

# Lecture 2: Graph width notions, dynamical programming

An Introduction to Parameterized Complexity

Clemens Grabmayer

Ph.D. Program, Advanced Period

Gran Sasso Science Institute

L'Aquila, Italy

Tuesday, June 11, 2024

# Course overview

Monday, June 10 10.30 – 12.30	Tuesday, June 11	Wednesday, June 12 10.30 – 12.30	Thursday, June 13	Friday, June 14
<b>Introduction &amp; basic FPT results</b> motivation for FPT kernelization, Crown Lemma, Sunflower Lemma		<b>Algorithmic Meta-Theorems</b> 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width		
	GDA		GDA	GDA
<i>Algorithmic Techniques</i>		<i>Formal-Method &amp; Algorithmic Techniques</i>		
	14.30 – 16.30			14.30 – 16.30
	<b>Notions of bounded graph width</b>			<b>FPT-Intractability Classes &amp; Hierarchies</b>
	path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. widths	GDA	GDA	motivation for FP-intractability results, FPT-reductions, class XP (slicewise polynomial), W- and A-Hierarchies, placing problems on these hierarchies

# Overview

- ▶ comparing parameterizations

# Overview

- ▶ comparing parameterizations
- ▶ dynamical programming on trees, example:
  - ▶ WEIGHTED-INDEPENDENT-SET (and VERTEX-COVER)
- ▶ path-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ tree-width
  - ▶ example: fpt-algorithm for bounded path-width

# Overview

- ▶ comparing parameterizations
- ▶ dynamical programming on trees, example:
  - ▶ WEIGHTED-INDEPENDENT-SET (and VERTEX-COVER)
- ▶ path-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ tree-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ fpt-results for other problems, obtained similarly

# Overview

- ▶ comparing parameterizations
- ▶ dynamical programming on trees, example:
  - ▶ WEIGHTED-INDEPENDENT-SET (and VERTEX-COVER)
- ▶ path-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ tree-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ fpt-results for other problems, obtained similarly
- ▶ other notions of width
  - ▶ clique-width
  - ▶ using  $f$ -width to define:
    - ▶ carving-width (and cut-width)
    - ▶ branch-width
    - ▶ rank-width

# Overview

- ▶ comparing parameterizations
- ▶ dynamical programming on trees, example:
  - ▶ WEIGHTED-INDEPENDENT-SET (and VERTEX-COVER)
- ▶ path-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ tree-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ fpt-results for other problems, obtained similarly
- ▶ other notions of width
  - ▶ clique-width
  - ▶ using  $f$ -width to define:
    - ▶ carving-width (and cut-width)
    - ▶ branch-width
    - ▶ rank-width
- ▶ comparing width-notions

# Fixed-Parameter tractable

A *parameterized problem* is a triple  $\langle Q, \Sigma, \kappa \rangle$  (short:  $\langle Q, \kappa \rangle$ ) where:

- ▷  $Q \subseteq \Sigma^*$  is the set of *(classical) problem instances*,
- ▷  $\kappa : \Sigma^* \rightarrow \mathbb{N}$  is a (general) function, *the parameterization*.



# Fixed-Parameter tractable

A *parameterized problem* is a triple  $\langle Q, \Sigma, \kappa \rangle$  (short:  $\langle Q, \kappa \rangle$ ) where:

- ▷  $Q \subseteq \Sigma^*$  is the set of *(classical) problem instances*,
- ▷  $\kappa : \Sigma^* \rightarrow \mathbb{N}$  is a (general) function, *the parameterization*.

Parameterized problem  $\langle Q, \Sigma, \kappa \rangle$

**Instance:**  $x \in \Sigma^*$ .

**Parameter:**  $\kappa(x)$ .

**Problem:** Is  $x \in Q$ ?

# Fixed-Parameter tractable

A *parameterized problem* is a triple  $\langle Q, \Sigma, \kappa \rangle$  (short:  $\langle Q, \kappa \rangle$ ) where:

- ▷  $Q \subseteq \Sigma^*$  is the set of *(classical) problem instances*,
- ▷  $\kappa : \Sigma^* \rightarrow \mathbb{N}$  is a (general) function, *the parameterization*.

## Definition

A parameterized problem  $\langle Q, \Sigma, \kappa \rangle$  is *fixed-parameter tractable* (is in **FPT**) if:

$\exists f : \mathbb{N} \rightarrow \mathbb{N}$  computable  $\exists p \in \mathbb{N}[X]$  polynomial

$\exists \mathbb{A}$  algorithm, takes inputs in  $\Sigma^*$

$\forall x \in \Sigma^* \left[ \mathbb{A} \text{ decides whether } x \in Q \text{ holds} \right.$   
 $\left. \text{in time } \leq f(\kappa(x)) \cdot p(|x|) \right]$

# Fixed-Parameter tractable

A *parameterized problem* is a triple  $\langle Q, \Sigma, \kappa \rangle$  (short:  $\langle Q, \kappa \rangle$ ) where:

- ▷  $Q \subseteq \Sigma^*$  is the set of (*classical*) *problem instances*,
- ▷  $\kappa : \Sigma^* \rightarrow \mathbb{N}$  is a (general) function, *the parameterization*.

## Definition

A parameterized problem  $\langle Q, \Sigma, \kappa \rangle$  is *fixed-parameter tractable* (is in **FPT**) if:

$$\begin{aligned} &\exists f : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \exists p \in \mathbb{N}[X] \text{ polynomial} \\ &\quad \exists \text{ algorithm, takes inputs in } \Sigma^* \\ &\quad \forall x \in \Sigma^* \left[ \text{algorithm decides whether } x \in Q \text{ holds} \right. \\ &\quad \quad \left. \text{in time } \leq f(\kappa(x)) \cdot p(|x|) \right] \end{aligned}$$

## †) Assumptions for a robust fpt-theory

$\kappa(x)$  is *polynomially computable*, or itself *fpt-computable*: for all  $x \in \Sigma^*$  in time  $\leq g(\kappa(x)) \cdot q(|x|)$  for  $g$  computable,  $q \in \mathbb{N}[X]$ .

# Comparing parameterizations

## Definition (computably bounded below)

Let  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  parameterizations.

- ▶  $\kappa_1 \succeq \kappa_2 : \iff \exists g : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \forall x \in \Sigma^* [g(\kappa_1(x)) \geq \kappa_2(x)]$ .
- ▶  $\kappa_1 \approx \kappa_2 : \iff \kappa_1 \succeq \kappa_2 \wedge \kappa_2 \succeq \kappa_1$ .
- ▶  $\kappa_1 \succ \kappa_2 : \iff \kappa_1 \succeq \kappa_2 \wedge \neg(\kappa_2 \succeq \kappa_1)$ .

# Comparing parameterizations

## Definition (computably bounded below)

Let  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  parameterizations.

- ▶  $\kappa_1 \geq \kappa_2 : \iff \exists g : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \forall x \in \Sigma^* [g(\kappa_1(x)) \geq \kappa_2(x)]$ .
- ▶  $\kappa_1 \approx \kappa_2 : \iff \kappa_1 \geq \kappa_2 \wedge \kappa_2 \geq \kappa_1$ .
- ▶  $\kappa_1 > \kappa_2 : \iff \kappa_1 \geq \kappa_2 \wedge \neg(\kappa_2 \geq \kappa_1)$ .

## Proposition

For all parameterized problems  $\langle Q, \kappa_1 \rangle$  and  $\langle Q, \kappa_2 \rangle$  with parameterizations  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  with  $\kappa_1 \geq \kappa_2$ :

$$\langle Q, \kappa_1 \rangle \in \text{FPT} \iff \langle Q, \kappa_2 \rangle \in \text{FPT}$$

# Comparing parameterizations

## Definition (computably bounded below)

Let  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  parameterizations.

- ▶  $\kappa_1 \geq \kappa_2 : \iff \exists g : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \forall x \in \Sigma^* [g(\kappa_1(x)) \geq \kappa_2(x)]$ .
- ▶  $\kappa_1 \approx \kappa_2 : \iff \kappa_1 \geq \kappa_2 \wedge \kappa_2 \geq \kappa_1$ .
- ▶  $\kappa_1 > \kappa_2 : \iff \kappa_1 \geq \kappa_2 \wedge \neg(\kappa_2 \geq \kappa_1)$ .

## Proposition

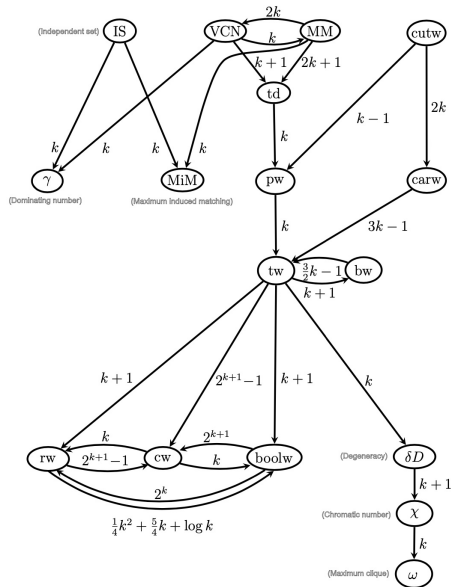
For all parameterized problems  $\langle Q, \kappa_1 \rangle$  and  $\langle Q, \kappa_2 \rangle$  with parameterizations  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  with  $\kappa_1 \geq \kappa_2$ :

$$\begin{aligned} \langle Q, \kappa_1 \rangle \in \text{FPT} &\iff \langle Q, \kappa_2 \rangle \in \text{FPT} \\ \langle Q, \kappa_1 \rangle \notin \text{FPT} &\implies \langle Q, \kappa_2 \rangle \notin \text{FPT} \end{aligned}$$

# Computably boundedness between notions of width

(from Sasák, [5])

$$wd_1 \succeq wd_2 : \Leftrightarrow wd_1 \xrightarrow{g} wd_2$$



# Computably boundedness between notions of width

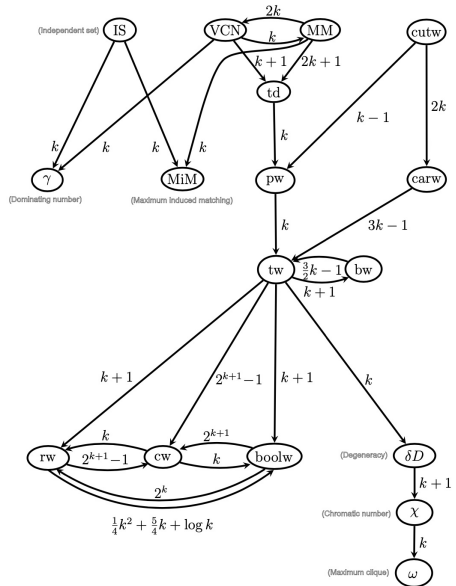
(from Sasák, [5])

$$wd_1 \geq wd_2 : \Leftrightarrow wd_1 \xrightarrow{g} wd_2$$

## ► FPT-results

transfer upwards

(and conversely to  $\xrightarrow{g}$ )





# Computably boundedness between notions of width

(from Sasák, [5])

$$wd_1 \geq wd_2 : \Leftrightarrow wd_1 \xrightarrow{g} wd_2$$

► FPT-results

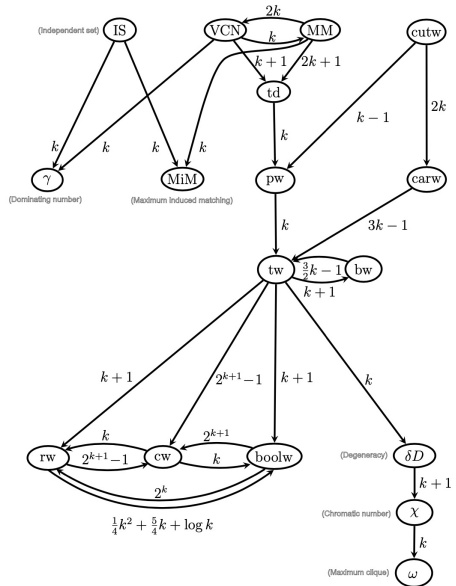
transfer upwards

(and conversely to  $\xrightarrow{g}$ )

► ( $\notin$  FPT)-results

transfer downwards

(and along  $\xrightarrow{g}$ )



# You Always Walk Alone (with your children)

Attività motoria con i figli:

*'la possibilità di uscire con i figli minori è consentita a un solo genitore per camminare purché questo avvenga in prossimità della propria abitazione'*

(Ministero dell'Interno)

# You Always Walk Alone (with your children)

Attività motoria **con i figli**:

*'la possibilità di uscire con i figli minori è consentita a un solo genitore per camminare purché questo avvenga in prossimità della propria abitazione'*

(Ministero dell'Interno)

## PHYSICAL-DISTANCE-WALKING

**Instance:** Graph  $\mathcal{G} = \langle V, E \rangle$  with  $V$  people who want to go for a walk in the next hour in a radius of 200m of their home, and edges in  $E$  between them if they live closer than 400m of each other. A number  $\ell \in \mathbb{N}$ .

