$ is a collection of paths.
Let $C$ be a Hamiltonian cycle.

- Note that $C \cap G[V_t]$ is a collection of paths.
- Partition of the bag $X_t$:
  - $X^0_t$: isolated in $G[V_t]$.
  - $X^1_t$: extremities of paths.
  - $X^2_t$: internal vertices.
Let $C$ be a Hamiltonian cycle.

- Note that $C \cap G[V_t]$ is a collection of paths.

- Partition of the bag $X_t$:
  - $X^0_t$: isolated in $G[V_t]$.
  - $X^1_t$: extremities of paths.
  - $X^2_t$: internal vertices.

For every node $t$ of the tree decomposition, we need to know if

$$(X^0_t, X^1_t, X^2_t, M)$$

where $M$ is a matching on $X^1_t$, corresponds to a partial solution.
Let $t$ be a forget node and $t'$ its child such that $X_t = X_{t'} \setminus \{v\}$.

Claim $X_t$ is a separator $\Rightarrow$

$\forall v \in V_t \setminus X_t$, $v$ is internal in every partial solution.
Let $t$ be a forget node and $t'$ its child such that $X_t = X_{t'} \setminus \{v\}$.

**Claim** $X_t$ is a separator $\Rightarrow$

$\forall v \in V_t \setminus X_t$, $v$ is internal in every partial solution.

$$(X_{t'}^0, X_{t'}^1, X_{t'}^2 \setminus \{v\}, M)$$ is a partial solution for $t$

$\iff$

$$(X_{t'}^0, X_{t'}^1, X_{t'}^2, M)$$ is a partial solution for $t'$ with $v \in X_{t'}^2$
Let \( t \) be an introduce node and \( t' \) its child such that \( X_t = X_{t'} \cup \{v\} \).
Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^0$. 

![Diagram showing nodes and connections]
Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^0$.

\[
(X_t^0 \cup \{v\}, X_{t'}^1, X_{t'}^2, M) \text{ is a partial solution for } t \\
\iff \\
(X_{t'}^0, X_{t'}^1, X_{t'}^2, M) \text{ is a partial solution for } t'
\]
Introduce node (2)

Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$. 

![Diagram showing nodes and edges with v highlighted]
Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.

**Fact** $X_{t'}$ is a separator $\Rightarrow N(v) \cap V_t \subseteq X_t$. 
Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.

**Fact** $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

- a vertex $u \in X_{t'}^1$ becomes internal $\Rightarrow u \in X_t^2$. 

\[ \text{Diagram showing the node and its children, with $v$ being an introduce node.} \]
Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.

**Fact** $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

- A vertex $u \in X_{t'}^1$ becomes internal $\Rightarrow$ $u \in X_t^2$.

$$(X_{t'}^0, X_{t'}^1 \cup \{v\} \setminus \{u\}, X_{t'}^2 \cup \{u\}, M')$$ is a partial solution for $t$

$\iff$

$$(X_{t'}^0, X_{t'}^1, X_{t'}^2, M)$$ is a partial solution for $t'$
Let \( t \) be an introduce node and \( t' \) its child such that \( X_t = X_{t'} \cup \{v\} \).

- Suppose: \( v \in X_t^1 \).

**Fact** \( X_{t'} \) is a separator \( \Rightarrow N(v) \cap V_t \subseteq X_t \).

- a vertex \( u \in X_{t'}^1 \) becomes internal \( \Rightarrow u \in X_t^2 \).
- or a vertex \( w \in X_{t'}^0 \) becomes extremity of a path \( \Rightarrow w \in X_t^1 \) (similar).
Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose $v \in X_t^2$.

**Fact** $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$. 
Introduce node (3)

Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.

**Fact** $X_{t'}$ is a separator $\Rightarrow N(v) \cap V_t \subseteq X_t$.

1 two vertices $u, u' \in X_t^1$ become internal $\Rightarrow u, u' \in X_t^2$. 

Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.

**Fact** $X_{t'}$ is a separator $\Rightarrow N(v) \cap V_t \subseteq X_t$.

1 two vertices $u, u' \in X_t^1$ become internal $\Rightarrow u, u' \in X_t^2$. 

Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.

### Fact

- $X_{t'}$ is a separator $\Rightarrow N(v) \cap V_t \subseteq X_t$.

- Two vertices $u, u' \in X_{t'}^1$ become internal $\Rightarrow u, u' \in X_t^2$.

- $(X_{t'}^0, X_{t'}^1 \setminus \{u, u'\}, X_{t'}^2 \cup \{v, u, u'\}, M')$ is a partial solution for $t$ $\iff$ $(X_{t'}^0, X_{t'}^1, X_{t'}^2, M)$ is a partial solution for $t'$
Introduce node (3)

Let $t$ be an introduce node and $t'$ its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.

