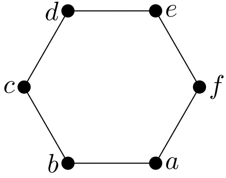
Locally checkable problems parameterized by mim-width

Carolina Lucía Gonzalez

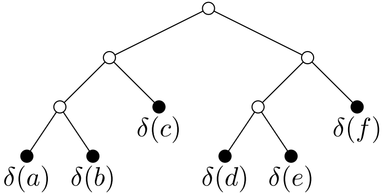
November 11, 2024

mim-width

For a graph G, a rooted binary tree T together with a bijection δ between $V(G)$ and the leafs of T is called a **binary decomposition tree** of G.



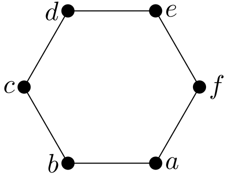
A graph G



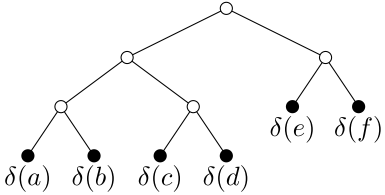
A binary decomposition tree (T, δ) of G

mim-width

For a graph G, a rooted binary tree T together with a bijection δ between $V(G)$ and the leafs of T is called a **binary decomposition tree** of G.



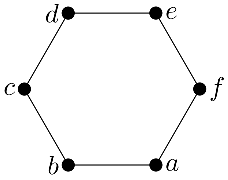
A graph G



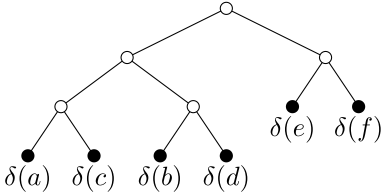
A binary decomposition tree (T, δ) of G

mim-width

For a graph G, a rooted binary tree T together with a bijection δ between $V(G)$ and the leafs of T is called a **binary decomposition tree** of G.



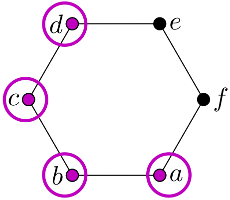
A graph G



A binary decomposition tree (T, δ) of G

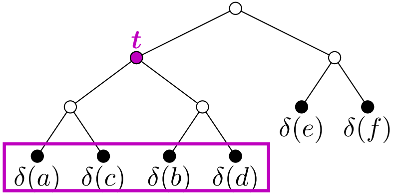
mim-width

For a node t in T, we denote by $\boldsymbol{T_t}$ the set of vertices of G which correspond to the leaves which are descendants of t.



A graph G

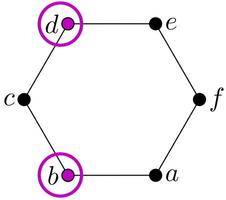
$$\boldsymbol{T_t = \{a, b, c, d\}}$$



A binary decomposition tree (T, δ) of G

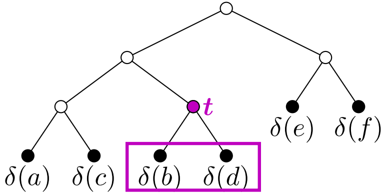
mim-width

For a node t in T, we denote by $\boldsymbol{T_t}$ the set of vertices of G which correspond to the leaves which are descendants of t.



A graph G

$\boldsymbol{T_t = \{b, d\}}$

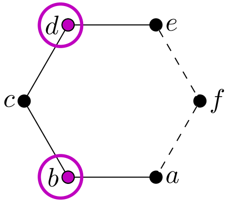


A binary decomposition tree (T, δ) of G

mim-width

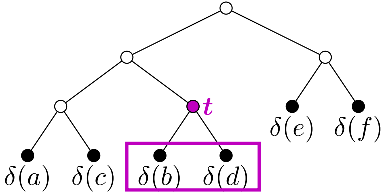
For a node t in T, we denote by $\boldsymbol{T_t}$ the set of vertices of G which correspond to the leaves which are descendants of t.

Let $\boldsymbol{mim(G, T_t)}$ denote the maximum size (measured as number of edges) of an induced matching in the bipartite graph $G[T_t, V(G) \setminus T_t]$.



A graph G

$$\boldsymbol{T_t = \{b, d\}}$$

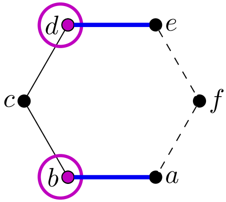


A binary decomposition tree (T, δ) of G

mim-width

For a node t in T, we denote by $\boldsymbol{T_t}$ the set of vertices of G which correspond to the leaves which are descendants of t.

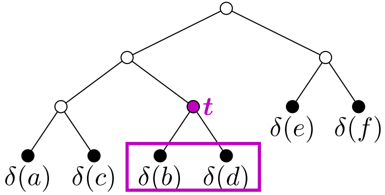
Let $\boldsymbol{mim(G, T_t)}$ denote the maximum size (measured as number of edges) of an induced matching in the bipartite graph $G[T_t, V(G) \setminus T_t]$.