**Problem:**

# You Always Walk Alone (with your children)

Attività motoria **con i figli**:

*'la possibilità di uscire con i figli minori è consentita a un solo genitore per camminare purché questo avvenga in prossimità della propria abitazione'*

(Ministero dell'Interno)

## PHYSICAL-DISTANCE-WALKING

**Instance:** Graph  $\mathcal{G} = \langle V, E \rangle$  with  $V$  people who want to go for a walk in the next hour in a radius of 200m of their home, and edges in  $E$  between them if they live closer than 400m of each other. A number  $\ell \in \mathbb{N}$ .

**Problem:** Is it possible that  $\ell$  or more people can go out in the next hour so that everybody walks alone (with their children)?

# You Always Walk Alone (with your children)

Attività motoria **con i figli**:

*'la possibilità di uscire con i figli minori è consentita a un solo genitore per camminare purché questo avvenga in prossimità della propria abitazione'*

(Ministero dell'Interno)

## PHYSICAL-DISTANCE-WALKING

**Instance:** Graph  $\mathcal{G} = \langle V, E \rangle$  with  $V$  people who want to go for a walk in the next hour in a radius of 200m of their home, and edges in  $E$  between them if they live closer than 400m of each other. A number  $\ell \in \mathbb{N}$ .

**Problem:** Is it possible that  $\ell$  or more people can go out in the next hour so that everybody walks alone (with their children)?

corresponds to: INDEPENDENT-SET

# Weighted Independent Set, and Vertex Cover

Let  $\mathcal{G} = \langle V, E \rangle$  a graph. For all  $S \subseteq V$ :

$S$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( \neg(u \in S \wedge v \in S) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin S \vee v \notin S )$

## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

# Weighted Independent Set, and Vertex Cover

Let  $\mathcal{G} = \langle V, E \rangle$  a graph. For all  $S \subseteq V$ :

$S$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( \neg(u \in S \wedge v \in S) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin S \vee v \notin S )$

## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

$S$  is a **vertex cover** of  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( u \in S \vee v \in S )$

# Weighted Independent Set, and Vertex Cover

Let  $\mathcal{G} = \langle V, E \rangle$  a graph. For all  $S \subseteq V$ :

$S$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( \neg(u \in S \wedge v \in S) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin S \vee v \notin S )$

## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

$S$  is a **vertex cover** of  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( u \in S \vee v \in S )$

## VERTEX-COVER

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and  $\ell \in \mathbb{N}$ .

**Problem:** Does  $\mathcal{G}$  have a vertex cover of size at most  $\ell$ ?



# Weighted Independent Set, and Vertex Cover

Let  $\mathcal{G} = \langle V, E \rangle$  a graph. For all  $S \subseteq V$ :

$S$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( \neg(u \in S \wedge v \in S) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin S \vee v \notin S )$

## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

$S$  is a **vertex cover** of  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( u \in S \vee v \in S ) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin V \setminus S \vee v \notin V \setminus S ) )$   
 $\iff V \setminus S$  is an independent set of  $\mathcal{G}$

## VERTEX-COVER

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and  $\ell \in \mathbb{N}$ .

**Problem:** Does  $\mathcal{G}$  have a vertex cover of size at most  $\ell$ ?

# Weighted Independent Set, and Vertex Cover

Let  $\mathcal{G} = \langle V, E \rangle$  a graph. For all  $S \subseteq V$ :

$S$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( \neg(u \in S \wedge v \in S) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin S \vee v \notin S )$

## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

$S$  is a **vertex cover** of  $\mathcal{G} : \iff \forall e = \{u, v\} \in E \ ( u \in S \vee v \in S ) )$   
 $\iff \forall e = \{u, v\} \in E \ ( u \notin V \setminus S \vee v \notin V \setminus S ) )$   
 $\iff V \setminus S$  is an independent set of  $\mathcal{G}$

## VERTEX-COVER

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and  $\ell \in \mathbb{N}$ .

**Problem:** Does  $\mathcal{G}$  have a vertex cover of size at most  $\ell$ ?

$S \subseteq V$  is **minimal** vertex cover  $\iff V \setminus S$  is **maximal** independent set

Hence: solution of WEIGHTED-INDEPENDENT-SET

$\implies$  solution of VERTEX-COVER.

# Weighted Ind. Set / Vertex Cover, width-parameterized

## $p^*$ -WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a weight function  $w : V \rightarrow \mathbb{R}_0^+$ .

**Parameter:** path-width / tree-width  $k$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{G}$ ?

## $p^*$ -VERTEX-COVER

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and  $\ell \in \mathbb{N}$ .

**Parameter:** path-width / tree-width  $k$ .

**Problem:** Does  $\mathcal{G}$  have a vertex cover of size at most  $\ell$ ?

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- ▶  $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,
- ▶  $B[v] :=$  max. weight of an ind. set in  $\mathcal{T}_v$  that does not contain  $v$ .

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- ▶  $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,
- ▶  $B[v] :=$  max. weight of an ind. set in  $\mathcal{T}_v$  that does not contain  $v$ .

Computation of  $A[v]$  and  $B[v]$ :

- ▶ in leafs:  $B[v] = 0$ ,

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- ▶  $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,
- ▶  $B[v] :=$  max. weight of an ind. set in  $\mathcal{T}_v$  that does not contain  $v$ .

Computation of  $A[v]$  and  $B[v]$ :

- ▶ in leafs:  $B[v] = 0, \quad A[v] = w(v).$



# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- ▶  $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,
- ▶  $B[v] :=$  max. weight of an ind. set in  $\mathcal{T}_v$  that does not contain  $v$ .

Computation of  $A[v]$  and  $B[v]$ :

- ▶ in leafs:  $B[v] = 0$ ,  $A[v] = w(v)$ .
- ▶ for inner vertices  $v$  with children  $v_1, \dots, v_q$ :

$$B[v] = \sum_{i=1}^q A[v_i],$$

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- ▶  $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,
- ▶  $B[v] :=$  max. weight of an ind. set in  $\mathcal{T}_v$  that does not contain  $v$ .

Computation of  $A[v]$  and  $B[v]$ :

- ▶ in leafs:  $B[v] = 0$ ,  $A[v] = w(v)$ .
- ▶ for inner vertices  $v$  with children  $v_1, \dots, v_q$ :

$$B[v] = \sum_{i=1}^q A[v_i], \quad A[v] = \max\{B[v], w(v) + \sum_{i=1}^q B[v_i]\}.$$

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

Obtain a directed tree  $\mathcal{T} = \langle T, F, r \rangle$  (pick a root  $r$ , orient edges away).

- ▶  $A[v] :=$  max. weight of an independent set in subtree  $\mathcal{T}_v$  at  $v$ ,
- ▶  $B[v] :=$  max. weight of an ind. set in  $\mathcal{T}_v$  that does not contain  $v$ .

Computation of  $A[v]$  and  $B[v]$ :

- ▶ in leafs:  $B[v] = 0$ ,  $A[v] = w(v)$ .
- ▶ for inner vertices  $v$  with children  $v_1, \dots, v_q$ :

$$B[v] = \sum_{i=1}^q A[v_i], \quad A[v] = \max\left\{B[v], w(v) + \sum_{i=1}^q B[v_i]\right\}.$$

**Solution:** value of  $A[r]$ , can be computed bottom-up in linear time.

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

## Theorem

On *trees* with  $n$  nodes,

WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(O(n))$ .

# Dynamical programming on trees (example)

## WEIGHTED-INDEPENDENT-SET

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and a weight function  $w : T \rightarrow \mathbb{R}_0^+$ .

**Problem:** What is the max. weight of an independent set of  $\mathcal{T}$ ?

## Theorem

On trees with  $n$  nodes,

WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(O(n))$ .

## VERTEX-COVER

**Instance:** A tree  $\mathcal{T} = \langle T, F \rangle$ , and  $\ell \in \mathbb{N}$ .

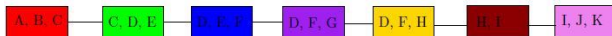
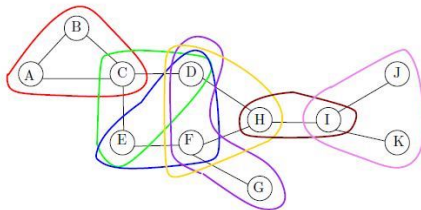
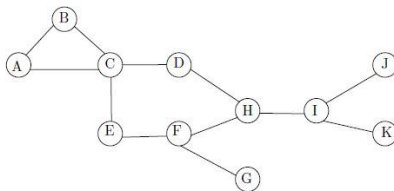
**Problem:** Does  $\mathcal{T}$  have a vertex cover of size at most  $\ell$ ?

## Corollary

On trees with  $n$  nodes,

VERTEX-COVER  $\in \text{DTIME}(O(n))$ .

# Path-decomposition (example)



# Path decompositions, and path-width

## Definition (Robertson–Seymour, 1983)

A *path decomposition* of a graph  $\mathcal{G} = \langle V, E \rangle$  is a sequence  $\langle B_1, B_2, \dots, B_r \rangle$  of bags  $B_i \subseteq V$  such that:

(P1)  $V = \bigcup_{i=1}^r B_i$  (every vertex of  $\mathcal{G}$  is in some bag).

# Path decompositions, and path-width

## Definition (Robertson–Seymour, 1983)

A **path decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a sequence  $\langle B_1, B_2, \dots, B_r \rangle$  of bags  $B_i \subseteq V$  such that:

(P1)  $V = \bigcup_{i=1}^r B_i$  (every vertex of  $\mathcal{G}$  is in some bag).

(P2)  $(\forall \{u, v\} \in E) (\exists i \in \{1, 2, \dots, r\}) [\{u, v\} \subseteq B_i]$   
(every edge of  $\mathcal{G}$  is realized in some bag).



# Path decompositions, and path-width

## Definition (Robertson–Seymour, 1983)

A **path decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a sequence  $\langle B_1, B_2, \dots, B_r \rangle$  of bags  $B_i \subseteq V$  such that:

- (P1)  $V = \bigcup_{i=1}^r B_i$  (every vertex of  $\mathcal{G}$  is in some bag).
- (P2)  $(\forall \{u, v\} \in E) (\exists i \in \{1, 2, \dots, r\}) [\{u, v\} \subseteq B_i]$   
(every edge of  $\mathcal{G}$  is realized in some bag).
- (P3)  $(\forall v \in V) (\exists i, k \in \{1, \dots, r\}, i \leq k) [\{j \mid v \in B_j\} = [i, k]]$   
(the list of bags that contains a vertex of  $\mathcal{G}$  is  $\langle B_i, \dots, B_k \rangle$  for some interval  $[i, k]$ )

# Path decompositions, and path-width

## Definition (Robertson–Seymour, 1983)

A **path decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a sequence  $\langle B_1, B_2, \dots, B_r \rangle$  of bags  $B_i \subseteq V$  such that:

(P1)  $V = \bigcup_{i=1}^r B_i$  (every vertex of  $\mathcal{G}$  is in some bag).

(P2)  $(\forall \{u, v\} \in E) (\exists i \in \{1, 2, \dots, r\}) [\{u, v\} \subseteq B_i]$   
(every edge of  $\mathcal{G}$  is realized in some bag).

(P3)  $(\forall v \in V) (\exists i, k \in \{1, \dots, r\}, i \leq k) [\{j \mid v \in B_j\} = [i, k]]$   
(the list of bags that contains a vertex of  $\mathcal{G}$  is  $\langle B_i, \dots, B_k \rangle$  for some interval  $[i, k]$ )

The **width** of path decomp.  $\langle B_1, B_2, \dots, B_r \rangle$  is  $\max \{|B_t| - 1 \mid 1 \leq t \leq r\}$ .

# Path decompositions, and path-width

## Definition (Robertson–Seymour, 1983)

A **path decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a sequence  $\langle B_1, B_2, \dots, B_r \rangle$  of bags  $B_i \subseteq V$  such that:

(P1)  $V = \bigcup_{i=1}^r B_i$  (every vertex of  $\mathcal{G}$  is in some bag).

(P2)  $(\forall \{u, v\} \in E) (\exists i \in \{1, 2, \dots, r\}) [\{u, v\} \subseteq B_i]$   
(every edge of  $\mathcal{G}$  is realized in some bag).

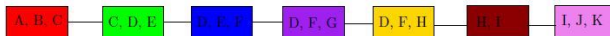
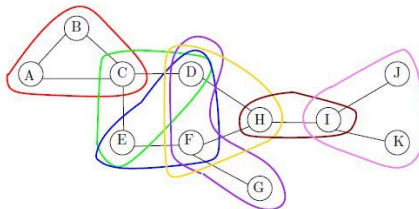
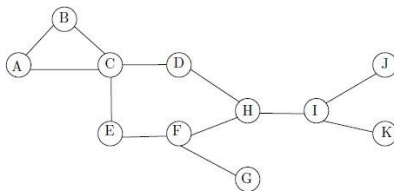
(P3)  $(\forall v \in V) (\exists i, k \in \{1, \dots, r\}, i \leq k) [\{j \mid v \in B_j\} = [i, k]]$   
(the list of bags that contains a vertex of  $\mathcal{G}$  is  $\langle B_i, \dots, B_k \rangle$  for some interval  $[i, k]$ )

The **width** of path decomp.  $\langle B_1, B_2, \dots, B_r \rangle$  is  $\max \{|B_t| - 1 \mid 1 \leq t \leq r\}$ .

