AGT 2009

On Parse Trees and Myhill–Nerode Tools for Graphs of Bounded Rank-width

Petr Hliněný* and Robert Ganian

Faculty of Informatics, Masaryk University

Botanická 68a, 602 00 Brno, Czech Republic

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

1 Measuring Graph "Width"

Motivation: Trees are easy to understand and to handle, so how "tree-like" our graph is in some well-defined sense?

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

1 Measuring Graph "Width"

Motivation: Trees are easy to understand and to handle, so how "tree-like" our graph is in some well-defined sense?

- 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* ...

1 Measuring Graph "Width"

Motivation: Trees are easy to understand and to handle, so how "tree-like" our graph is in some well-defined sense?

- 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], defined by operations on vertex–labeled 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.

Rank-Decomposition

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

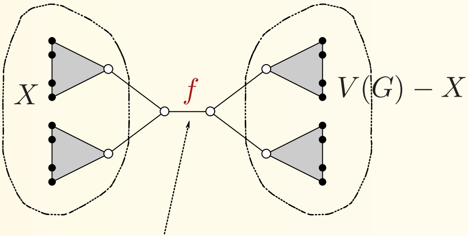
$$\varrho_G(X) = \text{rank of} \quad \begin{array}{cc} & V(G)-X \\ X & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \text{ modulo } 2$$

Rank-Decomposition

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

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

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



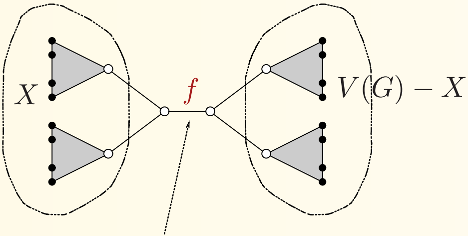
width$(e) = \varrho_G(X)$ where X is displayed by f in the tree.

Rank-Decomposition

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

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

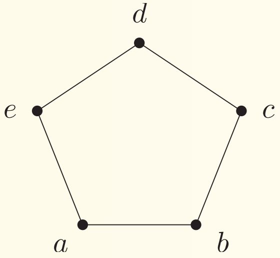
Definition. Decompose $V(G)$ one-to-one into the leaves of a subcubic tree. Then

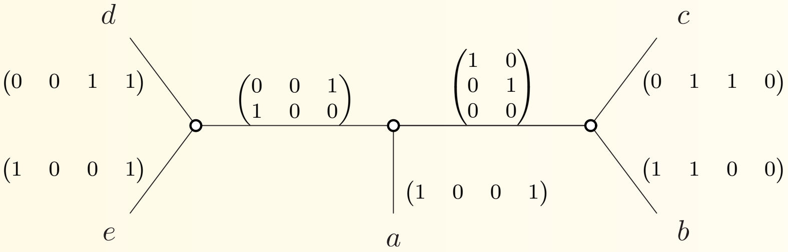


width$(e) = \varrho_G(X)$ where X is displayed by f in the tree.

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

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





Comparing these two

- Rank-width t is related to clique-width k as $\boldsymbol{t \leq k \leq 2^{t+1} - 1}$.
- Both these measures are NP-hard in general.

Comparing these two

- Rank-width t is related to clique-width k as $\boldsymbol{t \leq k \leq 2^{t+1} - 1}$.
- Both these measures are NP-hard in general.
- Clique-width *expressions* seem to be much more "explicit" than *rank-decompositions*, and more suited for design of actual algorithms.

On the other hand, however...

Comparing these two

- Rank-width t is related to clique-width k as $\boldsymbol{t \leq k \leq 2^{t+1} - 1}$.
- Both these measures are NP-hard in general.
- Clique-width *expressions* seem to be much more "explicit" than *rank-decompositions*, and more suited for design of actual algorithms.

On the other hand, however...

- [Corneil and Rotics, 05] Clique-width can really be up to exponentially higher than rank-width.

Comparing these two

- Rank-width t is related to clique-width k as $\boldsymbol{t \leq k \leq 2^{t+1} - 1}$.
- Both these measures are NP-hard in general.
- Clique-width *expressions* seem to be much more "explicit" than *rank-decompositions*, and more suited for design of actual algorithms.

On the other hand, however...