A graph G

$\boldsymbol{T_t = \{b, d\}}$

$\boldsymbol{mim(G, T_t) = 2}$



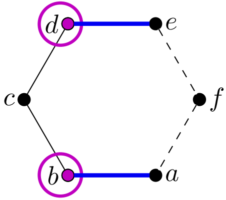
A binary decomposition tree (T, δ) of G

mim-width

For a node t in T, we denote by $\boldsymbol{T_t}$ the set of vertices of G which correspond to the leaves which are descendants of t.

Let $\boldsymbol{mim(G, T_t)}$ denote the maximum size (measured as number of edges) of an induced matching in the bipartite graph $G[T_t, V(G) \setminus T_t]$.

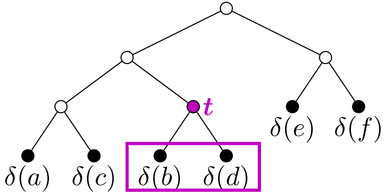
The **mim-width of a decomposition** is $\boldsymbol{mimw(T, \delta)} = \max_{t \in V(T)} \{mim(G, T_t)\}$.



A graph G

$\boldsymbol{T_t = \{b, d\}}$

$\boldsymbol{mim(G, T_t) = 2}$



A binary decomposition tree (T, δ) of G

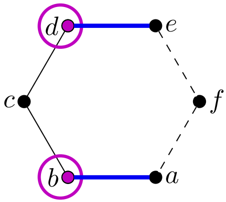
mim-width

For a node t in T, we denote by $\boldsymbol{T_t}$ the set of vertices of G which correspond to the leaves which are descendants of t.

Let $\boldsymbol{mim(G, T_t)}$ denote the maximum size (measured as number of edges) of an induced matching in the bipartite graph $G[T_t, V(G) \setminus T_t]$.

The **mim-width of a decomposition** is $\boldsymbol{mimw(T, \delta)} = \max_{t \in V(T)} \{mim(G, T_t)\}$.

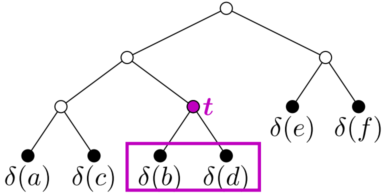
The **mim-width of a graph** G, $\boldsymbol{mimw(G)}$, is the minimum value of $mimw(T, \delta)$ over all binary decomposition trees (T, δ) of G.



A graph G

$\boldsymbol{T_t = \{b, d\}}$

$\boldsymbol{mim(G, T_t) = 2}$

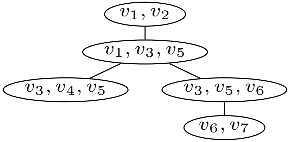


A binary decomposition tree (T, δ) of G

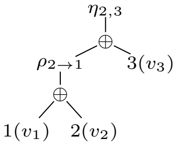
Width parameters

These parameters are associated with useful ways to describe the graph.

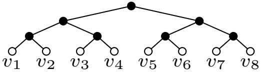
- treewidth: tree-decompositions



- clique-width: clique-width expressions

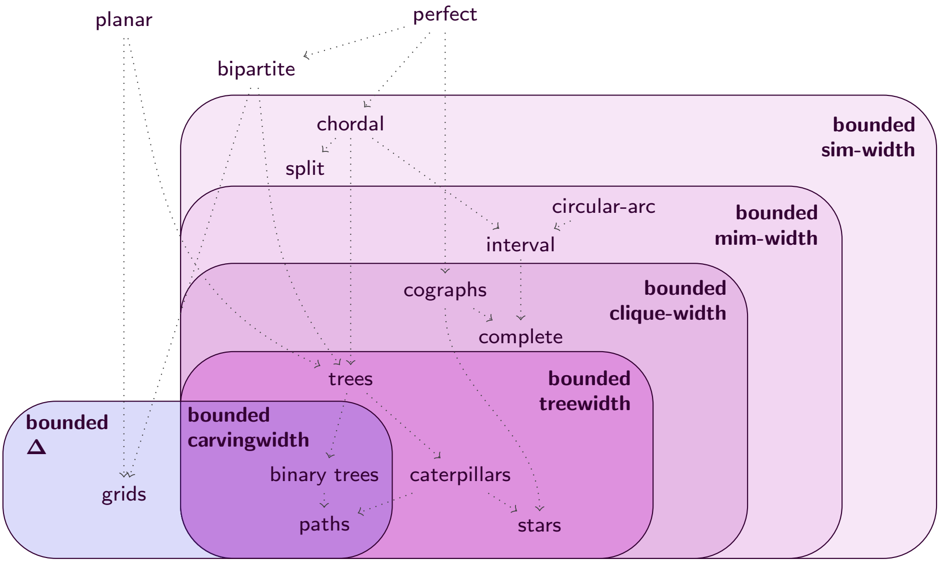


- mim-width: branch decompositions



$\to$ **Dynamic programming** gets along really well with them!