The **path-width**  $\text{pw}(\mathcal{G})$  of a graph  $\mathcal{G} = \langle V, E \rangle$  is defined by:

$\text{pw}(\mathcal{G}) :=$  minimal width of a path decomposition of  $\mathcal{G}$ .

# Path-decomposition (example)



# Path decomposition defines separations

## Lemma

*Let  $\langle B_1, B_2, \dots, B_r \rangle$  be a path decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Then for all  $i \in \{1, \dots, r-1\}$  it holds:*

- ▶  *$\langle \bigcup_{j=1}^i B_j, \bigcup_{j=i+1}^r B_j \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_i \cap B_{i+1}$ .*

# Path decomposition defines separations

## Lemma

Let  $\langle B_1, B_2, \dots, B_r \rangle$  be a path decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Then for all  $i \in \{1, \dots, r-1\}$  it holds:

▶  $\langle \bigcup_{j=1}^i B_j, \bigcup_{j=i+1}^r B_j \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_i \cap B_{i+1}$ .

▶ A pair  $\langle A, B \rangle$  of subsets  $A, B \subseteq V$  is a *separation* of  $\mathcal{G}$  if:

▶  $V = A \cup B$

▶ there is **no edge** between  $A \setminus B$  and  $B \setminus A$ .

$A \cap B$  is called the *separator* of a separation  $\langle A, B \rangle$ , and  $|A \cap B|$  is called its *order*.

# Path decomposition defines separations

## Lemma

Let  $\langle B_1, B_2, \dots, B_r \rangle$  be a path decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Then for all  $i \in \{1, \dots, r-1\}$  it holds:

- ▶  $\langle \bigcup_{j=1}^i B_j, \bigcup_{j=i+1}^r B_j \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_i \cap B_{i+1}$ .
- ▶  $\partial(\bigcup_{j=1}^i B_j) \subseteq B_i \cap B_{i+1}$ .

- ▶ A pair  $\langle A, B \rangle$  of subsets  $A, B \subseteq V$  is a *separation* of  $\mathcal{G}$  if:

- ▶  $V = A \cup B$
- ▶ there is **no edge** between  $A \setminus B$  and  $B \setminus A$ .

$A \cap B$  is called the *separator* of a separation  $\langle A, B \rangle$ , and  $|A \cap B|$  is called its *order*.

# Path decomposition defines separations

## Lemma

Let  $\langle B_1, B_2, \dots, B_r \rangle$  be a path decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Then for all  $i \in \{1, \dots, r-1\}$  it holds:

- ▶  $\langle \bigcup_{j=1}^i B_j, \bigcup_{j=i+1}^r B_j \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_i \cap B_{i+1}$ .
- ▶  $\partial(\bigcup_{j=1}^i B_j) \subseteq B_i \cap B_{i+1}$ .

- ▶ A pair  $\langle A, B \rangle$  of subsets  $A, B \subseteq V$  is a *separation* of  $\mathcal{G}$  if:

- ▶  $V = A \cup B$
- ▶ there is **no edge** between  $A \setminus B$  and  $B \setminus A$ .

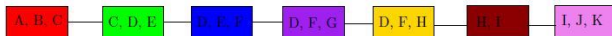
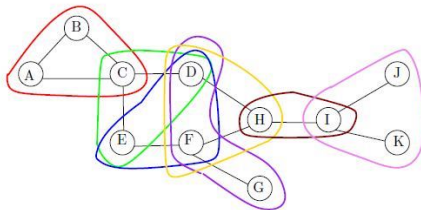
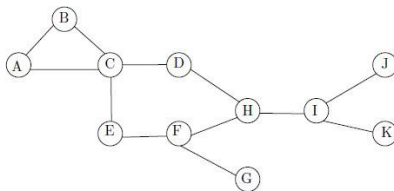
$A \cap B$  is called the *separator* of a separation  $\langle A, B \rangle$ , and  $|A \cap B|$  is called its *order*.

- ▶ The *border (set of border vertices)*  $\partial(A)$  of a set  $A \subseteq V$  of vertices consists of all vertices that have a neighbor in  $V \setminus A$ . Note that:

- ▶  $\partial(A) = \partial(V \setminus A)$ .
- ▶  $\langle A, (V \setminus A) \cup \partial(A) \rangle$  is a separation of  $\mathcal{G}$ , for all  $A \subseteq V$ .

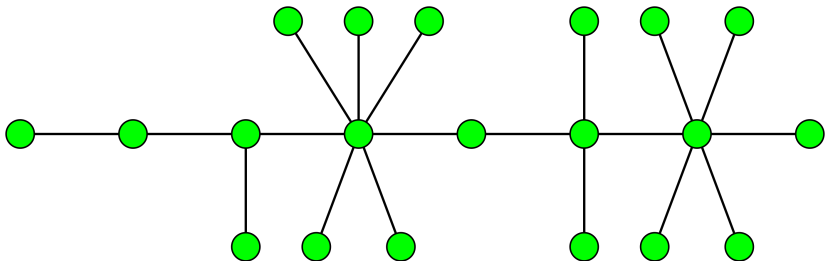


# Path-decomposition (example)



# Caterpillar

Path-width?



# Nice path decomposition

## Definition

A *path decomposition*  $\langle B_1, B_2, \dots, B_r \rangle$  of a graph  $\mathcal{G} = \langle V, E \rangle$  is *nice* if:

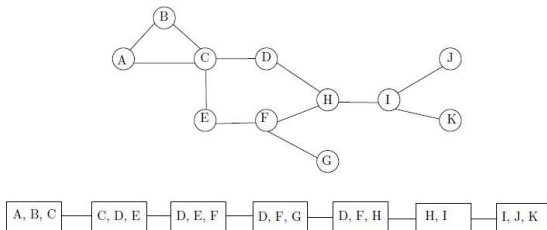
- ▶  $B_1 = B_r = \emptyset$
- ▶ Every index  $i > 1$  is either of:
  - ▶ **introduce index**: there is  $v \in V$  such that  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,
  - ▶ **forget index**: there is  $v \in V$  such that  $B_{i+1} = B_i \setminus \{v\}$  and  $v \in B_i$ .

# Nice path decomposition

## Definition

A *path decomposition*  $\langle B_1, B_2, \dots, B_r \rangle$  of a graph  $\mathcal{G} = \langle V, E \rangle$  is *nice* if:

- ▶  $B_1 = B_r = \emptyset$
- ▶ Every index  $i > 1$  is either of:
  - ▶ **introduce index**: there is  $v \in V$  such that  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,
  - ▶ **forget index**: there is  $v \in V$  such that  $B_{i+1} = B_i \setminus \{v\}$  and  $v \in B_i$ .

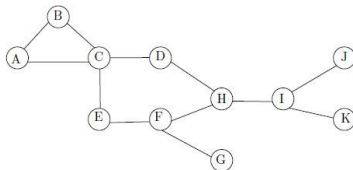


# Nice path decomposition

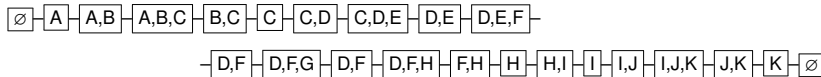
## Definition

A *path decomposition*  $\langle B_1, B_2, \dots, B_r \rangle$  of a graph  $\mathcal{G} = \langle V, E \rangle$  is *nice* if:

- ▶  $B_1 = B_r = \emptyset$
- ▶ Every index  $i > 1$  is either of:
  - ▶ **introduce index**: there is  $v \in V$  such that  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,
  - ▶ **forget index**: there is  $v \in V$  such that  $B_{i+1} = B_i \setminus \{v\}$  and  $v \in B_i$ .



Nice path decomposition:



# Nice path decomposition

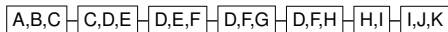
## Definition

A *path decomposition*  $\langle B_1, B_2, \dots, B_r \rangle$  of a graph  $\mathcal{G} = \langle V, E \rangle$  is *nice* if:

- ▶  $B_1 = B_r = \emptyset$
- ▶ Every index  $i > 1$  is either of:
  - ▶ **introduce index**: there is  $v \in V$  such that  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,
  - ▶ **forget index**: there is  $v \in V$  such that  $B_{i+1} = B_i \setminus \{v\}$  and  $v \in B_i$ .

## Lemma

From every *path decomposition*  $\langle B_1, B_2, \dots, B_r \rangle$  of a graph  $\mathcal{G} = \langle V, E \rangle$  of width  $k$  a *nice path decomposition*  $\langle B'_1, B'_2, \dots, B'_r \rangle$  of width  $k$  can be constructed in time  $O(k^2 \cdot \max\{r, n\})$  where  $n := |V|$ .



# Weighted Independent Set

Let  $\mathcal{G} = \langle V, E \rangle$  a graph.

$S \subseteq V$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \left( \neg(u \in S \wedge v \in S) \right)$ .

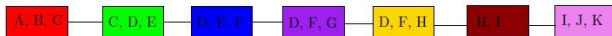
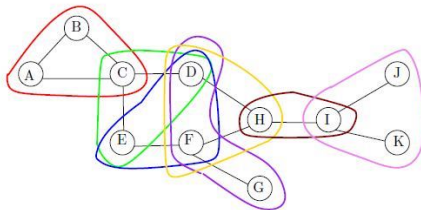
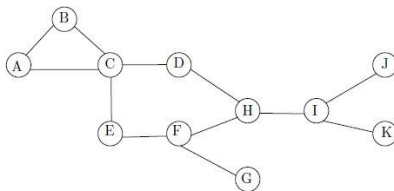
## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Parameter:** **path-width**  $k$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

# Path-decomposition (example)





# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a **nice path decomposition** of  $\mathcal{G} = \langle V, E \rangle$ .

Then for every  $i \in \{1, \dots, r\}$ , and every  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a **nice path decomposition** of  $\mathcal{G} = \langle V, E \rangle$ .

Then for every  $i \in \{1, \dots, r\}$ , and every  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[i, S]$  for **independent**  $S$ :

- Case  $i = 1$ :  $c[1, \emptyset] = 0$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a **nice path decomposition** of  $\mathcal{G} = \langle V, E \rangle$ .

Then for every  $i \in \{1, \dots, r\}$ , and every  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[i, S]$  for **independent**  $S$ :

► **Case**  $i = 1$ :  $c[1, \emptyset] = 0$

► **Case**  $i + 1$ :

►  $i + 1$  **introduces**  $v$ :  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,

$$c[i + 1, S] = \begin{cases} c[i, S] & \text{if } v \notin S, \end{cases}$$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a **nice path decomposition** of  $\mathcal{G} = \langle V, E \rangle$ .

Then for every  $i \in \{1, \dots, r\}$ , and every  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[i, S]$  for **independent**  $S$ :

- ▶ **Case**  $i = 1$ :  $c[1, \emptyset] = 0$
- ▶ **Case**  $i + 1$ :
  - ▶  $i + 1$  **introduces**  $v$ :  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,

$$c[i + 1, S] = \begin{cases} c[i, S] & \text{if } v \notin S, \\ c[i, S \setminus \{v\}] + w(v) & \text{if } v \in S; \end{cases}$$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a **nice path decomposition** of  $\mathcal{G} = \langle V, E \rangle$ .

Then for every  $i \in \{1, \dots, r\}$ , and every  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[i, S]$  for **independent**  $S$ :

- ▶ **Case**  $i = 1$ :  $c[1, \emptyset] = 0$
- ▶ **Case**  $i + 1$ :
  - ▶  $i + 1$  **introduces**  $v$ :  $B_{i+1} = B_i \cup \{v\}$  and  $v \notin B_i$ ,
 
$$c[i + 1, S] = \begin{cases} c[i, S] & \text{if } v \notin S, \\ c[i, S \setminus \{v\}] + \mathbf{w}(v) & \text{if } v \in S; \end{cases}$$
  - ▶  $i + 1$  **forgets**  $v$ :  $B_{i+1} = B_i \setminus \{v\}$  and  $v \in B_i$ ,
 
$$c[i + 1, S] = \max\{c[i, S], c[i, S \cup \{v\}]\}.$$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$



# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $i \in \{1, \dots, n\}$ :

►  $|B_i| \leq k + 1,$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $i \in \{1, \dots, n\}$ :

- ▶  $|B_i| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[i, S]$  at index  $i$ :  $2^{|B_i|} = 2^{k+1}$ ,

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $i \in \{1, \dots, n\}$ :

- ▶  $|B_i| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[i, S]$  at index  $i$ :  $2^{|B_i|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  adjacency/independence check for  $S \subseteq B_t$  possible in:  $O(k^2)$  using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $i \in \{1, \dots, n\}$ :

- ▶  $|B_i| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[i, S]$  at index  $i$ :  $2^{|B_i|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  adjacency/independence check for  $S \subseteq B_t$  possible in:  $O(k^2)$   
using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,
- ▶ time for comp. a value at  $i$ , using map of values at  $i - 1$ :  $\sim O(k)$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $i \in \{1, \dots, n\}$ :

- ▶  $|B_i| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[i, S]$  at index  $i$ :  $2^{|B_i|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  adjacency/independence check for  $S \subseteq B_t$  possible in:  $O(k^2)$   
using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,
- ▶ time for comp. a value at  $i$ , using map of values at  $i - 1$ :  $\sim O(k)$
- ▶ time for comp. all values at  $i$ , using values at  $i - 1$ :  $2^{k+1} \cdot O(k^2)$

# Dyn. programming using path-width (Weigh. Ind. Set)

Let  $\langle B_1, \dots, B_r \rangle$  be a nice path dec. of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ .

For every  $i \in \{1, \dots, r\}$ , and every independent  $S \subseteq B_i$ , we define:

$$c[i, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_i = \bigcup_{j=1}^i B_j \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[i, S]$ , the maximum possible weight of an independent set  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $i \in \{1, \dots, n\}$ :

- ▶  $|B_i| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[i, S]$  at index  $i$ :  $2^{|B_i|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  adjacency/independence check for  $S \subseteq B_t$  possible in:  $O(k^2)$  using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,
- ▶ time for comp. a value at  $i$ , using map of values at  $i - 1$ :  $\sim O(k)$
- ▶ time for comp. all values at  $i$ , using values at  $i - 1$ :  $2^{k+1} \cdot O(k^2)$

$\Rightarrow$  the time for computing all values at  $r$ :

$$(2^{k+1} \cdot O(k^2)) \cdot r + O(k^{O(1)} \cdot n) \in O(2^k \cdot k^{O(1)} \cdot n), \text{ since } r = 2n.$$

# Dynamical programming with path width (example)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *path-width*  $\text{pw}(\mathcal{G}) = k$ ,  
 $p^*$ -WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

# Dynamical programming with path width (example)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *path-width*  $\text{pw}(\mathcal{G}) = k$ ,  
 $p^*$ -WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

$S$  is a *minimal* vertex cover

$\iff V \setminus S$  is a *maximal* independent set.



# Dynamical programming with path width (example)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *path-width*  $\text{pw}(\mathcal{G}) = k$ ,  
 $p^*$ -WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

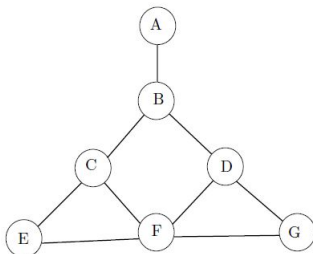
$S$  is a *minimal* vertex cover

$\iff V \setminus S$  is a *maximal* independent set.

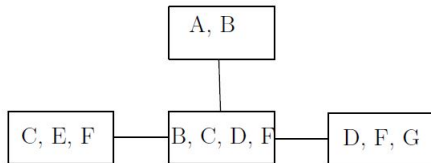
## Corollary

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *path-width*  $\text{pw}(\mathcal{G}) = k$ ,  
 $p^*$ -VERTEX-COVER  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

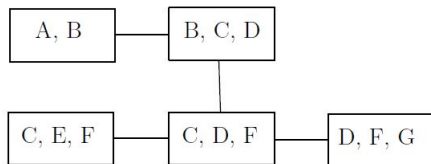
# Tree decomposition (example)



The original graph  $G$



A tree-decomposition of width 3



A tree-decomposition of width 2

# Tree decompositions, and tree-width

Definition (Bertelé–Brioschi, 1972, Halin, 1976, Robertson–Seymour, 1984)

A *tree decomposition* of a graph  $\mathcal{G} = \langle V, E \rangle$  is a pair  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  where  $\mathcal{T} = \langle T, F \rangle$  a (undirected, unrooted) tree, and  $B_t \subseteq V$  such that:

(T1)  $V = \bigcup_{t \in T} B_t$  (every vertex of  $\mathcal{G}$  is in some bag).

# Tree decompositions, and tree-width

Definition (Bertelé–Brioschi, 1972, Halin, 1976, Robertson–Seymour, 1984)

A *tree decomposition* of a graph  $\mathcal{G} = \langle V, E \rangle$  is a pair  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  where  $\mathcal{T} = \langle T, F \rangle$  a (undirected, unrooted) tree, and  $B_t \subseteq V$  such that:

(T1)  $V = \bigcup_{t \in T} B_t$  (every vertex of  $\mathcal{G}$  is in some bag).

(T2)  $(\forall \{u, v\} \in E) (\exists t \in T) [\{u, v\} \subseteq B_t]$   
(the vertices of every edge of  $\mathcal{G}$  are realized in some bag).

# Tree decompositions, and tree-width

Definition (Bertelé–Brioschi, 1972, Halin, 1976, Robertson–Seymour, 1984)

A **tree decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a pair  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  where  $\mathcal{T} = \langle T, F \rangle$  a (undirected, unrooted) tree, and  $B_t \subseteq V$  such that:

(T1)  $V = \bigcup_{t \in T} B_t$  (every vertex of  $\mathcal{G}$  is in some bag).

(T2)  $(\forall \{u, v\} \in E) (\exists t \in T) [\{u, v\} \subseteq B_t]$   
(the vertices of every edge of  $\mathcal{G}$  are realized in some bag).

(T3)  $(\forall v \in V) [\{t \in T \mid v \in B_t\} \text{ is connected subgraph of } \mathcal{T}]$   
(the tree vertices whose bags contain some vertex of  $\mathcal{G}$  form a connected subgraph of  $\mathcal{T}$ ).

# Tree decompositions, and tree-width

Definition (Bertelé–Brioschi, 1972, Halin, 1976, Robertson–Seymour, 1984)

A **tree decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a pair  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  where  $\mathcal{T} = \langle T, F \rangle$  a (undirected, unrooted) tree, and  $B_t \subseteq V$  such that:

(T1)  $V = \bigcup_{t \in T} B_t$  (every vertex of  $\mathcal{G}$  is in some bag).

(T2)  $(\forall \{u, v\} \in E) (\exists t \in T) [\{u, v\} \subseteq B_t]$   
(the vertices of every edge of  $\mathcal{G}$  are realized in some bag).

(T3)  $(\forall v \in V) [\{t \in T \mid v \in B_t\} \text{ is connected subgraph of } \mathcal{T}]$   
(the tree vertices whose bags contain some vertex of  $\mathcal{G}$  form a connected subgraph of  $\mathcal{T}$ ).

The **width** of a tree decomposition  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  is  

$$\max \{|B_t| - 1 \mid t \in T\}.$$

# Tree decompositions, and tree-width

Definition (Bertelé–Brioschi, 1972, Halin, 1976, Robertson–Seymour, 1984)

A **tree decomposition** of a graph  $\mathcal{G} = \langle V, E \rangle$  is a pair  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  where  $\mathcal{T} = \langle T, F \rangle$  a (undirected, unrooted) tree, and  $B_t \subseteq V$  such that:

(T1)  $V = \bigcup_{t \in T} B_t$  (every vertex of  $\mathcal{G}$  is in some bag).

(T2)  $(\forall \{u, v\} \in E) (\exists t \in T) [\{u, v\} \subseteq B_t]$   
(the vertices of every edge of  $\mathcal{G}$  are realized in some bag).

(T3)  $(\forall v \in V) [\{t \in T \mid v \in B_t\} \text{ is connected subgraph of } \mathcal{T}]$   
(the tree vertices whose bags contain some vertex of  $\mathcal{G}$  form a connected subgraph of  $\mathcal{T}$ ).

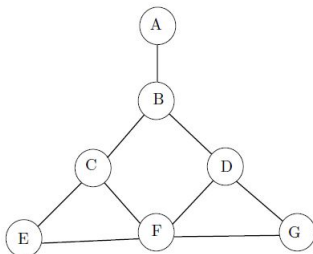
The **width** of a tree decomposition  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  is  

$$\max \{|B_t| - 1 \mid t \in T\}.$$

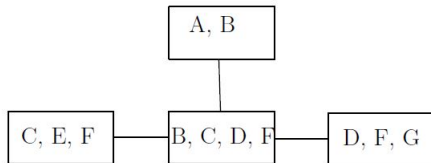
The **tree-width**  $tw(\mathcal{G})$  of a graph  $\mathcal{G} = \langle V, E \rangle$  is defined by:

$tw(\mathcal{G}) :=$  minimal width of a tree decomposition of  $\mathcal{G}$ .

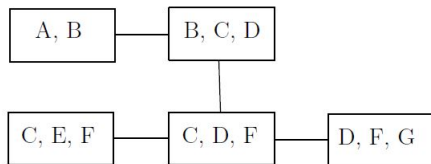
# Tree decomposition (example)



The original graph  $G$



A tree-decomposition of width 3



A tree-decomposition of width 2



# Tree decomposition defines separations

## Lemma

Let  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  be a tree decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ .  
 Let  $e = \langle a, b \rangle$  be an edge of  $\mathcal{T}$ . The  $\mathcal{T} \setminus e$  is the union of a tree  $\mathcal{T}_a$  containing  $a$ , and a tree  $\mathcal{T}_b$  containing  $b$ .

Then for  $A := \bigcup_{t \in V(\mathcal{T}_a)} B_t$  and  $B := \bigcup_{t \in V(\mathcal{T}_b)} B_t$  it holds:

- ▶  $\langle A, B \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_a \cap B_b$ .

Recall, for a graph  $\mathcal{G} = \langle V, E \rangle$ :

# Tree decomposition defines separations

## Lemma

Let  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  be a tree decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Let  $e = \langle a, b \rangle$  be an edge of  $\mathcal{T}$ . The  $\mathcal{T} \setminus e$  is the union of a tree  $\mathcal{T}_a$  containing  $a$ , and a tree  $\mathcal{T}_b$  containing  $b$ .

Then for  $A := \bigcup_{t \in V(\mathcal{T}_a)} B_t$  and  $B := \bigcup_{t \in V(\mathcal{T}_b)} B_t$  it holds:

- ▶  $\langle A, B \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_a \cap B_b$ .

Recall, for a graph  $\mathcal{G} = \langle V, E \rangle$ :

- ▶ A pair  $\langle A, B \rangle$  of subsets  $A, B \subseteq V$  is a *separation* of  $\mathcal{G}$  if:
  - ▶  $V = A \cup B$
  - ▶ there is **no edge** between  $A \setminus B$  and  $B \setminus A$ .

$A \cap B$  is called the *separator* of a separation  $\langle A, B \rangle$ , and  $|A \cap B|$  is called its *order*.

# Tree decomposition defines separations

## Lemma

Let  $\langle \mathcal{T}, \{B_t\}_{t \in \mathcal{T}} \rangle$  be a tree decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Let  $e = \langle a, b \rangle$  be an edge of  $\mathcal{T}$ . The  $\mathcal{T} \setminus e$  is the union of a tree  $\mathcal{T}_a$  containing  $a$ , and a tree  $\mathcal{T}_b$  containing  $b$ .

Then for  $A := \bigcup_{t \in V(\mathcal{T}_a)} B_t$  and  $B := \bigcup_{t \in V(\mathcal{T}_b)} B_t$  it holds:

- ▶  $\langle A, B \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_a \cap B_b$ .
- ▶  $\partial(A), \partial(B) \subseteq B_a \cap B_b$ .

Recall, for a graph  $\mathcal{G} = \langle V, E \rangle$ :

- ▶ A pair  $\langle A, B \rangle$  of subsets  $A, B \subseteq V$  is a *separation* of  $\mathcal{G}$  if:
  - ▶  $V = A \cup B$
  - ▶ there is **no edge** between  $A \setminus B$  and  $B \setminus A$ .

$A \cap B$  is called the *separator* of a separation  $\langle A, B \rangle$ , and  $|A \cap B|$  is called its *order*.

# Tree decomposition defines separations

## Lemma

Let  $\langle \mathcal{T}, \{B_t\}_{t \in \mathcal{T}} \rangle$  be a tree decomposition of a graph  $\mathcal{G} = \langle V, E \rangle$ . Let  $e = \langle a, b \rangle$  be an edge of  $\mathcal{T}$ . The  $\mathcal{T} \setminus e$  is the union of a tree  $\mathcal{T}_a$  containing  $a$ , and a tree  $\mathcal{T}_b$  containing  $b$ .

Then for  $A := \bigcup_{t \in V(\mathcal{T}_a)} B_t$  and  $B := \bigcup_{t \in V(\mathcal{T}_b)} B_t$  it holds:

- ▶  $\langle A, B \rangle$  is a separation of  $\mathcal{G}$  with separator  $B_a \cap B_b$ .
- ▶  $\partial(A), \partial(B) \subseteq B_a \cap B_b$ .

Recall, for a graph  $\mathcal{G} = \langle V, E \rangle$ :

- ▶ A pair  $\langle A, B \rangle$  of subsets  $A, B \subseteq V$  is a *separation* of  $\mathcal{G}$  if:
  - ▶  $V = A \cup B$
  - ▶ there is **no edge** between  $A \setminus B$  and  $B \setminus A$ .

$A \cap B$  is called the *separator* of a separation  $\langle A, B \rangle$ , and  $|A \cap B|$  is called its *order*.

- ▶ The *border (vertices)*  $\partial(A)$  of a set  $A \subseteq V$  of vertices consists of all vertices that have a neighbor in  $V \setminus A$ .

