Connected tree-width and connected cops and robber game

Christophe Paul

Connected treewidth and connected cops-and-robber game

–

Obstructions and algorithms

Christophe PAUL
(CNRS – Univ. Montpellier, LIRMM, France)

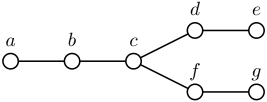
Joint work with **I. Adler** (University of Leeds, UK)
G. Mescoff (ENS Rennes, France)
D. Thilikos (CNRS – Univ. Montpellier, LIRMM, France)

CAALM Workshop, Chennai, January 25, 2019

A node search strategy

A search strategy is defined by a sequence of moves, each of these

- either add a searcher

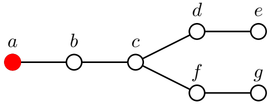


A node search strategy

A search strategy is defined by a sequence of moves, each of these

- either add a searcher

$\langle \{a\}, \dots \rangle$

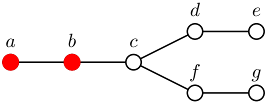


A node search strategy

A search strategy is defined by a sequence of moves, each of these

- either add a searcher

$\langle \{a\}, \{a, b\}, \ldots \rangle$

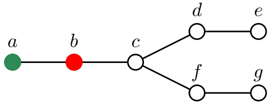


A node search strategy

A search strategy is defined by a sequence of moves, each of these

- either add a searcher
- or remove a searcher

$$\langle \{a\}, \{a, b\}, \{b\}, \ldots \rangle$$



More formally, we define $\mathcal{S} = \langle S_1, \ldots S_r \rangle$ such that

- for all $i \in [r]$, $S_i \subseteq V(G)$; (set of occupied positions)
- $|S_1| = 1$;
- for all $i \in [r-1]$, $|S_i \triangle S_{i-1}| = 1$.

Node search against…

…an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-

Lazy robber Agile robber

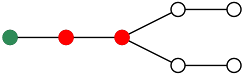
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

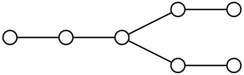
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

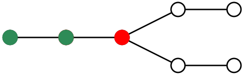
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

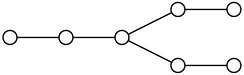
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-

Lazy robber　　　　Agile robber

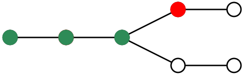
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

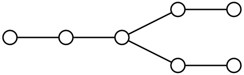
Node search against...

...an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

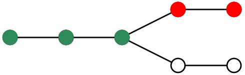
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

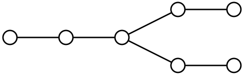
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

Node search against...

...an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-

Lazy robber   Agile robber

We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-

Lazy robber　　　　Agile robber

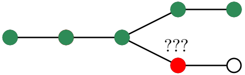
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

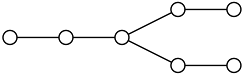
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position



Lazy robber



Agile robber

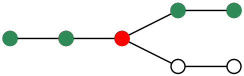
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

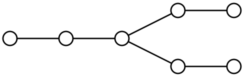
Node search against...

...an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

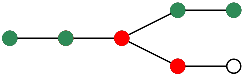
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

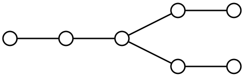
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

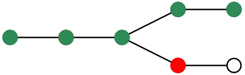
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

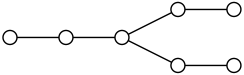
Node search against...

...an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

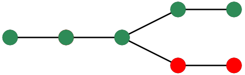
We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

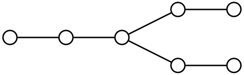
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
-



Lazy robber



Agile robber

We define the set of free locations in the case of a lazy robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i \cap (S_i \setminus S_{i-1})\}$

Node search against…

…an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
- agile : he can move (if possible) at any time

Lazy robber

Agile robber

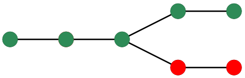
We define the set of free locations in the case of a agile robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i\}$

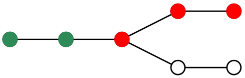
Node search against...

...an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
- agile : he can move (if possible) at any time



Lazy robber



Agile robber

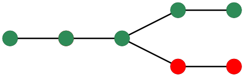
We define the set of free locations in the case of a agile robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i\}$

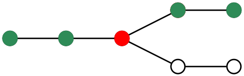
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
- agile : he can move (if possible) at any time



Lazy robber



Agile robber

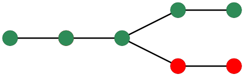
We define the set of free locations in the case of a agile robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i\}$

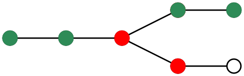
Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
- agile : he can move (if possible) at any time



Lazy robber



Agile robber

We define the set of free locations in the case of a agile robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i\}$

Node search against. . .

. . . an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
- agile : he can move (if possible) at any time

Lazy robber

Agile robber

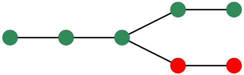
We define the set of free locations in the case of a agile robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i\}$

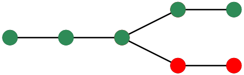
Node search against...