- [Corneil and Rotics, 05] Clique-width can really be up to exponentially higher than rank-width.
- [Oum and PH, 07] There is an *FPT algorithm* for computing an optimal rank-decomposition of a graph in time $O(f(t) \cdot n^3)$.

Comparing these two

- Rank-width t is related to clique-width k as $\boldsymbol{t \le k \le 2^{t+1} - 1}$.
- Both these measures are NP-hard in general.
- Clique-width *expressions* seem to be much more "explicit" than *rank-decompositions*, and more suited for design of actual algorithms.

On the other hand, however...

- [Corneil and Rotics, 05] Clique-width can really be up to exponentially higher than rank-width.
- [Oum and PH, 07] There is an *FPT algorithm* for computing an optimal rank-decomposition of a graph in time $O(f(t) \cdot n^3)$.
- And some new results suggest that algorithms designed on rank-decompositions run faster than those designed on clique-width expressions...

2 Dynamic Algorithms and Parse Trees

- A typical idea for a *dynamic algorithm* on a “tree-like” decomposition:
  - Capture all relevant information about the problem on a subtree.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k, and not on the graph size.

2 Dynamic Algorithms and Parse Trees

- A typical idea for a *dynamic algorithm* on a "tree-like" decomposition:
  - Capture all relevant information about the problem on a subtree.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k, and not on the graph size.

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

  Look for inspiration in traditional finite automata theory!

2 Dynamic Algorithms and Parse Trees

- A typical idea for a *dynamic algorithm* on a "tree-like" decomposition:
  - Capture all relevant information about the problem on a subtree.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k, and not on the graph size.

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

Look for inspiration in traditional finite automata theory!

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

2 Dynamic Algorithms and Parse Trees

- A typical idea for a *dynamic algorithm* on a "tree-like" decomposition:
  - Capture all relevant information about the problem on a subtree.
  - Process this information bottom-up in the decomposition.
  - Importantly, this information has size depending only on k, and not on the graph size.

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

  Look for inspiration in traditional finite automata theory!

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

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

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?

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

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 graphs $\mathcal{U}_k$ implicitly associated with
  - some (small) distinguished "*boundary of size k*" of each graph, and
  - a *join operation* $G \oplus H$ acting on the boundaries of disjoint G, H.
- Let $\mathcal{P}$ be a graph 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 $H \in \mathcal{U}_k$,

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

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 graphs $\mathcal{U}_k$ implicitly associated with
  - some (small) distinguished "*boundary of size k*" of each graph, and
  - a *join operation* $G \oplus H$ acting on the boundaries of disjoint G, H.
- Let $\mathcal{P}$ be a graph 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 $H \in \mathcal{U}_k$,

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

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

Parse Trees of decompositions

To give a real usable meaning to the above terms “boundary, join, and universe” we set them in the context of tree-shaped decompositions as follows. . .

Parse Trees of decompositions

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

- 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 $\oplus$	$\leftrightarrow$	the way pieces of G "stick together" in decomp.

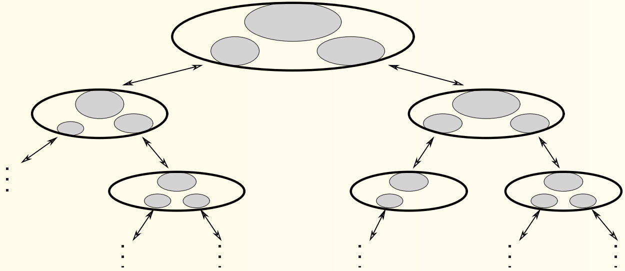
Parse Trees of decompositions

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

- 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 $\oplus$	$\leftrightarrow$	the way pieces of G "stick together" in decomp.

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



3 Parse Trees for Rank-Decompositions

Unlike for branch- or tree-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 rank!

3 Parse Trees for Rank-Decompositions

Unlike for branch- or tree-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 rank!

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

3 Parse Trees for Rank-Decompositions

Unlike for branch- or tree-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 rank!

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\ldots,t\}}$ (multi-colouring),
  - *join* $\sim$ bilinear form $\mathbf{g}$ over $GF(2)^t$ (i.e. "odd intersection") s.t.

$$\text{edge } uv \;\leftrightarrow\; lab(u) \cdot \mathbf{g} \cdot lab(v) = 1.$$

3 Parse Trees for Rank-Decompositions

Unlike for branch- or tree-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 rank!