Width parameters



Parameterized complexity

In a **parameterized problem** the input is considered to be two-part: (x, k)

Parameterized complexity

main input

In a **parameterized problem** the input is considered to be two-part: $\underset{\uparrow}{(x}, k)$

Parameterized complexity

In a **parameterized problem** the input is considered to be two-part: $\left(\overset{\text{main input}}{\overset{\downarrow}{x}}, \underset{\textbf{parameter}}{\underset{\uparrow}{k}}\right)$

Parameterized complexity

main input

In a **parameterized problem** the input is considered to be two-part: $(\boldsymbol{x}, \boldsymbol{k})$

parameter

FPT (fixed parameter tractable): $f(k) \cdot |x|^{O(1)}$

Parameterized complexity

In a **parameterized problem** the input is considered to be two-part: $\boldsymbol{(\underset{\text{main input}}{\overset{\downarrow}{x}}, \underset{\uparrow}{k})}$

main input

parameter

FPT (fixed parameter tractable): $f(k) \cdot |x|^{O(1)}$

XP (slice-wise polynomial): $f(k) \cdot |x|^{g(k)}$

Parameterized complexity

main input

In a **parameterized problem** the input is considered to be two-part: $(\boldsymbol{x}, \boldsymbol{k})$

parameter

FPT (fixed parameter tractable): $f(k) \cdot |x|^{O(1)}$

XP (slice-wise polynomial): $f(k) \cdot |x|^{g(k)}$

para-NP-hard: NP-hard for a fixed k

Parameterized complexity

main input

In a **parameterized problem** the input is considered to be two-part: $(\boldsymbol{x}, \boldsymbol{k})$

parameter

FPT (fixed parameter tractable): $f(k) \cdot |x|^{O(1)}$

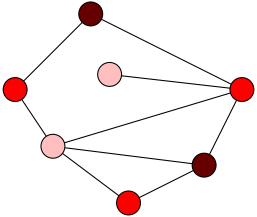
XP (slice-wise polynomial): $f(k) \cdot |x|^{g(k)}$

para-NP-hard: NP-hard for a fixed k

W[1]-hard: XP but *unlikely* to be FPT.

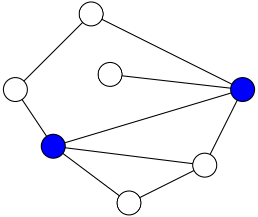
Introductory problems

k-coloring



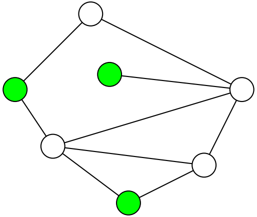
Does there exist a function $c \colon V \to \{1, \ldots, k\}$ such that $c(v) \neq c(u)\ \forall v \in V, u \in N(v)$?

Minimum Dominating Set



Minimum size of a set D such that $N[v] \cap D \neq \emptyset$ for all $v \in V$?

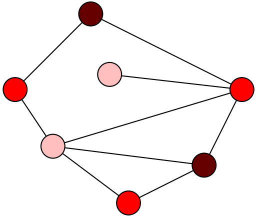
Maximum Independent Set



Maximum size of a set I such that $N(v) \cap I = \emptyset$ for all $v \in I$?

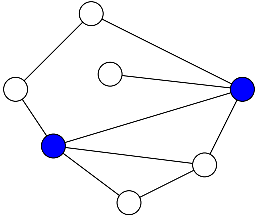
Introductory problems

k-coloring



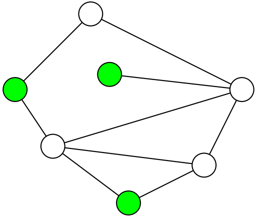
Does there exist a function $c\colon V \to \{1, \ldots, k\}$ such that $c(v) \neq c(u)\ \forall v \in V, u \in N(v)$?

Minimum Dominating Set



Minimum size of a set D such that $N[v] \cap D \neq \emptyset$ for all $v \in V$?

Maximum Independent Set

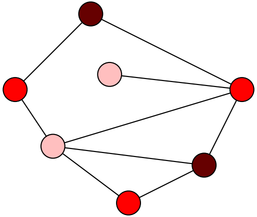


Maximum size of a set I such that $N(v) \cap I = \emptyset$ for all $v \in I$?

What do they have in common?

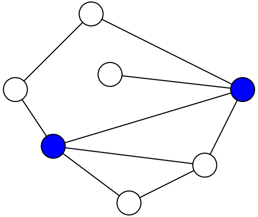
Introductory problems

k-coloring



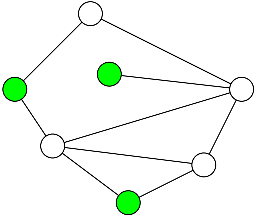
Does there exist a function $c\colon V \to \{1,\dots,k\}$ such that $c(v) \neq c(u)\ \forall v \in V, u \in N(v)$?

Minimum Dominating Set



Minimum size of a set D such that $N[v] \cap D \neq \emptyset$ for all $v \in V$?

Maximum Independent Set



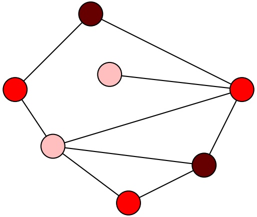
Maximum size of a set I such that $N(v) \cap I = \emptyset$ for all $v \in I$?

What do they have in common?

They are *partitioning (coloring) problems*

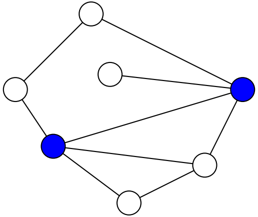
Introductory problems

k-coloring



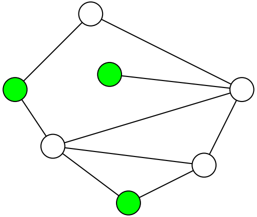
Does there exist a function $c\colon V \to \{1, \dots, k\}$ such that $c(v) \neq c(u)\ \forall v \in V, u \in N(v)$?