**Fact** $X_{t'}$ is a separator $\Rightarrow N(v) \cap V_t \subseteq X_t$.

1. two vertices $u, u' \in X_t^1$ become internal $\Rightarrow u, u' \in X_t^2$.
2. two vertices $w, w' \in X_t^0$ become extremities $\Rightarrow w, w' \in X_t^1$. 
Let \( t \) be an introduce node and \( t' \) its child such that \( X_t = X_{t'} \cup \{ v \} \).

- Suppose. \( v \in X_t^2 \).

Fact \( X_{t'} \) is a separator \( \Rightarrow \) \( N(v) \cap V_t \subseteq X_t \).

1. two vertices \( u, u' \in X_{t'}^1 \) become internal \( \Rightarrow u, u' \in X_t^2 \).
2. two vertices \( w, w' \in X_{t'}^0 \) become extremities \( \Rightarrow w, w' \in X_t^1 \).
3. \( w \in X_{t'}^0 \) becomes extremity and \( v \in X_{t'}^1 \) internal \( \Rightarrow w \in X_t^1, v \in X_t^2 \).
Let $t$ be a join node and $t_1, t_2$ its children such that $X_t = X_{t_1} = X_{t_2}$.

**Fact** For being compatible, partial solutions should verify:

- $X^2_{t_1} \subseteq X^0_{t_2}$ and $X^1_{t_1} \subseteq X^1_{t_2} \cup X^0_{t_2}$.
- $X^2_{t_2} \subseteq X^0_{t_1}$ and $X^1_{t_2} \subseteq X^1_{t_1} \cup X^0_{t_1}$.
- The union of matchings $M_1$ et $M_2$ does not create any cycle.
Let $t$ be a join node and $t_1, t_2$ its children such that $X_t = X_{t_1} = X_{t_2}$.

**Fact** For being compatible, partial solutions should verify:

- $X_{t_1}^2 \subseteq X_{t_2}^0$ and $X_{t_1}^1 \subseteq X_{t_2}^1 \cup X_{t_2}^0$.
- $X_{t_2}^2 \subseteq X_{t_1}^0$ and $X_{t_2}^1 \subseteq X_{t_1}^1 \cup X_{t_1}^0$.
- The union of matchings $M_1$ and $M_2$ does not create any cycle.
Hamiltonian Cycle on tree decompositions

Analysis of the running time, given a tree decomposition of width $k$:
Hamiltonian Cycle on tree decompositions

Analysis of the running time, given a tree decomposition of width $k$:

- Number of subproblems at each node: $3^k \cdot k!$. 

Can this approach be generalized to more problems?
Analysis of the **running time**, given a tree decomposition of width $k$:

- Number of subproblems at each node: $3^k \cdot k!$.

- Number of nodes in a nice tree decomposition: $k \cdot n$. 
Analysis of the running time, given a tree decomposition of width $k$:

- Number of subproblems at each node: $3^k \cdot k!$.
- Number of nodes in a nice tree decomposition: $k \cdot n$.

Total running time of the algorithm: $k^{O(k)} \cdot n$. 
**Hamiltonian Cycle on tree decompositions**

Analysis of the **running time**, given a tree decomposition of width $k$:

- Number of subproblems at each node: $3^k \cdot k!$.
- Number of nodes in a nice tree decomposition: $k \cdot n$.

Total running time of the algorithm: $k^{O(k)} \cdot n$. 
Hamiltonian Cycle on tree decompositions

Analysis of the running time, given a tree decomposition of width $k$:

- Number of subproblems at each node: $3^k \cdot k!$.
- Number of nodes in a nice tree decomposition: $k \cdot n$.

Total running time of the algorithm: $k^{O(k)} \cdot n$.

Can this approach be generalized to more problems?
1. Definition and simple properties

2. Dynamic programming on tree decompositions
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3. Brambles and duality

4. Computing treewidth
We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $\mathcal{U} = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) | v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\lor$, $\land$, $\Rightarrow$, $\neg$, $=$, $\neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.
- Relations $\in$, $\subseteq$ on vertex/edge sets.
- Quantifiers $\exists$, $\forall$ on vertex/edge variables or vertex/edge sets.
We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.
We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) | v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:
We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex, edge, } I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\lor, \land, \Rightarrow, \neg, =, \neq$. 
We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\lor, \land, \Rightarrow, \neg, =, \neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.
Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\lor, \land, \Rightarrow, \neg, =, \neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.
- Relations $\in, \subseteq$ on vertex/edge sets.
Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $G = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- “vertex” and “edge” are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) | v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\lor, \land, \Rightarrow, \neg, =, \neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.
- Relations $\in, \subseteq$ on vertex/edge sets.
- Quantifiers $\exists, \forall$ on vertex/edge variables or vertex/edge sets.

