Twin-width and polynomial kernels

Amadeus Reinald
ENS de Lyon

joint work with

Édouard Bonnet[1], Eun Jung Kim,
Stéphan Thomassé, Rémi Watrigant

Séminaire COATI
May 3rd, 2022, INRIA Sophia-Antipolis

[1] some slides have been recklessly stolen from Édouard

Overview

Twin-width:

Overview

Twin-width:

- graph invariant giving a **"structural complexity"** measure.

Overview

Twin-width:

- graph invariant giving a "**structural complexity**" measure.
- classes of small twin-width ⇒ efficient algorithms.

Overview

Twin-width:

- graph invariant giving a **"structural complexity"** measure.
- classes of small twin-width ⇒ efficient algorithms.

Polynomial kernels:

Overview

Twin-width:

- graph invariant giving a **"structural complexity"** measure.
- classes of small twin-width $\Rightarrow$ efficient algorithms.

Polynomial kernels:

- Even **more efficient** algorithms based on pre-processing.

Overview

Twin-width:

- graph invariant giving a **"structural complexity"** measure.
- classes of small twin-width ⇒ efficient algorithms.

Polynomial kernels:

- Even **more efficient** algorithms based on pre-processing.

Question: What problems admit polynomial kernels for classes of small twin-width ?

Our results

Theorem (Bonnet, Kim, R., Thomassé, Watrigant '21)
On classes of bounded twin-width, CONNECTED k-VERTEX COVER *admits kernels with* $O(k^{1.5})$ *vertices.*

Our results

Theorem (Bonnet, Kim, R., Thomassé, Watrigant '21)
On classes of bounded twin-width, Connected k-Vertex Cover *admits kernels with* $O(k^{1.5})$ *vertices.*

In this talk:

Theorem (Bonnet, Kim, R., Thomassé, Watrigant '21)
Unless coNP $\subseteq$ NP/*poly,* k-Dominating Set *does not admit a polynomial kernel on graphs of twin-width at most 4.*

Parameters for decision problems

Decision problems:

- "Does instance I admit a solution of size k ?",

Parameters for decision problems

Decision problems:

- "Does instance I admit a solution of size k ?",

For NP-hard problems, complexity exponential in $|I|$ (ETH)...

Parameters for decision problems

Decision problems:

- "Does instance I admit a solution of size k ?",

For NP-hard problems, complexity exponential in $|I|$ (ETH)...
$\hookrightarrow$ Is large $|I|$ really what makes a hard instance ?

Parameters for decision problems

Decision problems:

- "Does instance I admit a solution of size k ?",

For NP-hard problems, complexity exponential in $|I|$ (ETH)...
$\hookrightarrow$ Is large $|I|$ really what makes a hard instance ?

Goal: **Parameters** k capturing "complexity" better than $|I|$.

Parameters for decision problems

Decision problems:

- "Does instance I admit a solution of size k ?",

For NP-hard problems, complexity exponential in $|I|$ (ETH)...
$\hookrightarrow$ Is large $|I|$ really what makes a hard instance ?

Goal: **Parameters** k capturing "complexity" better than $|I|$.

- **Solution size**,

Parameters for decision problems

Decision problems:

- "Does instance I admit a solution of size k ?",

For NP-hard problems, complexity exponential in $|I|$ (ETH)...
$\hookrightarrow$ Is large $|I|$ really what makes a hard instance ?

Goal: **Parameters** k capturing "complexity" better than $|I|$.

- **Solution size**,
- Measures of "structural complexity" for graphs: diameter, genus, treewidth...

Fixed-Parameter Tractability (FPT)

Goal: small parameter instances ⇒ polytime algorithms.

Fixed-Parameter Tractability (FPT)

Goal: small parameter instances ⇒ polytime algorithms.

A **parameterized problem** has instances (I, k) with parameter k.

Fixed-Parameter Tractability (FPT)

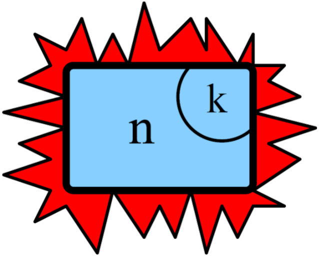
Goal: small parameter instances ⇒ polytime algorithms.
A **parameterized problem** has instances (I, k) with parameter k.

> **Definition** *A parameterized problem $\mathcal{Q}$ is* ***fixed-parameter tractable*** *(FPT) with respects to k if there is an algorithm deciding (I, k) in time $f(k)|I|^{O(1)}$, for some computable f.*

Fixed-Parameter Tractability (FPT)

Goal: small parameter instances ⇒ polytime algorithms.
A **parameterized problem** has instances (I, k) with parameter k.

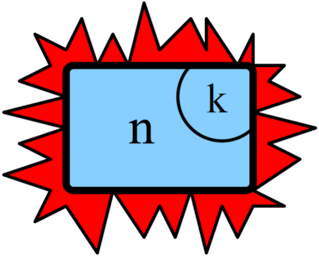
Definition *A parameterized problem $\mathcal{Q}$ is **fixed-parameter tractable** (FPT) with respects to k if there is an algorithm deciding (I, k) in time $f(k)|I|^{O(1)}$, for some computable f.*

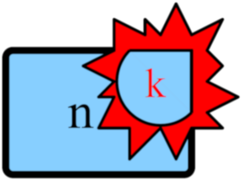


Fixed-Parameter Tractability (FPT)

Goal: small parameter instances ⇒ polytime algorithms.
A **parameterized problem** has instances (I, k) with parameter k.

Definition *A parameterized problem $\mathcal{Q}$ is* ***fixed-parameter tractable*** *(FPT) with respects to k if there is an algorithm deciding (I, k) in time $f(k)|I|^{O(1)}$, for some computable f.*





FPT Examples

Examples of FPT problems:

- k-Vertex Cover parameterized by solution size k:
  $\hookrightarrow$ instance (G, k) solvable in time $O(f(k)|G|^3)$.

FPT Examples

Examples of FPT problems:

- k-Vertex Cover parameterized by solution size k:
  $\hookrightarrow$ instance (G, k) solvable in time $O(f(k)|G|^3)$.
- Hamiltonian Cycle parameterized by treewidth tw:
  $\hookrightarrow$ instance (G, tw) solvable in time $O(tw^{tw}|G|)$

Kernels

How to devise an FPT algorithm? One strategy:

Kernels

How to devise an FPT algorithm? One strategy:

- Pre-process input to get rid of "useless" information,

Kernels

How to devise an FPT algorithm? One strategy:

- Pre-process input to get rid of "useless" information,
- obtain "small" sized instance dependent only on k,

Kernels

How to devise an FPT algorithm? One strategy:

- Pre-process input to get rid of "useless" information,
- obtain "small" sized instance dependent only on k,
- Run bruteforce → complexity explosion only in k.

Kernels

How to devise an FPT algorithm? One strategy:

- Pre-process input to get rid of "useless" information,
- obtain "small" sized instance dependent only on k,
- Run bruteforce → complexity explosion only in k.

Definition ***Kernel*** *for parameterized problem* $\mathcal{Q}$:

- *Polytime reduction from instance* (I, k) *for* $\mathcal{Q}$ *to equivalent instance* (I', k') *for* $\mathcal{Q}$.
- *output instance has size* $|I'| = h(k)$, *and* k' *is bounded by a function of* k.

Better than FPT: polynomial kernels

Theorem

A parameterized problem is FPT iff it admits a kernel.

Better than FPT: polynomial kernels

Theorem
A parameterized problem is FPT iff it admits a kernel.

For FPT problems, can we get even **more efficient?**

Better than FPT: polynomial kernels

Theorem
A parameterized problem is FPT iff it admits a kernel.

For FPT problems, can we get even **more efficient?**

- FPT ⇒ Kernel, but the kernel size can be very large...

Better than FPT: polynomial kernels

Theorem
A parameterized problem is FPT iff it admits a kernel.

For FPT problems, can we get even **more efficient?**

- FPT $\Rightarrow$ Kernel, but the kernel size can be very large...
- A kernel for parameter k is **polynomial** if $|I'| = O(k^p)$.

Better than FPT: polynomial kernels

Theorem

A parameterized problem is FPT iff it admits a kernel.

For FPT problems, can we get even **more efficient?**

- FPT ⇒ Kernel, but the kernel size can be very large...
- A kernel for parameter k is **polynomial** if $|I'| = O(k^p)$.