Minimum Dominating Set



Minimum size of a set D such that $N[v] \cap D \neq \emptyset$ for all $v \in V$?

Maximum Independent Set



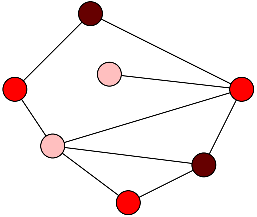
Maximum size of a set I such that $N(v) \cap I = \emptyset$ for all $v \in I$?

What do they have in common?

They are *partitioning (coloring) problems* where each vertex must satisfy a *local property*.

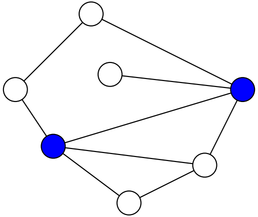
Introductory problems

k-coloring



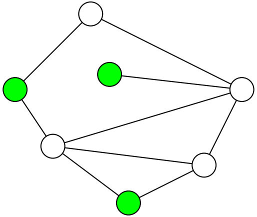
Does there exist a function $c\colon V \to \{1, \ldots, k\}$ such that $c(v) \neq c(u)\ \forall v \in V, u \in N(v)$?

Minimum Dominating Set



Minimum size of a set D such that $N[v] \cap D \neq \emptyset$ for all $v \in V$?

Maximum Independent Set



Maximum size of a set I such that $N(v) \cap I = \emptyset$ for all $v \in I$?

What do they have in common?

They are *partitioning (coloring) problems* where each vertex must satisfy a *local property*.

$\rightarrow$ **Locally checkable problems**

Why do we study locally checkable problems?

- Many well-known graph problems are locally checkable, with real-life applications:
  - Assigning resources (classrooms, processor registers, frequencies)
  - Placing facilities (recycling points, information centers, vending machines)

Why do we study locally checkable problems?

- Many well-known graph problems are locally checkable, with real-life applications:
  - Assigning resources (classrooms, processor registers, frequencies)
  - Placing facilities (recycling points, information centers, vending machines)
- Can we solve these problems efficiently?
  - For which graph classes?
  - How?
  - Can we generalize the ideas?

Related work

- Locally Checkable Vertex Partitioning (LCVP) problems
  [J.A. Telle, 1994]
- Local Condition Composition (LCC) and Edge Condition Composition (ECC) problems
  [H.L. Bodlaender, 1987]
- Monadic Second Order Logic (Courcelle's theorem)
  [B. Courcelle, 1990]
- A&C DN logic
  [B. Bergougnoux, J. Dreier and L. Jaffke, 2022]

What this talk is about

We abstract the notion of locally checkable problems into a **new framework**.

What this talk is about

We abstract the notion of locally checkable problems into a **new framework**.

What makes these problems $easy$ **or** $hard$**?**

What this talk is about

We abstract the notion of locally checkable problems into a **new framework**.

What makes these problems $easy$ or $hard$?

$\rightarrow$ Classify them according to different restrictions on the local property and on the set of colors.

What this talk is about

We abstract the notion of locally checkable problems into a **new framework**.

What makes these problems $easy$ or $hard$?

$\rightarrow$ Classify them according to different restrictions on the local property and on the set of colors.

Study the **complexity** of locally checkable problems **in terms of different width parameters** (carvingwidth, treewidth, clique-width and mim-width).

What this talk is about

We abstract the notion of locally checkable problems into a **new framework**.

What makes these problems $easy$ or $hard$?

$\rightarrow$ Classify them according to different restrictions on the local property and on the set of colors.

Study the **complexity** of locally checkable problems **in terms of different width parameters** (carvingwidth, treewidth, clique-width and mim-width).

$\rightarrow$ develop efficient algorithms under a set of restrictions

$\rightarrow$ find hard problems that do not satisfy all these restrictions

The framework (simplified)

Given a simple undirected graph G and

The framework (simplified)

Given a simple undirected graph G and

- **Colors**: a set of colors,

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\boldsymbol{check(v, c)}$: a check function (input: $v \in V(G)$ and $c \colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c \colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c \colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

The framework (simplified)

Given a simple undirected graph G and

- $\textbf{C}\textsc{olors}$: a set of colors,
- $\boldsymbol{check(v, c)}$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{\boldsymbol{v,i}}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$\boldsymbol{check(v, c|_{N[v]}) = \textbf{T}\textsc{rue}} \quad \text{for all } v \in V(G).$$

The weight c is $\mathrm{w}(c) = \sum_{v \in V(G)} \mathrm{w}_{v,c(v)}$.