(MSO$_1$/MSO$_2$)
Example 1  Expressing that \( \{u, v\} \in E(G) \):
\[
\exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e).
\]
Monadic second order logic of graphs: examples

Example 1 Expressing that $\{u, v\} \in E(G)$: $\exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e)$.

Example 2 Expressing that a set $S \subseteq V(G)$ is a dominating set.

$\text{DomSet}(S): \forall v \in V(G) \setminus S, \exists u \in S: \{u, v\} \in E(G)$. 

Other properties that can be expressed in MSO$^2$:
- A set being a vertex cover, independent set.
- A graph being $k$-colorable (for fixed $k$), having a Hamiltonian cycle.
Example 1 Expressing that \( \{u, v\} \in E(G) \): \( \exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e) \).

Example 2 Expressing that a set \( S \subseteq V(G) \) is a dominating set.

\[ \text{DomSet}(S) : \quad \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G). \]

Example 3 Expressing that a graph \( G = (V, E) \) is connected.
Monadic second order logic of graphs: examples

Example 1  Expressing that \( \{u, v\} \in E(G) \): \( \exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e) \).

Example 2  Expressing that a set \( S \subseteq V(G) \) is a dominating set.

\[ \text{DomSet}(S) : \quad \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G). \]

Example 3  Expressing that a graph \( G = (V, E) \) is connected.

- For every bipartition de \( V \), there is a transversal edge:
Example 1  Expressing that \( \{u, v\} \in E(G) \): \( \exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e) \).

Example 2  Expressing that a set \( S \subseteq V(G) \) is a dominating set.

\[ \text{DomSet}(S) : \quad \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G). \]

Example 3  Expressing that a graph \( G = (V, E) \) is connected.

- For every bipartition de \( V \), there is a transversal edge:

Expressing that two sets \( V_1, V_2 \) define a bipartition of \( V \):

\[ \forall v \in V, (v \in V_1 \lor v \in V_2) \land (v \in V_1 \Rightarrow v \notin V_2) \land (v \in V_2 \Rightarrow v \notin V_1). \]
Monadic second order logic of graphs: examples

**Example 1**  
Expressing that $\{u, v\} \in E(G)$:  
$\exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e)$.

**Example 2**  
Expressing that a set $S \subseteq V(G)$ is a dominating set.  
$\text{DomSet}(S): \quad \forall v \in V(G) \setminus S, \exists u \in S: \{u, v\} \in E(G)$.

**Example 3**  
Expressing that a graph $G = (V, E)$ is connected.  
- For every bipartition of $V$, there is a transversal edge:
  
  For expressing that two sets $V_1, V_2$ define a bipartition of $V$:
  
  $\forall v \in V, (v \in V_1 \lor v \in V_2) \land (v \in V_1 \Rightarrow v \notin V_2) \land (v \in V_2 \Rightarrow v \notin V_1)$.

  Connected:  
  $\forall$ bipartition $V_1, V_2, \exists v_1 \in V_1, \exists v_2 \in V_2, \{v_1, v_2\} \in E(G)$. 
Monadic second order logic of graphs: examples

**Example 1** Expressing that \( \{u, v\} \in E(G) \): \( \exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e) \).

**Example 2** Expressing that a set \( S \subseteq V(G) \) is a dominating set.

\[ \text{DomSet}(S) : \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G). \]

**Example 3** Expressing that a graph \( G = (V, E) \) is connected.

- For every bipartition of \( V \), there is a transversal edge:

Expressing that two sets \( V_1, V_2 \) define a bipartition of \( V \):

\[ \forall v \in V, (v \in V_1 \lor v \in V_2) \land (v \in V_1 \Rightarrow v \notin V_2) \land (v \in V_2 \Rightarrow v \notin V_1). \]

**Connected**: \( \forall \) bipartition \( V_1, V_2, \exists v_1 \in V_1, \exists v_2 \in V_2, \{v_1, v_2\} \in E(G). \)

Other properties that can be expressed in \( \text{MSO}_2 \):

- a set being a vertex cover, independent set.
Monadic second order logic of graphs: examples

**Example 1** Expressing that \( \{u, v\} \in E(G) \): \( \exists e \in E, \text{inc}(u, e) \land \text{inc}(v, e) \).

**Example 2** Expressing that a set \( S \subseteq V(G) \) is a dominating set.

\[ \text{DomSet}(S) : \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G). \]

**Example 3** Expressing that a graph \( G = (V, E) \) is connected.

- For every bipartition \( V \), there is a transversal edge:

Expressing that two sets \( V_1, V_2 \) define a bipartition of \( V \):

\[ \forall v \in V, (v \in V_1 \lor v \in V_2) \land (v \in V_1 \Rightarrow v \not\in V_2) \land (v \in V_2 \Rightarrow v \not\in V_1). \]

**Connected**: \( \forall \) bipartition \( V_1, V_2, \exists v_1 \in V_1, \exists v_2 \in V_2, \{v_1, v_2\} \in E(G). \)

Other properties that can be expressed in MSO\(_2\):

- a set being a vertex cover, independent set.
- a graph being \( k \)-colorable (for fixed \( k \)), having a Hamiltonian cycle.
Theorem (Courcelle. 1990)

Every problem expressible in MSO$_2$ can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$.  