...an invisible robber, that can be

- lazy : he escapes (if possible) if a searcher is landing at his position
- agile : he can move (if possible) at any time



Lazy robber



Agile robber

We define the set of free locations in the case of a agile robber :

- $F_1 = V(G) \setminus S_1$
- for all $i \geqslant 2$, $F_i = (F_{i-1} \setminus S_i) \cup \{v \in cc_{G-S_i}(u) \mid u \in F_i\}$

Properties and cost of a node search strategy

A node search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle$ is

- complete if $F_r = \emptyset$;
- monotone if for every $i \in [r-1]$, $F_{i+1} \subset F_i$.
  (there is no recontamination of a vertex)

Properties and cost of a node search strategy

A node search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle$ is

- complete if $F_r = \emptyset$;
- monotone if for every $i \in [r-1]$, $F_{i+1} \subset F_i$.
  (there is no recontamination of a vertex)

We define

$$\textbf{ans}(G) = \min\{\text{cost}(\mathcal{S}) \mid \mathcal{S} \text{ is a complete strategy against an agile robber}\}$$

$$\textbf{mans}(G) = \min\{\text{cost}(\mathcal{S}) \mid \mathcal{S} \text{ is a complete monotone} \ldots \text{agile robber}\}$$

$$\textbf{lns}(G) = \min\{\text{cost}(\mathcal{S}) \mid \mathcal{S} \text{ is a complete strategy against a lazy robber}\}$$

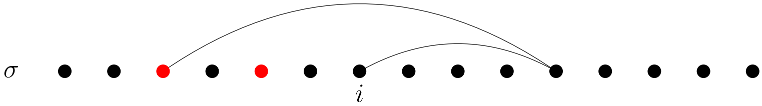
$$\textbf{mlns}(G) = \min\{\text{cost}(\mathcal{S}) \mid \mathcal{S} \text{ is a complete monotone} \ldots \text{lazy robber}\}$$

Known relationship between parameters

Theorem.

- treewidth corresponds to lazy strategies [DKT97]

$$\mathbf{tw}(G) = \mathbf{tvs}(G) = \mathbf{mlns}(G) - 1 = \mathbf{lns}(G) - 1$$



$$S_{\sigma}^{(t)}(i) = \{x \in V \mid \sigma(x) < i \wedge \exists (x, \sigma_i)\text{-path with internal vertices in } \sigma_{>i}\}$$

Known relationship between parameters

Theorem.

- treewidth corresponds to lazy strategies [DKT97]

$$\mathbf{tw}(G) = \mathbf{tvs}(G) = \mathbf{mlns}(G) - 1 = \mathbf{lns}(G) - 1$$

$$S_\sigma^{(t)}(i) = \{x \in V \mid \sigma(x) < i \land \exists (x, \sigma_i)\text{-path with internal vertices in } \sigma_{>i}\}$$

$$\mathbf{tvs}(G) = \min_\sigma \max_{i \in [n]} |S_\sigma^{(t)}(i)|$$

Known relationship between parameters

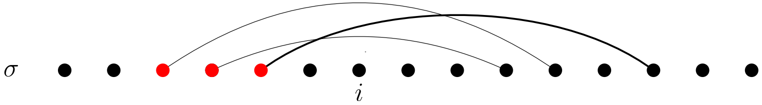
Theorem.

- treewidth corresponds to lazy strategies [DKT97]

$$\mathbf{tw}(G) = \mathbf{tvs}(G) = \mathbf{mlns}(G) - 1 = \mathbf{lns}(G) - 1$$

- pathwidth corresponds to agile strategies [Kin92, KP95]

$$\mathbf{pw}(G) = \mathbf{pvs}(G) = \mathbf{mans}(G) - 1 = \mathbf{ans}(G) - 1$$



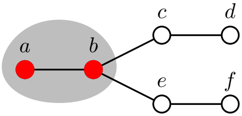
$$S_\sigma^{(p)}(i) = N_G(\sigma_{\geqslant i})$$

$$\mathbf{pvs}(G) = \min_\sigma \max_{i \in [n]} |S_\sigma^{(p)}(i)|$$

What about connected node search strategy ?

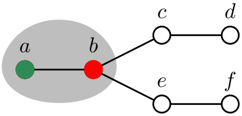
Hints : force to search the graph in a connected manner

$\rightsquigarrow$ the guarded space $\mathcal{G}_i = \overline{F_i}$ has to be connected



What about connected node search strategy ?

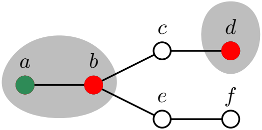
Hints : force to search the graph in a connected manner
$\rightsquigarrow$ the guarded space $\mathcal{G}_i = \overline{F_i}$ has to be connected



What about connected node search strategy ?

Hints : force to search the graph in a connected manner
$\rightsquigarrow$ the guarded space $\mathcal{G}_i = \overline{F_i}$ has to be connected

This is not a connected search !



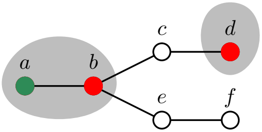
A node search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle$ is

- connected if for every $i \in [r]$, $\mathcal{G}_i$ is connected.

What about connected node search strategy ?

Hints : force to search the graph in a connected manner
$\rightsquigarrow$ the guarded space $\mathcal{G}_i = \overline{F_i}$ has to be connected

This is not a connected search !



A node search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle$ is

- connected if for every $i \in [r]$, $\mathcal{G}_i$ is connected.

Why connected search ?

- from the theoretical view point $\rightsquigarrow$ very natural constraint
- from the application view point:
  - cave exploration
  - maintenance of communications between searcher
  - ...

What about connected node search strategy ?

Questions

- What is the price of connectivity ?
- Can the **mclns**(.) parameter be expressed in terms of a layout parameter or a width parameter ?
- Can we characterize the set of graphs such that $\mathbf{mclns}(G) \leqslant k$?
- What is the complexity of deciding whether $\mathbf{mclns}(G) \leqslant k$?