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $w_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

$$\textsc{Colors} = \{\mathrm{s}, \overline{\mathrm{s}}\}$$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\mathit{check}(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$\mathit{check}(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

$\textsc{Colors} = \{\mathbf{s}, \overline{\mathrm{s}}\}$

$\mathit{check}(v, c) = \textsc{True}$ if and only if there exists $u \in N[v]$ such that $c(u) = \mathbf{s}$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

$$\textsc{Colors} = \{\mathbf{s}, \overline{\mathrm{s}}\}$$

$$check(v, c) = (\exists u \in N[v].\, c(u) = \mathbf{s})$$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\boldsymbol{check(v,c)}$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$\boldsymbol{check(v, c|_{N[v]}) = \textsc{True}} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

$\textsc{Colors} = \{\mathbf{s}, \overline{\mathrm{s}}\}$

$check(v,c) = \textsc{True}$ if and only if there exists $u \in N[v]$ such that $c(u) = \mathbf{s}$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\boldsymbol{check(v,c)}$: a check function (input: $v \in V(G)$ and $c \colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{\mathbf{w}}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c \colon V(G) \to \textsc{Colors}$ such that

$$\boldsymbol{check(v, c|_{N[v]}) = \textsc{True}} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

$$\textsc{Colors} = \{\mathbf{s}, \overline{\mathrm{s}}\}$$

$check(v,c) = \textsc{True}$ if and only if there exists $u \in N[v]$ such that $c(u) = \mathbf{s}$

$$\mathrm{w}_{v,i} = \begin{cases} 1 & \text{if } i = \mathbf{s} \\ 0 & \text{otherwise} \end{cases}$$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c \colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c \colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Minimum Dominating Set:

$\textsc{Colors} = \{\mathbf{s}, \overline{\mathrm{s}}\}$

$check(v, c) = \textsc{True}$ if and only if there exists $u \in N[v]$ such that $c(u) = \mathbf{s}$

$$\mathrm{w}_{v,i} = \begin{cases} 1 & \text{if } i = \mathbf{s} \\ 0 & \text{otherwise} \end{cases}$$

Order of the weights: $\leq$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Maximum Independent Set:

$$\textsc{Colors} = \{\mathbf{s}, \overline{\mathrm{s}}\}$$

$$check(v, c) = (c(v) = \mathbf{s} \Rightarrow \forall u \in N(v).\, c(u) = \overline{\mathrm{s}})$$

$$\mathrm{w}_{v,i} = \begin{cases} 1 & \text{if } i = \mathbf{s} \\ 0 & \text{otherwise} \end{cases}$$

Order of the weights: $\geq$

The framework (simplified)

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\boldsymbol{check(v,c)}$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathbf{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$\boldsymbol{check(v, c|_{N[v]}) = \textsc{True}} \quad \text{for all } v \in V(G).$$

k-coloring:

$\textsc{Colors} = \{1, \ldots, k\}$

$check(v,c) = (\forall u \in N(v).\, c(u) \neq c(v))$

$\mathrm{w}_{v,i} = 0$

Order of the weights: $\leq$

Global properties

A locally checkable problem with **global properties** $\Pi = \{P_1, \ldots, P_k\}$ is a locally checkable problem where it is additionally required that the assignment of colors to vertices c given as an answer satisfies each of the properties in Π, i.e., that $P_i(c) = \textsc{True}$ for each $i \in \{1, \ldots, k\}$.

Global properties

A locally checkable problem with **global properties** $\Pi = \{P_1, \ldots, P_k\}$ is a locally checkable problem where it is additionally required that the assignment of colors to vertices c given as an answer satisfies each of the properties in Π, i.e., that $P_i(c) = \textsc{True}$ for each $i \in \{1, \ldots, k\}$.

For example:

- "the number of vertices of color red or blue is at most 10"
- "there is an odd number of green vertices"

- "the subgraph induced by the set of red vertices is connected"
- "the subgraph induced by the set of blue and green vertices is not connected"

- "the subgraph induced by the set of green vertices is acyclic (i.e. is a forest)"
- "the subgraph induced by the set of blue vertices is not acyclic (i.e. has a cycle)"

Global properties

A locally checkable problem with **global properties** $\Pi = \{P_1, \ldots, P_k\}$ is a locally checkable problem where it is additionally required that the assignment of colors to vertices c given as an answer satisfies each of the properties in Π, i.e., that $P_i(c) = \textsc{True}$ for each $i \in \{1, \ldots, k\}$.

For example:

- "the number of vertices of color red or blue is at most 10"
- "there is an odd number of green vertices"

  $\longrightarrow$ **Size properties**
- "the subgraph induced by the set of red vertices is connected"
- "the subgraph induced by the set of blue and green vertices is not connected"

  $\longrightarrow$ **Connectivity properties**
- "the subgraph induced by the set of green vertices is acyclic (i.e. is a forest)"
- "the subgraph induced by the set of blue vertices is not acyclic (i.e. has a cycle)"

  $\longrightarrow$ **Acyclicity properties**

Main goal

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

Main goal

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $check(v, c)$: a check function (input: $v \in V(G)$ and $c \colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c \colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

We want to obtain efficient algorithms when parameterized by different width measures
$\rightarrow$ carvingwidth, treewidth, clique-width and mim-width

Main goal

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\boldsymbol{check(v,c)}$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

We want to obtain efficient algorithms when parameterized by different width measures
$\rightarrow$ carvingwidth, treewidth, clique-width and mim-width

Under which conditions?