Examples: Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, $k$-Coloring for fixed $k$, ...
Theorem (Courcelle. 1990)

*Every problem expressible in MSO$_2$ can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$."

The function $f(tw)$ depends on the structure of the MSO$_2$ formula.
Theorem (Courcelle. 1990)

*Every problem expressible in MSO\(_2\) can be solved in time \(f(tw) \cdot n\) on graphs on \(n\) vertices and treewidth at most \(tw\).*

The function \(f(tw)\) depends on the structure of the MSO\(_2\) formula.

Within the same running time, one can also optimize the size of a vertex/edge set satisfying an MSO\(_2\) formula.
Theorem (Courcelle. 1990)

Every problem expressible in MSO$_2$ can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$.

The function $f(tw)$ depends on the structure of the MSO$_2$ formula.

Within the same running time, one can also optimize the size of a vertex/edge set satisfying an MSO$_2$ formula.

Examples: Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, $k$-Coloring for fixed $k$, ...
Theorem (Courcelle. 1990)

Every problem expressible in MSO$_2$ can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$.

The function $f(tw)$ depends on the structure of the MSO$_2$ formula.

Within the same running time, one can also optimize the size of a vertex/edge set satisfying an MSO$_2$ formula.

Examples: Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, $k$-Coloring for fixed $k$, ...

In parameterized complexity: FPT parameterized by treewidth.
1 Definition and simple properties

2 Dynamic programming on tree decompositions
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3 Brambles and duality

4 Computing treewidth
Idea Measure the complexity of an algorithm in terms of the input size and an additional parameter.

This theory started in the late 80’s, by Downey and Fellows:

Today, it is a well-established and very active area.
A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $k$ is called the parameter.
Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $k$ is called the parameter.

- **$k$-Vertex Cover**: Does a graph $G$ contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?

- **$k$-Clique**: Does a graph $G$ contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise adjacent vertices?

- **Vertex $k$-Coloring**: Can the vertices of a graph be colored with $\leq k$ colors, so that any two adjacent vertices get different colors?
Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $k$ is called the parameter.

- **$k$-Vertex Cover**: Does a graph $G$ contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?

- **$k$-Clique**: Does a graph $G$ contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise adjacent vertices?

- **Vertex $k$-Coloring**: Can the vertices of a graph be colored with $\leq k$ colors, so that any two adjacent vertices get different colors?

These three problems are **NP-hard**, but are they equally hard?
They behave quite differently...

- **$k$-Vertex Cover**: Solvable in time $O(2^k \cdot (m + n))$

- **$k$-Clique**: Solvable in time $O(k^2 \cdot n^k)$

- **Vertex $k$-Coloring**: NP-hard for fixed $k = 3$. 
They behave quite differently...

- **$k$-Vertex Cover**: Solvable in time $O(2^k \cdot (m + n)) = f(k) \cdot n^{O(1)}$.

- **$k$-Clique**: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

- **Vertex $k$-Coloring**: NP-hard for fixed $k = 3$. 
They behave quite differently...

- **k-Vertex Cover**: Solvable in time $\mathcal{O}(2^k \cdot (m + n)) = f(k) \cdot n^{O(1)}$.

  The problem is **FPT** (fixed-parameter tractable)

- **k-Clique**: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

- **Vertex k-Coloring**: NP-hard for fixed $k = 3$. 

They behave quite differently...

- **k-Vertex Cover**: Solvable in time $O(2^k \cdot (m + n)) = f(k) \cdot n^{O(1)}$.

  The problem is **FPT** (fixed-parameter tractable)

- **k-Clique**: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

  The problem is **XP** (slice-wise polynomial)

- **Vertex k-Coloring**: **NP-hard** for fixed $k = 3$. 
They behave quite differently...

- **k-Vertex Cover**: Solvable in time $O(2^k \cdot (m + n)) = f(k) \cdot n^{O(1)}$.

  The problem is **FPT** (fixed-parameter tractable)

- **k-Clique**: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

  The problem is **XP** (slice-wise polynomial)

- **Vertex k-Coloring**: NP-hard for fixed $k = 3$.

  The problem is **para-NP-hard**
Why $k$-\textsc{Clique} may not be FPT?

$k$-\textsc{Clique}: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$. 
Why $k$-CLIQUE may not be FPT?