Results (1) – Parameter equivalence

Theorem 1 [Adler, P., Thilikos (GRASTA'17)]

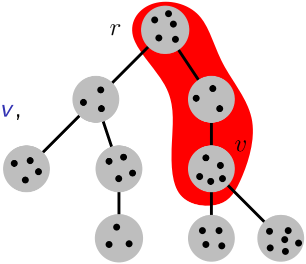
$$\mathbf{ctw}(G) = \mathbf{ctvs}(G) = \mathbf{mclns}(G) - 1$$

Results (1) – Parameter equivalence

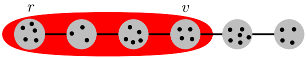
Theorem 1 [Adler, P., Thilikos (GRASTA'17)]

$$\mathbf{ctw}(G) = \mathbf{ctvs}(G) = \mathbf{mclns}(G) - 1$$

In a connected tree decomposition $(T, \mathcal{F})$, there exists a root r such that for every node v, $G[\cup\{X_u \mid u \in rTv\}]$ is connected



In a connected path decomposition, r is an extremity of the path:

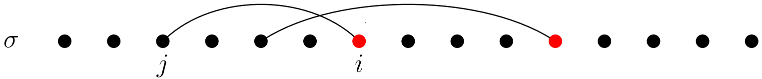


Results (1) – Parameter equivalence

Theorem 1 [Adler, P., Thilikos (GRASTA'17)]

$$\mathbf{ctw}(G) = \mathbf{ctvs}(G) = \mathbf{mclns}(G) - 1$$

Connected layout : for every i, there exists $j < i$ such that $\sigma_j \in N(\sigma_i)$

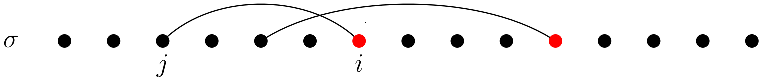


Results (1) – Parameter equivalence

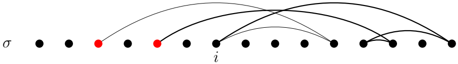
Theorem 1 [Adler, P., Thilikos (GRASTA'17)]

$$\textbf{ctw}(G) = \textbf{ctvs}(G) = \textbf{mclns}(G) - 1$$

Connected layout : for every i, there exists $j < i$ such that $\sigma_j \in N(\sigma_i)$



$\textbf{ctvs}(G) = \min_\sigma \max_{i \in [n]} |S^{(t)}_\sigma(i)|$, with σ a connected layout



$S^{(t)}_\sigma(i) = \{x \in V \mid \sigma(x) < i \wedge \exists (x, \sigma_i)\text{-path with internal vertices in } \sigma_{>i}\}$

Results (1) – Parameter equivalence

Theorem 1 [Adler, P., Thilikos, GRASTA'17]

$$\mathbf{ctw}(G) = \mathbf{ctvs}(G) = \mathbf{mclns}(G) - 1$$

Sketch of proof:

- $\mathbf{ctvs}(G) \leqslant \mathbf{mclns}(G) - 1$: search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle \rightsquigarrow$ layout σ

  $\sigma =$ vertices ordered by the first date they are occupied by a cops.

Results (1) – Parameter equivalence

Theorem 1 [Adler, P., Thilikos, GRASTA'17]

$$\mathbf{ctw}(G) = \mathbf{ctvs}(G) = \mathbf{mclns}(G) - 1$$

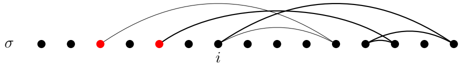
Sketch of proof:

- $\mathbf{ctvs}(G) \leqslant \mathbf{mclns}(G) - 1$: search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle \rightsquigarrow$ layout σ

  $\sigma =$ vertices ordered by the first date they are occupied by a cops.

- $\mathbf{ctw}(G) \leqslant \mathbf{ctvs}(G)$: connected layout $\sigma \rightsquigarrow$ tree-decomposition $(T, \mathcal{F})$

$$\mathcal{F} = \left\{ S_\sigma^{(t)}(i) \cup \{\sigma_i\} \ \middle| \ i \in [n] \right\}$$



Results (1) – Parameter equivalence

Theorem 1 [Adler, P., Thilikos, GRASTA'17]

$$\mathbf{ctw}(G) = \mathbf{ctvs}(G) = \mathbf{mclns}(G) - 1$$

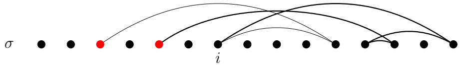
Sketch of proof:

- $\mathbf{ctvs}(G) \leqslant \mathbf{mclns}(G) - 1$: search strategy $\mathcal{S} = \langle S_1, \ldots S_r \rangle \rightsquigarrow$ layout σ

  $\sigma =$ vertices ordered by the first date they are occupied by a cops.

- $\mathbf{ctw}(G) \leqslant \mathbf{ctvs}(G)$: connected layout $\sigma \rightsquigarrow$ tree-decomposition $(T, \mathcal{F})$

$$\mathcal{F} = \left\{ S_\sigma^{(t)}(i) \cup \{\sigma_i\} \;\middle|\; i \in [n] \right\}$$

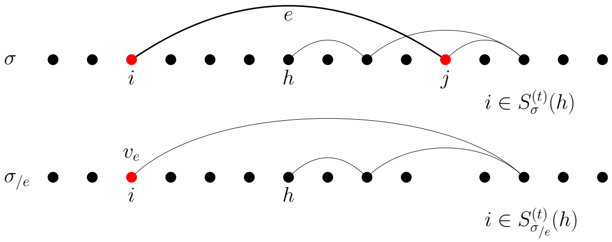


- $\mathbf{mclns}(G) \leqslant \mathbf{ctw}(G) + 1$: connected tree-decomposition $(T, \mathcal{F}) \rightsquigarrow \sigma$