Main goal

Given a simple undirected graph G and

- $\textsc{Colors}$: a set of colors,
- $\boldsymbol{check(v, c)}$: a check function (input: $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$, output: true or false),
- $\mathrm{w}_{v,i}$: for every $v \in V(G)$ and $i \in \textsc{Colors}$, a weight,
- an order of the weights,

find the minimum weight of an assignment of colors to vertices $c\colon V(G) \to \textsc{Colors}$ such that

$$check(v, c|_{N[v]}) = \textsc{True} \quad \text{for all } v \in V(G).$$

We want to obtain efficient algorithms when parameterized by different width measures
$\rightarrow$ carvingwidth, treewidth, clique-width and mim-width

Under which conditions?

Which global properties can we include?

Color-counting local property

We say that $check$ is **color-counting** if it only depends on:

Color-counting local property

We say that $check$ is **color-counting** if it only depends on:

- the vertex v,

Color-counting local property

We say that $check$ is **color-counting** if it only depends on:

- the vertex v,
- the color that v receives, and

Color-counting local property

We say that $check$ is **color-counting** if it only depends on:

- the vertex v,
- the color that v receives, and
- the number of neighbors of v of each color.

Color-counting local property

We say that $check$ is **color-counting** if it only depends on:

- the vertex v,
- the color that v receives, and
- the number of neighbors of v of each color.

In other words, if there exists a function $check'$ such that, for all $v \in V(G)$ and $c\colon N[v] \to \textsc{Colors}$:

$$\boldsymbol{check(v, c) = check'(v, c(v), \mu_1, \ldots, \mu_q)}$$

where $\mu_i = \{u \in N(v) : c(u) = a_i\}$ for every color $a_i \in \textsc{Colors}$.

Color-counting local property

We say that $check$ is **color-counting** if it only depends on:

- the vertex v,
- the color that v receives, and
- the number of neighbors of v of each color.

In other words, if there exists a function $check'$ such that, for all $v \in V(G)$ and $c \colon N[v] \to \textsc{Colors}$:

$$\boldsymbol{check(v, c) = check'(v, c(v), \mu_1, \ldots, \mu_q)}$$

where $\mu_i = \{u \in N(v) : c(u) = a_i\}$ for every color $a_i \in \textsc{Colors}$.

Example: Minimum Dominating Set

- $check'(v, i, \mu_{\mathbf{S}}, \mu_{\overline{\mathrm{S}}}) = (i = \mathbf{S} \vee \mu_{\mathbf{S}} \geq 1)$

d-stable local property

Let d be a positive integer.

We say that $check$ is **d-stable** if it only depends on:

- the vertex v,
- the color that v receives, and
- the number of neighbors of v of each color **up to d**.

In other words, if $check$ is color-counting and:

$$check'(v, c(v), \mu_1, \ldots, \mu_q) = check'(v, c(v), \min(d, \mu_1), \ldots, \min(d, \mu_q))$$

d-stable local property

Let d be a positive integer.

We say that $check$ is **d-stable** if it only depends on:

- the vertex v,
- the color that v receives, and
- the number of neighbors of v of each color **up to d**.

In other words, if $check$ is color-counting and:

$$\boldsymbol{check'(v, c(v), \mu_1, \ldots, \mu_q) = check'(v, c(v), \min(d, \mu_1), \ldots, \min(d, \mu_q))}$$

Examples:

- Minimum Dominating Set (1-stable)

d-stable local property

Let d be a positive integer.

We say that $check$ is **d-stable** if it only depends on:

- the vertex v,
- the color that v receives, and
- the number of neighbors of v of each color **up to d**.

In other words, if $check$ is color-counting and:

$$\boldsymbol{check'(v, c(v), \mu_1, \ldots, \mu_q) = check'(v, c(v), \min(d, \mu_1), \ldots, \min(d, \mu_q))}$$

Examples:

- Minimum Dominating Set (1-stable)
- LCVP problems

Locally checkable problems parameterized by mim-width

These results were published in an article together with Felix Mann (2024).

Restrictions on $check$	Restrictions on $\lvert\textsc{Colors}\rvert$	
	Constant	Polynomial
d-stable	XP$^{(1)}$, W[1]-h	para-NP-h
Color-counting	para-NP-h	para-NP-h

(1) given a binary decomposition tree.

Locally checkable problems parameterized by mim-width

These results were published in an article together with Felix Mann (2024).

Restrictions on $check$	Restrictions on $\lvert\textsc{Colors}\rvert$	
	Constant	Polynomial
d-stable	$\boxed{\mathrm{XP}^{(1)}\text{, W[1]-h}}$	para-NP-h
Color-counting	para-NP-h	para-NP-h

(1) given a binary decomposition tree.

Dynamic programming algorithm using the given binary decomposition tree.

Complexity: $n^{O(wd\mathcal{C})}$

where $n = |V(G)|$, $w = mimw(G)$, $\mathcal{C} = |\textsc{Colors}|$.

Locally checkable problems parameterized by mim-width

These results were published in an article together with Felix Mann (2024).

Restrictions on $check$	Restrictions on $\lvert \textsc{Colors} \rvert$	
	Constant	Polynomial
d-stable	$\boxed{\text{XP}^{(1)}\text{, W[1]-h}}$	para-NP-h
Color-counting	para-NP-h	para-NP-h

(1) given a binary decomposition tree.

Dynamic programming algorithm using the given binary decomposition tree.