Problems with polynomial kernels:

Better than FPT: polynomial kernels

Theorem

A parameterized problem is FPT iff it admits a kernel.

For FPT problems, can we get even **more efficient?**

- FPT ⇒ Kernel, but the kernel size can be very large...
- A kernel for parameter k is **polynomial** if $|I'| = O(k^p)$.

Problems with polynomial kernels:

- k-Vertex-Cover, k-Dominating-Set on sparse graphs ($K_{t,t}$-free).

Better than FPT: polynomial kernels

Theorem

A parameterized problem is FPT iff it admits a kernel.

For FPT problems, can we get even **more efficient?**

- FPT ⇒ Kernel, but the kernel size can be very large...
- A kernel for parameter k is **polynomial** if $|I'| = O(k^p)$.

Problems with polynomial kernels:

- k-Vertex-Cover, k-Dominating-Set on sparse graphs ($K_{t,t}$-free).

↪ can we prove that a problem has no polykernel?

Example: k-Longest Path

LONGEST PATH	**Parameter:** k
Input: Graph G, integer k	
Question: Does G have a simple path of length k?	

Example: k-Longest Path

Longest Path	**Parameter:** k
Input: Graph G, integer k	
Question: Does G have a simple path of length k?	

- NP-hard, but FPT: admits a $2^k n^c$ algorithm, thus a kernel,

Example: k-Longest Path

Longest Path	**Parameter:** k
Input: Graph G, integer k	
Question: Does G have a simple path of length k?	

- NP-hard, but FPT: admits a $2^k n^c$ algorithm, thus a kernel,
- what about a polynomial kernel?

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), ..., (G_t, k)$, with $n = |G_i|$,

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), \ldots, (G_t, k)$, with $n = |G_i|$,
- Take $G = \bigcup_{i=1}^{t} G_i$, and consider instance (G, k),

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), \ldots, (G_t, k)$, with $n = |G_i|$,
- Take $G = \bigcup_{i=1}^{t} G_i$, and consider instance (G, k),
- Polykernel returns (G', k') from (G, k) in $poly(n)$.

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), \ldots, (G_t, k)$, with $n = |G_i|$,
- Take $G = \bigcup_{i=1}^{t} G_i$, and consider instance (G, k),
- Polykernel returns (G', k') from (G, k) in $poly(n)$.

(G, k) is positive $\Leftrightarrow$ (G', k') is positive $\Leftrightarrow$ $\exists i : (G_i, k)$ is positive.

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), ..., (G_t, k)$, with $n = |G_i|$,
- Take $G = \bigcup_{i=1}^{t} G_i$, and consider instance (G, k),
- Polykernel returns (G', k') from (G, k) in $poly(n)$.

(G, k) is positive $\Leftrightarrow$ (G', k') is positive $\Leftrightarrow$ $\exists i : (G_i, k)$ is positive.

- G' is computed in polytime,

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), ..., (G_t, k)$, with $n = |G_i|$,
- Take $G = \bigcup_{i=1}^{t} G_i$, and consider instance (G, k),
- Polykernel returns (G', k') from (G, k) in $poly(n)$.

(G, k) is positive $\Leftrightarrow$ (G', k') is positive $\Leftrightarrow$ $\exists i : (G_i, k)$ is positive.

- G' is computed in polytime,
- G' is (very) small: $|G'| \leqslant k^c \leqslant n^c < t$: **less than one bit per initial instance**,

k-Longest Path has no polykernels (morally)

Assume polykernel of size k^c.

- Consider $t = n^c + 1$ instances $(G_1, k), ..., (G_t, k)$, with $n = |G_i|$,
- Take $G = \bigcup_{i=1}^{t} G_i$, and consider instance (G, k),
- Polykernel returns (G', k') from (G, k) in $poly(n)$.

(G, k) is positive $\Leftrightarrow$ (G', k') is positive $\Leftrightarrow$ $\exists i : (G_i, k)$ is positive.

- G' is computed in polytime,
- G' is (very) small: $|G'| \leqslant k^c \leqslant n^c < t$: **less than one bit per initial instance**,
- kernelization must have dismissed / "solved" one G_i entirely... in polynomial time → absurd.

Ruling out polynomial kernels

Formalization: given problem $\mathscr{L}$, and parameterized problem $\mathscr{Q}$.

Definition 2.1 (OR-composition) Polytime reduction:

Ruling out polynomial kernels

Formalization: given problem $\mathscr{L}$, and parameterized problem $\mathscr{Q}$.

Definition 2.2 (OR-composition) Polytime reduction:

- Input: Instances $I_1, \ldots, I_t$ for $\mathscr{L}$,

Ruling out polynomial kernels

Formalization: given problem $\mathscr{L}$, and parameterized problem $\mathscr{Q}$.

Definition 2.3 (OR-composition) Polytime reduction:

- Input: Instances $I_1, \ldots, I_t$ for $\mathscr{L}$,
- Output: Instance J for $\mathscr{Q}$.

Ruling out polynomial kernels

Formalization: given problem $\mathscr{L}$, and parameterized problem $\mathscr{Q}$.

Definition 2.4 (OR-composition) Polytime reduction:

- Input: Instances $I_1, \ldots, I_t$ for $\mathscr{L}$,
- Output: Instance J for $\mathscr{Q}$.

J is positive for $\mathscr{Q}$ $\Leftrightarrow$ $\exists i$ s.t. I_i is positive for $\mathscr{L}$.

Ruling out polynomial kernels

Formalization: given problem $\mathscr{L}$, and parameterized problem $\mathscr{Q}$.

Definition 2.5 (OR-composition) Polytime reduction:

- Input: Instances $I_1, \ldots, I_t$ for $\mathscr{L}$,
- Output: Instance J for $\mathscr{Q}$.

J is positive for $\mathscr{Q} \Leftrightarrow \exists i$ s.t. I_i is positive for $\mathscr{L}$.

Theorem (Bodlaender, Jansen, Kratsch '12)
If an NP-hard problem $\mathscr{L}$ admits an OR-composition into a parameterized problem $\mathscr{Q}$, then $\mathscr{Q}$ does not admit a polynomial kernel unless coNP $\subseteq$ NP/*poly.*

Graph invariants and FPT

"Meta-theorems" for problems expressible by logic formulas:

[2] given a tree decomposition

Graph invariants and FPT

"Meta-theorems" for problems expressible by logic formulas:

> **Theorem 1 (Courcelle '90)** *Given a **MSO formula** ϕ on a graph G, deciding whether ϕ is satisfied by G is **FPT** in $|\phi|$ on graphs of **bounded tree-width**[2].*

[2] given a tree decomposition

Graph invariants and FPT

"Meta-theorems" for problems expressible by logic formulas:

Theorem 1 (Courcelle '90) *Given a* ***MSO formula*** ϕ *on a graph* G*, deciding whether* ϕ *is satisfied by* G *is* ***FPT*** *in* $|\phi|$ *on graphs of* ***bounded tree-width***[2]*.*

Lot of **FO** problems are FPT on some classes of unbounded tw, goal:

- Analogue of Courcelle for FO logic ?
- With some **broader** parameter than tw

[2] given a tree decomposition

Graph invariants and FPT

"Meta-theorems" for problems expressible by logic formulas:

Theorem 1 (Courcelle '90) *Given a* ***MSO formula*** ϕ *on a graph* G*, deciding whether* ϕ *is satisfied by* G *is* ***FPT*** *in* $|\phi|$ *on graphs of* ***bounded tree-width***[2].

Lot of **FO** problems are FPT on some classes of unbounded tw, goal:

- Analogue of Courcelle for FO logic ?
- With some **broader** parameter than tw **twin-width**

[2] given a tree decomposition

Motivation

Definition *In graph G, $u, v \in V(G)$ are* ***twins*** *if $N(u) = N(v)$.*

Motivation

Definition *In graph G, $u, v \in V(G)$ are* ***twins*** *if $N(u) = N(v)$.*

- Twins "behave the same" w.r.t the rest of the graph,

Motivation

Definition *In graph G, $u, v \in V(G)$ are* ***twins*** *if $N(u) = N(v)$.*

- Twins "behave the same" w.r.t the rest of the graph,
- exploitable algorithmically: modular decomposition, cographs...

Motivation

Definition *In graph G, $u, v \in V(G)$ are* ***twins*** *if $N(u) = N(v)$.*

- Twins "behave the same" w.r.t the rest of the graph,
- exploitable algorithmically: modular decomposition, cographs...

