On efficient solvability of graph problems parameterized by "width" (rank-width)

Petr Hliněný

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic

e-mail: hlineny@fi.muni.cz http://www.fi.muni.cz/~hlineny

Talk based on joint work with R. Ganian and J. Obdržálek.

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.
  - Process this information bottom-up in the decomposition.

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k (ideally, not on the structure size), or at most polynomial size (cf. XP)...

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k (ideally, not on the structure size), or at most polynomial size (cf. XP)...

- How to understand words "*all relevant information about the problem*"? Use "tables"?

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k (ideally, not on the structure size), or at most polynomial size (cf. XP)...

- How to understand words "*all relevant information about the problem*"? Use "tables"? Or...

  Look for inspiration in traditional finite automata theory!

Theorem. [Myhill–Nerode, folklore]

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k (ideally, not on the structure size), or at most polynomial size (cf. XP)...

- How to understand words "*all relevant information about the problem*"? Use "tables"? Or...

  Look for inspiration in traditional finite automata theory!

Theorem. [Myhill–Nerode, folklore]
Finite automaton states (this is our information) $\iff$
right congruence classes on the words (of a regular language).

1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a **dynamic algorithm** on a *recursive decomposition*:
  - Capture all relevant inform. about the problem on a substructure.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k (ideally, not on the structure size), or at most polynomial size (cf. XP)...

- How to understand words "*all relevant information about the problem*"? Use "tables"? Or...

  Look for inspiration in traditional finite automata theory!

  Theorem. [Myhill–Nerode, folklore]
  Finite automaton states (this is our information) $\iff$
  right congruence classes on the words (of a regular language).

- Explicit comb. extensions of this concept appeared e.g. in the works [Abrahamson and Fellows, 93], [PH, 03], or [Ganian and PH, 08].

2 The Concept of a Canonical Equivalence

How does the right congruence extend
from formal words with the concatention operation
to, say, *graphs with a kind of a "join"* operation?

2 The Concept of a Canonical Equivalence

How does the right congruence extend
from formal words with the concatention operation
to, say, *graphs with a kind of a "join"* operation?

- Consider the universe of structures $\mathcal{U}_k$ implicitly associated with
  - some (small) distinguished "*boundary of size k*" of each graph, and
  - a *join operation* $G \otimes H$ acting on the boundaries of disjoint G, H.
- Let $\mathcal{P}$ be a (decision) property we study.

2 The Concept of a Canonical Equivalence

How does the right congruence extend
from formal words with the concatention operation
to, say, *graphs with a kind of a "join"* operation?

- Consider the universe of structures $\mathcal{U}_k$ implicitly associated with
  - some (small) distinguished "*boundary of size k*" of each graph, and
  - a *join operation* $G \otimes H$ acting on the boundaries of disjoint G, H.
- Let $\mathcal{P}$ be a (decision) property we study.

Definition. The *canonical equivalence* of $\mathcal{P}$ on $\mathcal{U}_k$ is defined:

$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2}$ for any $G_1, G_2 \in \mathcal{U}_k$ if and only if, for all $H \in \mathcal{U}_k$,

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}\,.$$

2 The Concept of a Canonical Equivalence

How does the right congruence extend
from formal words with the concatention operation
to, say, *graphs with a kind of a "join"* operation?

- Consider the universe of structures $\mathcal{U}_k$ implicitly associated with
  - some (small) distinguished "*boundary of size k*" of each graph, and
  - a *join operation* $G \otimes H$ acting on the boundaries of disjoint G, H.
- Let $\mathcal{P}$ be a (decision) property we study.

Definition. The *canonical equivalence* of $\mathcal{P}$ on $\mathcal{U}_k$ is defined:

$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2}$ for any $G_1, G_2 \in \mathcal{U}_k$ if and only if, for all $H \in \mathcal{U}_k$,

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P} \,.$$

- Informally, the classes of $\approx_{\mathcal{P},k}$ capture all information about the property $\mathcal{P}$ that can "cross" our boundary of size k
(regardless of the actual meaning of "boundary" and "join").

Decision properties, or more?

Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2}$ for any $G_1, G_2 \in \mathcal{U}_k$ if and only if, for all $H \in \mathcal{U}_k$,

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}$$

Decision properties, or more?

Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2}$ for any $G_1, G_2 \in \mathcal{U}_k$ if and only if, for all $H \in \mathcal{U}_k$,

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}\,.$$

- Not only deciding the exist. of a solution, but want to find it / optimize!

Decision properties, or more?

Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

$$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2} \text{ for any } G_1, G_2 \in \mathcal{U}_k \text{ if and only if, for all } H \in \mathcal{U}_k,$$
$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}\,.$$

- Not only deciding the exist. of a solution, but want to find it / optimize!
- So, let G_1, G_2 and H be assoc. with a "*solution fragment*", say φ.

Decision properties, or more?

Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

$$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2} \text{ for any } G_1, G_2 \in \mathcal{U}_k \text{ if and only if, for all } H \in \mathcal{U}_k,$$
$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}\,.$$

- Not only deciding the exist. of a solution, but want to find it / optimize!
- So, let G_1, G_2 and H be assoc. with a "*solution fragment*", say φ.