$\sigma =$ vertex ordering resulting from a traversal of $(T, \mathcal{F})$ starting at the root

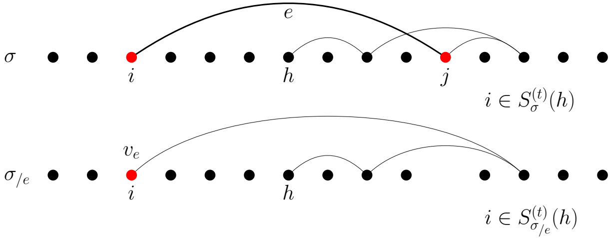
Contraction obstruction sets

Observation. The **mclns** parameter is closed under edge-contraction.



Contraction obstruction sets

Observation. The **mclns** parameter is closed under edge-contraction.

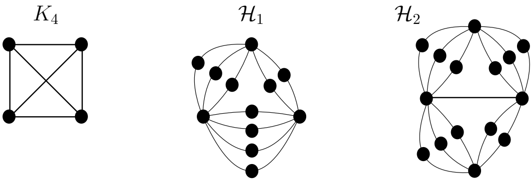


We define

- $\mathcal{C}_k = \{ G \mid \textbf{mclns}(G) \leqslant k \}$
- $\textbf{obs}(\mathcal{C}_k) = \{ G \mid \textbf{mclns}(G) > k \text{ and } \forall H, H \prec_c G, \textbf{mclns}(H) \leqslant k \}$

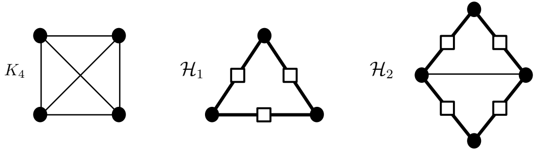
Results (2) – Obstruction set for $\mathcal{C}_2$

Theorem 2 [Adler, P., Thilikos (GRASTA'17)]
The set of obstructions for $\mathcal{C}_2$ is **obs**$(\mathcal{C}_2) = \{K_4\} \cup \mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{R}$ where



Results (2) – Obstruction set for $\mathcal{C}_2$

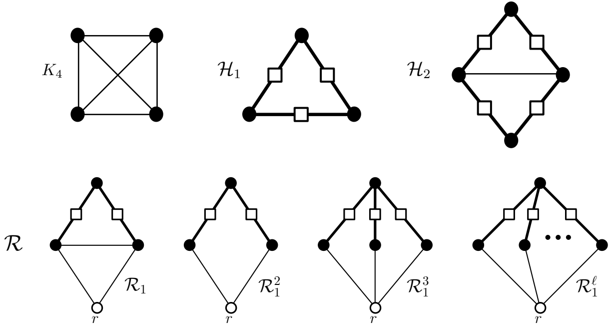
Theorem 2 [Adler, P., Thilikos (GRASTA'17)]
The set of obstructions for $\mathcal{C}_2$ is $\mathbf{obs}(\mathcal{C}_2) = \{K_4\} \cup \mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{R}$ where



- graphs of $\mathcal{H}_1 \cup \mathcal{H}_2$ are obtained by replacing thick subdivided edges by multiple subdivided edges;

Results (2) – Obstruction set for $\mathcal{C}_2$

Theorem 2 [Adler, P., Thilikos (GRASTA'17)]
The set of obstructions for $\mathcal{C}_2$ is $\mathbf{obs}(\mathcal{C}_2) = \{K_4\} \cup \mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{R}$ where

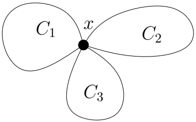


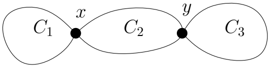
- graphs of $\mathcal{H}_1 \cup \mathcal{H}_2$ are obtained by replacing thick subdivided edges by multiple subdivided edges;
- graphs of $\mathcal{R}$ are obtained by gluing two graphs of $\mathcal{R}$ on their root vertex.

Obstruction set for $\mathcal{C}_2$ – some lemmas

Lemma. Let $G \in \mathbf{obs}(\mathcal{C}_k)$.

- If x is a cut-vertex, then $G - x$ contains two connected components;
- G contains at most one cut-vertex.

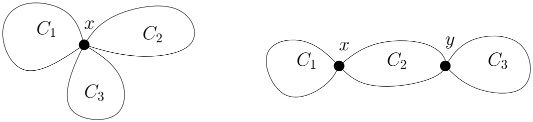




Obstruction set for $\mathcal{C}_2$ – some lemmas

Lemma. Let $G \in \mathbf{obs}(\mathcal{C}_k)$.

- If x is a cut-vertex, then $G - x$ contains two connected components;
- G contains at most one cut-vertex.



Sketch of proof: Suppose $G - x$ contains 3 connected components

As $G_{/C_1}$, $G_{/C_2}$, $G_{/C_3}$ are contractions:

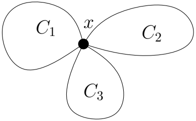
1. $\mathbf{ctvs}(C_1, x) \leqslant k$ or $\mathbf{ctvs}(C_2, x) \leqslant k$;
2. $\mathbf{ctvs}(C_2, x) \leqslant k$ or $\mathbf{ctvs}(C_3, x) \leqslant k$;
3. $\mathbf{ctvs}(C_3, x) \leqslant k$ or $\mathbf{ctvs}(C_1, x) \leqslant k$.