$k$-CLIQUE: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why $k$-CLIQUE may not be FPT?
Why $k$-CLIQUE may not be FPT?

$k$-CLIQUE: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why $k$-CLIQUE may not be FPT?

So far, nobody has managed to find an FPT algorithm. (also, nobody has found a poly-time algorithm for 3-SAT)
Why \textit{k-Clique} may not be FPT?

\textit{k-Clique}: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why \textit{k-Clique} may not be FPT?

So far, nobody has managed to find an FPT algorithm.

(also, nobody has found a poly-time algorithm for 3-SAT)

Working hypothesis of parameterized complexity: \textbf{\textit{k-Clique} is not FPT}

(in classical complexity: 3-SAT cannot be solved in poly-time)
How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.
Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from $A$ to $B$ is an algorithm such that:

$$\text{Instance } (x, k) \text{ of } A \quad \text{time } f(k) \cdot |x|^{O(1)} \quad \text{Instance } (x', k') \text{ of } B$$
How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from $A$ to $B$ is an algorithm such that:

1. Instance $(x, k)$ of $A$  
2. time $f(k) \cdot |x|^{O(1)}$  
3. Instance $(x', k')$ of $B$

1. $(x, k)$ is a Yes-instance of $A$ $\iff$ $(x', k')$ is a Yes-instance of $B$.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \rightarrow \mathbb{N}$.
How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from $A$ to $B$ is an algorithm such that:

1. Instance $(x, k)$ of $A$ is a \text{Yes}-instance of $A$ $\iff$ Instance $(x', k')$ of $B$ is a \text{Yes}-instance of $B$.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \rightarrow \mathbb{N}$.

$W[1]$-hard problem: $\exists$ parameterized reduction from $k$-\text{Clique} to it.

$W[2]$-hard problem: $\exists$ param. reduction from $k$-\text{Dominating Set} to it.
How to transfer hardness among parameterized problems?

Let \( A, B \subseteq \Sigma^* \times \mathbb{N} \) be two parameterized problems.

A parameterized reduction from \( A \) to \( B \) is an algorithm such that:

1. Instance \((x, k)\) of \( A \)  
   \[ \text{time } f(k) \cdot |x|^O(1) \]
   
   Instance \((x', k')\) of \( B \)

   \( (x, k) \) is a \( \text{\textsc{Yes}} \)-instance of \( A \) \( \iff \) \( (x', k') \) is a \( \text{\textsc{Yes}} \)-instance of \( B \).

2. \( k' \leq g(k) \) for some computable function \( g : \mathbb{N} \to \mathbb{N} \).

\( \text{\textsc{W[1]}} \)-hard problem: \( \exists \) parameterized reduction from \( k-\text{\textsc{Clique}} \) to it.

\( \text{\textsc{W[2]}} \)-hard problem: \( \exists \) param. reduction from \( k-\text{\textsc{Dominating Set}} \) to it.

\( \text{\textsc{W[i]}} \)-hard: strong evidence of \emph{not} being \textsc{FPT}. 
How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from $A$ to $B$ is an algorithm such that:

1. Instance $(x, k)$ of $A$ is a $\text{Yes}$-instance of $A \iff (x', k')$ is a $\text{Yes}$-instance of $B$.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

$W[1]$-hard problem: $\exists$ parameterized reduction from $k$-\text{Clique} to it.

$W[2]$-hard problem: $\exists$ param. reduction from $k$-\text{Dominating Set} to it.

$W[i]$-hard: strong evidence of not being FPT. Hypothesis: $\text{FPT} \neq W[1]$
Theorem (Courcelle. 1990)

Every problem expressible in $\text{MSO}_2$ can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$.

In parameterized complexity: FPT parameterized by treewidth.
Theorem (Courcelle. 1990)

Every problem expressible in $\text{MSO}_2$ can be solved in time $f(\text{tw}) \cdot n$ on graphs on $n$ vertices and treewidth at most $\text{tw}$.

In parameterized complexity: FPT parameterized by treewidth.

1. Are all “natural” graph problems FPT parameterized by treewidth?
Back to treewidth: only good news?

**Theorem (Courcelle. 1990)**

*Every problem expressible in MSO\(_2\) can be solved in time \(f(tw) \cdot n\) on graphs on \(n\) vertices and treewidth at most \(tw\).*

In parameterized complexity: FPT parameterized by treewidth.

1. Are all “natural” graph problems FPT parameterized by treewidth?

The vast majority, but not all of them:

- **List Coloring** is \(W[1]\)-hard parameterized by treewidth.

[Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]
Back to treewidth: only good news?

**Theorem (Courcelle. 1990)**

Every problem expressible in MSO\(_2\) can be solved in time \(f(tw) \cdot n\) on graphs on \(n\) vertices and treewidth at most \(tw\).

In parameterized complexity: FPT parameterized by treewidth.