Motivation

Definition *In graph G, $u, v \in V(G)$ are* ***twins*** *if $N(u) = N(v)$.*

- Twins "behave the same" w.r.t the rest of the graph,
- exploitable algorithmically: modular decomposition, cographs...

Idea:

- Obtain efficient algorithms by leveraging "**near-twins**",

Motivation

Definition *In graph G, $u, v \in V(G)$ are **twins** if $N(u) = N(v)$.*

- Twins "behave the same" w.r.t the rest of the graph,
- exploitable algorithmically: modular decomposition, cographs...

Idea:

- Obtain efficient algorithms by leveraging "**near-twins**",
- Efficiency depends on how "**near**" vertices are to being twins → twin-**width**.

Graphs and trigraphs

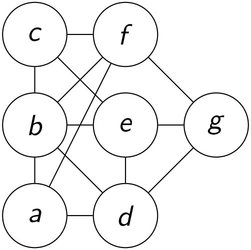


Fig. Graph: edges, non-edges

Graphs and trigraphs

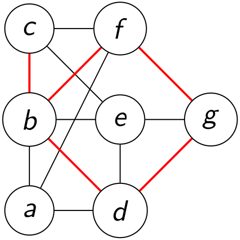
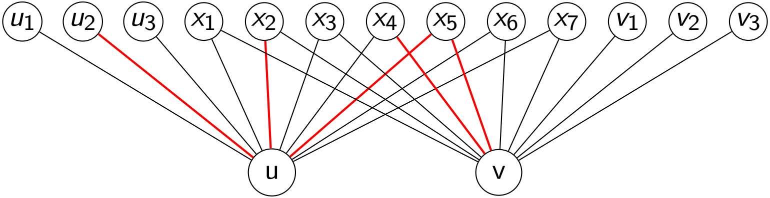


Fig. Trigraph: edges, non-edges, or **red** edges

Contractions in trigraphs

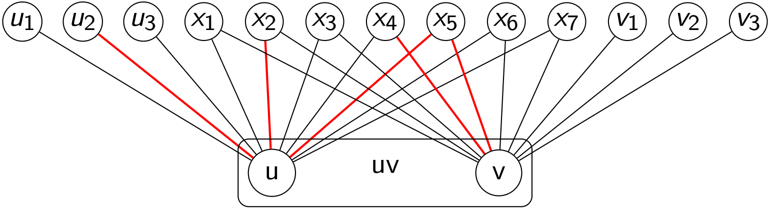
Intuitive goal: group near-twins together.



Contractions in trigraphs

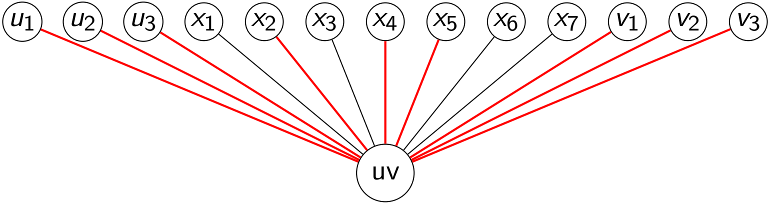
Intuitive goal: group near-twins together.

Contraction of u and v: record "twin" errors with red edges.



Contractions in trigraphs

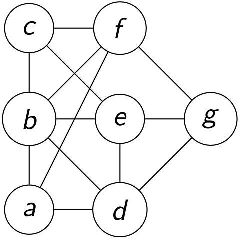
Intuitive goal: group near-twins together.
Contraction of u and v: record "twin" errors with red edges.



edges to $N(u) \triangle N(v)$ turn red, for $N(u) \cap N(v)$ red is absorbing

Twin-width

Definition (Twin-width) *tww(G): minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

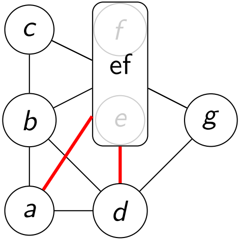


Maximum red degree $= 0$

overall maximum red degree = 0

Twin-width

Definition (Twin-width) *tww(G): minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

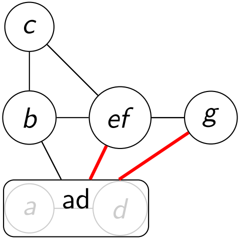


Maximum red degree = 2

overall maximum red degree = 2

Twin-width

Definition (Twin-width) *$tww(G)$: minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

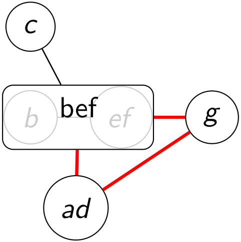


Maximum red degree = 2

overall maximum red degree = 2

Twin-width

Definition (Twin-width) *tww(G): minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

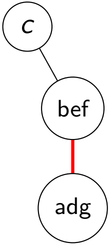


Maximum red degree = 2

overall maximum red degree = 2

Twin-width

Definition (Twin-width) *tww(G): minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

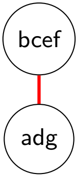


Maximum red degree = 1

overall maximum red degree = 2

Twin-width

Definition (Twin-width) *tww(G): minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*



Maximum red degree = 1
overall maximum red degree = 2

Twin-width

Definition (Twin-width) *$tww(G)$: minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

Maximum red degree $= 0$
overall maximum red degree = 2 .

Twin-width

Definition (Twin-width) *$tww(G)$: minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

Maximum red degree $= 0$
overall maximum red degree = 2 $\rightarrow tww(G) \leqslant 2$.

Twin-width

Definition (Twin-width) *tww*(G)*: minimal* d *such that* G *admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most* d.

- Contraction sequences → dynamic programming
-

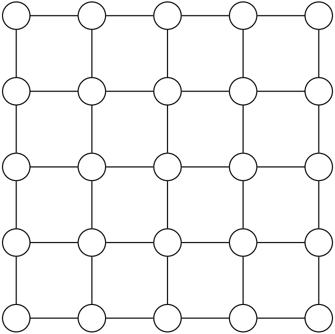
Twin-width

Definition (Twin-width) *$tww(G)$: minimal d such that G admits a* ***contraction sequence*** *where all trigraphs have* ***maximum red degree*** *at most d.*

- Contraction sequences → dynamic programming
- bounded red degree → few "complicated" updates.

Example: twin-width of grids

Grids have unbounded tree-width... what about twin-width?

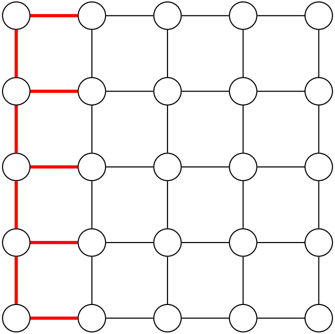


Maximum red degree = 0

overall maximum red degree = 0

Example: twin-width of grids

Grids have unbounded tree-width... what about twin-width?

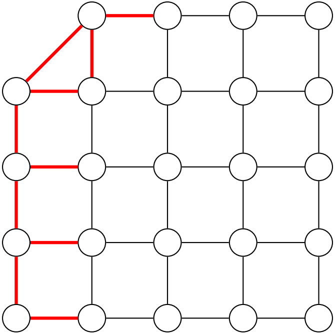


Maximum red degree = 3

overall maximum red degree = 3

Example: twin-width of grids

Grids have unbounded tree-width... what about twin-width?

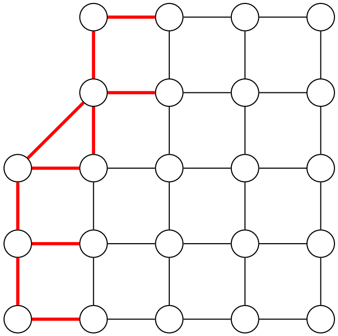


Maximum red degree = 4

overall maximum red degree = 4

Example: twin-width of grids

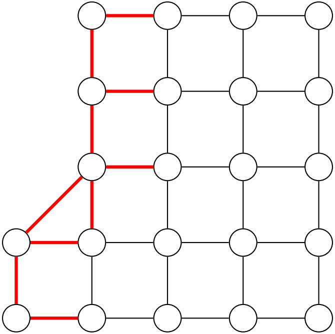
Grids have unbounded tree-width... what about twin-width?



Maximum red degree = 4
overall maximum red degree = 4

Example: twin-width of grids

Grids have unbounded tree-width... what about twin-width?