# Computing tree-width

## TREE-WIDTH

**Instance:** A graph  $\mathcal{G}$  and  $k \in \mathbb{N}$ .

**Problem:** Decide whether  $tw(\mathcal{G}) = k$ .

# Computing tree-width

## TREE-WIDTH

**Instance:** A graph  $\mathcal{G}$  and  $k \in \mathbb{N}$ .

**Problem:** Decide whether  $tw(\mathcal{G}) = k$ .

## Theorem

TREE-WIDTH is NP-complete.

# Computing tree-width

## TREE-WIDTH

**Instance:** A graph  $\mathcal{G}$  and  $k \in \mathbb{N}$ .

**Problem:** Decide whether  $tw(\mathcal{G}) = k$ .

## Theorem

TREE-WIDTH is NP-complete.

## $p$ -TREE-WIDTH

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$  and  $k \in \mathbb{N}$ .

**Parameter:**  $k$ .

**Problem:** Decide whether  $tw(\mathcal{G}) = k$ .

# Computing tree-width

## TREE-WIDTH

**Instance:** A graph  $\mathcal{G}$  and  $k \in \mathbb{N}$ .

**Problem:** Decide whether  $tw(\mathcal{G}) = k$ .

## Theorem

TREE-WIDTH is NP-complete.

## $p$ -TREE-WIDTH

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$  and  $k \in \mathbb{N}$ .

**Parameter:**  $k$ .

**Problem:** Decide whether  $tw(\mathcal{G}) = k$ .

## Theorem

$p$ -TREE-WIDTH is fixed-parameter tractable,  
in time  $2^{p(k)} \cdot n$  where  $n := |V|$ .



# Nice tree decomposition

## Definition

A *tree decomposition*  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  of graph  $\mathcal{G} = \langle V, E \rangle$  is *nice* if it is based on the choice of a leaf as *root*  $r$  and the parent–children relation away from  $r$  such that:

- ▶  $B_r = \emptyset$ , and  $B_\ell = \emptyset$  for every leaf  $\ell \in T$ .
- ▶ Every non-leaf node  $t \in T$  is of one of three types:
  - ▶ **introduce node**:  $t$  has exactly one child  $t'$  such that  $B_t = B_{t'} \cup \{v\}$ ; we say  $v$  is **introduced** at  $t$ .
  - ▶ **forget node**:  $t$  has exactly one child  $t'$  such that  $B_t = B_{t'} \setminus \{w\}$  for some  $w \in B_{t'}$ ; we say  $w$  is **forgotten** at  $t$ .
  - ▶ **join node**: a node  $t$  with two children  $t_1, t_2$  such that  $B_t = B_{t_1} = B_{t_2}$ .

# Nice tree decomposition

## Definition

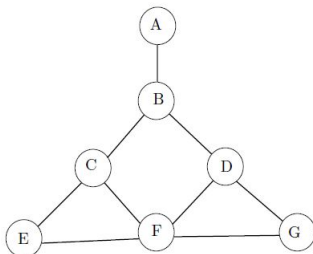
A *tree decomposition*  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  of graph  $\mathcal{G} = \langle V, E \rangle$  is *nice* if it is based on the choice of a leaf as *root*  $r$  and the parent–children relation away from  $r$  such that:

- ▶  $B_r = \emptyset$ , and  $B_\ell = \emptyset$  for every leaf  $\ell \in T$ .
- ▶ Every non-leaf node  $t \in T$  is of one of three types:
  - ▶ *introduce node*:  $t$  has exactly one child  $t'$  such that  $B_t = B_{t'} \cup \{v\}$ ; we say  $v$  is *introduced* at  $t$ .
  - ▶ *forget node*:  $t$  has exactly one child  $t'$  such that  $B_t = B_{t'} \setminus \{w\}$  for some  $w \in B_{t'}$ ; we say  $w$  is *forgotten* at  $t$ .
  - ▶ *join node*: a node  $t$  with two children  $t_1, t_2$  such that  $B_t = B_{t_1} = B_{t_2}$ .

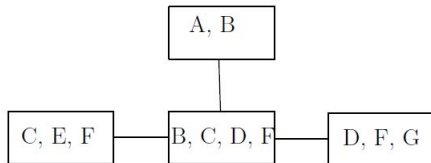
## Lemma

From every *tree decomposition*  $\langle \mathcal{T}, \{B_t\}_{t \in T} \rangle$  of a graph  $\mathcal{G} = \langle V, E \rangle$  of *width*  $k$  a *nice tree decomposition*  $\langle \mathcal{T}', \{B'_t\}_{t \in T'} \rangle$  of *width*  $k$  and with  $r := |V(\mathcal{T})| \in O(kn)$  vertices can be constructed in time  $O(k^2 \cdot \max\{r, n\})$  where  $n := |V|$ .

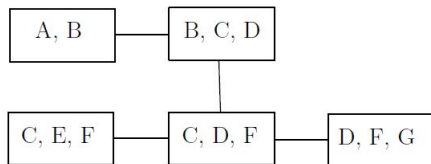
# Tree decomposition (example)



The original graph  $G$



A tree-decomposition of width 3



A tree-decomposition of width 2

# Weighted Independent Set

Let  $\mathcal{G} = \langle V, E \rangle$  a graph.

$S \subseteq V$  is **independent set** in  $\mathcal{G} : \iff \forall e = \{u, v\} \left( \neg(u \in S \wedge v \in S) \right)$ .

## WEIGHTED-INDEPENDENT-SET

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ , and a **weight function**  $w : V \rightarrow \mathbb{R}_0^+$ .

**Parameter:** **tree-width**  $k$ .

**Problem:** What is the **max. weight of an independent set** of  $\mathcal{G}$ ?

# Dynamical programming using tree-width (example)

For every node  $t$  of a [nice tree decomposition](#), and every  $S \subseteq B_t$ , we define:

$$c[t, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t \wedge \hat{S} \cap B_t = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

# Dynamical programming using tree-width (example)

For every node  $t$  of a **nice tree decomposition**, and every  $S \subseteq B_t$ , we define:

$$c[t, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t \wedge \hat{S} \cap B_t = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[t, S]$  for **independent**  $S$ :

- leaf node  $t$ :  $c[t, \emptyset] = 0$

# Dynamical programming using tree-width (example)

For every node  $t$  of a **nice tree decomposition**, and every  $S \subseteq B_t$ , we define:

$$c[t, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t \wedge \hat{S} \cap B_t = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[t, S]$  for **independent**  $S$ :

- ▶ leaf node  $t$ :  $c[t, \emptyset] = 0$
- ▶ introduction node  $t$  of vertex  $v$  with child  $t'$ :

$$c[t, S] = \begin{cases} c[t', S] & \text{if } v \notin S \end{cases}$$

# Dynamical programming using tree-width (example)

For every node  $t$  of a **nice tree decomposition**, and every  $S \subseteq B_t$ , we define:

$$c[t, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t \wedge \hat{S} \cap B_t = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[t, S]$  for **independent**  $S$ :

- ▶ leaf node  $t$ :  $c[t, \emptyset] = 0$
- ▶ introduction node  $t$  of vertex  $v$  with child  $t'$ :

$$c[t, S] = \begin{cases} c[t', S] & \text{if } v \notin S \\ c[t', S \setminus \{v\}] + w(v) & \text{otherwise} \end{cases}$$



# Dynamical programming using tree-width (example)

For every node  $t$  of a **nice tree decomposition**, and every  $S \subseteq B_t$ , we define:

$$c[t, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t \wedge \hat{S} \cap B_t = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[t, S]$  for **independent**  $S$ :

- ▶ leaf node  $t$ :  $c[t, \emptyset] = 0$
- ▶ introduction node  $t$  of vertex  $v$  with child  $t'$ :

$$c[t, S] = \begin{cases} c[t', S] & \text{if } v \notin S \\ c[t', S \setminus \{v\}] + w(v) & \text{otherwise} \end{cases}$$

- ▶ forget node  $t$  of vertex  $v$  with child  $t'$ :

$$c[t, S] = \max\{c[t', S], c[t', S \cup \{v\}]\}$$

# Dynamical programming using tree-width (example)

For every node  $t$  of a **nice tree decomposition**, and every  $S \subseteq B_t$ , we define:

$$c[t, S] := \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t \wedge \hat{S} \cap B_t = S & \\ \text{if } S \text{ is independent.} \end{cases}$$

**Recursive equations** for computing  $c[t, S]$  for **independent**  $S$ :

- ▶ leaf node  $t$ :  $c[t, \emptyset] = 0$
- ▶ introduction node  $t$  of vertex  $v$  with child  $t'$ :

$$c[t, S] = \begin{cases} c[t', S] & \text{if } v \notin S \\ c[t', S \setminus \{v\}] + w(v) & \text{otherwise} \end{cases}$$

- ▶ forget node  $t$  of vertex  $v$  with child  $t'$ :

$$c[t, S] = \max\{c[t', S], c[t', S \cup \{v\}]\}$$

- ▶ join node  $t$  with children  $t_1$  and  $t_2$ :

$$c[t, S] = c[t_1, S] + c[t_2, S] - w(S)$$

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set**  $S \subseteq V$  can be computed as:

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set**  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set**  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $t \in T$ :

$$\blacktriangleright |B_t| \leq k + 1,$$

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set**  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $t \in T$ :

- ▶  $|B_t| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[t, S]$  at index  $t$ :  $2^{|B_t|} = 2^{k+1}$ ,

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  **$k$** . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set  $S \subseteq V$**  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $t \in T$ :

- ▶  $|B_t| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[t, S]$  at index  $t$ :  $2^{|B_t|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  **adjacency/independence check** for  $S \subseteq B_t$  possible in:  $O(k^2)$  using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,



# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set**  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $t \in T$ :

- ▶  $|B_t| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[t, S]$  at index  $t$ :  $2^{|B_t|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  **adjacency/independence check** for  $S \subseteq B_t$  possible in:  $O(k^2)$  using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,
- ▶ time for comp. a value at  $t$ , using map of values at  $t - 1$ :  $O(k)$

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  **$k$** . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set  $S \subseteq V$**  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $t \in T$ :

- ▶  $|B_t| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[t, S]$  at index  $t$ :  $2^{|B_t|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  **adjacency/independence check** for  $S \subseteq B_t$  possible in:  $O(k^2)$  using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,
- ▶ time for comp. a value at  $t$ , using map of values at  $t - 1$ :  $O(k)$
- ▶ time for comp. all values at  $t$ , using values at  $t - 1$ :  $2^{k+1} \cdot O(k^2)$

# Dyn. programming using tree-width (Weigh. Ind. Set)

Let  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  be a **nice tree decomposition** of  $\mathcal{G} = \langle V, E \rangle$  of width  $k$ . For every  $t \in T$ , and every **independent**  $S \subseteq B_t$ :

$$c[t, S] := \begin{cases} \text{maximum possible weight of a set } \hat{S} \text{ such that} \\ \hat{S} \text{ is independent} \wedge S \subseteq \hat{S} \subseteq V_t = \bigcup_{s \in T_t} B_s \wedge \hat{S} \cap B_i = S \end{cases}$$

**Time Complexity:** Based on the values of  $c[t, S]$ , the **maximum possible weight of an independent set**  $S \subseteq V$  can be computed as:

$$= \max_{S \subseteq B_r} c[r, S] = c[r, \emptyset]$$

Then for all  $t \in T$ :

- ▶  $|B_t| \leq k + 1$ ,
- ▶  $\Rightarrow$  number of values  $c[t, S]$  at index  $t$ :  $2^{|B_t|} = 2^{k+1}$ ,
- ▶  $\Rightarrow$  **adjacency/independence check** for  $S \subseteq B_t$  possible in:  $O(k^2)$  using a datastructure computable in time  $O(k^{O(1)} \cdot n)$ ,
- ▶ time for comp. a value at  $t$ , using map of values at  $t - 1$ :  $O(k)$
- ▶ time for comp. all values at  $t$ , using values at  $t - 1$ :  $2^{k+1} \cdot O(k^2)$

$\Rightarrow$  the time for computing all values at the root  $r$ :

$$(2^{k+1} \cdot O(k^2)) \cdot |T| + O(k^{O(1)} \cdot n) \in O(2^k \cdot k^{O(1)} \cdot n), \text{ since } |T| \in O(k \cdot n).$$

# Dynamical programming with tree width (example)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *tree-width*  $tw(\mathcal{G}) = k$ ,  
 $p^*$ -WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

# Dynamical programming with tree width (example)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *tree-width*  $tw(\mathcal{G}) = k$ ,  
 $p^*$ -WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

$S$  is a *minimal* vertex cover

$\iff V \setminus S$  is a *maximal* independent set.

# Dynamical programming with tree width (example)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *tree-width*  $tw(\mathcal{G}) = k$ ,  
 $p^*$ -WEIGHTED-INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

$S$  is a *minimal* vertex cover

$\iff V \setminus S$  is a *maximal* independent set.

## Corollary

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and *tree-width*  $tw(\mathcal{G}) = k$ ,  
 $p^*$ -VERTEX-COVER  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ .

# Dyn. programming with tree-width: general strategy

We consider problem  $P$  for graphs  $\mathcal{G} = \langle V, E \rangle$  of size  $n$  and nice tree decompositions  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  of tree width  $k$ .