1. Are all “natural” graph problems FPT parameterized by treewidth?

   The vast majority, but not all of them:

   - **List Coloring** is \(W[1]\)-hard parameterized by treewidth.

     [Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]

   - Some problems are even \(NP\)-hard on graphs of constant treewidth: **Steiner Forest** (\(tw = 3\)), **Bandwidth** (\(tw = 1\)).
Back to treewidth: only good news?

Theorem (Courcelle. 1990)

*Every problem expressible in MSO$_2$ can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$."

In parameterized complexity: FPT parameterized by treewidth.

1. Are all “natural” graph problems FPT parameterized by treewidth?

The vast majority, but not all of them:

- **List Coloring** is $W[1]$-hard parameterized by treewidth.
  
  [Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]

- Some problems are even NP-hard on graphs of constant treewidth: **Steiner Forest** ($tw = 3$), **Bandwidth** ($tw = 1$).

1. Definition and simple properties

2. Dynamic programming on tree decompositions
   - Two simple algorithms
   - Courcelle’s theorem
   - Introduction to parameterized complexity

3. Brambles and duality

4. Computing treewidth
Brambles

How to provide a lower bound on the treewidth of a graph?
Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in $G$ from $A$ to $B$. 
Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in $G$ from $A$ to $B$.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.
Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in $G$ from $A$ to $B$.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.

A bramble in a graph $G$ is a family $B$ of pairwise touching connected vertex sets of $G$. 
How to provide a **lower bound** on the **treewidth** of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in $G$ from $A$ to $B$.

A set $S \subseteq V(G)$ is **connected** if $G[S]$ is connected.

A **bramble** in a graph $G$ is a family $B$ of pairwise touching connected vertex sets of $G$.

The **order** of a bramble $B$ in a graph $G$ is the minimum size of a vertex set $S \subseteq V(G)$ intersecting all the sets in $B$. 

**Theorem (Robertson and Seymour. 1993)**

For every $k \geq 0$ and graph $G$, the treewidth of $G$ is at least $k$ if and only if $G$ contains a bramble of order at least $k + 1$. 


Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in $G$ from $A$ to $B$.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.

A bramble in a graph $G$ is a family $\mathcal{B}$ of pairwise touching connected vertex sets of $G$.

The order of a bramble $\mathcal{B}$ in a graph $G$ is the minimum size of a vertex set $S \subseteq V(G)$ intersecting all the sets in $\mathcal{B}$.

Theorem (Robertson and Seymour. 1993)

For every $k \geq 0$ and graph $G$, the treewidth of $G$ is at least $k$ if and only if $G$ contains a bramble of order at least $k + 1$. 
Another dual notion to treewidth: linkedness

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are separable if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$. 

[slides borrowed from Christophe Paul]
Another dual notion to treewidth: linkedness

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are **separable** if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

- For $k \geq 1$, a set $X \subseteq V(G)$ is **$k$-linked** if $|X| \geq k$ and $orall Y, Z \subseteq X$, $|Y| = |Z| \leq k$, $Y$ and $Z$ are not separable.
Another dual notion to treewidth: linkedness

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are **separable** if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

- For $k \geq 1$, a set $X \subseteq V(G)$ is **$k$-linked** if $|X| \geq k$ and $\forall \ Y, Z \subseteq X, |Y| = |Z| \leq k$, $Y$ and $Z$ are not separable.

The $(k \times k)$-grid is $k$-linked
Another dual notion to treewidth: linkedness

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are separable if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

- For $k \geq 1$, a set $X \subseteq V(G)$ is $k$-linked if $|X| \geq k$ and $\forall Y, Z \subseteq X, |Y| = |Z| \leq k$, $Y$ and $Z$ are not separable.

The $(k \times k)$-grid is $k$-linked

$K_{2k,k}$ is also $k$-linked
Highly linked graphs have large treewidth

**Lemma**

If $G$ contains a $(k + 1)$-linked set $X$ with $|X| \geq 3k$, then $\text{tw}(G) \geq k$. 
Highly linked graphs have large treewidth

**Lemma**

If $G$ contains a $(k + 1)$-linked set $X$ with $|X| \geq 3k$, then $\text{tw}(G) \geq k$.

**Contradiction:** Consider a tree decomposition of $G$ of width $< k$.

Let $t$ be a “lowest” node with $|V_t \cap X| > 2k$. 
Highly linked graphs have large treewidth

**Lemma**

If $G$ contains a $(k + 1)$-linked set $X$ with $|X| \geq 3k$, then $\text{tw}(G) \geq k$.

**Contradiction:** Consider a tree decomposition of $G$ of width $< k$.

Let $t$ be a “lowest” node with $|V_t \cap X| > 2k$.