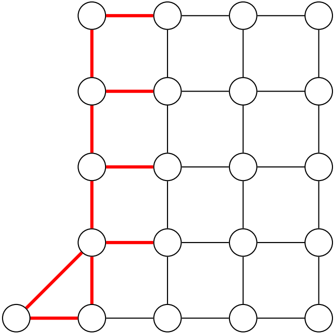


Maximum red degree = 4

overall maximum red degree = 4

Example: twin-width of grids

Grids have unbounded tree-width... what about twin-width?

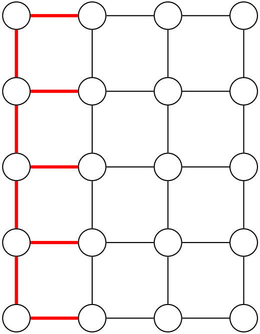


Maximum red degree = 4

overall maximum red degree = 4

Example: twin-width of grids

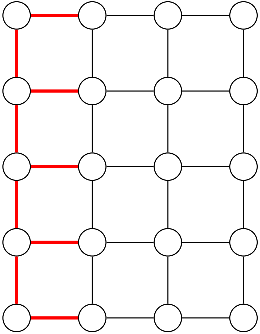
Grids have unbounded tree-width... what about twin-width?



Maximum red degree = 3
overall maximum red degree = 4

Example: twin-width of grids

Grids have unbounded tree-width... what about twin-width?

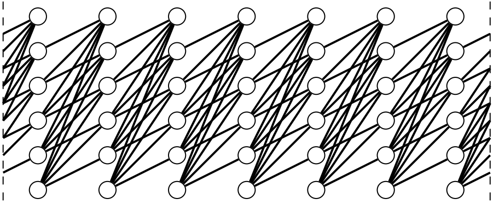


Maximum red degree = 3

overall maximum red degree = 4 ⇒ twin-width $\leqslant 4$

Example: Half-Graph Cycles

A dense graph with bounded twin-width:

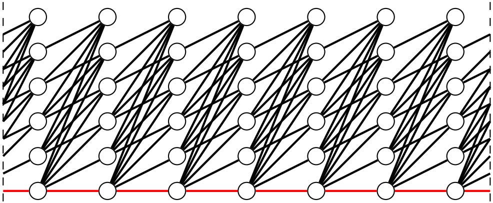


Maximum red degree = 0

overall maximum red degree = 0

Example: Half-Graph Cycles

A dense graph with bounded twin-width:

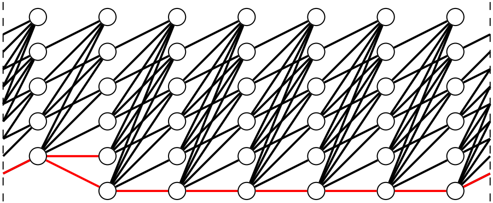


Maximum red degree = 2

overall maximum red degree = 2

Example: Half-Graph Cycles

A dense graph with bounded twin-width:

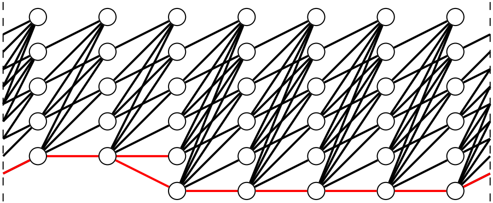


Maximum red degree = 3

overall maximum red degree = 3

Example: Half-Graph Cycles

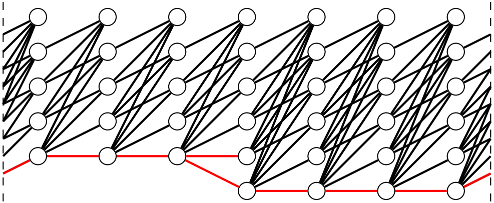
A dense graph with bounded twin-width:



Maximum red degree = 3
overall maximum red degree = 3

Example: Half-Graph Cycles

A dense graph with bounded twin-width:

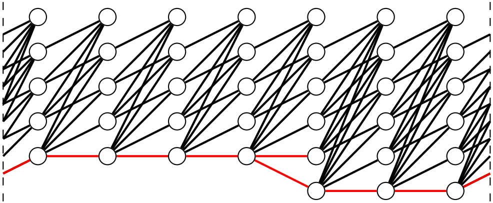


Maximum red degree = 3

overall maximum red degree = 3

Example: Half-Graph Cycles

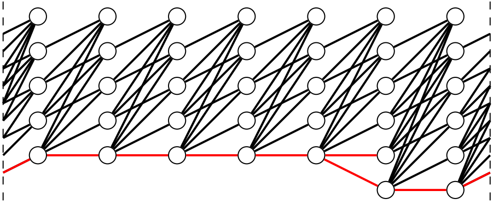
A dense graph with bounded twin-width:



Maximum red degree = 3
overall maximum red degree = 3

Example: Half-Graph Cycles

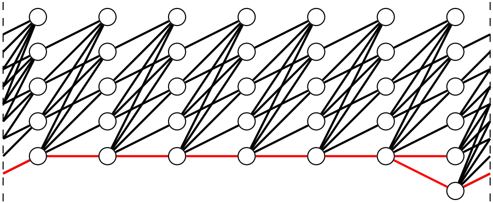
A dense graph with bounded twin-width:



Maximum red degree = 3
overall maximum red degree = 3

Example: Half-Graph Cycles

A dense graph with bounded twin-width:

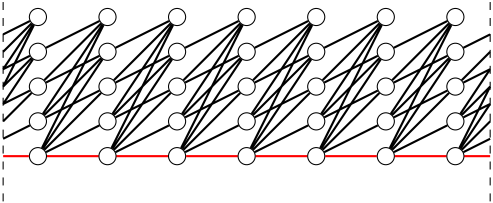


Maximum red degree $= 3$

overall maximum red degree = 3

Example: Half-Graph Cycles

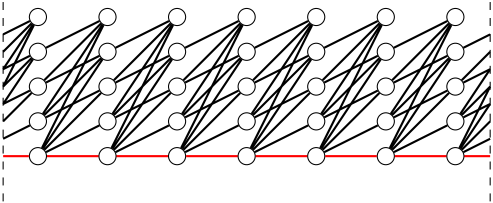
A dense graph with bounded twin-width:



Maximum red degree = 2
overall maximum red degree = 3

Example: Half-Graph Cycles

A dense graph with bounded twin-width:



Maximum red degree $= 2$

overall maximum red degree = 3 $\Rightarrow$ **twin-width** $\leqslant 3$

Twin-width : FPT first order problems

Recall Courcelle for MSO and tree-width:

[3] given a tree decomposition
[4] given a contraction sequence

Twin-width : FPT first order problems

Recall Courcelle for MSO and tree-width:

Theorem (Courcelle '90)
*Given a **MSO formula** ϕ on a graph G, deciding ϕ on G is **FPT** in $|\phi|$ on graphs of **bounded tree-width**[3].*

[3] given a tree decomposition
[4] given a contraction sequence

Twin-width : FPT first order problems

Recall Courcelle for MSO and tree-width:

> **Theorem (Courcelle '90)**
> *Given a **MSO formula** ϕ on a graph G, deciding ϕ on G is **FPT** in $|\phi|$ on graphs of **bounded tree-width**[3].*

Analogue: FO and twin-width:

[3] given a tree decomposition
[4] given a contraction sequence

Twin-width : FPT first order problems

Recall Courcelle for MSO and tree-width:

Theorem (Courcelle '90)
*Given a **MSO formula** ϕ on a graph G, deciding ϕ on G is **FPT** in $|\phi|$ on graphs of **bounded tree-width**[3].*

Analogue: FO and twin-width:

Theorem (Bonnet, Kim, Thomassé, Watrigant '20)
*Given a **FO formula** ϕ on a graph G, deciding ϕ on G is **FPT** with respects to $|\phi|$ on classes of **bounded twin-width**[4].*

[3] given a tree decomposition
[4] given a contraction sequence

Consequences

Some **problems** expressible in FO:

Consequences

Some **problems** expressible in FO:

- k-IS : $\exists x_1 ... \exists x_k \bigwedge_{1 \leqslant i \leqslant j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$,

Consequences

Some **problems** expressible in FO:

- k-IS : $\exists x_1 ... \exists x_k \bigwedge_{1 \leqslant i \leqslant j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$,
- k-DS : $\exists x_1 ... \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k}(x = x_i) \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i)$.

are **FPT in solution size** k for bounded tww.