Definition, II. The *canonical equivalence* of $\mathcal{P}$ on the *extended universe* $\mathcal{U}_k$ (of structures equipped with possible solution fragments) is defined:

$\boldsymbol{(G_1, \varphi_1) \approx_{\mathcal{P},k} (G_2, \varphi_2)}$ for $(G_i, \varphi_i) \in \mathcal{U}_k$ if and only if, for all $(H, \varphi) \in \mathcal{U}_k$,

$$(G_1, \varphi_1) \otimes (H, \varphi) \models \mathcal{P} \iff (G_2, \varphi_2) \otimes (H, \varphi) \models \mathcal{P}$$

Decision properties, or more?

Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

$$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2} \text{ for any } G_1, G_2 \in \mathcal{U}_k \text{ if and only if, for all } H \in \mathcal{U}_k,$$
$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}\,.$$

- Not only deciding the exist. of a solution, but want to find it / optimize!
- So, let G_1, G_2 and H be assoc. with a "*solution fragment*", say φ.

Definition, II. The *canonical equivalence* of $\mathcal{P}$ on the *extended universe* $\mathcal{U}_k$ (of structures equipped with possible solution fragments) is defined:

$\boldsymbol{(G_1, \varphi_1) \approx_{\mathcal{P},k} (G_2, \varphi_2)}$ for $(G_i, \varphi_i) \in \mathcal{U}_k$ if and only if, for all $(H, \varphi) \in \mathcal{U}_k$,

$$(G_1, \varphi_1) \otimes (H, \varphi) \models \mathcal{P} \iff (G_2, \varphi_2) \otimes (H, \varphi) \models \mathcal{P}\,.$$

- For simplicity, solution fragments φ can be "embedded" in $\mathcal{U}_k$ and $\otimes$.

Decision properties, or more?

Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

$\boldsymbol{G_1 \approx_{\mathcal{P},k} G_2}$ for any $G_1, G_2 \in \mathcal{U}_k$ if and only if, for all $H \in \mathcal{U}_k$,

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}\,.$$

- Not only deciding the exist. of a solution, but want to find it / optimize!
- So, let G_1, G_2 and H be assoc. with a "*solution fragment*", say φ.

Definition, II. The *canonical equivalence* of $\mathcal{P}$ on the *extended universe* $\mathcal{U}_k$ (of structures equipped with possible solution fragments) is defined:

$\boldsymbol{(G_1, \varphi_1) \approx_{\mathcal{P},k} (G_2, \varphi_2)}$ for $(G_i, \varphi_i) \in \mathcal{U}_k$ if and only if, for all $(H, \varphi) \in \mathcal{U}_k$,

$$(G_1, \varphi_1) \otimes (H, \varphi) \models \mathcal{P} \iff (G_2, \varphi_2) \otimes (H, \varphi) \models \mathcal{P}\,.$$

- For simplicity, solution fragments φ can be "embedded" in $\mathcal{U}_k$ and $\otimes$.
- Can, e.g., count the solutions in each class of $\approx_{\mathcal{P},k}$, or keep an opt. one.

3 From a Canonical equivalence to an Algorithm

To give an algorith. usable meaning to the terms "boundary, join, and universe," we set them in the context of *tree-shaped* decompositions as follows...

3 From a Canonical equivalence to an Algorithm

To give an algorith. usable meaning to the terms "boundary, join, and universe," we set them in the context of *tree-shaped* decompositions as follows...

Parse trees of decompositions

- Considering a rooted ??-decomposition of a graph G, we build on the following correspondence:

boundary size k $\leftrightarrow$ restricted bag-size / width / etc in decomposition

join operator $\otimes$ $\leftrightarrow$ the way pieces of G "*stick together*" in decomp.

3 From a Canonical equivalence to an Algorithm

To give an algorith. usable meaning to the terms "boundary, join, and universe," we set them in the context of *tree-shaped* decompositions as follows...

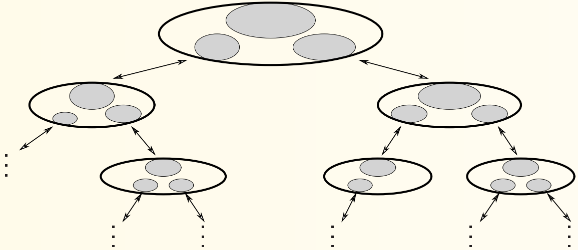
Parse trees of decompositions

- Considering a rooted ??-decomposition of a graph G, we build on the following correspondence:

boundary size k $\leftrightarrow$ restricted bag-size / width / etc in decomposition

join operator $\otimes$ $\leftrightarrow$ the way pieces of G "*stick together*" in decomp.

- This can be (visually) seen as...



The Myhill–Nerode theorem, and beyond

"Turn" a *canonical equivalence* into an algorithm. usable thing... The case of

- ★ a **finite** canonical index, i.e. $O(f(k))$ classes in the equivalence.

The Myhill–Nerode theorem, and beyond

"Turn" a *canonical equivalence* into an algorithm. usable thing... The case of

★ a **finite** canonical index, i.e. $O(f(k))$ classes in the equivalence.

Then immediately:

Theorem. Canonical equivalence classes $\iff$ the states of a finite tree automaton $\mathcal{A}$ for the property $\mathcal{P}$.

The Myhill–Nerode theorem, and beyond

"Turn" a *canonical equivalence* into an algorithm. usable thing. . . The case of

★ a **finite** canonical index, i.e. $O(f(k))$ classes in the equivalence.

Then immediately:

Theorem. Canonical equivalence classes $\iff$
the states of a finite tree automaton $\mathcal{A}$ for the property $\mathcal{P}$.