If $\exists i \in [\ell]$ such that $|V_{t_i} \cap X| \geq k$, then we can choose $Y \subseteq V_{t_i} \cap X$, $|Y| = k$ and $Z \subseteq (V \setminus V_{t_i}) \cap X$, $|Z| = k$.

But $S = X_{t_i} \cap X_t$ separates $Y$ and $Z$ and $|S| \leq k - 1$. 
Highly linked graphs have large treewidth

**Lemma**

If $G$ contains a $(k + 1)$-linked set $X$ with $|X| \geq 3k$, then $\text{tw}(G) \geq k$.

**Contradiction:** Consider a tree decomposition of $G$ of width $< k$.

Let $t$ be a “lowest” node with $|V_t \cap X| > 2k$.

Otherwise, let $W = V_{t_1} \cup \cdots \cup V_{t_i}$ with $|W \cap X| > k$ and $|(W \setminus V_{t_j}) \cap X| < k$ for $1 \leq j \leq i$.

$Y \subseteq W \cap X$, $|Y| = k + 1$ and $Z \subseteq (V \setminus W) \cap X$, $|Z| = k + 1$.

But $S = X_t$ separates $Y$ from $Z$ and $|S| \leq k$. 
Lemma

Given a vertex set $X$ of a graph $G$ and $k \leq |X|$, it is possible to decide whether $X$ is $k$-linked in time $f(k) \cdot n^{O(1)}$. 
Deciding linkedness is FPT

Lemma

Given a vertex set $X$ of a graph $G$ and $k \leq |X|$, it is possible to decide whether $X$ is $k$-linked in time $f(k) \cdot n^{O(1)}$.

- For every pair of subsets $Y, Z \subseteq X$ with $|Y| = |Z| \leq k$, we can test whether $Y$ and $Z$ are separable in polynomial time (flow algorithm).
Deciding linkedness is FPT

**Lemma**

Given a vertex set $X$ of a graph $G$ and $k \leq |X|$, it is possible to decide whether $X$ is $k$-linked in time $f(k) \cdot n^{O(1)}$.

- For every pair of subsets $Y, Z \subseteq X$ with $|Y| = |Z| \leq k$, we can test whether $Y$ and $Z$ are separable in polynomial time (flow algorithm).

- Complexity: $4^k \cdot n^{O(1)}$. 
Deciding linkedness is FPT

Lemma

Given a vertex set $X$ of a graph $G$ and $k \leq |X|$, it is possible to decide whether $X$ is $k$-linked in time $f(k) \cdot n^{O(1)}$.

- For every pair of subsets $Y, Z \subseteq X$ with $|Y| = |Z| \leq k$, we can test whether $Y$ and $Z$ are separable in polynomial time (flow algorithm).
- Complexity: $4^k \cdot n^{O(1)}$.

Remark: If $X$ is not $k$-linked we can find, within the same running time, two separable subsets $Y, Z \subseteq X$. 
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is **NP-complete.** [Arnborg, Corneil, Proskurowski. 1987]
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{O(k^3)} \cdot n$. [Bodlaender. 1996]
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]

- Can be solved in time $k^O(k^3) \cdot n$. [Bodlaender. 1996]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $O(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $O(k \cdot \sqrt{\log k})$ in time $n^{O(1)}$. [Feige, Hajiaghayi, Lee. 2008]
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]

- Can be solved in time $k^{O(k^3)} \cdot n$. [Bodlaender. 1996]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $O(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $O(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]

- Can be solved in time $k^{O(k^3)} \cdot n$. [Bodlaender. 1996]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $O(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $O(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $5k + 4$ in time $2^{O(k)} \cdot n$. [Bodlaender et al. 2016]
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]

- Can be solved in time $k^{O(k^3)} \cdot n$. [Bodlaender. 1996]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $O(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $O(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $5k + 4$ in time $2^{O(k)} \cdot n$. [Bodlaender et al. 2016]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $O(k \cdot \sqrt{\log k})$ in time $n^{O(1)}$. [Feige, Hajiaghayi, Lee. 2008]
Complexity of computing the treewidth of a graph

Given a graph $G$ on $n$ vertices and a positive integer $k$:

- Deciding whether $\text{tw}(G) \leq k$ is **NP-complete**. [Arnborg, Corneil, Proskurowski. 1987]

- Can be **solved** in time $k^{O(k^3)} \cdot n$. [Bodlaender. 1996]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $O(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $O(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $5k + 4$ in time $2^{O(k)} \cdot n$. [Bodlaender et al. 2016]

- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $O(k \cdot \sqrt{\log k})$ in time $n^{O(1)}$. [Feige, Hajiaghayi, Lee. 2008]
4-approximation of Robertson and Seymour

Idea

- We add vertices to a set $U$ in a greedy way, until $U = V(G)$. 