Consequences

Some **problems** expressible in FO:

- k-IS : $\exists x_1 ... \exists x_k \bigwedge_{1 \leqslant i \leqslant j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$,
- k-DS : $\exists x_1 ... \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} (x = x_i) \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i)$.

are **FPT in solution size** k for bounded tww.

Some **classes** of bounded twin-width:

Consequences

Some **problems** expressible in FO:

- k-IS : $\exists x_1 ... \exists x_k \bigwedge_{1 \leqslant i \leqslant j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$,
- k-DS : $\exists x_1 ... \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} (x = x_i) \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i)$.

are **FPT in solution size** k for bounded tww.

Some **classes** of bounded twin-width:

- Bounded tree-width, rank-width, queue number...

Consequences

Some **problems** expressible in FO:

- k-IS : $\exists x_1 ... \exists x_k \bigwedge_{1 \leqslant i \leqslant j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$,
- k-DS : $\exists x_1 ... \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} (x = x_i) \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i)$.

are **FPT in solution size** k for bounded tww.

Some **classes** of bounded twin-width:

- Bounded tree-width, rank-width, queue number...
- Proper minor closed → planar graphs.

Consequences

Some **problems** expressible in FO:

- k-IS : $\exists x_1 ... \exists x_k \bigwedge_{1 \leqslant i \leqslant j \leqslant k} \neg(x_i = x_j \vee E(x_i, x_j))$,
- k-DS : $\exists x_1 ... \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} (x = x_i) \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i)$.

are **FPT in solution size** k for bounded tww.

Some **classes** of bounded twin-width:

- Bounded tree-width, rank-width, queue number...
- Proper minor closed → planar graphs.

↪ very broad !

Twin-width and polykernels

On classes of bounded twin-width:

Twin-width and polykernels

On classes of bounded twin-width:

- Connected k-VC admits a $O(k^2)$ kernel,

Twin-width and polykernels

On classes of bounded twin-width:

- Connected k-VC admits a $O(k^2)$ kernel,
- k-Independent Set admits no polykernel.

Twin-width and polykernels

On classes of bounded twin-width:

- Connected k-VC admits a $O(k^2)$ kernel,
- k-Independent Set admits no polykernel.

What about dominating set ?

Twin-width and polykernels

On classes of bounded twin-width:

- Connected k-VC admits a $O(k^2)$ kernel,
- k-Independent Set admits no polykernel.

What about dominating set ?

k-Dominating Set	**Parameter:** k
Input: Graph G, integer k	
Question: Does G have a dominating set of size at most k?	

Twin-width and polykernels

On classes of bounded twin-width:

- Connected k-VC admits a $O(k^2)$ kernel,
- k-Independent Set admits no polykernel.

What about dominating set ?

k-DOMINATING SET	**Parameter:** k
Input: Graph G, integer k	
Question: Does G have a dominating set of size at most k?	

Admits polykernels on sparse graphs... **but not on bounded twin-width.**

OR-composition for k-DS

Recall the tool to prove no polykernels:

OR-composition for k-DS

Recall the tool to prove no polykernels:

Theorem (Bodlaender, Jansen, Kratsch '12)
If an NP-hard problem $\mathscr{L}$ admits an OR-composition into a parameterized problem $\mathscr{Q}$, then $\mathscr{Q}$ does not admit a polynomial kernel unless coNP $\subseteq$ NP/*poly.*

OR-composition for k-DS

Recall the tool to prove no polykernels:

Theorem (Bodlaender, Jansen, Kratsch '12)
If an NP-hard problem $\mathscr{L}$ admits an OR-composition into a parameterized problem $\mathscr{Q}$, then $\mathscr{Q}$ does not admit a polynomial kernel unless coNP $\subseteq$ NP/*poly.*

Goal: rule out polynomial kernels for k-DS on classes of bounded twin-width.

OR-composition for k-DS

Recall the tool to prove no polykernels:

> **Theorem (Bodlaender, Jansen, Kratsch '12)**
> *If an NP-hard problem $\mathscr{L}$ admits an OR-composition into a parameterized problem $\mathscr{Q}$, then $\mathscr{Q}$ does not admit a polynomial kernel unless* coNP $\subseteq$ NP/*poly.*

Goal: rule out polynomial kernels for k-DS on classes of bounded twin-width.

- $\mathscr{Q}$: k-DS on a class of bounded twin-width ($\leqslant 4$).

OR-composition for k-DS

Recall the tool to prove no polykernels:

> **Theorem (Bodlaender, Jansen, Kratsch '12)**
> *If an NP-hard problem $\mathscr{L}$ admits an OR-composition into a parameterized problem $\mathscr{Q}$, then $\mathscr{Q}$ does not admit a polynomial kernel unless* coNP $\subseteq$ NP/*poly.*

Goal: rule out polynomial kernels for k-DS on classes of bounded twin-width.

- $\mathscr{Q}$: k-DS on a class of bounded twin-width ($\leqslant 4$).
- $\mathscr{L}$: NP-hard problem tailored to be "easily" composable into $\mathscr{Q}$. A version of k-DS on graphs of bounded twin-width.

Tailored k-DS on grid-like graphs

NP-hard $\mathscr{L}$: k-DS on instances $(G, k, \mathscr{B} = \{B_1, \dots, B_k\})$ s.t:

- $\mathscr{B}$ partitions $V(G)$, G has 4-sequence $\rightarrow G/\mathscr{B}$,
- $G/\mathscr{B}$ is a subgraph of a grid,

Tailored k-DS on grid-like graphs

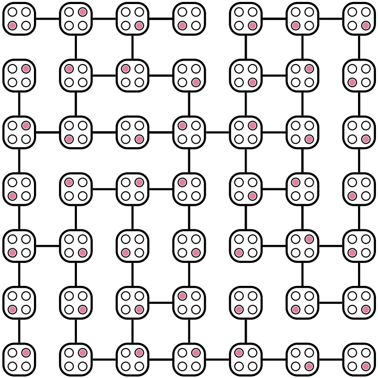
NP-hard $\mathscr{L}$: k-DS on instances $(G, k, \mathscr{B} = \{B_1, \ldots, B_k\})$ s.t:

- $\mathscr{B}$ partitions $V(G)$, G has 4-sequence $\rightarrow G/\mathscr{B}$,
- $G/\mathscr{B}$ is a subgraph of a grid,
- every dominating set of G intersects each B_i.

Tailored k-DS on grid-like graphs

NP-hard $\mathscr{L}$: k-DS on instances $(G, k, \mathscr{B} = \{B_1, \ldots, B_k\})$ s.t:

- $\mathscr{B}$ partitions $V(G)$, G has 4-sequence $\to G/\mathscr{B}$,
- $G/\mathscr{B}$ is a subgraph of a grid,
- every dominating set of G intersects each B_i.



OR-composition

Tailored k-**DS**, instances $(I_i)_{i \in [t]} \rightarrow k$-**DS**, $tww \leqslant 4$ instance (H, k)

OR-composition

Tailored k-**DS**, instances $(I_i)_{i \in [t]} \to k$-**DS**, *tww* $\leqslant 4$ instance (H, k)

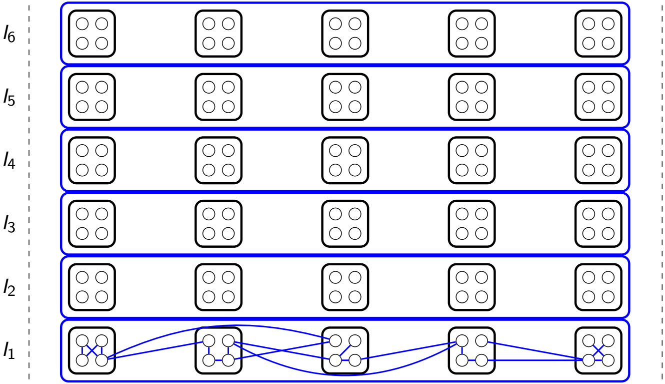


Fig. Instances ↔ layers, partition classes ↔ boxes.