# Dyn. programming with tree-width: general strategy

We consider problem  $P$  for graphs  $\mathcal{G} = \langle V, E \rangle$  of size  $n$  and nice tree decompositions  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  of tree width  $k$ .

- ▶ **Formulate** a family of properties that can be restricted to subtrees of  $\mathcal{T}$  such that
  - ▶ a solution of  $P$  can be obtained from the properties at the root of  $\mathcal{T}$ .
- ▶ **Find** recursion equations for bottom-up evaluation on  $\mathcal{T}$ .



# Dyn. programming with tree-width: general strategy

We consider problem  $P$  for graphs  $\mathcal{G} = \langle V, E \rangle$  of size  $n$  and nice tree decompositions  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  of tree width  $k$ .

- ▶ **Formulate** a family of properties that can be restricted to subtrees of  $\mathcal{T}$  such that
  - ▶ a solution of  $P$  can be obtained from the properties at the root of  $\mathcal{T}$ .
- ▶ **Find** recursion equations for bottom-up evaluation on  $\mathcal{T}$ .
- ▶ **Prove** correctness of these recursion equations by showing two inequalities for each type of node:
  - ▶ one relating an optimum solution for the node to some solutions for its children,
  - ▶ one relating optimum solutions for a node's children to a solution for the node.

# Dyn. programming with tree-width: general strategy

We consider problem  $P$  for graphs  $\mathcal{G} = \langle V, E \rangle$  of size  $n$  and nice tree decompositions  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  of tree width  $k$ .

- ▶ **Formulate** a family of properties that can be restricted to subtrees of  $\mathcal{T}$  such that
  - ▶ a solution of  $P$  can be obtained from the properties at the root of  $\mathcal{T}$ .
- ▶ **Find** recursion equations for bottom-up evaluation on  $\mathcal{T}$ .
- ▶ **Prove** correctness of these recursion equations by showing two inequalities for each type of node:
  - ▶ one relating an optimum solution for the node to some solutions for its children,
  - ▶ one relating optimum solutions for a node's children to a solution for the node.
- ▶ **Obtain** an estimate of the time needed to compute the properties in a node  $t$  depending on  $n$  and  $k$ .

# Dyn. programming with tree-width: general strategy

We consider problem  $P$  for graphs  $\mathcal{G} = \langle V, E \rangle$  of size  $n$  and nice tree decompositions  $\langle \mathcal{T} = \langle T, F, r \rangle, \{B_t\}_{t \in T} \rangle$  of tree width  $k$ .

- ▶ **Formulate** a family of properties that can be restricted to subtrees of  $\mathcal{T}$  such that
  - ▶ a solution of  $P$  can be obtained from the properties at the root of  $\mathcal{T}$ .
- ▶ **Find** recursion equations for bottom-up evaluation on  $\mathcal{T}$ .
- ▶ **Prove** correctness of these recursion equations by showing two inequalities for each type of node:
  - ▶ one relating an optimum solution for the node to some solutions for its children,
  - ▶ one relating optimum solutions for a node's children to a solution for the node.
- ▶ **Obtain** an estimate of the time needed to compute the properties in a node  $t$  depending on  $n$  and  $k$ .
- ▶ **Sum up** the time needed to compute the solution(s) at root  $r$  of  $\mathcal{T}$ .
- ▶ **Add** time needed to obtain the solution of  $P$  from properties at  $r$ .

# Dynamical programming: similar results (I)

## Theorem

For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and  $tw(\mathcal{G}) = k$ ,

- ▶  $p^*$ -VERTEX-COVER, INDEPENDENT-SET  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ ,
- ▶  $p^*$ -DOMINATING-SET  $\in \text{DTIME}(4^k \cdot k^{O(1)} \cdot n)$ ,
- ▶  $p^*$ -ODD CYCLE TRAVERSAL  $\in \text{DTIME}(3^k \cdot k^{O(1)} \cdot n)$ ,
- ▶  $p^*$ -MAXCUT  $\in \text{DTIME}(2^k \cdot k^{O(1)} \cdot n)$ ,
- ▶  $p^*$ - $q$ -COLORABILITY  $\in \text{DTIME}(q^k \cdot k^{O(1)} \cdot n)$ .

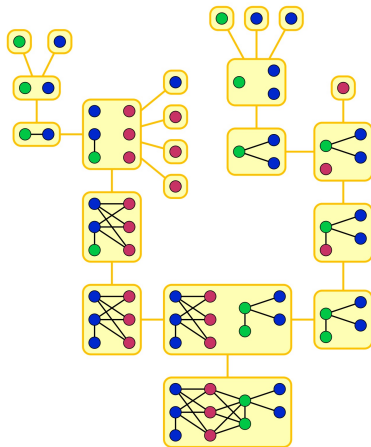
# Dynamical programming: similar results (II)

## Theorem

*For every graph  $\mathcal{G} = \langle V, E \rangle$  with  $|V| = n$  and  $tw(\mathcal{G}) = k$ , the following problems are in  $\text{DTIME}(k^{O(k)} \cdot n)$ :*

- ▶  $p^*$ -STEINER-TREE,
- ▶  $p^*$ -FEEDBACK-VERTEX-SET,
- ▶  $p^*$ -HAMILTONIAN-PATH *and*  $p^*$ -LONGEST-PATH,
- ▶  $p^*$ -HAMILTONIAN-CYCLE *and*  $p^*$ -LONGEST-CYCLE,
- ▶  $p^*$ -CHROMATIC-NUMBER,
- ▶  $p^*$ -CYCLE-PACKING,
- ▶  $p^*$ -CONNECTED-VERTEX-COVER,
- ▶  $p^*$ -CONNECTED-FEEDBACK-VERTEX-SET.

# Clique width (example)



# Clique-Width

For  $k \in \mathbb{N}$ , the  *$k$ -expressions* are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i \mid \text{edge}_{i-j}(\varphi) \mid \text{recolor}_{i \rightarrow j}(\varphi) \mid (\varphi_1 \oplus \varphi_2)$$

for  $i, j \in [k]$  with  $i \neq j$ .

# Clique-Width

For  $k \in \mathbb{N}$ , the  $k$ -expressions are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i \mid \text{edge}_{i-j}(\varphi) \mid \text{recolor}_{i \rightarrow j}(\varphi) \mid (\varphi_1 \oplus \varphi_2)$$

for  $i, j \in [k]$  with  $i \neq j$ .  $k$ -expressions  $\varphi$  generate graphs  $\mathcal{G}(\varphi)$ :



# Clique-Width

For  $k \in \mathbb{N}$ , the  $k$ -expressions are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i$$

for  $i, j \in [k]$  with  $i \neq j$ .  $k$ -expressions  $\varphi$  generate graphs  $\mathcal{G}(\varphi)$ :

- ▷  $\mathcal{G}(i)$  is the graph with a single vertex of color  $i$ .

# Clique-Width

For  $k \in \mathbb{N}$ , the  $k$ -expressions are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i \mid \text{edge}_{i-j}(\varphi)$$

for  $i, j \in [k]$  with  $i \neq j$ .  $k$ -expressions  $\varphi$  generate graphs  $\mathcal{G}(\varphi)$ :

- ▷  $\mathcal{G}(i)$  is the graph with a single vertex of color  $i$ .
- ▷  $\mathcal{G}(\text{edge}_{i-j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by adding edges between every vertex of color  $i$  and every vertex of color  $j$ .

# Clique-Width

For  $k \in \mathbb{N}$ , the  $k$ -expressions are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i \mid \text{edge}_{i-j}(\varphi) \mid \text{recolor}_{i \rightarrow j}(\varphi)$$

for  $i, j \in [k]$  with  $i \neq j$ .  $k$ -expressions  $\varphi$  generate graphs  $\mathcal{G}(\varphi)$ :

- ▷  $\mathcal{G}(i)$  is the graph with a single vertex of color  $i$ .
- ▷  $\mathcal{G}(\text{edge}_{i-j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by adding edges between every vertex of color  $i$  and every vertex of color  $j$ .
- ▷  $\mathcal{G}(\text{recolor}_{i \rightarrow j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by recoloring every vertex of color  $i$  by color  $j$ .

# Clique-Width

For  $k \in \mathbb{N}$ , the  $k$ -expressions are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i \mid \text{edge}_{i-j}(\varphi) \mid \text{recolor}_{i \rightarrow j}(\varphi) \mid (\varphi_1 \oplus \varphi_2)$$

for  $i, j \in [k]$  with  $i \neq j$ .  $k$ -expressions  $\varphi$  generate graphs  $\mathcal{G}(\varphi)$ :

- ▷  $\mathcal{G}(i)$  is the graph with a single vertex of color  $i$ .
- ▷  $\mathcal{G}(\text{edge}_{i-j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by adding edges between every vertex of color  $i$  and every vertex of color  $j$ .
- ▷  $\mathcal{G}(\text{recolor}_{i \rightarrow j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by recoloring every vertex of color  $i$  by color  $j$ .
- ▷  $\mathcal{G}(\varphi_1 \oplus \varphi_2)$  is the disjoint union of  $\mathcal{G}(\varphi_1)$  and  $\mathcal{G}(\varphi_2)$ .

# Clique-Width

For  $k \in \mathbb{N}$ , the  $k$ -expressions are defined by:

$$\varphi, \varphi_1, \varphi_2 ::= i \mid \text{edge}_{i-j}(\varphi) \mid \text{recolor}_{i \rightarrow j}(\varphi) \mid (\varphi_1 \oplus \varphi_2)$$

for  $i, j \in [k]$  with  $i \neq j$ .  $k$ -expressions  $\varphi$  generate graphs  $\mathcal{G}(\varphi)$ :

- ▷  $\mathcal{G}(i)$  is the graph with a single vertex of color  $i$ .
- ▷  $\mathcal{G}(\text{edge}_{i-j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by adding edges between every vertex of color  $i$  and every vertex of color  $j$ .
- ▷  $\mathcal{G}(\text{recolor}_{i \rightarrow j}(\varphi))$  results from  $\mathcal{G}(\varphi)$  by recoloring every vertex of color  $i$  by color  $j$ .
- ▷  $\mathcal{G}(\varphi_1 \oplus \varphi_2)$  is the disjoint union of  $\mathcal{G}(\varphi_1)$  and  $\mathcal{G}(\varphi_2)$ .

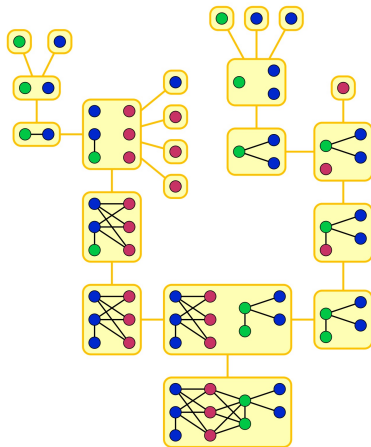
**Definition** (Courcelle, Engelfriet, Rozenberg, 1993, [2])

The **clique-width**  $\text{clw}(\mathcal{G})$  of  $\mathcal{G} = \langle V, E \rangle$  is defined by:

$$\text{clw}(\mathcal{G}) := \text{the least } k \in \mathbb{N} \text{ such that, for some } k\text{-expression } \varphi, \\ \mathcal{G} = \mathcal{G}(\varphi) \text{ (when removing colors)}$$

# Clique width (example)

Building a graph  $\mathcal{G}$  of clique-width  $c/w(\mathcal{G}) = 3$ :



# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.

# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
- ▶ The class of stars has clique-width 2.



# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
- ▶ The class of stars has clique-width 2.
- ▶ The class of trees has clique-width 3.

# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
- ▶ The class of stars has clique-width 2.
- ▶ The class of trees has clique-width 3.
- ▶ The class of  $n \times n$  grids has clique-width  $\Theta(n)$ .

# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
  - ▶ The class of stars has clique-width 2.
  - ▶ The class of trees has clique-width 3.
  - ▶ The class of  $n \times n$  grids has clique-width  $\Theta(n)$ .
- 
- ▶ subgraphs/induced subgraphs:
    - ▶ clique-width is preserved under taking induced subgraphs,
    - ▶ clique-width is **not preserved** under taking subgraphs (e.g. minors).

# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
  - ▶ The class of stars has clique-width 2.
  - ▶ The class of trees has clique-width 3.
  - ▶ The class of  $n \times n$  grids has clique-width  $\Theta(n)$ .
- 
- ▶ subgraphs/induced subgraphs:
    - ▶ clique-width is preserved under taking induced subgraphs,
    - ▶ clique-width is **not preserved** under taking subgraphs (e.g. minors).
  - ▶  $clw < tw$ :
    - ▶  $clw \leq tw$ :  $clw(\mathcal{G}) \leq 3 \cdot 2^{tw(\mathcal{G})-1}$
    - ▶  $\neg(tw \leq clw)$ : for example,  $clw(K_n) = 2$ , and  $tw(K_n) = n - 1$ .

# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
  - ▶ The class of stars has clique-width 2.
  - ▶ The class of trees has clique-width 3.
  - ▶ The class of  $n \times n$  grids has clique-width  $\Theta(n)$ .
- 
- ▶ subgraphs/induced subgraphs:
    - ▶ clique-width is preserved under taking induced subgraphs,
    - ▶ clique-width is **not preserved** under taking subgraphs (e.g. minors).
  - ▶  $c/w < tw$ :
    - ▶  $c/w \leq tw$ :  $c/w(\mathcal{G}) \leq 3 \cdot 2^{tw(\mathcal{G})-1}$
    - ▶  $\neg(tw \leq c/w)$ : for example,  $c/w(K_n) = 2$ , and  $tw(K_n) = n - 1$ .
  - ▶ Deciding whether  $c/w(\mathcal{G}) \leq k$  is **NP-hard**. With parameter  $k$  it is in XP (slice-wise polynomial), but unknown to be in FPT.

# Clique-Width (examples, properties, computability)

## Example

- ▶ The class of cliques has clique-width 2.
  - ▶ The class of stars has clique-width 2.
  - ▶ The class of trees has clique-width 3.
  - ▶ The class of  $n \times n$  grids has clique-width  $\Theta(n)$ .
- 
- ▶ subgraphs/induced subgraphs:
    - ▶ clique-width is preserved under taking induced subgraphs,
    - ▶ clique-width is **not preserved** under taking subgraphs (e.g. minors).
  - ▶  $c/w < tw$ :
    - ▶  $c/w \leq tw$ :  $c/w(\mathcal{G}) \leq 3 \cdot 2^{tw(\mathcal{G})-1}$
    - ▶  $\neg(tw \leq c/w)$ : for example,  $c/w(K_n) = 2$ , and  $tw(K_n) = n - 1$ .
  - ▶ Deciding whether  $c/w(\mathcal{G}) \leq k$  is **NP-hard**. With parameter  $k$  it is in XP (slice-wise polynomial), but unknown to be in FPT.
  - ▶ Every graph property expressible in **MSO (monadic second-order logic)** can be decided in linear time w.r.t. the graph's clique-width.

# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$$f \text{ is } \textit{symmetric} : \iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right];$$

# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$$f \text{ is } \textit{symmetric} : \iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right];$$

$$f \text{ is } \textit{fair} : \iff f(\emptyset) = f(U) = 0.$$

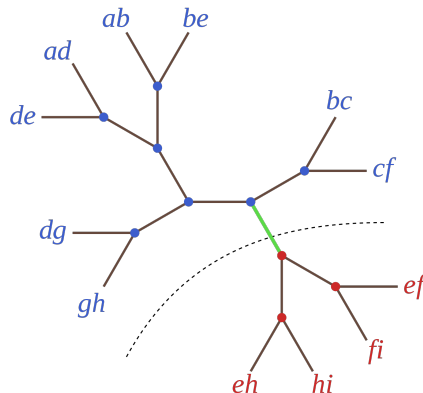


# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$f$  is *symmetric*:  $\iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right]$ ;

$f$  is *fair*:  $\iff f(\emptyset) = f(U) = 0$ .



# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$$f \text{ is } \textit{symmetric} : \iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right];$$

$$f \text{ is } \textit{fair} : \iff f(\emptyset) = f(U) = 0.$$

## Definition

Let  $U$  be a set,  $f : 2^U \rightarrow \mathbb{R}_0^+$  a cut function.

A *branch decomposition* of  $U$  is a pair  $\langle \mathcal{T}, \eta \rangle$  where:

- ▷  $\mathcal{T} = \langle T, F \rangle$  a tree.
- ▷  $\eta : U \rightarrow \text{Leafs}(\mathcal{T})$  a bijective function.

# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$$f \text{ is symmetric} : \iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right];$$

$$f \text{ is fair} : \iff f(\emptyset) = f(U) = 0.$$

## Definition

Let  $U$  be a set,  $f : 2^U \rightarrow \mathbb{R}_0^+$  a cut function.

A *branch decomposition* of  $U$  is a pair  $\langle \mathcal{T}, \eta \rangle$  where:

- ▷  $\mathcal{T} = \langle T, F \rangle$  a tree.
- ▷  $\eta : U \rightarrow \text{Leaves}(\mathcal{T})$  a bijective function.

Every edge  $e \in T$  splits the tree into two connected parts, and, via  $\eta$ , splits  $U$  into a partition  $\langle X_e, Y_e \rangle$ .

# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$$f \text{ is symmetric: } \iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right];$$

$$f \text{ is fair: } \iff f(\emptyset) = f(U) = 0.$$

## Definition

Let  $U$  be a set,  $f : 2^U \rightarrow \mathbb{R}_0^+$  a cut function.

A *branch decomposition* of  $U$  is a pair  $\langle \mathcal{T}, \eta \rangle$  where:

- ▷  $\mathcal{T} = \langle T, F \rangle$  a tree.
- ▷  $\eta : U \rightarrow \text{Leafs}(\mathcal{T})$  a bijective function.

Every edge  $e \in T$  splits the tree into two connected parts, and, via  $\eta$ , splits  $U$  into a partition  $\langle X_e, Y_e \rangle$ .

The *width* of an edge  $e \in T$  (with respect to  $f$ ) is  $f(X_e) = f(Y_e)$ . The *width of  $\langle \mathcal{T}, \eta \rangle$  w.r.t.  $f$*  is the maximum width over the edges of  $\mathcal{T}$ .

# $f$ -Width (of sets)

By a *cut function* or a *connectivity function* we mean a function  $f : 2^U \rightarrow \mathbb{R}_0^+$  such that:

$$f \text{ is symmetric: } \iff \forall X \subseteq U \left[ f(X) = f(U \setminus X) \right];$$

$$f \text{ is fair: } \iff f(\emptyset) = f(U) = 0.$$

## Definition

Let  $U$  be a set,  $f : 2^U \rightarrow \mathbb{R}_0^+$  a cut function.

A *branch decomposition* of  $U$  is a pair  $\langle \mathcal{T}, \eta \rangle$  where:

- ▷  $\mathcal{T} = \langle T, F \rangle$  a tree.
- ▷  $\eta : U \rightarrow \text{Leafs}(\mathcal{T})$  a bijective function.

Every edge  $e \in T$  splits the tree into two connected parts, and, via  $\eta$ , splits  $U$  into a partition  $\langle X_e, Y_e \rangle$ .

The *width* of an edge  $e \in T$  (with respect to  $f$ ) is  $f(X_e) = f(Y_e)$ . The *width of  $\langle \mathcal{T}, \eta \rangle$  w.r.t.  $f$*  is the maximum width over the edges of  $\mathcal{T}$ .

The  *$f$ -width  $w_f(U)$*  of  $U$  is defined as:

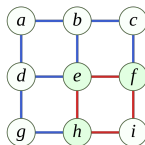
$$w_f(U) := \text{minimum width of branch decomp's of } U \text{ w.r.t. } f.$$

# Branch-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph. The *border (vertices)* of a set  $X \subseteq E$  of edges is defined by:

$$\partial(X) := \left\{ v \in V \mid \exists e_1 \in X \exists e_2 \in E \setminus X \right. \\ \left. [v \text{ is incident to } e_1 \text{ and } e_2] \right\}$$



# Branch-Width

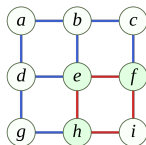
## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph. The *border (vertices)* of a set  $X \subseteq E$  of edges is defined by:

$$\partial(X) := \left\{ v \in V \mid \exists e_1 \in X \exists e_2 \in E \setminus X \right. \\ \left. [v \text{ is incident to } e_1 \text{ and } e_2] \right\}$$

The *branch-width*  $bw(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is defined as

$$bw(\mathcal{G}) := w_f(E) \quad \text{for } f: 2^E \rightarrow \mathbb{R}_0^+, X \mapsto |\partial(X)|$$



# Branch-Width

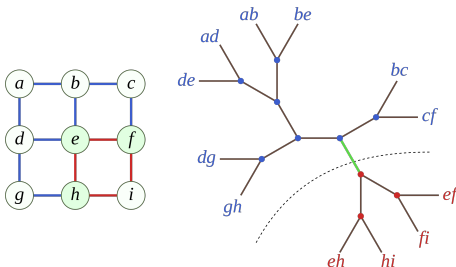
## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph. The *border (vertices)* of a set  $X \subseteq E$  of edges is defined by:

$$\partial(X) := \left\{ v \in V \mid \exists e_1 \in X \exists e_2 \in E \setminus X \right. \\ \left. [v \text{ is incident to } e_1 \text{ and } e_2] \right\}$$

The *branch-width*  $bw(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is defined as

$$bw(\mathcal{G}) := w_f(E) \quad \text{for } f: 2^E \rightarrow \mathbb{R}_0^+, X \mapsto |\partial(X)|$$





# Branch-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph. The *border (vertices)* of a set  $X \subseteq E$  of edges is defined by:

$$\partial(X) := \left\{ v \in V \mid \exists e_1 \in X \exists e_2 \in E \setminus X \right. \\ \left. \left[ v \text{ is incident to } e_1 \text{ and } e_2 \right] \right\}$$

The *branch-width*  $bw(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is defined as

$$bw(\mathcal{G}) := w_f(E) \quad \text{for } f : 2^E \rightarrow \mathbb{R}_0^+, X \mapsto |\partial(X)|$$

## Proposition

$bw(\mathcal{G}) \approx tw(\mathcal{G})$ , for every graph; more precisely:

$$bw(\mathcal{G}) \leq tw(\mathcal{G}) + 1 \leq \frac{3}{2} \cdot bw(\mathcal{G}).$$

# Rank-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph.

For  $X \subseteq V$  we define the  $GF(2)$ -matrix:

$$B_{\mathcal{G}}(X) := (b_{x,y})_{x \in X, y \in V \setminus X}, \text{ where, for all } x \in X, y \in V \setminus X: \\ b_{x,y} = 1 \iff \{x, y\} \in E.$$

( $B_{\mathcal{G}}(X)$  is the adjacency matrix of the bipartite graph induced by  $\mathcal{G}$  between  $X$  and  $V \setminus X$ .)

# Rank-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph.

For  $X \subseteq V$  we define the  $GF(2)$ -matrix:

$$B_{\mathcal{G}}(X) := (b_{x,y})_{x \in X, y \in V \setminus X}, \text{ where, for all } x \in X, y \in V \setminus X: \\ b_{x,y} = 1 \iff \{x, y\} \in E.$$

( $B_{\mathcal{G}}(X)$  is the adjacency matrix of the bipartite graph induced by  $\mathcal{G}$  between  $X$  and  $V \setminus X$ .)

The *rank-width*  $rw(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is:

$$rw(\mathcal{G}) := w_{\rho_{\mathcal{G}}}(E) \quad \text{for} \quad \rho_{\mathcal{G}} : 2^V \rightarrow \mathbb{N}_0, X \mapsto \text{rank of } B_{\mathcal{G}}(X)$$

# Rank-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph.

For  $X \subseteq V$  we define the  $GF(2)$ -matrix:

$$B_{\mathcal{G}}(X) := (b_{x,y})_{x \in X, y \in V \setminus X}, \text{ where, for all } x \in X, y \in V \setminus X:$$

$$b_{x,y} = 1 \iff \{x, y\} \in E.$$

( $B_{\mathcal{G}}(X)$  is the adjacency matrix of the bipartite graph induced by  $\mathcal{G}$  between  $X$  and  $V \setminus X$ .)

The *rank-width*  $rw(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is:

$$rw(\mathcal{G}) := w_{\rho_{\mathcal{G}}}(E) \quad \text{for} \quad \rho_{\mathcal{G}} : 2^V \rightarrow \mathbb{N}_0, X \mapsto \text{rank of } B_{\mathcal{G}}(X)$$

## Properties

- ▶  $rw(\mathcal{G}) \leq tw(\mathcal{G})$ .
- ▶ tree-width cannot be bounded functionally by rank-width:  
 $rw(K_n) = 1$ , but  $tw(K_n) = n - 1$ .

# Carving-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph.

For  $X \subseteq V$  the *edge-cut* of  $X$  is:

$$\text{cut}_{\mathcal{G}}(X) := \{e = \{u, v\} \in E \mid u \in X, v \in V \setminus X\} .$$

The *carving-width*  $\text{carw}(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is:

$$\text{carw}(\mathcal{G}) := w_{\text{cut}}(E) \quad \text{for} \quad \text{cut} : 2^V \rightarrow \mathbb{N}_0, X \mapsto |\text{cut}_{\mathcal{G}}(X)| .$$

# Carving-Width and Cut-Width

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph.

For  $X \subseteq V$  the **edge-cut** of  $X$  is:

$$\text{cut}_{\mathcal{G}}(X) := \{e = \{u, v\} \in E \mid u \in X, v \in V \setminus X\} .$$

The **carving-width**  $\text{carw}(\mathcal{G})$  of a graph  $\mathcal{G} = \langle G, E \rangle$  is:

$$\text{carw}(\mathcal{G}) := w_{\text{cut}}(E) \quad \text{for} \quad \text{cut} : 2^V \rightarrow \mathbb{N}_0, X \mapsto |\text{cut}_{\mathcal{G}}(X)| .$$

## Definition

Let  $\mathcal{G} = \langle V, E \rangle$  be a graph with  $n = |V|$ .