[slides borrowed from Christophe Paul]
4-approximation of Robertson and Seymour

Idea

- We add vertices to a set $U$ in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $T_U$ of $G[U]$ s.t. $\text{width}(T_U) < 4k$, unless we stop the algorithm and conclude that $\text{tw}(G) \geq k$. 

[slides borrowed from Christophe Paul]
4-approximation of Robertson and Seymour

Idea

- We add vertices to a set $U$ in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. $\text{width}(\mathcal{T}_U) < 4k$, unless we stop the algorithm and conclude that $\text{tw}(G) \geq k$.
4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]

Idea

- We add vertices to a set $U$ in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $T_U$ of $G[U]$ s.t. $\text{width}(T_U) < 4k$, unless we stop the algorithm and conclude that $\text{tw}(G) \geq k$.

Invariant

- Every connected component of $G - U$ has at most $3k$ neighbors in $U$. 
4-approximation of Robertson and Seymour

Idea

- We add vertices to a set $U$ in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. width($\mathcal{T}_U$) < 4k, unless we stop the algorithm and conclude that tw($G$) $\geq$ k.

Invariant

- Every connected component of $G - U$ has at most 3k neighbors in $U$.
- There exists a bag $X_t$ of $\mathcal{T}_U$ containing all these neighbors.
**Idea**

- We add vertices to a set $U$ in a **greedy** way, until $U = V(G)$.
- We maintain a tree decomposition $T_U$ of $G[U]$ s.t. $\text{width}(T_U) < 4k$, unless we stop the algorithm and conclude that $\text{tw}(G) \geq k$.

**Invariant**

- Every **connected component** of $G - U$ has at most $3k$ neighbors in $U$.
- There exists a bag $X_t$ of $T_U$ containing all these neighbors.

Initially, we start with $U$ being any set of $3k$ vertices.
Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$. 
Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

- If $|X| < 3k$: we add a node $t'$ neighbor of $t$ such that $X_{t'} = \{x\} \cup X$, with $x \in C$ being a neighbor of $X_t$. 
Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: we test whether $X$ is $(k + 1)$-linked in time $f(k) \cdot n^{O(1)}$. 
Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: we test whether $X$ is $(k + 1)$-linked in time $f(k) \cdot n^{O(1)}$:
  - If $X$ is $(k + 1)$-linked, then $tw(G) \geq k$, and we stop.
Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: we test whether $X$ is $(k + 1)$-linked in time $f(k) \cdot n^{o(1)}$:
  1. If $X$ is $(k + 1)$-linked, then $tw(G) \geq k$, and we stop.
  2. Otherwise, we find sets $Y, Z, S$ with $|S| < |Y| = |Z| \leq k + 1$
     and such that $S$ separates $Y$ and $Z$. 

Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: we test whether $X$ is $(k + 1)$-linked in time $f(k) \cdot n^{O(1)}$:
  1. If $X$ is $(k + 1)$-linked, then $tw(G) \geq k$, and we stop.
  2. Otherwise, we find sets $Y$, $Z$, $S$ with $|S| < |Y| = |Z| \leq k + 1$ and such that $S$ separates $Y$ and $Z$.
     We create a node $t'$ neighbor of $t$ s.t. $X_{t'} = (S \cap C) \cup X$. 

Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

1. If $|X| = 3k$: we test whether $X$ is $(k + 1)$-linked in time $f(k) \cdot n^{O(1)}$:
   - If $X$ is $(k + 1)$-linked, then $\text{tw}(G) \geq k$, and we stop.
   - Otherwise, we find sets $Y$, $Z$, $S$ with $|S| < |Y| = |Z| \leq k + 1$ and such that $S$ separates $Y$ and $Z$.

We create a node $t'$ neighbor of $t$ s.t. $X_{t'} = (S \cap C) \cup X$.

Obs: the neighbors of every new component $C' \subseteq C$ are in $(X \setminus Z) \cup (S \cap C)$ or in $(X \setminus Y) \cup (S \cap C)$.
Let $X$ be the neighbors of a component $C$ and $t$ be the node s.t. $X \subseteq X_t$.

1. If $|X| = 3k$: we test whether $X$ is $(k + 1)$-linked in time $f(k) \cdot n^{O(1)}$:
   
   1. If $X$ is $(k + 1)$-linked, then $tw(G) \geq k$, and we stop.
   2. Otherwise, we find sets $Y$, $Z$, $S$ with $|S| < |Y| = |Z| \leq k + 1$ and such that $S$ separates $Y$ and $Z$.

   We create a node $t'$ neighbor of $t$ s.t. $X_{t'} = (S \cap C) \cup X$.

   **Obs:** the neighbors of every new component $C' \subseteq C$ are in $(X \setminus Z) \cup (S \cap C)$ or in $(X \setminus Y) \cup (S \cap C)$ $\Rightarrow \leq 3k$ neighbors.
Gràcies!