- This automaton can be easily simulated in linear time.

The Myhill–Nerode theorem, and beyond

"Turn" a *canonical equivalence* into an algorithm. usable thing... The case of

★ a **finite** canonical index, i.e. $O(f(k))$ classes in the equivalence.

Then immediately:

Theorem. Canonical equivalence classes $\iff$
the states of a finite tree automaton $\mathcal{A}$ for the property $\mathcal{P}$.

- This automaton can be easily simulated in linear time.
- A little more work can find a satisfying valuation of the free variables in $\mathcal{P}$, or to enumerate all possible solutions.

The Myhill–Nerode theorem, and beyond

"Turn" a *canonical equivalence* into an algorithm. usable thing... The case of

★ a **finite** canonical index, i.e. $O(f(k))$ classes in the equivalence.

Then immediately:

Theorem. Canonical equivalence classes $\iff$ the states of a finite tree automaton $\mathcal{A}$ for the property $\mathcal{P}$.

- This automaton can be easily simulated in linear time.
- A little more work can find a satisfying valuation of the free variables in $\mathcal{P}$, or to enumerate all possible solutions.
- And most importantly, the transition function of $\mathcal{A}$ can be hard-coded into the algorithm!

The Myhill–Nerode theorem, and beyond

"Turn" a *canonical equivalence* into an algorithm. usable thing... The case of

★ a **finite** canonical index, i.e. $O(f(k))$ classes in the equivalence.

Then immediately:

Theorem. Canonical equivalence classes $\iff$ the states of a finite tree automaton $\mathcal{A}$ for the property $\mathcal{P}$.

- This automaton can be easily simulated in linear time.
- A little more work can find a satisfying valuation of the free variables in $\mathcal{P}$, or to enumerate all possible solutions.
- And most importantly, the transition function of $\mathcal{A}$ can be hard-coded into the algorithm!

  $\rightarrow$ We do not need to know the equivalence classes exactly and constructively, just enough to have some (weak) estimate on them...

. . . and beyond Myhill–Nerode

A *canonical equivalence* into an algorithm. usable thing. The second case of

- ★ a **polynomial** canonical index, i.e. $O(n^{f(k)})$ classes in the equivalence:

. . . and beyond Myhill–Nerode

A *canonical equivalence* into an algorithm. usable thing. The second case of

- ★ a **polynomial** canonical index, i.e. $O(n^{f(k)})$ classes in the equivalence:

Unfortunately, no finite automaton, no hard-coded transition function. . .

hence no immediate conclusion this time.

. . . and beyond Myhill–Nerode

A *canonical equivalence* into an algorithm. usable thing. The second case of

★ a **polynomial** canonical index, i.e. $O(n^{f(k)})$ classes in the equivalence:

Unfortunately, no finite automaton, no hard-coded transition function. . .

hence no immediate conclusion this time.

- Need to precisely describe the classes of (mostly; some *refinement* of) the canonical equivalence of $\mathcal{P}$.

. . . and beyond Myhill–Nerode

A *canonical equivalence* into an algorithm. usable thing. The second case of

★ a **polynomial** canonical index, i.e. $O(n^{f(k)})$ classes in the equivalence:

Unfortunately, no finite automaton, no hard-coded transition function. . .

hence no immediate conclusion this time.

- Need to precisely describe the classes of (mostly; some *refinement* of) the canonical equivalence of $\mathcal{P}$.
- Can this description be parsed along the tree in XP time?

. . . and beyond Myhill–Nerode

A *canonical equivalence* into an algorithm. usable thing. The second case of

★ a **polynomial** canonical index, i.e. $O(n^{f(k)})$ classes in the equivalence:

Unfortunately, no finite automaton, no hard-coded transition function. . .

hence no immediate conclusion this time.

- Need to precisely describe the classes of (mostly; some *refinement* of) the canonical equivalence of $\mathcal{P}$.
- Can this description be parsed along the tree in XP time? Not clear. . .

  In other words, can we compute the assoc. "transition funct." efficiently?
  If so, then everything else again works smoothly as above.

. . . and beyond Myhill–Nerode

A *canonical equivalence* into an algorithm. usable thing. The second case of

★ a **polynomial** canonical index, i.e. $O(n^{f(k)})$ classes in the equivalence:

Unfortunately, no finite automaton, no hard-coded transition function. . .

hence no immediate conclusion this time.

- Need to precisely describe the classes of (mostly; some *refinement* of) the canonical equivalence of $\mathcal{P}$.
- Can this description be parsed along the tree in XP time? Not clear. . .

  In other words, can we compute the assoc. "transition funct." efficiently? If so, then everything else again works smoothly as above.
- Both positive and negative examples will be given further.

4 Clique-width and Rank-width

How "tree-like" a graph is in some well-defined sense (the width)?

- A topic occuring both in pure theory (e.g. Graph Minors),
  and in algorithms (Fixed parameter tractability).

4 Clique-width and Rank-width

How "tree-like" a graph is in some well-defined sense (the width)?

- A topic occuring both in pure theory (e.g. Graph Minors),
  and in algorithms (Fixed parameter tractability).
- Many definitions known,
  e.g. *tree-width*, *path-width*, *branch-width*, *DAG-width* . . .

4 Clique-width and Rank-width

How "tree-like" a graph is in some well-defined sense (the width)?