For a permutation  $\pi : \{1, \dots, n\} \rightarrow V$  on  $V$  we define:

$$\text{width}(\pi) := \max_{1 \leq i \leq n} \text{cut}_{\mathcal{G}}(\{\pi(j) \mid 1 \leq j \leq i\}) .$$

The **cut-width**  $\text{cutw}(\mathcal{G})$  of  $\mathcal{G}$  is:

$$\text{cutw}(\mathcal{G}) := \min_{\pi \text{ perm. of } V} \text{width}(\pi) .$$

# Coverage in Multi-Interface Networks



# Coverage in Multi-Interface Networks



$CMI(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.



# Coverage in Multi-Interface Networks



$CMI(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Solution:** An allocation  $W_A : V \rightarrow 2^{\{1, \dots, a\}}$  of active interfaces **covering**  $\mathcal{G}$  such that  $W_A(v) \subseteq W(v)$ , and  $|W_A(v)| \leq p$  for all  $v \in V$ , if possible; otherwise, a negative answer.

# Coverage in Multi-Interface Networks



$CMI(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Solution:** An allocation  $W_A : V \rightarrow 2^{\{1, \dots, a\}}$  of active interfaces **covering**  $\mathcal{G}$  such that  $W_A(v) \subseteq W(v)$ , and  $|W_A(v)| \leq p$  for all  $v \in V$ , if possible; otherwise, a negative answer.

**Problem:** Obtain, if possible, a **minimal solution** with respect to the total cost of the interfaces that are activated, that is,  $c(W_A) = \sum_{v \in V} \sum_{i \in W_A(v)} c(i)$ .

# Coverage in Multi-Interface Networks

## Theorem

$CMI(2) \in \text{NP-complete}$ , also for graphs with max. node degree  $\geq 4$ .

# Coverage in Multi-Interface Networks (parameterized)

## Theorem

$CMI(2) \in \text{NP-complete}$ , also for graphs with max. node degree  $\geq 4$ .

$p^*$ -CMI( $p$ ) (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Parameter:** path-width / carving-width  $k$

**Problem:** Obtain, if possible, a minimal solution with respect to the total cost of the interfaces that are activated, that is,  

$$c(W_A) = \sum_{v \in V} \sum_{i \in W_A(v)} c(i).$$

# Coverage in Multi-Interface Networks (parameterized)

## Theorem

$CMI(2) \in \text{NP-complete}$ , also for graphs with max. node degree  $\geq 4$ .

$p^*\text{-CMI}(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Parameter:** path-width / carving-width  $k$

**Problem:** Obtain, if possible, a minimal solution with respect to the total cost of the interfaces that are activated, that is,  

$$c(W_A) = \sum_{v \in V} \sum_{i \in W_A(v)} c(i).$$

## Theorem (Aloisio, Navarra, 2020, [1])

- For path-width  $pw(\mathcal{G}) = k$ ,  
 $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot (a + \binom{a}{2})^{k+1}).$

# Coverage in Multi-Interface Networks (parameterized)

## Theorem

$CMI(2) \in \text{NP-complete}$ , also for graphs with max. node degree  $\geq 4$ .

$p^*\text{-CMI}(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Parameter:** path-width / carving-width  $k$

**Problem:** Obtain, if possible, a minimal solution with respect to the total cost of the interfaces that are activated, that is,  

$$c(W_A) = \sum_{v \in V} \sum_{i \in W_A(v)} c(i).$$

## Theorem (Aloisio, Navarra, 2020, [1])

- ▶ For path-width  $pw(\mathcal{G}) = k$ ,  
 $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot (a + \binom{a}{2})^{k+1}).$
- ▶ For carving-width  $carw(\mathcal{G}) = k$ ,  $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot a^{4k}).$

# Coverage in Multi-Interface Networks (parameterized)

Theorem (Aloisio, Navarra, 2020, [1])

- ▶ For *path-width*  $pw(\mathcal{G}) = k$ ,  
 $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot (a + \binom{a}{2})^{k+1})$ .
- ▶ For *carving-width*  $carw(\mathcal{G}) = k$ ,  $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot a^{4k})$ .

$(p^*)'\text{-CMI}(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Parameter:**  $a + (\text{path-width} / \text{carving-width } k)$

**Problem:** Obtain, if possible, a minimal solution with respect to the total cost of the interfaces that are activated, that is,  
 $c(W_A) = \sum_{v \in V} \sum_{i \in W_A(v)} c(i)$ .

# Coverage in Multi-Interface Networks (parameterized)

Theorem (Aloisio, Navarra, 2020, [1])

- ▶ For *path-width*  $pw(\mathcal{G}) = k$ ,  
 $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot (a + \binom{a}{2})^{k+1})$ .
- ▶ For *carving-width*  $carw(\mathcal{G}) = k$ ,  $p^*\text{-CMI}(2) \in \text{DTIME}(n \cdot a^{4k})$ .

$(p^*)'\text{-CMI}(p)$  (for  $p \in \mathbb{N}$ )

**Instance:** A graph  $\mathcal{G} = \langle V, E \rangle$ ,  $W : V \rightarrow 2^{\{1, \dots, a\}}$  available-interface allocation,  $c : \{1, \dots, a\} \rightarrow \mathbb{R}^+$  interface cost function.

**Parameter:**  $a + (\text{path-width} / \text{carving-width } k)$

**Problem:** Obtain, if possible, a minimal solution with respect to the total cost of the interfaces that are activated, that is,  
 $c(W_A) = \sum_{v \in V} \sum_{i \in W_A(v)} c(i)$ .

Corollary

$(p^*)'\text{-CMI}(p) \in \text{FPT}$ .



# Comparing parameterizations

## Definition (computably bounded)

Let  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  parameterizations.

- ▶  $\kappa_1 \succeq \kappa_2 : \iff \exists g : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \forall x \in \Sigma^* [g(\kappa_1(x)) \geq \kappa_2(x)]$ .
- ▶  $\kappa_1 \approx \kappa_2 : \iff \kappa_1 \succeq \kappa_2 \wedge \kappa_2 \succeq \kappa_1$ .
- ▶  $\kappa_1 \succ \kappa_2 : \iff \kappa_1 \succeq \kappa_2 \wedge \neg(\kappa_2 \succeq \kappa_1)$ .

# Comparing parameterizations

## Definition (computably bounded)

Let  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  parameterizations.

- ▶  $\kappa_1 \geq \kappa_2 : \iff \exists g : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \forall x \in \Sigma^* [g(\kappa_1(x)) \geq \kappa_2(x)]$ .
- ▶  $\kappa_1 \approx \kappa_2 : \iff \kappa_1 \geq \kappa_2 \wedge \kappa_2 \geq \kappa_1$ .
- ▶  $\kappa_1 > \kappa_2 : \iff \kappa_1 \geq \kappa_2 \wedge \neg(\kappa_2 \geq \kappa_1)$ .

## Proposition

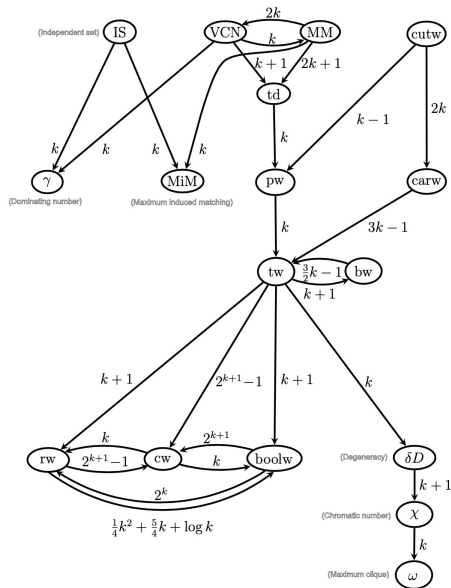
For all parameterized problems  $\langle Q, \kappa_1 \rangle$  and  $\langle Q, \kappa_2 \rangle$  with parameterizations  $\kappa_1, \kappa_2 : \Sigma^* \rightarrow \mathbb{N}$  with  $\kappa_1 \geq \kappa_2$ :

$$\begin{aligned} \langle Q, \kappa_1 \rangle \in \text{FPT} &\iff \langle Q, \kappa_2 \rangle \in \text{FPT} \\ \langle Q, \kappa_1 \rangle \notin \text{FPT} &\implies \langle Q, \kappa_2 \rangle \notin \text{FPT} \end{aligned}$$

# Computably boundedness between notions of width

(from Sasák, [5])

$$wd_1 \succeq wd_2 : \Leftarrow wd_1 \xrightarrow{g(k)} wd_2$$



# Computably boundedness between notions of width

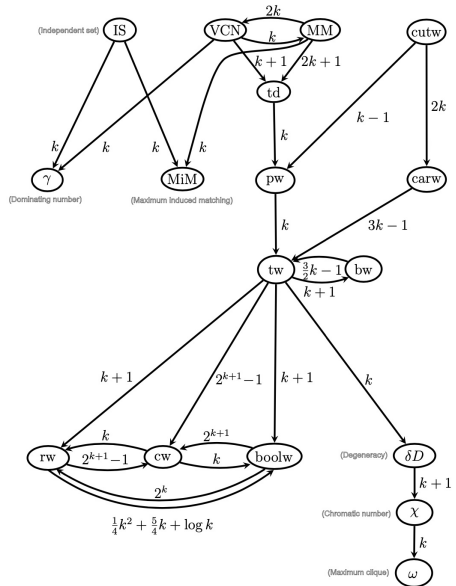
(from Sasák, [5])

$$wd_1 \geq wd_2 : \Leftarrow wd_1 \xrightarrow{g(k)} wd_2$$

► FPT-results

transfer upwards

(and conversely to  $\xrightarrow{g}$ )



# Computably boundedness between notions of width

(from Sasák, [5])

$$wd_1 \geq wd_2 : \Leftarrow wd_1 \xrightarrow{g(k)} wd_2$$

► FPT-results

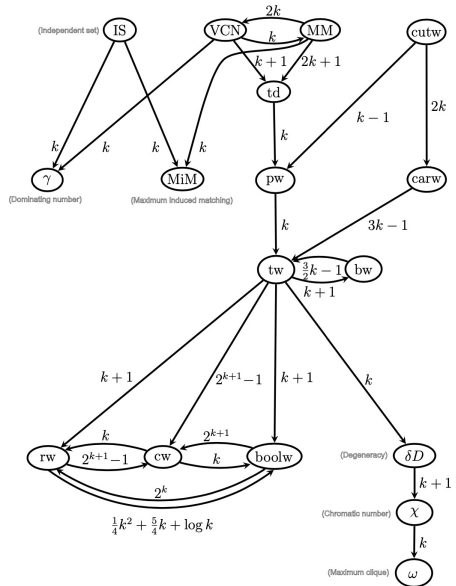
transfer upwards

(and conversely to  $\xrightarrow{g}$ )

► (∉ FPT)-results

transfer downwards

(and along  $\xrightarrow{g}$ )



# Summary

- ▶ comparing parameterizations
- ▶ dynamical programming on trees, example:
  - ▶ WEIGHTED-INDEPENDENT-SET (and VERTEX-COVER)
- ▶ path-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ tree-width
  - ▶ example: fpt-algorithm for bounded path-width
- ▶ fpt-results for other problems, obtained similarly
- ▶ other notions of width
  - ▶ clique-width
  - ▶ using  $f$ -width to define:
    - ▶ carving-width (and cut-width)
    - ▶ branch-width
    - ▶ rank-width
- ▶ example problem: coverage in multi-interface networks
- ▶ comparing width-notions

# Tomorrow

Monday, June 10 10.30 – 12.30	Tuesday, June 11	Wednesday, June 12 10.30 – 12.30	Thursday, June 13	Friday, June 14
<b>Introduction &amp; basic FPT results</b> motivation for FPT kernelization, Crown Lemma, Sunflower Lemma		<b>Algorithmic Meta-Theorems</b> 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width		
	GDA		GDA	GDA
<i>Algorithmic Techniques</i>		<i>Formal-Method &amp; Algorithmic Techniques</i>		
	14.30 – 16.30			14.30 – 16.30
	<b>Notions of bounded graph width</b>			<b>FPT-Intractability Classes &amp; Hierarchies</b>
	path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. widths	GDA	GDA	motivation for FP-intractability results, FPT-reductions, class XP (slicewise polynomial), W- and A-Hierarchies, placing problems on these hierarchies

# Tomorrow

- ▶ recalling notions from logic:
  - ▶ propositional, and first-order logic
  - ▶ monadic second-order logic (MSO)
- ▶ Courcelle's Theorem: obtaining FPT-results by
  - ▶ model-checking of MSO-properties  
on graphs and structures of bounded tree-/clique-width



# References I



Alessandro Aloisio and Alfredo Navarra.

Constrained connectivity in bounded x-width multi-interface networks.

*Algorithms*, 13(2), 2020.



Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg.

Handle-rewriting hypergraph grammars.

*Journal of Computer and System Sciences*, 46(2):218 – 270, 1993.



Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh.

*Parameterized Algorithms*.

Springer, 1st edition, 2015.

# References II



[Jörg Flum and Martin Grohe.](#)  
*Parameterized Complexity Theory.*  
 Springer, 2006.



[Róbert Sásak.](#)  
 Comparing 17 graph parameters.  
 Master's thesis, University of Bergen, Norway, 2010.