OR-composition

Tailored k-**DS**, instances $(I_i)_{i\in[t]} \to k$-**DS**, *tww* $\leqslant 4$ instance (H, k)

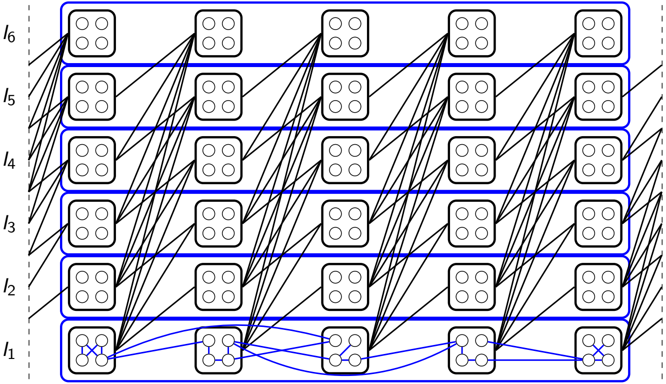


Fig. Instances ↔ layers, partition classes ↔ boxes.

In H**:** Top instance ↔ dummy, half graphs cycle between classes

Positive Tailored instance ⇒ Positive composition

Assume one instance (I_4 here) is positive, pick its solution in H:

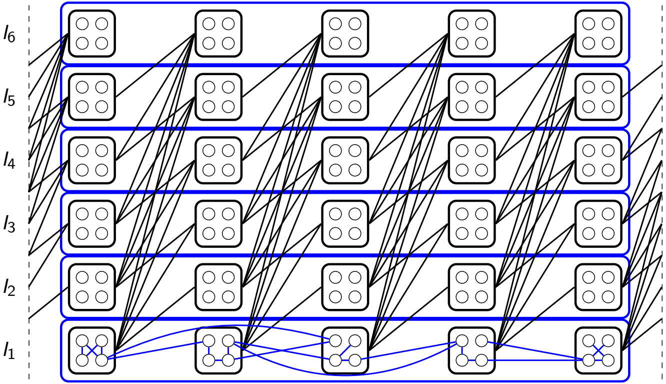


Fig. t Instances ↔ layers, k partition classes ↔ boxes

Positive Tailored instance ⇒ Positive composition

Assume one instance (I_4 here) is positive, pick its solution in H:

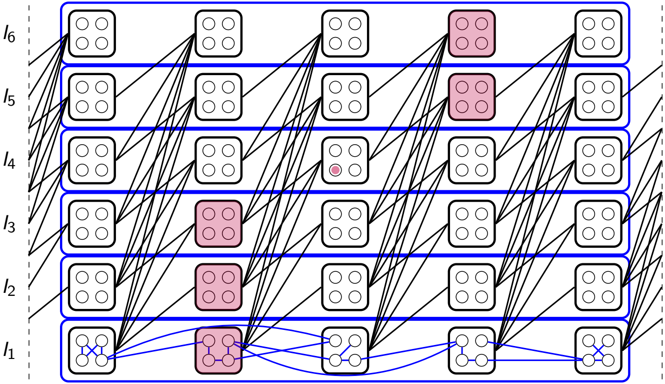


Fig. t Instances ↔ layers, k partition classes ↔ boxes

Positive Tailored instance ⇒ Positive composition

Assume one instance (I_4 here) is positive, pick its solution in H:

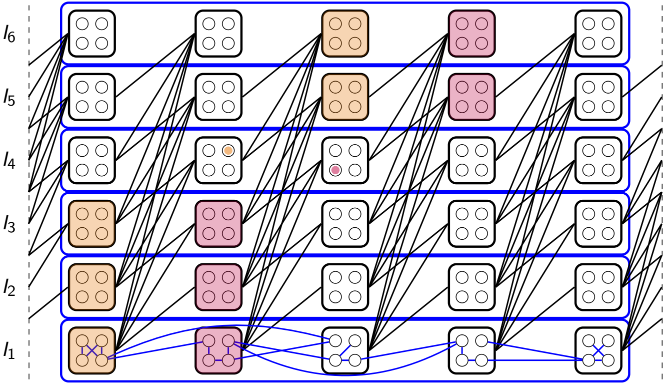


Fig. t Instances ↔ layers, k partition classes ↔ boxes

Positive Tailored instance ⇒ Positive composition

Assume one instance (I_4 here) is positive, pick its solution in H:

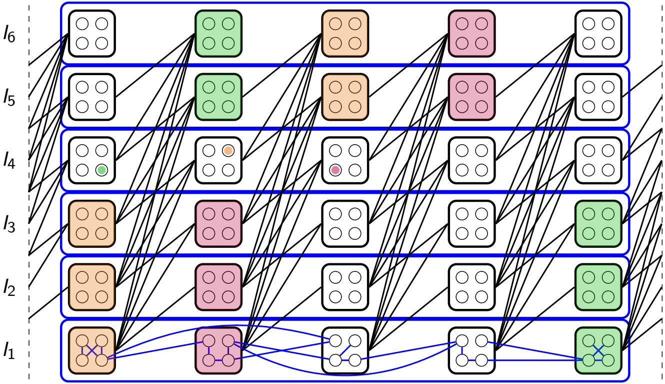


Fig. t Instances ↔ layers, k partition classes ↔ boxes

Positive Tailored instance ⇒ Positive composition

Assume one instance (I_4 here) is positive, pick its solution in H:

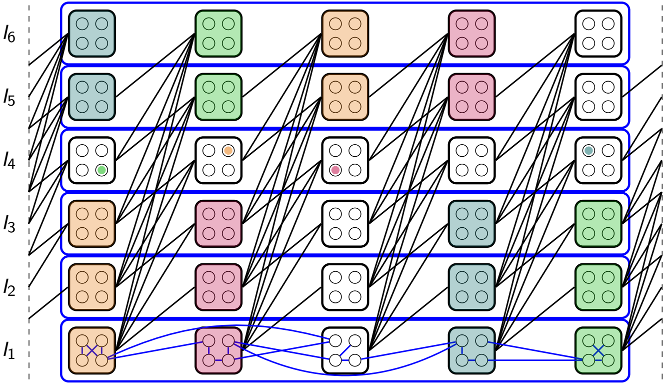


Fig. t Instances ↔ layers, k partition classes ↔ boxes

Positive Tailored instance ⇒ Positive composition

Assume one instance (I_4 here) is positive, pick its solution in H:

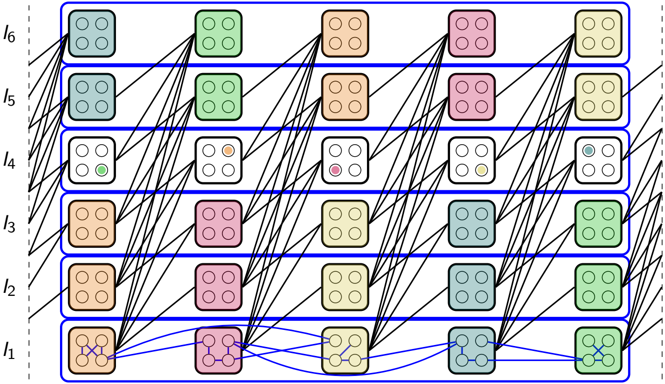
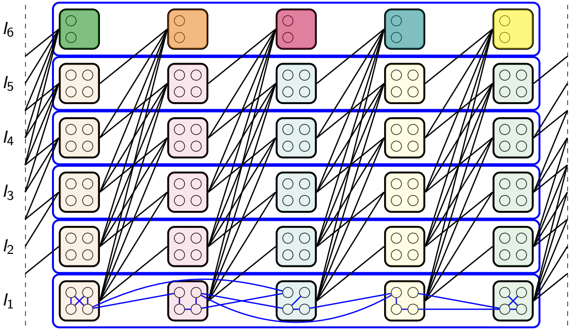


Fig. t Instances ↔ layers, k partition classes ↔ boxes

Positive composition ⇒ Positive tailored instance

Assume (H, k) is positive:

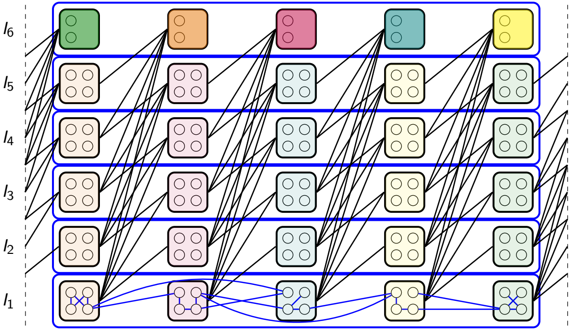
- Dummy instance I_6 forces one vertex per column,



Positive composition ⇒ Positive tailored instance

Assume (H, k) is positive:

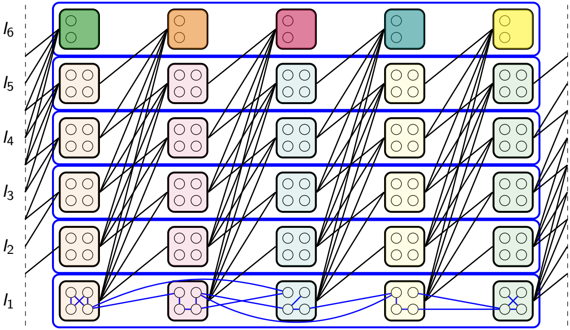
- Dummy instance I_6 forces one vertex per column,
- Ideally : all choices on a single layer i → positive I_i,



Positive composition ⇒ Positive tailored instance

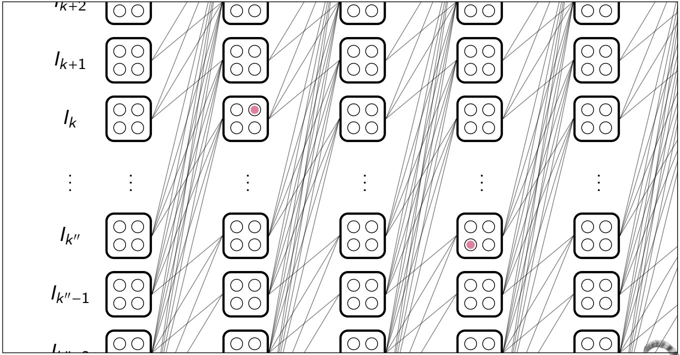
Assume (H, k) is positive:

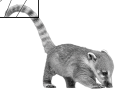
- Dummy instance I_6 forces one vertex per column,
- Ideally : all choices on a single layer i → positive I_i,
- Assume not...



Positive composition ⇒ Positive tailored instance

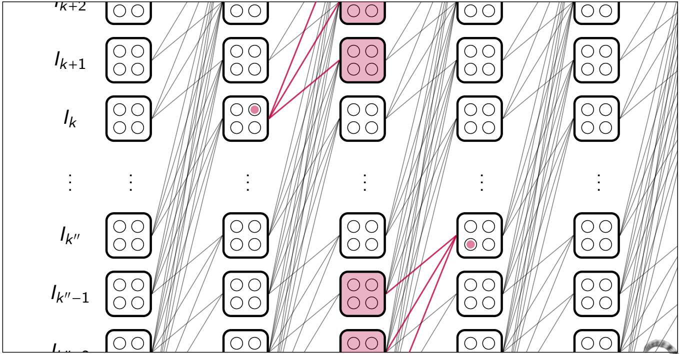
- Non-identical layer choices → domination "gap" on C_j,





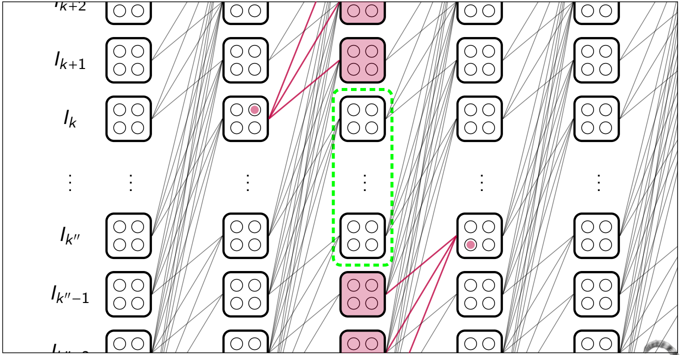
Positive composition ⇒ Positive tailored instance

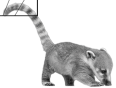
- Non-identical layer choices → domination "gap" on C_j,



Positive composition ⇒ Positive tailored instance

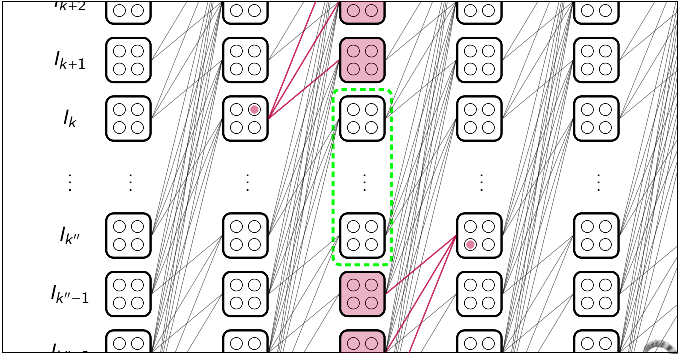
- Non-identical layer choices → domination "gap" on C_j,
- At least two classes not dominated by C_{j-1}, C_{j+1},

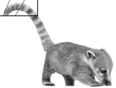




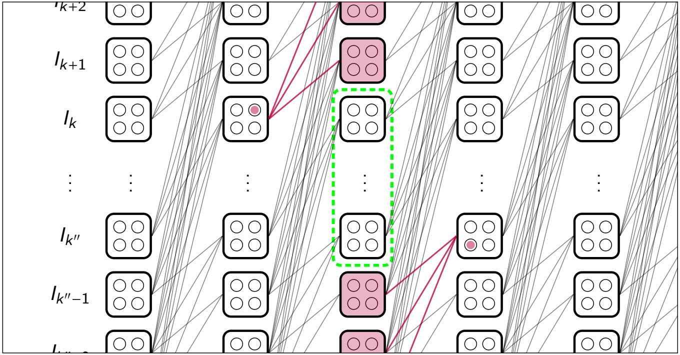
Positive composition ⇒ Positive tailored instance

- Non-identical layer choices → domination "gap" on C_j,
- At least two classes not dominated by C_{j-1}, C_{j+1},
- Only one choice left in C_j → absurd.





Positive composition ⇒ Positive tailored instance



Solution lies on a single layer → corresponding instance is dominated.

Bounding tww: contracting partition classes

- For any I_i: each partition class has same neighbours in $H - I_i$,

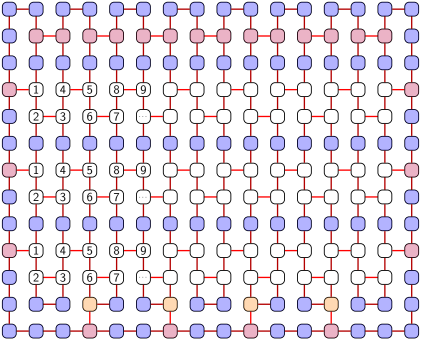


Fig. An instance after contracting classes.

Bounding tww: contracting partition classes

- For any I_i: each partition class has same neighbours in $H - I_i$,
- Contract each class → red edges created only in I_i

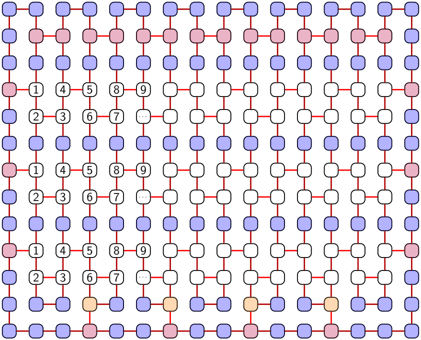


Fig. An instance after contracting classes.

Bounding tww: contracting partition classes

- For any I_i: each partition class has same neighbours in $H - I_i$,
- Contract each class → red edges created only in I_i
- Result: red subgraph of grid (altered for *tww* 4).

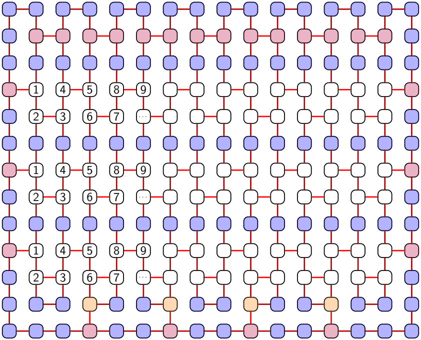


Fig. An instance after contracting classes.

Bounding tww: contracting instances together

- Between instances: half-graph cycle along vertices at the same position.

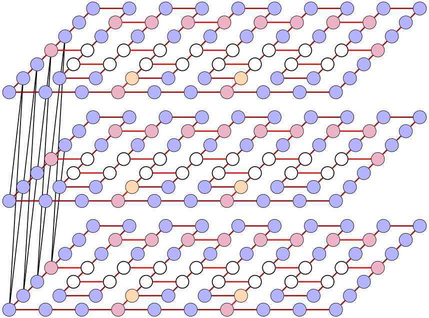


Fig. Instances and the half-graph cycle

Bounding tww: contracting instances together

- Between instances: half-graph cycle along vertices at the same position.
- Goal: Successively contract the bottommost two instances.