- A topic occuring both in pure theory (e.g. Graph Minors),
  and in algorithms (Fixed parameter tractability).
- Many definitions known,
  e.g. *tree-width*, *path-width*, *branch-width*, *DAG-width* ...

Clique-width – another graph complexity measure [Courcelle and Olariu, 00], defined by the operations on vertex–labeled $(1, 2, \ldots, k)$ graphs:

- create a new vertex with label i,
- take the disjoint union of two labeled graphs,
- add all edges between vertices of label i and label j,
- and relabel all vertices with label i to have label j.

4 Clique-width and Rank-width

How "tree-like" a graph is in some well-defined sense (the width)?

- A topic occuring both in pure theory (e.g. Graph Minors),
  and in algorithms (Fixed parameter tractability).
- Many definitions known,
  e.g. *tree-width*, *path-width*, *branch-width*, *DAG-width* . . .

Clique-width – another graph complexity measure [Courcelle and Olariu, 00], defined by the operations on vertex–labeled $(1, 2, \dots, k)$ graphs:

- create a new vertex with label i,
- take the disjoint union of two labeled graphs,
- add all edges between vertices of label i and label j,
- and relabel all vertices with label i to have label j.

$\longrightarrow$ giving the *expression tree* (parse tree) for clique-width.

4 Clique-width and Rank-width

How "tree-like" a graph is in some well-defined sense (the width)?

- A topic occuring both in pure theory (e.g. Graph Minors),
  and in algorithms (Fixed parameter tractability).

- Many definitions known,
  e.g. *tree-width*, *path-width*, *branch-width*, *DAG-width* . . .

Clique-width – another graph complexity measure [Courcelle and Olariu, 00], defined by the operations on vertex–labeled $(1, 2, \ldots, k)$ graphs:

– create a new vertex with label i,
– take the disjoint union of two labeled graphs,
– add all edges between vertices of label i and label j,
– and relabel all vertices with label i to have label j.

$\longrightarrow$ giving the *expression tree* (parse tree) for clique-width.

$\longrightarrow$ **A problem** – no known way how to construct an expression tree!

Rank-decomposition

- [Oum and Seymour, 03] Bringing the branch-decomposition approach to measure "complexity" of vertex subsets $X \subseteq V(G)$ via *cut-rank*:

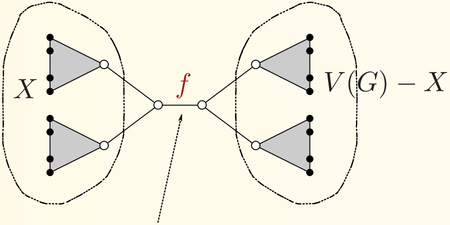
$$\varrho_G(X) = \text{rank of} \quad \begin{array}{c} \\ X \end{array} \overset{V(G)-X}{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}} \text{ modulo } 2$$

Rank-decomposition

- [Oum and Seymour, 03] Bringing the branch-decomposition approach to measure "complexity" of vertex subsets $X \subseteq V(G)$ via *cut-rank*:

$$\varrho_G(X) = \text{rank of } \quad X \overset{V(G)-X}{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}} \text{ modulo } 2$$

Definition. Decompose $V(G)$ one-to-one into the leaves of a subcubic tree. Then



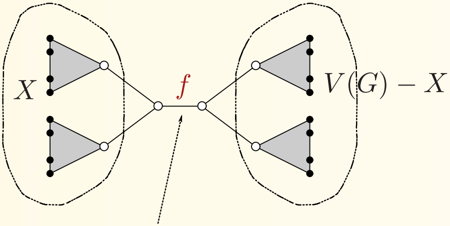
width$(e) = \varrho_G(X)$ where X is displayed by f in the tree.

Rank-decomposition

- [Oum and Seymour, 03] Bringing the branch-decomposition approach to measure "complexity" of vertex subsets $X \subseteq V(G)$ via *cut-rank*:

$$\varrho_G(X) = \text{rank of} \quad X \overset{V(G)-X}{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}} \text{ modulo } 2$$

Definition. Decompose $V(G)$ one-to-one into the leaves of a subcubic tree. Then



width$(e) = \varrho_G(X)$ where X is displayed by f in the tree.

- **Rank-width** $= \min_{\text{rank-decs. of } G} \max \{\text{width}(f) : f \text{ tree edge}\}$

Comparing rank-width to clique-width

- Rank-width is related to clique-width as $rw \leq cw \leq 2^{rw+1} - 1$.

Comparing rank-width to clique-width

- Rank-width is related to clique-width as $\boldsymbol{rw \leq cw \leq 2^{rw+1} - 1}$.
- Clique-width can really be up to exponentially higher than rank-width.

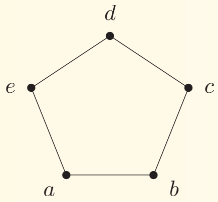
Comparing rank-width to clique-width

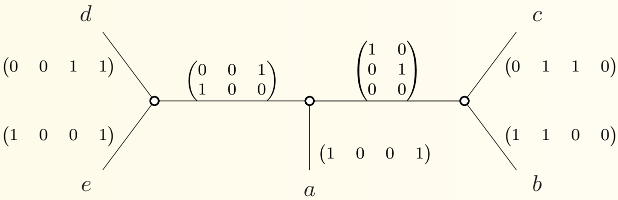
- Rank-width is related to clique-width as $\boldsymbol{rw \leq cw \leq 2^{rw+1} - 1}$.
- Clique-width can really be up to exponentially higher than rank-width.
- [Oum and PH, 08] There is an *FPT algorithm* for computing an optimal width-t rank-decomposition of a graph in time $O(f(t) \cdot n^3)$.