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),
  - *join* $\sim$ bilinear form $\mathbf{g}$ over $GF(2)^t$ (i.e. "odd intersection") s.t.

$$\text{edge } uv \;\leftrightarrow\; lab(u) \cdot \mathbf{g} \cdot lab(v) = 1.$$

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

$$(G_1, lab^1) \otimes[\mathbf{g} \mid f_1, f_2] \ (G_2, lab^2) \ = \ (H, lab)$$

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

3 Parse Trees for Rank-Decompositions

Unlike for branch- or tree-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 rank!

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\ldots,t\}}$ (multi-colouring),
  - *join* $\sim$ bilinear form $\mathbf{g}$ over $GF(2)^t$ (i.e. "odd intersection") s.t.

$$\text{edge } uv \;\leftrightarrow\; lab(u) \cdot \mathbf{g} \cdot lab(v) = 1.$$

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

$$(G_1, lab^1) \otimes[\mathbf{g} \mid f_1, f_2] \; (G_2, lab^2) \;=\; (H, lab)$$

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

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

3 Parse Trees for Rank-Decompositions

Unlike for branch- or tree-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 rank!

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - *boundary* $\sim$ labeling $lab : V(G) \to 2^{\{1,2,\dots,t\}}$ (multi-colouring),
  - *join* $\sim$ bilinear form $\mathbf{g}$ over $GF(2)^t$ (i.e. "odd intersection") s.t.

$$\text{edge } uv \;\leftrightarrow\; lab(u) \cdot \mathbf{g} \cdot lab(v) = 1.$$

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

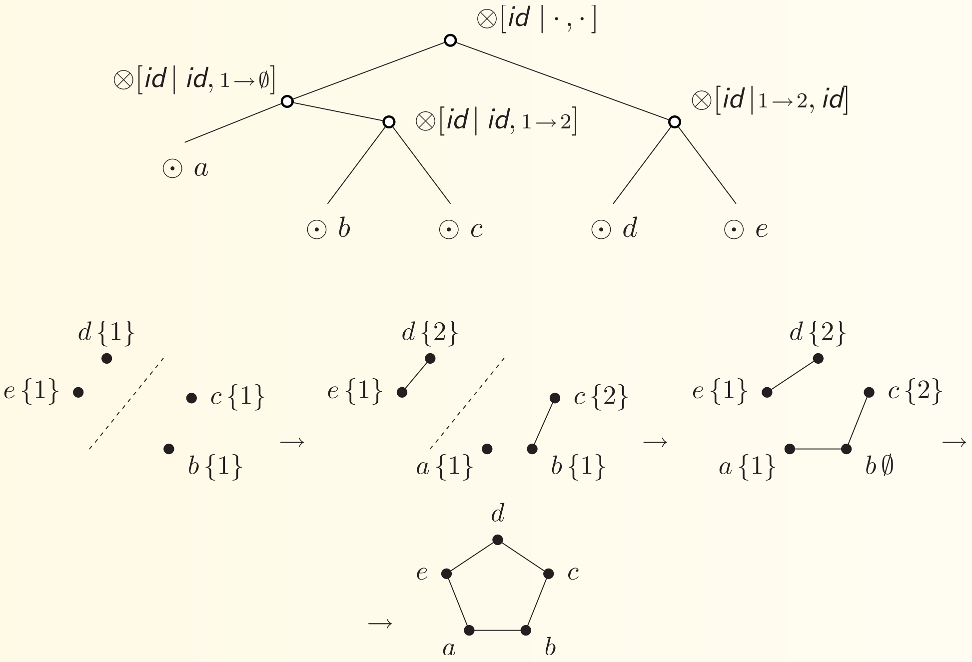
$$(G_1, lab^1) \;\otimes[\mathbf{g} \mid f_1, f_2]\; (G_2, lab^2) \;=\; (H, lab)$$

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

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

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

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



4 Canonical Equivalence and Algorithms

So, how can one use a canonical equivalence when designing actual algorithms?

4 Canonical Equivalence and Algorithms

So, how can one use a canonical equivalence when designing actual algorithms?

- Let us recall...