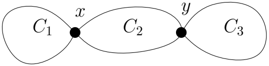
$\Rightarrow$ there exists σ such that $\mathbf{ctvs}(G, \sigma) \leqslant k$: contradiction.

Obstruction set for $\mathcal{C}_2$ – some lemmas

Lemma. Let $G \in \mathbf{obs}(\mathcal{C}_k)$.

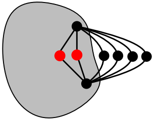
- If x is a cut-vertex, then $G - x$ contains two connected components;
- G contains at most one cut-vertex.





Twin-expansion Lemma.
Let x and y are two twin-vertices of degree 2 of a graph G and G^+ be the graph obtained from G by adding an arbitrary number of twins of x and y. Then

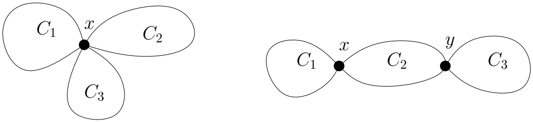
$$G \in \mathbf{obs}(\mathcal{C}_k) \text{ if and only if } G^+ \in \mathbf{obs}(\mathcal{C}_k).$$



Obstruction set for $\mathcal{C}_2$ – some lemmas

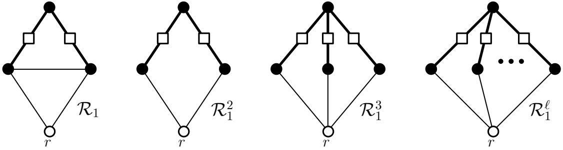
Lemma. Let $G \in \mathbf{obs}(\mathcal{C}_k)$.

- If x is a cut-vertex, then $G - x$ contains two connected components;
- G contains at most one cut-vertex.



Lemma. For every $k \geq 1$ and every connected graph G, $G \in \mathcal{O}_k$ is not a biconnected graph iff $G \in \{\mathbf{A} \oplus \mathbf{B} \mid \mathbf{A}, \mathbf{B} \in \mathcal{R}\}$.

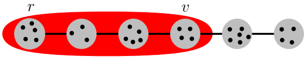
$\mathcal{R}$



Results (3) – Price of connectivity

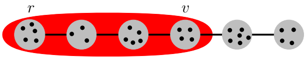
Theorem [Derenioswki'12]

$\mathbf{pw}(G) \leqslant \mathbf{cpw}(G) \leqslant 2 \cdot \mathbf{pw}(G) + 1$

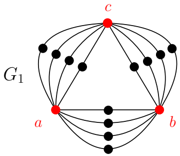


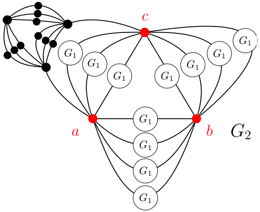
Results (3) – Price of connectivity

Theorem [Derenioswki'12]
$\mathbf{pw}(G) \leqslant \mathbf{cpw}(G) \leqslant 2 \cdot \mathbf{pw}(G) + 1$



Theorem [Adler, P., Thilikos, (GRASTA 2017)]
$\forall n \in \mathbb{N}$, $\exists G_n$ such that $\mathbf{mlns}(G_n) = 3$ and $\mathbf{mclns}(G_n) = 3 + n$

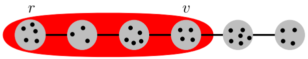




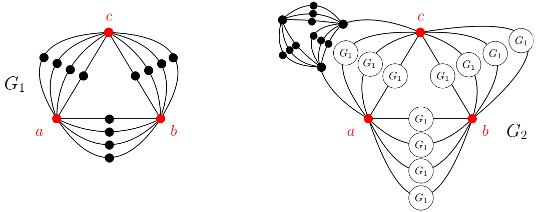
	tw	ctw	# of levels	# of parallel edges in highest level
G_1	2	3	1	4
G_2	2	4	2	5
G_3	2	5	3	6
G_4	2	6	4	7

Results (3) – Price of connectivity

Theorem [Derenioswki'12]
$\mathbf{pw}(G) \leqslant \mathbf{cpw}(G) \leqslant 2 \cdot \mathbf{pw}(G) + 1$



Theorem [Adler, P., Thilikos, (GRASTA 2017)]
$\forall n \in \mathbb{N}$, $\exists G_n$ such that $\mathbf{mlns}(G_n) = 3$ and $\mathbf{mclns}(G_n) = 3 + n$
and $|V(G_n)| = O(2^n)$. [Fraigniaud, Nisee'08]



	tw	ctw	# of levels	# of parallel edges in highest level
G_1	2	3	1	4
G_2	2	4	2	5
G_3	2	5	3	6
G_4	2	6	4	7

Computing the connected treewidth

$\rightsquigarrow$ A graph H is a contraction of a graph G, denoted $H \leqslant_c G$, if H is obtained from G by a series of contractions.

$\rightsquigarrow$ A graph H is a minor of a graph G, denoted $H \leqslant_m G$, if H is obtained from a subgraph G' of G by a series of contractions.

Computing the connected treewidth

$\rightsquigarrow$ A graph H is a contraction of a graph G, denoted $H \leqslant_c G$, if H is obtained from G by a series of contractions.

$\rightsquigarrow$ A graph H is a minor of a graph G, denoted $H \leqslant_m G$, if H is obtained from a subgraph G' of G by a series of contractions.

Theorem [Roberston & Seymour'84-04, Bodlaender'96]
There is an algorithm that, given a graph G and an integer k, decide whether $\mathbf{tw}(G) \leqslant k$ in $f(k) \cdot n^{O(1)}$ steps.