Comparing rank-width to clique-width

- Rank-width is related to clique-width as $\boldsymbol{rw \le cw \le 2^{rw+1} - 1}$.
- Clique-width can really be up to exponentially higher than rank-width.
- [Oum and PH, 08] There is an *FPT algorithm* for computing an optimal width-t rank-decomposition of a graph in time $O(f(t) \cdot n^3)$.

An example. Cycle C_5 and its *rank-decomposition* of width 2:





Parse trees for rank-decompositions

Unlike for tree- or clique-decompositions with obvious parse trees, what is the "boundary" and "join operation" for rank-width?

Our "boundary" includes all vertices, and "join" is just an implicit matrix.

Parse trees for rank-decompositions

Unlike for tree- or clique-decompositions with obvious parse trees, what is the "boundary" and "join operation" for rank-width?

Our "boundary" includes all vertices, and "join" is just an implicit matrix.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),

Parse trees for rank-decompositions

Unlike for tree- or clique-decompositions with obvious parse trees, what is the "boundary" and "join operation" for rank-width?

Our "boundary" includes all vertices, and "join" is just an implicit matrix.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),
  - *join* $\sim$ edge $uv \leftrightarrow lab(u) \cdot lab(v) = 1$ over $GF(2)^t$,
    i.e. "odd intersection" of vertex labelings.

Parse trees for rank-decompositions

Unlike for tree- or clique- decompositions with obvious parse trees, what is the "boundary" and "join operation" for rank-width?

Our "boundary" includes all vertices, and "join" is just an implicit matrix.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),
  - *join* $\sim$ edge $uv \leftrightarrow lab(u) \cdot lab(v) = 1$ over $GF(2)^t$,
    i.e. "odd intersection" of vertex labelings.
- Join $\to$ a *composition* operator with relabelings f_1, f_2, g;

$$(G_1, lab^1) \otimes[\boldsymbol{g} \mid \boldsymbol{f}_1, \boldsymbol{f}_2] (G_2, lab^2) = (H, lab)$$

$\Longrightarrow$ the rank-width **parse tree** [Ganian and PH, 08]:

Parse trees for rank-decompositions

Unlike for tree- or clique- decompositions with obvious parse trees, what is the "boundary" and "join operation" for rank-width?

Our "boundary" includes all vertices, and "join" is just an implicit matrix.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),
  - *join* $\sim$ edge $uv \;\leftrightarrow\; lab(u) \cdot lab(v) = 1$ over $GF(2)^t$,
    i.e. "odd intersection" of vertex labelings.

- Join $\;\to\;$ a *composition* operator with relabelings f_1, f_2, g;

$$(G_1, lab^1) \;\otimes[\boldsymbol{g} \mid \boldsymbol{f}_1, \boldsymbol{f}_2]\; (G_2, lab^2) \;=\; (H, lab)$$

$\Longrightarrow$ the rank-width **parse tree** [Ganian and PH, 08]:

t-**labeling** parse tree for $G \iff$ rank-width of $G \leq \boldsymbol{t}$.

Parse trees for rank-decompositions

Unlike for tree- or clique- decompositions with obvious parse trees, what is the "boundary" and "join operation" for rank-width?

Our "boundary" includes all vertices, and "join" is just an implicit matrix.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),
  - *join* $\sim$ edge $uv \leftrightarrow lab(u) \cdot lab(v) = 1$ over $GF(2)^t$, i.e. "odd intersection" of vertex labelings.
- Join $\to$ a *composition* operator with relabelings f_1, f_2, g;

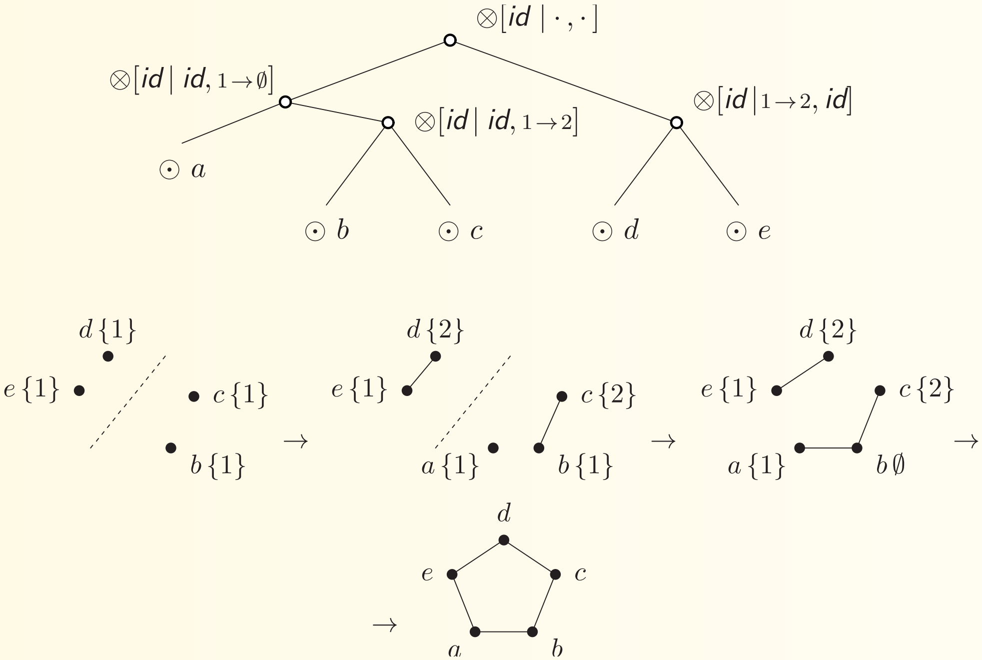
$$(G_1, lab^1) \otimes[\boldsymbol{g} \mid \boldsymbol{f}_1, \boldsymbol{f}_2] \; (G_2, lab^2) \; = \; (H, lab)$$