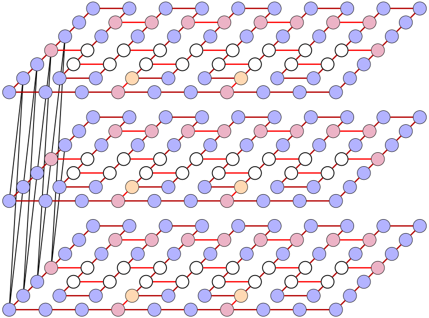


Fig. Instances and the half-graph cycle

Bounding tww: contracting instances together

- Between instances: half-graph cycle along vertices at the same position.
- Goal: Successively contract the bottommost two instances.
- Contracting two vertices at the **same position** creates red edges only **locally** → induction

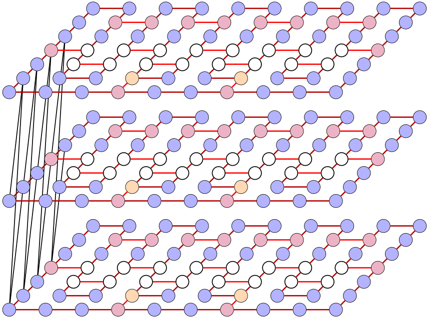


Fig. Instances and the half-graph cycle

Concluding

- NP-hard Tailored k-DS OR-composes into k-DS,

[5] even if a 4-sequence of the graph is given

Concluding

- NP-hard Tailored k-DS OR-composes into k-DS,
- The composed instance has tww at most four.

[5] even if a 4-sequence of the graph is given

Concluding

- NP-hard Tailored *k*-DS OR-composes into *k*-DS,
- The composed instance has tww at most four.

Theorem (Bonnet, Kim, R., Thomassé, Watrigant '21)
Unless coNP $\subseteq$ NP/*poly, k*-DOMINATING SET *on graphs of twin-width at most 4 does not admit a polynomial kernel*[5]*.*

[5] even if a 4-sequence of the graph is given

Further Directions

- Does k-DS on tww $\leqslant 3$ admit polynomial kernels?

Further Directions

- Does k-DS on tww $\leqslant 3$ admit polynomial kernels?
- Is there a linear kernel for Connected-VC?

Further Directions

- Does k-DS on tww $\leqslant 3$ admit polynomial kernels?
- Is there a linear kernel for Connected-VC?

Thank you!

Bibliography

É. Bonnet, E. J. Kim, S. Thomassé, and R. Watrigant. "Twin-width I: tractable FO model checking". In: (2020). arXiv: 2004.14789 [cs.DS]

H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. "Kernelization Lower Bounds By Cross-Composition". In: (2012). arXiv: 1206.5941 [cs.CC]

G. Philip, V. Raman, and S. Sikdar. "Polynomial Kernels for Dominating Set in Graphs of Bounded Degeneracy and Beyond". In: *ACM Trans. Algorithms* 9.1 (Dec. 2012). ISSN: 1549-6325. DOI: 10.1145/2390176.2390187. URL: https://doi.org/10.1145/2390176.2390187

É. Bonnet, E. J. Kim, A. Reinald, S. Thomassé, and R. Watrigant. "Twin-width and polynomial kernels". In: (2021). arXiv: 2107.02882 [cs.DS]

A (sub)quadratic kernel for C-VC on VC-density 1

Lemma (Bonnet, Kim, R., Thomassé, Watrigant '21)
There exists f s.t. for any G of tww(d) and $X \subseteq V(G)$: number of distinct neighborhoods in X is at most $f(d)|X|$.

A (sub)quadratic kernel for C-VC on VC-density 1

Lemma (Bonnet, Kim, R., Thomassé, Watrigant '21)
There exists f s.t. for any G of tww(d) and $X \subseteq V(G)$: number of distinct neighborhoods in X is at most $f(d)|X|$.

Kernel pre-processing:

- Bounded tww G: take X 2-approx for VC,

A (sub)quadratic kernel for C-VC on VC-density 1

Lemma (Bonnet, Kim, R., Thomassé, Watrigant '21)
There exists f s.t. for any G of tww(d) and $X \subseteq V(G)$: number of distinct neighborhoods in X is at most $f(d)|X|$.

Kernel pre-processing:

- Bounded tww G: take X 2-approx for VC,
- if $|X| \geqslant 2k+1$: no solution,

A (sub)quadratic kernel for C-VC on VC-density 1

> **Lemma (Bonnet, Kim, R., Thomassé, Watrigant '21)**
> *There exists f s.t. for any G of tww(d) and $X \subseteq V(G)$: number of distinct neighborhoods in X is at most $f(d)|X|$.*

Kernel pre-processing:

- Bounded tww G: take X 2-approx for VC,
- if $|X| \geqslant 2k+1$: no solution,
- $\rightarrow$ in X, number of distinct neighbourhoods $\leqslant f(d)k$

A (sub)quadratic kernel for C-VC on VC-density 1

Reduction rule: If there is $S \subseteq V(G) \setminus X$ with identical neighbourhood in X and $|S| > k$, delete a vertex of S.

A (sub)quadratic kernel for C-VC on VC-density 1

Reduction rule: If there is $S \subseteq V(G) \setminus X$ with identical neighbourhood in X and $|S| > k$, delete a vertex of S.

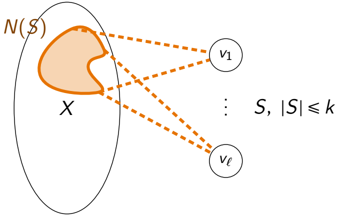


Fig. Resulting instance: (G', k)

A (sub)quadratic kernel for C-VC on VC-density 1

Reduction rule: If there is $S \subseteq V(G) \setminus X$ with identical neighbourhood in X and $|S| > k$, delete a vertex of S.

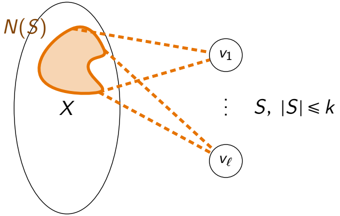


Fig. Resulting instance: (G', k)

If (G, k) has a solution, replace deletion with twin → reconnects T

A (sub)quadratic kernel for C-VC on VC-density 1

Reduction rule: If there is $S \subseteq V(G) \setminus X$ with identical neighbourhood in X and $|S| > k$, delete a vertex of S.

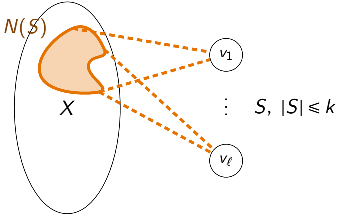


Fig. Resulting instance: (G', k)

If (G,k) has a solution, replace deletion with twin → reconnects T
If (G',k) has a solution T, T is also a solution for (G,k),

A (sub)quadratic kernel for C-VC on VC-density 1

Reduction rule: If there is $S \subseteq V(G) \setminus X$ with identical neighbourhood in X and $|S| > k$, delete a vertex of S.

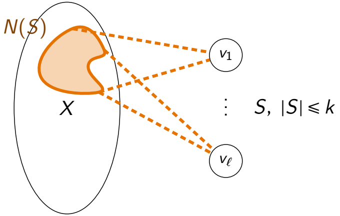


Fig. Resulting instance: (G', k)

If (G, k) has a solution, replace deletion with twin → reconnects T

If (G', k) has a solution T, T is also a solution for (G, k),

- Take deleted s, $S \setminus s \not\subseteq T$ as ind. set with $|S| \geqslant k$,
- Therefore $N(S) \subseteq T$, and s is covered by T in G.

Concluding

- Applying the reduction yields an equivalent instance,

Concluding

- Applying the reduction yields an equivalent instance,
- The kernel has size at most $f(d) * k^2$,

Concluding

- Applying the reduction yields an equivalent instance,
- The kernel has size at most $f(d) * k^2$,

Concluding

- Applying the reduction yields an equivalent instance,
- The kernel has size at most $f(d) * k^2$,

Theorem (Bonnet, Kim, R., Thomassé, Watrigant '21)
On classes of VC-density 1, CONNECTED k-VERTEX COVER *admits kernels with $O(k^2)$ vertices (and even $O(k^{1.5})$).*