- $\rightsquigarrow$ $\mathbf{tw}(.)$ is a parameter closed under minor.
- $\rightsquigarrow$ graphs are well-quasi-ordered by the minor relation.
- $\rightsquigarrow$ minor testing can be performed in FPT-time.

Computing the connected treewidth

$\rightsquigarrow$ A graph H is a contraction of a graph G, denoted $H \leqslant_c G$, if H is obtained from G by a series of contractions.

$\rightsquigarrow$ A graph H is a minor of a graph G, denoted $H \leqslant_m G$, if H is obtained from a subgraph G' of G by a series of contractions.

Observation: $\mathcal{C}_k$ is closed under contraction not under minor !

- Can we decide whether $\mathbf{ctw}(G) \leqslant k$ in time

$$f(k) \cdot n^{O(1)} \text{ (FPT)} \quad \text{or} \quad n^{f(k)} \text{ (XP) ?}$$

Theorem [Dereniowski, Osula, Rzazweski'18]
There is an algorithm that, given a graph G and an integer k, decides whether $\mathbf{cpw}(G) \leqslant k$ in $n^{O(k^2)}$ steps.

Computing the connected treewidth

$\rightsquigarrow$ A graph H is a contraction of a graph G, denoted $H \leqslant_c G$, if H is obtained from G by a series of contractions.

$\rightsquigarrow$ A graph H is a minor of a graph G, denoted $H \leqslant_m G$, if H is obtained from a subgraph G' of G by a series of contractions.

Observation: $\mathcal{C}_k$ is closed under contraction not under minor !

- Can we decide whether $\mathbf{ctw}(G) \leqslant k$ in time

$$f(k) \cdot n^{O(1)} \text{ (FPT)} \quad \text{or} \quad n^{f(k)} \text{ (XP) ?}$$

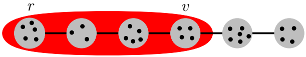
Theorem [Dereniowski, Osula, Rzazweski'18]
There is an algorithm that, given a graph G and an integer k, decides whether $\mathbf{cpw}(G) \leqslant k$ in $n^{O(k^2)}$ steps.

Theorem [Kante, P., Thilikos (GRASTA 2018)]
There is an algorithm that, given a graph G and an integer k, decides whether $\mathbf{cpw}(G) \leqslant k$ in $f(k) \cdot n^{O(1)}$ steps.

(Connected) path-decomposition and pathwidth

A path-decomposition of a graph G is a sequence $\mathcal{B} = [B_1, \dots B_r]$ st.

- for every $i \in [r]$, $B_i \subseteq V(G)$;
- for every $v \in V(G)$, $\exists i, j \in [r]$ st. $\forall i \leqslant k \leqslant j$, $v \in B_k$.



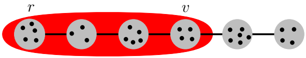
The path-decomposition $\mathcal{B}$ is connected if

- for every $i \in [r]$, the subgraph $G[\cup_{j \leqslant i} B_j]$ is connected.

(Connected) path-decomposition and pathwidth

A path-decomposition of a graph G is a sequence $\mathcal{B} = [B_1, \dots B_r]$ st.

- for every $i \in [r]$, $B_i \subseteq V(G)$;
- for every $v \in V(G)$, $\exists i, j \in [r]$ st. $\forall i \leqslant k \leqslant j$, $v \in B_k$.



The path-decomposition $\mathcal{B}$ is connected if

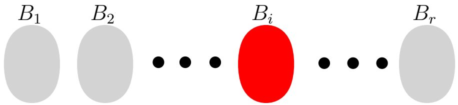
- for every $i \in [r]$, the subgraph $G[\cup_{j \leqslant i} B_j]$ is connected.

Theorem [Derenioswki'12] $\mathbf{pw}(G) \leqslant \mathbf{cpw}(G) \leqslant 2 \cdot \mathbf{pw}(G) + 1$

$\rightsquigarrow$ we may assume that

- $\mathbf{pw}(G) \leqslant 2k + 1$.
- $\mathcal{B} = [B_1, \dots B_r]$ is a nice path-decomposition of with at most $2k + 1$.

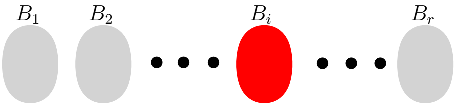
DP algorithm – connected path-decomposition of rooted graphs



At step i, we aim at computing a connected path-decomposition $\mathcal{A} = [A_1, \ldots A_q]$ of the rooted graph (G_i, B_i) where $G_i = G[\cup_{j \leqslant i} B_j]$.

Observation: The graph G_i may not be connected.

DP algorithm – connected path-decomposition of rooted graphs

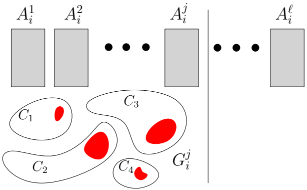


At step i, we aim at computing a connected path-decomposition $\mathcal{A} = [A_1, \ldots A_q]$ of the rooted graph (G_i, B_i) where $G_i = G[\cup_{j \leqslant i} B_j]$.

Observation: The graph G_i may not be connected.

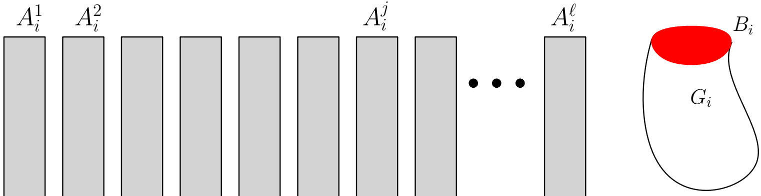
A path-decomposition $\mathcal{A}_i = [A_i^1, \ldots A_i^\ell]$ of a rooted graph (G_i, B_i) is connected if

- for every $j \in [\ell]$, every connected component of $G_i^j = G[\cup_{k \leqslant j} A_i^j]$ intersects B_i.