$\Longrightarrow$ the rank-width **parse tree** [Ganian and PH, 08]:

t-**labeling** parse tree for G $\iff$ rank-width of $G \leq \boldsymbol{t}$.

- Independently considered related notion of R_t-*join* decompositions by [Bui-Xuan, Telle, and Vatshelle, 08].

A parse tree. An example generating the cycle C_5 (of rank-width 2):



5 The nice examples: Two XP-time Algorithms

★ HAM = the *Hamiltonian Path* problem in a graph
of bounded rank-width / clique-width:

5 The nice examples: Two XP-time Algorithms

★ HAM = the *Hamiltonian Path* problem in a graph
of bounded rank-width / clique-width:

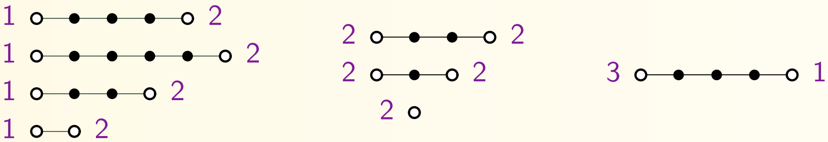
– a solution fragment $\sim$ *linear forest* F spanning a subraph;

– a canonical equivalence class of F $\sim$ the multiset of
label-pairs identifying the ends of paths in F.

5 The nice examples: Two XP-time Algorithms

★ HAM = the *Hamiltonian Path* problem in a graph of bounded rank-width / clique-width:

- a solution fragment $\sim$ *linear forest* F spanning a subraph;
- a canonical equivalence class of F $\sim$ the multiset of label-pairs identifying the ends of paths in F.



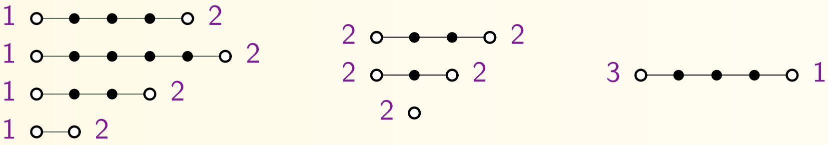
So, the number of classes is $\leq O\left(n^{4^{rw}}\right) \sim O\left(n^{cw^2}\right)$, but is this enough to say?

5 The nice examples: Two XP-time Algorithms

★ HAM = the *Hamiltonian Path* problem in a graph of bounded rank-width / clique-width:

– a solution fragment ∼ *linear forest* F spanning a subraph;

– a canonical equivalence class of F ∼ the multiset of label-pairs identifying the ends of paths in F.



So, the number of classes is $\leq O\left(n^{4^{rw}}\right) \sim O\left(n^{cw^2}\right)$, but is this enough to say?

- No, we must give also an algorithm how to “combine / process” our information on parse trees – not hard-coded this time!
- In this particular case the processing algorithm runs very smoothly. . .

The second example for rank-width

- ★ COL = *Chromatic Number* of a graph (i.e. to output the number):

The second example for rank-width

★ COL = *Chromatic Number* of a graph (i.e. to output the number):

- a solution fragment $\sim$ a valid *colour partition* of a subgraph;
- a canonical equivalence class $\sim$ the multiset of vector subspaces of $GF(2)^{rw}$ spanned by these colour parts.

The second example for rank-width

★ COL = *Chromatic Number* of a graph (i.e. to output the number):

- a solution fragment $\sim$ a valid *colour partition* of a subgraph;
- a canonical equivalence class $\sim$ the multiset of vector subspaces of $GF(2)^{rw}$ spanned by these colour parts.

  Note; for rank-width it is enough to know the subspace of a label set instead of the set itself – speed-up compared to clique-width.

- Again, the number of classes is $O\left(n^{2^{rw^2}}\right) \prec O\left(n^{2^{cw}}\right)$,

The second example for rank-width

★ COL = *Chromatic Number* of a graph (i.e. to output the number):

- a solution fragment $\sim$ a valid *colour partition* of a subgraph;
- a canonical equivalence class $\sim$ the multiset of vector subspaces of $GF(2)^{rw}$ spanned by these colour parts.

  Note; for rank-width it is enough to know the subspace of a label set instead of the set itself – speed-up compared to clique-width.

- Again, the number of classes is $O\left(n^{2^{rw^2}}\right) \prec O\left(n^{2^{cw}}\right)$,
- and there is a reasonably straightforward algorithm to "combine / process" this information on parse trees.

6 And the naughty ex.: the MinLOB Problem

★ MinLOB = *Minimum Leaf Outbanching* in a digraph:

– Given G and ℓ;
is there an *out-directed spanning tree* of G with $\leq \ell$ leaves?