Theorem. [Myhill–Nerode, folklore]
A finite automaton accepts a given language $\iff$
the number of *right congruence* classes on the words is finite.

4 Canonical Equivalence and Algorithms

So, how can one use a canonical equivalence when designing actual algorithms?

- Let us recall...

Theorem. [Myhill–Nerode, folklore]
A finite automaton accepts a given language $\iff$ the number of *right congruence* classes on the words is finite.

- This automaton is constructible and can be emulated in linear time.

4 Canonical Equivalence and Algorithms

So, how can one use a canonical equivalence when designing actual algorithms?

- Let us recall...

Theorem. [Myhill–Nerode, folklore]
A finite automaton accepts a given language $\iff$
the number of *right congruence* classes on the words is finite.

- This automaton is constructible and can be emulated in linear time.

- For parse trees, a straightforward generalization reads:

Theorem. (Analogy of [Myhill–Nerode])
$\mathcal{P}$ is accepted by a finite tree automaton on parse trees of boundary size $\leq k$
$\iff$ the *canonical equivalence* $\approx_{\mathcal{P},k}$ has finitely many classes on $\mathcal{U}_k$.

(Actually, this is a "metatheorem" which requires several more unspoken technical conditions on the parse trees to hold true...)

Extended canonical equivalence

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

- To apply this concept to predicates $\mathcal{P}(X_1, \dots)$ with free variables, we extend the universe $\mathcal{U}_k$ to *partially-equipped* graphs of boundary $\leq k$.

Extended canonical equivalence

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

- To apply this concept to predicates $\mathcal{P}(X_1, \ldots)$ with free variables, we extend the universe $\mathcal{U}_k$ to *partially-equipped* graphs of boundary $\leq k$.

Theorem. [Ganian and PH, 08]

Suppose ϕ is a formula in the language MS_1. Then the canonical equivalence $\approx_{\phi,t}$ has finite index in the universe of t-labeled partially-equipped graphs.

Extended canonical equivalence

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

- To apply this concept to predicates $\mathcal{P}(X_1, \dots)$ with free variables, we extend the universe $\mathcal{U}_k$ to *partially-equipped* graphs of boundary $\leq k$.

Theorem. [Ganian and PH, 08]

Suppose ϕ is a formula in the language MS_1. Then the canonical equivalence $\approx_{\phi,t}$ has finite index in the universe of t-labeled partially-equipped graphs.

- From that one easily concludes an older result:

Theorem. [Courcelle, Makowsky, and Rotics 00]

All *LinEMSO graph optimization* problems (in MS_1 language – only vertices!) on the graphs of bounded rank-width t can be solved in time $\boldsymbol{O(f(t) \cdot n)}$.

Core idea: In dynamic processing of the given parse tree, record optimal representatives of each class of the extended canonical equivalence $\approx_{\phi,t}$ ...

Faster new algorithms

Furthermore, the concept of a canonical equivalence gives us a fine control over the runtime dependency on the width parameter – we simply estimate its index.

Consider the universe of partially-equipped t-labeled graphs (of rank-width $\leq t$).

- As shown already by [Bui-Xuan, Telle, and Vatshelle, 08];

  the canonical equivalence of *independent-set*(X) has index $\leq 2^{t(t+1)/4}$ (this relates to the number of subspaces of $GF(2)^t$).

Faster new algorithms

Furthermore, the concept of a canonical equivalence gives us a fine control over the runtime dependency on the width parameter – we simply estimate its index.

Consider the universe of partially-equipped t-labeled graphs (of rank-width $\leq t$).

- As shown already by [Bui-Xuan, Telle, and Vatshelle, 08];

  the canonical equivalence of *independent-set*(X) has index $\leq 2^{t(t+1)/4}$ (this relates to the number of subspaces of $GF(2)^t$).

Theorem. [Bui-Xuan, Telle, and Vatshelle, 08]
The *independent set* problem can be solved in time $O\big(2^{t(t+1)/2} \cdot t^3 \cdot |V(G)|\big)$, and the c*-colourability* (fixed c) in time $O\big(2^{ct(t+1)/2} \cdot ct^3 \cdot |V(G)|\big)$.

Faster new algorithms

Furthermore, the concept of a canonical equivalence gives us a fine control over the runtime dependency on the width parameter – we simply estimate its index.