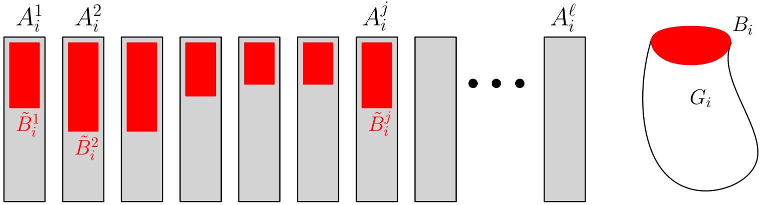
DP algorithm – encoding

$\mathcal{A}_i = [A_i^1, \ldots A_i^j, \ldots A_i^\ell]$ is a connected path-decomposition of (G_i, B_i)



DP algorithm – encoding

$\mathcal{A}_i = [A_i^1, \ldots A_i^j, \ldots A_i^\ell]$ is a connected path-decomposition of (G_i, B_i)

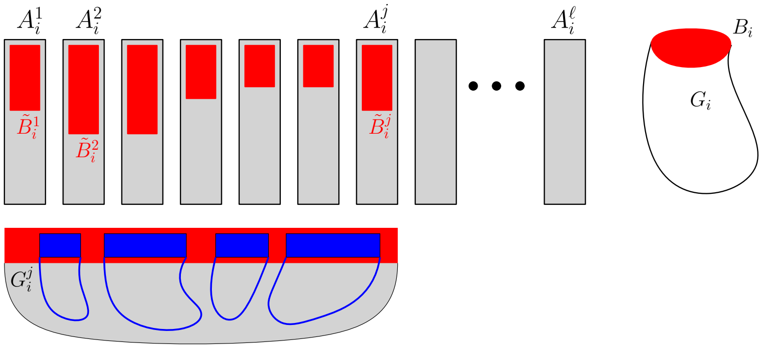


Each bag A_i^j is represented by a basic triple

$$\tilde{t}_i^j = (\tilde{B}_i^j = B_i \cap A_i^j \quad , \quad \tilde{\mathcal{C}}_i^j \quad , \quad z_i^j = |A_i^j \setminus B_i|)$$

DP algorithm – encoding

$\mathcal{A}_i = [A_i^1, \dots A_i^j, \dots A_i^\ell]$ is a connected path-decomposition of (G_i, B_i)



Each bag A_i^j is represented by a basic triple

$$\tilde{t}_i^j = (\tilde{B}_i^j = B_i \cap A_i^j \ , \ \tilde{\mathcal{C}}_i^j \ , \ z_i^j = |A_i^j \setminus B_i|)$$

where $\tilde{\mathcal{C}}_i^j$ is a partition of V_i^j such that every part X is the intersection of B_i with a connected component of G_i^j.

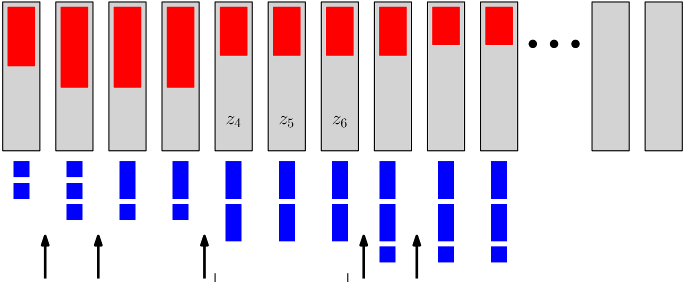
DP algorithm – encoding

Observation: The size of a basic triple is $O(\mathbf{pw}(G))$.
But ℓ can be arbitrarily large.

DP algorithm – encoding

Observation: The size of a basic triple is $O(\mathbf{pw}(G))$.
But ℓ can be arbitrarily large.

$\rightsquigarrow$ we need to compress the sequence of basic triples $[\tilde{t}_i^1, \ldots, \tilde{t}_i^\ell]$.

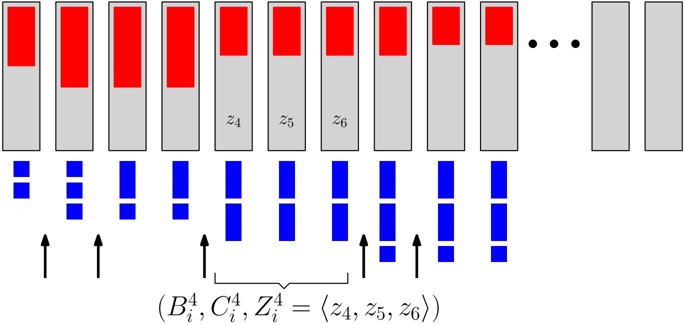


$(B_i^4, C_i^4, Z_i^4 = \langle z_4, z_5, z_6 \rangle)$

DP algorithm – encoding

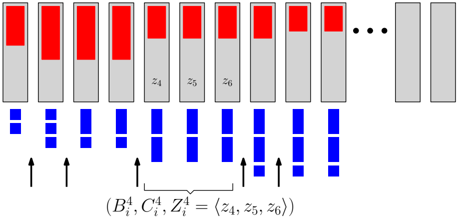
Observation: The size of a basic triple is $O(\mathbf{pw}(G))$.
But ℓ can be arbitrarily large.