6 And the naughty ex.: the MinLOB Problem

★ MinLOB = *Minimum Leaf Outbanching* in a digraph:

– Given G and ℓ;
  is there an *out-directed spanning tree* of G with $\leq \ell$ leaves?

– For constant ℓ this problem simply generalizes Hamiltonian Path, but what for ℓ on the input? Quite a difference...

6 And the naughty ex.: the MinLOB Problem

★ MinLOB = *Minimum Leaf Outbanching* in a digraph:

- Given G and ℓ;
  is there an *out-directed spanning tree* of G with $\leq \ell$ leaves?

- For constant ℓ this problem simply generalizes Hamiltonian Path, but what for ℓ on the input? Quite a difference...

• Trying the same approach as previously

- a solution fragment $\sim$ an *out-forest* in a subdigraph;
- a canonical equivalence class $\sim$???

6 And the naughty ex.: the MinLOB Problem

★ MinLOB = *Minimum Leaf Outbanching* in a digraph:

- Given G and ℓ;
  is there an *out-directed spanning tree* of G with $\leq \ell$ leaves?

- For constant ℓ this problem simply generalizes Hamiltonian Path, but what for ℓ on the input? Quite a difference. . .

• Trying the same approach as previously

- a solution fragment $\sim$ an *out-forest* in a subdigraph;
- a canonical equivalence class $\sim$???
  For each tree of our out-forest, the root label and the multiset of "non-leaf" labels are significant (to connect with other fragments).

6 And the naughty ex.: the MinLOB Problem

★ MinLOB = *Minimum Leaf Outbanching* in a digraph:

- Given G and ℓ;
  is there an *out-directed spanning tree* of G with $\leq \ell$ leaves?

- For constant ℓ this problem simply generalizes Hamiltonian Path, but what for ℓ on the input? Quite a difference...

• Trying the same approach as previously

- a solution fragment $\sim$ an *out-forest* in a subdigraph;
- a canonical equivalence class $\sim$???

  For each tree of our out-forest, the root label and the multiset of "non-leaf" labels are significant (to connect with other fragments).

  No, this simple adaptation would give a bound exponential in n.

6 And the naughty ex.: the MinLOB Problem

★ MinLOB = *Minimum Leaf Outbanching* in a digraph:

- Given G and ℓ;
  is there an *out-directed spanning tree* of G with $\leq \ell$ leaves?

- For constant ℓ this problem simply generalizes Hamiltonian Path, but what for ℓ on the input? Quite a difference...

- Trying the same approach as previously
  - a solution fragment $\sim$ an *out-forest* in a subdigraph;
  - a canonical equivalence class $\sim$???
    For each tree of our out-forest, the root label and the multiset of "non-leaf" labels are significant (to connect with other fragments).

    No, this simple adaptation would give a bound exponential in n.

Actually, it looks like we face here a new situation not observed before among the known XP algorithms on bounded clique-width / rank-width graphs!

Bounding the canonical index of MinLOB

Recall: Outbranching $\rightarrow$ a solution fragment $\sim$ out-forest $\rightarrow$ *out-trees*.

Bounding the canonical index of MinLOB

Recall: Outbranching $\rightarrow$ a solution fragment $\sim$ out-forest $\rightarrow$ *out-trees*.

Shape of an out-tree T $=$ the pair (a, B) where

- a is the root label of T, and
- B the label set of the *active* (i.e., non-leaf in the result) vertices.

Bounding the canonical index of MinLOB

Recall: Outbranching $\rightarrow$ a solution fragment $\sim$ out-forest $\rightarrow$ *out-trees*.

Shape of an out-tree T $=$ the pair (a, B) where

- a is the root label of T, and
- B the label set of the *active* (i.e., non-leaf in the result) vertices.

Signature of an out-forest F $=$ the triple of

- the number of vertices of F to become the out-branching leaves,
- the label multiset of the active vertices of F, and

Bounding the canonical index of MinLOB

Recall: Outbranching $\rightarrow$ a solution fragment $\sim$ out-forest $\rightarrow$ *out-trees*.

Shape of an out-tree T $=$ the pair (a, B) where

- a is the root label of T, and
- B the label set of the *active* (i.e., non-leaf in the result) vertices.

Signature of an out-forest F $=$ the triple of

- the number of vertices of F to become the out-branching leaves,
- the label multiset of the active vertices of F, and
- the numbers of out-trees in F of every possible shape (finitely many).

Bounding the canonical index of MinLOB

Recall: Outbranching $\rightarrow$ a solution fragment $\sim$ out-forest $\rightarrow$ *out-trees*.

Shape of an out-tree T $=$ the pair (a, B) where

- a is the root label of T, and
- B the label set of the *active* (i.e., non-leaf in the result) vertices.

Signature of an out-forest F $=$ the triple of

- the number of vertices of F to become the out-branching leaves,
- the label multiset of the active vertices of F, and
- the numbers of out-trees in F of every possible shape (finitely many).

Theorem. If two out-forests have the same signature, then they are canonically equivalent for MinLOB.

Bounding the canonical index of MinLOB

Recall: Outbranching $\rightarrow$ a solution fragment $\sim$ out-forest $\rightarrow$ *out-trees*.

Shape of an out-tree T $=$ the pair (a, B) where

- a is the root label of T, and
- B the label set of the *active* (i.e., non-leaf in the result) vertices.