Consider the universe of partially-equipped t-labeled graphs (of rank-width $\leq t$).

- As shown already by [Bui-Xuan, Telle, and Vatshelle, 08];

  the canonical equivalence of *independent-set*(X) has index $\leq 2^{t(t+1)/4}$ (this relates to the number of subspaces of $GF(2)^t$).

Theorem. [Bui-Xuan, Telle, and Vatshelle, 08]
The *independent set* problem can be solved in time $O\big(2^{t(t+1)/2} \cdot t^3 \cdot |V(G)|\big)$, and the *$c$-colourability* (fixed c) in time $O\big(2^{ct(t+1)/2} \cdot ct^3 \cdot |V(G)|\big)$.

- An extesion: the canonical equiv. of *clique*(X) has index $\leq 2^{(t+1)(t+2)/4}$.

Faster new algorithms

Furthermore, the concept of a canonical equivalence gives us a fine control over the runtime dependency on the width parameter – we simply estimate its index.

Consider the universe of partially-equipped t-labeled graphs (of rank-width $\leq t$).

- As shown already by [Bui-Xuan, Telle, and Vatshelle, 08];

  the canonical equivalence of *independent-set*(X) has index $\leq 2^{t(t+1)/4}$ (this relates to the number of subspaces of $GF(2)^t$).

Theorem. [Bui-Xuan, Telle, and Vatshelle, 08]
The *independent set* problem can be solved in time $O\big(2^{t(t+1)/2} \cdot t^3 \cdot |V(G)|\big)$, and the c-*colourability* (fixed c) in time $O\big(2^{ct(t+1)/2} \cdot ct^3 \cdot |V(G)|\big)$.

- An extesion: the canonical equiv. of *clique*(X) has index $\leq 2^{(t+1)(t+2)/4}$.

Theorem. [Ganian and PH, 08]
Split graphs can be recognized in time $O\big(2^{(t+1)^2} \cdot t^3 \cdot |V(G)|\big)$, and so called c-*co-colourability* problem can be solved in time $O\big(2^{ct(t+1)} \cdot ct^3 \cdot |V(G)|\big)$.

5 The new extension: PCE Scheme

(PCE = prepartitioned canonical equivalence)

Starting point: The *dominating-set*(X) predicate has a double-exponential number of canonical equivalence classes. Yet solvable with single-exponential dependency on the rank-width [Bui-Xuan, Telle, and Vatshelle, 08].

5 The new extension: PCE Scheme

(PCE = prepartitioned canonical equivalence)

Starting point: The *dominating-set*(X) predicate has a double-exponential number of canonical equivalence classes. Yet solvable with single-exponential dependency on the rank-width [Bui-Xuan, Telle, and Vatshelle, 08].

How to cope with this in our formalism?

- Canonical equivalence records only the information we already know.

5 The new extension: PCE Scheme

(PCE = prepartitioned canonical equivalence)

Starting point: The *dominating-set*(X) predicate has a double-exponential number of canonical equivalence classes. Yet solvable with single-exponential dependency on the rank-width [Bui-Xuan, Telle, and Vatshelle, 08].

How to cope with this in our formalism?

- Canonical equivalence records only the information we already know.
- What can we do with the future information we get from further dynamic processing of our graph?
  Possible at all?

5 The new extension: PCE Scheme

(PCE = prepartitioned canonical equivalence)

Starting point: The *dominating-set*(X) predicate has a double-exponential number of canonical equivalence classes. Yet solvable with single-exponential dependency on the rank-width [Bui-Xuan, Telle, and Vatshelle, 08].

How to cope with this in our formalism?

- Canonical equivalence records only the information we already know.
- What can we do with the future information we get from further dynamic processing of our graph?
  Possible at all?
- Yes, we work with an "*expectation*" of future graph data (of H), and record known information wrt. **all** these possible "expectations".

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

$$G_1 \oplus H \models \mathcal{P} \iff G_2 \oplus H \models \mathcal{P}\,.$$

What is a PCE scheme

Consider the universe $\mathcal{U}_t$ of part.-equipped t-labeled graphs (of rank-width $\leq t$).