Complexity: $n^{O(wd\mathcal{C})}$
where $n = |V(G)|$, $w = mimw(G)$, $\mathcal{C} = |\textsc{Colors}|$.

We may add some global properties:

- Size
- Connectivity

Locally checkable problems parameterized by mim-width

These results were published in an article together with Felix Mann (2024).

Restrictions on $check$	Restrictions on $\lvert\textsc{Colors}\rvert$	
	Constant	Polynomial
d-stable	XP$^{(1)}$, W[1]-h	para-NP-h
Color-counting	para-NP-h	para-NP-h

(1) given a binary decomposition tree.

W[1]-hard example: Minimum Dominating Set

Locally checkable problems parameterized by mim-width

These results were published in an article together with Felix Mann (2024).

Restrictions on $check$	Restrictions on $|\textsc{Colors}|$	
	Constant	Polynomial
d-stable	XP$^{(1)}$, W[1]-h	para-NP-h
Color-counting	para-NP-h	para-NP-h

(1) given a binary decomposition tree.

Examples of para-NP-hard problems: Max-Cut, L(2,1)-labeling

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \ldots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \quad \text{for all } v \in V \text{ with } f(v) = 0?$$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \ldots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \text{ for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \ldots, k\}$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \ldots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \text{ for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \ldots, k\}$
- $check(v, c) = \left(c(v) = 0 \Rightarrow \sum_{u \in N(v)} c(u) \geq k \right)$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v\in V} f(v)$ of a function $f\colon V \to \{0,\ldots,k\}$ such that

$$\sum_{u\in N(v)} f(u) \geq k \text{ for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0,\ldots,k\}$
- $check(v,c) = \left(c(v) = 0 \Rightarrow \sum_{u\in N(v)} c(u) \geq k\right)$
- $\mathrm{w}_{v,i} = i$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \dots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \text{ for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \dots, k\}$
- $check(v, c) = \left(c(v) = 0 \Rightarrow \sum_{u \in N(v)} c(u) \geq k \right)$
- $\mathrm{w}_{v,i} = i$
- Order of weights: $\leq$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f \colon V \to \{0, \dots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \text{ for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \dots, k\}$

  $\to$ $k + 1$ colors

- $check(v, c) = \left(c(v) = 0 \Rightarrow \sum_{u \in N(v)} c(u) \geq k \right)$

- $\mathrm{w}_{v,i} = i$

- Order of weights: $\leq$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \ldots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \text{ for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \ldots, k\}$

  $\rightarrow$ $\boldsymbol{k+1}$ **colors**

- $check(v, c) = \left(c(v) = 0 \Rightarrow \sum_{u \in N(v)} c(u) \geq k \right)$

  $\rightarrow$ **color-counting**

- $\mathrm{w}_{v,i} = i$
- Order of weights: $\leq$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v\in V} f(v)$ of a function $f\colon V \to \{0,\ldots,k\}$ such that

$$\sum_{u\in N(v)} f(u) \geq k \quad \text{for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0,\ldots,k\}$

  $\rightarrow$ $k+1$ colors

- $check'(v,i,\mu_0,\ldots,\mu_k) = \left(i = 0 \Rightarrow \sum_{j=0}^{k} j\cdot\mu_j \geq k\right)$

  $\rightarrow$ color-counting

- $\mathrm{w}_{v,i} = i$
- Order of weights: $\leq$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \dots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \quad \text{for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \dots, k\}$

  $\rightarrow$ **$k+1$ colors**

- $check'(v, i, \mu_0, \dots, \mu_k) = \left(i = 0 \Rightarrow \sum_{j=0}^{k} j \cdot \mu_j \geq k\right)$

  $\rightarrow$ **color-counting, and k-stable**

- $\mathrm{W}_{v,i} = i$

- Order of weights: $\leq$

Example: Roman $\{k\}$-domination

Given a graph G, what is the minimum weight $\sum_{v \in V} f(v)$ of a function $f\colon V \to \{0, \dots, k\}$ such that

$$\sum_{u \in N(v)} f(u) \geq k \quad \text{for all } v \in V \text{ with } f(v) = 0?$$

- $\textsc{Colors} = \{0, \dots, k\}$

  $\rightarrow$ $k+1$ colors

- $check'(v, i, \mu_0, \dots, \mu_k) = \left(i = 0 \Rightarrow \sum_{j=0}^{k} j \cdot \mu_j \geq k \right)$

  $\rightarrow$ color-counting, and k-stable

- $\mathrm{W}_{v,i} = i$
- Order of weights: $\leq$

$\rightarrow$ XP parameterized by mim-width

A&C DN logic

The **distance neighborhood logic with acyclicity and connectivity constraints** was presented by Bergougnoux, Dreier and Jaffke in 2022.

They proved that every problem expressible by a formula ϕ of constant length (and another constant parameter, namely, $d(\phi)$) in this logic is XP parameterized by the mim-width of a given decomposition.

A&C DN logic

The **distance neighborhood logic with acyclicity and connectivity constraints** was presented by Bergougnoux, Dreier and Jaffke in 2022.

They proved that every problem expressible by a formula ϕ of constant length (and another constant parameter, namely, $d(\phi)$) in this logic is XP parameterized by the mim-width of a given decomposition.