Signature of an out-forest F $=$ the triple of

- the number of vertices of F to become the out-branching leaves,
- the label multiset of the active vertices of F, and
- the numbers of out-trees in F of every possible shape (finitely many).

Theorem. If two out-forests have the same signature, then they are canonically equivalent for MinLOB.

$\Longrightarrow$ The number of equivalence classes of MinLOB is in XP.

What about an algorithm, though?

An XP Algorithm for MinLOB on Rank-width

Fact. Inform. on possible out-forest signs. cannot be processed on a parse tree.

So, what can we do better?

An XP Algorithm for MinLOB on Rank-width

Fact. Inform. on possible out-forest signs. cannot be processed on a parse tree.

So, what can we do better?

- Active vertices $\rightarrow$ *potentially active* vertices:
  - a notion bound to a particular parse tree;
  - roughly saying that a vertex has been active somewhen before, and some other stays active with the same label.

An XP Algorithm for MinLOB on Rank-width

Fact. Inform. on possible out-forest signs. cannot be processed on a parse tree.

So, what can we do better?

- Active vertices $\rightarrow$ *potentially active* vertices:
  - a notion bound to a particular parse tree;
  - roughly saying that a vertex has been active somewhen before, and some other stays active with the same label.
- Signature $\rightarrow$ *weak signature* tracing potentialy active shapes:
  - a notion suited right for dynamic processing on a parse tree.

An XP Algorithm for MinLOB on Rank-width

Fact. Inform. on possible out-forest signs. cannot be processed on a parse tree.

So, what can we do better?

- Active vertices $\rightarrow$ *potentially active* vertices:
  - a notion bound to a particular parse tree;
  - roughly saying that a vertex has been active somewhen before, and some other stays active with the same label.
- Signature $\rightarrow$ *weak signature* tracing potentialy active shapes:
  - a notion suited right for dynamic processing on a parse tree.

Theorem. If a "singleton" weak signature is found on a parse tree then the parsed graph contains an out-branching of the same number of leaves (constructively).

An XP Algorithm for MinLOB on Rank-width

Fact. Inform. on possible out-forest signs. cannot be processed on a parse tree.

So, what can we do better?

- Active vertices $\rightarrow$ *potentially active* vertices:
  - a notion bound to a particular parse tree;
  - roughly saying that a vertex has been active somewhen before, and some other stays active with the same label.
- Signature $\rightarrow$ *weak signature* tracing potentialy active shapes:
  - a notion suited right for dynamic processing on a parse tree.

Theorem. If a "singleton" weak signature is found on a parse tree then the parsed graph contains an out-branching of the same number of leaves (constructively).

$\Longrightarrow$ There is an XP algorithm for MinLOB on digraphs of bounded rank-width / clique-width, ...

An XP Algorithm for MinLOB on Rank-width

Fact. Inform. on possible out-forest signs. cannot be processed on a parse tree.

So, what can we do better?

- Active vertices $\rightarrow$ *potentially active* vertices:
  - a notion bound to a particular parse tree;
  - roughly saying that a vertex has been active somewhen before, and some other stays active with the same label.
- Signature $\rightarrow$ *weak signature* tracing potentialy active shapes:
  - a notion suited right for dynamic processing on a parse tree.

Theorem. If a "singleton" weak signature is found on a parse tree then the parsed graph contains an out-branching of the same number of leaves (constructively).

$\Longrightarrow$ There is an XP algorithm for MinLOB on digraphs of bounded rank-width / clique-width, . . .
but it does not fit into the Myhill–Nerode-like scheme!

7 Final remarks

The “naughty example” of the MinLOB problem and its XP algorithm on digraphs of bounded rank-width / clique-width raises some intrusive questions. . .

Namely:

- Is there a better refinement of the canonical equivalence of MinLOB, i.e. one that can be directly processed along a parse tree in XP time?

7 Final remarks

The "naughty example" of the MinLOB problem and its XP algorithm on digraphs of bounded rank-width / clique-width raises some intrusive questions...

Namely:

- Is there a better refinement of the canonical equivalence of MinLOB, i.e. one that can be directly processed along a parse tree in XP time?
- Actually, are there more similar "naughty examples"?

7 Final remarks

The "naughty example" of the MinLOB problem and its XP algorithm on digraphs of bounded rank-width / clique-width raises some intrusive questions...

Namely:

- Is there a better refinement of the canonical equivalence of MinLOB, i.e. one that can be directly processed along a parse tree in XP time?
- Actually, are there more similar "naughty examples"?
- And more generally; is there an example of a property $\mathcal{P}$ such that the canonical equivalence of $\mathcal{P}$ has $O(n^{f(k)})$ classes, and yet deciding $\mathcal{P}$ is not in XP wrt. the width k?

7 Final remarks

The "naughty example" of the MinLOB problem and its XP algorithm on digraphs of bounded rank-width / clique-width raises some intrusive questions...

Namely:

- Is there a better refinement of the canonical equivalence of MinLOB, i.e. one that can be directly processed along a parse tree in XP time?
- Actually, are there more similar "naughty examples"?
- And more generally; is there an example of a property $\mathcal{P}$ such that the canonical equivalence of $\mathcal{P}$ has $O(n^{f(k)})$ classes, and yet deciding $\mathcal{P}$ is not in XP wrt. the width k?

THANK YOU FOR YOUR ATTENTION