Definition. A property π has a *prepartitioned canonical equivalence scheme* (PCE scheme) if, for all t, there exist partitions $\mathcal{B}_t$ and $\mathcal{A}_t^B$, $B \in \mathcal{B}_t$, of $\mathcal{U}_t$:

- Classes of $\mathcal{B}_t$ present our "expectation" of future data (graph H).
- Wrt. particular expectation $B \in \mathcal{B}_t$, we record only a class of $\mathcal{A}_t^B$ the (so far processed) graph G_1 belongs to.

What is a PCE scheme

Consider the universe $\mathcal{U}_t$ of part.-equipped t-labeled graphs (of rank-width $\leq t$).

Definition. A property π has a *prepartitioned canonical equivalence scheme* (PCE scheme) if, for all t, there exist partitions $\mathcal{B}_t$ and $\mathcal{A}_t^B$, $B \in \mathcal{B}_t$, of $\mathcal{U}_t$:

- Classes of $\mathcal{B}_t$ present our "expectation" of future data (graph H).
- Wrt. particular expectation $B \in \mathcal{B}_t$, we record only a class of $\mathcal{A}_t^B$ the (so far processed) graph G_1 belongs to.

(i) $\mathcal{B}_t$ is "compatible" with the composition oper. occuring in the parse trees.

What is a PCE scheme

Consider the universe $\mathcal{U}_t$ of part.-equipped t-labeled graphs (of rank-width $\leq t$).

Definition. A property π has a *prepartitioned canonical equivalence scheme* (PCE scheme) if, for all t, there exist partitions $\mathcal{B}_t$ and $\mathcal{A}_t^B$, $B \in \mathcal{B}_t$, of $\mathcal{U}_t$:

- Classes of $\mathcal{B}_t$ present our "expectation" of future data (graph H).
- Wrt. particular expectation $B \in \mathcal{B}_t$, we record only a class of $\mathcal{A}_t^B$ the (so far processed) graph G_1 belongs to.

(i) $\mathcal{B}_t$ is "compatible" with the composition oper. occuring in the parse trees.

(ii) The $\mathcal{A}_t^B$-class of our graph is "uniq. determined" from a $\mathcal{B}_t$-expectation.

What is a PCE scheme

Consider the universe $\mathcal{U}_t$ of part.-equipped t-labeled graphs (of rank-width $\leq t$).

Definition. A property π has a *prepartitioned canonical equivalence scheme* (PCE scheme) if, for all t, there exist partitions $\mathcal{B}_t$ and $\mathcal{A}_t^B$, $B \in \mathcal{B}_t$, of $\mathcal{U}_t$:

– Classes of $\mathcal{B}_t$ present our "expectation" of future data (graph H).

– Wrt. particular expectation $B \in \mathcal{B}_t$, we record only a class of $\mathcal{A}_t^B$ the (so far processed) graph G_1 belongs to.

(i) $\mathcal{B}_t$ is "compatible" with the composition oper. occuring in the parse trees.

(ii) The $\mathcal{A}_t^B$-class of our graph is "uniq. determined" from a $\mathcal{B}_t$-expectation.

(iii) There is a constant d independent of t such that the following equivalence $\sim_\pi^{A,B}$ on A has index $\leq d$ (even $d = 1$) for all $B \in \mathcal{B}_t$ and $A \in \mathcal{A}_t^B$:

It is $\bar{G}_1 \sim_\pi^{A,B} \bar{G}_2$ if and only if $\bar{G}_1, \bar{G}_2 \in A$ and

$$\bar{G}_1 \otimes \bar{H} \models \pi \iff \bar{G}_2 \otimes \bar{H} \models \pi \quad \text{for all } \boldsymbol{\bar{H} \in B}.$$

Algorithms coming from PCE schemes

- Re-using the idea of an independent-set canonical classes, and employing "expectations", one gets:

Theorem. cf. [Bui-Xuan, Telle, and Vatshelle, 08]
The *dominating set* problem can be solved in time $O\big(2^{3t(t+1)/4} \cdot t^3 \cdot |V(G)|\big)$.

Algorithms coming from PCE schemes

- Re-using the idea of an independent-set canonical classes, and employing "expectations", one gets:

Theorem. cf. [Bui-Xuan, Telle, and Vatshelle, 08]
The *dominating set* problem can be solved in time $O\big(2^{3t(t+1)/4} \cdot t^3 \cdot |V(G)|\big)$.