$\rightsquigarrow$ we need to compress the sequence of basic triples $[\tilde{t}_i^1, \ldots, \tilde{t}_i^\ell]$.



$\rightsquigarrow$ Each sequence Z_i^j of integers in $[1, k]$ will be represented by its characteristic sequence of size $O(k)$. [Bodlaender & Kloks, 1996]

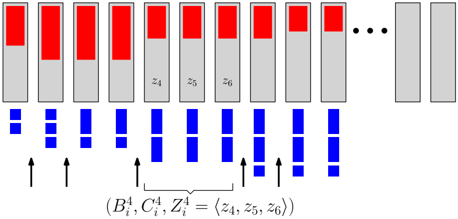
DP algorithm – encoding



Lemma [Representative sequence]

The size of the representative sequence for the path-decomposition $[A_i^1, \ldots A_i^\ell]$ of (G_i, B_i) is $O(\mathbf{pw}(G)^2)$.

DP algorithm – encoding



Lemma [Representative sequence]
The size of the representative sequence for the path-decomposition $[A_i^1, \ldots A_i^\ell]$ of (G_i, B_i) is $O(\mathbf{pw}(G)^2)$.

Lemma [Congruency]
If two boundaried graphs (G_1, B) and (G_2, B) have the same representative sequence, then for every boundaried graph (H, B)

$$\mathbf{cpw}((G_1, B) \oplus (H, B)) \leqslant k \Leftrightarrow \mathbf{cpw}((G_2, B) \oplus (H, B)) \leqslant k$$

DP algorithm

$\rightsquigarrow$ Build the set of characteristic sequence for (G_{i+1}, B_{i+1}) using the one of (G_i, B_i)

- Introduce node $B_{i+1} = B_i \cup \{v_{insert}\}$
- Forget node $B_i = B_{i+1} \cup \{v_{forget}\}$

DP algorithm

$\rightsquigarrow$ Build the set of characteristic sequence for (G_{i+1}, B_{i+1}) using the one of (G_i, B_i)

- Introduce node $B_{i+1} = B_i \cup \{v_{insert}\}$
- Forget node $B_i = B_{i+1} \cup \{v_{forget}\}$

Theorem [Kanté, P. Thilikos]

Given a graph G, we can decide if $\mathbf{cpw}(G) \leqslant k$ in time $2^{O(k^2)} \cdot n$.

Conclusion

Open problems

- What is the complexity of deciding whether $\mathbf{ctw}(G) \leqslant k$?

  $\rightsquigarrow$ Can it be solved in FPT time, or even XP time ?
  $\rightsquigarrow$ Or provide an hardness proof.

- What is the complexity of deciding whether $\mathbf{ctw}(G) \leqslant k$ when parameterized by $\mathbf{tw}(G)$? (assuming a positive answer to the previous question)

  $\rightsquigarrow$ Can it be solved in FPT time, or even XP time ?
  $\rightsquigarrow$ Or provide an hardness proof.

Conclusion

Open problems

- What is the complexity of deciding whether $\mathbf{ctw}(G) \leqslant k$?

  $\rightsquigarrow$ Can it be solved in FPT time, or even XP time ?
  $\rightsquigarrow$ Or provide an hardness proof.

- What is the complexity of deciding whether $\mathbf{ctw}(G) \leqslant k$ when parameterized by $\mathbf{tw}(G)$? (assuming a positive answer to the previous question)

  $\rightsquigarrow$ Can it be solved in FPT time, or even XP time ?
  $\rightsquigarrow$ Or provide an hardness proof.

  Theorem [Mescoff, P., Thilikos (GRASTA 2018)]
  If G is a series-parallel graph (i.e. $\mathbf{tw}(G) = 2$),
  then we can decide if $\mathbf{ctw}(G) \leqslant k$ in time $n^{O(1)}$.

Conclusion

Open problems

- What is the complexity of deciding whether $\mathbf{ctw}(G) \leqslant k$?

  $\rightsquigarrow$ Can it be solved in FPT time, or even XP time ?
  $\rightsquigarrow$ Or provide an hardness proof.

- What is the complexity of deciding whether $\mathbf{ctw}(G) \leqslant k$ when parameterized by $\mathbf{tw}(G)$? (assuming a positive answer to the previous question)

  $\rightsquigarrow$ Can it be solved in FPT time, or even XP time ?
  $\rightsquigarrow$ Or provide an hardness proof.

  Theorem [Mescoff, P., Thilikos (GRASTA 2018)]
  If G is a series-parallel graph (i.e. $\mathbf{tw}(G) = 2$),
  then we can decide if $\mathbf{ctw}(G) \leqslant k$ in time $n^{O(1)}$.

- Identify problems that are hard with respect to $\mathbf{tw}(.)$ but not with respect to $\mathbf{ctw}(.)$.

- Describe the set of obstructions for $k \geqslant 3$.

Conclusion – connected treewidth

- [P. Fraigniaud, N. Nisse, LATIN'06]

  $\rightsquigarrow$ To each edge e_T of the tree-decomposition we associate two graphs $G_1^{e_T}$ and $G_2^{e_T}$ that need to be connected.

- [P. Jégou, C. Terrioux, Constraints'17], [Diestel, Combinatorica'17]

  $\rightsquigarrow$ every bag of the tree decomposition $(T, \mathcal{F})$ induces a connected subgraph

  - [IA, Constraints] : efficient heuristics based on the structure of the constraint network to fasten backtracking strategies;
  - [Graph theory] : duality theorem, relation to graph hyperbolicity.

Thank to the organizers !