Problems that can be modeled by using A&C DN logic formulas of constant length and constant $d(\cdot)$

Problems that can be modeled as locally checkable problems with

- global properties of size, connectivity and acyclicity
- constant number of colors,
- d-stable functions at t distances (for some constants d, t)

A&C DN logic

The **distance neighborhood logic with acyclicity and connectivity constraints** was presented by Bergougnoux, Dreier and Jaffke in 2022.

They proved that every problem expressible by a formula ϕ of constant length (and another constant parameter, namely, $d(\phi)$) in this logic is XP parameterized by the mim-width of a given decomposition.

Problems that can be modeled by using A&C DN logic formulas of constant length and constant $d(\cdot)$

$\equiv$

Problems that can be modeled as locally checkable problems with

- global properties of size, connectivity and acyclicity
- constant number of colors,
- d-stable functions at t distances (for some constants d, t)

The framework (complete)

r-locally checkable problem

Given a graph G and

- $\textsc{Colors}$: a set of colors,
- L_v: for each vertex v, a subset of $\textsc{Colors}$ of permitted colors,
- $\textsc{Labels}$: a set of edge labels,
- ℓ_e: for each edge e, a label in $\textsc{Labels}$,
- $(\textsc{Weights}, \preceq, \oplus)$: a weight set,
- $\mathrm{w}(v, c)$: a weight function (input: vertex v and coloring c of $N^r[v]$, output: a weight),
- $check(v, c)$: check function (input: vertex v and coloring c of $N^r[v]$, output: true/false)

find the minimum weight of a coloring c such that $c(v) \in L_v$ and $check(v, c|_{N^r[v]}) = \textsc{True}\ \forall v \in V$.

Global properties considered:

- Connectivity
- Acyclicity
- Local condition composition

Summary of results

	$\lvert\textsc{Colors}\rvert$	ℓ_e	$check$, w	Global properties
carw	Polynomial	Yes	Computable in polynomial time	Connectivity, acyclicity, local cond. comp.
tw	Polynomial	Yes	Polynomial partial neighborhood system	Conectivity, acyclicity, local cond. comp.
cw	Polynomial	No	Polynomial universal partial neigh. sys.	Local condition composition
mimw	Constant	No	d-stable	Connectivity, acyclicity, local cond. comp.

The results presented throughout this whole thesis generalize those of some previous frameworks (such as LCC and ECC problems [Bodlaender, 1987] and distance LCVP problems [Jaffke, Kwon, Strømme and Telle, 2019]).

The results about the parameterization by mim-width are *almost equivalent* to those obtained by Bergougnoux, Dreier and Jaffke (2022) for A&C DN logic.

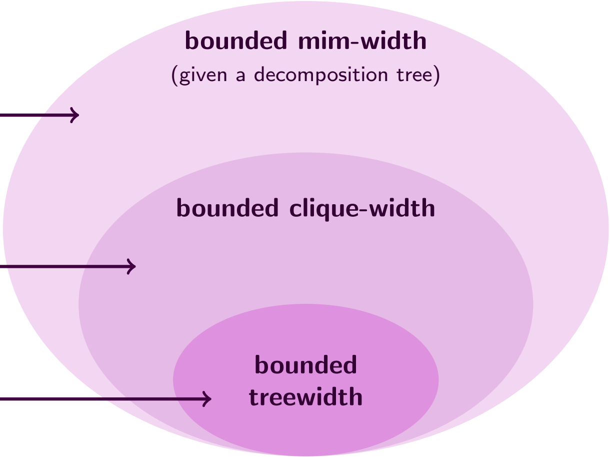
We also included PTAS on planar graphs for some locally checkable problems.

Summary of applications

Roman $\{k\}$-domination$^{(*)}$
Maximum dual domination
k-maximum happy vertices$^{(*)}$
Conflict-free k-coloring
M-GPF

k-community
Max-PDS
Maximum γ-quasi-clique
Signed Roman domination
Global $[k]$-Roman domination
GPF with polynomial k, u

Minimum chromatic violation
Grundy domination$^{(*)}$
Additive coloring



$^{(*)}$**PTAS on planar graphs**

Open questions and future work

- Consider other width parameters (for example: sim-width)

Open questions and future work

- Consider other width parameters (for example: sim-width)
- Is there any NP-hard problem that is FPT parameterized by mim-width?

Open questions and future work

- Consider other width parameters (for example: sim-width)
- Is there any NP-hard problem that is FPT parameterized by mim-width?
- Is Minimum Odd Dominating Set NP-hard on proper interval graphs?

Open questions and future work

- Consider other width parameters (for example: sim-width)
- Is there any NP-hard problem that is FPT parameterized by mim-width?
- Is Minimum Odd Dominating Set NP-hard on proper interval graphs?
- We proved that some problems (such as Grundy domination) are XP parameterized by treewidth, but are they FPT?

Open questions and future work

- Consider other width parameters (for example: sim-width)
- Is there any NP-hard problem that is FPT parameterized by mim-width?
- Is Minimum Odd Dominating Set NP-hard on proper interval graphs?
- We proved that some problems (such as Grundy domination) are XP parameterized by treewidth, but are they FPT?
- Can we give PTAS for locally checkable problems on more graph classes?