- Furthermore, with some more involved linear algebra;

Theorem. [Ganian and PH, 08] The *acyclic-set*(X) and *connected-set*(X) predicates have PCE schemes of "size" $2^{O(t^2)}$.

Algorithms coming from PCE schemes

- Re-using the idea of an independent-set canonical classes, and employing "expectations", one gets:

Theorem. cf. [Bui-Xuan, Telle, and Vatshelle, 08]
The *dominating set* problem can be solved in time $O\big(2^{3t(t+1)/4} \cdot t^3 \cdot |V(G)|\big)$.

- Furthermore, with some more involved linear algebra;

Theorem. [Ganian and PH, 08] The *acyclic-set*(X) and *connected-set*(X) predicates have PCE schemes of "size" $2^{O(t^2)}$.

Corrolaries.

- The *acyclic colouring* problem solvable in $O\big(2^{5\,c^2t^2} \cdot c^2t^3 \cdot |V(G)|\big)$.

Algorithms coming from PCE schemes

- Re-using the idea of an independent-set canonical classes, and employing "expectations", one gets:

Theorem. cf. [Bui-Xuan, Telle, and Vatshelle, 08]
The *dominating set* problem can be solved in time $O\big(2^{3t(t+1)/4} \cdot t^3 \cdot |V(G)|\big)$.

- Furthermore, with some more involved linear algebra;

Theorem. [Ganian and PH, 08] The *acyclic-set*(X) and *connected-set*(X) predicates have PCE schemes of "size" $2^{O(t^2)}$.

Corrolaries.

- The *acyclic colouring* problem solvable in $O\big(2^{5\,c^2t^2} \cdot c^2t^3 \cdot |V(G)|\big)$.
- Other problems like connected dominating set, feedback vertex set, etc, have $O\big(2^{O(t^2)} \cdot |V(G)|\big)$ algorithms on graphs of rank-width t ...

6 Conclusions

- Parse trees give a useful tool for algorithms on graphs of bounded width,
  - giving an accessible "bridge" between design of specific algorithms and those very general results (like the MSO theorem).

6 Conclusions

- Parse trees give a useful tool for algorithms on graphs of bounded width,
  - giving an accessible "bridge" between design of specific algorithms and those very general results (like the MSO theorem).
- Focus on the precise number of canonical equivalence classes gives a fine control over the runtime of a dynamic algorithm wrt. our width parameter.
  - not being considered in depth before. . .

6 Conclusions

- Parse trees give a useful tool for algorithms on graphs of bounded width,
  - giving an accessible "bridge" between design of specific algorithms and those very general results (like the MSO theorem).
- Focus on the precise number of canonical equivalence classes gives a fine control over the runtime of a dynamic algorithm wrt. our width parameter.
  - not being considered in depth before. . .
- Even more, one can work with an "expectation" of future data and achieve additional speed-up, as with our new PCE scheme formalism.
  - can this be useful in other areas of algorithmic design?

6 Conclusions

- Parse trees give a useful tool for algorithms on graphs of bounded width,
  - giving an accessible "bridge" between design of specific algorithms and those very general results (like the MSO theorem).
- Focus on the precise number of canonical equivalence classes gives a fine control over the runtime of a dynamic algorithm wrt. our width parameter.
  - not being considered in depth before...
- Even more, one can work with an "expectation" of future data and achieve additional speed-up, as with our new PCE scheme formalism.
  - can this be useful in other areas of algorithmic design?
  - and is there a room for even more powerful speed-up techniques on parse trees? Where and how?

6 Conclusions

- Parse trees give a useful tool for algorithms on graphs of bounded width,
  - giving an accessible "bridge" between design of specific algorithms and those very general results (like the MSO theorem).
- Focus on the precise number of canonical equivalence classes gives a fine control over the runtime of a dynamic algorithm wrt. our width parameter.
  - not being considered in depth before...
- Even more, one can work with an "expectation" of future data and achieve additional speed-up, as with our new PCE scheme formalism.
  - can this be useful in other areas of algorithmic design?
  - and is there a room for even more powerful speed-up techniques on parse trees? Where and how?

THANK YOU FOR YOUR ATTENTION